Provable Secure Parallel Gadgets

Francesco Berti[1], Sebastian Faust[2] and Maximilian Orlt[2]

[1] Bar-Ilan University, Ramat-Gan 529002, Israel francesco.berti@biu.ac.il
[2] Department of Computer Science, TU Darmstadt, Darmstadt, Germany
{sebastian.faust,maximilian.orlt}@tu-darmstadt.de

Abstract. Side-channel attacks are a fundamental threat to the security of cryptographic implementations. One of the most prominent countermeasures against side-channel attacks is masking, where each intermediate value of the computation is secret shared, thereby concealing the computation's sensitive information. An important security model to study the security of masking schemes is the *random probing model*, in which the adversary obtains each intermediate value of the computation with some probability p. To construct secure masking schemes, an important building block is the *refreshing gadget*, which updates the randomness of the secret shared intermediate values. Recently, Dziembowski, Faust, and Zebrowski (ASIACRYPT'19) analyzed the security of a simple refreshing gadget by using a new technique called the *leakage diagram*. In this work, we follow the approach of Dziembowski et al. and significantly improve its methodology. Concretely, we refine the notion of a leakage diagram via so-called *dependency graphs*, and show how to use this technique for arbitrary complex circuits via composition results and approximation techniques. To illustrate the power of our new techniques, as a case study, we designed provably secure parallel gadgets for the random probing model, and adapted the ISW multiplication such that all gadgets can be parallelized. Finally, we evaluate concrete security levels, and show how our new methodology can further improve the concrete security level of masking schemes. This results in a compiler provable secure up to a noise level of $O(1)$ for affine circuits and $O(1/n^2)$ in general.

Keywords: Random Probing Model · Masking · Composability · Leakage Diagram

1 Introduction

Context. Proving the security of cryptographic schemes is the de-facto standard of modern cryptography. The most widely used security model is the black-box model, where the adversary has access to the inputs and outputs but has no knowledge or control over the algorithm's inner workings. It is well known, however, that real-world implementations may reveal information about the inner workings, and in particular about the secret key of a cryptographic scheme. Multiple side-channel attacks exploit physical phenomena such as power consumption [24], cache accesses [10], acoustic signals [18], or timing [23].

Masking schemes. A popular countermeasure against power analysis attacks is *masking*. At a high-level, the idea is to conceal sensitive intermediate values through secret sharing. A masking scheme relies on an *encoding function* that takes as input a value on a wire x and shares it over multiple wires that carry the shares $x_0, \ldots, x_{n-1}$. The encoding function we consider in this work samples $x_0, \ldots, x_{n-1}$ uniformly at random such that $x = \sum_i x_i$, where n is called the order of the masking scheme. If $x \in \mathbb{F}_2$, then such masking schemes are called Boolean masking. The main challenge in designing secure masking schemes is to develop operations – often called *gadgets* – that securely compute on shared values. Security here means that even if the adversary learns information on the internals of the gadget, such information does not reveal sensitive information. In addition, we need a method to compose gadgets without violating security. This is often done

via the *refreshing algorithm*, which takes as input a sharing $x_0, \ldots, x_{n-1}$ that encodes x, and outputs a fresh sharing $x'_0, \ldots, x'_{n-1}$ of the same secret value. Here, for security, we have to guarantee that even given leakage from the refreshing procedure, the output $x'_0, \ldots, x'_{n-1}$ is a fresh encoding of x.

Security analysis of masking countermeasure. As the design of the masking scheme is complex, we analyze their security using security proofs. To this end, we require a *leakage model* to describe the leakage emitting from a masked device formally. The most widely used leakage model is the *t-threshold probing model* originally introduced in the seminal work of Ishai, Sahai, and Wagner (ISW) [22]. In this model, cryptographic computation is described as a Boolean (sometimes arithmetic) circuit, where the adversary is allowed to probe up to t wires and learn the values carried on these wires during the computation.

Although a security analysis in the threshold probing model provides the first evidence of the soundness of a masking scheme, it does not accurately model the quantitative nature of leakage, thereby excluding important types of attacks [34, 12]. To address this problem, Prouff and Rivain introduced the *noisy leakage model* [30]. In this model, the adversary obtains a noisy version of each wire, where the noise is sampled from a certain distribution (e.g., the Gaussian distribution). Noisy leakages accurately model physical leakage from power consumption and, in particular, allow for quantitative statements about the noise required to conceal sensitive information – crucial information for cryptographic engineers. In detail, they define the noise as a set of probabilistic leakage functions that are restricted by an upper bound using the Euclidean Norm (or statistical distance) as a metric. An important shortcoming of the noisy leakage model, however, is that it is very hard to work with. Concretely, in comparison to the threshold probing model, security proofs are highly cumbersome, and proving the security of natural constructions often requires to rely on unrealistic assumptions (e.g., the use of leak-free gates). To resolve these problems, somewhat surprisingly, Duc et al. [14] showed that noisy leakages and the seemingly much weaker threshold probing model of ISW are related. For their proof, they considered an intermediate model – the *p-random probing model* – and showed that security in this model directly implies security against noisy leakages. The p-random probing model considers a particular noise distribution, where each wire leaks with probability p, while the adversary obtains no knowledge of the wire's value with probability $1 - p$. The security in the random probing model only implies security in the noisy model with a loss of the field size. There are two approaches to avoiding the security loss. The first approach was presented by Dziembowski et al.[15] who proposed the *average random probing model*, a modified version of the random probing model. However, security proofs in this model are still rather complex for two reasons. First, they assume a more powerful class of leakage functions because the adversary can choose leakage functions where only the average leakage probability is p. In other words, for any possible input value, the leakage probability p can be different (up to p times the field size). Second, the adversary also learns the internal randomness used by the leakage function to decide whether a value leaks or not. Hence, the adversary even learns something about the values when the leakage function does not output the value. An alternative approach is given by Goudarzi et al. [29]. They eliminate the field size by using an alternative metric for the noisy model. In other words, they do not modify the probing model where the actual proof is done, but the noisy model that should model the natural leakage. In detail, they use a worst-case metric called (Average) Related Error and show that security in the random probing model tightly implies security in the modified noisy model. Since the result of Goudarzi et al. and Duc et al., security in the random probing model has been studied intensively by the research community [2, 4, 7, 11, 16]. There are two important goals in this research area. First, we aim to design masking schemes that obtain security for values of p independent of the order n of the masking scheme and are, in particular, close to 1. This is important as it implies that the masked computation remains secure in the presence of larger amounts of leakage. Second, the masking schemes that we design need to be efficient, where efficiency is typically measured in terms of circuit and randomness complexity. In particular, since all of our gadgets have low depth, the latency of the compiled circuits is significantly improved. Our main contribution is to improve on both of these goals for

certain classes of masked computation.

1.1 Contribution

Improved analysis of refreshing gadgets. As discussed above, the main ingredient of any masked computation is a secure refreshing gadget. It is typically placed throughout the masked computation to ensure composition. In addition, refreshing gadgets also have applications for key refreshing, e.g., as part of a masked AES, where the secret key has to be refreshed periodically to ensure security. There is a large body of literature on designing secure refreshing schemes. For our work, the most important is the work of Dziembowski et al. [16], who gave a security analysis of a very simple and efficient refreshing. Their scheme essentially uses only n randomness and n operations, which is optimal for a refreshing gadget of order n masking. Dziembowski et al. show that this simple refreshing gadget surprisingly is $O(\sqrt{p}^n)$-secure. Our first contribution is to improve their construction and show that it achieves asymptotically better security of $O(p^n)$.

Improved analysis of affine masked computation. As a second contribution, we extend our analysis of the refreshing gadget to protect affine computation. Affine computation (i.e., addition and multiplication by a constant) is frequently used in cryptographic schemes since it is less costly than non-linear operations. This is the reason why, for instance, many symmetric cryptographic schemes make massive use of affine computation. In our work, we give an improved analysis for simple masked affine computation. In particular, we consider a very simple addition gadget, which computes the addition of two sharings $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$ by adding the shares component-wise, i.e., $c_i = a_i + b_i$ followed by a refreshing of $c_0, \ldots, c_{n-1}$. We can show that this gadget remains $O(p^n)$-secure, where earlier works either require significantly more randomness (namely, [4] with $O(n^{2.4})$-randomness required, while ours needs $O(n)$-randomness to refresh the inputs) or require more noise (namely [16], $O(\sqrt{p}^n)$).

We also show how to extend our results to the masked computation of non-linear operations. Our multiplication gadget is essentially the widely used and analyzed ISW multiplication. While it is known that asymptotically, there are more advanced constructions that achieve security for a constant p, we improve the analysis of the ISW multiplication for small share numbers. Concretely, we can prove the security up to $[18p + 2(1 - (1 - p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1})]^n$, instead of $(32np + 4n\sqrt{3p})^n)$ as in [16]. Interestingly, this illustrates that the security for larger share number n is better than previously assumed [16]. We believe that this is a worthwhile goal due to the following two reasons. First, it was shown [11] that the ISW-multiplication achieves better security than more advanced constructions for small values of n. Second, the ISW multiplication is widely used in many masking schemes, and hence it is important to better understand its security in the random probing model.

Parallel computation. Finally, we note that all our constructions are highly parallelizable. Parallel gadgets [13, 3] are particularly interesting for masked circuits as they are faster due to executing many operations at the same time. In addition, it is also more challenging to perform a side-channel attack against a parallel implementation than against a serial one. The basic idea is that parallel computations can increase the noise in the attacks, as shown in [27].

1.2 Related Work

Proof techniques. Analyzing leakage resilience of circuits via graphs was already proposed in [31] at Crypto 2015. They described a transformation of circuits based on graphs to generalize the ISW Multiplication and showed that it is closely related to Threshold Implementations [28] and the Trichina gate [33]. In particular, they give a generalized graph for the multiplication gadget using different layers, such as linear and non-linear layers, and compare the security of the different multiplication gadgets. In contrast to the work in [31], the work of [16] did not

use the graph to analyze a gadget but a full circuit in the random probing model. They also use a graph based approach that is in particular useful to analyze the linear layers of circuits. We formalized the approach of [16], and give tighter security proves in the random probing model. Further, we propose gadgets with lower latency. Alternative approaches to analyze the security in the random probing model were proposed in [2, 4, 7, 9]. They introduce definitions for random probing composability based on counting the number of probes at the inputs and outputs of gadgets that are needed to simulate the leakage. To improve this approach, we could follow the recent work of Cassiers et al. [11] and tighten the analysis. Concretely, in [11], the authors use a definition, which they call the *Probe Distribution Table (PDT)*. The *PDT* allows a tighter analysis since it considers the concrete wires that the simulator needs. The drawback of the *PDT* approach is that the table grows exponentially with the number of shares of the gadgets, and thus a generic analysis is not possible. The work of [4, 7, 5] allows analysis for generic order, but it only provides security proofs for circuits with special structures. For this reason, the constructions are typically less efficient, as discussed above.

Compiler. As mentioned in Section 1.1, many compilers produce masked circuits with provable security in the random probing model. At Eurocrypt 2016, Andrychowicz et al. [2] presented a compiler with constant leakage probability using expander graphs. This rather is a feasibility result since expander graphs require a high number of shares. Two years later, Goudarzi et al. [19] gave a compiler for polynomial sharing requiring noise $p = O(1/\log(n))$. Here, they presented an NTT-based secure multiplication with complexity $O(n\log(n))$. The compiler was further improved in [20] to allow more general fields $\mathbb{F}$ and complexity $\Theta(n\log(n))$. They use the additive FFT algorithm proposed by Gao and Mateerin 2010 [17] to avoid the limitations of the classical NTT. With self-folding bases, a generalization of Cantor bases, they further optimized the gadget. However, the field size still restricts the number of shares $n < |\mathbb{F}|$ due to the share-wise different support points of poly sharings. Considering affine circuits, our compiler is more efficient. For example, our refresh gadget has linear complexity and does not use multiplication gates, while the one in [20] uses $n\log(n)/2$ multiplications. Further, our compiler allows a leakage rate of $O(1)$ for affine circuits instead of $O(1/\log(n))$. Regarding non-affine circuits, their construction has better complexities with respect to efficiency and security due to their NTT-based multiplication. However, for our compiler, we slightly modified the ISW multiplication such that it is parallelizable.

To allow security for a constant leakage probability p, Ananth et al. [1] proposed a *modular approach* how to compose a secure compiler multiple times. Finally, several follow-up works further improved this approach [4, 8, 6]. However, as described in Section 1.1, this approach leads to relatively costly circuits with randomness complexity of at least $O(n^{2.4})$ for affine and non-affine circuits, while our compiler only requires $O(n)$ and $O(n^2)$, respectively. In particular, our work analyzes the widely used ISW multiplication that is still promising for reasonable share number ($2 \geq n \geq 32$) and noise parameters [11]. For this reason, we try to close the gap between practice and theory and give a tighter security analysis in the random probing model.

2 Background

Notations. Let $[n] := \{0, 1, \ldots n-1\}$. Let $(\mathbb{F}, +, \cdot)$ be a finite field with its addition and multiplication (and let $-$ be its subtraction). We denote with $\mathbf{x}$ and $(x_i)_{i\in[n]}$ vectors with coefficients in the field $x_i \in \mathbb{F}$. Let X_0, X_1 be two random variables over a set $\mathcal{X}$. Their *statistical distance* is: $\Delta(X_0; X_1) := \frac{1}{2}\sum_{x\in\mathcal{X}} |\Pr[X_0 = x] - \Pr[X_1 = x]|$. If $\Delta(X_0; X_1) \leq \epsilon$, we say that X_0 *and* X_1 *are* ϵ*-close*.

Directed graphs. A *directed graph* is a pair $G = (V, E)$ with a set of vertices/nodes V and a set of edges $E \subseteq \{(x, y) | (x, y) \in V^2 \text{ and } x \neq y\}$, which are ordered tuples of vertices. Further, we write (x, y) to refer to such edges. We call x its *source node* and y its *destination node*. When we draw our graphs, we represent the edge (x, y) with an arrow from x pointing to y. We write $-(x, y) := (y, x)$

to exchange destination and source. A *sub-graph* $G' \subset G$ is a graph $G' = (V, E')$ with $E' \subset E$. This allows us to define unions of sub-graphs $G' = (V, E')$, $G'' = (V, E'')$ with $G' \cup G'' := (V, E' \cup E'')$. Note that all sub-graphs also have all nodes V. In our work, we are only interested in the edges and assume that each sub-graph still consists of all nodes. Further, if we consider graphs $G' = (V', E')$, $G'' = (V'', E'')$ with different nodes $V' \neq V''$, we also write $G = G' \cup G'' := (V' \cup V'', E' \cup E'')$. Hence, G is a graph consisting of two (sometimes unconnected) sub-graphs G' and G''. Let G be a graph. A *path* is the image of continuous functions $f : [0, 1] \subset \mathbb{R} \to G$. A *loop* is a path s.t. $f(0) = f(1)$. We consider only loops s.t. $f_{|(0,1)}$ is injective to avoid loops of type (x, y), $-(x, y)$ or containing it as a sub-loop.

Circuits. An *arithmetic circuit* over a finite field $\mathbb{F}$ is a labeled acyclic graph. Its edges are the *wires*, and its vertices are the *gates*. The edges pointing to a gate are the *input wires* of the gate, while those coming from it are the *output wires*. We use the following gates: *addition*  : with fan-in 2 and fan-out 1, outputting the addition of the 2 input variables; *subtraction*  : as the addition one, outputting the subtraction; *multiplication*  as the addition, outputting the multiplication; *constant*  : with fan-in 0 and fan-out 1, outputting the constant value a; *random*  : with fan-in 0 and fan-out 1 outputting a uniform random variable; *copy* : with fan-in 1 and fan-out 2, outputting 2 copies of the input variable ; *input* I: with fan-in 1 and fan-out 1, outputting the input variable; *output* O: with fan-in 1 and fan-out 1, outputting the output variable. The last two gates (I and O) are added for syntactic reasons. A *complete* circuit is a circuit where there is an I gate at every input wire of the circuit and an O gate at every output wire; otherwise, the circuit is *incomplete*. The *completion* of an incomplete circuit is the addition of I and O whenever needed to make the circuit complete. An *affine* circuit is a circuit without multiplication gates. We denote with $\mathcal{W}(\mathsf{C})$ the set of wires of the circuit C.
A wire carries a *variable*. We say that two variables, x and y, are the *same variable* if the wires carrying x and y are connected only via copy gates. The *value* of a variable x is the value that is carried on the wire carrying x during an execution with fixed inputs and randomness.

Masking. One of the most common countermeasures against side-channel attacks is *masking*. The idea is to split the sensitive variables into n shares and then perform the computations on these shares and finally recover the output. We use an *encoding* scheme $(\mathtt{Enc}, \mathtt{Dec})$ to encode variables, *gadgets* to perform computations on encoded variables, and a *refreshing gadget* to securely compose multiple gadgets. We discuss these individual components below in more detail.

Encoding/decoding schemes. An *encoding* scheme $\mathtt{Enc}$ is a probabilistic algorithm that takes as input $x \in \mathbb{F}$ and outputs an n-tuple $(x_0, ..., x_{n-1}) = \mathtt{Enc}(x)$, where n is the *masking order*. The *decoding* scheme $\mathtt{Dec}$ takes as input an n-tuple $(x_0, ..., x_{n-1})$ and outputs $x = \mathtt{Dec}(x_0, ..., x_{n-1})$. For correctness, we want that for any $x \in \mathbb{F}$ it holds that $\mathtt{Dec}(\mathtt{Enc}(x)) = x$. For security, we need that any subset of $n - 1$ shares of $\mathtt{Enc}(x)$ are independent of x. We use *arithmetic encoding*. $\mathtt{Enc}(x)$ provides a randomized n-tuple $x_0, ..., x_{n-1}$ s.t. $\sum_{i=0}^{n-1} x_i = x$, and $\mathtt{Dec}(x_0, ..., x_{n-1}) = \sum_{i=0}^{n-1} x_i$. In the following, we will often denote an encoding of x by $(x_i)_{i\in[n]}$.

Gadgets. To perform computations on encoded variables, we construct *gadgets*. Gadgets are made out of simple gates such that even if the internals of the gadgets leak, the adversary will not learn any "useful" information. Suppose we have a gate implementing the function $f : \mathbb{F}^l \to \mathbb{F}^k$ (e.g.,$l = 2$ and $k = 1$, for). The corresponding gadget G_f is composed of many gates and performs the same operation where the input wires hold l encodings and the output wires carry k encodings of the outputs. We require *soundness* from the gadgets, i.e., gadgets that perform the same operation as the underlying gate, just in the encoded domain. Formally, we have $\forall \mathbf{x} = (x^0, ..., x^{l-1}) \in \mathbb{F}^l$,

$$f(x^0, ..., x^{l-1}) = (\mathtt{Dec}(y_i^0)_{i\in[n]}, \ldots, \mathtt{Dec}(y_i^{k-1})_{i\in[n]})$$

with $((y_i^0)_{i\in[n]}, \ldots, (y_i^{k-1})_{i\in[n]}) \leftarrow \mathsf{G}_f(\mathtt{Enc}(x^0), \ldots, \mathtt{Enc}(x^{l-1}))$.

Refreshing schemes. Refreshing schemes (or refreshing gadgets) are gadgets G_f where f is the identity.[1] The scheme takes as input an encoding $(x_i)_{i\in[n]}$, and outputs a re-randomized encoding $(y_i)_{i\in[n]}$, such that $\mathtt{Dec}((x_i)_{i\in[n]}) = \mathtt{Dec}((y_i)_{i\in[n]})$. In this work, we consider refreshing schemes using a linear number of random gates $\$$.

In Figure 1a the simple refresh `sRef` of [16] is depicted, initially introduced in [32]. The gadget adds a random value to each input $y_i \leftarrow x_i + r_i$ with $i = 0, \ldots n-2$ and subtracts each random value from the last input $y_{n-1} \leftarrow x_{n-1} - (r_0 + \cdots + r_{n-2})$.

This work considers an alternative to `sRef` that we call `pRef`; see Figure 1b. This gadget was initially introduced in [3] and has the key feature that it is highly parallelizable. `pRef` takes as an input $(x_i)_{i\in[n]}$ with n random values r_i and processes them in two parallel steps. In the first step, it computes $b_i \leftarrow x_i + r_i$, and in the second step, it subtracts r_{i-1} from b_i such that $y_i \leftarrow b_i - r_{i-1 \pmod n}$ for all $i \in [n]$, obtaining $(y_i)_{i\in[n]}$.

Circuit compilers. Given the components from above, we can transform circuit C into a masked circuit $\widehat{\mathsf{C}}$. This is done via the concept of a *circuit compiler* $\mathtt{CC}$. $\mathtt{CC}$ works as follows: First, $\mathtt{CC}$ replaces each wire carrying x with a *bundle* of n-wires carrying an encoding of x, $(x_i)_{i\in[n]}$. Next, it replaces all gates in C with the corresponding sound gadgets, input I gates with $\widehat{\mathsf{I}}$ *input encoders* (which encodes the input), and output O gates with $\widehat{\mathsf{O}}$ *output decoders*. Finally, between every two gadgets, the compiler $\mathtt{CC}$ adds a *refreshing gadget* to ensure secure composition. The masked transformation $\widehat{\mathsf{C}}$ of a complete circuit C is *sound* if $\widehat{\mathsf{C}}(\mathbf{x}) = \mathsf{C}(\mathbf{x})$ for every possible input $\mathbf{x}$ of C. For an incomplete circuit C, we say that the transformation $\widehat{\mathsf{C}}$ is *sound* if the transformation of its completion is sound. A compiler $\mathtt{CC}$ is *sound* if for all circuits C the transformation $\widehat{\mathsf{C}} = \mathtt{CC}(\mathsf{C})$ is sound.

Random probing model. As discussed in the introduction, we use the p-random probing model, originally introduced in [22] to model side-channel leakage of the transformed circuit $\widehat{\mathsf{C}}$. In the p-random probing model, each wire leaks the value that it carries with probability p. Following [22], we assume that the wires of the input encoders $\widehat{\mathsf{I}}$ and output decoders $\widehat{\mathsf{O}}$ *do not leak*. Notice that, as in [22], this is without loss of generality when we move from stateless to stateful circuits. To make it explicit what wires leak, we will denote in the following with $\mathcal{W}'(\widehat{\mathsf{C}}) \subset \mathcal{W}(\widehat{\mathsf{C}})$ the set of wires of the circuit $\widehat{\mathsf{C}}$ that can leak. The definition below formalizes security in the p-random probing model.

The transformed circuit is private if its leakage reveals nothing about its inputs and outputs. We can define this with a security experiment.

Definition 1 (Privacy [16])**.** Let C be a circuit with fan-in k with input $\mathbf{x} = (x_1, ..., x_k)$. Further, let $\widehat{\mathsf{C}}$ be a sound transformation of C and $p \in [0, 1]$ its leakage probability. The *leakage experiment* $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ is defined as follows:

- We fed $\mathbf{x}$ to $\widehat{\mathsf{C}}$ resulting in some assignments of the wires of $\widehat{\mathsf{C}}$. If C is incomplete, the input bundle corresponding to the input wire containing x is fed with an encoding $(x_i)_{i\in[n]}$ of x.
- Each wire w of $\mathcal{W}'(\widehat{\mathsf{C}})$ is added to $\mathcal{L}_p(\widehat{\mathsf{C}})$ with probability p.
- Output: $(\mathcal{L}_p(\widehat{\mathsf{C}}), A_{|\mathcal{L}_p(\widehat{\mathsf{C}})})$, where A is the set of the values carried by the wires of $\mathcal{W}$ during the circuit evaluation of $\widehat{\mathsf{C}}$ on input $\mathbf{x}$.

$\widehat{\mathsf{C}}$ is (p, ϵ)-*private* if there is a simulation algorithm that, not knowing $\mathbf{x}$, outputs a random variable that is ϵ-close to the actual output of $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$

[1] We emphasize that this does not imply that G_f is also the identity. Since the gadget can be probabilistic, the encoding of the outputs can be re-randomized.

$\texttt{sRef}((x_i)_{i\in[n]})$

for $j \in \{0 \dots n-2\}$
  $r_j \leftarrow_\$ \mathbb{F}$
$c_0 \leftarrow 0$
for $j \in \{0 \dots n-2\}$
  $c_{j+1} \leftarrow c_j + r_j$
  $y_j \leftarrow x_j + r_j$
$y_{n-1} \leftarrow x_{n-1} - c_{n-1}$
return $(y_i)_{i\in[n]}$

(a) Simple refresh gadget [32].

$\texttt{pRef}((x_i)_{i\in[n]})$

for $j \in \{0 \dots n-1\}$
  $r_j \leftarrow_\$ \mathbb{F}$
for $j \in \{0 \dots n-1\}$
  $b_j \leftarrow x_j + r_j$
  $y_j \leftarrow b_j - r_{j-1 \pmod n}$
return $(y_i)_{i\in[n]}$

(b) Parallel refresh gadget [3].

Figure 1: Refresh gadgets with linear random complexity.

In other words, the masked circuit $\widehat{\mathsf{C}}$ is (p, ϵ)-*private* if the leakage in experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ can be simulated independently from the inputs up to ϵ statistical distance. More precisely, if for any two inputs $\mathbf{x}, \mathbf{x'}$ of the circuit, the distributions $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ and $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$ are ϵ-close, then $\widehat{\mathsf{C}}$ is (p, ϵ)-private. This observation was used to prove security in [16]. Therefore, they defined *Extended Leakage Shiftability* to describe when the leakage is independent of the input. In particular, shiftability accurately describes the fact that we can change the input of a circuit so that the observed leakage does not contradict the new input.

Definition 2 (Leakage Shiftability [16])**.** Let $\widehat{\mathsf{C}}$ be a sound transformation of a circuit C. We say that an output L of the experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ is *shiftable* to $\mathbf{x'}$ if it can be output of the experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$.

In other words, let $L \leftarrow \mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ be the leakage with $L = A_{|\mathcal{L}_p(\widehat{\mathsf{C}})}$, where A is the set of the values carried by the wires during the circuit evaluation of $\widehat{\mathsf{C}}$ on input $\mathbf{x}$. Then, L is shiftable if there is an assignment A' with the same probability during the circuit evaluation of $\widehat{\mathsf{C}}$ on input $\mathbf{x'}$ s.t. it still holds for both leakages $L = A'_{|\mathcal{L}_p(\widehat{\mathsf{C}})}$. In this case, we can shift the values of the variables from A to A' without modifying the values of the variables leaked. This technique allows more fine-grained security analyzes than simulatability since we show where we can modify the input encodings of each gadget (without ignoring where exactly, as done for simulatability). So, if the leakage is shiftable and the leakage of the shifted encoding has the same distribution as the unshifted one, the leakage is independent of the encoding. Hence, the leakage can be simulated without knowing the encoded value. This property was also used in [16] to prove their compiler security.

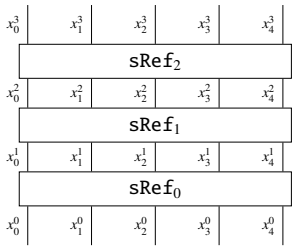
Corollary 1 ([16])**.** *Let $\widehat{\mathsf{C}}$ be the sound transformation of a circuit C via Dziembowski et al.'s compiler [16]. If*

$$\Pr[\mathsf{Leak}(\widehat{\mathsf{C}}, \boldsymbol{x}, p) \textit{ is not shiftable to } \boldsymbol{x'}, \textit{ for any } \boldsymbol{x'}] \leq \epsilon,$$

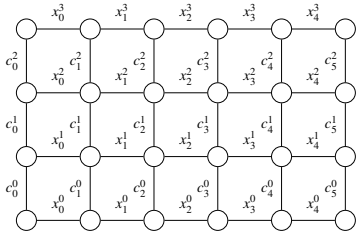
for any input $\boldsymbol{x}, \boldsymbol{x'}$ then $\widehat{\mathsf{C}}$ is (p, ϵ)-secure.

To compute the shift probability, the authors give a new technique to transform this problem into a graph path problem. Next, we present the class of graphs they consider.

Original leakage diagram. Dziembowski et al. [16] introduced the concept of *leakage diagrams*. They represent all the variables on a graph and those variables whose values are leaked in a subgraph called the *leakage diagram*. For example, they represent multiple consecutive executions of `sRef` (depicted in Figure 2a) with the diagram depicted in Figure 2b. The edges of the diagram



(a) Simple refresh gadget [16].



(b) The graph of the simple refresh used for the leakage diagram [16].

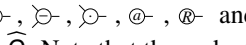
Figure 2: Refresh gadgets with linear random complexity.

represent the intermediate values that are computed during the execution of this circuit. The intermediate values of each `sRef` execution are represented by two consecutive rows and by the vertical edges between these two rows. On the lower row, there are n edges that represent the input shares (one edge per share), while on the upper row, there are n edges representing the output shares of a refreshing gadget. The vertical edges between two rows represent the partial sum c^i_j of the random values used during that execution. Since $c^i_{j+1} = c^i_j + r^i_j$, the variable r^i_j is represented by both the edges c^i_j and c^i_{j+1}. They also add the edges corresponding to the variable c^i_n. These c^i_n are defined similarly to the others as follows $c^i_n = c^i_{n-1} + x^i_{n-1} - x^{i-1}_{n-1}$. Thus, it always holds $c^i_n = 0$ because $c^i_n = c^i_{n-1} + x^i_{n-1} - x^{i-1}_{n-1} = c^i_{n-1} + (x^{i-1}_{n-1} - c^i_{n-1}) - x^{i-1}_{n-1} = 0$.

During k executions of `sRef` the adversary receives a leakage $\mathcal{L}_p$ (see Def. 1). The *leakage diagram* corresponding to $\mathcal{L}_p$ is the subgraph of Fig. 2b composed by all the edges corresponding to the variables belonging to $\mathcal{L}_p$ (and all c^i_0 and c^i_n since they are always equal to 0. Thus their values are always known by the adversary). Further, Dziembowski et al. [16] proved that the leakage can be simulated independently from the input $x = \mathtt{Dec}((x^0_i)_{i\in[n]})$ if there is no path from the left to the right of the leakage diagram. The technique was extended to analyze more complex masked circuits where each gadget's output is refreshed with the `sRef` gadget. Further, they show that the security can be bounded with the probability that there is such a path.

3 Parallel Compiler

The circuit compiler we present in this paper has the key feature that its operations are highly parallelizable. It uses the standard gadgets for addition (cf. Fig. 3a), copy (cf. Fig. 3b), random (cf. Fig. 3d), and constant (cf. Fig. 3c) operations. When considering their circuit representation, the gadgets have a low depth; hence, they can be executed highly parallelly. The more interesting gadgets are the ones for the multiplication of two encoded inputs (cf. Fig. 3e) and for refreshing an encoding (cf. Fig. 1b). The multiplication gadgets with input encodings $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$ computes the tensor product $a_i b_j$ of all shares. As in the ISW multiplication, this results in n^2 products $a_i b_j$ that we need to compress to a random encoding of the output by appropriately adding up these values and blinding the intermediate results by injecting fresh randomness. In contrast to the ISW multiplication that has depth n, we change the way in which this final addition is done to reduce the depth to $log(n)$. This significantly reduces the latency of the gadget from n to $\log(n)$. The refresh gadget re-randomizes an encoding such that it still decodes to the same value. The refresh gadget of our compiler has the key feature that it only has depth 2 in contrast to the simple refreshing from [16], which has an asymptotic depth of n.

Compiler CC^p. The parallel compiler CC^p takes as input an arbitrary circuit C using the gates  , and replaces each gate with the corresponding gadget from Figure 3 in $\widehat{\mathsf{C}}$. Note that the gadget for  works by slightly modifying the `Add` gadget such that the second input is first share-wise transformed to its additive inverse, i.e., by setting $-b_i \leftarrow b_i$. At the high level, the topology of C and $\widehat{\mathsf{C}}$ is the same, i.e., if two gates are connected by wires in C, then the corresponding gadgets are connected in the same way through wire bundles in $\widehat{\mathsf{C}}$. These wire bundles carry the encodings of the variables corresponding to the wires in C. Finally, to guarantee composability, the compiler CC^p inserts refresh gadgets between each computational gadget to inject further randomness. By applying the compiler to a circuit C, we get a parallelizable masked circuit that we denote with $\widehat{\mathsf{C}} \leftarrow \mathsf{CC}^p(\mathsf{C})$. We start by showing the soundness of the compiler CC^p in the following corollary.

Corollary 2. *Let $\mathsf{C} : \mathbb{F}^s \rightarrow \mathbb{F}^t$ be an arbitrary circuit and $\widehat{\mathsf{C}} \leftarrow \mathsf{CC}^p(\mathsf{C})$. For any $x^0, \ldots, x^{s-1} \in \mathbb{F}$, we have:*

$$((y_i^0)_{i\in[n]}, \ldots (y_i^{t-1})_{i\in[n]}) \leftarrow \widehat{\mathsf{C}}(\mathtt{Enc}(x^0), \ldots, \mathtt{Enc}(x^{s-1}))$$

with $\mathsf{C}(x^0, \ldots, x^{s-1}) = (\mathtt{Dec}((y_i^0)_{i\in[n]}), \ldots, \mathtt{Dec}((y_i^{t-1})_{i\in[n]}))$.

Proof. The proof of the corollary immediately follows from the soundness of the gadgets depicted in Figure 3. □

Our compiler has similar features to the compiler of [16]. More precisely, we can show that Corollary 1 also holds for our compiler.

Corollary 3. *Let $\widehat{\mathsf{C}}$ be the sound transformation of a circuit C via our compiler, CC^p. If*

$$\Pr[\mathsf{Leak}(\widehat{\mathsf{C}}, \boldsymbol{x}, p) \textit{ is not shiftable to } \boldsymbol{x'}, \textit{ for any } \boldsymbol{x'}] \leq \epsilon,$$

for any input $\boldsymbol{x}, \boldsymbol{x'}$ then $\widehat{\mathsf{C}}$ is (p, ϵ)-secure.

Proof. This proof is similar to the one in [16]. According to Definition 1, we need to simulate $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ independently from the input $\mathbf{x}$ up to ϵ statistical distance. Therefore, we prove that the distribution of $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$-experiment is ϵ-close to $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$-experiment for any $\mathbf{x}, \mathbf{x'}$ if it is shiftable. This immediately gives the required simulator[2]. So, we are left with the proof that shiftability for any $\mathbf{x}, \mathbf{x'}$ implies that the outputs of the experiments $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ and $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$ are ϵ-close. Let n be the number of shares used by $\widehat{\mathsf{C}}$. We will follow a sequence of Games where the first game represents the circuit with input $\mathbf{x}$ and the last one with input $\mathbf{x'}$:

Game 0: The leakage experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$.

Game 1: The modified Game 0, where we have modified how `pRef` gadgets pick the randomness. Instead of picking the randomness uniformly at random, `pRef` picks uniformly at random a new encoding of x, $(x_0^*, ..., x_n^*)$, then, via the randomness reconstructor $\mathit{RandR}(\mathtt{pRef})$ (described in Fig. 4), it computes the internal randomness. Hence, the output shares $(x_0^i, ..., x_n^i)$ are the random encoding $(x_0^*, ..., x_n^*)$.
Transition between Game 0 and Game 1: We show that the randomness reconstructor $\mathit{RandR}(\mathtt{pRef})$ outputs randomness indistinguishable from that used by `pRef` of Game 0. In both cases, r_0 is picked uniformly at random, and hence, r_0 is picked in the same way. Since it holds $r_j = x_j - y_j - r_{j-1}$ and all y_j are picked uniformly at random, all r_j have the same distribution as if they are picked uniformly at random. Since Game 0 and Game 1 only differ in how the randomness is used, and the randomness in Game 0 and Game 1 is indistinguishable. Hence, both games are indistinguishable.

[2]The simulator takes a random input $\mathbf{x'}$ and outputs $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$.

$\texttt{Add}((a_i)_{i\in[n]}, (b_i)_{i\in[n]})$

for $i \in \{0 \dots n-1\}$

$c_i = a_i + b_i$

return $(c_i)_{i\in[n]}$

(a) The addition gadget `Add` for $\overset{\oplus}{\rightarrow}$ and $\overset{\ominus}{\rightarrow}$ with $b_i \leftarrow -b_i$.

$\texttt{Copy}((a_i)_{i\in[n]})$

for $i \in \{0 \dots n-1\}$

$b_i = a_i$

$c_i = a_i$

return $(b_i)_{i\in[n]}, (c_i)_{i\in[n]}$

(b) The copy gadget `Copy` for $\overset{\copyright}{\rightarrow}$.

$\texttt{Const}_a$

return $(a_i)_{i\in[n]}$

(c) The constant gadget $\texttt{Const}_a$ for $\overset{@}{\rightarrow}$ with $(a_i)_{i\in[n]} \leftarrow \texttt{Enc}(a)$.

`Rand()`

for $i \in \{0 \dots n-1\}$

$r_j \leftarrow_{\$} \mathbb{F}$

return $(r_i)_{i\in[n]}$

(d) The random gadget `Rand` for $\overset{\circledR}{\rightarrow}$.

$\texttt{Mult}((a_i)_{i\in[n]}, (b_i)_{i\in[n]})$

for $i \in \{1, ..., n-1\}$

  for $j \in \{0, ..., i-1\}$

    $z_{0,i,j} \leftarrow_{\$} \mathbb{F}$

for $i \in \{0 \dots n-1\}$

  for $j \in \{i+1, ..., n-1\}$

    $w_{i,j} = a_i \cdot b_j - z_{0,j,i}$

    $z_{0,i,j} = w_{i,j} + a_j \cdot b_i$

for $i \in \{0, ..., n-1\}$

  $z_{0,i,i} = a_i \cdot b_i$

$L = log_2(n+1)$

for $i \in \{0, n-1\}$

  for $l \in \{1, ..., L\}$

    for $j \in \{0, ..., 2^{L-l} - 1\}$

      $z_{l,i,j} = z_{l-1,i,2j} + z_{l-1,i,2j+1}$

for $i \in \{0, ..., n-1\}$

  $c_i = z_{L,i,0}$

return $(c_i)_{i\in[n]}$

(e) The multiplication gadget `Mult` for $\overset{\otimes}{\rightarrow}$.

Figure 3: Parallel gadgets.

Input: $(x_0, ..., x_n), (y_0, ..., x_n^{i+1})$
$r_{n-1} \leftarrow_{\$} \mathbb{F}$
for $j = 1, ..., n-1$
$r_j = x_j - y_j - r_{j-1}$
endfor
Return $(r_0, ..., r_n)$

Figure 4: The Randomness Reconstructor *Rand*(pRef) for $(y_i)_{i\in[n]} \leftarrow \mathtt{pRef}((x_i)_{i\in[n]})$.

Game 2: It is the modified Game 1, where we have replaced the non-leaked intermediate values of the variables in such a way that $\widehat{\mathsf{C}}$ has the input **x'**. Note that this is possible due to the shiftablity assumption, and hence, we can apply shiftabilty without giving further details.
Transition from Game 1 to Game 2: Since an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$-experiment can be shifted to an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$-experiment except with probability ϵ, we can do this shift with probability $1 - \epsilon$. Due to the ϵ, this is the only game hop in the proof with a loss.

Game 3: Game 3 is the modified Game 2, where we have replaced the input and output encodings of all gadgets with random encodings of the same value as we did in Game 1. In other words, every input/output encoding is again replaced with a random encoding that still decodes to the same value (the one that we got in Game 2).
Transition between Game 2 and Game 3: Since the inputs and outputs of both games are still random encodings, we cannot distinguish Games 2 and 3. Note that the encodings in Game 2 are still random because the shiftability depends only on the leaked variables and not on the values that this variable assumes.

Game 4: The leakage experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$.
Transition between Game 3 and Game 4: It is the inverse of the transition between Game 0 and Game 1. Thus, using *RandR*(pRef), we can prove that these two games are indistinguishable.

As mentioned in the transition from Game 1 to Game 2, this is the only step with a security loss of ϵ. This proves the claim since it immediately flows that Game 0 and Game 4 are ϵ-close. □

3.1 Dependency Graph for our Gadgets

The values carried by the wires of a masked circuit can be considered as a set of random variables randomized by the random input encodings and the internal random gates. When we analyze such random variables X and Y representing intermediate wires, they can carry values $x, y \in \mathbb{F}$ with $\Pr[X = x] \geq 0$ and $\Pr[Y = y] \geq 0$. When we analyze the leakage resilience of circuits, we can distinguish two cases (i) intermediate values are independent $\Pr[X = x, Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$ or (ii) they are dependent $\Pr[X = x, Y = y] \neq \Pr[X = x] \cdot \Pr[Y = y]$. To describe the dependencies of such a set of random variables T occurring as intermediate values during the computation of a masked circuit, we use a *dependency graph* (DG), represented as a *directed labeled* graph where all edges have a source and destination node. Further, each edge has a label containing at least one variable $x \in T$. For any subset of random variables $S \subset T$, we get a subgraph consisting of all edges whose labels contain at least a variable in S. Further, the edges of the dependency graph are linked so that any subset S of variables describes a subgraph with no loops (a *loop* is a path with the same starting and ending point) if the set S consists of random variables that are independent of the decoded inputs or outputs of the circuit.

Definition 3. Let C be a masked circuit C with intermediate values T, and $\mathcal{G}$ a labeled graph with k edges $e_0, e_1, \ldots, e_{k-1}$ each labeled with T_i such that $\bigcup_{i\in[k]} T_i = T$. $\mathcal{G}$ is a dependency graph if for

each sub graph $\mathcal{G}' \subset \mathcal{G}$ with edges e_i $i \in I \subset [k]$, it holds

$$\mathcal{G}' \text{ has no loop} \Rightarrow \bigcup_{i \in I} T_i \text{ is independent of the unmasked inputs.}$$

Dependency graphs are helpful for three reasons: First, when we consider the leakage as a subset S of the random variables T, we can represent the leakage as a subgraph, the so-called leakage diagram, $\mathcal{LD}$ (see Definition 5). Second, using the graph property that all subsets with elements dependent on the decoding of the inputs are structured as loops; we can classify leakage diagrams as "good" or "bad" (see Section 4.1). All "good" leakage diagrams correspond to leakages that can be simulated without knowing sensitive values. Third, with the classification, we can upper bound the probability that the leakage corresponds to a "bad" leakage diagram if all wires leak their values with probability p. Hence, we can analyze the leakage resilience of a circuit (Sec. 5). In the following, we describe the dependency graphs of a simple encoding and for each gadget used by our compiler. Then, we show how to compose the dependency graphs of our gadgets to get the dependency graph for any output of our compiler.

Dependency graphs of masked values. Let us consider an encoding of a secret x, with $(x_i)_{i \in [n]} \leftarrow \texttt{Enc}(x)$. The corresponding set of random variables is

$$T = \{x_0, x_1, \ldots, x_{n-1}\} .$$

We represent this with the dependency graph depicted in Figure 5a with $T_i = \{x_i\}$. There are n edges, each labeled with one of the variables of T. For simplicity, we call the edge labeled with $\{x_i\}$ the x_i-edge. All n edges form a loop, which we can see as a "circle" [3]. It is easy to see that this "circle" is the only loop in the graph, and any strict subset of the n edges does not form a loop. This describes the abovementioned property that dependent random variables form a loop. The variables $x_0, x_1, \ldots, x_{n-1}$ describe a loop because they depend on the secret with $\texttt{Dec}((x_i)_{i \in [n]}) = x$. However, any strict subset $S \subset T$ is a set of independent random variables due to the security property of the secret sharing, and that is why they do not form a loop in the dependency graph.

Further, the dependency graph is a directed graph, and the x_i-edge connects the destination node of the x_{i-1}-edge with the source node of the x_{i+1}-edges.[4] The direction of an edge represents the sign of the edges labels. Thus, we can also think that there is an edge labeled $-x_i$ which connects the source node of the x_{i+1}-edge with the destination node of the x_{i-1}-edge. In detail, the path ove rall x_i's can be considered as the sum over all x_i's. If the path also consists of edges with opposite directions, we subtract the variables represented as an edge with the opposite direction instead of adding them. Since the random variables of the `Rand` and `Const` gadget only consist of output variables, their dependency graphs are simple dependency graphs of maskings. Next, we give the dependency graph of `pRef`.

Dependency graph for `pRef`. Let `pRef` refresh an input $(x_i)_{i \in [n]}$ with random values $(r_i)_{i \in [n]}$. As intermediate variables it computes $(b_i)_{i \in [n]}$ with $b_i = x_i + r_i$ and outputs $(y_i)_{i \in [n]}$ with $y_i = b_i - r_{i-1}$. This leads to the set of random variables

$$T_{\texttt{pRef}} = \{x_0, x_1, \ldots, x_{n-1}, r_0, r_1, \ldots, r_{n-1}, b_0, b_1, \ldots, b_{n-1}, y_0, y_1, \ldots, y_{n-1}\}.$$

The dependency graph of `pRef` is depicted in Figure 5b. Each edge is labeled by a single value of $T_{\texttt{pRef}}$. Thus, we can use the same convention as before, and the edge labeled with $\{x\}$ is the x-edge for all $x \in T_{\texttt{pRef}}$. The dependency graph (Fig. 5b) forms a skeleton of a cylinder. The $x_0, ..., x_{n-1}$-edges form a loop, which is the bottom circle of the cylinder, while the $y_0, ..., y_{n-1}$-edges form a loop, which is the top circle of the cylinder. These two loops are identical to the dependency graph of masked values described in the previous paragraph. The remaining edges form the lateral

[3] In the following, when we use *circles* and *rectangles* for the elemental geometrical shapes, while *loops* for the graph loops defined before. Clearly, "circles" and "rectangles" are loops if they are defined only with the edges of a graph.

[4] To simplify the notion, we omit the (mod n) in all the operations with the index of variables, as for $i - 1$.

surface of the cylinder. More precisely, this lateral surface consists of n rectangles defined by the x_i, r_i, y_i, r_{i-1}-edges. The loops defined by those rectangles describe the subset $\{x_i, r_i, y_i, r_{i-1}\}$ of dependent random variables because $x_i + r_i - r_{i-1} = y_i$. Here it becomes clear why the dependency graph is a directed graph because the r_{i-i} has the opposite direction when we consider the alternative path r_{i-1}, x_i r_i that connects the same nodes as y_i. Hence we only add x_i and r_i but subtract r_{i-1} to compute y_i. Further, the n rectangles each have a diagonal edge: the b_i-edges that describe the remaining intermediate values b_i. We add them to the graph such that they fulfill the same additive properties as the n rectangles. In detail, an alternative path for the edge b_i is r_{i-1}, y_i or x_i, r_i because it holds $b_i = r_{i-1} + y_i$ and $b_i = x_i + r_i$. This construction fulfills the property again that all subsets $S \subset T_{\texttt{pRef}}$ that depend on the decoding of the input (or output) form a loop in the graph. More precisely, they form a loop that orbits the lateral surface of the cylinder structure of the graph. In Proposition 6 (Sec. 4.2), we give the formal proof.

Dependency graph for `Copy`. The copy gadget `Copy` (Fig. 3b) takes as input an encoding $(a_i)_{i\in[n]}$ and outputs two encodings $(b_i)_{i\in[n]}$ and $(c_i)_{i\in[n]}$ with $a_i = b_i = c_i$ for all i. Note that the dependency graph only considers the random variables carried by the wires, and a_i, b_i, and c_i represent the same variable. Thus,

$$T_{\texttt{Copy}} = \{a_0, ..., a_{n-1}, b_0, ..., b_{n-1}, c_0, ..., c_{n-1}\} = \{a_0, \dots, a_{n-1}\}$$

results in the same dependency graph as the usual masking described above. The dependency graph is the graph depicted in Figure 5a, with $T_i = \{a_i, b_i, c_i\} = \{a_i\}$. Again, it is clear that the graph is a dependency graph because any set of possible leaked values is a set of independent values if the set does not describe a subgraph with a loop.

Dependency graph for `Add`. The addition gadget `Add` (Fig. 3a) is a share-wise addition. It takes as input two encodings $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$ and outputs an encoding $(c_i)_{i\in[n]}$ with $c_i = a_i + b_i$ for all $i \in [n]$. This leads to

$$T_{\texttt{Add}} = \{a_0, ..., a_{n-1}, b_0, ..., b_{n-1}, c_0, ..., c_{n-1}\}.$$

A possible dependency graph is depicted in Figure 5a with $T_i = \{a_i, b_i, c_i\}$. Compared with the graphs presented before, the difference is that each edge represents multiple different variables. More precisely, we map all values of each share-wise computation to a single edge. However, it holds again that any strict sub-graph has no loop and describes a subset that is independent of the decoded input or output (see Sec. 4.2 Prop. 5). Note that this fact immediately follows with the same argument as the one for dependency graphs of usual masking when we consider the approximation that an adversary learns all variables a_i, b_i, c_i if at least one is leaked.

Dependency graph for `Mult`. The `Mult` gadget takes as input two encodings $(a_i)_{i\in[n]}$, and $(b_i)_{i\in[n]}$, and outputs an encoding $(c_i)_{i\in[n]}$ with

$$\texttt{Dec}((c_i)_{i\in[n]}) = \texttt{Dec}((a_i)_{i\in[n]}) \cdot \texttt{Dec}((b_i)_{i\in[n]}) \ .$$

Therefore, the gadget computes the intermediate values $z_{l,i,j}$, and $w_{i,j}$, as defined in Figure 3e. This leads to the set of random variables generated by the circuit

$$T_{\texttt{Mult}} = \{a_0, ..., a_{n-1}, b_0, ..., b_{n-1}, a_i \cdot b_j, z_{l,i,j}, w_{i,j} \text{ and } i, j \in [n], l \in [L+1]\},$$

with $L = \log_2(n+1)$. Considering Figure 3e, it turns out that $w_{i,j}$ and $z_{l,i,j}$ are not defined for all j and l. For the sake of simplicity, we omit the precise treatment and assume that all the variables $w_{\widehat{i,j}}$ not defined by the algorithm are zero. We see them as not elements in $T_{\texttt{Mult}}$. A possible dependency graph is depicted in Figure 5a with

$$T_i = \{a_i, b_i, c_i, a_i \cdot b_i, a_i \cdot b_j, a_j \cdot b_i, w_{i,j}, w_{j,i}, z_{l,i,j} \text{ and } j \in [n], l \in [L+1]\} \ .$$

It is very similar to the graph used for Add. The idea is to label the T_i-edge with the i^{th} share of the inputs and outputs a_i, b_i, c_i. The difference to Add is that Mult also has intermediate values that we still have to add to the graph. Therefore, we add to T_i all the monomials $a_i b_j$ and $a_j b_i$, $j \in [n]$. Note the monomial $a_i b_j$ (and $a_j b_i$) belongs to *both* labels T_i and T_j. Finally, we add all intermediate addends $w_{i,j}$s and the $z_{l,i,j}$ of the i^{th} output share to T_i. This is inspired by [22, 16]. In Section 4.2 (Proposition 5), we prove that the variables $S \subset T_{\texttt{Mult}}$ that do not describe a sub-graph with a loop are independent of the decoding of $(a_i)_{i\in[n]}$, $(b_i)_{i\in[n]}$, and $(c_i)_{i\in[n]}$. Next, we give the composition results to construct the dependency graph of any output of our compiler.

3.2 Composition of Dependency Graphs

In the previous section, we introduced the dependency graphs for our gadgets. Since our compiler always outputs a composition of those gadgets, we are interested in how to get the dependency graphs for the composition of those gadgets. Therefore, we give composition results to obtain the dependency graphs of composed gadgets G_1 and G_2 with dependency graphs DG_1 and DG_2, respectively. In [11], they distinguish two different compositions, the sequential composition written $G = \mathsf{G}_1 \circ \mathsf{G}_2$ where G_1 gets as input the output of G_2, or the parallel composition written $G = \mathsf{G}_1 \| \mathsf{G}_2$ where both gadgets compute parallel and independently of each other. When we consider parallel compositions of two gadgets where both gadgets run independently (with no shared inputs), it is easy to see that the dependency graphs of both gadgets do not affect each other. Hence, the dependency graph of $\mathsf{DG}_{\mathsf{G}_1\|\mathsf{G}_2}$ can be seen as a union of sets $\mathsf{DG}_1 \cup \mathsf{DG}_2$ where both graphs are considered as one graph but there is no edge connecting DG_1 and DG_2 because there is no further dependency generated by the parallel composition. To compute the dependency graph of the sequential composed gadgets $\mathsf{G}_1 \circ \mathsf{G}_2$ out of DG_1 and DG_2 we use a modified union of both dependency graphs. When we consider sequential compositions, an output wire becomes an input wire of another gadget. Hence, two wires merge to only one wire, and a modified union is required where the two edges of such connected wires become the same. For this reason, we define a function f (so-called attaching function) that maps the edges of the G_2's output wires to the according edges of the G_1's input wires. The result is a union of both graphs where f defines which edges of DG_1 and DG_2 are the same, and we can write

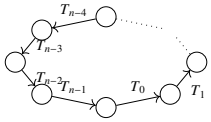
$$\mathsf{DG}_{\mathsf{G}_1\circ\mathsf{G}_2} = \mathsf{DG}_{\mathsf{G}_1} \cup_f \mathsf{DG}_{\mathsf{G}_2}$$

For example, let $\mathsf{G}_1 = \texttt{pRef}$ and $\mathsf{G}_2 = \texttt{Add}$, then the output $(c_i)_{i\in[n]}$ is the input of pRef. This can be described with the attaching function f that maps the edge of c_i in $\mathsf{DG}_{\texttt{pRef}}$ to the edge of T_i in $\mathsf{DG}_{\texttt{Add}}$, and the resulting dependency graph $\mathsf{DG}_{\texttt{pRef}\circ\texttt{Add}} = \mathsf{DG}_{\texttt{pRef}} \cup_f \mathsf{DG}_{\texttt{Add}}$ is depicted in Figure 6a where T_i is labeled with the inputs and outputs of the addition gadget $\{a_i, b_i, c_i\}$ as the dependency graph of Add (Fig. 5a). Further, due to the composition, the function f merges the edges related to the output of the addition with the edges related to the input of the refresh. For this reason, the edge labeled with T_i is also the edge that represents the input edge of the dependency graph of pRef (Fig. 5b).

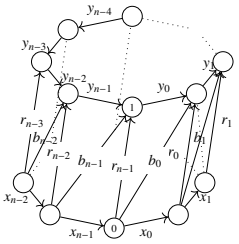
Additionally, we can also refresh the inputs of the addition. Let G be an addition or multiplication gadget with $(c_i)_{i\in[n]} \leftarrow \mathsf{G}((a_i)_{i\in[n]}, (b_i)_{i\in[n]})$ where $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$ are refreshed outputs $(a_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$, $(b_i)_{i\in[n]} \leftarrow \texttt{pRef}((y_i)_{i\in[n]})$, respectively, and $(c_i)_{i\in[n]}$ is refreshed afterwards $(z_i)_{i\in[n]} \leftarrow \texttt{pRef}((c_i)_{i\in[n]})$. This composition can be written as $\texttt{pRef} \circ \mathsf{G} \circ (\texttt{pRef}\|\texttt{pRef})$ because the refresh of $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$ is a parallel composition, and the remaining ones are sequential. This results in a dependency graph

$$\mathsf{DG}_{\texttt{pRef}(\mathsf{G}(\texttt{pRef}(\cdot),\texttt{pRef}(\cdot)))} = \mathsf{DG}_{\texttt{pRef}} \cup_{f_1} \mathsf{DG}_{\mathsf{G}} \cup_{f_2} (\mathsf{DG}_{\texttt{pRef}} \cup \mathsf{DG}_{\texttt{pRef}})$$

depicted in Figure 6b with f_1 mapping the input edges $\mathsf{DG}_{\texttt{pRef}((c_i)_{i\in[n]})}$ to the output edges of DG_{G}, and f_2 input edges DG_{G} to the output edges $\mathsf{DG}_{\texttt{pRef}((x_i)_{i\in[n]})}$ and $\mathsf{DG}_{\texttt{pRef}((y_i)_{i\in[n]})}$. Note that T_i is determined by the choice of G, and in case of Add it is $\{a_i, b_i, c_i\}$. Formally, the operation defined by $\mathsf{DG}_1 \cup_f \mathsf{DG}_2$ is a topological definition called *adjunction space* and can be formalized as follows.



(a) Dependency graph of `Enc`, `Dec`, `Copy`, `Add`, or `Mult` with labels T_i defined in Section 3.1.



(b) The dependency graph for $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$.

Figure 5: Dependency graphs of the gadgets.

Definition 4 (Attaching)**.** Let DG_1 and DG_2 two disjoint dependency graphs, and f a function as described above mapping some edges of DG_1 to edges in DG_2. The composed graph is

$$\mathsf{DG}_1 \cup_f \mathsf{DG}_2 = (\mathsf{DG}_1 \cup \mathsf{DG}_2)/\sim,$$

where $\sim$ is the smallest equivalence relation with $x \sim f(x)$.

The adjunction space of two dependency graphs preserves the properties of the underlying dependency graphs and merges them in such a way that the resulting graph describes the dependencies from both dependency graphs simultaneously. For the sake of simplicity, we can consider the composition as described above, where we merge the edges of two dependency graphs if the according wires become one due to the circuit composition. More formally, we will prove in Propositions 5, 6, 8, and Theorem 10 that the variables contained *only* in the labels of the edges of a sub-graph that does not contain a loop are independent of the inputs or outputs.

The key observation is that the dependency graphs for all our gadgets are either the loop depicted in Figure 5a or the skeleton of a cylinder, as shown in Figure 5. Further, the compiler places a refresh gadget between every gadget that is not a refresh gadget. This means that the resulting dependency graph can be seen as a composition of cylinders (defined by the refresh gadgets), where the bottom and the top of the cylinder are labeled with the T_i's defined of the gadgets between the refresh gadgets.

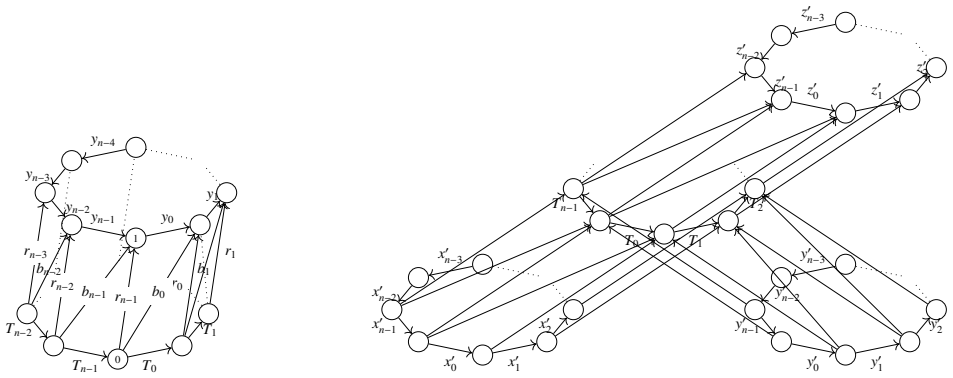
4 Security Analyzes of the Gadgets

Before we analyze the privacy of our compiler's output in Section 5, we first give the privacy of our gadgets in this section. First, in Section 4.1, we formally show how to describe the leakage with sub-graphs of the gadgets' dependency graphs presented in the previous chapter (Sec. 3.1). Then, in Section 4.2, we give all leakages that are not shiftable using the sub-graphs of our dependency graph, and finally, we compute the probabilities of such sub-graphs under the condition that each wire leaks its value with probability p.

4.1 Leakage Diagram

Using Corollary 3, it is enough to show shiftability for the privacy proof. To characterize which outputs $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ of the experiment in Definition 1 are shiftable, we start representing the leakage as a subgraph of the dependency graph. As already discussed in the random probing model, the adversary receives via leakage the values carried by some of the wires, $\mathcal{L}_p(\widehat{\mathsf{C}}) \subseteq \mathcal{W}'(\widehat{\mathsf{C}})$ (Def. 1). We can represent these variables as a subgraph of the dependency graph.

Definition 5 (Leakage diagram)**.** Let DG be the dependency graph of the circuit $\widehat{\mathsf{C}}$ and $\mathcal{L}_p(\widehat{\mathsf{C}})$ be the set of wires that leak in the experiment $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$.The *leakage diagram*, $\mathcal{LD}(\mathcal{L}_p(\widehat{\mathsf{C}}), \mathsf{C})$,



(a) Dependency graph of $\texttt{pRef} \circ \mathsf{G}$, with $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ taking as input the output of G. The DG_{G} is described in Figure 5a while $\mathsf{DG}_{\texttt{pRef}}$ is described in Figure 5b. Note that the label T_i contains x_i.

(b) The dependency graph of the composition of $(x_i)_{i\in[n]} = \texttt{pRef}((x'_i)_{i\in[n]})$ with $(y_i)_{i\in[n]} = \texttt{pRef}((y'_i)_{i\in[n]})$, $(z_i)_{i\in[n]} = \mathsf{G}((x_i)_{i\in[n]}, (y_i)_{i\in[n]})$ (thus, G is either the `Add` or the `Mult` gadget) and $(z'_i)_{i\in[n]} = \texttt{pRef}((z_i)_{i\in[n]})$. (For simplicity, we have omitted the edges of b_is and r_is.)

Figure 6: Dependency graphs of composed gadgets.

corresponding to the leakage $\mathcal{L}_p(\widehat{\mathsf{C}})$ is the subgraph of DG composed by all edges whose label contains at least one of the variables carried by the wires in $\mathcal{L}_p(\widehat{\mathsf{C}})$.

Since $\mathcal{L}_p(\widehat{\mathsf{C}})$ is randomized by the leakage probability p, it is a random variable over all possible subsets of the wires $\mathcal{W}'(\widehat{\mathsf{C}})$ that may leak during the computation of $\widehat{\mathsf{C}}$. Hence, also $\mathcal{LD}(\mathcal{L}_p(\widehat{\mathsf{C}}), \widehat{\mathsf{C}})$ is a random variable over all possible sub-graphs of the dependency graph. Next, we give some examples of leakage diagrams LD_i with $\Pr[\mathcal{LD}(\mathcal{L}_p(\widehat{\mathsf{C}}), \widehat{\mathsf{C}}) = LD_i] > 0$. They are also represented in the full version. For this reason, we consider the `pRef` gadget refreshing an encoding $(x_i)_{i\in[n]} \leftarrow \texttt{Enc}(x)$ of a secret $x \in \mathbb{F}$

$$(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$$

with random values r_i, and intermediate values $y_i = b_i - r_{i+1}$ and $b_i = x_i + r_i$ defined in Figure 1b.

1) $LD_1 = (x_0, ..., , x_{n-1})$ reveals the secret because $\texttt{Dec}((x_i)_{i\in[n]}) = x$

2) $LD_2 = (y_0, ..., , y_{n-1})$ reveals the secret because $\texttt{Dec}((y_i)_{i\in[n]}) = \texttt{Dec}((x_i)_{i\in[n]}) = x$

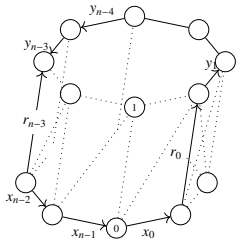
3) Let $LD_3 = (x_0, ..., x_{i-1}, r_{i-1}, y_i, r_i, x_{i+1}, ..., x_{n-1})$. Since $y_i = x_i + r_i - r_{i+1}$, we have that these values reveal x. In fact,

$$\left(\sum_{j\in[n], j\neq i} x_j\right) + r_i + y_i + r_{i+1} = \left(\sum_{j\in[n], j\neq i} x_j\right) + x_i = \texttt{Dec}((x_i)_{i\in[n]}) = x.$$

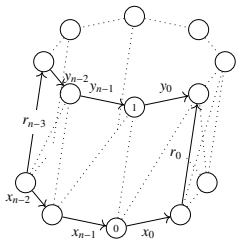
4) Let $LD_4 = (x_0, ..., x_i, r_i, y_{i+1}, ..., y_{n-1})$. We observe that if instead of r_{n-1}, we have $r'_{n-1} = r_{n-1} + x' - x$, these values come from a refreshing of x'. In fact

$$\left(\sum_{j=0}^{i-1} x_j\right) + r_i + \left(\sum_{j=i}^{n-1} y_j\right) = \left(\sum_{j=0}^{n-1} x_j\right) - r_{n-1} = x - r_{n-1}$$

since $r_{j-1} + y_j = x_j + r_j$. Thus, LD_4 does not reveal x because the leakage is shiftable to x'. Note that, $LD'_4 = LD_4 \cup \{r_{n-1}\}$ would reveal x similarly to B3).



(a) Leakage diagram of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ with an orbiting loop.



(b) Leakage diagram of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ with a not orbiting loop.

Figure 7: Dependency graphs of the gadgets.

5) Now, consider $LD_5 = \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{i-1}, y_{i+1}, ..., y_{n-1}, r_0, ..., r_{n-1}\}$. LD_5 does not reveal x since $LD_5 \cup \{x_i\}$ comes from $\mathcal{L}(\texttt{pRef}((x_i)_{i\in[n]}, \mathbf{r}), p)$, while $LD_5 \cup \{x_i + x' - x\}$ comes from $\mathcal{L}(\texttt{pRef}((x'_i)_{i\in[n]}, \mathbf{r}), p)$, where $(x'_i)_{i\in[n]}$ is defined as $x'_j = x_j$ if $j \neq i$ and $x'_i = x + i + x' - x$, $\mathbf{r} = (r + 0, ..., r_{n-1})$. We prove this in Section 4.2, Proposition 6.

6) Finally, consider $LD_6 = (x_1, b_1, r_1)$, which clearly does not reveal the secret, as we will prove in the next section.

Considering the examples above, we observe that if the leakage diagram reveals information about the secret, the leakage diagram also consists of at least one loop. This is not surprising because this is exactly the property that we presented in Section 3, and immediately follows from the dependency graph property described in Section 3.1. However, the presence of a loop does not always prevent shiftability, e.g., LD_4, LD_5, ad LD_6 have a loop and are still shiftable. For tightness reasons, we want to distinguish loops that reveal the secret from loops that do not reveal the secret. For this reason, we remember that the dependency graphs of the gadgets (Fig. 5) are either a circle or the skeleton of a cylinder. Hence, its compositions are composed *hollow* cylinders, as depicted in Figure 6. We observe all loops revealing the secret *orbit* around this hollow cylinder structure. Further, the loops that do not reveal the secret do not orbit the hollow structure. Figure 7 illustrates the differences between such loops. The first loop is an example of a loop that does orbit the skeleton, while the second one gives an example of a loop that does not orbit the skeleton. In the following, we say that a leakage diagram *orbits* if it contains such an orbiting loop. Similarly to the adjunction space used to compose the dependency graphs, also the property of *hollow* and *orbit* are well-known in Topology and can be described with the topological definitions *simply connected* and *homotopically equivalent*: *Hollow* means that all the loops carrying the shares of a value are not *simply connected*, and *orbiting* means that the loop is *homotopically equivalent* to a loop containing all x_i-edges of a sharing $(x_i)_{i\in[n]}$. Roughly speaking, two paths are *homotopically equivalent* (Defined in the full version) if we can put an infinitely elastic rope over the first path, and we can slide it to cover the second path. For this, we assume that the DG of pRef contains the lateral surface of the cylinder, and the rope, as in LD_3, can slide over it. With this, it is easy to see that LD_3 is homotopically equivalent to LD_1 and LD_2. Thus, the Topology allows us to distinguish between "bad" leakage diagrams from good, and all not orbiting leakage diagrams represent shiftable leakages. This intuition will be proved in the following subsection (Sec. 4.2). In the following, we will also call "good" diagrams *shiftable diagrams*.

4.2 Security Analysis for our Gadgets

To avoid that, we have to prove the security of any possible leakage diagram. So we start with a useful observation.

Proposition 1. *If a leakage diagram LD is shiftable, all sub graphs LD′ with LD′ ⊂ LD are also shiftable.*

Hence, when we prove shiftability for a given leakage diagram, it immediately follows shiftability for any subgraph.

Proof. A leakage diagram LD is shiftable if all the outcomes L of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ which are represented by the leakage diagram LD are shiftable to $\mathbf{x'}$ for any $\mathbf{x'} \neq \mathbf{x}$. That is, (Def. 2) L can be also an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$-experiment.

Now, consider the leakage diagram LD' with $LD' \subset LD$. Let L' be an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$, which can be represented by the leakage diagram LD'. Since $LD' \subset LD$, there exists an outcome L of the Leak experiment represented by LD s.t. $L_{|\text{var} \in LD'} = L'$, where with $var \in \mathcal{LD}'$, we mean the variables carried only by the labels of the edges in LD'. Since LD is shiftable, then L can be an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$-experiment. Thus, $L' = L_{|\text{var} \in LD'}$ can be an outcome of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x'}, p)$-experiment. Thus, all outcomes of the Leak-experiment represented by LD' are shiftable to $\mathbf{x'}$. Thus, the leakage diagram LD' is shiftable. □

For simplicity, we use this result and introduce the concept of *maximal diagram* $\mathcal{MAX}(LD)$ for any leakage diagram LD of our gadgets. $\mathcal{MAX}(LD)$ is a subgraph of the dependency graph, which contains LD

$$LD \subseteq \mathcal{MAX}(LD) \subset \mathsf{DG}$$

and is maximal in the sense that if we add any further edge, it orbits as depicted in Figure 7a. Due to Proposition 1 we only need to consider all possible maximal diagrams for the security proofs of our gadgets when we want to show that leakage diagrams are shiftable if they do not orbit. The existence of maximal diagrams is proved in the following Proposition.

Proposition 2. *Let LD be a leakage diagram that does not orbit. Then, there exists a maximal diagram LD' containing it.*

Proof. Let LD be a subgraph of DG, which does not orbit. Suppose that $\forall$ edge $e \in \mathsf{DG} \setminus LD$, $LD \cup \{e\}$ orbits. Then, LD is a maximal simply connected subgraph containing LD.

Otherwise, there is an $e \in \mathsf{DG} \setminus LD$ s.t $LD^1 = LD \cup \{e\}$ that does not orbit. Then, we iterate with LD^1 until there is no such an e left. Since there is only a finite number of edges that can be added, this sequence of subgraphs must end after at most I step (with $I \leq |\{e \in \mathsf{DG} \setminus \mathcal{LD}\}|$). Hence, the final LD^j is a maximal diagram. □

In general, there are many different maximal diagrams, as we will see in the next section.

Maximal diagrams. Since we use the maximal diagrams in most of the following proofs, we characterize them for all of our gadgets. The dependency graphs of `Enc`, `Dec`, `Copy`, `Add`, `Mult` (Fig. 5a) are a circle with n edges, all subgraphs containing all edges except one are maximal, that is, we have n different maximal diagrams M_i defined as follows

$$\mathcal{M}_i = \{T_0, ...T_{i-1}, T_{i+1}, ..., T_{n-1}\} = \mathsf{DG} \setminus \{T_i\}$$

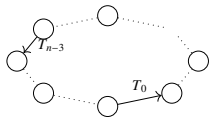
where with $T_j \in \mathcal{M}_i$, we mean that $\mathcal{M}_i$ contains the edge labeled with T_j as depicted in Figure 8b and 8b. We formally prove this in the following Proposition.

Proposition 3. *Let* DG *be the dependency graph of one of the* `Enc`, `Dec`, `Copy`, `Add`, *and* `Mult` *gadget. Then, there are n different maximals (Sec. 3.1)*

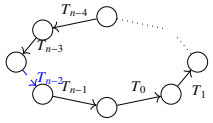
$$\mathcal{M}_i = \{T_0, ..., T_{n-1}\} = \mathsf{DG} \setminus \{T_i\}$$

for $i = 0, ..., n-1$.

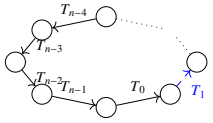
Proof. $\mathcal{M}_i$ is a line starting from $node_{i+1}$, the starting node of the T_{i+1}-edge, to $node_i$, the arriving node of the T_{i-1}-edge (and the starting node of the T_i-edge). Thus, it is simply connected. Moreover, $\mathcal{M}_i \cup \{T_i\} = \mathsf{DG}$, where T_i is the only edge in DG and not in $\mathcal{M}_i$. Since DG orbits, $\mathcal{M}_i$ is maximal. In fact, any other subset LD that does not orbit and is composed of less than $n-1$ edges is a subset of one of these $\mathcal{M}_i$. Hence, there is no other maximal apart from the $\mathcal{M}_i$'s. □



(a) A possible leakage diagram LD only consisting of two edges (solid).



(b) Maximal leakage diagram $\mathcal{MAX}(LD) = \mathcal{M}_{n-2}$ (solid, black edges).



(c) Maximal leakage diagram $\mathcal{MAX}(LD) = \mathcal{M}_1$ (solid, black edges).

Figure 8: Two possible maximal leakage diagrams (Fig. 8b and 8b) of a possible leakage diagram LD (Fig. 8a). The blue dashed edge is the missing one so that the diagram does not orbit.

Further, note that $\mathcal{MAX}(LD)$ is not always unique. As an example, suppose that $LD = \emptyset$, then, there are n different maximal simply connected subgraph $\mathcal{MAX}^i$ containing LD, with

$$\mathcal{MAX}^i = \{e_0, ..., e_{i-1}, e_{i+1}, ..., e_{n-1}\},$$

that is, $\mathcal{MAX}^i$ contains all edges except e_i. All the $\mathcal{MAX}^i$'s contain LD, do not orbit, and, adding the only remaining edge (e_i), they orbit.

The dependency graph of `pRef` describes the skeleton of a cylinder (Fig. 5b). There are two families of maximal diagrams (both depicted in Figure 9): The first family is called $\mathcal{M}^{\text{right}}$ since there is a gap of missing edges turning right, which we call *RGap* (Fig. 9a).

$$\mathcal{M}^{\text{right}}_{i,j} := \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, r_j, ..., r_{i-1}, b_j, ..., b_i\},$$

$$RGap_{i,j} = \{x_i, y_j, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\}$$

for $i, j \in [n]$ with $\mathcal{M}^{\text{right}}_{i,j} = \mathsf{DG} \setminus RGap_{i,j}$. Note that $r_j, ..., r_{i-1}$ is a short way to write: if $i < j$ $r_0, ...r_{i-1}, r_j, ..., r_{n-1}$; if $i = j$ $\emptyset$; if $i > j$ $r_{j+1}, ..., r_{i-1}$. The same holds for the $b_{i'}$s. The second family is called $\mathcal{M}^{\text{left}}$ since there is a gap, *LGap* (Fig. 9b), which turns left (or stays straight for $LGap_{i,i}$)

$$\mathcal{M}^{\text{left}}_{i,j} := \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\},$$

$$LGap_{i,j} = \{x_i, y_j, r_j, ..., r_{i-1}, b_j, ..., b_i\}$$

for $i, j \in [n]$ with $\mathcal{M}^{\text{left}}_{i,j} = \mathsf{DG} \setminus LGap_{i,j}$. The proof that these are maximal diagrams and that the classification is complete can be found in the following Proposition:

Proposition 4. *The maximal diagrams of* $\mathsf{DG}_{\texttt{pRef}}$ *are either of the type* $\mathcal{M}^{right}_{i,j}$ *or* $\mathcal{M}^{left}_{i,j}$ *with* $i, j \in [n]$

Proof. In other words, if LD is a maximal diagram of $\mathsf{DG}_{\texttt{pRef}}$, there are $i, j \in [n]$ s.t. either $LD = \mathcal{M}^{\text{right}}_{i,j}$ or $LD = \mathcal{M}^{\text{left}}_{i,j}$.

First, we start observing that if LD is a maximal diagram, then it must be connected. Otherwise, since it does not orbit, thus all connected components are simply connected (since the fundamental group of $\mathsf{DG}_{\texttt{pRef}}$ is $\mathbb{Z}$). Thus, we can deform them homotopically to be points. Now adding a single edge to a set of points, it cannot make it not simply connected (and, thus, orbiting).

Second, in any maximal diagram, there exists one and only one $i \in [n]$ s.t. $x_i \notin LD$, and one and only one $j \in [n]$ s.t. $y_j \notin LD$. In fact, let us suppose that there exist two edges x_i and $x_{i'}$ not in LD. Let $node_i$ and the $node_{i'}$ be the source nodes of the edge x_i, and $x_{i'}$, respectively, $node_{i+1}$ and $node_{i'+1}$ be their respective destination nodes. Now, look at LD. Suppose that $i \neq i' + 1, i' - 1$. Since LD is maximal, there is a path *path* from $node_{i+1}$ to $node_i$ s.t. $path \cup \{x_i$ orbits. Similarly, there is a path $path'$ from $node_{i'+1}$ to $node_{i'}$ s.t. $path' \cup \{x_{i'}$ orbits. For the structure of $\mathsf{DG}_{\texttt{pRef}}$ *path* and $path'$ meet in two points. Thus, taking a part of *path* and a part $path'$, we have a path that orbits. Therefore LD is not maximal since it orbits. Now, we observe that if $x_i \notin LD$ then $\{b_i, r_i\} \not\subseteq LD$, otherwise, the path

$$x_0, ..., x_{i-1}, b_i, r_i, x_{i+1}, ..., x_{n-1}$$

is a loop in LD orbiting around the hollow graph, thus, LD would orbit. Thus, either $r_i \in LD$ or $b_i \in LD$. Similarly, if $y_j \notin LD$ then $\{b_j, r_j\} \nsubseteq LD$, otherwise, the path

$$y_0, ..., y_j, r_j, b_j, y_j, ..., y_{n-1}$$

is a loop in LD orbiting around the hollow graph. Here, we consider the case $i = j$. It depends if either b_i or r_i belongs to LD

- *Case $r_i \in LD$. ($\mathcal{M}^{left}_{i,i}$)* Since we cannot have both b_i and r_i in a maximal simply connected subgraph containing $\{x_i, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{i-1}, y_{i+1}, ..., y_{n-1}\}$, the natural maximal diagram is

$$LD' = \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{i-1}, y_{i+1}, ..., y_{n-1}, \\ r_0, ..., r_{n-1}, b_0, ..., b_{i-1}, b_{i+1}, ..., b_{n-1}\}.$$

  We observe that it does not orbit since there is no loop turning around the hollow graph since the square defined by x_i, r_i, y_i and r_{i-1} is never crossed. Moreover, if we add only the edge in $\mathsf{DG} \setminus LD$, that is, b_i, we have a not simply connected subgraph since it is DG. Thus, LD' is a maximal simply connected subgraph.
- *Case $b_i \in LD$. (Case $\mathcal{M}^{right}_{i,i}$)* We observe that

$$LD' := \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{i-1}, y_{i+1}, ..., y_{n-1}, b_i\}$$

  does not orbit since there is no non-trivial loop. In fact,

$$x_{i+1}, ..., x_{n-1}, x_0, ..., x_{i-1}, b_i, y_{i+1}, ..., y_{n-1}, y_0, ..., y_{i-1}$$

  is a line which is not a loop.
  Moreover, if we add r_l to LD', we have that

$$y_0, ..., y_l, r_l, x_{l+1}, ..., x_{i-1}, b_i, y_{i+1}, ..., y_{n-1} \quad \text{for } l < i, \quad \text{or}$$

$$x_0, ..., x_{i-1}, b_i, y_{i+1}, ..., y_l, r_l, x_{l+1}, ..., x_{n+1}, ..., x_{n-1} \quad \text{for } l > i, \quad \text{or}$$

  is a loop in $LD' \cup \{r_l\}$ which orbits [5]. Similarly, if we add b_l to LD, we have that

$$y_0, ..., y_l, b_l, x_l, ..., x_{i-1}, b_i, y_{i+1}, ..., y_{n-1} \quad \text{for } l < i, \quad \text{or}$$

$$x_0, ..., x_{i-1}, b_i, y_{i+1}, ..., y_l, b_l, x_l, ..., x_{n+1}, ..., x_{n-1} \quad \text{for } l > i$$

  is a loop in $LD' \cup \{b_l\}$ which orbits. Thus LD' is a maximal simply connected subgraph.

The other cases (which are done in a similar way) can be found in the full version. □

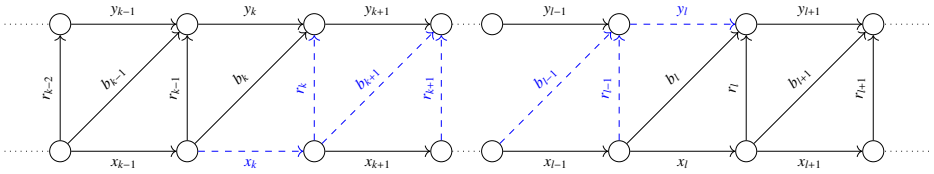
Using the maximal diagrams, we can prove the security of our gadgets with the help of Proposition 1.

Security for `Enc`, `Dec`, `Copy`, `Add`, `Mult`. Now, for all gadgets except `pRef`, we prove the condition mentioned in Section 4.1. In other words, we prove any leakage diagram that does not orbit implies that the leakage is shiftable to another input. Formally,

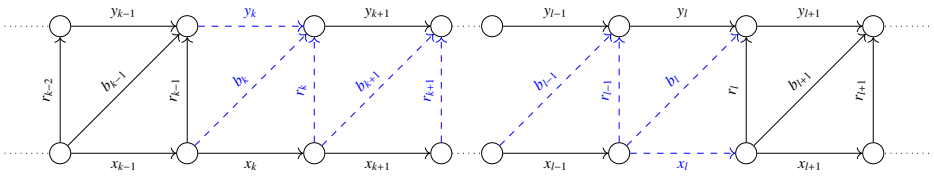
Proposition 5. *Let* G *be the gadget* `Add`, `Mult`, *or* `Copy` *defined in Figure 3. An outcome L of the* $\mathsf{Leak}(\mathsf{G}, \boldsymbol{x}, p)$ *experiment is shiftable to any* $\boldsymbol{x}$*', if the leakage diagram corresponding to L, does not orbit the dependency graph.*

We first give a high-level proof idea.

[5] The inequalities regarding l, i as all the inequalities in the remaining in the proof are done considering $l, i \in \mathbb{Z}$ (and not in $\mathbb{Z}_n$)].



(a) Edges in $RGap_{k,l}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.



(b) Edges in $LGap_{k,l}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.

Figure 9: Example of maximal leakage diagrams of `pRef`. The blue dashed edges represent $RGap_{k,l}$ and $LGap_{k,l}$, respectively. The remaining solid edges are the maximal leakage diagrams $\mathcal{M}^{\text{right}}_{k,l} = \mathsf{DG} \setminus RGap_{k,l}$ and $\mathcal{M}^{\text{left}}_{k,l} = \mathsf{DG} \setminus LGap_{k,l}$.

Proof sketch. The proof is substantially the same for all gadgets: we consider the maximal $\mathcal{M}_i$ and the values of the variables it contains, and we show that we can modify the values carried by the T_i label. Hence we can shift the i^{th} share of each input and output encoding such, and prove that this modification does not change the distribution of the values in M_i. Hence, the distribution of the values in M_i is the same for $\mathsf{Leak}(\mathsf{G}, \mathbf{x}, p)$ experiment and the shifted one $\mathsf{Leak}(\mathsf{G}, \mathbf{x'}, p)$. □

Next, we give the formal proof.

Proof. We prove the claim for all maximal graphs. Thus, using Proposition 1, we have that the claim holds for all not orbiting leakage diagrams. The high-level idea is that we can shift change the inputs and outputs of the edge that are not part of the maximal graph because they are uniformly distributed and not revealed due to the leakage. We start with the simplest case, (i) the masking of `Enc`, `Dec` and dependency graphs of `Copy`, then we prove the claim for (ii) `Add`, and (iii) `Mult`.

(i) The masking of `Enc`, `Dec` and dependency graphs of `Copy` since the edges of the dependency graph are labeled with a single variable. Thus, the input is x, and the variables leaked are those in the labels of the edges of $\mathcal{M}_i$ for an $i \in [n]$. $\mathcal{M}_i$ contains all x_j with $j \neq i$. If we modify the not leaked value x_i to $x'_i = x_i + x' - x$, we have shifted the output $\mathsf{Leak}(\mathsf{G}, x, p)$ to the output of the experiment $\mathsf{Leak}(\mathsf{G}, x', p)$, since $x_0, ..., x_{i-1}, x'_i, x_{i+1}, ..., x_{n-1}$ is an encoding of x'. Hence, $\mathcal{M}_i$ has the same distribution for x and x', and therefore $\mathcal{M}_i$ is shiftable for any $i \in [n]$.

(ii) The dependency graph's edges of `Add` are labeled with multiple variables. Let us suppose that `Add` takes as input $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$, and outputs $(c_i)_{i\in[n]}$, then each edge is labeled with $T_i = \{a_i, b_i, c_i\}$. Thus, the leaked variable of the maximal graph $\mathcal{M}_i$ are all a_j, b_j, and c_j with $j \neq i$. If we modify the values a_i, b_i, and c_i which are not leaked in $a'_i = a_i + a' - a$, $b'_i = b_i + b' - b$, and $c'_i = c_i + a' + b' - a - b$, we have shifted the output $\mathsf{Leak}(\texttt{Add}, (a, b), p)$ to the output $\mathsf{Leak}(\texttt{Add}, (a', b'), p)$. This hold because $d_0, ..., d_{i-1}, d'_i, d_{i+1}, ..., d_{n-1}$ is an encoding of d for $d = a, b, c$, and $a_j + b_j = c_j \; \forall j \neq i$, and $a'_i + b'_i = c'_i$. Note that the input sharings are uniformly distributed, and therefore the modified shares are also uniformly distributed.

(iii) `Mult`: let us suppose that `Mult` takes as input $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$, and outputs $(c_i)_{i\in[n]}$. Thus, the variables leaked are those *only* in the labels of the edges of $\mathcal{M}_i$ (Def. 5), that is, $a_{i'}, b_{i'}, c_{i'}, a_{i'}b_{i'}, a_{i'}b_j, a_j b_{i'}, w_{i',j}, w_{j,i'}, z_{l,i',j}$ for $i' \neq i$, and $j \in [n]$ and $l \in [0, ..., L]$ with $L = \log_2(n+1)$. Let $\delta_1 = a' - a$ and $\delta_2 = b' - b$. We replace the values of the variable in M_i

as follows: $a'_i = a' + \delta_1$, $b'_i = b_i + \delta_2$, $(a_i b_i)' := a_i b_i + \delta_1 b_i + \delta_2 a_i + \delta_1\delta_2$, $(a_i b_j)' := a_i b_j + \delta_1 b_j$, $(a_j b_i)' := a_j b_i + a_j \delta_2$, $(w_{i,j})' := w_{i,j} + \delta_1 b_j$, and $(w_{j,i})' := w_{j,i} + \delta_2 a_j$. For c'_i and $z_{l,i,j}$, since they are the sum of many $w_{i,j}$s and $a_{j'} b_i$, the modification is done according to the previous modifications. This can be seen as an output of the $\mathsf{Leak}(\mathtt{Mult}, (a', b'), p)$ as we now prove, modifying the proof of [16]:

In [16] is the proof for all values except for the $z_{l,i,j}$s. The only difference from our `Mult` from that in [16], is that the final addition is done sequentially thew (Figure 3e, while we do it parallel). But, we can observe that $\forall l = 1, ..., \log(n)$ and $\forall i, j$

$$z_{l,i,j} = a_i b_i + \sum_{I=2^l j}^{2^l(j+1)-1} z_{0,i,I} = a_i b_i + \sum_{I=0}^{2^l(j+1)-1} z_{0,i,I} - \sum_{I=0}^{2^l j-1} z_{0,i,I}$$

Hence, all $\sum_{I=0}^{2^l(j+1)-1} z_{0,i,I}$, $\sum^{2^l(j+1)-1} z_{0,i,j} - \sum^{2^l(j+1)-1} z_{0,i,j}$ are partial sums of the serial sum of the multiplication of Dziembowski et al. [16], and all these values are mapped on the ith edge. Thus, since their modifications cannot be detected, ours cannot be detected too.

In this way, we have shifted the output of the $\mathsf{Leak}(\mathtt{Mult}, (a, b), p)$ to an output of the experiment $\mathsf{Leak}(\mathtt{Mult}, (a', b'), p)$.

The results of (i),(ii), and (iii) conclude the proof. □

The proposition gives us the following privacy result for our gadgets.

Theorem 1. *Let* `Add`, `Mult` *and* `Copy` *be the gadgets defined in Figure 3. Then,* `Add` *and* `Copy` *are* $(p, (3p)^n)$*-private, for* $p \leq 1/3$*, and* `Mult` *is* $(p, \tilde{p})$*-private with*

$$\tilde{p} = 1 - (1 - p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1}.$$

Proof. Using Proposition 5 and the classification of maximal diagrams, the only problems happen if all edges are leaked. Thus, these gadgets are secure if the leakage diagram does not contain all edges. In other words, they are secure with probability $1 - \Pr[LD = \{T_0, ..., T_n\}]$ and thus, they are

$$(p, \Pr[LD = \{T_0, ..., T_n\}])\text{-private.}$$

We first bound $\Pr[LD = \{T_0, ..., T_n\}]$ for (i) `Add` and `Copy`, and then for (ii) `Mult`.

(i) For `Add` and `Copy`, all edges are added independently with probability $1 - (1 - p)^3 \leq 3p$ since there are three variables mapped to each edge: For `Add` there are 3 variables on the label $T_i = \{a_i, b_i, c_i\}$ (Section 3.1); For `Copy` there are 3 variables on the label $T_i = \{a_i, b_i, c_i\}$ (Section 3.1) with $a_i = b_i = c_i$. In both cases, each variable is carried by a single edge. Thus, the probability that all edges belong to the leakage diagram is $1 - (1 - 3p)^n \leq (3p)^n$ which concludes the proof.

(ii) For `Mult`, as in [16], we can prove that if we add each edge independently with probability $p' = 2(1 - (1 - \sqrt{3p})^{8n})$, the leakage diagrams obtained in this way contain the leakage diagrams obtained in the $\mathsf{Leak}(\mathtt{Mult}, (x, y), p)$-experiment.

We prove this fact by doing a proof very similar to the one in [16] (which is inspired by the original proof in [22]). The two differences are the fact that our `Mult` is slightly different from theirs and, more substantially, a different analytical treatment of the bound. We start observing that there are at most $8n$ wires which carry at least one variable present in the label T_i of the T_i-edge. a_i is used in n multiplication. Thus, there are n wires carrying it. Similarly, b_i is carried by n wires. There are $2n$ different $z_{l,i,j}$ for $l > 1$. There n different $a_i b_j$ and $a_j b_i$ (for $j \in [n]$). Finally, there are at most n between $z_{0,i,j}$ and $w_{i,j}$ wires and at n different $w_{j,i}$ wires.

Thus, we add the edge T_i with probability at most $1-(1-p)^{8n}$ and the edges T_i and T_j with probability at most $3p$.

It is easier to work with independent variables, but in the previous situation, the edges T_is are not independently added to LD. Thus, we try to add all the T_i edges to LD independently. To do this, we add T_i with the probability

$$1-(1-p)^{8n}+1-(1-\sqrt{3p})^{n-1}.$$

The proof is the same as in [16], where we have not used the approximation $1-(1-p)^{8n} \leq 8np$. [6]

This proves the claim of the theorem. □

Note that we do not use the approximation $1-(1-\sqrt{3p})^{n-1} \leq (n-1)\sqrt{3p}$ because it is not tight and makes the security much worse.

Security for `pRef`. With the same technique, we can analyze `pRef`. The only difference is the more complex dependency graph since it forms the skeleton of a cylinder and not a simple loop. Formally:

Proposition 6. *An outcome L of the* $\mathsf{Leak}(\mathtt{pRef}, x, p)$ *experiment, is shiftable to* x'*, if the corresponding leakage diagram, does not orbit.*

Proof. Again due to Proposition 1, we only prove the claim for maximal diagrams. We have two types of maximal diagrams $\mathcal{M}^{\text{right}}_{i,j}$ and $\mathcal{M}^{\text{left}}_{i,j}$. For the proof, we show how to shift/modify the variables in (i) $RGap_{i,j} = \mathsf{DG} \setminus \mathcal{M}^{\text{right}}_{i,j}$ and (ii) $LGap_{i,j} = \mathsf{DG} \setminus \mathcal{M}^{\text{left}}_{i,j}$. It consists of x_i, y_j, and some b_ls and r_ls.

(i) $\mathcal{M}^{\text{right}}_{i,j}$: we shift an output of $\mathsf{Leak}(\mathtt{pRef}, x, p)$ to an output of $\mathsf{Leak}(\mathtt{pRef}, x', p)$ as follows: x_i is modified in $x'_i = x_i + x' - x$, y_j is modified in $y'_j = x_j + x' - x$, r_l is modified in $r'_l = r_l - (x' - x)$, for the r_ls in $RGap_{i,j}$, and b_l is modified in $b'_l = b_l - (x' - x)$, for the b_ls in $RGap_{i,j}$. Note that the modified shares and intermediate values are all uniformly random, and the modification is done in such a way that they are still uniformly random and the intermediate dependencies are consistent.

(ii) With the same technique, we can modify $\mathcal{M}^{\text{left}}_{i,j}$: we shift an output of $\mathsf{Leak}(\mathtt{pRef}, x, p)$ to an output of $\mathsf{Leak}(\mathtt{pRef}, x', p)$ as follows: x_i is modified in $x'_i = x_i + x' - x$, y_j is modified in $y'_j = x_j + x' - x$, r_l is modified in $r'_l = r_l + (x' - x)$, for the r_ls in $LGap_{i,j}$, and b_l is modified in $b'_l = b_l + (x' - x)$, for the b_ls in $LGap_{i,j}$.

We have to prove that every shift results in an outcome of the $\mathsf{Leak}(\mathtt{pRef}, x', p)$ experiment. We prove one case and refer to the full version for the other cases.

Let $\delta = x' - x$. Let A be the values carried by the variables of `pRef` during the $\mathsf{Leak}(\mathtt{pRef}, x, \mathbf{r}, p)$-experiment, where $\mathbf{r} = r_0, ..., r_{n-1}$ is the randomness used.

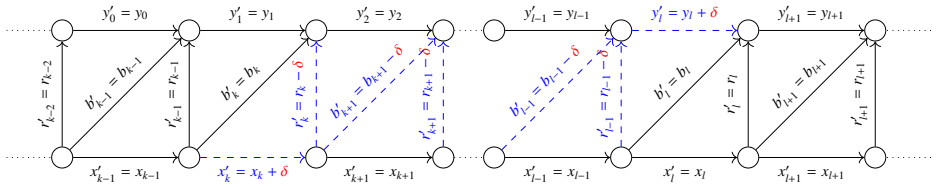
The maximal is of type $\mathcal{M}^{right}_{i,i}$. The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1},$$
$$r_0 - \delta, ..., r_{n-1} - \delta, b_0 - \delta, ..., b_{i-1} - \delta, b_i, b_{i+1} - \delta, ..., b_{n-1} - \delta\}.$$

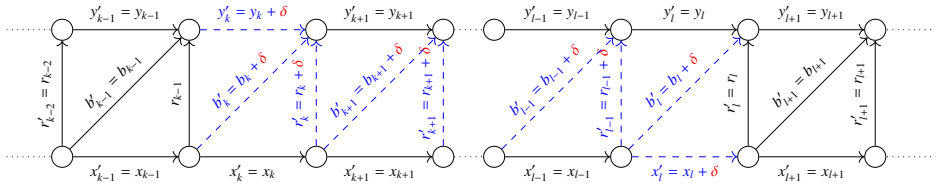
First, we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

[6] We have omitted here the approximation $1-(1-p)^{8n}+1-(1-\sqrt{3p})^{n-1} \leq [2(1-(1-\sqrt{3p})^{n-1})]^n$, which was used in the original version of the paper.



(a) Values of the variables in $RGap_{k,l}$ in the $\mathsf{Leak}(\mathtt{pRef}, x', p)$ experiment. $\delta = x' - x$.



(b) Values of the variables in $LGap_{k,l}$ in the $\mathsf{Leak}(\mathtt{pRef}, x', p)$ experiment. $\delta = x' - x$.

Figure 10: Example of the shifting depicted in Proposition 5. The solid edges are in $\mathcal{LD}$. The maximal is of type $\mathcal{M}^{\text{right}}_{k,l}$ and $RGap_{k,l} = \mathsf{DG} \setminus \mathcal{M}^{\text{right}}_{k,l}$ (10a), and $\mathcal{M}^{\text{left}}_{k,l}$ and $LGap_{k,l} = \mathsf{DG} \setminus \mathcal{M}^{\text{left}}_{k,l}$ (10b).

The same holds for the shares of y.
Now, let us consider pRef with input $x_0, ..., x_{i-1}, x_i+\delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0-\delta, ..., r_{n-1}-\delta$. For $i' \neq i$, $b_{i'} - \delta = x_{i'} + (r_{i'} - \delta)$ and $y_{i'} = (b_{i'} - \delta) - (r_{i'-1} - \delta)$. In stead, for i, $b_i = (x_i + \delta) + (r_i - \delta)$, and $y_i + \delta = (b_i) - (r_{i-1} - \delta)$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 - \delta, ..., r_{n-1} - \delta$.

□

As already discussed before (Section 3.1), the security of the Const and Rand gadgets is given by the security of the gadgets where their output wires are used. This gives us the following privacy result for pRef.

Theorem 2. *Let* pRef *be the refreshing gadget defined in Figure 1b. Then,* pRef *is* $(p, \tilde{p})$*-private, with* $\tilde{p} = 2 \cdot [(1 + 2(3p))3p]^n$*. If* $p \leq 1/6$ *we can approximate* $\tilde{p} \leq 2 \cdot (6p)^n$*.*

Proof. As in the security proof of the other gadgets, with Corollary 3 and Proposition 6, we only need to compute Pr[LD orbits] because it holds

$$\mathtt{pRef} \text{ is } (p, \Pr[LD \text{ orbits}])\text{-private.}$$

First, we observe that every edge of $\mathsf{DG}(\mathtt{pRef})$ (Figure 5b) has a label containing a single variable. We refer to Figure 5b for the notations. All variables are carried by a single wire except the r_i's, which are carried by three wires (Figure 1b). Thus, we can assume that each edge is added to LD independently with probability at most $1 - (1 - p)^3 \leq 3p$. With the result of Proposition 6, we know that the circuit is private if there is no loop that orbit. Moreover, any orbit must contain at least one of the source nodes of the edges x_0 or y_0. Next, we approximate the probability that in LD there is an orbit containing the source node of x_0 called $node^0_0$. (event E_0). Since such an orbit is at least n edges-long, we can upper-bound Pr[E_0] with the probability that the leakage diagram contains a path starting from $node_0$ with n edges. Since each node is connected with at most 4 other nodes, we have 3^n different n long paths[7] and the probability for each path is $(3p)^n$. This results in

$$\Pr[E_0] \leq (9p)^n.$$

But, we can refine the bound of E_0 as follows. We observe that we are trying to bound the probability of an orbiting path starting from $node^0_0$. Before, we observed that every path between

[7]We omit paths where an edge is crossed more than once. In fact, this possibility does not give anything to the adversary.

these two nodes is at least n edges long. However, it is easy to see that there is only one n-edges long path that describes a path from $node_0^0$ and arriving to it, $2(n-1)$ paths that are $n+1$-edges long, and so on. We observe that given a node y, the paths from y to $node_0^0$ which have the minimal length[8] keeping the direction of the orbit [9], takes as the first edge from y at most two different edges.

To prove this, it is enough to observe that if the node y is on the x_i^0-edges orbits, thus, it is node $node_j^0$ for a $j \in [n]$, the source node of the x_j^0-edge. Thus, if the path to $node_0^0$ keeping the direction of the orbit, has to turn left, the shortest path consists of the path $x_{j-1}^0, ..., x_0^0$; otherwise, if it turns right, the shortest path is taking the path $x_j^0, ..., x_{n-1}^0$. They are the shortest path because every time a path leaves the x_i^0 orbits, it needs an additional edge to rejoin it. Moreover, it is not possible to have paths that skip the ith edge (that is, do not contain any edge indexed with i) for $i = 0, ..., j-1$ if it turns left, or with $i = j, ..., n-1$ if it turns right.

Instead, if y is on the x_i^1-edges orbits, thus, it is node $node_j^1$ for a $j \in [n]$, the source node of the x_j^1-edge. Thus, if the path to $node_0^0$ keeping the direction of the orbit has to turn right, it must take either the edge x_j^1 or r_{j-1}. In fact, the first path needs $1+n-j$ edges (see above), while the shortest paths using the-x_j^1 can take as well at most $1+n-j$ edges. The proof that the shortest path needs $1+n-j$ edges can be easily done by induction. We iterate the previous argument until we arrive at $node_0^1$ for which the shortest path to $node_0^0$ is clearly r_{n-1}.

Finally, if the path to $node_0^0$ keeping the direction of the orbit has to turn left, it must take either the edge b_{j-1} or x_{j-1}^1 (for $j = 1$, it must take b_0). In fact, the first path needs $j-1$ edges (see above, the case for y as node $node_j^0$), while the shortest paths using the-x_{j-1}^1 can take as well at most $j-1$ edges. The proof that the shortest path needs $j-1$ edges can be easily done by induction. We iterate the previous argument until we arrive at $node_1^1$ for which the shortest path to $node_0^0$ is clearly b_0. For $j = 0$, since we have to turn left (the case where you have to go down via r_{n-1} has already been treated in turning right, we must take x_{n-1}^1 or b_{n-1}.

Finally, we have to explain why

$$\Pr[E_0] \leq [(1+2(3p))3p]^n.$$

This happens because if we consider a loop starting from $node_0^0$, which orbits, and we do not take the shortest path, which is $x_0^0, ..., x_{n-1}^0$, and we take the e-th edge to deviate. In this case, we need to take one of the other two edges connected to the node we have arrived after we have taken the e-edge. But, this happens with probability at most $3p$. From there, we need at most $n-i$ edges to end our loop, where is the number of edges taken on the $x_0^0, ..., x_{n-1}^0$ before deviating via the e-edge.

Thus, if a path from y to $node_0^0$ does not pass through one of these two edges, it is at least $l+1$ edges long. Thus, the probability that it belongs to LD is at most $(3p)^{l+1}$. Thus, we can upper-bound

$$\Pr[E_0] \leq [(1+2(3p))3p]^n.$$

Further, if $6p < 1$ it holds

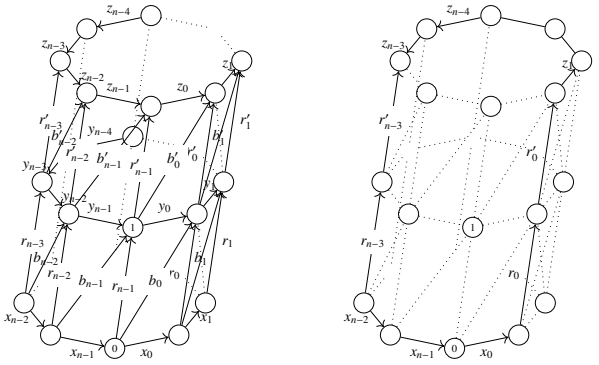
$$[(1+2(3p))3p]^n \leq (6p)^n.$$

Clearly, the same holds for E_1, which is the event that there is an orbiting loop starting from the source node of the edge y_0. Thus, $\Pr[LD \text{ orbits}] \leq \Pr[E_0] + \Pr[E_1] = 2[(1+2(3p))3p]^n$ for $p \leq 1/3$ and $2(6p)^n$ for $p \leq 1/6$. □

With Theorem 1 and 2, we have proven the security of the gadgets used by our compiler. However, it is well known that it is insufficient to prove their compositions' security. Next, we examine the security of compositions to argue about the security of the compiler used in the paper.

[8] minimal means with the least number of edges.

[9] That is, making the loop $node_0^0, ..., y, node_0^0$ orbit.



(a) The dependency graph for $(z_i)_{i\in[n]} \leftarrow \texttt{pRef}(\texttt{pRef}((x_i)_{i\in[n]}))$.

(b) The dependency graph for $(z_i)_{i\in[n]} \leftarrow \texttt{pRef}(\texttt{pRef}((x_i)_{i\in[n]}))$.

Figure 11: Dependency graphs of two composed refresh gadgets.

5 Security Analyzes for Circuits

In Section 4, we have investigated the privacy of the gadgets used by our compiler. In this section, we analyze the security of their compositions. Hence, we compute the privacy of our compiler's output. We start with the composition of pRef gadgets (Sec. 5.1), and then, we give a more general composition result for all gadgets used by our compiler (Sec.5.2)

5.1 Security Analysis for the Composition of pRef

In this section, we consider k sequential compositions of pRef gadgets (kpRef). Let $(x_i^0)_{i\in[n]}$ be the input of the first refresh gadget, then the j^{th} pRef, denoted with $\texttt{pRef}^j$ computes $(x_i^j)_{i\in[n]} \leftarrow \texttt{pRef}^j((x_i^{j-1})_{i\in[n]})$. Further, the internal variables used by $\texttt{pRef}^j$ are defined as r_i^j and b_i^j. Thus, the final output is $(x_i^k)_{i\in[n]} \leftarrow k\texttt{pRef}((x_i^0)_{i\in[n]})$. In Figure 11a, we depicted the dependency graph of two composed refresh gadgets. Compared to the more general compositions with multiple input and output sharings, as depicted in Figure 6, the dependency graph of the composed refresh gadgets is relatively simple. More precisely, it is still the skeleton of a cylinder. This cylinder is given by putting the cylinder of the dependency graph of $\texttt{pRef}^j$ (denoted with DG^j) over the cylinder of $\texttt{pRef}^{j-1}$ (denoted with DG^{j-1}). In other words, the only difference with the security proof of the single refresh gadget is the length of the cylinder. The dependency graph of the composition of k refresh gadgets is a cylinder constructed out of the k cylinders (DG^j) of the dependency graph of pRef. As mentioned in the previous section, it is not sufficient to analyze the security of each gadget. This is easy to see when we remember that we have shown in Section 4.2 that pRef is secure if the leakage diagram does not orbit the cylinder. Here, it might be the case that the leakage diagram of its composition orbits the cylinder, but if we consider each leakage diagram of the composed refresh gadgets separately, it does not orbit each subcylinder. For example, Figure 11b illustrates such a case. Next, we use the same technique as in Section 4.2 to prove the security of the composition.

First, we classify the maximal diagrams of kpRef. One way to describe the gap in the cylinder is to use the multiple subgraphs *RGap* and *LGap* as defined in the proof of pRef and compose them in such a way that they describe such a gap from the bottom to the top of the cylinder. However, such a gap can also have some detours such that the gap goes partly in the opposite direction. To cover all such cases, we have to consider four more "gap sets" than *RGap* and *LGap*:

(i) $BLGap_{i,i'} := \{x_i^{j-1}, x_{i'}^{j-1}, r_{i'}^j, ..., r_{i-1}^j, b_{i'+1}^j, ..., b_i^j\}$ describes a gap that starts from the bottom circle of DG^j, then turns left, and, finally, ends in the bottom circle.

(ii) $BRGap_{i,i'} := \{x_i^{j-1}, x_{i'}^{j-1}, r_i^j, ..., r_{i'-1}^j, b_{i+1}^j, ..., b_{i'}^j\}$ describes a gap that starts from the bottom circle of DG^j, then turns right, and, finally, it ends in the bottom circle.

(iii) $ULGap_{i,i'} := \{x_i^j, x_{i'}^j, r_{i'}^j, ..., r_{i-1}^j, b_{i'}^j, ..., b_{i-1}^j\}$ describes a gap that starts from the top circle of DG^j, then turns left, and, finally, it ends in the top circle.

(iv) $URGap_{i,i'} := \{x_i^j, x_{i'}^j, r_i^j, ..., r_{i'-1}^j, b_i^j, ..., b_{i'-1}^j\}$ describes a gap that starts from the top circle of DG^j, then turns right, and, finally, it ends in the top circle.

We can use these gaps to classify maximal diagrams for kpRef. The next claim describes the structure of such maximal graphs if we consider each sub-graph of each pRef separately.

Proposition 7. *Let $(x_i^0)_{i\in[n]}$ be the input of k*pRef *and $(x_i^j)_{i\in[n]} \leftarrow$ pRef$((x_i^{j-1})_{i\in[n]})$ the jth* pRef *of k*pRef*. Further, let DG^j be the sub-graph of $\mathsf{DG}_{\texttt{pRef}}$ containing the variables used in* pRef^j *and $\mathcal{MAX}$ a maximal diagram of k*pRef*. It holds*

$$\mathcal{MAX} \cap \mathsf{DG}^j = \mathsf{DG}^j \setminus \left[\left(\bigcup_{I\in\mathcal{I}_1} RGap_{i_I,i'_I}\right) \cup \left(\bigcup_{I\in\mathcal{I}_2} LGap_{i_I,i'_I}\right) \cup \left(\bigcup_{I\in\mathcal{I}_3} BLGap_{i_I,i'_I}\right) \cup \left(\bigcup_{I\in\mathcal{I}_4} BRGap_{i_I,i'_I}\right) \cup \left(\bigcup_{I\in\mathcal{I}_5} ULGap_{i_I,i'_I}\right) \cup \left(\bigcup_{I\in\mathcal{I}_6} URGap_{i_I,i'_I}\right)\right]$$

For all $LD^j := \mathcal{MAX} \cap \mathsf{DG}^j$, we get $|\mathcal{I}_1| + |\mathcal{I}_2|$ is odd and all gaps are pairwise disjoint. Moreover, for LD^1 and LD^k it holds $|\mathcal{I}_1| + |\mathcal{I}_2| = 1$, for LD^1 we get $|\mathcal{I}_3| = |\mathcal{I}_4| = 0$, while for LD^k we get $|\mathcal{I}_5| = |\mathcal{I}_6| = 0$. Each gap RGap and LGap of DG^j is connected to one and only one gap of DG^{j+1}, and to one and only one gap of DG^{j-1}. Each gap BLGap and BRGap is connected to two distinct gaps of DG^{j-1}, and each gap ULGap and URGap is connected to two distinct gaps of DG^{j+1}.

Proof sketch. Even when the claim looks relatively complex, it only formalizes how the maximal graph $\mathcal{MAX}$ of kpRef looks when we cut it into the sub-graphs LD^j describing the individual $(x_i^j)_{i\in[n]} \leftarrow$ pRef$((x_i^{j-1})_{i\in[n]})$. The proof idea is that the composition of all LD^j together still describes a gap that avoids a path around the dependency graph of kpRef. In other words, we can see it as construction where we can construct and compose such LD^j so that there is still a gap, and each further added edge would close the gap. Next, we give the formalized proof for this claim.

Proof. First, we prove that all the gaps are connected as described, that is, each gap *RGap* and *LGap* of DG^j is connected to at least one gap of DG^{j+1}, and at least one gap of DG^{j-1}; each gap *BLGap* and *BRGap* is connected to two distinct gaps of DG^{j-1}; and each gap *ULGap* and *URGap* is connected to two distinct gaps of DG^{j+1}. If this is not the case, we simply consider the perimeter of the gap, which is not connected. It is easy to see that this perimeter is homotopically equivalent to the gap on the $(x_i^j)_{i\in[n]}$-orbit or $(x_i^{j-1})_{i\in[n]}$-orbit where it starts. Thus, we can add the edge missing on that orbit without making $\mathcal{MAX}$ orbiting. Thus, $\mathcal{MAX}$ is not maximal, which is absurd by hypothesis.

We proceed by induction over k. For $k = 1$, the proof has already been done in Proposition 7.

Using the same argument as in the proof of Proposition 4, there is only one edge missing in the orbit defined by the $(x_i^0)_{i\in[n]}$-edges. Let j be this missing edge. Thus, there must be a gap either $RGap^{j,j'}$ or $LGap_{j,j'}$ in DG^1. Otherwise, taking the perimeter of the gap starting from x_j^0, we have a path in *LD* which is homotopically equivalent to x_j^0. Thus, *LD* orbits since it contains the path $x_{j+1}^0, ..., x_{j-1}^0$ and a path homotopically equivalent to x_j^0. All the remaining gaps are of type *ULGap* and *URGap* since they cannot go out. We have only to prove that all these gaps are disconnected. If one of these gaps is connected to the *LGap* gap or the *RGap* gap, then we have a problem. In fact, it means that from x_j^0, we can go out of the DG^1 either from x_i^1 or $x_{i'}^1$. But going out from these two gaps, we must arrive at the $(x_i^k)_{i\in[n]}$-orbit. (Otherwise, we can use the same argument to prove that the gaps are connected, to prove that $\mathcal{MAX}$ is not maximal). Thus, we can have two possibilities: 1) these two gaps do not reconnect. Thus there are two missing edges in $(x_i^k)_{i\in[n]}$,

which is absurd due to the argument that we have explained in the proof of Proposition 4. 2)these two gaps reconnect. Thus $\mathcal{MAX}$ is disconnected, which is absurd due to an argument presented in the proof of Proposition 4. Similarly, we can prove that two gaps that start and ends on the top circle of DG^1, that is, the $(x_i^1)_{i\in[n]}$-one cannot intersect between each other.

Again we do the same analysis for $\mathsf{DG}^k \cap \mathcal{MAX}$. Using the same argument as for DG^1, we can prove that there must be a gap of type *RGap* or *LGap* and some gaps of type *BLGap* and *BRGap*.

Now we consider $\mathcal{MAX} \cap \mathsf{DG}^{2->k-1}$, where with $\mathsf{DG}^{2->k-1}$ we denote DG of the composition of $\mathtt{pRef}^2$ until $\mathtt{pRef}^{k-1}$. There are I holes in $(x_i^1)_{i\in[n]}$, that is, I missing edges, I' missing edges in $(x_i^{k-1})_{i\in[n]}$.

Each of these holes must be connected via a gap in $\mathsf{DG}^{2->k-1}$ to one and only one other hole. Otherwise, we have the argument against splitting or terminating at a dead end.

Now we can use the induction. Since each of these gaps is contained there, we can use the induction hypothesis to prove that each of these gaps has the desired shape.

Finally, we have to prove that $|\mathcal{I}_1| + |\mathcal{I}_2|$. We observe that we have proved that there is a "snake" of gaps that starts from x_j^0 the missing edge in the $(x_i^0)_{i\in[n]}$-orbit), to $x_{j'}^k$ (the missing edge in the $(x_i^k)_{i\in[n]}$-orbit). Each time we cross from the top to the bottom, the $(x_i^l)_{i\in[n]}$-orbit, we must cross from the bottom to the top in the same orbit. Moreover, one additional time we must cross from the top to the bottom. This concludes the proof. □

In other words, a maximal diagram can be described as a dependency graph with a gap from the bottom to the top consisting of the six subgraphs defined above. We can use this classification to prove the security for $k\mathtt{pRef}$.

Proposition 8. *An outcome L of the experiment* $\mathsf{Leak}(k\mathtt{pRef}, x, p)$ *is shiftable if its leakage diagram does not orbit the dependency graph.*

Proof sketch. The high-level idea is similar to Proposition 6. We have shown that a single refresh is shiftable if the leakage diagram does not orbit the cylinder structure of its dependency graph. In detail, we have proven that a not orbiting graph of a single refresh is a sub-graph of $RGap_{i,j} = \mathsf{DG} \setminus \mathcal{M}_{i,j}^{\text{right}}$ or $LGap_{i,j} = \mathsf{DG} \setminus \mathcal{M}_{i,j}^{\text{left}}$. Further, this implies that the input share x_i can be set to an arbitrary value, and we can compute the according share y_j without changing the distribution of the leaked values. Hence, x_i and y_j are shiftable. It is easy to see that this also holds for composed gadgets with $(y_i)_{i\in[n]} \leftarrow \mathtt{pRef}^1((x_i)_{i\in[n]})$, and $(z_i)_{i\in[n]} \leftarrow \mathtt{pRef}^2((y_i)_{i\in[n]})$. If x_i and y_j are shiftable in $\mathtt{pRef}^1$, and y_j and z_k are shiftable in $\mathtt{pRef}^2$, it follows that x_i and z_k are shiftable in $\mathtt{pRef}^2(\mathtt{pRef}^1(\cdot))$. In detail, we can set x_i to an arbitrary value and compute accordingly y_j. Since y_j and z_k are shiftable as well, we can also compute the according z_k for any y_j. For the formal proof, we also show the shiftability for the four additional types of gaps we have introduced above and extend the technique to an arbitrary number of compositions.

Proof. Similar to the security proofs of the gadgets, we use Proposition 1, and only prove the claim for maximal diagrams. We use the same approach as for the maximal diagram of a single refresh. Hence, using Proposition 1, we have to prove the claim for all maximal diagrams. Therefore, we want to show how to modify the values in the gaps $LGap_{i,i'}^j$,$RGap_{i,i'}^j$, $BLGap_{i,i'}^j$, $BRGap_{i,i'}^j$, $ULGap_{i,i'}^j$, and $URGap_{i,i'}^j$ shifting an output of $\mathsf{Leak}(\mathtt{pRef}, x, p)$ in another one without being detected. For $LGap_{i,i'}^j$ and $RGap_{i,i'}^j$ the proof was already done in Proposition 6. Hence it remains to prove it for the other four constructions:

(i) $BLGap_{i,i'}^j$: x_i is modified in $x_i' = x_i + \gamma$, $x_{i'}$ is modified in $x_{i'}' = x_{i'} - \gamma$, r_l is modified in $r_l' = r_l + \gamma$, for the r_ls in $BLGap_{i,j}$, and b_l is modified in $b_l' = b_l + \gamma$, for the b_ls in $BLGap_{i,j}$. for any γ. Note that $x_i' + x_{i'}' = x_i + x_{i'}$.

(ii) $BRGap_{i,i'}$: x_i is modified in $x_i' = x_i + \gamma$, $x_{i'}$ is modified in $x_{i'}' = x_{i'} - \gamma$, r_l is modified in $r_l' = r_l - \gamma$, for the r_ls in $BRGap_{i,j}$, and b_l is modified in $b_l' = b_l - \gamma$, for the b_ls in $BRGap_{i,j}$. Here, as well, $x_i' + x_{i'}' = x_i + x_{i'}$.

(iii) $ULGap_{i,i'}$: y_i is modified in $y'_i = y_i + \gamma$, $y_{i'}$ is modified in $y'_{i'} = y_{i'} - \gamma$, r_l is modified in $r'_l = r_l - \gamma$, for the r_ls in $ULGap_{i,j}$, and b_l is modified in $b'_l = b_l - \gamma$, for the b_ls in $ULGap_{i,j}$. Similarly to before $y'_i + y'_{i'} = y_i + y_{i'}$.

(iv) $URGap_{i,i'}$: y_i is modified in $y'_i = y_i + \gamma$, $y_{i'}$ is modified in $y'_{i'} = y_{i'} - \gamma$, r_l is modified in $r'_l = r_l + \gamma$, for the r_ls in $URGap_{i,j}$, and b_l is modified in $b'_l = b_l + \gamma$, for the b_ls in $URGap_{i,j}$. Here, as well, $y'_i + y'_{i'} = y_i + y_{i'}$.

Hence, we have modified an outcome $\mathsf{Leak}(\mathtt{pRef}, x, p)$ to another $\mathsf{Leak}(\mathtt{pRef}, x, p)$. Here, we do the proof for one case and refer to the full version for the other cases.

The gap is of type $BLGap_{i,i'}$ with $i < i'$ [10]. The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}, y_0, ..., y_{n-1}, r_0 + \gamma, ..., r_{i-1} + \gamma,$$
$$r_i, ..., r_{i'-1}, r_{i'} + \gamma, ..., r_{n-1} + \gamma, b_0 + \gamma, ..., b_i + \gamma, b_{i+1}, ..., b_{i'}, b_{i'+1} + \gamma, ..., b_{n-1} + \gamma\}.$$

First, we observe that

$$x_0 + ... + x_{i-1} + (x_i + \gamma) + x_{i+1} + ... + x_{i'-1} + (x_{i'} - \gamma), x_{i'+1}, ..., x_{n-1} = x_0 + ... + x_{n-1} + \gamma - \gamma = x.$$

Since we have not touched the shares of y, they carry an encoding of y which is equal to x.
Now, let us consider $\mathtt{pRef}$ with input $x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}$ and randomness $r_0 + \gamma, ..., r_{i-1} + \gamma, r_i, ..., r_{i'-1}, r_{i'} + \gamma, ..., r_{n-1} + \gamma$. For $0 \leq j < i$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. For i, $b_i + \gamma = (x_i + \gamma) + r_i$, and $y_i = (b_i + \gamma) - (r_{i-1} + \gamma)$. For $i < j < i'$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i', $b_{i'} = (x_{i'} - \gamma) + (r_{i'} + \gamma)$, and $y_{i'} = b_{i'} - r_{i'}$. For $i' < j \leq n - 1$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. Thus, the values carried by A' are those of an honest evaluation of $\mathtt{pRef}$ with input $x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}$ and randomness $r_0 + \gamma, ..., r_{i-1} + \gamma, r_i, ..., r_{i'-1}, r_{i'} + \gamma, ..., r_{n-1} + \gamma$. For all the other gaps, we proceed in a similar way. For more details we refer to the full version. It remains to explain how we can use the modifications just described to do the shift in the maximal diagram of $k\mathtt{pRef}$. Therefore, we start with DG^1. Using the classification of the maximal of $k\mathtt{pRef}$ there is a single gap starting from its bottom circle. We modify its values according to the proof of Proposition 6. Then, there is another gap in DG^2 that starts from the edge just modified before. We can modify it coherently starting from this modified value. Iterating, due to our classification, there is always a gap in another DG^j which starts from the final edge of the gap just modified before. Finally, we arrive at the single gap of DG^k which ends in the top circle.

Formally, we have proved the results for all the additional gaps introduced in Proposition 7. Every gap is connected only to two other gaps (except for the first and the last.) Now, we start observing that there are

$$\sum_{j=0}^{k} |\mathcal{I}_3^j| + \sum_{j=0}^{k} |\mathcal{I}_4^j| = \sum_{j=0}^{k} |\mathcal{I}_5^j| + \sum_{j=0}^{k} |\mathcal{I}_6^j|,$$

where with $\mathcal{I}_6^l$, we denote the set $\mathcal{I}_6^j$ in the classification of $\mathcal{MAX} \cap \mathsf{DG}^j$ (Proposition 7).

Since every time when we follow a $RGap_{i,i'}$ or a $LGap_{i,i'}$ gap, we keep the same modifications for x_i and $y_{i'}$, while when we follow $BLGap$, or $BRGap$, $ULGap$, or $URGap$, we modify x_i and $y_{i'}$ with two opposite values, and we follow an even number of $BLGap$, $BRGap$, $ULGap$, and $URGap$, we have that x_j^k is modified with $+\delta = x' - x$, where x_j^k is the single edge missing in the $(x_i^k)_{i\in[n]}$-orbit. Thus, we have modified the values of the variable carried by $\mathsf{DG} \setminus \mathcal{MAX}$ to modify an outcome of $\mathsf{Leak}(k\mathtt{pRef}, x, p)$ in one of $\mathsf{Leak}(k\mathtt{pRef}, x', p)$. We give the formal description in the full version. □

An example of a maximal for $k\mathtt{pRef}$ is depicted in the full version. The proposition gives the following privacy result for our compositions. It only remains to compute the probability that a leakage diagram does not orbit the cylinder described by the dependency graph of $k\mathtt{pRef}$.

[10] For simplicity, when we use $i < i'$, we assume that $i, i' \in [n] \subset \mathbb{Z}$ and not in $\mathbb{Z}_n$.

Theorem 3. *Let k*pRef *be the composition of k refreshing gadget, defined in Figure 1b. Let n be the number of shares used. Then, k*pRef *is* $(p, (k+1)\cdot[(2+3(3p))3p]^n))$*-private, for* $p \leq 1/3$ *and* $(p, 2\cdot(9p)^n))$.

Proof. Let *k*pRef takes as input $(x_0^0, ..., x_{n-1}^0)$ and outputs $(x_0^k, ..., x_{n-1}^k)$, with the *i*th pRef gadget, denoted with $\texttt{pRef}^i$ does $(x_0^i, ..., x_{n-1}^i) \leftarrow \texttt{pRef}(x_0^{i-1}, ..., x_{n-1}^{i-1})$. The proof is completely similar to the one of Theorem 2 with the following differences: 1) We consider the events E_i, for $i = 0, ..., k$ (instead of $i = 0, 1$), which consists in the event that there is an orbit containing the source node of the edge x_0^i (denoted with $node_0^i$). 2) Each node is connected with 6 other nodes (and not 4). Thus, there are at most 5^n different paths. 3) There exists a single n-edges long loop orbiting from $node_0^i$, $4(n-1)\,n+1$-edges long, and so on. From 1) and 2), we obtain that we can bound $\Pr[E_i] \leq (15p)^n$. Moreover, we can use the same argument to obtain, for $p \leq 1/9$ that

$$\Pr[E_i] \leq [(2+3(3p))3p]^n \leq (9p)^n.$$

To prove this, we can reuse the same arguments as in the proof of Theorem 2. We must add also the argument that no edge connects the $(x_i^j)_{i\in[n]}$-orbit and the $(x_i^{j'})_{i\in[n]}$-orbit if $j' \neq j-1, j, j+1$. Moreover, if node y belongs to the $(x_i^l)_{i\in[n]}$-orbit, the shortest path to $node_0^j$ can never cross and edge to go from the $(x_i^L)_{i\in[n]}$-orbit to a node in the $(x_i^{L'})_{i\in[n]}$-orbit if $|L'-j| > |L-j|$ (because we cannot skip orbits).

Finally, we explain why it holds

$$\Pr[E_0] \leq [(2+3(3p))3p]^n.$$

This is due to the fact that not all edges directly give the shortest path and require further edges that are also added with probability p, and this happens with probability at most $3p$. □

5.2 Security Analysis for the Composition of Gadgets

Now, we analyze the security of an arbitrary output of our compiler $\widehat{\mathsf{C}} \leftarrow \texttt{CC}(\mathsf{C})$, where C can be any circuit described in the background. In Figure 6b, we depicted the dependency graph of Add (or Mult) gadgets composed with pRef gadgets to refresh its inputs and outputs. Since our compiler puts refresh gadgets after every output of Add, Mult, and Copy gadget, the dependency graph of our compiler's output is always a composition of dependency graphs depicted in Figure 6a. As mentioned, this leads to slightly more complex dependency graphs than the one described in the previous section (Sec. 5.1.) However, the main idea is the same. We have multiple dependency graphs, as depicted in Figure 6a, sharing the same upper or lower circle as already described in the previous section (Figure 11a.) The only difference is that gadgets can also have two input (or output) sharings. Therefore the cylinder does not only share the bottom (or top) of the cylinder with one but two further cylinders, as depicted in Figure 6b. Hence, the dependency graph of $\widehat{\mathsf{C}}$ is still a composition of multiple cylinders. Since the dependency graph describes multiple cylinders connected by shared upper and lower circles, we can distinguish orbiting loops from not orbiting loops again. For this reason, we can use the same technique as in Proposition 8 to prove the security of the composition. Again, we start classifying the maximals of $\widehat{\mathsf{C}}$.

Proposition 9. *Let* $\mathcal{MAX}$ *be a maximal diagram*

(i) For each circuit-input encoding and circuit-output encoding, there is only one edge of the encoding that does not belong to $\mathcal{MAX}$.

(ii) The intersection of the DG *of all* pRef *with* $\mathcal{MAX}$ *has an odd number of components which have the characterization described in Proposition 8.*

(iii) Each of these "gap sets" is connected to others to form gaps from each circuit input encoding to any output encoding of the circuit.

Proof. i) We exploit the same argument as presented in the proof of Proposition 7 to prove that this holds. Substantially, if there is more than one hole in the input/output orbit edge, the graph is either not maximal or it orbits. If there are more than two holes on the input $(x_i)_{i\in[n]}$-orbit, we must have a gap starting from each. These two gaps either intersect or do not intersect. If they intersect, then we can show that the maximal diagram orbit with an argument similar to the one introduced in the proof of Proposition 7. In the second case, $\mathcal{MAX}$ is not connected. If they do not connect, we cannot assume that they go to two different other input/output orbits. Because otherwise, we can homotopically reduce these two input sharings to the copy where the gaps split. Thus we have non-connection.

ii) The idea is the same as in the proof of Proposition 7. It comes from the fact that the gap cannot split or have a dead end in the $\mathsf{DG}_{\mathtt{pRef}}$ with the same idea as in Proposition 7.

iii) This happens because if these gaps are not connected to each other, we can homotopically take them away as done in the proof of Proposition 7.

□

As before, our classification allows us to prove the security of not orbiting leakage diagrams.

Proposition 10. *Let $\widehat{\mathsf{C}}$ be a masked circuit obtained from our compiler. An outcome L of the $\mathsf{Leak}(\widehat{\mathsf{C}}, \boldsymbol{x}, p)$ experiment, is shiftable to $\boldsymbol{x}'$ if its leakage diagram does not orbit the dependency graph.*

Proof sketch. The only difference to the proof in Section 5.1 is that we have gadgets with multiple input and output sharing. Hence we do not have only one bottom and top. (E.g. Figure 6b has two bottom circles). However, the high-level idea is the same.

Proof. When we have such dependency graphs, a leakage diagram does not orbit when gaps exist (as defined in the proof of Proposition 8) from every bottom and top circle. We have described the maximals above. Using this classification, we can modify the values of the variables carried by the edges in $\mathsf{DG}_{\widehat{\mathsf{C}}} \setminus \mathcal{MAX}$ as in the proof of Proposition 8. We start with an arbitrary input encoding, then we modify the values on the subgraphs *RGap*, *LGap*, *BRGap*, *BLGap*, *ULGap*, and *URGap* as in the previous example (and as described in the Proposition 5, 6, and 8). Iterating, we arrive at modifying all the values of $\mathsf{DG} \setminus \mathcal{MAX}$. With this technique, we can modify all inputs. Hence, L from $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ is shiftable if its leakage diagram is not orbiting. The details that this modification gives the same output of $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$ is given now.[11]

We start with the input orbit sharings. From there, we can change the gaps until we arrive to a gap that arrives in a gap where two different inputs are modified. There using Proposition 5, we wait until we have arrived to modify the other input with gaps. We can go on, and we can arrive that at least we can modify one of the gaps with multiple inputs. [This happens due to the structure of the gasp in $\mathcal{MAX}$.]

This operation is correct because even here, every time that we go up, we add the shift, while every time we go down, we subtract the shift. This works as in the proof of Proposition 8.

□

Using Corollary 3 and Proposition 10, we can bound the actual security of the circuits obtained via our compiler.

Theorem 4. *Let $\widehat{\mathsf{C}}$ be a circuit obtained via our compiler, $\widehat{\mathsf{C}} \leftarrow \mathsf{CC}^p(\mathsf{C})$, and $|\mathsf{C}|$ be the number of gates of the circuit C, I the number of input gates and O the number of output gates. Then, $\widehat{\mathsf{C}}$ is $(p, (|\mathsf{C}| + I + O)\widehat{p}^n)$-private, with*

$$\widehat{p} = 18p + 2(1 - (1-p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1})$$

[11] In the proof of Proposition 8 this step is easier because each gap set meets at its end a single another gap set.

If C *is affine, then,* $\widehat{\mathsf{C}}$ *is* $(p, (|\mathsf{C}| + I + O)(12p)^n)$*. If circuit* C *is complete, then* $\widehat{\mathsf{C}}$ *is* $(p, (|\mathsf{C}|\widehat{p}^n)$*-private, or* $\widehat{\mathsf{C}}$ *is* $(p, |\mathsf{C}|(12p)^n)$*-private.*

Proof. The proof is similar to the proof of Theorem 3 with the following differences: 1) We have at most $|\mathsf{C}| + I + O$ different sharings (orbits) containing the shares of an input, intermediate, or output encoding. If the circuit is complete, the number of such orbits is at most $|\mathsf{C}|$. Thus we have $|\mathsf{C}| + I + O$ events E_i to consider. 2) For each node of DG, there are at most 8 edges. 3) Each edge is added to LD with probability at most $p' = 1 - (1-p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1}$ for general circuits (or $p' = 3p$ for affine circuits),see Proposition 5 and 6. Thus, we can prove that $\Pr[E_i] \leq (7p')^n$. Doing a more detailed analysis (full version), similar to the one done in Theorem 2 and 3, we obtain that $\Pr[E_i] \leq [6np + 2(1 - (1-p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1})]^n$ (App. A.4) Putting everything together, we obtain the claim. □

With Theorem 4, we can discuss the security results for our compiler.

5.3 Compiler Security

In the previous section, we have proven that any complete circuit $\widehat{\mathsf{C}}$ obtained via our compiler, $\widehat{\mathsf{C}} \leftarrow \mathsf{CC}^p(\mathsf{C})$ is $(p, (|\mathsf{C}|)\widehat{p}^n)$-private, with $\widehat{p} = [6np + 2(1 - (1-p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1})]$. Further, if C is affine, then, $\widehat{\mathsf{C}}$ is $(p, (|\mathsf{C}| + I + O)(12p)^n)$-private. This section discusses the results and demonstrates the improvements compared to the state-of-the-art. As in [22, 16] the condition $\widehat{p} < 1$ requires an upper-bound for the leakage probability p. In detail, Theorem 4 requires the following. Unfortunately, given $\alpha \in [0, 1]$, we are not able to explicitly compute the maximal p given n, s.t. $[6np + 2(1 - (1-p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1})]^n \leq \alpha$. Here, we try to study the asymptotic behaviour of p which makes the previous equation lower than α. Since, we have a term $(1 - (1 - \sqrt{3p})^{n-1})^n$, we wonder that we are secure if $p = O(n^{-2})$. This is the case, as we prove in the following theorem:

Theorem 5. *The compiler is secure for any leakage probability* p *with* $p = O(\frac{1}{n^2})$.

Proof. Let $p = Cn^{-2}$ for a constant C

$$\lim_{n\to\infty}\left[18\frac{C}{n^2} + 2\left(1 - \left(1 - \frac{C}{n^2}\right)^{8n} + 1 - \left(1 - \sqrt{3\frac{C}{n^2}}\right)^{n-1}\right)\right]^n$$
$$\lim_{n\to\infty}\left[\frac{18C}{n^2} + 2\left(1 - \left(1 - \frac{C}{n^2}\right)^{8n} + 1 - \left(1 - \frac{\sqrt{3C}}{n}\right)^{n-1}\right)\right]^n$$
$$\lim_{n\to\infty}\left[\frac{18C}{n^2} + 2\left(1 - e^{8n\log\left(1-\frac{C}{n^2}\right)} + 1 - e^{(n-1)\log\left(1-\frac{\sqrt{3C}}{n}\right)}\right)\right]^n$$

Using the following well known result from analysis

$$\log(1 + \epsilon_n) \sim \epsilon_n \text{ for } n \to \infty \text{ and } \epsilon_n \to 0$$

the previous limit becomes

$$\lim_{n\to\infty}\left[\frac{18C}{n^2} + 2\left(1 - e^{8n\frac{-C}{n^2}} + 1 - e^{(n-1)\frac{\sqrt{-3C}}{n}}\right)\right]^n$$
$$\lim_{n\to\infty}\left[\frac{18C}{n^2} + 2\left(1 - e^{\frac{-8C}{n}} + 1 - e^{\frac{-(n-1)\sqrt{3C}}{n}}\right)\right]^n$$
$$\lim_{n\to\infty}\left[\frac{18C}{n^2} + 2\left(1 - e^{\frac{-8C}{n}} + 1 - e^{-\sqrt{3C}}\right)\right]^n.$$

Using the well-known result from analysis

$$e^{\epsilon_n} \sim 1 + \epsilon_n \text{ for } n \to \infty \text{ and } \epsilon_n \to 0$$

we have that

$$\lim_{n\to\infty} \left[\frac{18C}{n^2} + 2\left(1 - 1 + \frac{8C}{n} + 1 - e^{\sqrt{-3C}}\right)\right]^n.$$

Now, since $\frac{8C}{n} \to 0$ and $\frac{18C}{n^2} \to 0$ when $n \to \infty$, we have that if $1/2 < e^{-\sqrt{3C}} < 1 + 1/2$, then

$$\frac{18C}{n^2} + 2\left(1 - e^{\frac{-8C}{n}} + 1 - e^{-\sqrt{3C}}\right) < 1.$$

Thus,

$$\lim_{n\to\infty} \left[\frac{6C}{n} + 2\left(1 - e^{\frac{-8C}{n}} + 1 - e^{-\sqrt{3C}}\right)\right]^n = 0.$$

But, the condition

$$1/2 < e^{-\sqrt{3C}} < 1 + 1/2 \Leftrightarrow \log(1/2) < -\sqrt{3C} < \log(1 + 1/2)$$
$$[\log(1 + 1/2)]^2 < 3C < \log(1/2)^2 \Leftrightarrow \frac{[\log(1 + 1/2)]^2}{3} < C < \frac{[\log(1 + 1/2)]^2}{3}.$$

□

We have experimentally proved this in Fig 12.

Interestingly, for reasonable n, we have that we behave significantly better than [16] even if $p = O(n)$, as we prove in Fig. 13. We explain the reason behind this in App. B.

6 Comparison with Dziemobwski et al. [16]

Our work is inspired by Dziembowski et al. [16]. Thus, we want to summarize the differences between their work and ours. Our compiler provides gadgets that work parallelly. In particular, our refresh gadget, `pRef`, works in 3 clock cycles, while their `sRef` works in $O(n)$ cycles. Moreover, our multiplication gadget, `Mult` works in $O(\log(n))$ cycles, while theirs, the ISW [22] works with $O(n)$ cycles. The other gadgets, which are the same in both compilers, need a constant number of cycles. Thus, our compiler is significantly faster than theirs for affine and general circuits.

Proof technique. Their proof technique is a particular case of our generalized one. In fact, it is enough to consider the graph depicted in Figure 2b. Since the values of the variables c_0^i and c_n^i are always 0, we can consider them as equal and glue the edge carrying c_0^i with c_n^i $\forall i$. In this way, we obtain a cylinder (the formal proof is in the full version.) In other words, the encodings of a value can also be considered like an orbit. Since $c_i^n = 0$ is fixed and not secret, we must assume that the adversary knows c_i^n. Thus, we consider the edges carrying c_i^n as always "leaked; hence, it always belongs to the leakage diagram. This condition means that our property of not orbiting becomes the property in which the leakage diagram's left, and right sides are not connected. Finally, their way to modify (that is, choosing the so-called *modifications vectors*) is a way to select a maximal containing the leakage diagram as we did in our proofs. Further, we show in the full version why we cannot use their technique for our gadgets. Informally, their technique does not work for our *LD* because choosing the so-called modification vectors would have been challenging (since there are no edges that must belong to the leakage diagrams).

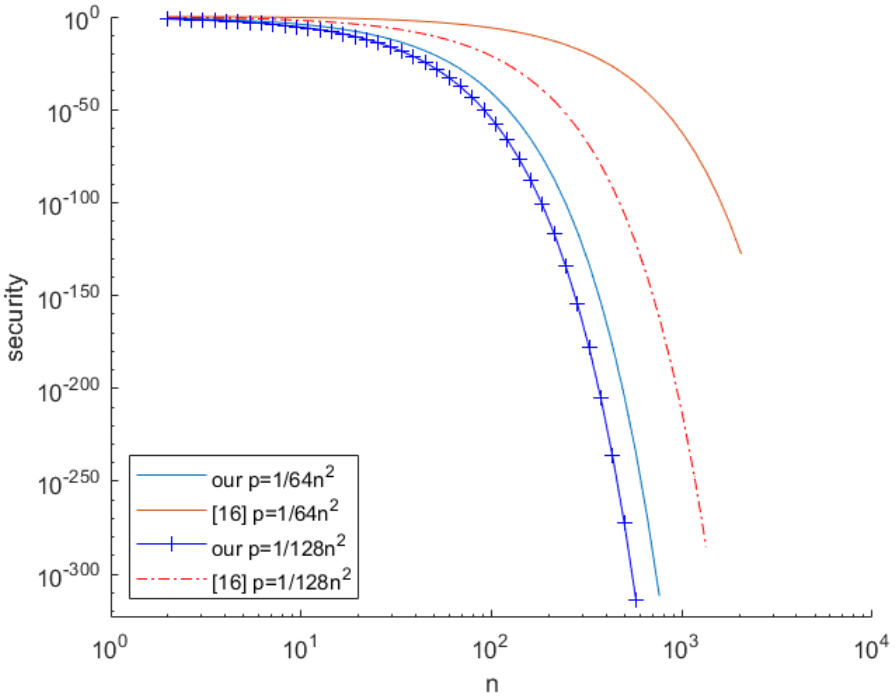


Figure 12: Comparison between the security of our masked circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale, with $p = O(1/n^2)$ (n the number of shares) and a single gate ($k = 1$). For general circuits the results must be multiplied with the number of gates of the circuit. We took $p = (1/64n^2)$ because we have to offset the factor $32np$ in [16].

Bound differences. Since every edge of $\mathsf{DG}_{\mathtt{pRef}}$ (Figure 5b) contains a single variable, while the c^i_j-edges contain two variables (Figure 2b), we have that our affine compiled circuits are $(O([9p]^n), p)$-private, while theirs is $(O([8\sqrt{3p}]^n), p)$-private. Thus, we have gained an order of magnitude. Second, by doing a more detailed analysis, we have proved that for the `Mult` gadgets and, thus, for the general compiled circuits, the security is $[18p+2(1-(1-p)^{8n}+1-(1-\sqrt{3p})^{n-1})]^n$, instead of $(32np + 4n\sqrt{3p})^n$. This observation proves better security gains as we have showed in Figs. 12 and 13 and with additional figures in App. D.

7 Conclusion

In this paper, we have started from the graphs introduced by Dziemboswki et al. [16]. Then, we showed how to use a broader class of graphs. Our graphs are more general than [16], and we used them to prove the security in the random probing model of our parallel compiler. Using this, we have proved that our compiler has $O(p^n)$-security for the affine case and $O((n^2p)^n)$-security for the general case. Besides being parallel, our compiler has the advantage that it is one of the simplest possible. This graph technique is interesting and could be applied to other compilers. Moreover, we believe the same technique can be applied to other probing models, such as the t-threshold probing model or the average-random probing model, or considering leakage models where glitches are considered, or also considering security in the presence of faults. We also believe that our technique can be applied to gadgets corresponding to bigger circuits, as those used to mask public-key encryption scheme. Finding dependency graphs which can be useful is still an interesting challenge. Finally, it might be interesting to improve the security bounds of our `Mult`

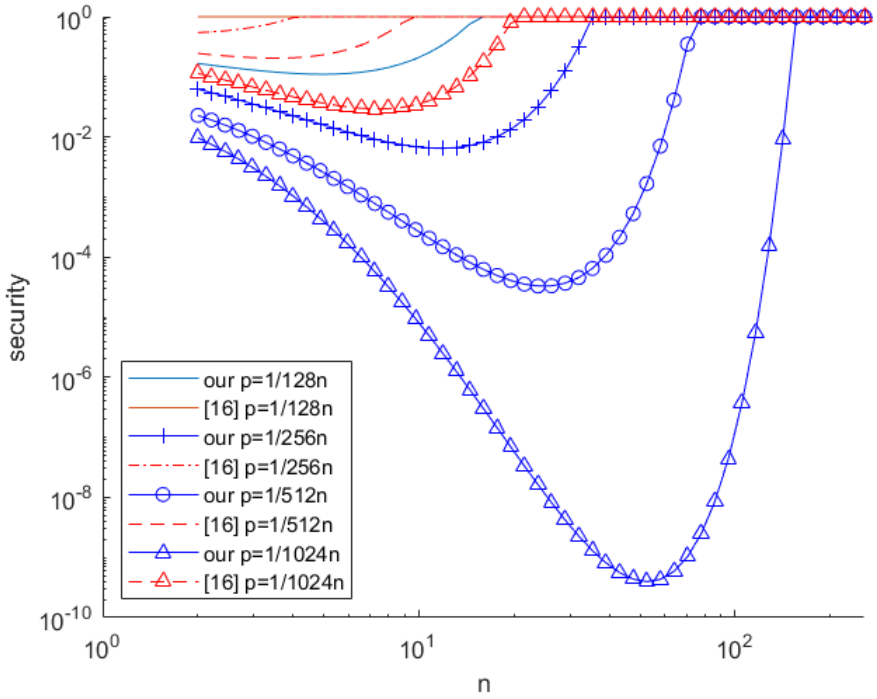


Figure 13: Comparison between the security of our masked circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale, with $p = O(1/n^2)$ (n the number of shares) and a single gate ($k = 1$). For general circuits the results must be multiplied with the number of gates of the circuit. We remember that in our choice of p we have to consider the factor $32np$ in [16].

and Add-gadget using improved graphs.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) via the DFG CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the European Commission(ERCEA), ERC Grant Agreement 101044770 CRYPTOLAYER. F. Berti was funded by Israel Science Foundation, ISF grant 2569/21.

We would like to thank Stefan Dziembowski and Karol Zebrowski for helpful discussions on earlier versions of this work, and our reviewers for the many helpful comments to improve the paper. Further, we thank Jürgen Pulkus for identifying an error in a previous version of this paper.

References

[1] Ananth, P., Ishai, Y., Sahai, A.: Private circuits: A modular approach. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 427–455. Springer (2018). https://doi.org/10.1007/978-3-319-96878-0_15, https://doi.org/10.1007/978-3-319-96878-0_15

[2] Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with o(1/\log (n)) leakage rate. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 586–615. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_21, https://doi.org/10.1007/978-3-662-49896-5_21

[3] Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10210, pp. 535–566 (2017). https://doi.org/10.1007/978-3-319-56620-7_19, https://doi.org/10.1007/978-3-319-56620-7_19

[4] Belaïd, S., Coron, J., Prouff, E., Rivain, M., Taleb, A.R.: Random probing security: Verification, composition, expansion and new constructions. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12170, pp. 339–368. Springer (2020). https://doi.org/10.1007/978-3-030-56784-2_12, https://doi.org/10.1007/978-3-030-56784-2_12

[5] Belaïd, S., Mercadier, D., Rivain, M., Taleb, A.R.: Ironmask: Versatile verification of masking security. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 142–160. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833600, https://doi.org/10.1109/SP46214.2022.9833600

[6] Belaïd, S., Mercadier, D., Rivain, M., Taleb, A.R.: Ironmask: Versatile verification of masking security. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 142–160. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833600, https://doi.org/10.1109/SP46214.2022.9833600

[7] Belaïd, S., Rivain, M., Taleb, A.R.: On the power of expansion: More efficient constructions in the random probing model **12697**, 313–343 (2021). https://doi.org/10.1007/978-3-030-77886-6_11, https://doi.org/10.1007/978-3-030-77886-6_11

[8] Belaïd, S., Rivain, M., Taleb, A.R.: On the power of expansion: More efficient constructions in the random probing model. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12697, pp. 313–343. Springer (2021). https://doi.org/10.1007/978-3-030-77886-6_11, https://doi.org/10.1007/978-3-030-77886-6_11

[9] Bogdanov, A., Ishai, Y., Srinivasan, A.: Unconditionally secure computation against low-complexity leakage. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11693, pp. 387–416. Springer (2019). https://doi.org/10.1007/978-3-030-26951-7_14, https://doi.org/10.1007/978-3-030-26951-7_14

[10] Bonneau, J., Mironov, I.: Cache-collision timing attacks against AES. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International

Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 201–215. Springer (2006). https://doi.org/10.1007/11894063_16, https://doi.org/10.1007/11894063_16

[11] Cassiers, G., Faust, S., Orlt, M., Standaert, F.: Towards tight random probing security. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 185–214. Springer (2021). https://doi.org/10.1007/978-3-030-84252-9_7, https://doi.org/10.1007/978-3-030-84252-9_7

[12] Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6476, pp. 46–61. Springer (2010). https://doi.org/10.1007/978-3-642-17650-0_5, https://doi.org/10.1007/978-3-642-17650-0_5

[13] Coron, J., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 410–424. Springer (2013). https://doi.org/10.1007/978-3-662-43933-3_21, https://doi.org/10.1007/978-3-662-43933-3_21

[14] Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. J. Cryptol. **32**(1), 151–177 (2019). https://doi.org/10.1007/s00145-018-9284-1, https://doi.org/10.1007/s00145-018-9284-1

[15] Dziembowski, S., Faust, S., Skorski, M.: Noisy leakage revisited. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 159–188. Springer (2015). https://doi.org/10.1007/978-3-662-46803-6_6, https://doi.org/10.1007/978-3-662-46803-6_6

[16] Dziembowski, S., Faust, S., Zebrowski, K.: Simple refreshing in the noisy leakage model. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 315–344. Springer (2019). https://doi.org/10.1007/978-3-030-34618-8_11, https://doi.org/10.1007/978-3-030-34618-8_11

[17] Gao, S., Mateer, T.D.: Additive fast fourier transforms over finite fields. IEEE Trans. Inf. Theory **56**(12), 6265–6272 (2010). https://doi.org/10.1109/TIT.2010.2079016, https://doi.org/10.1109/TIT.2010.2079016

[18] Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 444–461. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_25, https://doi.org/10.1007/978-3-662-44371-2_25

[19] Goudarzi, D., Joux, A., Rivain, M.: How to securely compute with noisy leakage in quasilinear complexity. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information

Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 547–574. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_19, https://doi.org/10.1007/978-3-030-03329-3_19

[20] Goudarzi, D., Prest, T., Rivain, M., Vergnaud, D.: Probing security through input-output separation and revisited quasilinear masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 599–640 (2021). https://doi.org/10.46586/tches.v2021.i3.599-640, https://doi.org/10.46586/tches.v2021.i3.599-640

[21] Hatcher, A.: Algebraic topology. Cambridge University Press (2005)

[22] Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27, https://doi.org/10.1007/978-3-540-45146-4_27

[23] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5_9, https://doi.org/10.1007/3-540-68697-5_9

[24] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25

[25] Lee, J.: Introduction to Topological Manifolds. Graduate Texts in Mathematics, Springer New York (2010), https://books.google.de/books?id=0XfowAEACAAJ

[26] Manetti, M.: Topology. UNITEXT, Springer International Publishing (2015), https://books.google.de/books?id=89zyCQAAQBAJ

[27] Medwed, M., Standaert, F., Joux, A.: Towards super-exponential side-channel security with efficient leakage-resilient prfs. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7428, pp. 193–212. Springer (2012). https://doi.org/10.1007/978-3-642-33027-8_12, https://doi.org/10.1007/978-3-642-33027-8_12

[28] Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4307, pp. 529–545. Springer (2006). https://doi.org/10.1007/11935308_38, https://doi.org/10.1007/11935308_38

[29] Prest, T., Goudarzi, D., Martinelli, A., Passelègue, A.: Unifying leakage models on a rényi day. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11692, pp. 683–712. Springer (2019). https://doi.org/10.1007/978-3-030-26948-7_24, https://doi.org/10.1007/978-3-030-26948-7_24

[30] Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 142–159. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_9, https://doi.org/10.1007/978-3-642-38348-9_9

[31] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 764–783. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_37, https://doi.org/10.1007/978-3-662-47989-6_37

[32] Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010). https://doi.org/10.1007/978-3-642-15031-9_28, https://doi.org/10.1007/978-3-642-15031-9_28

[33] Trichina, E.: Combinational logic design for AES subbyte transformation on masked data. IACR Cryptol. ePrint Arch. p. 236 (2003), http://eprint.iacr.org/2003/236

[34] Walter, C.D.: Sliding windows succumbs to big mac attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 286–299. Springer (2001). https://doi.org/10.1007/3-540-44709-1_24, https://doi.org/10.1007/3-540-44709-1_24

Supplementary Material

A Detailed Proofs

A.1 Classification of the Maximals for `pRef`

Proof. Now, we study the left cases:

- *Case $i < j$.* It depends if either b_i or r_i belongs to LD.
  - *Case $r_i \in LD$. (Case $\mathcal{M}^{left}_{i,j}$)* We start observing that no edge in

$$\{r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_i, b_j, ..., b_{n-1}\}$$

    can be in LD. In fact, the loop

$$x_0, ..., x_l, r_l, y_{l+1}, ..., y_i, r_i, x_{i+1}, ..., x_{n-1} \text{ for } 0 \leq l < i, \text{ or}$$

$$y_0, ..., y_i, r_i, x_{i+1}, ..., x_l, r_l, y_{l+1}, ..., y_{n-1} \text{ for } j \leq l < n-1, \text{ or}$$

$$x_0, ..., x_l, b_l, y_{l+1}, ..., y_i, r_i, x_{i+1}, ..., x_{n-1} \text{ for } 0 \leq l < i, \text{ or}$$

$$y_0, ...y_i, r_i, x_i, ..., x_{l-1}, b_l, y_{l+1}, ..., y_{n-1} \text{ for } j \leq l \leq n-1$$

    orbits and it is contained in $LD \cup \{e\}$ with $e = r_l$ or b_l (we are bypassing the edge x_i or y_j). To show that

$$\mathsf{DG} \setminus \{r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_i, b_j, ..., b_{n-1}\}$$

    is simply connected, we observe that we can draw a line from the top orbit of the cylinder crossing only the edges

$$x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_i, b_j, ..., b_{n-1}).$$

    If we cut a cylinder over such a line we obtain a rectangle (topologically), which is simply connected. Since, we have not cut any edge of the LD, then, LD is simply connected [12]. Thus,

$$LD = \mathsf{DG} \setminus \{x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_i, b_j, ..., b_{n-1}\}$$

    is a maximal diagram. Moreover,

$$\mathsf{DG} \setminus \{x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_i, b_{ji}, ..., b_{n-1}\} =$$
$$\{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\}.$$

  - *Case $b_i \in LD$. (Case $\mathcal{M}^{right}_{i,j}$)* We observe that no edge in

$$\{r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\}$$

    can belong to LD. Otherwise either

$$x_0, ...x_{i-1}, b_i, y_{i+1}, ..., y_l, r_l, x_{l+1}, ..., x_{n-1} \text{ for } i \leq l < j-1, \text{ or}$$

$$x_0, ...x_{i-1}, b_i, y_{i+1}, ..., y_l, b_l, x_l, ..., x_{n-1} \text{ for } i+1 \leq l < j-1$$

[12] In fact, if Y is a subspace of a simply connected space X, thus, retractable to a point (and the retraction $f : X \to P$ where P is a point can be easily defined as $f' : Y \to P$ as $f' := f_{|Y}$))

is a loop which orbits and it is contained in $LD \cup \{e\}$ with $e = r_l$ or b_l (we bypass the missing edge x_i). As done before, to show that

$$\mathsf{DG} \setminus \{x_i, y_j, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\}$$

does not orbit, it is enough to observe that we can draw a line from the top orbit of the cylinder to the bottom orbit of the cylinder crossing only the edges

$$(x_i, y_j, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}).$$

If we cut a cylinder over such a line we obtain a rectangle (topologically), which is simply connected. Similarly as above, this proves that LD is simply connected. Thus,

$$LD = \mathsf{DG} \setminus \{x_i, y_j, r_i, ..., r_{j-1}, b_{i+1}, ..., b_{j-1}\} = \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, r_0, ..., r_{i-1}, rj, ..., r_{n-1}, b_0, ..., b_i, b_j, ..., b_{n-1}\}$$

is a maximal diagram.

- *Case $i > j$.* It depends if either b_i or r_i belongs to LD.
  - *Case $b_i \in LD$. (Case $\mathcal{M}_{i,j}^{right}$)* We start observing that no edge in

$$\{r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}\}$$

can be in LD. In fact, the loop

$$y_0, ..., y_l, r_l, x_{l+1}, ..., x_{i-1}, b_i, y_{i+1}, ..., y_{n-1} \text{ for } 0 \leq l < j, \text{ or}$$

$$x_0, ..., x_{i-1}, b_i, y_{i+1}, ..., y_l, r_l, x_{l+1}, ..., x_{n-1} \text{ for } i \leq l \leq n-1, \text{ or}$$

$$y_0, ..., y_l, b_l, x_{l-1}, ..., x_i, b_i, y_{j+1}, ..., y_{n-1} \text{ for } 0 \leq l < j, \text{ or}$$

$$x_0, ...x_{i-1}, b_i, y_{i+1}, ..., y_l, b_l, x_l, ..., x_{n-1} \text{ for } i < l \leq n-1$$

orbits and it is contained in $LD \cup \{e\}$ with $e = r_l$ or b_l (we are bypassing the edge x_i or y_j). As done before, to show that

$$\mathsf{DG} \setminus \{x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}\}$$

does not orbit, we observe that we can draw a line from the top orbit of the cylinder crossing only the edges

$$(x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}).$$

Similarly as above, this proves that LD is simply connected. Thus,

$$LD = \mathsf{DG} \setminus \{x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}\} = \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, r_{j+1}, ..., r_{i-1}, b_j, ..., b_j\}$$

is a maximal diagram.

  - *Case $r_i \in LD$. (Case $\mathcal{M}_{i,j}^{left}$)* We start observing that no edge in

$$\{r_j, ..., r_{i-1}, b_i, ..., b_j\}$$

can be in LD. In fact, the loop

$$x_0, ..., x_l, r_l, y_{l+1}, ..., y_i, r_i, x_{i+1}, ..., x_{n-1} \text{ for } j \leq l < i, \text{ or}$$

$$x_0, ..., x_l, b_l, y_{l+1}, ..., y_i, r_i, x_{i+1}, ..., x_{n-1} \text{ for } j \leq l \leq i, \text{ or}$$

orbits and it is contained in $LD \cup \{e\}$ with $e = r_l$ or b_l (we are bypassing the edge x_i or y_j). As done before, to show that

$$\mathsf{DG} \setminus \{x_i, y_j, r_j, ..., r_{i-1}, b_i, ..., b_j\}$$

is simply connected, we observe that we can draw a line from the top orbit of the cylinder crossing only the edges

$$(x_i, y_j, r_j, ..., r_{i-1}, b_i, ..., b_j).$$

If we cut a cylinder over such a line we obtain a rectangle (topologically), which is simply connected. Since, we have not cut any edge of the LD, then, LD is simply connected, thus, it does not orbit. Thus,

$$\begin{aligned} LD = \mathsf{DG} \setminus \{x_i, y_j, r_0, ..., r_{i-1}, r_j, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}\} = \\ \{x_0, ..., x_{i-1}, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_{j+1}, ..., y_{n-1}, \\ r_0, ..., r_{j-1}, r_i, ..., r_{n-1}, b_0, ..., b_{j-1}, b_{i+1}, ..., b_{n-1}\} \end{aligned}$$

is a maximal diagram.

Note that cases *iii*) and *v*) are the same (as it happens for cases *iv*) and *vi*)). It is enough to index the shares with indexes in $[j, ..., j + n - 1]$.
Thus, we have classified the possible maximal simply connected subgraph of the DG of `pRef`. □

A.2 Proof of Proposition 6

Proof. Let $\delta = x' - x$. Let A be the values carried by the variables of `pRef` during the $\mathsf{Leak}(\texttt{pRef}, x, \mathbf{r}, p)$-experiment, where $\mathbf{r} = r_0, ..., r_{n-1}$ is the randomness used.

- *The maximal is of type $\mathcal{M}^{right}_{i,j}$ with $i < j$* [13]. The values carried in A' are

$$\begin{aligned} \{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1}, \\ r_0, ..., r_{i-1}, r_i - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{n-1}, b_0, ..., b_i, b_{i+1} - \delta, ..., b_{j-1} - \delta, b_j, ..., b_{n-1}\}. \end{aligned}$$

  First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

  The same holds for the shares of y.
  Now, let us consider `pRef` with input $x_0, ..., x_{i-1}, x_i+\delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{n-1} - \delta$. For $0 \leq i' < i$, $b_{i'} = x_{i'} + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. For i, $b_i = (x_i + \delta) + (r_i - \delta)$, and $y_i = b_i - r_{i-1}$. For $i < i' < j$ $b_{i'} - \delta = x_{i'} + (r_{i'} - \delta)$, and $y_{i'} = (b_{i'} - \delta) - (r_{i'-1} - \delta)$. For j, $b_j = x_j + r_j$, and $y_j + \delta = b_j - (r_{j-1} - \delta)$. For $i < i' \leq n - 1$, $b_{i'} = x_i + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. Thus, the values carried by A' are those of an honest evaluation of `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{n-1} - \delta$.

- *The maximal is of type $\mathcal{M}^{right}_{i,j}$ with $i > j$.* The values carried in A' are

$$\begin{aligned} \{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1}, \\ r_0 - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{i-1}, r_j - \delta, ..., r_{n-1} - \delta, \\ b_0 - \delta, ..., b_{j-1} - \delta, b_j, ..., b_i, b_{i+1} - \delta, ..., b_{n-1} - \delta\}. \end{aligned}$$

[13] For simplicity, when we use $i < j$, we assume that $i, j \in [n] \subset \mathbb{Z}$ and not in $\mathbb{Z}_n$.

First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

The same holds for the shares of y.
Now, let us consider `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{i-1}, r_i - \delta, ..., r_{n-1} - \delta$. For $0 \leq i' < j$ $b_{i'} - \delta = x_{i'} + (r_{i'} - \delta)$, and $y_{i'} = (b_{i'} - \delta) - (r_{i'-1} - \delta)$. For j, $b_j = x_j + r_j$, and $y_j + \delta = b_j - (r_{j-1} - \delta)$. For $j < i' < i$, $b_{i'} = x_i + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. For i, $b_i = (x_i + \delta) + (r_i - \delta)$, and $y_i = b_i - r_{i-1}$. For $i < i' \leq n - 1$ $b_{i'} - \delta = x_{i'} + (r_{i'} - \delta)$, and $y_{i'} = (b_{i'} - \delta) - (r_{i'-1} - \delta)$. Thus, the values carried by A' are those of an honest evaluation of `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 - \delta, ..., r_{j-1} - \delta, r_j, ..., r_{i-1}, r_j - \delta, ..., r_{n-1} - \delta$.

- *The maximal is of type* $\mathcal{M}^{left}_{i,i}$. The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1},$$
$$r_0, ..., r_{n-1}, b_0, ..., b_{i-1}, b_i + \delta, b_{i+1}, ..., b_{n-1}\}.$$

First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

The same holds for the shares of y.
Now, let us consider `pRef` with input $x_0, ..., x_{i-1}, x_i+\delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{n-1}$. For $i' \neq i$, $b_{i'} = x_{i'} + r_{i'}$ and $y_{i'} = b_{i'} - r_{i'-1}$. In stead, for i, $b_i + \delta = (x_i + \delta) + r_i$, and $y_i + \delta = (b_i + \delta) - r_{i-1}$. Thus, the values carried by A' are those of an honest evaluation of `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{n-1}$.

- *The maximal is of type* $\mathcal{M}^{left}_{i,j}$ *with* $i < j$. The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1},$$
$$r_0 + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{j-1}, r_j + \delta, ..., r_{n-1} + \delta,$$
$$b_0 + \delta, ..., b_i + \delta, b_{i+1}, ..., b_{j-1}, b_j + \delta, ..., b_{n-1} + \delta\}.$$

First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

The same holds for the shares of y.
Now, let us consider `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{j-1}, r_j + \delta, ..., r_{n-1} + \delta$. For $0 \leq i' < i$ $b_{i'} + \delta = x_{i'} + (r_{i'} + \delta)$, and $y_{i'} = (b_{i'} + \delta) - (r_{i'-1} + \delta)$. For i, $b_i + \delta = (x_i + \delta) + (r_i)$, and $y_i = (b_i + \delta) - (r_{i-1} + \delta)$. For $i < i' < j$, $b_{i'} = x_i + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. For j, $b_j + \delta = x_j + (r_j + \delta)$, and $y_j + \delta = (b_j + \delta) - r_{j-1}$. For $j < i' \leq n - 1$ $b_{i'} + \delta = x_{i'} + (r_{i'} + \delta)$, and $y_{i'} = (b_{i'} + \delta) - (r_{i'-1} + \delta)$. Thus, the values carried by A' are those of an honest evaluation of `pRef` with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{j-1}, r_j + \delta, ..., r_{n-1} + \delta$.

- *The maximal is of type* $\mathcal{M}^{left}_{i,j}$ *with* $i > j$. The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{j-1}, y_i + \delta, y_{i+1}, ..., y_{n-1},$$
$$r_0, ..., r_{j-1}, r_j + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{n-1},$$
$$b_0, ..., b_{j-1}, b_j + \delta, ..., b_i + \delta, b_{i+1}, ..., b_{n-1}\}.$$

First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \delta) + x_{i+1} + ... + x_{n-1} = x_0 + ... + x_{n-1} + \delta = x + (x' - x) = x'.$$

The same holds for the shares of y.
Now, let us consider pRef with input $x_0, ..., x_{i-1}, x_i+\delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{j-1}$, $r_j + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{n-1}$. For $0 \leq i' < j$, $b_{i'} = x_i + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. For j, $b_j + \delta = x_j + (r_j + \delta)$, and $y_j + \delta = (b_j + \delta) - r_{j-1}$. For $j < i' < i$ $b_{i'} + \delta = x_{i'} + (r_{i'} + \delta)$, and $y_{i'} = (b_{i'} + \delta) - (r_{i'-1} + \delta)$. For i, $b_i + \delta = (x_i + \delta) + (r_i)$, and $y_i = (b_i + \delta) - (r_{i-1} + \delta)$. For $i < i' \leq n - 1$, $b_{i'} = x_i + r_{i'}$, and $y_{i'} = b_{i'} - r_{i'-1}$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{i-1}, x_i + \delta, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{j-1}, r_j + \delta, ..., r_{i-1} + \delta, r_i, ..., r_{n-1}$.

Having treated all the cases of maximal diagrams (see Proposition 4), we have concluded. □

A.3 Proof of Proposition 8

Proof. Let A be the values carried by the variables of pRef during the Leak(pRef, $x, \mathbf{r}, p$)-experiment, where $\mathbf{r} = r_0, ..., r_{n-1}$ is the randomness used.

- *The gap is of type $BLGap_{i,i'}$ with $i' < i$.* The values carried in A' are

$$\{x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{n-1}, r_0, ..., r_{i'-1},$$
$$r_{i'} + \gamma, ..., r_{i-1} + \gamma, r_i, ..., r_{n-1}, b_0, ..., b_{i'}, b_{i'+1} + \gamma, ..., b_i + \gamma, b_{i+1}, ..., b_{n-1}\}.$$

  First we observe that

$$x_0 + ... + x_{i'-1} + (x_{i'} - \gamma) + x_{i'+1} + ... + x_{i-1} + (x_i - \gamma), x_{i+1}, ..., x_{n-1} = x_0 + ... + x_{n-1} + \gamma - \gamma = x.$$

  Since, we have not touched the shares of y, they carry an encoding of y which is equal to x. Now, let us consider pRef with input $x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i'-1}, r_{i'} + \gamma, ..., r_{i-1} + \gamma, r_i, ..., r_{n-1}$. For $0 \leq j < i'$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i', $b_{i'} = (x_{i'} - \gamma) + (r_{i'} + \gamma)$, and $y_{i'} = b_{i'} - r_{i'-1}$. For $i' < j < i$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. For i, $b_i + \gamma = (x_i + \gamma) + r_i$, and $y_i = (b_i + \gamma) - (r_{i-1} + \gamma)$. For $i < j \leq n - 1$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i'-1}, r_{i'} + \gamma, ..., r_{i-1} + \gamma, r_i, ..., r_{n-1}$.

- *The gap is of type $BRGap_{i,i'}$ with $i < i'$.* The values carried in A' are

$$\{x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}, y_0, ..., y_{n-1}, r_0, ..., r_{i-1},$$
$$r_i - \gamma, ..., r_{i'-1} - \gamma, r_{i'}, ..., r_{n-1}, b_0, ..., b_i, b_{i+1} - \gamma, ..., b_{i'} - \gamma, b_{i'+1}, ..., b_{n-1}\}.$$

  First we observe that

$$x_0 + ... + x_{i-1} + (x_i + \gamma) + x_{i+1} + ... + x_{i'-1} + (x_{i'} - \gamma), x_{i'+1}, ..., x_{n-1} = x_0 + ... + x_{n-1} + \gamma - \gamma = x.$$

  Since, we have not touched the shares of y, they carry an encoding of y which is equal to x. Now, let us consider pRef with input $x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i - \gamma, ..., r_{i'-1} - \gamma, r_{i'}, ..., r_{n-1}$. For $0 \leq j < i$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i, $b_i = (x_i + \gamma) + (r_i - \gamma)$, and $y_i = b_i - r_{i-1}$. For $i < j < i'$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. For i', $(b_{i'} - \gamma) = (x_{i'} - \gamma) + r_{i'}$, and $y_{i'} = (b_{i'} - \gamma) - (r_{i'} - \gamma)$. For $i' < j \leq n - 1$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i - \gamma, ..., r_{i'-1} - \gamma, r_{i'}, ..., r_{n-1}$.

- *The gap is of type $BRGap_{i,i'}$ with $i' < i$.* The values carried in A' are

$$\{x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}, y_0, ..., y_{n-1}, r_0 - \gamma, ..., r_{i'-1} - \gamma,$$
$$r_{i'}, ..., r_{i-1}, r_i - \gamma, ..., r_{n-1} - \gamma, b_0 - \gamma, ..., b_{i'} - \gamma, b_{i'+1}, ..., b_i, b_{i+1} - \gamma, ..., b_{n-1} - \gamma\}.$$

First we observe that

$$x_0 + ... + x_{i'-1} + (x_{i'} - \gamma) + x_{i'+1} + ... + x_{i-1} + (x_i - \gamma), x_{i+1}, ..., x_{n-1} = x_0 + ... + x_{n-1} + \gamma - \gamma = x.$$

Since, we have not touched the shares of y, they carry an encoding of y which is equal to x. Now, let us consider pRef with input $x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 - \gamma, ..., r_{i'-1} - \gamma, r_{i'}, ..., r_{i-1}, r_i - \gamma, ..., r_{n-1} - \gamma$. For $0 \leq < j < i'$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. For i', $(b_{i'} - \gamma) = (x_{i'} - \gamma) + r_{i'}$, and $y_{i'} = (b_{i'} - \gamma) - (r_{i'} - \gamma)$. For $i' < j < i$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i, $b_i = (x_i + \gamma) + (r_i - \gamma)$, and $y_i = b_i - r_{i-1}$. For $i < j \leq n - 1$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{i'-1}, x_{i'} - \gamma, x_{i'+1}, ..., x_{i-1}, x_i + \gamma, x_{i+1}, ..., x_{n-1}$ and randomness $r_0 - \gamma, ..., r_{i'-1} - \gamma$.

- *The gap is of type $ULGap_{i,i'}$ with $i < i'$.* The values carried in A' are

$$\{x_0, ..., x_{n-1}, y_0, ..., y_{i-1}, x_i + \gamma, y_{i+1}, ..., y_{i'-1}, y_{i'} - \gamma, y_{i'+1}, ...y_{n-1}, r_0 - \gamma, ..., r_{i-1} - \gamma,$$
$$r_i, ..., r_{i'-1}, r_{i'} - \gamma, ..., r_{n-1} - \gamma, b_0 - \gamma, ..., b_{i-1} - \gamma, b_i, ..., b_{i'-1}, b_{i'} - \gamma, ..., b_{n-1} - \gamma\}.$$

First we observe that

$$y_0 + ... + y_{i-1} + (y_i + \gamma) + y_{i+1} + ... + y_{i'-1} + (y_{i'} - \gamma), y_{i'+1}, ..., y_{n-1} = y_0 + ... + y_{n-1} + \gamma - \gamma = y.$$

Since, we have not touched the shares of x, they carry an encoding of x..
Now, let us consider pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0 - \gamma, ..., r_{i-1} - \gamma, r_i, ..., r_{i'-1}, r_{i'} - \gamma, ..., r_{n-1} - \gamma$. For $0 \leq j < i$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. For i, $b_i = x_i + r_i$, and $y_i + \gamma = b_i - (r_{i-1} - \gamma)$. For $i < j < i'$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i', $b_{i'} - \gamma = x_{i'} + (r_{i'} - \gamma)$, and $y_{i'} - \gamma = (b_{i'} - \gamma) - r_{i'-1}$. For $i' < j \leq n-1$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0 - \gamma, ..., r_{i-1} - \gamma, r_i, ..., r_{i'-1}, r_{i'} - \gamma, ..., r_{n-1} - \gamma$.

- *The gap is of type $ULGap_{i,i'}$ with $i' < i$.* The values carried in A' are

$$\{x_0, ..., x_{n-1}, y_0, ..., y_{i'-1}, y_{i'} - \gamma, y_{i'+1}, ..., y_{i-1}, y_i + \gamma, y_{i+1}, ..., y_{n-1}, r_0, ..., r_{i'-1},$$
$$r_{i'} - \gamma, ..., r_{i-1} - \gamma, r_i, ..., r_{n-1}, b_0, ..., b_{i'-1}, b_{i'} - \gamma, ..., b_{i-1} - \gamma, b_i, ..., b_{n-1}\}.$$

First we observe that we have not touched the shares of x. Second,

$$y_0 + ... + y_{i'-1} + (y_{i'} - \gamma) + y_{i'+1} + ... + y_{i-1} + (y_i - \gamma), y_{i+1}, ..., y_{n-1} = y_0 + ... + y_{n-1} + \gamma - \gamma = y.$$

Now, let us consider pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0, ..., r_{i'-1}, r_{i'} - \gamma, ..., r_{i-1} - \gamma, r_i, ..., r_{n-1}$. For $0 \leq j < i'$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i', $b_{i'} - \gamma = x_{i'} + (r_{i'} - \gamma)$, and $y_{i'} - \gamma = (b_{i'} - \gamma) - r_{i'}$. For $i' < j < i$, $b_j - \gamma = x_j + (r_j - \gamma)$, and $y_j = (b_j - \gamma) - (r_{j-1} - \gamma)$. For i, $b_i = x_i + r_i$, and $y_i + \gamma = b_i - (r_{i-1} - \gamma)$. For $i < j \leq n-1$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0, ..., r_{i'-1}, r_{i'} - \gamma, ..., r_{i-1} - \gamma, r_i, ..., r_{n-1}$.

- *The gap is of type $URGap_{i,i'}$ with $i < i'$.* The values carried in A' are

$$\{x_0, ...x_{n-1}, y_0, ..., , y_{i-1}, y_i + \gamma, y_{i+1}, ..., y_{i'-1}, y_{i'} - \gamma, y_{i'+1}, ..., y_{n-1}, r_0, ..., r_{i-1},$$
$$r_i + \gamma, ..., r_{i'-1} + \gamma, r_{i'}, ..., r_{n-1}, b_0, ..., b_{i-1}, b_i + \gamma, ..., b_{i'-1} + \gamma, b_{i'}, ..., b_{n-1}\}.$$

First we observe that we have not touched the shares of x. Second,

$$y_0 + ... + y_{i-1} + (y_i + \gamma) + y_{i+1} + ... + y_{i'-1} + (y_{i'} - \gamma), y_{i'+1}, ..., y_{n-1} = y_0 + ... + y_{n-1} + \gamma - \gamma = y.$$

Now, let us consider pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i + \gamma, ..., r_{i'-1} + \gamma, r_{i'}, ..., r_{n-1}$. For $0 \leq j < i$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i, $b_i + \gamma = x_i + (r_i + \gamma)$, and $y_i + \gamma = (b_i + \gamma) - r_{i-1}$. For $i < j < i'$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. For i', $b_{i'} = x_{i'} + r_{i'}$, and $y_{i'} - \gamma = b_{i'} - (r_{i'} + \gamma)$. For $i' < j \leq n - 1$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0, ..., r_{i-1}, r_i + \gamma, ..., r_{i'-1} + \gamma, r_{i'}, ..., r_{n-1}$.

- *The gap is of type $URGap_{i,i'}$ with $i' < i$.* The values carried in A' are

$$\{x_0, ..., x_{n-1}, y_0, ..., y_{i'-1}, y_{i'} - \gamma, y_{i'+1}, ..., y_{i-1}, y_i + \gamma, y_{i+1}, ..., y_{n-1}, r_0 + \gamma, ..., r_{i'-1} + \gamma,$$
$$r_{i'}, ..., r_{i-1}, r_i + \gamma, ..., r_{n-1} + \gamma, b_0 + \gamma, ..., b_{i'} + \gamma, b_{i'+1}, ..., b_i, b_{i+1} + \gamma, ..., b_{n-1} + \gamma\}.$$

  First we observe that we have not touched the shares of x, Second,

$$y_0 + ... + y_{i'-1} + (y_{i'} - \gamma) + y_{i'+1} + ... + y_{i-1} + (y_i - \gamma), y_{i+1}, ..., y_{n-1} = y_0 + ... + y_{n-1} + \gamma - \gamma = y.$$

  Now, let us consider pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0+\gamma, ..., r_{i'-1}+\gamma, r_{i'}, ..., r_{i-1}, r_i+\gamma, ..., r_{n-1} + \gamma$. For $0 \leq < j < i'$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. For i', $b_{i'} = x_{i'} + r_{i'}$, and $y_{i'} - \gamma = b_{i'} - (r_{i'} = \gamma)$. For $i' < j < i$ $b_j = x_j + r_j$, and $y_j = b_j - r_j$. For i, $b_i + \gamma = x_i + (r_i + \gamma)$, and $y_i + \gamma = (b_i + \gamma) - r_{i-1}$. For $i < j \leq n - 1$, $b_j + \gamma = x_j + (r_j + \gamma)$, and $y_j = (b_j + \gamma) - (r_{j-1} + \gamma)$. Thus, the values carried by A' are those of an honest evaluation of pRef with input $x_0, ..., x_{n-1}$ and randomness $r_0 + \gamma, ..., r_{i'-1} + \gamma$.

□

A.4 Proof of Theorem 4

Proof. Finally, we explain why it holds

$$\Pr[E_i] \leq [(2 + 5p')p']^n \leq [4p']^n.$$

This is due to the fact that not all edges directly give the shortest path and require further edges that are also added with probability p'. Moreover, if $p' < 1/3$, $5p' < 5/3 < 2$, it holds $(2 + 5p')p']^n \leq [4p']^n$.

Regarding the general circuits, we observe that there are at most 8 edges connected to each node of the dependency graph and each node belong to one orbit sharing, which can correspond to any gadget. Moreover, there can be at most 3 pRef connected to this gadget, thus, 3 dependency graph of pRef connected to that orbit. Thus, the 8 edges can come as follows: 2 of one orbit sharing, 2 for each pRef (one vertical containing an r_i and one diagonal containing either b_i (if the orbit shared is the output orbit of pRef) or b_{i-1} (if the orbit shared is the input orbit of pRef). Now, we want to bound the probability that at least one of this edge is leaked: this is $\leq$ Pr [one of the edge of the same orbit leaks] + Pr [one edge of pRef is leaked]. But we have already computed the probability that one edge of the same orbit is leaked. The worst case is if the gadget is Mult (Prop. 5) and it is $1 - (1 - p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1}$, while for pRef the probability that an edge is added to the leakage diagram is bounded by $3p$ (Prop. 6). Since to have a leakage diagram which orbits around the dependency graph, we need to have at least n different consecutive edges in the leakage diagram, thus the probability that there is an orbit containing the first node of the ith orbit is bounded by

$$\Pr[E_i] \leq [2(1 - (1 - p)^{8n} + 1 - (1 - \sqrt{3p})^{n-1}) + 6(3p)]^n.$$

□

B Other Results

The goal of this section is to prove that the probability of `Mult` on n-shares to be secure in presence of leakage p is negligible for $p = O(\sqrt{n})$ asymptotically.

We need to introduce some results from analysis. We start from a well known lemma

Lemma 1. *Let $x \in [0, 1]$, let $y \in [1, \infty]$, then*

$$1 - (1 - x)^y \leq yx.$$

Proof. Let $f(x) := 1 - (1 - x)^y - yx$. The claim is equivalent to say that $f(x) \leq 0 \ \forall x \in [0, 1]$. We observe that $f(0) = 0$. Then, to show that $f(x) \leq 0$, it is enough to show that $f'(x) \leq 0$, where f' is the derivative. f' is well defined $\forall y \geq 1 \ \forall p \in [0, 1]$. But, for $y > 0$,

$$f'(x) = -y(1 - x)^{y-1} - yx$$

Since $x \leq 1$, then $|1 - x| \leq 1$. Since y is positive, then $(1 - x)^{y-1} \leq 1$, thus $f'(x) \leq 0 \ \forall x \in [0, 1]$. □

This is a very loose approximation for $1-(1-x)^y$. Nevertheless, it is used in many cryptographic papers about masking, see for example, [16]. In fact, bounding $1 - (1 - x)^y \leq yx$ is good only when x is close to zero, because, using standard analysis

$$f(x) = f(0) + f'(0)x + o(x^2).$$

On the other hand this bounding is very loose when x is not close to zero as we show for example in Figure 14. Finally, note that in our Theorems we have to elevate $(1 - (1 - p)^n)$ to the nth power (as, for example in Theorem 1). And the difference between $(1 - (1 - p)^n)^n$ and $(np)^n$ is much bigger, as shown in Figure 15. This is the reason to leave $(1 - (1 - p)^n)^n$ in Theorem 1 and Theorem 4 [14].

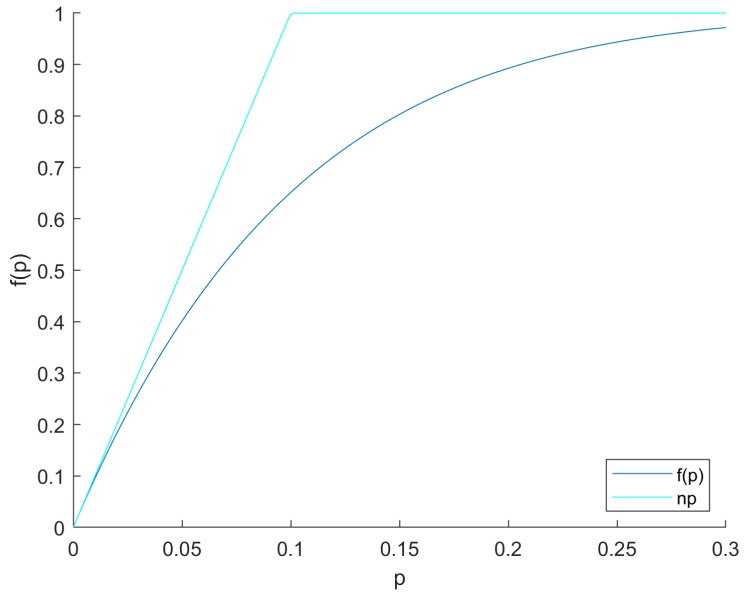


Figure 14: Comparison between $f(p) = 1 - (1 - p)^n$ and np for $n = 10$ (for clarity we have set $np = 1$ when it should be bigger than 1.)

[14] We have removed one subsection in this appendix.

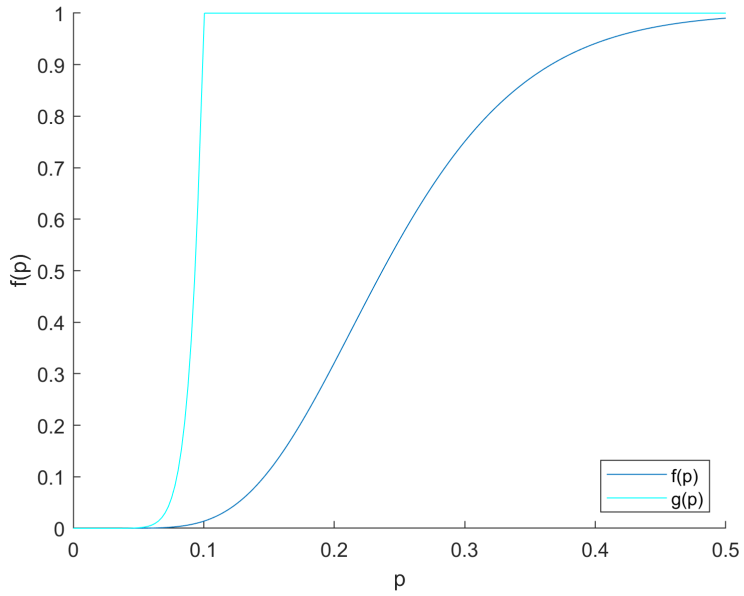


Figure 15: Comparison between $f(p) = (1-(1-p)^n)^n$ and $g(p) = (np)^n$ for $n = 10$ (for clarity we have set $g(p) = 1$ when it should be bigger than 1.)

C Background in Topology

This section is devoted to present some basic concepts of topology. There are many good books for them, as [25, 21]. All the definitions, propositions and theorems are taken from [26] when not explicitly stated. In the first section, we give the definition and the results needed, while in the second part we give more concepts which can be useful to understand the concept introduced in the first section. At the end, we explain how we can compute the fundamental group of our dependency graphs when built attaching the lateral surfaces.

C.1 Essential Concepts

Definition 6 (Connected component)**.** Let X be a topological space. A subspace $C \subset X$ is called *connected component* of X (or just *component*) if it satisfies two properties:

- C is connected
- $C \subset A$ and C connected implies $C = A$.

Definition 7 (Attaching)**.** Take disjoint spaces X, Y, a subspace $K \subset X$ and continuous map $f : K \to Y$. Put on $X \cup Y$ the disjoint-union topology (...), the one for which the inclusions $X \leftarrow X \cup Y$ and $Y \leftarrow X \cup Y$ are open immersions. Now define a new space

$$X \cup_f Y = (X \cup Y)/\sim,$$

where $\sim$ is the smallest equivalence relation, for which $x \sim f(x)$ for every $x \in K$. This definition makes sense when $K = \emptyset$, as well.
The space $X \cup_f Y$ is said to be obtained *attaching X to Y via f; the map f is called attaching map.*

Definition 8 (Homotopy)**.** Two continuous maps $f_0, f_1 : X \to Y$ are said to be *homotopic* if there is a continuous function

$$F : X \times [0,1] \to Y$$

s.t. $F(x,0) = f_0(x)$ and $F(x_1) = f_1(x)$ $\forall x \in X$. Such an F is called a *homotopy* between f_0 and f_1.

The homotopy relation is an equivalence relation over the set $C(X, Y)$ which consists over all the continuous map from X to Y.

Definition 9 (Path space)**.** The *space of paths* in X between points $a, b \in X$ is

$$\Omega(X, a, b) = \{\alpha : I \to X \mid a \text{ continuous}, \alpha(0) = a, \alpha(1) = b\}.$$

where $I = [0, 1] \subset \mathbb{R}$.
A loop is a path where $\alpha(0) = \alpha(1)$,

Given two loops α and α' we can define a group operation as the loop obtained appending α' to α. Formally,

$$\alpha \star \beta(t) := \begin{cases} \alpha(2t) & \text{if } 0 \leq t \leq \frac{t}{2} \\ \beta(2t-1) & \text{if } \frac{t}{2} \leq t \leq 1 \end{cases}$$

Let X a space and a point $a \in X$, one defines $\pi_1(X, a)$ as the quotient of $\Omega(X, a, a)$ by a path homotopy equivalence. For any loop $\alpha \in \Omega(X, a, a)$, we write $[\alpha]$ for its homotopy class. It can be proven that it is $\pi_1(X, a)$ is a group.

Definition 10 (Fundamental group)**.** The group $\pi_1(X, a)$ is called the *fundamental group*, or *first homotopy group* of X with base point a.

Definition 11 (Simply connected)**.** A space is *simply connected* if it is path connected if it is path connected and its fundamental group is trivial.

Roughly speaking a space is simply connected if there are no holes inside this.

C.1.1 The Fundamental Group of the Cylinder

We observe that we can see the cylinder C described in the previous subsection as the product of the circumference S_1 with $S_1 := \{(x, y) \in \mathbb{R}^2 \text{ s.t. } x^2 + y^2 = 1\}$ with $[0, 1]$. That is, $C = S_1 \times [0, 1]$. Here, we want to compute its fundamental group.

To do this, we have the following results:

Proposition 11. *The fundamental group of the product of two spaces is isomorphic to the product of the fundamental groups:*

$$\pi_1(X \times Y, (a, b)) \simeq \pi_1(X, a) \times \pi_1(Y, b).$$

Proof. See Propo.11.17 [26]. □

Moreover, we have that

Proposition 12. *The hypercube $I^n = \prod_{i=1}^{n} I \subset \mathbb{R}^n$, where I is an interval, is simply connected.*

Proof. See Example 11.15 [26]. □

Thus, $\pi_1(I^n) = 0$, where 0 is the trivial group (the one composed by a single element, the identity).

Proposition 13. *The fundamental group of the circumference is homomorphic to $\mathbb{Z}$, that is, $\pi_1(S_1) \simeq \mathbb{Z}$.*

Proof. See example 13.16 applying corollary 13.15 [26]. □

Thus, we obtain the following proposition

Proposition 14. *Let C be a cylinder, that is, let $C = S_1 \times [0, 1]$. Then, $\pi_1(C) \simeq \mathbb{Z}$.*

Proof. Using Prop. 11, we have that

$$\pi_1(C) \simeq \pi_1(S_1) \times \pi_1([0, 1]) \simeq \mathbb{Z} \times 0 \simeq \mathbb{Z}$$

(the second equality is given by Prop. 13 and Prop. 12). □

C.2 Other Definitions

Definition 12 (Topological space)**.** A *topology* on a set X is a family $\mathcal{T}$ of subsets of X called *open sets* satisfying the following requirements:

A1 $\emptyset$ and X are open sets;

A2 the union of any number of open sets is an open set;

A3 the intersection of two open sets is an open set

A set X equipped with a topology is called *topological space*.

An example is $\mathbb{R}^m$ with the topology defined by the infinite union and the finite intersection of the subsets of the shape $(a_1, b_1) \times ... \times (a_m, b_m)$.

Definition 13 (Closed sets)**.** Let X be a topological space. A subset $C \subset X$ is called *closed* if $X \setminus C$ is open.

Definition 14 (Continuous function)**.** A map $f : X \to Y$ between two topological spaces is *continuous* if the pre-image

$$f^{-1}(A) = \{x \in X | f(x) \in A\}$$

of any open set $A \subset Y$ is open in X.

Definition 15 (Homeomorphism)**.** A *homeomorphism* is a continuous and bijective map with continuous inverse.

Definition 16 (Closure)**.** Given a subset B in a topological space X, one writes $\overline{B}$ for the intersection of all closed sets containing B:

$$\overline{B} = \cap\{C | B \subset C \subset X,\ C \text{ closed }\}.$$

Equivalently, $\overline{B}$ is the smallest closed set in X containing B and it is called the *closure* of B; its points are said to be *adherent* to B.

Definition 17 (Interior)**.** The *interior* B° of a subset B in X is the union of all open sets contained in B. Equivalently, B° is the largest open set contained in B; its points are called *interior points* in B.

Definition 18 (Boundary)**.** The *boundary* of a subset B in a topological space is the closed set $\partial B = \overline{B} - B^\circ = \overline{B} \cap \overline{X \setminus B}$. hence boundary points are those adherent to B and to $X \setminus B$.

Definition 19 (Neighbourhood)**.** Let X be a topological space and $x \in X$ a point. A subset $U \subset X$ is called a *neighbourhood of* x, if there is an open set V s. t. $x \in V$ and $V \subset U$, i.e. if x is an interior point of U.

Definition 20 (Embedding)**.** An *embedding* is a map that is a homeomorphism onto its image. [21]

Definition 21 (Connection)**.** A topological space X is *connected* if the only subsets that are both open and closed are $\emptyset$ and X. A non-connected topological space is called *disconnected*.

Definition 22 (Path connection)**.** A topological space X is *path(wise)-connected* if, given any two points $x, y \in X$, there is a continuous mapping $\alpha : [0, 1] \to X$ s. t. $\alpha(0) = x$ and $\alpha(1) = y$. Such an α is called *path* from x to y.

Path connected implies connected, on the other hand there are connected spaces which are not path-connected.

Definition 23 (Identification)**.** A continuous and onto (that is, surjective) map $f : X \to Y$ is called an *identification* if the open sets on Y are precisely the subset $A \subset Y$ s.t. $f^{-1}(A)$ is open in X.

Definition 24 (Quotient topology)**.** Let X be a topological space. Let Y be a set. Let $f : X \to Y$. The *quotient topology* is the only topology on Y that makes f an identification and the finest topology for which f is continuous.

Definition 25 (Quotient space)**.** Let X be a topological space. Let $\sim$ be an equivalence relation on X. We denote by $X/\sim$ the set of equivalence classes and by $\pi : X \to X/\sim$, the surjective mapping $x \in X$ to its class $\pi(x) = [x]$. The topological space $X/\sim$ equipped with the quotient topology of π is called the *quotient space*.

Definition 26 (Hausdorff)**.** A topological space is called *Hausdorff space* or *T2 space*, if any two distinct points admit disjoint neighbourhoods.

Definition 27 (Manifold)**.** A topological space M is a *m-dimensional manifold* if M is Hausdorff, every point in M has an open neighbourhood homeomorphic to an open space of $\mathbb{R}^m$ and every connected component is second countable.

If $m = 2$, then M is called a *surface*.
In some books, the third conditions is replaced with *paracompactness*:

Definition 28 (Paracompactness)**.** A space is called *paracompact* if every open cover posses a locally finite open refinement.

In stead, we are interested in a manifold with a boundary. Formally

Definition 29 (Manifold with boundary [25])**.** A topological space M is a m-dimensional *manifold with boundary* if M is Hausdorff, every point in M has an open neighbourhood homeomorphic to an open space of the Euclidean half space $\mathbb{R}^m_+$ where the last component is not negative[15] and every connected component is second countable.

Definition 30 (Covering space)**.** Let X be a connected space. A space E together with a continuous map $p : E \to X$ is a *covering space of* X if every point $x \in X$ is contained in an open set $V \subset X$ whose pre-image $p^{-1}(V)$ is the disjoint union of open sets U_i with the property that $p : U_i \to V$ is a homeomorphism for every i.

The following is a definition from theory of category:

Definition 31 (Commutative diagrams [21])**.** A diagram of maps with the property that any two composition of maps starting at one point in the diagram and ending in another are equal is commutative diagram.

C.3 The Cylinder as a Quotient Space

Consider the subset $I = [0, 1] \times [0, 1] \subset \mathbb{R}^2$.
Defining over I a relation of equivalence $\sim$ as follows: $(x, y) \sim (x', y')$ if $(x, y) = (x', y')$ or if $x = 0$ and $x' = 1$ and $y = y'$ (or vice versa).
It is easy to prove that this is an equivalence relation.
That is, we identify the right edge of I with its left edge.
We consider the quotient space $X/\sim$.
It is possible to prove that $X/\sim$ is homeomorphich to a closed cylinder (i.e. for example the subspace of $\mathbb{R}^3$ defined by $\{x^2 + y^2 = 1,\ 0 \le z \le 1\}$ [26]. Thus, we call $X/\sim$ *closed cylinder*.
It is easy to see that the closed cylinder is a surface (2-manifold) with boundary[16].
Given the Def. 24, there is a continuous map $\pi : I \to I/\sim$.

[15] that is, $\mathbb{R}^m_+ := \{(a_1, ..., a_m) \in \mathbb{R}^> \ :\ a_m \ge 0\}$.

[16] The *open cylinder* is, for example, the subspace of $\mathbb{R}^3$ defined by $\{x^2 + y^2 = 1,\ 0 < z < 1\}$ which is homeomorphic to the quotient of $I' = [0, 1] \times (0, 1)$ with respect to the same relation of equivalence $\sim$.

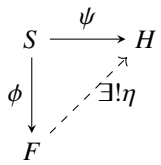


Figure 16: The universal property of free groups.

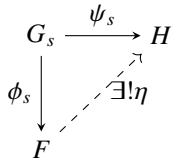


Figure 17: The universal property of free products of a family of groups.

C.4 Fundamental Group and Attaching - Van Kampen's Theorem

The goal of this section is to compute the fundamental group of a space from the fundamental group of its subspaces. For this, we need to talk about free groups

Definition 32 (Free group)**.** Let S be a group. A *free group generated by* S is a group F together with a map $\phi : S \to F$ with the following universal property: for every group H and every map $\psi : S \to H$ there exists a unique homomorphism $\eta : F \to H$ s.t. $\psi = \eta \circ \phi$.

The property is represented in Figure 16.

It can be proved that for any set, free groups exist. In particular, if $S = \emptyset$, its free group is $F = 0$ (the trivial group), while if $|S| = 1$, its free group is isomorphic to $\mathbb{Z}$. Finally, the free group generated by S is formed by all elements of the type $s_1^{a_1} ... s_m^{a_m}$ with $a_i \in \mathbb{Z}$ with $m \in \mathbb{N}$, $s_i \neq s_{i+1}$ and $s_i \in \S\ \forall S$. The operation on these elements and the proof that this is a group can be found in [26].

Definition 33 (Free product)**.** The *free product* of a family of groups $\{G_s \,|s \in S\}$ is a group F together with a homomorphism $\phi_s : G_s \to F$, for every $s \in S$, so that the following universal property holds: for every group H and every family of homomorphism $\psi_s : G_s \to H$, $s \in S$, there is a unique homomorphism $\eta : F \to H$ s.t. $\psi_s = \eta \circ \phi_s$ for every s.

The property is represented in Figure 17.

We denote the free product of the groups G_1 and G_2 with $G_1 * G_2$.

Here we present a simplified version of Van Kampen's theorem with respect to [26].

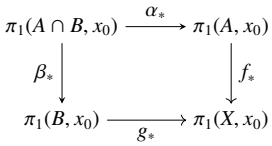
Theorem 6 (Van Kampen)**.** *Lat A, B be open in a space X with $X = A \cup B$; suppose that A, B and $A \cap B$ are path connected, $A \cap B \neq \emptyset$ and the fix a point $x_0 \in A \cap B$. First, the inclusion $A \subset X$, $B \subset X, A \cap B \subset A$, and $A \cap B \subset B$ induce the commutative diagram of homomorphisms depicted in Figure 18a with respect to the homomorphism $\alpha_*, \beta_*, f_*, g_*$. Second, there is a isomorphism between $\dfrac{\pi_1(A, x_0) * \pi_1(B, x_0)}{N}$ and $\pi_1(X, x_0)$ where N is the smallest normal subgroup containing elements of the type $\hat{\alpha}(\gamma) \cdot \hat{(\beta)}(\gamma)^{-1}$ for any $\gamma \in \pi_1(A \cap B, x_0)$, where $\hat{\alpha}$ and $\hat{\beta}$ are defined in Figure 18c using the universal property of free products.*

Note that N is the kernel of h.

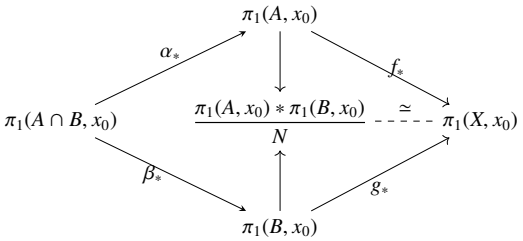
This allows us to compute the fundamental group of a space when we attach another space.

C.5 Fundamental Group of our Dependency Graphs

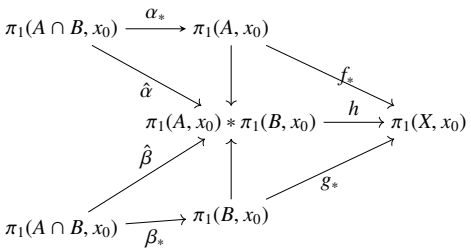
We start observing that the fundamental group of our DG for every gadget is $\mathbb{Z}$.



(a) The first commutative diagram of Van Kampen's theorem.



(b) The second commutative diagram of Van Kampen's theorem.



(c) The definitions of $\hat{\alpha}$ and $\hat{\beta}$ using the universal property of the free product.

Figure 18: Commutative diagrams for Van Kampen's theorem.

For `pRef`, it is easy to see that when we add the lateral surface, we make $x_i \star r_i \star -b_i = 1_G$, and $b_i \star -y_i \star -r_{i-1} = 1_G$. Thus, the fundamental group of $\mathsf{DG}_{\texttt{pRef}} = \mathbb{Z}$.

Now, we consider a complex circuit. We start considering the topology of the underlying circuit. Its fundamental group is $\mathbb{Z}^{\#C-\#O}$ where $\#C$ is the number of copy gates and $\#O$ is the number of output.

When we use our compiler, using the previous argument, we can prove that the fundamental group is $\mathbb{Z}^{\#C-\#O+1}$. We can also see it as a consequence of Proposition 11.

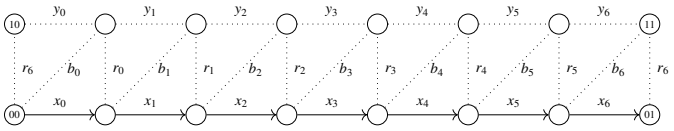
D Additional Figures

E Representation of our Composed Dependency Graphs in $\mathbb{R}^4$

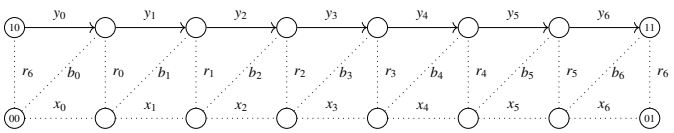
This section is devoted to explain how we can embed our dependency graphs in a vector space.

We explain it for the dependency graph depicted in Figure 6b. We fix the notation. Let us suppose that $(x_i)_{i\in[n]} = \texttt{pRef}((x'_i)_{i\in[n]})$ (with dependency graph DG_1) with $(y_i)_{i\in[n]} = \texttt{pRef}((y'_i)_{i\in[n]})$ (with DG_2), $(z_i)_{i\in[n]} = \mathsf{G}((x_i)_{i\in[n]}, (y_i)_{i\in[n]})$ (thus, G is either the `Add` or the `Mult` gadget, with DG_3) and $(z'_i)_{i\in[n]} = \texttt{pRef}((z_i)_{i\in[n]})$ (with DG_4). The first dependency graph, DG_1, contains a circle formed by the n edges labeled with x_i, the second, DG_2 contains a circle formed by the n edges labeled with y_i, the fourth, DG_4 contains a circle formed by the n edges labeled with z_i, while the third, DG_3 contains a circle of n edges, whose label contains x_i, y_i, and z_i. The composed dependency graph, DG, is the gluing of DG_1, DG_2, DG_4 and DG_3 obtained identifying the x_i-edge of DG_1 with the y_i-edge of DG_2, with the z_i-edge of DG_4 and with the edge of DG_3 whose label contains x_i, y_i, z_i.

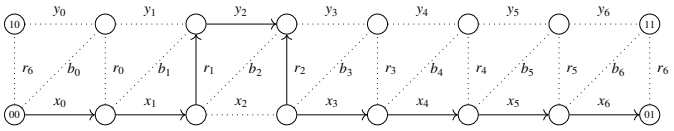
For example, considering Figure 6b, we can see the graph as a composition of three cylinders. However, all cylinder share the loop with the labels T_i. If we see the construction as the surfaces of three cylinders in a three dimensional space it is not possible that three cylinders intersect in the



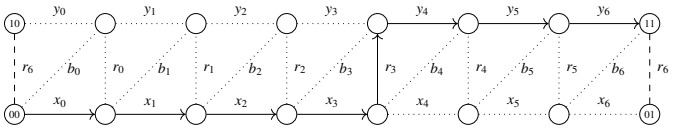
(a) LD_1.



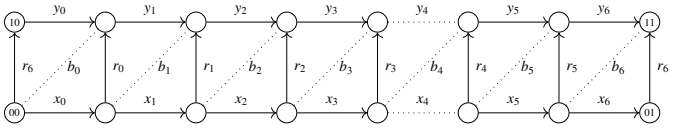
(b) LD_2.



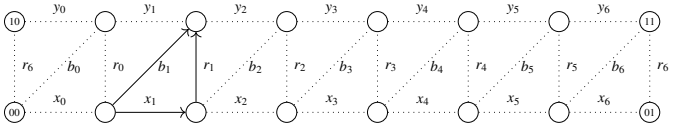
(c) LD_3 where we have chosen $i = 2$.



(d) LD_4 with $i = 3$. LD'_4 is LD_4 with the dashed edge.

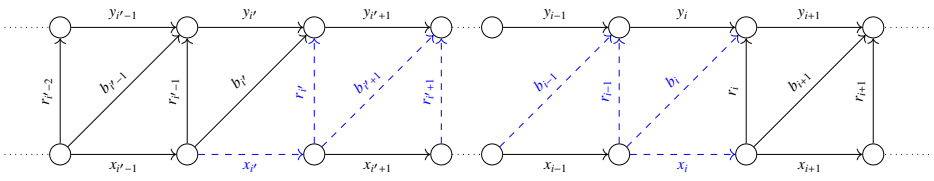


(e) LD_5.

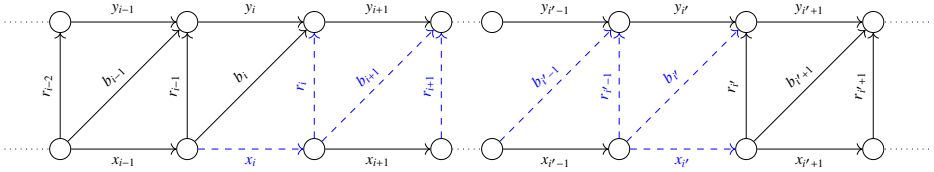


(f) LD_6.

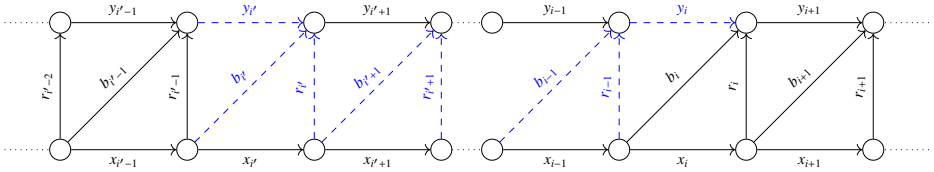
Figure 19: Figure of the example leakage diagrams depicted in Section 4.1. The edges belonging to the leakage diagrams are the solid ones. For simplicity, we have chosen $n = 7$, and we have used the cylindrical projection of $\mathsf{DG}_{\mathtt{pRef}}$ (that is the two edges denoted with r_6 should be glued together, and the nodes denoted with 00 and 10, should be identified with the nodes 00 and 11 respectively).



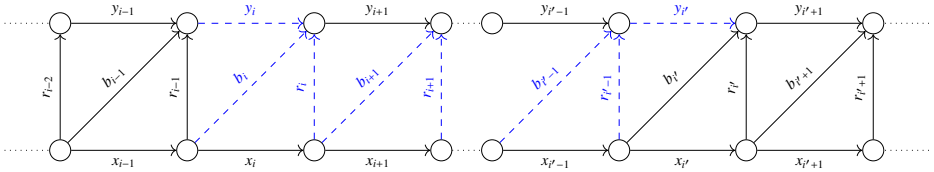
(a) Edges in $BLGap_{i,i'}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.



(b) Edges in $BRGap_{i,i'}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.



(c) Edges in $ULGap_{i,i'}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.



(d) Edges in $URGap_{i,i'}$ (dashed, blue) of the dependency graph of $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$ experiment.

Figure 20: Example of the gaps $BLGap$, $BRGap$, $ULGap$, and $URGap$ in the dependency graph of `pRef`. The blue dashed edges represent $BLGap_{i,i'}$, $BRGap_{i,i'}$, $ULGap_{i,i'}$, and $URGap_{i,i'}$, respectively. Note that $BLGap_{i,i'} = BRGap_{i',i}$, and $ULGap_{i,i'} = URGap_{i',i}$.

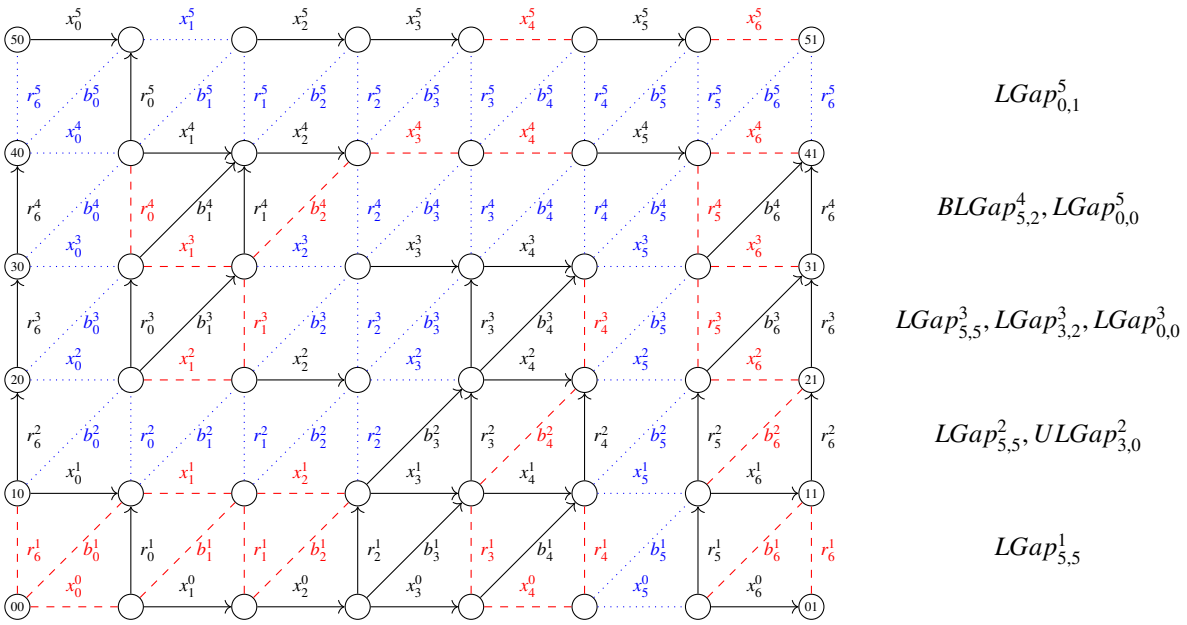


Figure 21: This is the cylindrical projection of the DG of kpRef with $k = 5$, $n = 7$. The leakage diagram is formed by the solid (black edges). A maximal diagram $\mathcal{MAX}$ containing it is formed by the solid (black) edges and the dashed (red) edges. The dotted (blue) edges are the remaining else. There is also the classification of the gaps, according to Proposition 7.

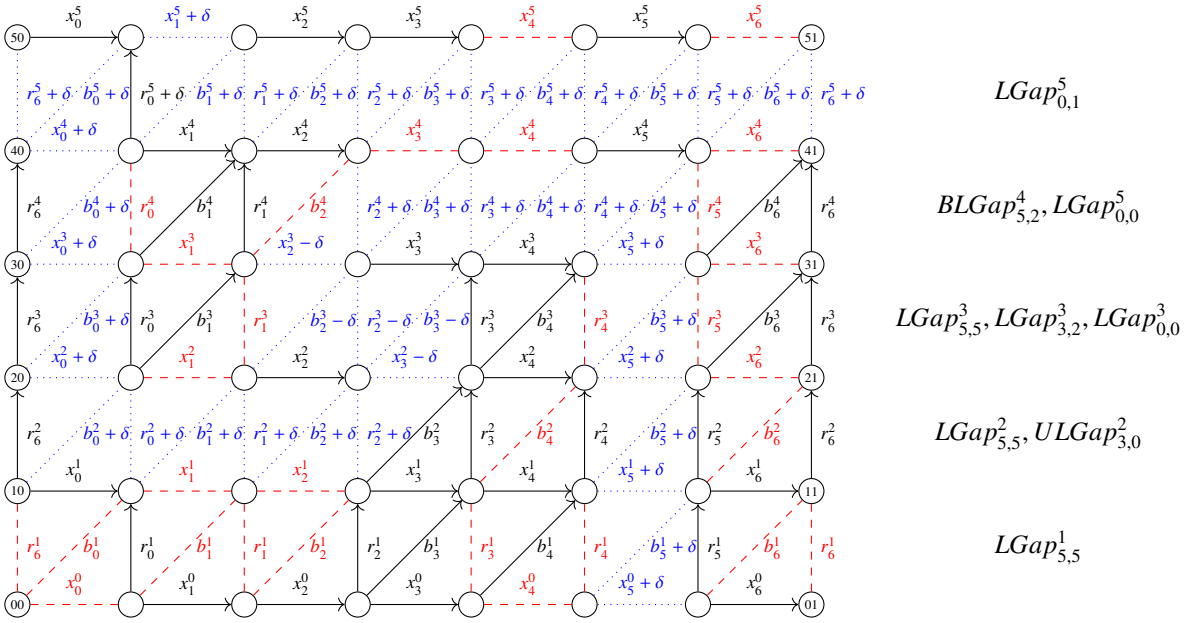


Figure 22: How we modify the values carried by $\mathsf{DG}_{\mathtt{pRef}} \setminus \mathcal{MAX}$ for Figure 21.

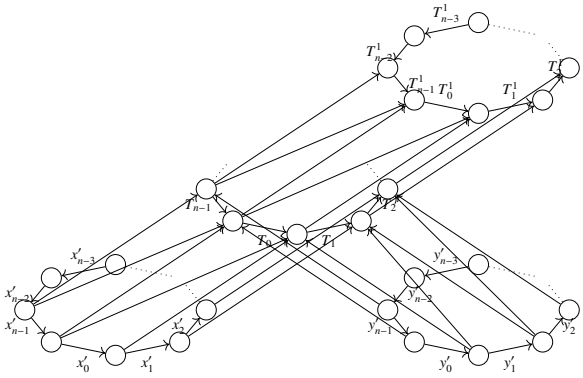


Figure 23: DG of a circuit with no copy gadget. For simplicity we have removed the r_i^j and b_i^j-labels. Its embedding in $\mathbb{R}^4$ is depicted in Appendix E.

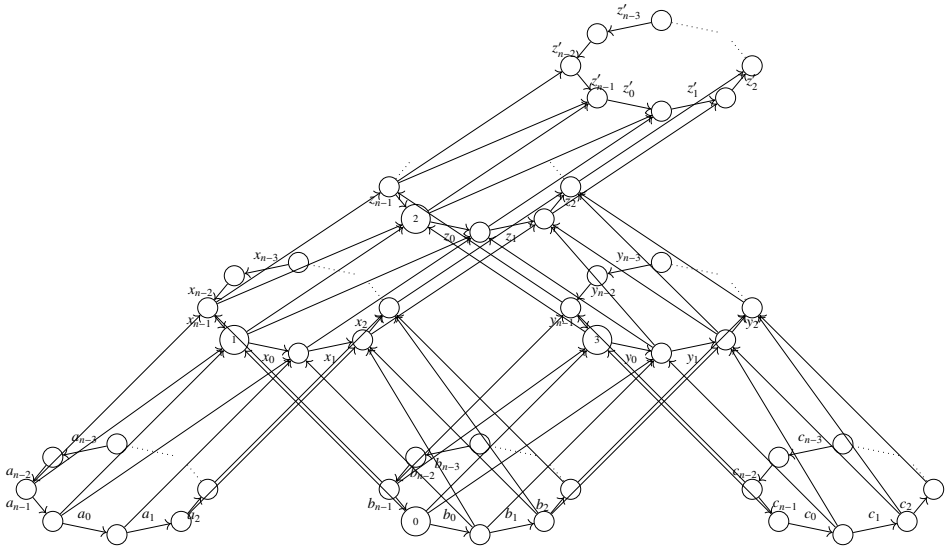


Figure 24: DG of a circuit with copy gadgets. We are computing $(a+b)(b+c)$ and refresh the final output. $x = a + b$, $y = b + c$ and $z = xy$ and $z' = \texttt{pRef}(z)$. Note that the hollow graph contains a loop which is not simply connected or equivalent to an orbit-loop (the one defined by arrows that connects the nodes identified with the numbers 0,1,2,3 and the arrows directly connecting them). For simplicity we have removed the r_i^j and b_i^j-labels and we have not drawn all the r_i^j and b_i^j-edges. Its embedding in $\mathbb{R}^4$ is depicted in Appendix E.

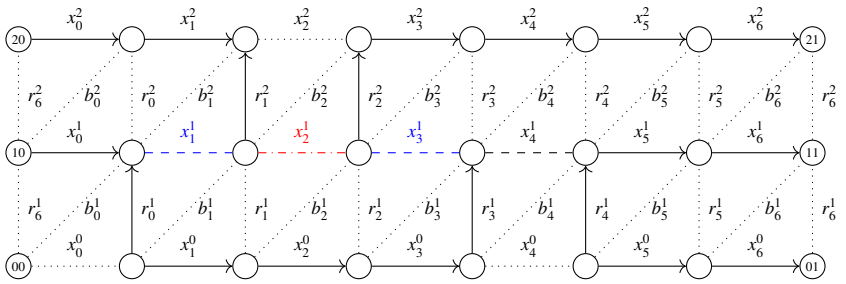


Figure 25: This is an example of 2pRef with $n = 7$. In this example, if we follow the idea of Dziembowski et al. [16], we should modify the dashed edges (blue) (considering edges connected with the first node of the x^1 orbit). But, this is not possible since we should modify only the dash dotted edge (red). This can be easily proved considering that for $\texttt{pRef}^2$, we have clearly a $LGap_{2,2}$. No other gap is possible.

loop labeled with T_i. But if we assume a higher dimensional space, it is possible due to the formula

$$\dim(V + U) = \dim(V) + \dim(U) - \dim(V \cap U)$$

where V, U are two vector subspaces each representing one of the two cylinders on the bottom, and $V + U$ is the space generated by V and U that contains both cylinders sharing the same loop T_i's on the top. In detail, we can see $\mathsf{DG}_1 \subset V$ and $\mathsf{DG}_2 \subset U$ with $\dim(V) = \dim(U) = 3$. If we suppose that $V, U \subset \mathbb{R}^4$, we can have that $\dim(V + U) = 4$, thus $\dim(V \cap U) = 2$. Thus, $\mathsf{DG}_1 \cap \mathsf{DG}_2$ can be contained in a plane, thus, it may contain a circle.

F Random Probing Composability (RPC)

In this section, we want to prove that our results are always better that what can be achieved with IronMask on our gadget. We start stating RPC, then we prove that our security estimation are always better for our refreshing gadget in App., then,

Note that these results hold also for the expandability strategy (as proved in Appendix F.4).

F.1 Random Probing Composition (RPC)

Since it is difficult to bound the RP (random probing)-security of a complete circuits, Belaid et al. [4] proposed a security notion for gadgets that can be composed:

Definition 34 ([4])**.** Let $n, l, m \in \mathbb{N}$. An n-share gadget $\texttt{Gadget} : (\mathbb{F}^n)^l \to (\mathbb{F}^n)^l$ is (t, p, ϵ)-*random probing composable (*RPC*)* for some $t \in \mathbb{N}$ and $p, \epsilon \in [0, 1]$ if there exists a deterministic simulator $\texttt{Sim}_1^{\texttt{Gadget}}$ and a probabilistic simulator $\texttt{Sim}_2^{\texttt{Gadget}}$ s.t for every input $\widehat{\mathbf{x}} \in (\mathbb{F}^n)^l$ and for ever set of collection $J_1 \subset [n],\ ...,\ J_m \subset [n]$ of cardinals $|J_1| \leq t$, ..., $|J_m| \leq t$, the random experiment

$$\mathcal{W} \leftarrow \mathsf{LeakingWires}(\texttt{Gadget}, p)$$

$$\mathbf{I} \leftarrow \texttt{Sim}_1^{\texttt{Gadget}}(\mathcal{W}, \mathbf{J})$$

$$out \leftarrow \texttt{Sim}_2^{\texttt{Gadget}}(\widehat{\mathbf{x}_{|\mathbf{I}}})$$

yields

$$\Pr[(|I_1| > t) \vee ... \vee (|I_l| > t)] \leq \epsilon \text{ and}$$

$$out \overset{id}{=} (\mathsf{AssignWires}(\texttt{Gadget}, \mathcal{W}, \widehat{\mathbf{x}}), \widehat{\mathbf{y}_{|\mathbf{J}}})$$

where $\mathbf{J} = (J_1, ..., J_m)$ and $\widehat{\mathbf{y}} = \texttt{Gadget}(\widehat{\mathbf{x}})$. Let $f : \mathbb{R} \leftarrow \mathbb{R}$.

The gadget Gadget is (t, f)-RPC if it is $(t, p, f(p))$-RPC for every $p \in [0, 1]$.

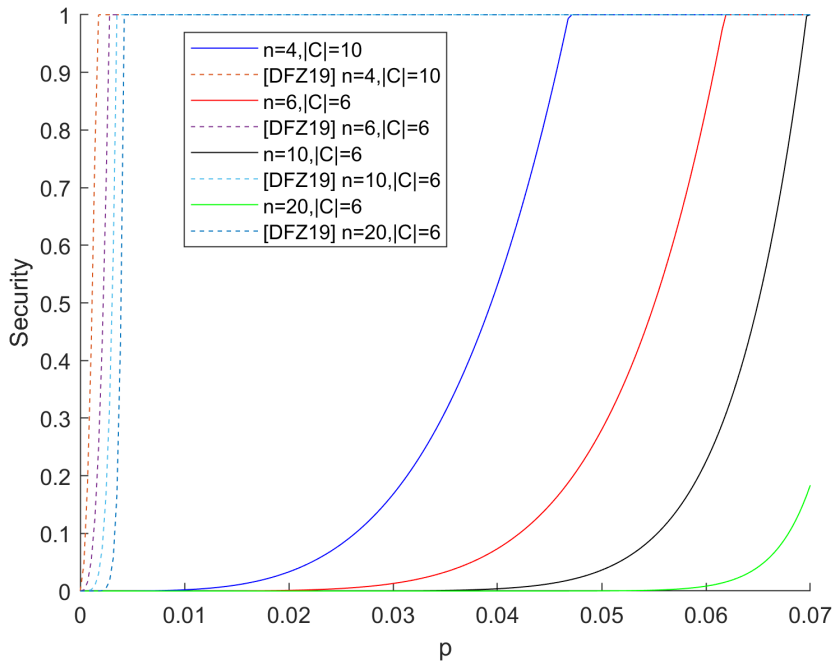


Figure 26: Comparison between the security of our compiler for affine circuits (solid lines) and the one of [16] (dotted lines), with $n = 4, 6, 10, 20$ shares and circuit size $|C| = 6$. The security is defined as with Def. 1. The lines represent the bounds for the security obtained by our Thm.4 and their paper.

That is, the first simulator, receiving as input the output shares leaked and the wires that leak, outputs the input shares that are needed to the second simulator to perfectly simulate the values carried by the wires in $\mathcal{W}$. This definition allows to bound the security of a circuit composed by multiple gadgets, only knowing the RPC-security of each gadget. This is proved in [4] via the following:

Theorem 7. *Let $t, n \in \mathbb{N}$ with $t < n$. Let $p, \epsilon \in [0, 1]$ and let* CC *be a standard compiler with (t, p, ϵ)-*RPC *n-share base gadgets. For every (randomized) arithmetic circuit* C *composed of* |C| *gadgets, the compiled circuit* CC(C) *is $(p, |C|\epsilon)$-random probing secure. Equivalently, the standard circuit compiler* CC *is (p, ϵ)-random probing secure.*

With respect to Thm. [4], we have added $t < n$; otherwise the previous theorem does not make sense, since every n-share gadget is $(n, p, 1)$-RPC $\forall p$ (as proved in [4]).

IronMask [5] can compute the RPC-security of a circuit. Thus, using the previous theorem, Ironmask can prove the security of a circuit if it can compute the RPC security of all gadgets.

F.2 Upper Bounding the RPC-Security of `pRef`

To lower bound the RPC security of `pRef`, we start showing that we can recover the value refreshed via an attack that assumes that only 2 wires are leaked per execution. This attack inspired by Dziembowski et al. [16] is depicted in Fig. It is assumed that in the ith execution the wires carrying the variables b^i_{i-1} and r^i_{n-1}. This leakage reveals the refreshed values. It is easy to see that $b^1_0 + b^2_1 + ... + b^n_{n-1} - r^n_{n-1} - ... - r^1_{n-1}) = x^0_0 + ... + x^n_{n-1} = x$, since that path orbits $\mathsf{DG}_{n\texttt{pRef}}$. We can generalize this attack starting with b^0_q. This attacks needs that for every execution of the `pRef` gadgets the values $(b^j_{q+j \pmod n}, r^j_{q+n-1 \pmod n})$ are leaked.

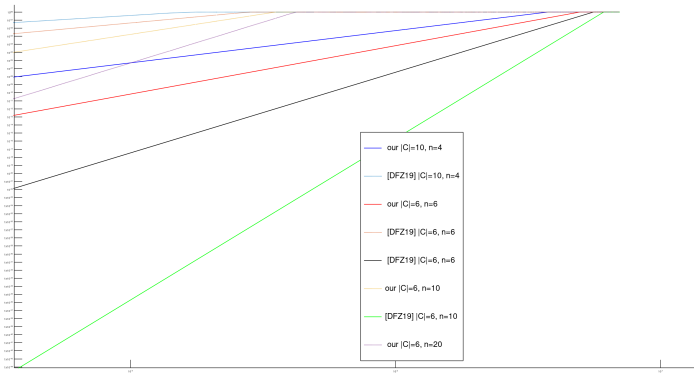


Figure 27: Comparison between the security of our transformation for affine circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale. $|C|$ is the size of the circuit, n, the number of shares.

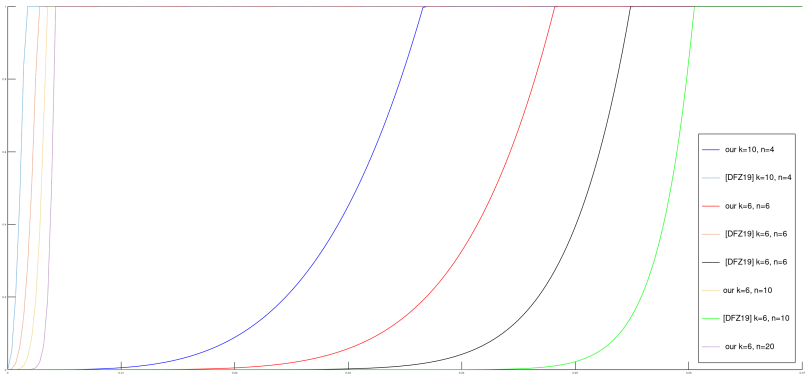


Figure 28: Comparison between the security of our refreshing gadget (solid lines) and the one of [16] (dotted lines). k is the number of times the refreshing gadget is used and n the number of shares.

In particular, we can observe that $\forall t \in \mathbb{N}$, if $(b_j, r_{j+t \pmod n})$ are leaked we cannot simulate the wires leaked and the set of t output variables $(y_{j+1 \pmod n}, \ldots, y_{j+t \pmod n})$ without knowing at least $t+1$ input variables $(x_j, x_{j+t \pmod n})$ since

$$b_j - r_{j+t \pmod n} + \sum_{l=j+1}^{j+t \pmod n} y_{l \pmod n} = \sum_{l=j}^{j+t \pmod n} x_{l \pmod n}.$$

This allows us to lower bound the RPC security of `pRef`

Theorem 8. *Consider* `pRef` *for n shares, $\forall p \in [0,1]$ and $t < n$* `pRef` *is (t, p, ϵ)-*RPC *secure with*

$$\epsilon \geq \sum_{i=1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \left[1 - \left(1 - [1 - (1-p)^3]\right)^i\right].$$

Proof. For the prove we give a bound for the probability ϵ that we need more then t wires for the simulation of the (t, p, ϵ)-RPC secure refresh in the p-random probing model. When we consider the algorithm of `pRef`, we observe that if both b_j and $r_{j+t \pmod n}$ leak (for any j), then, we cannot simulate this leakage with the leakage of any t output shares only using t input shares. This follows immediately from the fact discussed above.

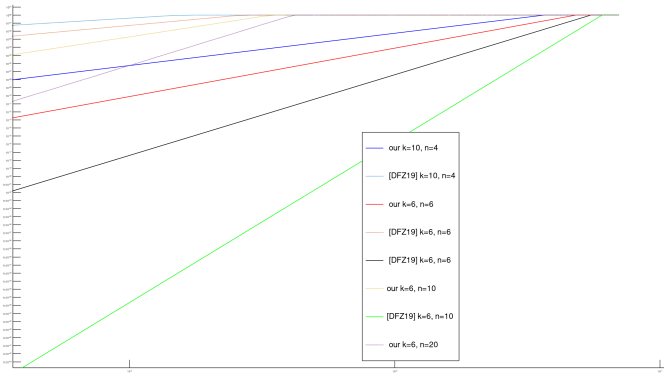


Figure 29: Comparison between the security of our refreshing gadget (solid lines) and the one of [16] (dotted lines) in logarithmic scale. k is the number of times the refreshing gadget is used and n the number of shares.

Thus, $\epsilon > Pr[E]$ where E is the event that there is a couple $(b_j, r_{j+t \pmod n})$ between the leaked variables. To compute $Pr[E]$, we start observing that the probabilities that b_j and $r_{j+t \pmod n}$ leak are independent. After that, we observe that the probability that i different b_j leak is $\binom{n}{i}p^i(1-p)^{n-i}$. Then, we observe that the probability that $r_{j+t \pmod n}$ leaks is $1-(1-p)^3$. Finally, we observe that the probability that at least one of i variable is leaked is $1-(1-\tilde{p})^i$. Summing everything, we obtain the thesis. □ □

For sake of simplicity we give a further bound of the equation in the theorem:

Proposition 15.

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[1-\left(1-[1-(1-p)^3]\right)^i\right] \geq p(1-(1-p)^n).$$

Proof.

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[1-\left(1-[1-(1-p)^3]\right)^i\right] \geq$$

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[1-(1-[1-(1-p)])^i\right] =$$

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[1-(1-p)^i\right] \geq$$

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[1-(1-p)\right] =$$

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}p =$$

$$p\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i} = p(1-(1-p)^n)$$

The first and third inequalities are due to the fact that $(1-p)^i \leq (1-p)\ \forall i \geq 1$. The last equality is due to the fact that

$$\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i} = \left(\sum_{i=0}^{n}\binom{n}{i}p^i(1-p)^{n-i}\right) - (1-p)^n = 1-(1-p)^n$$

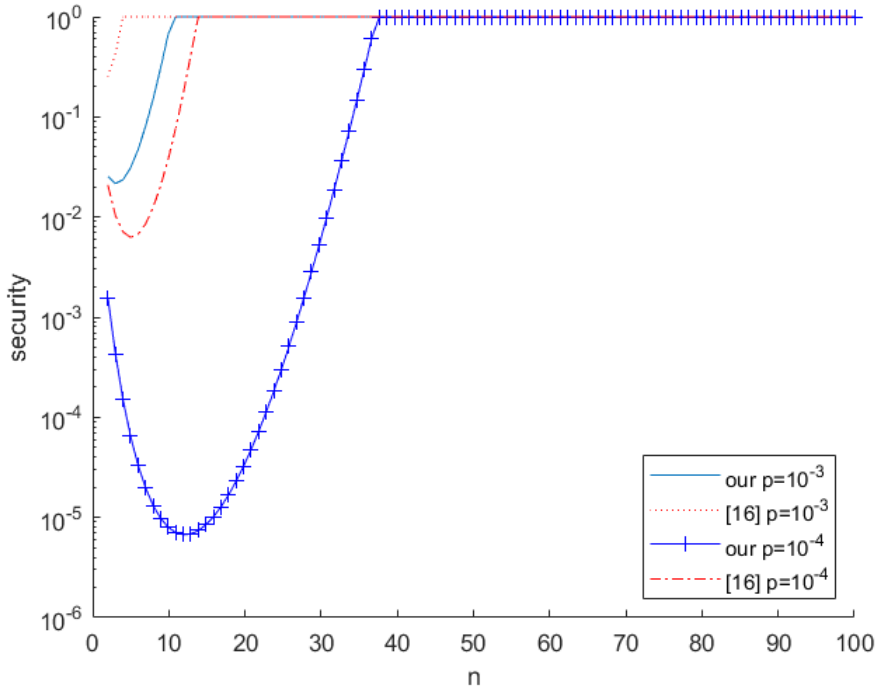


Figure 30: Comparison between the security of our masked circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale, for two fixed $p = 10^{-3}, 10^{-4}$ where the number of shares n is between 2 and 100. $p = O(1/n^2)$ and a single gate ($k = 1$). For general circuits the results must be multiplied with the number of gates of the circuit. We remember that in our choice of p we have to consider the factor $32np$ in [16].

since $\binom{n}{i}p^i(1-p)^{n-i}$ is the probability that in a binomial distribution with parameters n and p, we get i. □

It is easy to observe that around 0 the previous bound (for RPC) is asymptotical to np^2 (remember that $(1-p)^n \sim 1 - np + o(p)$), for a security for k executions of $\texttt{pRef} \sim knp^2$ (via Thm. 7). In stead, the bound of $\texttt{pRef}$, thanks to Thm. 3 is $\leq (k+1)(9p)^n$. It seems clear that (at least for $n > 3$) we should be better. [We have used the same approximation for both bounds, thus, the analysis of Section 5.3 do not influence our comparison]. This is proved in the following

Proposition 16. *Let $n \geq 2$, $p \leq 10^{-1}$, $(k+1)(9p)^n \leq kp(1-(1-p)^n)$ if $p \leq min(10^{-1}, 1-\alpha)$ with*

$$\alpha = \sqrt[n-1]{\frac{(k+1)(n-1)9^2}{kn}\left(\frac{9}{10}\right)^{n-2}}.$$

Proof. First, we observe that for $p = 0$ the claim holds, since both sides are equal to 0. In stead, for $p > 0$, the claim is equivalent to ask that $(k+1)9^n p^{n-1} \leq k(1-(1-p)^n)$ Let $f(p) := (k+1)9^n p^{n-1} - k(1-(1-p)^n)$. Since $f(0) = 0$, to prove the claim it is enough to prove that $f'(p) \leq 0$. But

$$f'(p) = (k+1)(n-1)9^n p^{n-2} - kn(1-p)^{n-1}$$

Thus, we have to prove that

$$kn(1-p)^{n-1} \geq (k+1)(n-1)9^n p^{n-2}$$

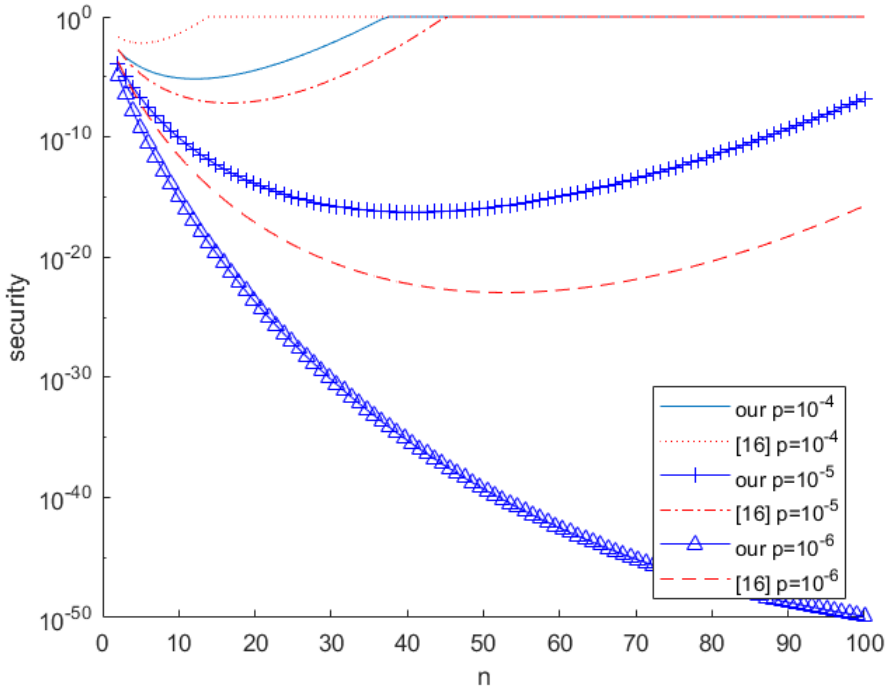


Figure 31: Comparison between the security of our masked circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale, for two fixed $p = 10^{-4}, 10^{-5}$, and 10^{-6} where the number of shares n is between 2 and 100, and a single gate ($k = 1$). For general circuits the results must be multiplied with the number of gates of the circuit. We remember that in our choice of p we have to consider the factor $32np$ in [16].

we observe that since we have assumed that $p \leq 10^{-1}$, it is enough to find for which p the following holds:

$$kn(1-p)^{n-1} \geq (k+1)(n-1)9^n\left(\frac{1}{10}\right)^{n-2} \Leftrightarrow (1-p)^{n-1} \geq \frac{(k+1)(n-1)9^2}{kn}\left(\frac{9}{10}\right)^{n-2} \Leftrightarrow$$

$$1-p \geq \sqrt[n-1]{\frac{(k+1)(n-1)9^2}{kn}\left(\frac{9}{10}\right)^{n-2}} \Leftrightarrow p \leq 1 - \sqrt[n-1]{\frac{(k+1)(n-1)9^2}{kn}\left(\frac{9}{10}\right)^{n-2}}$$

which is the claim. □

Note that for $n \leftarrow \infty$, then $\alpha \leftarrow 10^{-1}$. But we asked that $p \leq 10$. Finally note that in our approximation, we have done many approximations, thus, in practice, we believe, our bounds are much better. We do a function comparison of these two bounds for some fixed n and k in Fig. from which it is possible to see how much we are better.

Anyway, we compute α for some n and k in Tab..

F.3 Upper Bounding the RPC-security of the `Add`, `Mult` and Copy Gadget

In this section we analyze our compiler. Therefore, we give the security of our parallel gadgets. The addition gadget is clearly $(t, p, 0)$-RPC secure, since we have only input and output wires. But the masked circuit generated by our compiler is dominated by the RPC-bound of the `pRef` gadget. Note that due to the continuous nature of leakage covered by the random probing model it is clear that those refreshes are unavoidable. In stead, for the multiplication gadget, it is easy to see that

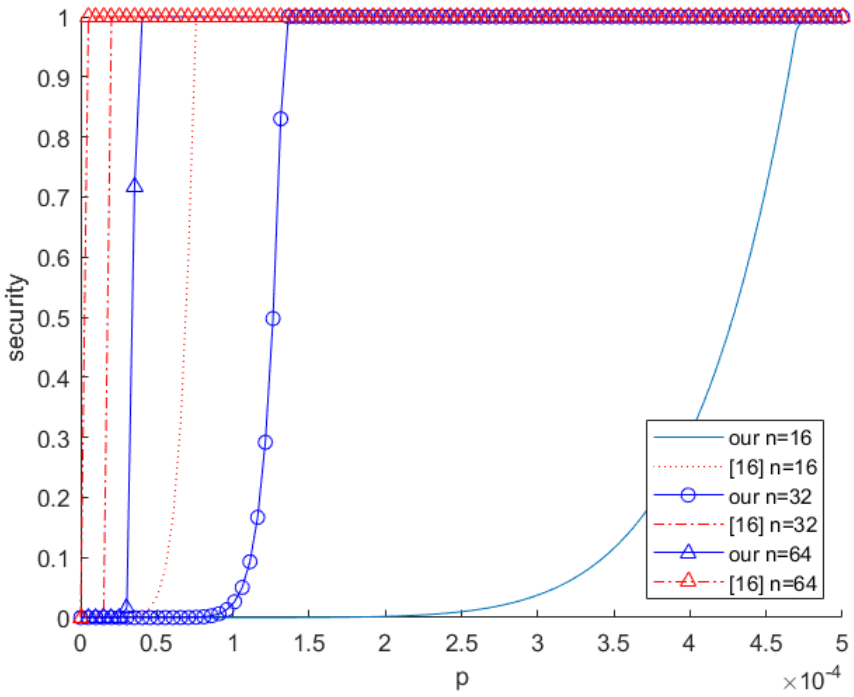


Figure 32: Comparison between the security of our masked circuits (solid lines) and the one of [16] (dotted lines) in logarithmic scale,in function of p,, for three fixed number of shares $n = 16, 32, 64$. $p = O(1/n^2)$ and a single gate ($k = 1$). For general circuits the results must be multiplied with the number of gates of the circuit. We remember that in our choice of p we have to consider the factor $32np$ in [16].

F.4 Random Probing Expandability (RPE)

Since it is difficult to assess the RP security of a complex circuit, [1] proposed a *gadget expansion strategy*.

Expanded strategy. The idea is to compile a circuit replacing the gates with gadgets and then compile the new circuit replacing the gates of the compiled circuit with the gadgets and iterate. Suppose that we have 3 n-shares gadgets: `Add`, `Copy` and `Mult` and denote CC the standard circuit compiler for these gadgets. We obtain three new n^2-shares gadgets by simply applying CC to each gadget: $\texttt{Add}^{(2)} = \mathsf{CC}(\texttt{Add})$, $\texttt{Mult}^{(2)} = \mathsf{CC}(\texttt{Mult})$, $\texttt{Copy}^{(2)} = \mathsf{CC}(\texttt{Copy})$. This process can be iterated an arbitrary number of times, say N to an input circuit C:

$$\mathsf{C} \overset{\mathsf{CC}}{\leftarrow} \widehat{\mathsf{C}}_1 \overset{\mathsf{CC}}{\leftarrow} \ldots \overset{\mathsf{CC}}{\leftarrow} \widehat{\mathsf{C}}_k.$$

Thus, we have an *expanding circuit compiler*. Formally,

Definition 35 ([4])**.** Let C be the standard circuit compiler with sharing order n and base gadget `Add`, `Mult` and `Copy`. The *expanding circuit compiler* with *expansion level* k and base compiler CC is the circuit compiler $(\mathsf{C}^{(k)}, \texttt{Enc}^{(k)}, \texttt{Dec}^{(k)})$ satisfying the following:

- The input encoding $\texttt{Enc}^{(k)}$ is an (n^k)-linear encoding.
- The output decoding $\texttt{Dec}^{(k)}$ is the (n^k)-linear decoding mapping.
- The circuit compilation is defined as

$$\mathsf{CC}^{(k)}(\cdot) = \underbrace{\mathsf{CC} \circ \mathsf{CC} \circ \ldots \circ \mathsf{CC}}_{k \text{ times}}(\cdot)$$

To prove that this strategy is good to build a secure algorithm, Belaid et al. [4] defined this security notion:

Definition 36 ([4])**.** Let $n \in \mathbb{N}$. An n-share gadget $\texttt{Gadget} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^n$ is (t, f)-*random probing expandable (*RPE*)* if there exists a deterministic simulator $\texttt{Sim}_1^{\texttt{Gadget}}$ and a probabilistic simulator $\texttt{Sim}_2^{\texttt{Gadget}}$ s.t for every input $(\widehat{x}, \widehat{y}) \in \mathbb{F}^n \times \mathbb{F}^n$ and for ever set of collection $J \subset [n]$ and for every $p \in [0, 1]$, the random experiment

$$\mathcal{W} \leftarrow \mathsf{LeakingWires}(\texttt{Gadget}, p)$$

$$(I_1, I_2, J') \leftarrow \texttt{Sim}_1^{\texttt{Gadget}}(\mathcal{W}, J)$$

$$out \leftarrow \texttt{Sim}_2^{\texttt{Gadget}}(\mathcal{W}, J', \widehat{x}_{|I_1}, \widehat{y}_{|I_2})$$

ensures that

1) the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr[\mathcal{F}_1] = \Pr[\mathcal{F}_1] = \epsilon \text{ and } \Pr[\mathcal{F}_1 \wedge \mathcal{F}_2] = \epsilon^2.$$

with $\epsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent), 2) J' is such that $J' = J$ if $|J'| \leq t$ and $J' \subset [n]$ with $|J'| = n - 1$ otherwise, 3) the output distribution is

$$out \overset{id}{=} (\mathsf{AssignWires}(\texttt{Gadget}, \mathcal{W}, (\widehat{x}, \widehat{y})), \widehat{z}_{|J'})$$

where $\widehat{z} = \texttt{Gadget}(\widehat{x}, \widehat{y})$.

This definition can be used to give the security of an expanding compiler:

Theorem 9 ([4])**.** *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \leftarrow \mathbb{R}$. Let* Add, Mult *and* Copy *be n-shares gadgets for the addition, multiplication and copy on $\mathbb{F}$. Let* CC *be the standard circuit compiler with sharing order n and base gadgets* Add, Mult *and* Copy*. Let $\mathsf{CC}^{(k)}$ be the expanding circuit compiler with base compiler* C*. If the base gadget* Add, Mult *and* Copy *are (t, f)-*RPE*, then $\mathsf{CC}^{(k)}$ is $(p, 2f^{(k)}(p))$-random probing secure.*

RPE and RPC are related via this proposition:

Proposition 17 ([4])**.** *Let $f : \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$. Let* Gadget *be a n-share gadget. If* Gadget *is (t, f)-*RPE *then* Gadget *is (t, f')-*RPC *with $f'(\cdot) = 2f(\cdot)$.*

Thus, the previous results are also extended to the expandability strategy.

RPE for the other gadgets. Already in the paper when RPE has been introduced, [4], it has been proved that the classical addition gadget (which is ours) cannot achieve RPE if at least a wire leaks. In addition also for the multiplication gadget,