Private Circuits: A Modular Approach

Prabhanjan Ananth[1(✉)], Yuval Ishai[2], and Amit Sahai[3]

[1] CSAIL, MIT, Cambridge, USA
prabhanjan@csail.mit.edu
[2] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[3] UCLA, Los Angeles, USA
sahai@cs.ucla.edu

Abstract. We consider the problem of protecting general computations against constant-rate random leakage. That is, the computation is performed by a randomized boolean circuit that maps a randomly encoded input to a randomly encoded output, such that even if the value of every wire is independently leaked with some constant probability $p > 0$, the leakage reveals essentially nothing about the input.

In this work we provide a conceptually simple, modular approach for solving the above problem, providing a simpler and self-contained alternative to previous constructions of Ajtai (STOC 2011) and Andrychowicz et al. (Eurocrypt 2016). We also obtain several extensions and generalizations of this result. In particular, we show that for every leakage probability $p < 1$, there is a finite basis $\mathbb{B}$ such that leakage-resilient computation with leakage probability p can be realized using circuits over the basis $\mathbb{B}$. We obtain similar positive results for the stronger notion of *leakage tolerance*, where the input is not encoded, but the leakage from the entire computation can be simulated given random p'-leakage of input values alone, for any $p < p' < 1$. Finally, we complement this by a negative result, showing that for every basis $\mathbb{B}$ there is some leakage probability $p < 1$ such that for any $p' < 1$, leakage tolerance as above *cannot* be achieved in general.

We show that our modular approach is also useful for protecting computations against *worst case* leakage. In this model, we require that leakage of any $\mathbf{t}$ (adversarially chosen) wires reveal nothing about the input. By combining our construction with a previous derandomization technique of Ishai et al. (ICALP 2013), we show that security in this setting can be achieved with $O(\mathbf{t}^{1+\varepsilon})$ random bits, for every constant $\varepsilon > 0$. This (near-optimal) bound significantly improves upon previous constructions that required more than $\mathbf{t}^3$ random bits.

1 Introduction

Ishai, Sahai, and Wagner [ISW03] introduced the fundamental notion of a leakage-resilient circuit compiler, which in its simplest form is defined as follows. The compiler consists of a triple of algorithms (Compile, Encode, Decode). Given any circuit C, the compiled version of the circuit $\hat{C} = \mathsf{Compile}(C)$ takes a

H. Shacham and A. Boldyreva (Eds.): CRYPTO 2018, LNCS 10993, pp. 427–455, 2018.
https://doi.org/10.1007/978-3-319-96878-0_15

randomly encoded input $\hat{x} = \mathsf{Encode}(x)$ and (using additional fresh randomness) produces an encoded output $\hat{y}$ such that $C(x) = \mathsf{Decode}(\hat{y})$. Furthermore, suppose each wire in the compiled circuit $\hat{C}$ leaks its value[1] with some probability $p > 0$, independently for each wire. Then, informally speaking, we require that the leaked wire values reveal essentially nothing about the input x to the circuit.

The above notion of resilience to random leakage can be seen as a natural cryptographic analogue of the classical notion of fault-tolerant computation due to von Neumann [vN56] and Pippenger [Pip85], where every gate in a circuit can *fail* with some constant probability. In addition to being of theoretical interest, the random leakage model is motivated by the fact that resilience to a notion of "noisy leakage", which captures many instances of real-life side channel attacks, can be reduced to resilience to random leakage [DDF14]. The random leakage model is also motivated by its application to "oblivious zero-knowledge PCPs", where every proof symbol is queried independently with probability p, which in turn are useful for constructing zero-knowledge proofs that only involve unidirectional communication over noisy channels [GIK+15].

We turn to discuss the state of the art on constructing leakage-resilient circuit compilers with respect to leakage probability p. The original work of [ISW03] only achieved security for values of p that vanish both with the circuit size and the level of security. Ajtai [Ajt11] achieved the first leakage-resilient circuit compiler that tolerated some (unspecified) constant probability of leakage p. However, to say the least, Ajtai's result is quite intricate and poorly understood. A more recent work of Andrychowicz, Dziembowski, and Faust [ADF16] obtained a simpler derivation of Ajtai's result. However, their construction is still quite involved and relies on heavy tools such as expander graphs (also used in Ajtai's construction) and algebraic geometric codes. The present work is motivated by the following, informally stated, question:

Is there a "simple" method of building leakage-resilient circuit compilers that can tolerate some constant *probability of leakage $p > 0$?*

1.1 Our Contribution

Our main contribution is an affirmative answer to the above question. We present a conceptually simple, modular approach for solving the above problem, providing a simpler and self-contained alternative to the constructions from [Ajt11, ADF16]. In particular, our construction avoids the use of explicit constant-degree expanders or algebraic geometric codes.

Roughly speaking, our construction uses a recursive amplification technique that starts with a constant-size gadget, which only achieves a weak level of security, and amplifies security by a careful composition of the gadget with itself. The existence of the finite gadget, in turn, follows readily from results on information-theoretic secure multiparty computation (MPC), such as the initial feasibility

[1] The original model of [ISW03] considers the worst-case notion of $\mathbf{t}$-private circuits, where the leakage consists of an adversarially chosen set of $\mathbf{t}$ wires. We will discuss this alternative model later.

results from [BOGW88, CCD88]. We refer the reader to Sect. 1.2 for a more detailed overview of our technique.

We then extend the above result and generalize it in several directions, and also present some negative results. Concretely, we obtain the following results regarding constant-rate random leakage:

- For every leakage probability $p < 1$, there is a finite basis $\mathbb{B}$ such that leakage-resilient computation with leakage probability p can be realized using circuits over the basis $\mathbb{B}$.
- We obtain a similar positive result for the stronger[2] notion of *leakage tolerance*, where the input is not encoded, but the leakage from the entire computation can be simulated given random p'-leakage of input values alone, for any $p < p' < 1$.
- Finally, we complement this by a negative result, showing that for every basis $\mathbb{B}$ there is some leakage probability $p = p_{\mathbb{B}} < 1$ such that for any $p' < 1$, leakage tolerance as above *cannot* be achieved in general, where $p_{\mathbb{B}}$ tends to 1 as $\mathbb{B}$ grows. The negative result is based on impossibility results for information-theoretic MPC without an honest majority [CK91].

Our work leaves open two natural open questions. First, in the case of binary circuits, there is a huge gap between the tiny leakage probability guaranteed by the analysis of our construction (roughly $p = 2^{-14}$) and the best one could hope for. This is the case even in the stronger model of leakage tolerance, where our negative result only rules out constructions that tolerate $p > 0.8$ leakage probability.

A second question is the possibility of tolerating higher leakage probability (arbitrarily close to 1) for the weaker notion of *leakage-resilient* circuits with input encoder. A partial explanation for the difficulty of this question is the possibility of using the input encoder to generate correlated randomness that enables information-theoretic MPC with no honest majority.[3]

Private Circuits with Near-Optimal Randomness. As an unexpected application of our technique, we show that the modular approach is also useful for protecting computations in the more standard model of *worst case leakage*. Indeed, we show that essentially the same construction that is secure in the random probing model is also secure in the worst case leakage model with threshold $\mathbf{t}$. Using this observation and a certain "randomness locality" feature of our construction, and building on robust local pseudo-random generators [IKL+13], we

[2] Note that leakage-tolerance can be easily used to achieve leakage-resilience by letting the encoder apply to the input a secret sharing scheme that tolerates a p'-fraction of leakage, where the compiler is applied to an augmented circuit that starts by reconstructing the input from its shares.

[3] Indeed, the technique of Beaver [Bea91] can be used to obtain resilience to an arbitrary leakage probability $p < 1$, but at the cost of allowing the output of the input encoder to be bigger than the circuit size. In contrast, our definition of leakage-resilient circuit compiler requires the output of the input encoder to be a fixed polynomial in the input length, independently of the size of the circuit.

obtain leakage tolerant circuit compilers with leakage parameter $\mathbf{t}$ that use only $O(\mathbf{t}^{1+\varepsilon})$ random bits, for any constant $\varepsilon > 0$. We show that this bound is nearly tight by observing that at least $\mathbf{t}$ random bits are required to protect computations against worst case leakage. Our upper bound on the randomness complexity is a major improvement over the best previous upper bound of $O(\mathbf{t}^{3+\varepsilon})$ from [IKL+13].

We present our results formally in Sect. 3.3.

1.2 Technical Overview

In this section, we give a high level overview of the composition-based approach that we utilize to get our main result. We use the composition-based approach to achieve constructions of leakage-resilient and leakage tolerant circuit compilers in both the worst-case probing and random probing settings. For the most part of the current discussion, we focus on achieving leakage resilient circuit compilers in the random probing setting.

In the composition-based approach, we start with a leakage-resilient circuit compiler CC_0 secure against $\mathbf{p}$-random probing attacks and has constant simulation error ε. By $\mathbf{p}$-random probing attacks, we mean that every wire in the compiled circuit is leaked with probability $\mathbf{p}$. We refer to this leakage-resilient circuit compiler as a base gadget. The goal is to recursively compose this base gadget to obtain a leakage-resilient circuit compiler also secure against $\mathbf{p}$-random probing attacks but the failure probability is negligible (in the size of the circuit being compiled).

First Attempt. A naive approach to compose is as follows: to compile a circuit C, compute $\mathsf{CC}_0.\mathsf{Compile}(\cdots \mathsf{CC}_0.\mathsf{Compile}(C)\cdots)$. In the k^{th} step, $\mathsf{CC}_0.\mathsf{Compile}$ is executed for k levels of recursion. Its easy to see that leakage on the resulting compiled circuit cannot be simulated only if it holds that the simulation of $\mathsf{CC}_0.\mathsf{Compile}$ fails for every level of recursion. That is, the failure probability of the resulting circuit compiler is ε^k for k levels of recursion. If we set k to be the size of C then we obtain negligible simulation error, as desired. However, as the simulation error reduces with every recursion step, the size of the compiled circuit increases with every recursion step. Even if the compiled circuit in the base gadget had constant overhead, the size of the compiled circuit obtained after k steps grows exponential in k. This means that we need to devise a composition mechanism where the error probability degrades much faster than the size growth of the compiled circuit.

Our Approach: In a Nutshell. Our idea is to cleverly compose n gadgets, each with simulation error ε, in such a way that the composed gadget fails only if at least t of the gadgets fail, for some parameters t, n with $t < n$. Our composition mechanism ensures that the size of the composed gadget incurs a constant blowup whereas the simulation error degrades exponentially in $\frac{1}{\varepsilon}$.

To realize such a composition mechanism, we employ techniques from Cohen et al. [CDI+13]. Cohen et al. showed how to employ player emulation strategy [HM00] to achieve a conceptually simpler construction of secure MPC in the honest majority setting. While the goal of Cohen et al. is seemingly unrelated to the problem we are trying to solve, we show that the player emulation strategy employed by their work can be adapted to our context.

We first recall their approach. They showed how to transform a threshold formula, composed solely of threshold gates, into a secure MPC protocol. In more detail, they start with a T-out-N threshold formula composed of t-out-n threshold gates. They then show how to transform a secure MPC protocol for n parties tolerating t corruptions into a MPC protocol for N parties tolerating at most T corruptions (also written as T-out-N secure MPC). At a high level, their transformation proceeds as follows: they replace the topmost t-out-n threshold gate with a T-out-N secure MPC. That is, every input wire of the topmost gate corresponds to a party in the secure MPC protocol. Every party in this MPC is emulated by a T-out-N secure MPC. In other words, for every gate input to the topmost gate, the corresponding player is replaced with a t-out-n secure MPC. For instance, if the topmost gate had exactly N gates as its children then the resulting MPC has n^2 number of parties and can tolerate at most t^2 number of corruptions. This process can be continued as long as the secure MPC protocol still satisfies polynomial efficiency.

Armed with their methodology, we show how to construct a leakage-resilient circuit compiler. We start with a t-out-n secure MPC protocol Π in the passive security model. The functionality associated with this protocol takes as input n shares of two bits (a, b) and outputs n shares of NAND(a, b)[4]. This secure MPC protocol will be our base gadget for NAND with respect to some constant probability of wire leakage and constant simulation error. We then compose this base gadget as follows: in the k^{th} level of recursion, we start with Π and *emulate* the computation of every gate in Π with an inner gadget computed from $(k-1)^{th}$ level of recursion. Why is this secure? the hope is that the resulting gadget can be simulated by simulating all the inner gadgets. Unfortunately, this doesn't work since some of the inner gadgets can fail. However, we can map the inner gadgets that fail to corrupting the corresponding parties in Π. And thus, as long as at most t inner gadgets fail, we can invoke the simulator of Π to simulate the composed gadget. We can show that the probability that at most t inner gadgets fail degrades exponentially in $\frac{1}{\varepsilon_{k-1}}$, where ε_{k-1} is the simulation error of the inner gadget. On the other hand, the size of the composed gadget grows only by a constant factor. Expanding this out, we can conclude that after k steps the size grows exponential in k whereas the simulation error degrades *doubly* exponential in k. Substituting k to be logarithmic in the size of C, we attain the desired result. While the current discussion focusses on the analysis for the random probing setting, similar (and a much simpler) analysis can also be done for the worst-case probing setting. Specifically, we can show that after k levels of recursion, the circuit compiler is secure against worst case probing attacks with leakage parameter t^k.

[4] We consider NAND gates because they are universal gates. In fact we can substitute NAND with any other universal basis.

Security Issues. Recall that the simulation of the composed gadget requires simulating all the inner gadgets. Since the inner gadgets are connected to each other, we need to ensure that these different simulations are consistent with each other. To give an example, suppose there are two inner gadgets connected by a wire w. The simulators for these two different inner gadgets could assign conflicting values to w. At its core, we handle this problem by keeping a budget of wires "in reserve", and define a notion of composable simulation that can make use of this flexibility to resolve conflicts between simulators for components that share wires. For example, if two simulators S_1 and S_2 "want to disagree" about a wire w, we will break the tie by allowing simulator S_1 to decide the value in wire w, and asking the other simulator S_2 to use one of the reserve wires to make up for the fact that S_2 did not get its wish for the value of wire w. This is possible because of the flexibility inherent in the secret sharing schemes underlying the MPC protocols of the base gadget. Similar notions of composable leakage-resilient circuit compliers were considered in [BBD+16,BBP+16,BBP+17].

From NAND *to arbitrary circuits.* So far the above approach shows how to design a gadget for NAND tolerating constant wire leakage probability and with negligible simulation error. The fact that we design gadgets just for NAND gates is crucially used to argue that the size of the composed gadget blows up only by a constant factor in each step. We show how to use this gadget to design a gadget for any circuit over NAND basis: to compile C, we replace every gate in C with a gadget for NAND. We then show how to stitch these different gadgets together to obtain a gadget for C.

Final Template. We now lay out our template. We first define a special case of leakage-resilient circuit compilers, called *composable* circuit compilers. This notion will incorporate the composition simulation mechanism mentioned earlier.

– The first step is to design a composable circuit compiler for NAND tolerating constant wire leakage probability and has constant simulation error.
– We then apply our composition approach to obtain a composable circuit compiler for NAND tolerating constant wire leakage probability and has negligible simulation error.
– Finally, we show how to bootstrap a composable circuit compiler for NAND to obtain a composable circuit compiler for any circuit. The resulting compiler still tolerates constant wire leakage probability and has negligible simulation error.

A leakage tolerant circuit compiler can be constructed by additionally designing a leakage resilient input encoder.

Randomness Complexity. As discussed above, an unexpected feature of our construction is that it allows us to obtain leakage tolerant circuit compilers in the worst case probing setting with near-optimal randomness complexity. This application relies on the fact that after k levels of recursion, the compiled circuit has *randomness locality* of $O(k)$. (The randomness locality of a circuit compiler is said to be d if the value assigned to every wire during the evaluation of a compiled circuit depends on the inputs and at most d randomness gates.) In particular, we can construct a compiler with randomness locality $O(\log(\mathbf{t}))$ that is secure against $\mathbf{t}$-worst case probing attacks. This can be argued by observing that the initial compiled circuit has constant randomness locality and in every recursion step, the randomness locality increases by a constant. Combining this with a result from [IKL+13], we obtain a circuit compiler secure in the worst case probing model with threshold $\mathbf{t}$ and randomness complexity $\mathbf{t}^{1+\varepsilon}$. This improves upon the bound of $\mathbf{t}^{3+\varepsilon}$ in [IKL+13].

Organization. We first present the necessary preliminaries in Sect. 2. We then define the notion of circuit compilers in Sect. 3. We define leakage resilience and leakage tolerance in the same section. The notion of composable circuit compilers, that will be a building block for both leakage tolerant and leakage resilient circuit compilers, is presented in Sect. 4.1. We present the starting step (base case) in the composition step in Sect. 4.2. The composition step itself is presented in Sect. 4.3. The result of the composition step doesn't quite meet our efficiency requirements and so we present the exponential-to-polynomial transformation in Sect. 4.4. Finally, we combine all these steps to present the main construction of a composable circuit compiler in Sect. 4.5.

Armed with a construction of composable circuit compiler, we present a construction of leakage *tolerant* circuit compilers in Sect. 5. We also present negative results that upper bounds the leakage rate in the random probing model in the same section. We show that the construction of leakage tolerant circuit compiler can be transformed to have small randomness complexity. This is shown in Sect. 7. In the same section, we show a lower bound on randomness complexity of leakage tolerant circuit compilers.

We show implication of composable circuit compilers to leakage *resilient* circuit compilers in Sect. 6.

2 Preliminaries

We use the abbreviation PPT for probabilistic polynomial time. Some notational conventions are presented below.

- Suppose A is a probabilistic algorithm. We use the notation $y \leftarrow A(x)$ to denote that the output of an execution of A on input x is y.
- Suppose $\mathcal{D}$ is a probability distribution with support $\mathcal{V}$. We denote the sampling algorithm associated with $\mathcal{D}$ to be $\mathsf{Sampler}$. We denote by $x \xleftarrow{\$} \mathsf{Sampler}$ if the output of an execution of $\mathsf{Sampler}$ is x. For every $x \in \mathcal{V}$, $\mathsf{Sampler}$ outputs x with probability p_x, as specified by $\mathcal{D}$. Unless specified otherwise, we only consider efficiently sampleable distributions. We also consider parameterized distributions of the form $\mathcal{D} = \{\mathcal{D}_{aux}\}$. In this case, there is a sampling algorithm $\mathsf{Sampler}$ defined for all these distributions. $\mathsf{Sampler}$ takes as input aux and outputs an element in the support of $\mathcal{D}_{aux}$.

- Consider two probability distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ with discrete support $\mathcal{V}$ and let their associated sampling algorithms be $\mathsf{Sampler}_1$ and $\mathsf{Sampler}_2$. We denote $\mathcal{D}_0 \approx_{s,\varepsilon} \mathcal{D}_1$ if the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are ε-statistically close. That is, $\sum_{v \in \mathcal{V}} |\mathsf{Pr}[v \leftarrow \mathsf{Sampler}_1] - \mathsf{Pr}[v \leftarrow \mathsf{Sampler}_2]| \leq 2\varepsilon$.

Circuits. A deterministic boolean circuit C is a directed acyclic graph whose vertices are boolean gates and whose edges are wires. The boolean gates belong to a basis $\mathbb{B}$. An example of a basis is $\mathbb{B} = \{\mathbf{AND}, \mathbf{OR}, \mathbf{NOT}\}$. We will assume without loss of generality that every gate has fan-in (the number of input wires) at most 2 and fan-out[5] (the number of output wires) at most 2. A randomized circuit is a circuit augmented with random-bit gates. A random-bit gate, denoted by **RAND**, is a gate with fan-in 0 that produces a random bit and sends it along its output wire; the bit is selected uniformly and independently of everything else afresh for each invocation of the circuit. We also consider basis consisting of functions (possibly randomized) on finite domains (as opposed to just boolean gates). The size of a circuit is defined to be the number of gates in the circuit.

2.1 Information Theoretic Secure MPC

We now provide the necessary background of secure multiparty computation. In this work, we focus on information theoretic security. We first present the syntax and then the security definitions.

Syntax. We define a secure multiparty computation protocol $\mathit{\Pi}$ for n parties $P_1, \ldots, P_n$ associated with an n-party functionality $F : \{0,1\}^{\ell_1} \times \cdots \times \{0,1\}^{\ell_n} \times \{0,1\}^{\ell_r} \rightarrow \{0,1\}^{\ell_{y_1}} \times \cdots \times \{0,1\}^{\ell_{y_n}}$. We denote ℓ_i to be the length of the i^{th} party's input, ℓ_{y_i} to be the length of the i^{th} party's output and ℓ_r is the length of the randomness input to F. In any given execution of the protocol, the i^{th} party receives as input $x_i \in \{0,1\}^{\ell_i}$ and all the parties jointly compute the functionality $F(x_1, \ldots, x_n; r)$, where $r \in \{0,1\}^{\ell_r}$ is sampled uniformly at random. In the end, party P_i outputs y_i, where $(y_1, \ldots, y_n) = F(x_1, \ldots, x_n; r)$.

We defined such n-party functionalities that additionally receive the randomness as input to be *randomized functionalities*. In this work we only consider randomized n-party functionalities and henceforth, the input randomness will be implicit in the description of the functionality.

Semi-honest Adversaries. We consider the adversarial model where the adversaries follow the instructions of the protocol. That is, they receive their inputs from the environment, behave as prescribed by the protocol and finally output their view of the protocol. Such type of adversaries are referred to as semi-honest adversaries.

We define semi-honest security below. Denote $\mathsf{Real}^{\mathit{\Pi}}_{F,S}(x_1, \ldots, x_n)$ to be the joint distribution over the outputs of all the parties along with the views of the parties indexed by the set S.

[5] If a circuit has arbitrary fan-out, then this can be transformed into another circuit of fan-out 2 with a loss of logarithmic factor in the depth.

Definition 1 (Semi-Honest Security). *Consider a n-party functionality F as defined above. Fix a set of inputs $(x_1, \ldots, x_n)$, where $x_i \in \{0,1\}^{\ell_i}$ and let r_i be the randomness of the i^{th} party. Let Π be a n-party protocol implementing F. We say that Π satisfies* **ε-statistical security against semi-honest adversaries** *if for every subset of parties S, there exists a PPT simulator* Sim *such that:*

$$\{\ (\{y_i\}_{i \notin S}, \mathsf{Sim}\,(\{y_i\}_{i \in S}, \{x_i\}_{i \in S}))\ \} \approx_{s,\varepsilon} \left\{ \mathsf{Real}^{\Pi}_{F,S}(x_1, \ldots, x_n) \right\},$$

where y_i is the i^{th} output of $F(x_1, \ldots, x_n)$. If the above two distributions are identical, then we say that Π satisfies **perfect security against semi-honest adversaries**.

Starting with the work of [BOGW88, CCD88], several constructions construct semi-honest secure multi-party computation protocol in the information-theoretic setting assuming that a majority of the parties are honest.

We consider the notion of randomness locality of a secure MPC protocol.

Definition 2 (Randomness Locality). *A semi-honest secure multiparty computation protocol for a functionality F is said to have randomness locality d if every value computed in the protocol is determined by the inputs of all parties and at most d random bits (either as input to the functionality or to the parties).*

3 Circuit Compilers

We define the notion of circuit compilers. This notion allows for transforming an input x, a circuit C (See Sect. 2 for a definition of circuits) into an encoded input $\widehat{x}$ and a randomized circuit $\widehat{C}$ such that evaluation of $\widehat{C}$ on $\widehat{x}$ yields an encoding $\widehat{C(x)}$. The decode algorithm then decodes $\widehat{C(x)}$ to yield $C(x)$.

Definition 3 (Circuit Compilers). *A circuit compiler* CC *defined for a class of circuits $\mathcal{C}$ comprises of the following algorithms* $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *defined below:*

- **Circuit Compilation**, $\mathsf{Compile}(C)$*: It is a deterministic algorithm that takes as input circuit C and outputs a randomized circuit $\widehat{C}$.*
- **Input Encoding**, $\mathsf{Encode}(x)$*: This is a probabilistic algorithm that takes as input x and outputs an encoded input $\widehat{x}$.*
- **Output Decoding**, $\mathsf{Decode}(\widehat{y})$*: This is a deterministic algorithm that takes as input an encoding $\widehat{y}$ and outputs the plain text string y.*

The algorithms defined above satisfies the following properties:

- **Correctness of Evaluation***: For every circuit $C \in \mathcal{C}$ of input length ℓ, every $x \in \{0,1\}^{\ell}$, it always holds that $y = C(x)$, where:*
  - $\widehat{C} \leftarrow \mathsf{Compile}(C)$.
  - $\widehat{x} \leftarrow \mathsf{Encode}(x)$.
  - $\widehat{y} \leftarrow \widehat{C}(\widehat{x})$.

  - $y \leftarrow \mathsf{Decode}(\widehat{y})$.
- **Efficiency**: *Consider a parameter* $k \in \mathbb{N}$. *We require that the running time of* $\mathsf{Compile}(C)$ *to be* $\text{poly}(k, |C|)$, *the running time of* $\mathsf{Encode}(x)$ *to be* $\text{poly}(k, |x|)$ *and the running time of* $\mathsf{Decode}(\widehat{C(x)})$ *to be* $\text{poly}(k, |C(x)|)$. *We emphasize that the encoding complexity only grow poly-logarithmically in terms of the size of* C. *Typically,* k *will be set to* $\text{poly}(\log(|C|))$.

Few remarks are in order.

Remark 1. The standard basis we consider in this work is $\{\mathbf{AND}, \mathbf{XOR}\}$. Unless otherwise specified, all the circuits considered in this work will be defined over the standard basis. Also unless otherwise specified, the compiled circuit is over the same basis as the original circuit.

Remark 2. Later, we also consider circuit compilers with relaxed efficiency guarantees, where we allow for the running time of the algorithms to be exponential in the parameter k.

Additional Properties. We are interested in circuit compilers that have (i) low randomness locality: every value in the execution of the compiled circuit depends only on few random bits and, (ii) low randomness complexity: only a small amount of randomness should be used in the evaluation of the compiled circuit.

We capture these two properties formally below.

Definition 4 (Randomness Locality). *Consider a circuit compiler* CC *defined for a class of circuits* $\mathcal{C}$ *comprising of the following algorithms* $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$. CC *has d-randomness locality if for every circuit* $C \in \mathcal{C}$, *input* x, *the value of every wire in the computation of* $\widehat{C}$ *on* $\widehat{x}$ *is determined by at most* d *random-bit gates in* $\widehat{C}$ *and* $\widehat{x}$, *where (i)* $\widehat{C} \leftarrow \mathsf{Compile}(C)$ *and, (ii)* $\widehat{x} \leftarrow \mathsf{Encode}(x)$.

Definition 5 (Randomness Complexity). *Consider a circuit compiler* CC *defined for a class of circuits* $\mathcal{C}$ *comprising of the following algorithms* $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$. CC *has randomness complexity* r *if the number of random-bit gates in the compiled circuit is at most* r.

Non-Boolean Basis. In this work, we also consider a setting where the compiled circuit is defined over a basis that is different from the basis of the original circuit (before compilation). We define this formally below.

Definition 6. *Consider two collections of finite functions* $\mathbb{B}'$ *and* $\mathbb{B}$. *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *is defined over* $\mathbb{B}'$ *(written* CC *over* $\mathbb{B}'$*) for a class of circuits* $\mathcal{C}$ *over* $\mathbb{B}$ *if it holds that for every* $C \in \mathcal{C}$ *over basis* $\mathbb{B}$, *the compiled circuit* $\widehat{C}$, *generated as* $\widehat{C} \leftarrow \mathsf{Compile}(C)$, *is defined over basis* $\mathbb{B}'$.

We next define the security guarantees associated with circuit compilers.

3.1 Leakage Resilience

We adopt the definition of leakage resilient circuit compilers from [GIM+16].

Definition 7. *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a class of circuits* $\mathcal{C}$ *is said to be* ε*-leakage resilient against a class of randomized leakage functions* $\mathcal{L}$ *if the following holds:*

There exists a PPT simulator Sim *such that for every circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *and* $C \in \mathcal{C}$*, input* $x \in \{0,1\}^\ell$*, leakage function* $L_{comp} \in \mathcal{L}$*, the distribution* $L_{comp}(\widehat{C},\ \widehat{x})$ *is* ε*-statistically close to* $\mathsf{Sim}\,(C)$*, where* $\widehat{C} \leftarrow \mathsf{Compile}(C)$ *and* $\widehat{x} \leftarrow \mathsf{Encode}(x)$*.*

Informally, the above definition states that the leakage L_{comp} on the computation of the compiled circuit $\widehat{C}$ on encoded input $\widehat{x}$ reveals no information about the input x.

Remark 3. While the above notion considers leakage only on a single computation, this notion already implies the stronger multi-leakage setting where there are multiple encoded inputs and a leakage function is computed on every computation of $\widehat{C}$. This follows from a standard hybrid argument[6].

p-*Random Probing Attacks* [ISW03, Ajt11, ADF16]. In this work, we are interested in the following probabilistic leakage function: every wire in the computation of the compiled circuit $\widehat{C}$ on the encoded input $\widehat{x}$ is leaked independently with probability $\mathbf{p}$.

More formally, denote the leakage function $\mathcal{L}_{\mathbf{p}} = \{L_{comp}\}$, where the probabilistic function L_{comp} is defined below.

$L_{comp}\left(\widehat{C}, \widehat{x}\right)$: construct the set of leaked values $\mathcal{S}^C_{\mathsf{leak}}$ as follows. For every wire w (input wires included) in $\widehat{C}$ and value v_w assigned to w during the computation of $\widehat{C}$ on $\widehat{x}$, include (w, v_w) with probability $\mathbf{p}$ in $\mathcal{S}^C_{\mathsf{leak}}$. Also, include (w', v_w) in $\mathcal{S}^C_{\mathsf{leak}}$, if w' and w are two output wires of the same gate. Output $\mathcal{S}^C_{\mathsf{leak}}$.
We define leakage resilient circuit compilers with respect to the leakage function defined above.

Definition 8 (Leakage Resilience Against Random Probing Attacks). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a family of circuits* $\mathcal{C}$ *is said to be* $(\mathbf{p}, \varepsilon)$*-leakage resilient against random probing attacks if* CC *is* ε*-leakage resilient against* $\mathcal{L}_{\mathbf{p}}$*. Moreover, we define the leakage rate of* CC *to be* $\mathbf{p}$*.*

t-*Probing (Worst Case Probing) Attacks.* We also consider **t**-probing attacks, where the adversary is allowed to observe any t wires in the computation of the compiled circuit. We define the class of leakage functions $\mathcal{L}_{\mathbf{t}} = \{L^S_{comp}\}_{|S| \leq t}$, where L^S_{comp} is defined below.

[6] Here we use the fact that the circuit compilation algorithm is deterministic.

$L^S_{comp}\left(\widehat{C},\widehat{x}\right)$: construct the set of leaked values $\mathcal{S}^C_{\mathsf{leak}}$ as follows. For every wire $w \in S$ and v_w assigned to w during the computation of $\widehat{C}$ on $\widehat{x}$, include (w, v_w) in $\mathcal{S}^C_{\mathsf{leak}}$. Also, include (w', v_w) in $\mathcal{S}^C_{\mathsf{leak}}$, if w' and w are two output wires of the same gate. Output $\mathcal{S}^C_{\mathsf{leak}}$.

Definition 9 (Leakage Resilience Against Worst Case Probing Attacks). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a family of circuits* $\mathcal{C}$ *is said to be leakage resilient against* $\mathbf{t}$*-probing attacks if* CC *is leakage resilient against* $\mathcal{L}_{\mathbf{t}}$*. Moreover, we define the leakage parameter of* CC *to be* $\mathbf{t}$.

3.2 Leakage Tolerance

Another notion we study is leakage tolerant circuit compilers. In this notion, unlike leakage resilient circuit compilers, Encode is an identity function. Consequently, we need to formalize the security definition so that the leakage on the computation of $\widehat{C}$ on x can be simulated with bounded leakage on the input x.

Definition 10. *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a class of circuits* $\mathcal{C}$ *is said to be* ε*-leakage tolerant against a class of leakage functions* $\mathcal{L}$ *if the following two conditions hold:*

- Encode *is an identity function.*
- *There exists a simulator* Sim *such that for every circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *and* $C \in \mathcal{C}$*, input* $x \in \{0,1\}^\ell$*, leakage function* $L = (L_{comp}, L_{inp}) \in \mathcal{L}$*, the distribution* $L_{comp}(\widehat{C},\ \widehat{x})$ *is* ε*-statistically close to* $\mathsf{Sim}\left(C, L_{inp}(x)\right)$*, where* $\widehat{C} \leftarrow \mathsf{Compile}(C)$ *and* $\widehat{x} \leftarrow \mathsf{Encode}(x)$*.*

Henceforth, we omit Encode *algorithm and denote a leakage tolerant circuit compiler to consist of* $(\mathsf{Compile}, \mathsf{Decode})$*.*

$(\mathbf{p}, \mathbf{p}')$*-Random Probing Attacks.* As before, we are interested in the following probabilistic leakage function: every wire in the computation of the compiled circuit $\widehat{C}$ on the encoded input $\widehat{x}$ is leaked independently with probability $\mathbf{p}$.

More formally, denote the leakage function $\mathcal{L}_{\mathbf{p},\mathbf{p}'} = \{(L_{comp}, L_{inp})\}$, where the probabilistic functions L_{comp} is as defined in Sect. 3.1 and L_{inp} is defined below.

$L_{inp}(x)$: construct the set of leaked values $\mathcal{S}^I_{\mathsf{leak}}$ as follows. For every input wire w carrying the i^{th} bit of x, include (w, x_i) in $\mathcal{S}^I_{\mathsf{leak}}$ with probability $\mathbf{p}'$. If (w, x_i) is included, also include (w', x_i) in $\mathcal{S}^I_{\mathsf{leak}}$, where w' is the other input wire carrying x_i. Output $\mathcal{S}^I_{\mathsf{leak}}$.

We define leakage tolerance against random probing attacks below.

Definition 11 (Leakage Tolerance Against Random Probing Attacks). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Decode})$ *for a family of circuits* $\mathcal{C}$ *is said to be* $(\mathbf{p}, \mathbf{p}', \varepsilon)$*-leakage tolerant against random probing attacks if* CC *is* ε*-leakage tolerant against* $\mathcal{L}_{\mathbf{p},\mathbf{p}'}$*. Moreover, we define the leakage rate of* CC *to be* $\mathbf{p}$*.*

$\mathbf{t}$-*Probing (Worst Case Probing) Attacks.* As before, we are interested in the class of leakage functions where the adversary is allowed to query a $\mathbf{t}$-sized subset of wire values in the circuit. We consider the class of leakage functions $\mathcal{L}_{\mathbf{t}} = \{(L_{comp}^{S}, L_{inp}^{S'})\}_{|S'| \leq \mathbf{t}}$, where L_{comp}^{S} is as defined in Sect. 3.1 and $L_{inp}^{S'}$ is defined below.

$L_{inp}^{S'}\left(\widehat{C}, \widehat{x}\right)$: construct the set of leaked values $\mathcal{S}_{\mathsf{leak}}^{I}$ as follows. include (w, x_i) in $\mathcal{S}_{\mathsf{leak}}^{I}$ if and only if $w \in S'$ and wire w carries the i^{th} bit of x. If w' also carries the i^{th} bit of x, include (w', x_i) in $\mathcal{S}_{\mathsf{leak}}^{I}$. Output the set $\mathcal{S}_{\mathsf{leak}}^{I}$.

Definition 12 (Leakage Tolerance Against Worst Case Probing Attacks). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a family of circuits* $\mathcal{C}$ *is said to be leakage tolerant against* $\mathbf{t}$*-probing attacks if* CC *is leakage tolerant against* $\mathcal{L}_{\mathbf{t}}$*. Moreover, we define the leakage parameter of* CC *to be* $\mathbf{t}$*.*

3.3 Our Results

We state our results below.

<u>Worst Case Probing</u>:

Randomness Complexity. We prove positive and negative results on the randomness complexity of leakage tolerant circuit compilers. We prove this is in the worst case probing regime. The proofs for both the theorems can be found in Sect. 7.

Theorem 1 (Randomness Complexity: Positive Result). *There is a leakage tolerant circuit compiler such that given a circuit of size* s *and worst-case leakage bound* $\mathbf{t}$*, the compiler outputs a circuit of size* $s \cdot \mathrm{poly}(\mathbf{t})$ *which is perfectly secure against* $\mathbf{t}$ *(worst-case) probing attacks and uses only* $\mathbf{t}^{1+\varepsilon}$ *random bits.*

Theorem 2 (Randomness Complexity: Negative Result). *The number of random bits used in any leakage tolerant circuit compiler secure against* $\mathbf{t}$*-probing attacks is at least* $\mathbf{t}$*.*

En route to proving the above positive result, we prove that there is a construction of leakage tolerant circuit compiler that has randomness locality $\log(\mathbf{t})$. This is shown in Sect. 5.2.

Lemma 1 (Randomness Locality). *There is a leakage tolerant circuit compiler secure against* $\mathbf{t}$*-probing attacks satisfying* $O(\log(\mathbf{t}))$*-randomness locality.*

<u>Random Probing</u>:

Leakage Tolerance: Positive Results. We show the following results in Sect. 3.2.

Theorem 3 (Boolean Basis). *There exist constants* $0 < \mathbf{p} < \mathbf{p}' < 1$ *such that there is a* $(\mathbf{p}, \mathbf{p}', \epsilon)$*-leakage tolerant circuit compiler, where* ϵ *is negligible in the circuit size.*

Theorem 4 (Finite Basis). *For any* $0 < \mathbf{p}' < \mathbf{p} < 1$ *there is a basis* $\mathbb{B}$ *over which there is a* $(\mathbf{p}, \mathbf{p}', \epsilon)$*-leakage tolerant circuit compiler, where* ϵ *is negligible in the circuit size.*

Leakage Tolerance: Negative Result. The following theorem upper bounds the rate of a leakage tolerant circuit compiler in the random probing model. We prove this theorem in the full version.

Theorem 5. *For any basis* $\mathbb{B}$ *there is* $0 < \mathbf{p} < 1$*, such that for any* $0 < \mathbf{p}' < 1$*, there is no* $(\mathbf{p}, \mathbf{p}', 0.1)$*-leakage tolerant circuit compiler over* $\mathbb{B}$*.*

Leakage Resilience: Positive Results. We demonstrate a construction of leakage resilient circuit compiler over boolean basis. Both the theorems below are shown in Sect. 6.

Theorem 6 (Boolean Basis). *There is a constant* $0 < \mathbf{p} < 1$ *such that there is a* $(\mathbf{p}, \epsilon)$*-leakage resilient circuit compiler and* ϵ *is negligible in the circuit size.*

We prove a result about finite basis in the full version.

Theorem 7 (Finite Basis). *For any* $0 < \mathbf{p} < 1$ *there is a basis* $\mathbb{B}$ *over which there is a* $(\mathbf{p}, \epsilon)$*-leakage resilient circuit compiler, where* ϵ *is negligible in the circuit size.*

4 Composition Theorem: Intermediate Step

We present a composition theorem, a key step in our constructions of leakage tolerant and leakage resilient circuit compilers. We identify a type of circuit compilers satisfying some properties, that we call *composable circuit compilers*. This notion will be associated with 'composition-friendly' properties.

Before we formally define the properties, we motivate the use of composable circuit compilers.

- In our composition theorem, we need to 'attach' different composable circuit compiler gadgets. For instance, the output wires of composable compiler CC_1 will be the input wires of another compiler CC_2. In order to ensure correctness, we need to make sure that the output encoding of CC_1 is the same as the input encoding of CC_2. We guarantee this by introducing XOR encoding property that states that the input encoding and output encoding are additive secret shares.
- While the above bullet resolves the issue of correctness, this raises some security concerns. In particular, when we simulate CC_1 and CC_2 separately, conflicting values could be assigned to the wires that join CC_1 and CC_2. These issues have been studied in the prior works, mainly in the context of worst case leakage [BBD+16, BBP+16, BBP+17]. And largely, this was not formally studied for the random probing setting. We formulate the following simulation definition to handle this issue in the probabilistic setting: the simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ (termed as partial simulator) will work in two main steps:
  - In the first step, the simulator first determines the wires to be leaked. Then, Sim_1 determines a 'shadow' of input and output wires that additionally need to be simulated.

- In the second step, the values for the input and output wires selected in the above step is assigned values. Then Sim_2 is executed to assign the internal wire values.

At a high level Sim works as follows: first $\mathsf{CC}_1.\mathsf{Sim}_1$ and $\mathsf{CC}_2.\mathsf{Sim}_1$ is executed to obtain the shadow of input and output wires that need to be simulated. At this point, we take the union of the output wires of CC_1 and input wires of CC_1 that need to be simulated. Then, we assign the values to all the wires. Once this is done, we independently execute $\mathsf{CC}_1.\mathsf{Sim}_2$ and $\mathsf{CC}_2.\mathsf{Sim}_2$ to obtain the simulated wire values in both CC_1 and CC_2, as desired.

4.1 Composable Circuit Compilers

The syntax of composable circuit compilers is the same as that of circuit compilers (Definition 3). In addition, it is required to satisfy the properties stated next.

XOR Encoding Property. We start with XOR encoding property. This property states that the input encoding (resp., output encoding) is an additive secret sharing of the inputs (resp., outputs).

Definition 13 (N-XOR Encoding). *A circuit compiler* $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a family of circuits $\mathcal{C}$ is said to have* ***N-XOR encoding property*** *if the following always holds: for every circuit $C \in \mathcal{C}, x \in \{0,1\}^{\ell}$,*

- $\mathsf{Encode}(x)$ *computes XOR secret sharing of x_i for every $i \in [\ell]$, where x_i is the i^{th} input bit of x. It then outputs the concatenation of the XOR secret shares of all the bits of x.*
  It outputs $\widehat{x} = (\widehat{x}^1, \ldots, \widehat{x}^{\ell}) \in \{0,1\}^{\ell N}$, where $x_i = \oplus_{j=1}^{N} \widehat{x}^i_j$. That is, x_i is a XOR secret sharing of $\{\widehat{x}^i_j\}_{j \in [N]}$.
- *Let $\widehat{x} \leftarrow \mathsf{Encode}(x)$ and $\widehat{C} \leftarrow \mathsf{Compile}(C)$. Upon evaluation, denote the output encoding to be $\widehat{y} \leftarrow \widehat{C}(\widehat{x})$. Suppose $C(x) = y \in \{0,1\}^{\ell'}$ and $\widehat{y} = (\widehat{y}^1, \ldots, \widehat{y}^{\ell'}) \in \{0,1\}^{\ell' N}$. We require that $\{\widehat{y}^i_j\}$ is a XOR secret sharing of y_i, i.e., $y_i = \oplus_{j=1}^{N} \widehat{y}^j_i$.*

When N is clear from the context, we drop it from the notation.

Composable Security (Random Probing Setting). Next, we define the composable security property. We first deal with the random probing setting. There are two parts associated with this security property.

- **Partial simulation:** This states that, conditioned on the simulator not aborting, the leakage of all the wires in the compiled circuit can be perfectly simulated by the leakage of a fraction of values assigned to the input and output wires alone.
- **Simulation with Abort:** We require that the simulator aborts with small probability.

Before stating the formal definition of composable security, we first set up some notation. We formalize the leakage function L_{comp} defined in the previous section in terms of the following sampler algorithm, $\mathsf{RPDistr}^w_{\mathbf{p}}(\cdot,\cdot)$[7].

<u>SAMPLER $\mathsf{RPDistr}^w_{\mathbf{p}}(\widehat{C},\widehat{x})$:</u> Denote the set of wires in $\widehat{C}$ as $\mathcal{W}$. Consider the computation of $\widehat{C}$ on input encoding $\widehat{x}$. For every wire $w \in \mathcal{W}$, denote $\mathbf{val}(w)$ to be the value assigned to w during the evaluation of $\widehat{C}$ on $\widehat{x}$.

We construct the set $\mathcal{S}_{\mathsf{leak}}$ as follows: initially $\mathcal{S}_{\mathsf{leak}}$ is assigned to be $\{\}$. For every $w \in \mathcal{W}$, with probability $\mathbf{p}$, include $(w, \mathbf{val}(w))$ in $\mathcal{S}_{\mathsf{leak}}$ (i.e., with probability $(1-\mathbf{p})$, the pair $(w, \mathbf{val}(w))$ is not included). Output $\mathcal{S}_{\mathsf{leak}}$.

We define the notion of partial simulator below.

Definition 14 (Partial Simulator: Random Probing). *A partial simulator* Sim *defined by a deterministic polynomial time algorithm* Sim_1 *and probabilistic polynomial time algorithm* Sim_2 *executes as follows: On input a circuit* $\widehat{C}$*,*

- *Denote $\mathcal{W}$ to be the set of wires in $\widehat{C}$. Construct a set $\mathcal{W}_{lk}$ as follows: include every wire $w \in \mathcal{W}$ in the set $\mathcal{W}_{lk}$ with probability $\mathbf{p}$.*
- $\mathsf{Sim}_1(\widehat{C}, \mathcal{W}_{lk})$ *outputs $(\mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. $\mathcal{W}^{inp}$ is a subset of input wires, $\mathcal{W}^{out}$ is a subset of output wires and I denotes a set of indices.*
- *For every wire $w \in \mathcal{W}^{inp}$, include $(w, v_w) \in S^{inp}$ such that v_w is a bit sampled uniformly at random. Similarly, construct the set S^{out}.*
- $\mathsf{Sim}_2\left(\widehat{C}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I\right)$ *outputs $\mathcal{S}_{lk}$.*

Finally, Sim *outputs $\mathcal{S}_{lk}$.*

We now define the notion of composable security in the random probing model.

Definition 15 (Composable Security: Random Probing). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for $\mathcal{C}$, consisting of circuits of input length ℓ, is said to be* $(\mathbf{p}, \varepsilon)$**-composable secure** *against random probing attacks if there exists a probabilistic polynomial time partial simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that the following holds:*

- **p-Partial Simulation:** *for every circuit $C \in \mathcal{C}$, input $x \in \{0,1\}^{\ell}$,*

$$\left\{\mathsf{RPDistr}^w_{\mathbf{p}}\left(\widehat{C},\widehat{x}\right)\right\} \equiv \left\{\mathsf{Sim}(\widehat{C})\big|_{L \leftarrow \mathsf{Sim}(\widehat{C}) \wedge L \neq \perp}\right\},$$

  where, $\widehat{C} \leftarrow \mathsf{Compile}(C)$ and $\widehat{x} \leftarrow \mathsf{Encode}(x)$. That is, conditioned on the simulator not aborting, its output distribution is identical to $\mathsf{RPDistr}^w_{\mathbf{p}}(\widehat{C},\widehat{x})$.
- ε**-Simulation with Abort:** *For every $C \in \mathcal{C}$, $\mathsf{Sim}(\widehat{C})$ aborts with probability ε.*

[7] The superscript w is used to signify leakage of wire values.

Composable Security (Worst Case Probing). We define the composable security in the worst case probing setting. This will be defined along the same lines as in the random probing setting.

Intuitively, we want to capture the following guarantee: simulation of a subset of wires in the circuit can be carried out given a subset of input wire values and a subset of output wire values. We formalize this in terms of partial simulator below.

Definition 16 (Partial Simulator: Worst Case Probing). *A partial simulator* Sim*, associated with a parameter* $\mathbf{t}$*, defined by a deterministic polynomial time algorithm* Sim_1 *and probabilistic polynomial time algorithm* Sim_2 *executes as follows: On input a circuit* $\widehat{C}$ *and a set of wires* $\mathcal{W}_{lk}$ *of size at most* $\mathbf{t}$*,*

- $\mathsf{Sim}_1(\widehat{C}, \mathcal{W}_{lk})$ *outputs* $(\mathcal{W}^{inp}, \mathcal{W}^{out})$*. The sets* $\mathcal{W}^{inp}$ *and* $\mathcal{W}^{out}$ *(of size at most* $\mathbf{t}$*) respectively denote the subset of input and output wires whose values are necessary to simulate the values of the wires in* $\mathcal{W}_{lk}$*.*
- *For every wire* $w \in \mathcal{W}^{inp}$*, include* $(w, v_w) \in S^{inp}$ *such that* v_w *is a bit sampled uniformly at random. Similarly, construct the set* S^{out}*.*
- $\mathsf{Sim}_2\left(\widehat{C}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}\right)$ *outputs* $\mathcal{S}_{lk}$*.*

Finally, Sim *outputs* $\mathcal{S}_{lk}$*.*

We now define the notion of composable security in the context of worst case probing. Before that, we formalize the leakage function L_{comp} defined in the previous section in terms of the following algorithm $\mathsf{WCDistr}^{\mathsf{w}}_{S}$, parameterized by a $\mathbf{t}$-sized set S.

<u>SAMPLER $\mathsf{WCDistr}^{\mathsf{w}}_{S}(\widehat{C}, \widehat{x})$:</u> On input circuit $\widehat{C}$, input encoding $\widehat{x}$, construct the set $\mathcal{S}_{\mathsf{leak}}$ as follows: For every wire $w \in \widehat{C}$, let v_w be the value assigned to the wire w during the execution of $\widehat{C}$ on $\widehat{x}$. Include (w, v_w) in $\mathcal{S}_{\mathsf{leak}}$ for every $w \in S$. Output $\mathcal{S}_{\mathsf{leak}}$.

Definition 17 (Composable Security: Worst Case Probing). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a class of circuits* $\mathcal{C}$ *is said to be* **t-composable secure** *against* $\mathbf{t}$*-probing attacks if there exists a probabilistic polynomial time partial simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$*, associated with a parameter* $\mathbf{t}$*, such that the following holds:*

- **t-Partial Simulation:** *for every circuit* $C \in \mathcal{C}$*, input* $x \in \{0,1\}^{\ell}$*,*

$$\left\{\mathsf{WCDistr}^{\mathsf{w}}_{\mathcal{W}_{lk}}\left(\widehat{C}, \widehat{x}\right)\right\} \equiv \left\{\mathsf{Sim}(\widehat{C}, \mathcal{W}_{lk})\right\},$$

  where $\widehat{C} \leftarrow \mathsf{Compile}(C)$*,* $\widehat{x} \leftarrow \mathsf{Encode}(x)$ *and* $\mathcal{W}_{lk}$ *is any subset of wires in* $\widehat{C}$ *of size at most* $\mathbf{t}$*.*

Main Definition. We now give definitions of composable circuit compilers for the random probing and the worst case probing models.

Definition 18 (Composable Circuit Compilers: Random Probing). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *is said to be a* $(\mathbf{p}, \varepsilon)$*-secure composable circuit compiler in the random probing model if* CC *satisfies:*

- *XOR encoding property.*
- $(\mathbf{p}, \varepsilon)$*-composable security.*

We refer to CC *as a secure composable circuit compiler and in particular, omit* $(\mathbf{p}, \varepsilon)$ *if this is clear from the context.*

Definition 19 (Composable Circuit Compilers: Worst Case Probing). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *is said to be a* $\mathbf{t}$*-secure composable circuit compiler in the worst case probing model if* CC *satisfies:*

- *XOR encoding property.*
- $\mathbf{t}$*-composable security.*

We refer to CC *as a secure composable circuit compiler and in particular, omit* $\mathbf{t}$ *if this is clear from the context.*

L-efficient Composable CC. En route to constructing composable circuit compiler, we construct an intermediate composable circuit compiler that produces exponentially sized compiled circuits. We define the following notion to capture this step.

Definition 20 (L-efficient Composable CC). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *is an* L*-efficient composable circuit compiler for a class of circuits* $\mathcal{C}$ *if for every* $C \in \mathcal{C}$*, we have* $|\widehat{C}| \leq L(|C|)$*, where* $\widehat{C} \leftarrow \mathsf{Compile}(C)$*.*

In particular, CC *is a composable circuit compiler if* L *is a polynomial.*

4.2 Base Case: Constant Simulation Error

We construct a composable circuit compiler $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ for a class of circuits $\mathcal{C}$. Let Π be a perfectly semi-honest secure n-party computation protocol for an n-party randomized[8] functionality $F = F[C]$ (defined in Fig. 1) tolerating t number of corruptions.

[8] Recall that a randomized n-party functionality is one that in addition to taking n inputs, also takes as input randomness.

n-party functionality, $F[C]$

Input: $(\widehat{x}_1^1||\cdots||\widehat{x}_1^\ell \ ; \ \cdots \ ; \ \widehat{x}_n^1||\cdots||\widehat{x}_n^\ell)$, where ℓ is the input length of C.

– It then computes $x_i = \oplus_{j=1}^n \widehat{x}_j^i$ for every $i \in [\ell]$. Denote x to be a bit string, where the i^{th} bit of x is x_i.
– It then computes $C(x)$ to obtain y. Let y_i be the i^{th} output bit of y. Let the length of y be ℓ_y.
– Sample bits $\widehat{y}_j^i$ uniformly at random such that $y_i = \oplus_{j=1}^n \widehat{y}_j^i$ for every $i \in [\ell_y]$. Set $\widehat{\mathbf{y}}^i = (\widehat{y}_1^i, \ldots, \widehat{y}_n^i)$, for every $i \in [n]$. Output $(\widehat{\mathbf{y}}^1, \ldots, \widehat{\mathbf{y}}^{\ell_y})$.

Fig. 1. Functionality $F[C]$, parameterized by a circuit C.

We describe the scheme below.

Circuit Compilation, $\mathsf{Compile}(C)$: This algorithm takes as input circuit $C \in \mathcal{C}$. We associate a boolean circuit Ckt_Π with Π such that the following holds:

– Protocol Π on input $(\widehat{\mathbf{x}}^1; \ldots; \widehat{\mathbf{x}}^n)$, where $\widehat{\mathbf{x}}^i$ is i^{th} party's input, outputs $(\widehat{\mathbf{y}}^1; \ldots; \widehat{\mathbf{y}}^n)$ if and only if Ckt_Π on input $\widehat{\mathbf{x}}^1||\cdots||\widehat{\mathbf{x}}^n$ outputs $(\widehat{\mathbf{y}}^1; \ldots; \widehat{\mathbf{y}}^n)$.
– Furthermore, the gates of Ckt_Π can be partitioned into n sub-circuits such that the i^{th} sub-circuit implements the i^{th} party in Π. Denote the i^{th} sub-circuit to be Ckt_i. Also, denote the number of gates in Ckt_Π to be $\mathsf{N_g}$.
– The wires between the sub-circuits are analogous to the communication channels between the corresponding parties.

Output $\widehat{C} = \mathsf{Ckt}_\Pi$.

Input encoding, $\mathsf{Encode}(x)$: On input $x \in \{0,1\}^\ell$, it outputs the encoding $\widehat{x} = ((x_1^1, \ldots, x_\ell^1), \ldots, (x_1^n, \ldots, x_\ell^n))$, where $x_i = \oplus_{j=1}^n x_i^j$.

Output decoding, $\mathsf{Decode}(\widehat{y})$: It takes as input encoding $\widehat{y} = ((y_1^1, \ldots, y_{\ell'}^1), \ldots, (y_1^n, \ldots, y_{\ell'}^n))$. It then outputs y, where the i^{th} bit of y is $y_i = \oplus_{j=1}^n y_i^j$.

We prove the following two propositions in the full version.

Proposition 1 (Worst Case Probing). *Let Π be a perfectly semi-honest secure n-party computation protocol for n-party functionality F (defined in Fig. 1) tolerating t corruptions and having randomness locality d. Then, CC is a $\mathbf{t}$-secure composable circuit compiler secure against $\mathbf{t}$-probing attacks. Moreover, the randomness locality of CC is d.*

Proposition 2 (Random Probing). *Let Π be a perfectly semi-honest secure n-party computation protocol for n-party functionality F (defined in Fig. 1)* *tolerating t corruptions and having randomness locality d. Then there is a constant $\mathbf{p} > 0$ such that* CC *is a* $(\mathbf{p}, \varepsilon_0)$*-secure composable circuit compiler, where* $\varepsilon_0 = e^{-\frac{(1+t)^2}{12\mathrm{N_g}} \cdot \frac{1}{\mathbf{p}}}$. *Moreover, the randomness locality of* CC *is d.*

4.3 Composition Step

We present the main composition step in this section. It allows for transforming a composable circuit compiler CC_K satisfying $(\mathbf{p}, \varepsilon_K)$-composable security in the random probing setting (resp., $\mathbf{t}_K$-composable security in the worst case) into CC_{K+1} satisfying $(\mathbf{p}, \varepsilon_{K+1})$-composable security (resp., $t \cdot \mathbf{t}_K$-composable security in the worst case), where ε_{K+1} is (exponentially) smaller than ε_K. In terms of efficiency, the efficiency of CC_{K+1} degrades by a constant factor. The main tool we use to prove the composition theorem is a perfectly secure MPC protocol that tolerates at most t corruptions.

We first present the transformation of CC_K into CC_{K+1}. Let $\mathsf{CC}_K = (\mathsf{Compile}_K, \mathsf{Encode}_K, \mathsf{Decode}_K)$ be a composable circuit compiler. We now build CC_{K+1} as follows:

Circuit Compilation, $\mathsf{CC}_{K+1}.\mathsf{Compile}(C)$: It takes as input a circuit C and outputs a compiled circuit $\widehat{C}$. There are two steps involved in the construction of $\widehat{C}$. In Step I, we first consider a MPC protocol Π[9] for a randomized functionality F and using this we construct a circuit Ckt_Π. In Step II, we convert Ckt_Π into another circuit Ckt^*_Π. In this step, we make use of the compiler CC_K. The output of this algorithm is $\widehat{C} = \mathsf{Ckt}^*_\Pi$.

Step I: Constructing Ckt_Π. Consider a n-party functionality $F = F[C]$; see Fig. 1.

Let Π denote a n-party information theoretically secure computation protocol for F. Construct Ckt_Π as done in Sect. 4.2.

Step II: Transforming Ckt_Π into Ckt^*_Π. Replace every gate in Ckt_Π with the CC_K gadgets and then show how to "stitch" all these gadgets together.

- Replacing Gate by CC_K gadget: For every gate G in the circuit Ckt_Π, we execute the compiler $\mathsf{CC}_K.\mathsf{Compile}(G)$ to obtain $\widehat{G}$.
- "Stitching" Gadgets: We created CC_K gadgets for every gate in the circuit. Now we show how to connect these gadgets with each other.

Let G_k be a gate in Ckt_Π. Let G'_k and G''_k be two gates such that the output wires from these two gates are inputs to G_k. Let $\widehat{G_k} \leftarrow \mathsf{CC}_K.\mathsf{Compile}(G_k)$, $\widehat{G'_k} \leftarrow \mathsf{CC}_K.\mathsf{Compile}(G'_k)$ and $\widehat{G''_k} \leftarrow \mathsf{CC}_K.\mathsf{Compile}(G''_k)$. We connect the output of $\widehat{G'_k}$ and $\widehat{G''_k}$ with the input of $\widehat{G_k}$. That is, the output encodings of $\widehat{G'_k}$ and $\widehat{G''_k}$ form

[9] The parties in this protocol are equipped with randomness gates.

the input encoding to $\widehat{G_k}$. Here, we use the fact that the output encoding and the input encoding are computed using the same secret sharing scheme, and in particular we use the XOR secret sharing scheme.

We perform the above operation for every gate in Ckt_{Π}.
We denote the result of applying Step I and II to Ckt_{Π} to be the circuit Ckt^*_{Π}. Furthermore, we denote Ckt^*_i to be the circuit obtained by applying Steps I and II to sub-circuits Ckt_i. Note that Ckt^*_i is a sub-circuit of Ckt_{Π}. Moreover, Ckt^*_i takes as input XOR secret sharing of the i^{th} party's input and outputs XOR secret sharing of the i^{th} party's output.

Output $\widehat{C} = \mathsf{Ckt}^*_{\Pi}$.

Input Encoding, $\mathsf{CC}_{K+1}.\mathsf{Encode}(x)$: On input x, compute $(x_{1,1}, \ldots, x_{\ell,1}), \ldots, (x_{1,n}, \ldots, x_{\ell,n}))$, where $x_i = \oplus_{j=1}^{n} x_{i,j}$. Compute $\widehat{x_{i,j}} \leftarrow \mathsf{CC}_K.\mathsf{Encode}(x_{i,j})$, for every $i \in [\ell]$ and $j \in [n]$. Output $\left(\{\widehat{x_{i,j}}\}_{i\in[\ell],j\in[n]}\right)$.

Output Encoding, $\mathsf{CC}_{K+1}.\mathsf{Decode}(\widehat{y})$: On input $\left(\{\widehat{y_{i,j}}\}_{i\in[\ell'],j\in[n]}\right)$, first compute $\mathsf{CC}_K.\mathsf{Decode}(\widehat{y_{i,j}})$ to obtain $y_{i,j}$, for every $i \in [\ell'], j \in [n]$. It computes y, where the the i^{th} bit of the output is computed as $y_i = \oplus_{j=1}^{n} \widehat{y}^i_j$. Output $y = y_1 || \cdots || y_n$.

We prove the following two propositions in the full version.

Proposition 3 (Worst Case Probing). *Suppose* CC_K *is* $\mathbf{t}_K$*-composable secure against* $\mathbf{t}_K$*-probing attacks and* Π *is perfectly secure tolerating* t *number of corruptions. Then,* CC_{K+1} *is* $t \cdot \mathbf{t}_K$*-composable secure against* $\mathbf{t}$*-probing attacks. If* CC_K *has randomness locality* d_K *and* Π *has randomness locality* d *then* CC_{K+1} *has randomness locality* $2d + d_K$*.*

Proposition 4 (Random Probing). *Let* CC_K *satisfy* $(\mathbf{p}, \varepsilon_K)$*-composable security property. Then,* CC_{K+1} *satisfies* $(\mathbf{p}, \varepsilon_{K+1})$*-composable security property, where* $\varepsilon_{K+1} = e^{-\frac{(1+t)^2}{12\mathsf{N_g}} \cdot \frac{1}{\varepsilon_K}}$*. If* CC_K *has randomness locality* d_K *and* Π *has randomness locality* d *then* CC_{K+1} *has randomness locality* $2d + d_K$*.*

4.4 Stitching Transformation: Exp to Poly Efficiency

Consider a L_{exp}-efficient composable circuit compiler $\mathsf{CC}_{\mathsf{exp}}$ for a basis of gates $\mathbb{B}$, where L_{exp} is a exponential function. We construct a L_{poly}-efficient composable circuit compiler $\mathsf{CC}_{\text{poly}}$ for a class of all circuits $\mathcal{C}$ over the basis $\mathbb{B}$, where L_{poly} is a polynomial.

We describe the construction below.

Circuit compilation, $\mathsf{CC}_{\text{poly}}.\mathsf{Compile}(C)$: It takes as input circuit $C \in \mathcal{C}$. For every gate G in C, it computes $\widehat{G} \leftarrow \mathsf{CC}_{\text{exp}}.\mathsf{Compile}(G)$ to obtain the gadget $\widehat{G}$.

Once it computes all the gadgets, it then 'stitches' all the gadgets together. The stitching operation is performed as follows: let G_k be a gate in C. Let G'_k and G''_k be two gates such that the output wires from these two gates are inputs to G_k. We connect the output of $\widehat{G'_k}$ and $\widehat{G''_k}$ with the input of $\widehat{G_k}$. That is, the output encodings of $\widehat{G'_k}$ and $\widehat{G''_k}$ form the input encoding to $\widehat{G_k}$. Here, we use the fact that the output encoding and the input encoding are computed using the same secret sharing scheme, i.e., the XOR secret sharing scheme. Denote the resulting circuit obtained after stitching all the gadgets together to be $\widehat{C}$. Output $\widehat{C}$.

Input Encoding, $\mathsf{CC}_{\mathrm{poly}}.\mathsf{Encode}(x)$: It takes as input x and then computes the XOR secret sharing of every bit of x. Output the concatenation of the XOR secret shares of all the bits of x, denoted by $\widehat{x}$.

Output Decoding, $\mathsf{CC}_{\mathrm{poly}}.\mathsf{Decode}(\widehat{y})$: On input $\widehat{y}$, parse it as $((\widehat{y}_1^1, \ldots, \widehat{y}_n^1), \ldots, (\widehat{y}_1^{\ell'}, \ldots, \widehat{y}_n^{\ell'}))$. Reconstruct the i^{th} bit of the output as $y_i = \oplus_{j=1}^{n} \widehat{y}_j^i$. Output $y = y_1 || \cdots || y_n$.

We prove the following two propositions in the full version.

Proposition 5 (Worst Case Probing). *Suppose* $\mathsf{CC}_{\mathsf{exp}}$ *satisfies* $\mathbf{t}$*-composable security. Then* $\mathsf{CC}_{\mathrm{poly}}$ *satisfies* $\mathbf{t}$*-composable security. If* $\mathsf{CC}_{\mathsf{exp}}$ *has randomness locality* d *then* $\mathsf{CC}_{\mathrm{poly}}$ *has randomness locality* d.

Proposition 6 (Random Probing). *Let* $\mathsf{CC}_{\mathsf{exp}}$ *satisfies* $(\mathbf{p}, \varepsilon_{\mathrm{exp}})$*-composable security.* $\mathsf{CC}_{\mathrm{poly}}$*, associated with circuits of size* s*, satisfies* $(\mathbf{p}, s \cdot \varepsilon_{\mathrm{exp}})$*-composable security. If* $\mathsf{CC}_{\mathsf{exp}}$ *has randomness locality* d *then* $\mathsf{CC}_{\mathrm{poly}}$ *has randomness locality* d.

4.5 Main Construction: Formal Description

We now combine all the components we developed in the previous sections to obtain a construction of composable circuit compiler. In particular, the main construction consists of the following main steps:

- Start with a secure MPC protocol Π for a constant number of parties.
- Apply the base case compiler to obtain a composable circuit compiler, which has constant simulation error in the case of random probing model and tolerates constant threshold in the case of worst case probing model.
- Recursively apply the composition step on the base compiler obtain from the above bullet. The resulting compiler, after sufficiently many iterations, satisfies negligible error in the random probing setting and satisfies a large threshold in the case of worst case probing model.
- The disadvantage with the compiler resulting from the previous step is that the size of the compiled circuit could be exponentially larger than the original circuit. To improve the efficiency from exponential to polynomial, we apply the exponential-to-polynomial transformation.

Proof: Worst Case Probing
We sketch the construction in Fig. 2.

Construction of CC_{main}

- **Circuit compilation, $\mathsf{CC}_{main}.\mathsf{Compile}(C)$**: On input a circuit C, it executes the following steps:
  - It transforms Π into a composable circuit compiler $\mathsf{CC}_{\mathsf{base}}$ satisfying t-composable security, where $\mathbf{t} = t$ and L_1-efficiency.
  - Set $\mathsf{CC}_1 = \mathsf{CC}_{\mathsf{base}}$ with $\mathbf{t}_0 = t$. Repeat the following process for $i = 1, \ldots, K$: Using the composition theorem, satisfying $\mathbf{t}_i$-composable security, it transforms CC_i into a composable circuit compiler CC_{i+1} satisfying $\mathbf{t}_{i+1}$-composable security. Moreover, $\mathbf{t}_K = t^K$.
  - It transforms CC_K into a composable circuit compiler CC^* satisfying $f \cdot L_1^K(k)$-efficiency and t^K-composable security property, where f is a linear function.
  - It finally executes $\mathsf{CC}^*(C)$ to obtain the compiled circuit $\widehat{C}$.
  - Output $\widehat{C}$.
- **Input encoding, $\mathsf{CC}_{main}.\mathsf{Encode}(x)$**: It computes the XOR secret sharing of every bit of x. Output the concatenation of the XOR secret shares of all the bits of x, denoted by $\widehat{x}$.
- **Output encoding, $\mathsf{CC}_{main}.\mathsf{Decode}(\widehat{y})$**: It reconstructs the XOR secret sharing of every bit of y. Output y.

Fig. 2. Construction of CC_{main}

Proposition 7. *Let $K \in \mathbb{N}$. Consider a MPC protocol Π for a n-party functionality F (Fig. 1) and tolerating at most t with randomness locality d. Then, CC_{main} is a t^K-composable secure composable circuit compiler secure against worst case probing attacks for all circuits satisfying $(L_1(k))^K \cdot f$-efficiency, where:*

- *$L_1(k)$ is a constant and f is a linear function,*
- *c is a constant,*
- *Moreover, the randomness locality of CC_{main} is $O(K)$.*

Instantiation. By instantiating the tools in the above proposition, we get the following proposition.

Proposition 8. *Consider a parameter* $\mathbf{t} > 0$. *There is a composable circuit compiler satisfying* $\mathbf{t}$*-composable security against worst case probing attacks satisfying randomness locality* $O(\log(\mathbf{t}))$.

Proof. Suppose we have a MPC protocol Π for the n-party functionality F (Fig. 1) tolerating at most t corruptions, for some constant n (for instance, [BOGW88, CCD88]). We then obtain a circuit compiler CC_{main}, which is t^K-composable secure and satisfy $c^K \cdot f$-efficiency, where c is a constant and f is a linear function. Setting $K = \lceil \frac{\log(\mathbf{t})}{\log(t)} \rceil$, we have that CC_{main} is $\mathbf{t}$-composable secure and satisfying polynomial efficiency, as desired. Moreover, the randomness locality of CC_{main} is $O(K) = O(\log(\mathbf{t}))$. This completes the proof.

We present the constructions in the worst case and random probing models below. The proofs are deferred to the full version.

Proof: Random Probing. We now present a construction (Fig. 3) of composable circuit compiler for a class of circuits $\mathcal{C}$ over basis $\mathbb{B}$ starting from a MPC protocol Π for the n-party functionality F that can tolerate t semi-honest adversaries. We denote this construction by CC_{main}.

Proposition 9. *Let* $K \in \mathbb{N}$. *Consider a MPC protocol* Π *for a* n*-party functionality* F *and tolerating at most* t *corruptions with randomness locality* d *satisfying the property that* $e^{\frac{12\mathsf{N_g}}{(1+t)^2}} \geq \left(\frac{12\mathsf{N_g}}{(1+t)^2}\right)^4$, *where* $\mathsf{N_g}$ *is the number of gates in the implementation of* Π.

Then, CC_{main} *is a* $(\mathbf{p}, c^{c^K})$*-secure composable circuit compiler for all circuits satisfying* $(L_1(k))^K \cdot f$*-efficiency, where:*

- $\mathbf{p} = \frac{(1+t)^2}{48\mathsf{N_g}\ln(\frac{12\mathsf{N_g}}{(1+t)^2})}$
- $L_1(k)$ *is a constant and* f *is a linear function,*
- c *is a constant,*
- $\mathsf{N_g}$ *is the number of gates in the circuit* Ckt_Π

Moreover, the randomness complexity of CC_{main} *is* $O(K)$.

Instantiation. We use a specific instantiation of the MPC protocol in the above proposition to get the following result.

Proposition 10. *There is a construction of a composable circuit compiler for* $\mathcal{C}$ *satisfying* $(\mathbf{p}, \mathsf{negl})$*-composable security, where* $\mathbf{p} = 6.5 \times 10^{-5}$.

5 Leakage Tolerant Circuit Compilers

In this section, we present a construction of leakage tolerant circuit compiler with constant leakage rate. Later, we present a negative result on the leakage rate of a leakage tolerant circuit compiler.

Construction of CC_{main}

- **Circuit compilation,** $\mathsf{CC}_{main}.\mathsf{Compile}(C)$: On input a circuit C, it executes the following steps:
  - It transforms Π into a composable circuit compiler $\mathsf{CC}_{\mathsf{base}}$ satisfying $(\mathbf{p}, \varepsilon_1)$-composable security, where $\varepsilon_1 = e^{-\frac{(1+t)^2}{12\mathsf{N_g}} \cdot \frac{1}{\mathbf{p}}}$ and L_1-efficiency.
  - Set $\mathsf{CC}_1 = \mathsf{CC}_{\mathsf{base}}$. Repeat the following process for $i = 1, \ldots, K$: Using the composition step, it transforms CC_i into a composable circuit compiler CC_{i+1} satisfying $(\mathbf{p}, \varepsilon_{i+1})$-security.
  - Using the exponential-to-polynomial transformation, it transforms CC_K into a composable circuit compiler CC^* satisfying $f \cdot L_1^K(k)$-efficiency and $(\mathbf{p}, s \cdot \varepsilon_K)$-composable security property, where f is a linear function.
  - It finally executes $\mathsf{CC}^*(C)$ to obtain the compiled circuit $\widehat{C}$.
  - Output $\widehat{C}$.
- **Input encoding,** $\mathsf{CC}_{main}.\mathsf{Encode}(x)$: It computes the XOR secret sharing of every bit of x. Output the concatenation of the XOR secret shares of all the bits of x, denoted by $\widehat{x}$.
- **Output encoding,** $\mathsf{CC}_{main}.\mathsf{Decode}(\widehat{y})$: It reconstructs the XOR secret sharing of every bit of y. Output y.

Fig. 3. Construction of CC_{main}

5.1 Construction: Random Probing

We prove the following proposition.

Proposition 11. *Let CC_{comp} be a composable compiler for a class of circuits $\mathcal{C}$ satisfying $(\mathbf{p}, \varepsilon)$-composable security. Then, CC_{LT} is a $(\mathbf{p}, \mathbf{p}', \varepsilon')$-leakage tolerant circuit compiler for $\mathcal{C}$ secure against random probing attacks, where $\mathbf{p}' = (1+\eta)^2 \left(1 - (1-\mathbf{p})^6\right)$ and $\varepsilon' = \varepsilon + \frac{1}{e^{c \cdot n}}$, for arbitrarily small constant $\eta > 0$.*

To prove the above theorem, we start with a composable secure circuit compiler and then attach a leakage tolerant circuit that computes the additive shares of input. In particular, we need to prove that the leakage of values in the sharing circuit can be simulated with leakage on the input bits.
Combining with Proposition 10 obtain the following proposition.

Proposition 12. *Consider a basis $\mathbb{B}$. There is a construction of $(\mathbf{p}, \mathbf{p}', \mathsf{negl})$-leakage tolerant circuit compiler against random probing attacks for all circuits over $\mathbb{B}$ of size s, where $\mathbf{p} = 6.5 \times 10^{-5}$ and $\mathbf{p}' = 3.9 \times 10^{-4}$.*

Non-Boolean Basis. We show how to achieve a leakage tolerant compiler with leakage rate arbitrarily close to 1 with the compiled circuit defined over a non-boolean basis. The starting point is a composable circuit compiler where the compiled circuit with leakage rate arbitrarily close to 1 and over a large basis.

Proposition 13. *Let* $\delta > 0$*. Consider a basis* $\mathbb{B}'$ *consisting of all randomized functions mapping* n *bits to* n *bits. Suppose there is a construction of a composable circuit compiler* $\mathsf{CC}_{\mathsf{NB}}$ *over* $\mathbb{B}'$ *for* $\mathcal{C}$ *over* $\mathbb{B}$ *satisfying* $(\mathbf{p}, \varepsilon)$*-composable security. Then there is a construction of* $(\mathbf{p}, \mathbf{p}', \varepsilon')$*-secure leakage tolerant circuit compiler over* $\mathbb{B}'$ *for* $\mathcal{C}$ *over* $\mathbb{B}$*, where* $\mathbf{p}' = 1 - ((1-\mathbf{p})^2) \cdot (1-\mathbf{p}^n)^2)$ *and* $\varepsilon' = \varepsilon + \frac{1}{e^{c \cdot n}}$*, for some constant* c*.*

5.2 Construction: Worst Case Probing

We present the construction of a leakage tolerant circuit compiler in the worst case probing model.

Proposition 14. *For any basis* $\mathbb{B}$ *and any* $\mathbf{t} > 0$*, there is a construction of leakage tolerant circuit compiler secure against* $\mathbf{t}$*-probing attacks. Moreover, this compiler has randomness locality* $O(\log(\mathbf{t}))$*.*

Proof. From Proposition 8, there is a construction of $\mathbf{t}$-secure composable circuit compiler CC_{comp}. We construct a leakage tolerant circuit compiler CC_{LT} as follows:

- $\mathsf{Compile}(C)$: On input C, it does the following:
  - Compute $\mathsf{CC}_{comp}.\mathsf{Compile}(C)$ to obtain the compiled circuit $\mathsf{CC}_{comp}.\widehat{C}$.
  - Constructs a circuit $\widehat{C}$ that takes as input x,
    - ∗ Computes N shares of every bit of x, where N is determined the input length of $\mathsf{CC}_{comp}.\widehat{C}$. In particular, for every i, it computes shares of x_i as follows: $(x_i \oplus r_1, r_1 \oplus r_2, \ldots, r_{N-2} \oplus r_{N-1}, r_{N-1})$, where r_i is sampled freshly at random. For every i^{th} bit, since there are two input wires carrying x_i, we perform the sharing process twice.
    - ∗ Compute $\mathsf{CC}_{comp}.\widehat{C}$ on the shares of x as computed in the bullet above.
- $\mathsf{Decode}(\widehat{y})$: It parses $\widehat{y}$ as $(\widehat{y}^1, \ldots, \widehat{y}^\ell)$ and reconstructs the shares in $\widehat{y}^i$ to obtain the value y_i.

We claim that CC_{comp} is a $\mathbf{t}$-secure leakage tolerant circuit compiler. The correctness and efficiency properties of CC_{comp} follow from the respective properties of CC_{LT}. To argue security, we first note that any $\mathbf{t}$ wires of leakage in the sharing circuit can be simulated with $\mathbf{t}$ input and output wires of leakage of the sharing circuit (this follows from the fact that every wire in the sharing circuit is either an input or an output wire). The $\mathbf{t}$-composable security of CC_{comp} then implies the security of CC_{LT}.

Next, we show that CC_{comp} has randomness locality $O(\log(\mathbf{t}))$. We first note that the sharing circuit has constant randomness locality. This combined with the fact that CC has $O(\log(\mathbf{t}))$ randomness locality proves the result.

6 Leakage Resilient Circuit Compilers

In this section, we give upper bounds for leakage resilient circuit compilers. Note that any structural circuit compiler for circuit class $\mathcal{C}$ is also a leakage resilient circuit compiler for $\mathcal{C}$. Using this fact, we state the following theorem.

Theorem 8. *There is a construction of* $(\mathbf{p}, \exp(-s))$*-leakage resilient circuit compiler for all circuits over* $\mathbb{B}$ *of size* s*, secure against random probing attacks, where* $\mathbf{p} = 6.5 \times 10^{-5}$*.*

The proof of the above theorem follows from Proposition 10.

7 Randomness Complexity

We present a construction of leakage tolerant circuit compiler with near optimal randomness complexity. To show this, we use two lemmas from [IKL+13]. We first state a lemma about the existence of explicit robust r-wise PRGs. We refer the reader to [IKL+13] for the definition of strong $(\mathbf{t}, q)$ robust r-wise PRGs.

Lemma 2 ([IKL+13]). *For any* $\eta > 0$*, there exists* $\delta, c > 0$*, such that for any* $m \leq \exp n^{\delta}$*, there is an explicit* d*-strong* $(n^{1-\eta}, 21)$*-robust* r*-wise independent PRG* $G : \{0,1\}^n \rightarrow \{0,1\}^m$ *for* $r = n^{1-\eta}$ *and* $d \leq \log^c(m)$*.*

The following theorem[10] states that any $\mathbf{t}$-leakage tolerant circuit compiler establishes the connection between randomness locality and randomness complexity.

Lemma 3 ([IKL+13]). *Consider a* $q \cdot \mathbf{t}$*-leakage tolerant circuit compiler. Suppose the compiled circuit uses* m *random bits and makes an* d*-local use of its randomness. Let* $G : \{0,1\}^n \rightarrow \{0,1\}^m$ *be a strong* $(\mathbf{t}, q)$*-robust* r*-wise PRG with* $r \geq \mathbf{t} \cdot \max(d, q)$*. Then there is a leakage tolerant circuit compiler secure against* $\mathbf{t}$*-probing attacks which uses* n *random bits.*

Recall that the leakage tolerant compiler in Theorem 14 has randomness locality $O(\log(\mathbf{t}))$. This fact along with the above two lemmas yields the following theorem.

Theorem 9. *For any* $\mathbf{t} > 0$*, there is a construction of leakage tolerant circuit compiler secure against* $\mathbf{t}$*-probing attacks using* $\mathbf{t}^{1+\varepsilon} \cdot \mathrm{polylog}(|C|)$ *random bits.*

Acknowledgements. We thank Jean-Sébastien Coron, Stefan Dziembowski, and Sebastian Faust for helpful discussions. The second author was supported in part by ERC grant 742754, ISF grant 1709/14, NSF-BSF grant 2015782, and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India. The third author's research is supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty

[10] They phrase this in the language of private circuits and so we rephrase their theorem in our language.

Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[ADF16] Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 586–615. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_21

[Ajt11] Ajtai, M.: Secure computation with information leaking to an adversary. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, pp. 715–724. ACM (2011)

[BBD+16] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 116–129. ACM (2016)

[BBP+16] Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Randomness complexity of private circuits for multiplication. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 616–648. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_22

[BBP+17] Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 397–426. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_14

[Bea91] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34

[BOGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM (1988)

[CCD88] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 11–19. ACM (1988)

[CDI+13] Cohen, G., et al.: Efficient multiparty protocols via log-depth threshold formulae. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 185–202. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_11

[CK91] Chor, B., Kushilevitz, E.: A zero-one law for boolean privacy. SIAM J. Discret. Math. **4**(1), 36–47 (1991)

[DDF14] Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24

[GIK+15] Garg, S., Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with one-way communication. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 191–208. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_10

[GIM+16] Goyal, V., Ishai, Y., Maji, H.K, Sahai, A., Sherstov, A.A.: Bounded-communication leakage resilience via parity-resilient circuits. In: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pp. 1–10. IEEE (2016)

[HM00] Hirt, M., Maurer, U.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000)

[IKL+13] Ishai, Y., et al.: Robust pseudorandom generators. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 576–588. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39206-1_49

[ISW03] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27

[Pip85] Pippenger, N.: On networks of noisy gates. In: FOCS, pp. 30–38 (1985)

[vN56] von Neumann, J.: Probabilistic logics and synthesis of reliable organisms from unreliable components. Autom. Stud. **34**, 43–98 (1956)