IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 420–459. DOI:10.46586/tches.v2023.i4.420-459

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 $O(1/\sqrt{n})$ 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 [KJJ99], cache accesses [BM06], acoustic signals [GST14], or timing [Koc96].

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.

Received: 2023-04-15 Accepted: 2023-06-15 Published: 2023-08-31

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) [ISW03]. 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 [Wal01, CFG+10]. To address this problem, Prouff and Rivain introduced the *noisy leakage model* [PR13]. 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. [DDF19] 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.[DFS15] 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. [PGMP19]. 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 [ADF16, BCP+20, BRT21a, CFOS21, DFZ19]. 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. [DFZ19], 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, [BCP+20] with $O(n^{2.4})$-randomness required, while ours needs $O(n)$-randomness to refresh the inputs) or require more noise (namely [DFZ19], $O(\sqrt{p}^n)$).

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 asymptotic analysis of the ISW. Concretely, we can prove the security up to $p = O(\frac{1}{\sqrt{n}})$, instead of $p = O(\frac{1}{n})$ as in [ISW03, DFZ19, BCP+20]. We believe that this is a worthwhile goal due to the following two reasons. First, it was shown [CFOS21] 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 [CPRR13, BDF+17] 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 [MSJ12].

1.2 Related Work

Proof techniques. Analyzing leakage resilience of circuits via graphs was already proposed in [RBN+15] 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 [NRR06] and the Trichina gate [Tri03]. 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 [RBN+15], the work of [DFZ19] 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 [DFZ19], 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 [ADF16, BCP+20, BRT21a, BIS19]. 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. [CFOS21] and tighten the analysis. Concretely, in [CFOS21], 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 [BCP+20, BRT21a, BMRT22a] 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. [ADF16] 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. [GJR18] 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 [GPRV21] 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 [GM10] 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 [GPRV21] 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. [AIS18] proposed a *modular approach* how to compose a secure compiler multiple times. Finally, several follow-up works further improved this approach [BCP+20, BRT21b, BMRT22b]. 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 [CFOS21]. For this reason, we try to close the gap between practice and theory and give a tighter security analysis in the random probing model. In detail, we prove that the ISW multiplication tolerates a leakage rate $O(1/\sqrt{n})$ instead of $O(1/n)$ [GPRV21]. For this reason, we re-consider the ISW-based compiler of [DFZ19] and improve its compiler regarding running time and security.

2 Background

Notations. Let $[n] := \{0, 1, \dots n-1\}$. 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 $(\texttt{Enc}, \texttt{Dec})$ 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 $\texttt{Enc}$ is a probabilistic algorithm that takes as input $x \in \mathbb{F}$ and outputs an n-tuple $(x_0, ..., x_{n-1}) = \texttt{Enc}(x)$, where n is the *masking order*. The *decoding* scheme $\texttt{Dec}$ takes as input an n-tuple $(x_0, ..., x_{n-1})$ and outputs $x = \texttt{Dec}(x_0, ..., x_{n-1})$. For correctness, we want that for any $x \in \mathbb{F}$ it holds that $\texttt{Dec}(\texttt{Enc}(x)) = x$. For security, we need that any subset of $n-1$ shares of $\texttt{Enc}(x)$ are independent of x. We use *arithmetic encoding*. $\texttt{Enc}(x)$ provides a randomized n-tuple $x_0, ..., x_{n-1}$ s.t. $\sum_{i=0}^{n-1} x_i = x$, and $\texttt{Dec}(x_0, ..., x_{n-1}) = \sum_{i=0}^{n-1} x_i$. 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 $\forall \mathbf{x} = (x^0, ..., x^{l-1}) \in \mathbb{F}^l$,

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

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

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 $\texttt{Dec}((x_i)_{i\in[n]}) = \texttt{Dec}((y_i)_{i\in[n]})$. In this work, we consider refreshing schemes using a linear number of random gates ⊛.

In Figure 1a the simple refresh $\texttt{sRef}$ of [DFZ19] is depicted, initially introduced in [RP10]. 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 $\texttt{sRef}$ that we call $\texttt{pRef}$; see Figure 1b. This gadget was initially introduced in [BDF+17] and has the key feature that it is highly parallelizable. $\texttt{pRef}$ 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* $\texttt{CC}$. $\texttt{CC}$ works as follows: First, $\texttt{CC}$ 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 $\texttt{CC}$ 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

[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.

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 [ISW03] 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 [ISW03], 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 [ISW03], 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 [DFZ19])**.** 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 $\mathsf{Leak}(\widehat{\mathsf{C}}, \mathbf{x}, p)$

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 [DFZ19]. 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 [DFZ19])**.** 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 [DFZ19] to prove their compiler security.

Corollary 1 ([DFZ19])**.** *Let* $\widehat{\mathsf{C}}$ *be the sound transformation of a circuit* C *via Dziembowski et al.'s compiler [DFZ19]. If*

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

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.

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

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 [RP10].

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

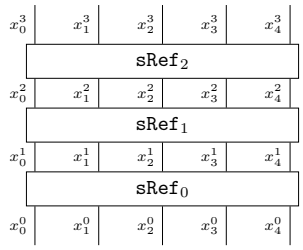
for $j \in \{0 \dots n-1\}$
  $r_j \leftarrow_{\$} \mathbb{F}$
for $j \in \{0 \dots n-1\}$
  $b_j \leftarrow x_j + r_j$
  $y_j \leftarrow b_j - r_{j-1} \pmod n$
return $(y_i)_{i\in[n]}$

(b) Parallel refresh gadget [BDF+17].

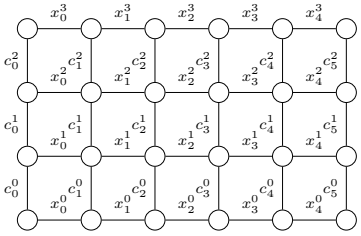
Figure 1: Refresh gadgets with linear random complexity.

Original leakage diagram. Dziembowski et al. [DFZ19] introduced the concept of *leakage diagrams*. They represent all the variables on a graph and those variables whose values are leaked in a sub-graph 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 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_j^i of the random values used during that execution. Since $c_{j+1}^i = c_j^i + r_j^i$, the variable r_j^i is represented by both the edges c_j^i and c_{j+1}^i. They also add the edges corresponding to the variable c_n^i. These c_n^i are defined similarly to the others as follows $c_n^i = c_{n-1}^i + x_{n-1}^i - x_{n-1}^{i-1}$. Thus, it always holds $c_n^i = 0$ because $c_n^i = c_{n-1}^i + x_{n-1}^i - x_{n-1}^{i-1} = c_{n-1}^i + (x_{n-1}^{i-1} - c_{n-1}^i) - x_{n-1}^{i-1} = 0$.

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_0^i and c_n^i since they are always equal to 0. Thus their values are always known by the adversary). Further, Dziembowski et al. [DFZ19] proved that the leakage can be simulated independently from the input $x = \texttt{Dec}((x_i^0)_{i\in[n]})$ 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.



(a) Simple refresh gadget [DFZ19].

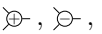


(b) The graph of the simple refresh used for the leakage diagram [DFZ19].

Figure 2: Refresh gadgets with linear random complexity.

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_ib_j of all shares. As in the ISW multiplication, this results in n^2 products a_ib_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 [DFZ19], which has an asymptotic depth of n.

Compiler $\mathtt{CC}^p$. The parallel compiler $\mathtt{CC}^p$ 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 $\mathtt{CC}^p$ 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 $\widehat{\mathsf{C}} \leftarrow \mathtt{CC}^p(\mathsf{C})$. We start by showing the soundness of the compiler $\mathtt{CC}^p$ in the following corollary.

Corollary 2. *Let $\mathsf{C} : \mathbb{F}^s \to \mathbb{F}^t$ be an arbitrary circuit and $\widehat{\mathsf{C}} \leftarrow \mathtt{CC}^p(\mathsf{C})$. 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 $\mathsf{C}(x^0, \ldots, x^{s-1}) = (\mathtt{Dec}((y_i^0)_{i\in[n]}), \ldots, \mathtt{Dec}((y_i^{t-1})_{i\in[n]}))$.

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

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 $\oplus$ and $\ominus$ 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 $\mathbb{C}$.

$\texttt{Const}_a$

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

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

$\texttt{Rand}()$

for $i \in \{0 \dots n-1\}$
  $r_j \leftarrow_\$ \mathbb{F}$
return $(r_i)_{i\in[n]}$

(d) The random gadget `Rand` for $\mathbb{R}$.

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

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 $\odot$.

Figure 3: Parallel gadgets.

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 [DFZ19]. 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,* $\texttt{CC}^p$*. If*

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

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

Proof. This proof is similar to the one in [DFZ19]. 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].

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

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 $RandR(\texttt{pRef})$ (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 $RandR(\texttt{pRef})$ 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.

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 $\mathbf{x'}$. 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 $RandR(\texttt{pRef})$, 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

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

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

[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.

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 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,

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

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

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

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_ib_j and a_jb_i, $j \in [n]$. Note the monomial a_ib_j (and a_jb_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 [ISW03, DFZ19]. 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 [CFOS21], 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 $\texttt{pRef}$. 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 $\texttt{Add}$ (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 $\texttt{pRef}$ (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 $\texttt{Add}$ 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.

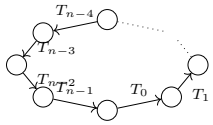
Definition 4 (Attaching)**.** Let DG_1 and DG_2 two disjoint dependency graphs, and f a function as described above mapping some edges of DG_1 to edges in DG_2. The composed graph is

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

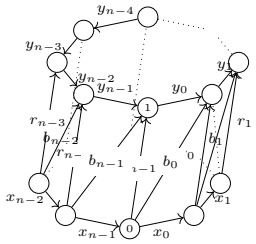
where $\sim$ is the smallest equivalence relation with $x \sim f(x)$.

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,

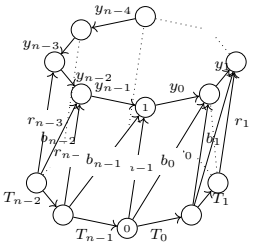


(a) Dependency graph of `Enc`, `Dec`, `Copy`, `Add`, or `Mult` with labels T_i defined in Section 3.1.

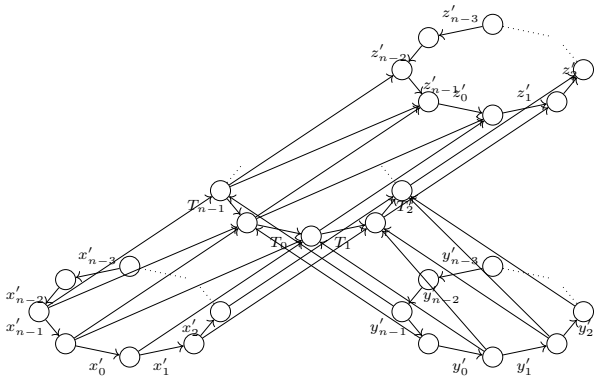


(b) The dependency graph for $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}((x_i)_{i\in[n]})$.

Figure 5: Dependency graphs of the gadgets.



(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.

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*, $\mathcal{LD}(\mathcal{L}_p(\widehat{\mathsf{C}}), \mathsf{C})$, 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 $\mathtt{pRef}$ gadget refreshing an encoding $(x_i)_{i\in[n]} \leftarrow \mathtt{Enc}(x)$ of a secret $x \in \mathbb{F}$

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

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

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

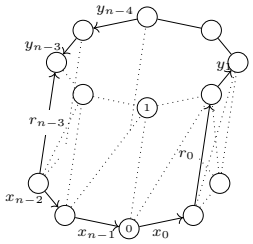
4) Let $LD_4 = (x_0, ..., x_i, r_i, y_{i+1}, ..., y_{n-1})$. We observe that if instead of r_{n-1}, we have $r'_{n-1} = r_{n-1} + x' - x$, these values come from a refreshing of x'. In fact

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

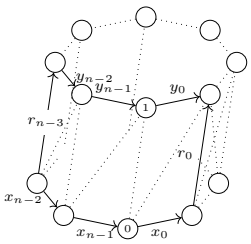
since $r_{j-1} + y_j = x_j + r_j$. Thus, LD_4 does not reveal x because the leakage is shiftable to x'.

Note that, $LD'_4 = LD_4 \cup \{r_{n-1}\}$ would reveal x similarly to B3).

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 $\mathcal{L}(\mathtt{pRef}((x_i)_{i\in[n]}, \mathbf{r}), p)$, while $LD_5 \cup \{x_i + x' - x\}$ comes from $\mathcal{L}(\mathtt{pRef}((x'_i)_{i\in[n]}, \mathbf{r}), p)$, 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.



(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.

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 $LD' \subset LD$ 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

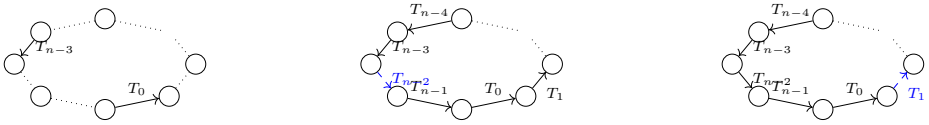
$$\mathcal{M}_i = \{T_0, ...T_{i-1}, T_{i+1}, ..., T_{n-1}\} = \mathsf{DG} \setminus \{T_i\}$$

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



(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.

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. □

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 $\mathsf{DG}_{\mathtt{pRef}}$ $path$ 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 $\{b_i, r_i\} \nsubseteq LD$, 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 $r_i \in LD$. ($\mathcal{M}_{i,i}^{left}$)* 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 $\mathcal{M}_{i,i}^{right}$)* 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

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

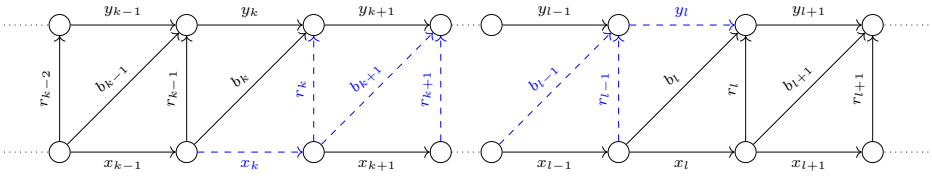
$$x_0, ..., x_{i-1}, b_i, y_{i+1}, ..., y_l, b_l, x_l, ..., x_{n+1}, ..., x_{n-1} \quad \text{for } l > i$$

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

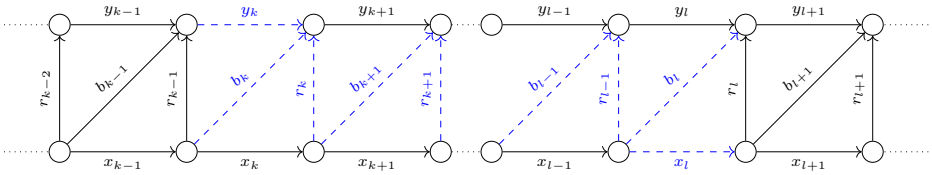
The other cases (which are done in a similar way) can be found in the full version. □

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

[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 $\mathcal{M}^{\text{left}}_{k,l} = \mathsf{DG} \setminus LGap_{k,l}$.

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.

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 $\mathsf{Leak}(\mathsf{G}, \mathbf{x}\text{'}, p)$. □

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 $\mathsf{Leak}(\mathtt{Add}, (a, b), p)$ to the output $\mathsf{Leak}(\mathtt{Add}, (a', b'), p)$. 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 [DFZ19]:

In [DFZ19] is the proof for all values except for the $z_{l,i,j}$s. The only difference from our `Mult` from that in [DFZ19], 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. [DFZ19], 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*

$$\tilde{p} = 2(1 - (1 - \sqrt{3p})^{8n}).$$

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 [DFZ19], 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 $\mathsf{Leak}(\texttt{Mult}, (x, y), p)$-experiment.

We prove this fact by doing a proof very similar to the one in [DFZ19] (which is inspired by the original proof in [ISW03]). 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 $1 - 3p$.

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 [DFZ19], where we have not used the approximation $1 - (1 - p)^{8n} \leq 8np$. A detailed discussion is given in the full version.

Thus, the probability that all edges belong to LD is bounded by $[2(1-(1-\sqrt{3p})^{8n})]^n$.

This proves the claim of the theorem. □

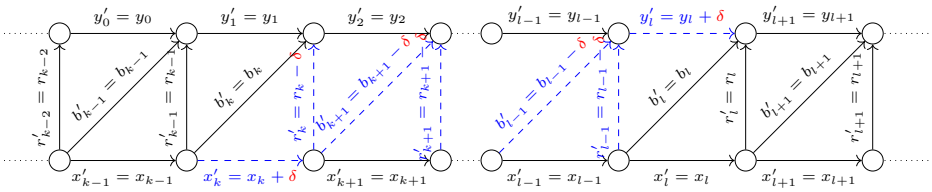
Note that we do not use the approximation $2(1 - (1 - \sqrt{3p})^{8n}) \leq 16n\sqrt{3p}$ because it is not tight and makes the security much worse.

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:

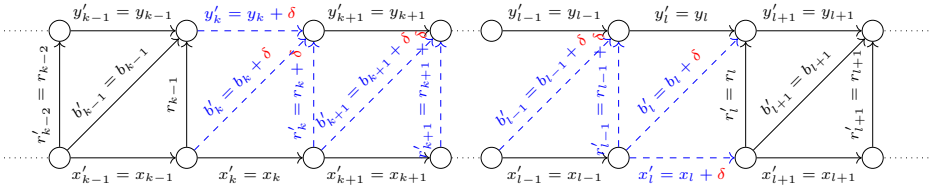
Proposition 6. *An outcome L of the $\mathsf{Leak}(\texttt{pRef}, x, p)$ experiment, is shiftable to x', if the corresponding leakage diagram, does not orbit.*

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 $\mathsf{Leak}(\texttt{pRef}, x, p)$ to an output of $\mathsf{Leak}(\texttt{pRef}, x', p)$ 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.



(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).

(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 $\mathtt{pRef}$ 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

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

The same holds for the shares of y.
Now, let us consider $\mathtt{pRef}$ 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 $\mathtt{pRef}$ 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 $\mathtt{pRef}$.

Theorem 2. *Let* $\mathtt{pRef}$ *be the refreshing gadget defined in Figure 1b. Then,* $\mathtt{pRef}$ *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

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

First, we observe that every edge of DG(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[6] 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 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[7] keeping the direction of the orbit [8], 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.

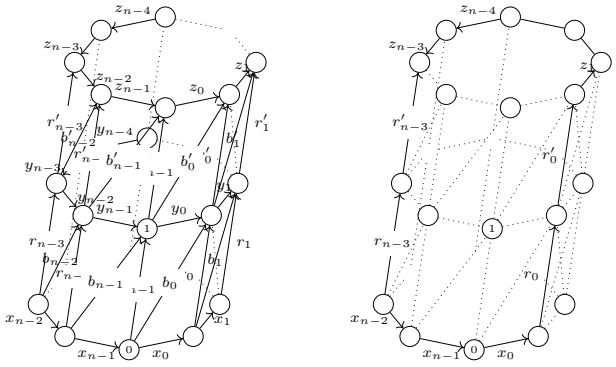
Instead, if y is on the x_i^1-edges orbits, thus, it is node $node_j^1$ for a $j \in [n]$, the source node of the x_j^1-edge. Thus, if the path to $node_0^0$ keeping the direction of the orbit has to turn right, it must take either the edge x_j^1 or r_{j-1}. In fact, the first path needs $1+n-j$ edges (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}.

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

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

[8] 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.

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.

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 $(x_i^j)_{i\in[n]} \leftarrow \texttt{pRef}((x_i^{j-1})_{i\in[n]})$ the jth* pRef *of k*pRef*. Further, let DG^j be the sub-graph of $\mathsf{DG}_{\texttt{pRef}}$ containing the variables used in $\texttt{pRef}^j$ 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 $k\texttt{pRef}$ looks when we cut it into the sub-graphs LD^j describing the individual $(x_i^j)_{i\in[n]} \leftarrow \texttt{pRef}((x_i^{j-1})_{i\in[n]})$. 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 $k\texttt{pRef}$. 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 $\texttt{pRef}^2$ until $\texttt{pRef}^{k-1}$. 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 $k\texttt{pRef}$.

Proposition 8. *An outcome L of the experiment $\mathsf{Leak}(k\texttt{pRef}, x, p)$ 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 $(y_i)_{i\in[n]} \leftarrow \texttt{pRef}^1((x_i)_{i\in[n]})$, and $(z_i)_{i\in[n]} \leftarrow \texttt{pRef}^2((y_i)_{i\in[n]})$. If x_i and y_j are shiftable in $\texttt{pRef}^1$, and y_j and z_k are shiftable in $\texttt{pRef}^2$, it follows that x_i and z_k are shiftable in $\texttt{pRef}^2(\texttt{pRef}^1(\cdot))$. 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 $\mathsf{Leak}(\texttt{pRef}, x, p)$ 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 $\mathsf{Leak}(\texttt{pRef}, x, p)$ to another $\mathsf{Leak}(\texttt{pRef}, x, p)$. 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 $i < i'$ [9]. The values carried in A' are

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

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

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 $\texttt{pRef}$ 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 $\texttt{pRef}$ 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 $k\texttt{pRef}$. Therefore, we start with DG^1. Using the classification of the maximal of $k\texttt{pRef}$ 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 $\mathsf{DG}^{j'}$ 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 $\mathsf{Leak}(k\texttt{pRef}, x, p)$ in one of $\mathsf{Leak}(k\texttt{pRef}, x', p)$. We give the formal description in the full version. □

An example of a maximal for $k\texttt{pRef}$ 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 $k\texttt{pRef}$.

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

Proof. Let $k\texttt{pRef}$ takes as input $(x_0^0, ..., x_{n-1}^0)$ and outputs $(x_0^k, ..., x_{n-1}^k)$, with the ith $\texttt{pRef}$ 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 $\widehat{\mathsf{C}} \leftarrow \mathtt{CC}(\mathsf{C})$, 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.[10]

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, $\widehat{\mathsf{C}} \leftarrow \mathtt{CC}^p(\mathsf{C})$, 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*

$$\widehat{p} = 8[1 - (1 - \sqrt{3p})^{8n}].$$

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.

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

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 $p' = 2(1-(1-\sqrt{3p})^{8n})$ 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 $\Pr[E_i] \leq (4p')^n$. 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, $\widehat{\mathsf{C}} \leftarrow \mathtt{CC}^p(\mathsf{C})$ is $(p, (|\mathsf{C}|)\widehat{p}^n)$-private, with $\widehat{p} = 8[1-(1-\sqrt{3p})^{8n}]$. 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 [ISW03, DFZ19] the condition $\widehat{p} < 1$ requires an upper-bound for the leakage probability p. In detail, Theorem 4 requires the following.

Proposition 11. *Let $p \in [0, \frac{1}{3}]$ be the leakage probability and $\alpha \in (0,1]$. It holds $\widehat{p}^n = 8^n(1-(1-\sqrt{3p})^{8n})^n \leq \alpha$ if*

$$p \leq \frac{\sqrt{1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}}}{3} < 1$$

Proof. If p is smaller than $1/3$ we get $8^n(1-(1-\sqrt{3p})^{8n})^n \leq \alpha$ for an alpha $\alpha \in (0,1]$. The claim is a simple transformation.

$$8^n(1-(1-\sqrt{3p})^{8n})^n \leq \alpha \Leftrightarrow p \leq \frac{\sqrt{1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}}}{3}$$

For more details, we refer to the full version. □

Further, Proposition 11 gives us the required asymptotic behavior of p for our compiler.

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

Proof. Using Proposition 11, we can define the upper-bound p with

$$F_\alpha(n) := \frac{\sqrt{1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}}}{3}.$$

To prove the claim, we will show that there exists a constant C s.t.

$$\lim_{n\to\infty} \frac{F_\alpha(n)}{\frac{1}{\sqrt{n}}} = C.$$

We can transform the term to

$$\lim_{n\to\infty} \frac{F_\alpha(n)}{\frac{1}{\sqrt{n}}} = \lim_{n\to\infty} \sqrt{n}\frac{\sqrt{1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}}}{3} = \lim_{n\to\infty} \frac{1}{3}\sqrt{n\left(1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}\right)}.$$

It remains to study

$$\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}} = e^{\frac{1}{8n}log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right)}.$$

The previous equivalence is true since $0 < \frac{\alpha^{\frac{1}{n}}}{8} \leq \frac{7}{8}$. Now, if $\alpha \neq 0$, then $\lim_{n\to\infty} \frac{1}{8n} log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right) = 0$, thus, using standard analysis techniques, we have that

$$e^{\frac{1}{8n}log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right)} \sim \frac{1}{8n} log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right) + 1.$$

(two functions are asymptotically equivalent if they have the same limit). Thus,

$$\lim_{n\to\infty} \frac{1}{3}\sqrt{n\left(1-\sqrt[8n]{1-\frac{\alpha^{\frac{1}{n}}}{8}}\right)} = \lim_{n\to\infty} \frac{1}{3}\sqrt{n\left(1-1-\frac{1}{8n} log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right)\right)}$$

$$= \lim_{n\to\infty} \frac{1}{3}\sqrt{\frac{n}{8n} log\left(1-\frac{\alpha^{\frac{1}{n}}}{8}\right)} = \lim_{n\to\infty} \frac{1}{3}\sqrt{\frac{1}{8} log\left(1-\frac{1}{8}\right)} = C$$

where in the second to last equality, we have used the fact that if $\alpha > 0$, then $\lim_{n\to\infty} \alpha^{\frac{1}{n}} = 1$. Since C is a constant, this proves the claim of the theorem. □

This estimation is a significant improvement with respect both to the compiler of [DFZ19] and to the knowledge that ISW-like multiplication gadgets are only secure for $p = O(\frac{1}{n})$ [DDF19].

6 Comparison with Dziemobwski et al. [DFZ19]

Our work is inspired by Dziembowski et al. [DFZ19]. 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 [ISW03] 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).

Bound differences. Since every edge of $\mathsf{DG}_{\texttt{pRef}}$ (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, it is enough to need $p = O(\frac{1}{\sqrt{n}})$, instead of $p = O(\frac{1}{n})$.

7 Conclusion

In this paper, we have started from the graphs introduced by Dziemboswki et al. [DFZ19]. Then, we showed how to use a broader class of graphs. Our graphs are more general than [DFZ19], 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 $O(\sqrt{np})^n$-security for the general case. For the first time, we have proved that asymptotically a compiler using the ISW multiplication gadget can be $O(\sqrt{np})^n$-secure and not $O(np)^n$-secure. 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` 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. Further, we thank our reviewers for the many helpful comments to improve the paper.

References

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with o(1/\log (n)) leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *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*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615. Springer, 2016.

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*,

volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.

[BCP+20] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368. Springer, 2020.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *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*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.

[BIS19] Andrej Bogdanov, Yuval Ishai, and Akshayaram Srinivasan. Unconditionally secure computation against low-complexity leakage. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 387–416. Springer, 2019.

[BM06] Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against AES. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2006.

[BMRT22a] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 142–160. IEEE, 2022.

[BMRT22b] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 142–160. IEEE, 2022.

[BRT21a] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. 12697:313–343, 2021.

[BRT21b] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *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*, volume 12697 of *Lecture Notes in Computer Science*, pages 313–343. Springer, 2021.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2010.

[CFOS21] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214. Springer, 2021.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.

[DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *J. Cryptol.*, 32(1):151–177, 2019.

[DFS15] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. Noisy leakage revisited. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9057 of *Lecture Notes in Computer Science*, pages 159–188. Springer, 2015.

[DFZ19] Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. Simple refreshing in the noisy leakage model. In Steven D. Galbraith and Shiho Moriai, editors, *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*, volume 11923 of *Lecture Notes in Computer Science*, pages 315–344. Springer, 2019.

[GJR18] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Thomas Peyrin and Steven D. Galbraith, editors, *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*, volume 11273 of *Lecture Notes in Computer Science*, pages 547–574. Springer, 2018.

[GM10] Shuhong Gao and Todd D. Mateer. Additive fast fourier transforms over finite fields. *IEEE Trans. Inf. Theory*, 56(12):6265–6272, 2010.

[GPRV21] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):599–640, 2021.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology*

Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[MSJ12] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient prfs. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 193–212. Springer, 2012.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[PGMP19] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a rényi day. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 683–712. Springer, 2019.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[Tri03] Elena Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptol. ePrint Arch.*, page 236, 2003.

[Wal01] Colin D. Walter. Sliding windows succumbs to big mac attack. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2001.