Private Circuits: A Modular Approach[*]

Prabhanjan Ananth[†]
CSAIL, MIT

Yuval Ishai[‡]
Technion

Amit Sahai[§]
UCLA

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.

[*]This is an extended and corrected full version of [AIS18].

[†]`prabhanjan@csail.mit.edu`

[‡]`yuvali@cs.technion.ac.il`

[§]`sahai@cs.ucla.edu`

Contents

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 $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$. Given any circuit C, the compiled version of the circuit $\hat{C} = \mathsf{Compile}(C)$ takes a 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 results from [BOGW88, CCD88]. We refer the reader to Section 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 $p < p' < 1$.

[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.

[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.

- 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]

We present our results formally in Section 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.

In the composition-based approach, we start with a leakage-resilient circuit compiler CC_0 secure against $\mathbf{p}$-random probing attacks and that 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 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.

[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.

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. Moreover, 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 (for d steps, where d is the depth of the formula) 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 $\mathrm{NAND}(a, b)$[4]. This secure MPC protocol will be our base gadget for NAND; the security of MPC protocol can be invoked to prove that the base gadget is secure with respect to constant probability of wire leakage and constant simulation error, call it ε_0. We then compose this base gadget recursively as follows: in the k^{th} level of recursion, we start with Π and *emulate* the computation of every gate in Π with the gadget computed using $(k-1)$ levels of recursion, called the inner gadget. The protocol Π and the $(k-1)^{th}$ level gadget offer two layers of protection for the k^{th}-level gadget. Why should this be secure? if all the inner gadgets can always be simulated (i.e, no simulation error) then the resulting k^{th}-level gadget can also always be simulated. Unfortunately, this is not true since the simulator of the inner gadget does fail with probability ε_{k-1}. So far, we have used the security of only layer of protection, we now will use the security of the second layer of protection; i.e., we will invoke the security of Π. The insight here is that we can map the failure of inner gadgets 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.

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

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

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 final template. We first define a special case of leakage-resilient circuit compilers, called *composable* circuit compilers. This notion will incorporate the composition-friendly 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.

Organization. We first present the necessary preliminaries in Section 2. We then define the notion of circuit compilers in Section 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 Section 4.1. We present the construction of composable circuit compilers in the following steps:

- We present the starting step (base case) in the composition step in Section 4.2.
- The composition step itself is presented in Section 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 Section 4.4.
- Finally, we combine all these steps to present the main construction of a composable circuit compiler in Section 4.5.

Armed with a construction of composable circuit compiler, we present a construction of leakage *tolerant* circuit compilers in Section 5. We also present negative results that upper bounds the leakage rate in the random probing model in the same section.

We show implication of composable circuit compilers to leakage *resilient* circuit compilers in Section 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, $\sum_{v \in \mathcal{V}} |\Pr[v \leftarrow \mathsf{Sampler}_1] - \Pr[v \leftarrow \mathsf{Sampler}_2]| \leq 2\varepsilon$.

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 Π for n parties $P_1, \ldots, P_n$ associated with an n-party functionality $F : \{0,1\}^{\ell_1} \times \cdots \times \{0,1\}^{\ell_n} \times \{0,1\}^{\ell_r} \to \{0,1\}^{\ell_{y_1}} \times \cdots \times \{0,1\}^{\ell_{y_n}}$. 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 $\mathsf{Real}^{\Pi}_{F,S}(x_1, \ldots, x_n)$ to be the joint distribution over the outputs of all the parties along with the views of the parties indexed by the set S.

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:*

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

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 multiparty computation protocol in the information-theoretic setting assuming that a majority of the parties are honest.

[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.

3 Circuit Compilers

We define the notion of circuit compilers. This notion allows for transforming an input x, a circuit C (See Section 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 2 (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* $\mathrm{poly}(k, |C|)$*, the running time of* $\mathsf{Encode}(x)$ *to be* $\mathrm{poly}(k, |x|)$ *and the running time of* $\mathsf{Decode}(\widehat{C(x)})$ *to be* $\mathrm{poly}(k, |C(x)|)$*. We emphasize that the encoding complexity only grow poly-logarithmically in terms of the size of* C*. Typically,* k *will be set to* $\mathrm{poly}(\log(|C|))$*.*

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*.*

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 3. *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 4. *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 5 (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* $\mathbf{p}$*.*

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 6. *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* $\mathsf{Sim}\,(C, L_{inp}(x))$*, 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* $(\mathsf{Compile}, \mathsf{Decode})$*.*

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

$(\mathbf{p}, \mathbf{p}')$-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 Section 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 7 (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* $\mathbf{p}$*.*

3.3 Our Results

We state our results[7] below.

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

Theorem 1 (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 2 (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 present this result in Section 3.2.

Theorem 3. *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 Section 6.

Theorem 4 (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.*

In the same section, we present a construction of leakage resilient circuit compiler over finite basis.

Theorem 5 (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 need for composable circuit compilers.

[7]Special thanks to Jean-Sébastien Coron for pointing out an error in our result on the randomness complexity of private circuits (Theorem 1 of our conference version [AIS18]); we have retracted this result from the full version.

- In our composition theorem, we need to 'attach' different circuit compiler gadgets. For instance, the output wires of circuit 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 2). 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 8 (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, $\mathsf{RPDistr}^{w}_{\mathbf{p}}(\cdot,\cdot)$[8].

<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 9 (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 10 (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* $x \in \{0,1\}^{\ell}$*,*

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

  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*.*

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

4.1.1 Main Definition

We now present the definition of composable circuit compiler for the random probing model.

Definition 11 (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.*

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 12 (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[9] functionality $F = F[C]$ (defined in Figure 1) tolerating t number of corruptions with $t \geq 2$.

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 $(\widehat{\mathbf{y}}^1, \ldots, \widehat{\mathbf{y}}^{\ell_y})$.

Figure 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 $C : \{0,1\}^\ell \to \{0,1\}^{\ell'} \in \mathcal{C}$. 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 $(\widehat{\mathbf{y}}^1; \ldots; \widehat{\mathbf{y}}^n)$.

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

- 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 $\widehat{x} = (\widehat{\mathbf{x}}^1; \ldots; \widehat{\mathbf{x}}^n)$, where $\widehat{\mathbf{x}}^j = (\widehat{x}_1^j || \ldots || \widehat{x}_{\ell}^j)$ and $x_i = \oplus_{j=1}^n \widehat{x}_i^j$.

Output decoding, $\mathsf{Decode}(\widehat{y})$: It takes as input encoding $\widehat{y} = (\widehat{\mathbf{y}}^1, \ldots, \widehat{\mathbf{y}}^n)$ and outputs y, where the i^{th} output bit of y is computed as $y_i = \oplus_{j=1}^n \widehat{y}_i^j$ with $\widehat{\mathbf{y}}^j = (\widehat{y}_1^j, \ldots, \widehat{y}_{\ell'}^j)$.

We first prove the correctness and efficiency properties of the above scheme.

Lemma 1. CC *satisfies correctness of encoding and correctness of evaluation properties.*

Proof. The correctness of encoding property follows from the correctness of the XOR secret sharing scheme.

The following bullets proves the correctness of evaluation property: consider an input x and a circuit $C : \{0,1\}^{\ell} \to \{0,1\}^{\ell'}$.

- By construction, the input encoding is a XOR secret sharing of the input x.
- The correctness of protocol Π proves that the output of the evaluation of $\widehat{C}$ on $\widehat{x}$ is a XOR sharing of $C(x)$.
- Thus, by construction, the output of the decoding algorithm is reconstruction of the XOR sharing of $C(x)$.

□

Lemma 2. CC *satisfies the efficiency property.*

Proof. This follows from the fact that the total computational complexity of Π is polynomial in n, ℓ and $|C|$. □

Lemma 3. CC *satisfies n-XOR encoding property.*

Proof. The proof of this lemma follows from the construction of the encoding algorithm. □

We now prove that CC is composable secure against random probing attacks.

Proposition 1. *Let Π be a perfectly semi-honest secure n-party computation protocol for n-party functionality F (defined in Figure 1) tolerating t corruptions, with $t \geq 2$. Then,* CC *is a $(\mathbf{p}, \varepsilon_0)$-secure composable circuit compiler, where $\varepsilon_0 = (\mathsf{N_g}\mathbf{p})^{t+1}$.*

Proof. We already proved the correctness and efficiency properties of CC earlier. It suffices to prove the $(\mathbf{p}, \varepsilon_0)$-composable security of CC.

Consider a circuit $C \in \mathcal{C}$ with input length ℓ and let $x \in \{0,1\}^{\ell}$. Let $\widehat{C} \leftarrow \mathsf{Compile}(C)$ and let $\widehat{x} \leftarrow \mathsf{Encode}(x)$. Let Ckt_i denotes the sub-circuit that implements the i^{th} party.

We first describe a partial simulator, denoted by $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$. This will be defined along the lines of partial simulator in the worst case setting.

$\mathsf{Sim}(\widehat{C})$: It takes as input compiled circuit $\widehat{C}$ and does the following: Let $\mathcal{W}$ be the set of wires in $\widehat{C}$. Construct a set of leaked wires $\mathcal{W}_{lk}$ as follows: include every wire $w \in \mathcal{W}_{lk}$ with probability $\mathbf{p}$. It then executes

$\mathsf{Sim}_1(\widehat{C}, \mathcal{W}_{lk})$, which is defined below.

$\mathsf{Sim}_1(\widehat{C}, \mathcal{W}_{lk})$: It takes as input compiled circuit $\widehat{C}$ and a set of leaked wires $\mathcal{W}_{lk}$. The first step is to calculate the set of sub-circuits of $\widehat{C}$ that are compromised. Recall that $\widehat{C}$ can be partitioned into sub-circuits $Ckt_1, \ldots, Ckt_n$, where Ckt_i is the i^{th} sub-circuit implementing the i^{th} party P_i. Construct a set $I \subseteq [n]$. Include $i \in [n]$ in the set I if and only if there exists a wire $w \in Ckt_i$ such that $w \in \mathcal{W}_{lk}$.

Now construct the set of input and output wires that need to be additionally leaked to carry out the simulation. Construct $\mathcal{W}^{inp}$ as follows: include $w \in \mathcal{W}$ in the set $\mathcal{W}^{inp}$ if and only if w is an input wire in Ckt_i and $i \in I$. Similarly construct the set $\mathcal{W}^{out}$.

Output the set $(\mathcal{W}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$.

Once Sim_1 is executed, construct a set S^{inp} as follows: for every wire $w \in \mathcal{W}^{inp}$, sample a uniformly random bit v_w and include $(w, v_w) \in S^{inp}$. Similarly, construct the set S^{out}.

$\mathsf{Sim}_2(\widehat{C}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)$: It first checks if $|I| \geq t+1$ and if the check passes, it aborts. Otherwise, define a probabilistic polynomial time semi-honest adversary $\mathcal{A}_{\mathsf{MPC}}$ for Π as follows: it corrupts party P_i, for every $i \in I$. Upon termination of the protocol, it outputs the computation tableau of all parties P_i, for $i \in I$. Now, the security of Π guarantees that there exists a simulator $\mathsf{Sim}_{\mathsf{MPC}}$ such that it simulates $\mathcal{A}_{\mathsf{MPC}}$ in the ideal world. The output of $\mathsf{Sim}_{\mathsf{MPC}}$ are the simulated wire values of all the parties indexed by I. We denote $\mathcal{S}_{\mathsf{leak}}$ to consist of (w, v_w), for every wire $w \in \mathcal{W}_{lk}$ and v_w is the value assigned to w by $\mathsf{Sim}_{\mathsf{MPC}}$.

Finally, Sim outputs $\mathcal{S}_{\mathsf{leak}}$.

Now that we have described Sim, we prove that CC satisfies composable security property. That is, we prove:

- $\left\{\mathsf{RPDistr}^w_{\mathbf{p}}\left(\widehat{C}, \widehat{x}\right)\right\} \equiv \left\{\mathsf{Sim}(\widehat{C})\big|_{L \leftarrow \mathsf{Sim}(\widehat{C}) \wedge L \neq \perp}\right\}$
- $\mathsf{Sim}(\widehat{C})$ aborts with probability ε_0.

Consider the following hybrids.

Hyb_1: The output of this hybrid is $\left\{\mathsf{RPDistr}^w_{\mathbf{p}}\left(\widehat{C}, \widehat{x}\right)\right\}$.

Hyb_2: The output of this hybrid is $\left\{\mathsf{Hyb.Sim}\left(\widehat{C}\right)\right\}$.

We define the following hybrid partial simulator $\mathsf{Hyb.Sim} = (\mathsf{Hyb.Sim}_1, \mathsf{Hyb.Sim}_2)$.

Hybrid Simulator, $\mathsf{Hyb.Sim}(\widehat{C})$: It takes as input compiled circuit $\widehat{C}$ and does the following: Let $\mathcal{W}$ be the set of wires in $\widehat{C}$. Construct a set of leaked wires $\mathcal{W}_{lk}$ as follows: include every wire $w \in \mathcal{W}_{lk}$ with probability $\mathbf{p}$. It then executes $\mathsf{Hyb.Sim}_1(\widehat{C}, \mathcal{W}_{lk})$, which is defined below.

$\mathsf{Hyb.Sim}_1(\widehat{C}, \mathcal{W}_{lk})$: execute $\mathsf{Sim}_1(\widehat{C}, \mathcal{W}_{lk})$ to obtain $(\mathcal{W}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$.

Once Sim_1 is executed, construct a set S^{inp} as follows: for every wire $w \in \mathcal{W}^{inp}$, sample a uniformly random bit v_w and include $(w, v_w) \in S^{inp}$. Similarly, construct the set S^{out}.

$\mathsf{Hyb.Sim}_2(\widehat{C}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)$: It first checks if $|I| \geq t+1$ and if so, it aborts. Otherwise, execute $\widehat{C}(\widehat{x})$ honestly. Construct the set of leaked wire values $\mathcal{S}_{\mathsf{leak}}$ as follows. For every wire $w \in \mathcal{W}$, include $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$, where v_w is the value assigned to the wire w during the evaluation of $\widehat{C}(\widehat{x})$. Output $\mathcal{S}_{\mathsf{leak}}$.

Finally, $\mathsf{Hyb.Sim}$ outputs $\mathcal{S}_{\mathsf{leak}}$.

Claim 1. *The output distributions of hybrids* Hyb_1 *and* Hyb_2 *are* ε_0*-close.*

Proof. The output distributions of Hyb_1 and Hyb_2 differ only in the event when the number of leaked wires (which is nothing but $|I|$) is at least $t+1$. Therefore, it suffices to upper bound the probability of $|I| \geq t+1$.

We prove the following.

$$\mathsf{Pr}\left[|I| \geq t+1 \ : \ (\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I) \leftarrow \mathsf{Hyb.Sim}_1(\widehat{C}, \mathcal{W}_{lk})\right] \leq \varepsilon_0$$

Let $\mathbf{X}$ be the random variable that calculates the number of wires that leak. We have, $\mu = \mathbb{E}[\mathbf{X}] = \mathsf{N}_{\mathsf{g}}\mathbf{p}$. Let δ be such that $(1+\delta)\mu = t+1$. We use the following Chernoff bound.

Lemma 4 (Chernoff Bound [MU05])**.** *Let* $X = \sum_{i=1}^{n} X_i$ *be the sum of 0/1 independent random variables. Then for any* $\beta > 0$*,*

$$\mathsf{Pr}\left[X > (1+\beta)\mathbb{E}[X]\right] \leq \left(\frac{e^{\beta}}{(1+\beta)^{(1+\beta)}}\right)^{\mathbb{E}[X]}$$

Using the above Chernoff bound, we bound the error below.

$$\begin{aligned}
\mathsf{Pr}\left[|I| \geq t+1 \ : \ (\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I) \leftarrow \mathsf{Hyb.Sim}_1(\widehat{C}, \mathcal{W}_{lk})\right] &= \mathsf{Pr}[\mathbf{X} \geq t+1] \\
&= \mathsf{Pr}[\mathbf{X} \geq (1+\delta)\mu] \\
&\leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \\
&\leq \left(\frac{e^{\delta\mu}}{(1+\delta)^{(1+\delta)\mu}}\right) \cdot e^{\mu} \ (\because \mu > 0) \\
&= \left(\frac{e^{t+1}}{\left(\frac{t+1}{\mu}\right)^{t+1}}\right) \\
&= \left(\frac{e^{t+1}}{(t+1)^{t+1}}\right) \cdot \mu^{t+1} \\
&\leq \mu^{t+1} \ (\because \ t \geq 2) \\
&= (\mathsf{N}_{\mathsf{g}}\mathbf{p})^{t+1}
\end{aligned}$$

This completes the proof. □

Hyb_3: The output of this hybrid is the output of simulator Sim.

Claim 2. *The output distributions of* Hyb_2 *and* Hyb_3 *are identical.*

Proof. The difference between the output distributions of Hyb_2 and Hyb_3 is in the simulation of wire values of Ckt_i, for every $i \in I$. In particular, both Hyb_2 and Hyb_3 abort if $|I| > t$ and if $|I| \leq t$ then Hyb_2 assigns wire values by executing $\widehat{C}$ while Hyb_3 assigns wire values by executing $\mathsf{Sim}_{\mathsf{MPC}}$. In the corresponding MPC protocol Π, we view party P_i as being corrupted and there are less than t corruptions in Π. Thus, the claim that the output distributions of Hyb_2 and Hyb_3 are identical follows from the perfect security of Π. □

From the above claims, it follows that the output distributions of Hyb_1 and Hyb_3 are ε_0-close. Moreover, conditioned on Sim not aborting, we have that $\mathsf{Sim}(\widehat{C})$ perfectly simulates the leakage on $\widehat{C}(\widehat{x})$

□

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 into CC_{K+1} satisfying $(\mathbf{p}, \varepsilon_{K+1})$-composable security, 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[10] 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 Figure 1.

Let Π denote a n-party information theoretically secure computation protocol for F. Construct Ckt_Π as done in Section 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 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 $y = y_1 || \cdots || y_n$.

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

Properties of CC_{K+1}: We show that CC_{K+1} satisfies the properties of a composable circuit compiler.

Lemma 5 (Correctness)**.** *Let CC_K satisfy correctness of evaluation and correctness of encoding properties and let Π satisfy correctness property. Then, CC_{K+1} satisfies correctness of evaluation and correctness of encoding properties.*

Proof. Let $\widehat{C} \leftarrow \mathsf{CC}_{K+1}.\mathsf{Compile}(C)$. The proof of the lemma follows from the observations below.

- From the correctness of Π, it follows that Ckt_Π computes the same functionality as circuit C.
- The correctness of CC_K implies that the circuit Ckt_Π^* takes as input XOR secret sharing of input x, computes Ckt_Π (and hence, C) on x to obtain y and finally, computes the XOR secret sharing of y. Recall that $\widehat{C} = \mathsf{Ckt}_\Pi^*$.
- The input encoding $\mathsf{CC}_{K+1}.\mathsf{Encode}(\cdot)$ computes XOR secret sharing of the input. The output decoding $\mathsf{CC}_{K+1}.\mathsf{Encode}(\cdot)$ computes reconstruction of XOR secret sharing of the output.

Thus, $\mathsf{CC}_{K+1}.\mathsf{Decode}\,(\ \mathsf{CC}_{K+1}.\mathsf{Compile}\,(\mathsf{CC}_{K+1})(\mathsf{CC}_{K+1}.\mathsf{Encode}(\cdot)\ ))$ is functionally equivalent to C. □

Lemma 6 (Efficiency)**.** *Let L be the total computational complexity of Π for the functionality F. Suppose it holds that $|\mathsf{CC}_K.\mathsf{Compile}(G)| \leq L^K$ for some gate G then it holds that $|\mathsf{CC}_{K+1}.\mathsf{Compile}(G)| \leq L^{K+1}$.*

Proof. Recall that $\mathsf{CC}_{K+1}.\mathsf{Compile}(\cdot)$ was obtained by replacing every gate in Π with a gadget generated using $\mathsf{CC}_K.\mathsf{Compile}(\cdot)$. Thus, the size of $\mathsf{CC}_{K+1}.\mathsf{Compile}(\cdot)$ is nothing but the product of the total computational complexity of Π and the size of every gadget computed using $\mathsf{CC}_K.\mathsf{Compile}(\cdot)$. □

The following corollary is immediate from the above lemma.

Corollary 1. *Suppose $|\mathsf{CC}_{\mathsf{base}}.\mathsf{Compile}(G)|$ is a constant, for some gate G. We have $|\mathsf{CC}_K.\mathsf{Compile}(G)|$ to be a polynomial in N as long as $K \leq \log(N)$.*

Lemma 7. *CC_{K+1} satisfies XOR encoding property.*

Proof. This is immediate from the description of the compiler, CC_{K+1}. □

We now prove the security of CC_{K+1}. We show that CC_{K+1} is secure against random probing attacks if CC_K is secure against random probing attacks.

Proposition 2 (Security)**.** *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 $\varepsilon_{K+1} = (\mathsf{N_g}\varepsilon_K)^{t+1}$.*

Proof. We first construct a partial simulator Sim_{K+1} for the $(K+1)^{th}$ step. Let $\mathsf{Sim}_K = (\mathsf{Sim}_K^1, \mathsf{Sim}_K^2)$ be a partial simulator associated with CC_K such that CC_K satisfies $(\mathbf{p}, \varepsilon)$-composable security property with respect to Sim_K. We also employ the simulator of Π – to define this, first we need to define the real world adversary participating in Π. $\mathcal{A}_{\mathsf{MPC}}$ is a semi-honest adversary that corrupts a subset of the parties and outputs its entire view after the execution of the protocol. That is, it outputs the set $\{(w, v_w) : w \in Ckt_i \wedge i \in I\}$, where Ckt_i is the circuit implementation of party P_i and I consists of indices of all the parties that are corrupted by $\mathcal{A}$. Here, v_w denotes the value carried by the wire w in the execution of the protocol. We denote $\mathsf{Sim}_{\mathsf{MPC}}^{\Pi}$ to be the ideal world adversary corresponding to $\mathcal{A}$.

Denote the partial simulator to be $\mathsf{Sim}_{K+1} = (\mathsf{Sim}_{K+1}^1, \mathsf{Sim}_{K+1}^2)$. We describe Sim_{K+1} below.

Partial Simulator, $\mathsf{Sim}_{K+1}(\widehat{C})$. It takes as input compiled 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 $\mathbf{p}$. We next describe Sim^1_{K+1} and Sim_{K+2}; before that we establish some notation. Let Ckt_Π be the circuit obtained by applying Step I on the circuit C. Recall that Ckt_Π can be partitioned into sub-circuits $Ckt_1, \dots, Ckt_n$, where Ckt_i implements the i^{th} party in Π. Let Ckt^*_Π be the circuit obtained by applying Step II on Ckt_Π. Correspondingly, let $Ckt^*_1, \dots, Ckt^*_n$ be the partitions of Ckt^*_Π.

$\underline{\mathsf{Sim}^1_{K+1}(\widehat{C}, \mathcal{W}_{lk})}$: The goal is to determine the set of input and output wires of $\widehat{C}$ that will be necessary for the next stage. Looking ahead, values assigned to this set of wires will be necessary to simulate the internal wire values of $\widehat{C}$. As a first step, we calculate the set of sub-circuits of $\widehat{C}$ that cannot be simulated by the simulator of CC_K. Denote this set by I. Initialize $I = \emptyset$.

For every gate $G \in \mathsf{Ckt}_\Pi$, do the following: let $\widehat{G} \leftarrow \mathsf{CC}_{K+1}.\mathsf{Compile}(G)$ and let $\mathcal{W}_G \subseteq \mathcal{W}$ be the set of leaked wires in the gadget $\widehat{G}$. Execute $\mathsf{Sim}_K(\widehat{G}, \mathcal{W}_G)$ and if the execution fails, include i in the set I, where G belongs to the sub-circuit Ckt_i.

We now construct the set $\mathcal{W}^{inp}$ as follows:

- Consider the circuit Encode. Recall that Encode outputs a XOR secret sharing of the input. Every output wire of Encode corresponds to a secret share of a input bit. That is, there is mapping ψ that acts upon the output wire w and outputs 'j' if w corresponds to a secret share of the j^{th} input bit. Set $\mathcal{W}^{inp}$ to consists of all wires w such that: (i) there is $j \in [n]$ such that w is an input wire of Ckt^*_j and, (ii) $j \in I$.

Similarly construct the set $\mathcal{W}^{out}$. That is, $\mathcal{W}^{out}$ consists of all the output wires w that satisfy the following condition: $w \in Ckt^*_j$ for some $j \in [n]$ and $j \in I$. Output $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. This completes the description of Sim^1_{K+1}.

Let $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$ be the output of Sim^1_{K+1}. Construct the sets S^{inp} and S^{out} as follows. 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}.

$\underline{\mathsf{Sim}^2_{K+1}(\widehat{C}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)}$: The goal is to compute the simulated values $\mathcal{S}_{lk}$ for the leaked wires in the set $\mathcal{W}_{lk}$. If $|I| > t$ then abort. Otherwise, initialize $\mathcal{S}_{lk} = \emptyset$. Recall that $\widehat{C}$ can be partitioned into sub-circuits $\{Ckt^*_i\}_{i \in [n]}$. We consider two cases below.

Simulation of Wire Values in $\{Ckt^*_i\}_{i \in I}$: Execute the simulator of the MPC protocol $\mathsf{Sim}^\Pi_{\mathsf{MPC}}(I, \{S^{inp}_i\}_{i \in [\ell]}, \{S^{out}_i\}_{i \in [\ell']})$ to obtain the set S_{MPC}. The set S_{MPC} simulates the wire values in the sub-circuits $\{Ckt_i\}_{i \in I}$ (corresponding to the corrupted parties) of Ckt_Π. Using this, we construct the set S^*_{MPC}, which will consist of the simulated wire values in the sub-circuits $\{Ckt^*_i\}_{i \in I}$ of Ckt^*_Π.

Since the output distributions of $\mathcal{A}_{\mathsf{MPC}}$ and S_{MPC} are identically distributed, S_{MPC} can be expressed as $\cup_{i \in I} T_i$ and T_i consists of pairs of the form (w, v_w) for every wire $w \in Ckt_i$ and v_w is the value carried by w during the simulation. For every gate $G \in Ckt_i$, let w^{inp}_1, w^{inp}_2 be the input wires and w^{out}_1, w^{out}_2 be the output wires of G. Let $\{v^{inp}_j, v^{out}_j\}_{j \in \{1,2\}}$ be such that $(w^{inp}_j, v^{inp}_j) \in S_{\mathsf{MPC}}$ and let $(w^{out}_j, v^{out}_j) \in S_{\mathsf{MPC}}$ for $j \in \{1, 2\}$. Generate the simulated values corresponding to the gadget $\widehat{G}$, where $\widehat{G} \leftarrow \mathsf{Compile}(G)$, as follows:

- Compute $\widehat{v} \leftarrow \mathsf{Encode}(v^{inp}_1 || v^{inp}_2)$
- Compute the circuit $\widehat{G}$ on the input encoding $\widehat{v}$.
- Initialize the set, $S^G_{\mathsf{MPC}} = \emptyset$. For every wire $w \in \widehat{G}$, if v_w was the value carried by w in $\widehat{G}(\widehat{v})$ then include the pair (w, v_w) in S^G_{MPC}.

We have computed the simulated wire values for all the gadgets in the sub-circuits $\{Ckt^*_i\}_{i \in I}$. Now, compute the set S^*_{MPC} as: $S^*_{\mathsf{MPC}} = \cup_{G \in Ckt^*_i, i \in I} S^G_{\mathsf{MPC}}$. Assign $S_{lk} = S^*_{\mathsf{MPC}}$.

Simulation of Wire Values in $\{Ckt_i^*\}_{i \notin I}$: We now simulate the values for the leaked wires in the sub-circuits that are not indexed by the set I. For every gadget $\widehat{G} \in Ckt_i^*$ for $i \notin I$, do the following:

- Consider the set $\mathcal{W}_G^{lk} = \widehat{G} \cap \mathcal{W}_{lk}$. That is, $\mathcal{W}_G^{lk}$ is the set of wires in $\widehat{G}$ that are leaked.
- Execute $\mathsf{Sim}_K^1(\widehat{G}, \mathcal{W}_G^{lk})$ to obtain $(\mathcal{W}_G^{lk}, \mathcal{W}_G^{inp}, \mathcal{W}_G^{out}, I_G)$.

Construct S_G^{inp} and S_G^{out} for every $\widehat{G} \in Ckt_i^*$ recursively as follows. If G is an input gate, then include (w, v_w) in S_G^{inp} for every $w \in \mathcal{W}_G^{inp}$, where v_w is picked at random. Similarly construct S_G^{out} by including in S_G^{out}, pairs of the form (w, v_w) for every $w \in \mathcal{W}_G^{out}$ and where v_w is a bit picked uniformly at random. Suppose G is not an input gate, then let G' and G'' be gates such that they are connected to the input wires of G. By recursion, we have already constructed $S_{G'}^{inp}$ and $S_{G''}^{inp}$. Set $S_G^{inp} = S_{G'}^{inp} \cup S_{G''}^{inp}$. Construct S_G^{out} by including in S_G^{out}, pairs of the form (w, v_w) for every $w \in \mathcal{W}_G^{out}$ and where v_w is a bit picked uniformly at random.

For every $\widehat{G} \in Ckt_i^*$, execute $\mathsf{Sim}_K^2(\mathcal{W}_G^{lk}, \mathcal{W}_G^{inp}, \mathcal{W}_G^{out}, S_G^{inp}, S_G^{out})$ to obtain $\mathcal{S}_G^{lk}$. Include all the elements of $\mathcal{S}_G^{lk}$ in the set $\mathcal{S}_{lk}$.

Output the set of leaked values $\mathcal{S}_{lk}$. This completes the description of Sim_{K+1}.
We now argue that the simulated distribution of leaked wire values is statistically-close to the real distribution of leaked wire values. We employ the standard hybrid argument to argue this.

Consider a circuit $C \in \mathcal{C}$ and inputs $x \in \{0,1\}^\ell$, where ℓ is the input length of C. Let $\widehat{C} \leftarrow \mathsf{CC}_{K+1}.\mathsf{Compile}(C)$ and let $\widehat{x} \leftarrow \mathsf{CC}_{K+1}.\mathsf{Encode}(x)$ for $i \in [q]$. We prove:

- $\left\{\mathsf{RPDistr}_{\mathbf{p}}^w\left(\widehat{C}, \widehat{x}\right)\right\} \equiv \left\{\mathsf{Sim}_{K+1}(\widehat{C})\big|_{L \leftarrow \mathsf{Sim}_{K+1}(\widehat{C}) \wedge L \neq \perp}\right\}$,
- $\mathsf{Sim}_{K+1}(\widehat{C})$ aborts with probability ε

We state the hybrids below.

<u>**Hybrid** Hyb_1</u>: The output of this hybrid is:

$$\left\{\mathsf{RPDistr}_{\mathbf{p}}^w\left(\widehat{C}, \widehat{x}\right)\right\}$$

That is, the output of this hybrid is the distribution of leaked wire values in the evaluation of $\widehat{C}$ on $\widehat{x}$, for every $i \in [q]$.

<u>**Hybrid** Hyb_2</u>: We define a hybrid simulator denoted by $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1} = (\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}^1, \mathsf{Hyb}_2.\mathsf{Sim}_{K+1}^2)$ below. The output of this hybrid is,

$$\left\{\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}\left(\widehat{C}, \widehat{x}\right)\right\}$$

Description of $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$**.** It takes as input compiled circuit $\widehat{C}$ and input $\widehat{x}$. 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 $\mathbf{p}$. We next describe Sim_{K+1}^1 and Sim_{K+1}^2; before that we establish some notation. Let Ckt_Π be the circuit obtained by applying Step I on the circuit C. Recall that Ckt_Π can be partitioned into sub-circuits $Ckt_1, \ldots, Ckt_n$, where Ckt_i implements the i^{th} party in Π. Let Ckt_Π^* be the circuit obtained by applying Step II on Ckt_Π. Correspondingly, let $Ckt_1^*, \ldots, Ckt_n^*$ be the partitions of Ckt_Π^*.

$\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}^1(\widehat{C}, \mathcal{W}_{lk})$: It executes $\mathsf{Sim}_{K+1}^1(\widehat{C}, \mathcal{W}_{lk})$ to obtain $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. This completes the description of $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}^1$.

Let $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$ be the output of $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}^1$. Construct the sets S^{inp} and S^{out} as follows. 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}.

We describe $\mathsf{Hyb}_2.\mathsf{Sim}^2_{K+1}$ below. The two differences between Sim^1_{K+1} and $\mathsf{Hyb}_2.\mathsf{Sim}^1_{K+1}$ are (i) the simulator will not abort if $I \geq t$ and, (ii) instead of simulating the sub-circuits indexed by I using the simulator $\mathsf{Sim}_{\mathsf{MPC}}$ we instead use the values obtained in the real execution of the MPC protocol Π.

$\mathsf{Hyb}.\mathsf{Sim}^2_{K+1}(\widehat{C}, \widehat{x}, \mathcal{W}_{lk}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)$: The goal is to compute the simulated values $\mathcal{S}_{lk}$ for the leaked wires in the set $\mathcal{W}_{lk}$. Initialize $\mathcal{S}_{lk} = \emptyset$. Recall that $\widehat{C}$ can be partitioned into sub-circuits $\{Ckt^*_i\}_{i\in[n]}$. We consider two cases below.

Simulation of Wire Values in $\{Ckt^*_i\}_{i\in I}$: Evaluate the compiled circuit $\widehat{C}$ on $\widehat{x}$. For every wire $w \in Ckt^*_i$ such that $w \in \mathcal{W}_{lk}$, include (w, v_w) in $\mathcal{S}_{lk}$ if and only if v_w is the value carried by the wire w in the evaluation of $\widehat{C}(\widehat{x})$.
Simulation of Wire Values in $\{Ckt^*_i\}_{i\notin I}$: This is identical to the analogous step in the description of Sim_{K+1}.

Output the set of leaked values $\mathcal{S}_{lk}$.

Lemma 8. *Assuming ε_K-simulation with abort property of CC_K, the output distributions of hybrids Hyb_1 and Hyb_2 are identical.*

Proof. We argue that $\mathsf{RPDistr}^w_{\mathbf{p}}(\widehat{C}, \widehat{x})$ is identically distributed to $\mathsf{Hyb}.\mathsf{Sim}_{K+1}(\widehat{C}, \widehat{x})$. Once we show this, the proof of lemma follows from standard hybrid argument.

The distribution of leaked wires $\mathcal{W}_{lk}$ in $\mathsf{RPDistr}^w_{\mathbf{p}}$ is identical to that of $\mathsf{Hyb}_2.\mathsf{Sim}$. Let $\{Ckt^*_i\}_{i\in[n]}$ be the sub-circuits in $\widehat{C}$. The set of simulated wire values for the sub-circuits $\{Ckt^*_i\}_{i\in I}$, where I is as constructed in $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$, is the same for both $\mathsf{RPDistr}^w_{\mathbf{p}}$ and $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$.

We now focus on the leaked wire values in the sub-circuits $\{Ckt^*_i\}_{i\notin I}$. We use the security of CC_K to argue this. For every $i \notin I$, for every gadget $\widehat{G} \in Ckt^*_i$, let $\mathcal{D}^{lk}_G$ denote the distribution of leaked wire values in $\widehat{G}$ as generated in $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$. From the description of $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$, it follows that $\mathcal{D}^{lk}_G$ is identical to the output distribution of $\mathsf{Sim}_K(\widehat{G})$. Moreover, $\mathsf{Sim}_K(\widehat{G})$ does not abort. Otherwise, i would have been included in the set I. Thus, we can apply the security of CC_K to argue that $\mathcal{D}^{lk}_G$ is identically distributed with the leaked wire values of the gadget $\widehat{G}$ in the distribution $\mathsf{RPDistr}^w_{\mathbf{p}}(\widehat{C}, \widehat{x})$. Since the wire values are independently leaked, we can then use hybrid argument to argue that the distribution of the leaked wire values in $\{Ckt^*_i\}_{i\notin I}$ is identical in both $\mathsf{RPDistr}^w_{\mathbf{p}}$ and $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$. Thus, the proof of the lemma follows. □

Hybrid Hyb_3: As before, we define a hybrid simulator $\mathsf{Hyb}_3.\mathsf{Sim}_{K+1} = (\mathsf{Hyb}_3.\mathsf{Sim}^1_{K+1}, \mathsf{Hyb}_3.\mathsf{Sim}^2_{K+1})$. The output of this hybrid is,

$$\left\{\mathsf{Hyb}_3.\mathsf{Sim}_{K+1}\left(\widehat{C}, \widehat{x}\right)\right\}$$

Description of $\mathsf{Hyb}_3.\mathsf{Sim}_{K+1}$. This simulator is identical to the previous hybrid simulator $\mathsf{Hyb}_2.\mathsf{Sim}_{K+1}$, except that this simulator aborts if $|I| > t$ (specifically, $\mathsf{Hyb}_3.\mathsf{Sim}^2_{K+1}$ aborts).

Lemma 9. *The output distributions of hybrids Hyb_2 and Hyb_3 are ε_{K+1}-close.*

Proof. To prove this lemma, it suffices to consider the indistinguishability of hybrids Hyb_2 and Hyb_3 when there is only one input (instead of q inputs). In this case, let I be as computed in $\mathsf{Hyb}_3.\mathsf{Sim}_{K+1}$. Observe that the probability that $|I| > t$ is the same as the distinguishing advantage between hybrids Hyb_2 and Hyb_3. We calculate the probability that $|I| > t$ below. For the general case, when there are q inputs, we apply the hybrid argument and incur a security loss of q.

Claim 3. *Let $\mathcal{W}$ be the set of wires in $\widehat{C}$. For every wire $w \in \mathcal{W}$, include it in $\mathcal{W}_{lk}$ with probability $\mathbf{p}$. We have,*

$$\mathsf{Pr}\left[|I| > t \ : \ (\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I) \leftarrow \mathsf{Hyb}_2.\mathsf{Sim}^1_{K+1}(\widehat{C}, \mathcal{W})\right] \leq \varepsilon_{K+1},$$

where ε_{K+1} is as defined in the statement of the lemma.

Proof. Let $\mathbf{X}$ be the random variable that calculates the number of instantiations of Sim_K that fail. We have, $\mu = \mathbb{E}[\mathbf{X}] = \mathsf{N_g}\varepsilon_K$. We use Chernoff bound (Lemma 4) to calculate ε_{K+1}. Let $(\delta+1)\mu = t+1$.

$$\begin{aligned}
\mathsf{Pr}[\text{At least } (t+1) \text{ instantiations of } \mathsf{Sim}_{K+1} \text{ fail}] &= \mathsf{Pr}[\mathbf{X} \geq t+1] \\
&= \mathsf{Pr}[\mathbf{X} \geq (1+\delta)\mu] \\
&\leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \\
&\leq \left(\frac{e^{\delta\mu}}{(1+\delta)^{(1+\delta)\mu}}\right) \cdot e^{\mu} \ (\because \mu > 0) \\
&= \left(\frac{e^{t+1}}{\left(\frac{t+1}{\mu}\right)^{t+1}}\right) \\
&= \left(\frac{e^{t+1}}{(t+1)^{t+1}}\right) \cdot \mu^{t+1} \\
&\leq \mu^{t+1} \ (\because \ t \geq 2) \\
&= (\mathsf{N_g}\varepsilon_K)^{t+1}
\end{aligned}$$

This completes the proof. □

□

Hyb_4: The output of this hybrid is,

$$\left\{\mathsf{Sim}_{K+1}\left(\widehat{C}\right)\right\}$$

Lemma 10. *Assuming the perfect security of Π, hybrids Hyb_3 and Hyb_4 are identically distributed.*

Proof. The only difference between Hyb_3 and Hyb_4 is in the simulation of the wires in the sub-circuits indexed by I. For simplicity, we consider the case when there is only one input x^1 (i.e, $q = 1$). The general case, when q is arbitrary, follows from standard hybrid argument.

- We perform the following operations in Hyb_3:
  - Apply Step I to circuit C to obtain the circuit Ckt_Π. Recall that Ckt_Π is a circuit representation of the protocol Π. It is divided into sub-circuits $Ckt_1, \ldots, Ckt_n$, with Ckt_i representing party P_i. Then, apply Step II on Ckt_Π to obtain Ckt^*_Π. The corresponding partitions are denoted by $Ckt^*_1, \ldots, Ckt^*_n$.
  - Let $\mathcal{W}$ be the total set of wires in $\widehat{C}$. Denote by $\mathcal{W}_{lk}$, the set of leaked wires computed by including every wire $w \in \mathcal{W}$ in $\mathcal{W}_{lk}$ with probability $\mathbf{p}$.
  - Compute $\mathsf{Hyb}_3.\mathsf{Sim}_{K+1}(\widehat{C}, \mathcal{W}_{lk})$ (note that both $\mathsf{Hyb}_3.\mathsf{Sim}_{K+1}$ and $\mathsf{Hyb}_4.\mathsf{Sim}_{K+1}$ are identical). Let the output of this step be $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. The simulator aborts if $|I| > t$.
  - The values for the leaked wires in the sub-circuits not indexed by I are simulated using Sim_K.
  - The values for the leaked wires in the sub-circuits indexed by I, $\{Ckt^*_i\}_{i \in I}$, are simulated as follows: first compute Ckt_i on input x^1, for $i \in I$, and then using the wire values generated during this computation to generate values corresponding to leaked wires of $\{Ckt^*_i\}$.

- In Hyb_4, except the last bullet above, all the other bullets are the same. In this case, generate values for the leaked wires in the sub-circuits indexed by I, $\{Ckt_i^*\}_{i \in I}$, by first executing $\mathsf{Sim}_{\mathsf{MPC}}$ to generate wire values for $\{Ckt_i\}_{i \in I}$ and using this, generate wire values for $\{Ckt_i^*\}_{i \in I}$.

Hyb_3 and Hyb_4 abort, i.e., when $|I| > t$, with the same probability. When $|I| \leq t$, we invoke the perfect security of Π to argue that Hyb_3 and Hyb_4 are identically distributed. □

□

From the above theorems, we have the following theorem.

Theorem 6. *Suppose CC_K is a composable circuit compiler satisfying L^K-efficiency and $(\mathbf{p}, \varepsilon_K)$-composable security. Then, CC_{K+1} satisfies L^{K+1}-efficiency and $(\mathbf{p}, \varepsilon_{K+1})$-composable security, where $\varepsilon_{K+1} = (\mathsf{N_g}\varepsilon_K)^{t+1}$.*

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 $\mathsf{CC}_{\mathrm{poly}}$ 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, $\mathsf{CC}_{\mathrm{poly}}.\mathsf{Compile}(C)$: It takes as input circuit $C \in \mathcal{C}$. For every gate G in C, it computes $\widehat{G} \leftarrow \mathsf{CC}_{\mathrm{exp}}.\mathsf{Compile}(G)$ 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 that the above scheme satisfies the properties of a composable circuit compiler.

Lemma 11. *$\mathsf{CC}_{\mathrm{poly}}$ satisfies the following: (i) correctness of evaluation property, (ii) correctness of encoding property and, (iii) correctness of n-XOR encoding property.*

Proof. We argue correctness of evaluation property inductively. Consider a circuit $C \in \mathcal{C}$ and an input x. Let $\widehat{C} \leftarrow \mathsf{CC}_{\mathrm{poly}}.\mathsf{Compile}(C)$ and $\widehat{x} \leftarrow \mathsf{CC}_{\mathrm{poly}}.\mathsf{Encode}(x)$. Consider the evaluation of $\widehat{C}$ on $\widehat{x}$. We make the following observation: for any gate G in the circuit C, if the input encoding of $\widehat{G}$ encodes the value v then the evaluation of $\widehat{G}$ on the encoding of v yields an output encoding that encodes the value w, where $w = G(v)$. This observation follows from the correctness of $\mathsf{CC}_{\mathsf{exp}}$. By applying this observation inductively, the correctness of evaluation property of $\mathsf{CC}_{\mathrm{poly}}$ follows.

Observe that (iii) follows by construction and moreover, (iii) implies (ii). □

Lemma 12. *$\mathsf{CC}_{\mathrm{poly}}$ is L_{poly}-efficient, where L_{poly} is a polynomial.*

Proof. Let $\widehat{C} \leftarrow \mathsf{CC}_{\rm poly}.\mathsf{Compile}(C)$, for $C \in \mathcal{C}$. We have $\widehat{C} = |C| \cdot \max_{\forall G \in C}(|\widehat{G}|)$, where $\max_{\forall G \in C}(|\widehat{G}|)$ denotes the maximum size of a gadget associated to any gate in $\widehat{C}$.

From L_{exp}-efficiency of $\mathsf{CC}_{\mathsf{exp}}$ and since the size of any gate is a constant, we have $\max_{\forall G \in C}(|\widehat{G}|)$ is a constant. Thus, we have $|\widehat{C}| = \mathbf{c} \cdot |C|$, for some constant $\mathbf{c}$. □

Lemma 13. *Let $\mathsf{CC}_{\mathsf{exp}}$ satisfies $(\mathbf{p}, \varepsilon_{\rm exp})$-composable security. $\mathsf{CC}_{\rm poly}$, associated with circuits of size s, satisfies $(\mathbf{p}, s \cdot \varepsilon_{\rm exp})$-composable security.*

Proof. Let $\mathsf{Sim}_{\mathsf{exp}}$ be a partial simulator such that $\mathsf{CC}_{\mathsf{exp}}$ satisfies composable security with respect to $\mathsf{Sim}_{\mathsf{exp}} = (\mathsf{Sim}^1_{\mathsf{exp}}, \mathsf{Sim}^2_{\mathsf{exp}})$. We use this to construct a partial simulator $\mathsf{Sim}_{\rm poly} = (\mathsf{Sim}^1_{\rm poly}, \mathsf{Sim}^2_{\rm poly})$.

Partial Simulator, $\mathsf{Sim}_{\rm poly}(\widehat{C})$: 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 $\mathcal{W}_{lk}$ with probability $\mathbf{p}$. Next compute $\mathsf{Sim}^1_{\rm poly}(\widehat{C}, \mathcal{W}_{lk})$.

$\mathsf{Sim}^1_{\rm poly}(\widehat{C}, \mathcal{W}_{lk})$: Let $\mathcal{W}_{lk} = \cup_{G \in C} \mathcal{W}^G_{lk}$, where $\mathcal{W}^G_{lk}$ is a subset of the wires in the gadget $\widehat{G} \leftarrow \mathsf{CC}_{\mathsf{exp}}.\mathsf{Compile}(G)$. Observe that the sets $\mathcal{W}^{G_1}_{lk}$ and $\mathcal{W}^{G_2}_{lk}$ for two different gates G_1 and G_2 need not be distinct. For every gate $G \in C$, compute $\mathsf{Sim}^1_{\mathsf{exp}}(\widehat{G}, \mathcal{W}^G_{lk})$ to obtain $(\mathcal{W}^G_{lk}, \mathcal{W}^{inp,G}, \mathcal{W}^{out,G}, I^G)$. Let $\mathcal{W}^{inp} = \cup_{G \in C} \mathcal{W}^{inp,G}$. Similarly, let $\mathcal{W}^{out} = \cup_{G \in C} \mathcal{W}^{out,G}$. Finally, set $I = \cup_{G \in C} I^G$.

Output $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$.

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}. Observe that S^{inp} can be decomposed as $S^{inp} = \cup_{G \in C} S^{inp,G}$, where the marginal distribution of $S^{inp,G}$ is $\mathcal{W}^G_{lk}$. Similarly, S^{out} can be decomposed as $S^{out} = \cup_{G \in C} S^{out,G}$.

Next, compute $\mathsf{Sim}^2_{\rm poly}$ as follows.

$\mathsf{Sim}^2_{\rm poly}\left(\widehat{C}, \mathcal{W}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I\right)$: for every gate G in C, compute $\mathsf{Sim}^2_{\mathsf{exp}}(\widehat{G}, \mathcal{W}_G, \mathcal{W}^{inp,G}, S^{inp,G}, \mathcal{W}^{out,G}, S^{out,G}, I^G)$, where $\mathcal{W}_G$ is the set of wires in the gadget $\widehat{G}$. If for any gate G, $\mathsf{Sim}^2_{\mathsf{exp}}(\cdot)$ fails, abort. Else, denote the output of $\mathsf{Sim}^2_{\mathsf{exp}}(\widehat{G}, \mathcal{W}_G, \mathcal{W}^{inp,G}, S^{inp,G}, \mathcal{W}^{out,G}, S^{out,G}, I^G)$ to be $\mathcal{S}^G_{\mathsf{leak}}$. Output the set $\mathcal{S}_{\mathsf{leak}} = \cup_{G \in C} \mathcal{S}^G_{\mathsf{leak}}$.

This completes the description of $\mathsf{Sim}^2_{\rm poly}$. We prove the following claim.

Claim 4. *The following two properties are satisfied:*

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

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

  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 $\mathsf{RPDistr}^w_{\mathbf{p}}$.

- **ε-Simulation with Abort**: *For every $C \in \mathcal{C}$, $x \in \{0,1\}^\ell$, $\mathsf{Sim}_{\rm poly}(\widehat{C})$ aborts with probability $s \cdot \varepsilon$.*

Proof. First, we argue that the probability that Sim_{poly} aborts is $s \cdot \varepsilon$. To see this, note that the probability that $\mathsf{Sim}_{\rm exp}$ fails for every gate in the circuit is ε. Moreover, $\mathsf{Sim}_{\rm poly}$ fails only if $\mathsf{Sim}_{\rm exp}$ fails for any gate. By union bound, we have $\mathsf{Sim}_{\rm exp}$ fails is at most $s \cdot \varepsilon$.

We now argue $\mathbf{p}$-partial simulation property. Let us condition on the event that none of $\mathsf{Sim}_{\rm exp}$ aborts. First, note that $\mathsf{Sim}_{\rm exp}$, for every gate, is executed independently. Moreover, conditioned on the event that $\mathsf{Sim}_{\rm exp}(\widehat{G})$ does not abort for a gate G, its output is identically distributed to leakage on the computation of $\widehat{G}$. Thus, the joint output distribution of $\mathsf{Sim}_{\rm exp}$ on all the compiled gates in the circuits is identical to the leakage on the computation of $\widehat{C}$. This proves the claim. □

□

From the above lemmas, we have the following theorem.

Theorem 7. *Suppose* CC_{exp} *is a composable circuit compiler satisfying* L_{exp}*-efficiency and* $(\mathbf{p}, \varepsilon_{\text{exp}})$*-composable security. Then,* $\mathsf{CC}_{\text{poly}}$ *is a composable circuit compiler for* $\mathcal{C}$ *satisfying* $L_{\text{exp}}(k) \cdot f$*-efficiency* $(\mathbf{p}, s \cdot \varepsilon_{\text{exp}})$*, where* s *is the size of the circuit in* $\mathcal{C}$ *being compiled,* k *is a constant and* f *is a linear function.*

That is, every circuit C compiled using $\mathsf{CC}_{\text{poly}}$ has efficiency at most $L_{\text{exp}}(k) \cdot f(|C|)$.

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.

We now present a construction (Figure 2) 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 3. *Let* $K \in \mathbb{N}$*. Consider a MPC protocol* Π *for a* n*-party functionality* F *and tolerating at most* t *corruptions, with* $t \geq 2$*.*

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:*

- $\mathbf{p} = \frac{1}{\mathsf{N}_{\mathsf{g}}^2}$ *,*
- $L_1(k)$ *is a constant and* f *is a linear function,*
- c *is a constant,*
- N_{g} *is the number of gates in the circuit* Ckt_Π

Proof. We prove that CC_{main} satisfies all the properties of a composable circuit compiler.

Lemma 14. *The correctness of* Π *implies the correctness of* CC_{main}*.*

Proof. It suffices to show that CC^* satisfies the correctness property of a composable circuit compiler. From Lemma 1, the correctness of Π implies the correctness of $\mathsf{CC}_{\mathsf{base}}$. From Lemma 5, the correctness of $\mathsf{CC}_{\mathsf{base}}$ implies the correctness of CC_K. From Lemma 11, the correctness of CC_K implies the correctness of CC^*. □

Lemma 15. *Let the total computational complexity of* Π *be* L_1*.* CC_{main} *satisfies* $(L_1(k))^K \cdot f$*-efficiency, where* k *is a constant and* f *is a linear function.*

Proof. From Lemma 2, $\mathsf{CC}_{\mathsf{base}}$ satisfies L_1-efficiency. From Lemma 6, CC_K satisfies L_1^K-efficiency. From Lemma 12, CC^* satisfies $f \cdot L_1^K$-efficiency, where f is a linear function. □

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 $\varepsilon_1 = (\mathsf{N}_{\mathsf{g}}\mathbf{p})^{t+1}$ and L_1-efficiency.
  - Set $\mathsf{CC}_1 = \mathsf{CC}_{\mathsf{base}}$. Repeat the following process for $i = 1, \ldots, K-1$: 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.

Figure 2: Construction of CC_{main}

Lemma 16. *Let Π be perfectly secure. Then, CC_{main} satisfies $(\mathbf{p}, c^{c^K})$-composable security, for some constant c.*

Proof. Note that $\mathsf{CC}_{\mathsf{base}}$ is $(\mathbf{p}, \varepsilon_1)$-composable secure, where $\varepsilon_1 =$. From Proposition 2, CC_K satisfies $(\mathbf{p}, \varepsilon_K)$-composable security, where $\varepsilon_K = (\mathsf{N}_{\mathsf{g}}\varepsilon_{K-1})^{t+1}$. From Theorem 13, CC^* satisfies $(\mathbf{p}, s \cdot \varepsilon_K)$-composable security.

Consider the following claim.

Claim 5. $\varepsilon_K \leq \frac{1}{\mathsf{N}_{\mathsf{g}}^{t^K+1}}$

Proof. We prove the following subclaim.

SubClaim 1. $\varepsilon_1 \leq \frac{1}{\mathsf{N}_{\mathsf{g}}^{t+1}}$

Proof. Recall that $\varepsilon_1 \leq (\mathsf{N}_{\mathsf{g}}\mathbf{p})^{t+1}$. Subtituting $\mathbf{p} = \frac{1}{\mathsf{N}_{\mathsf{g}}^2}$, we obtain the proof of the subclaim. □

We prove the claim by induction. This is true for the base case from Subclaim 1. Assume that the statement of the claim is true for κ iterations. That is, $\varepsilon_\kappa \leq \frac{1}{\mathsf{N_g}^{t^\kappa+1}}$. We prove the statement for $(\kappa+1)^{th}$ iteration.

$$\begin{aligned}
\varepsilon_{\kappa+1} &\leq (\mathsf{N_g}\varepsilon_\kappa)^{t+1} \\
&\leq \left(\mathsf{N_g}\frac{1}{\mathsf{N_g}^{t^\kappa+1}}\right)^{t+1} \\
&\leq \frac{1}{\mathsf{N_g}^{t^\kappa\cdot(t+1)}} \\
&\leq \frac{1}{\mathsf{N_g}^{t^{(\kappa+1)}+1}}
\end{aligned}$$

This proves the claim.

□

□

□

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

Proposition 4. *There is a construction of a composable circuit compiler for $\mathcal{C}$ satisfying $(\mathbf{p}, \mathsf{negl})$-composable security, where $\mathbf{p} = 3 \times 10^{-8}$.*

Proof. We prove this by instantiating Proposition 3 with a specific semi-honest secure multiparty computation protocol for n-party functionality F (Figure 1) tolerating at most t corruptions. In particular, we instantiate this with the construction of [Mau02]. We recall the construction for completeness.

The protocol of [Mau02] proceeds as follows: suppose C is the circuit being securely computed. Let the input of i^{th} party be x_i and let ℓ_x be the maximum size of the inputs of all the parties. Every party receives an output bit at the end of the protocol.

- **Secret Sharing Step**: First, share x_i additively into $s_1, \ldots, s_k$ shares, where $k = \binom{n}{t}$. Denote $\{S_1, \ldots, S_k\}$ to be all possible sets of size t. Party j receives a share s_i if and only if $j \notin S_i$. Note that every party has $\ell_x\binom{n-1}{t}$ number of shares. Thus, to share a bit, we need k randomness gates and one addition gate. The complexity of sharing is $k+1$.
- **Addition**: Every party locally adds all his shares. The total complexity of this step is $n\binom{n-1}{t}$.
- **Multiplication**:
  - Let $\{s_i\}$ and $\{t_j\}$ be the set of shares. Consider the set $S = \{(i,j)\}$. Partition S into sets $U_1, \ldots, U_n$ such that $(i,j) \in U_m$ if $m \in \overline{T_i} \cap \overline{T_j}$. Party m computes $r_m = \sum_{(i,j)\in U_m} s_i t_j$.
  - Share r_m among all the players.

  The total computational complexity of this step is at most $\binom{n-1}{t}^2 + 2n\binom{n}{t}$.
- **Output Recovery**: At the end of the protocol, every party broadcasts its shares to all other parties. Every party adds all the shares it receives. The complexity of this step is $\binom{n}{t}$.

Thus, the total computational complexity of this protocol is $|C| \cdot (\binom{n-1}{t}^2 + 2n\binom{n}{t})$.

We now determine the complexity of the circuit representing the functionality F (Figure 1). We first represent $F = F[G]$ by the following circuit:

- It takes as input n shares of two bits and then reconstructs it to obtain bits a and b. This reconstruction can be performed by a circuit of size $2(n-1)$.

- It then computes a gate G (with fan-in and fan-out being 2) on a and b to obtain the output c. The complexity of this step is 1.
- Finally, it computes n additive shares of c twice. The complexity of this step is $2(n-1)$.

Thus, the complexity of F is $4n-3$. Thus, we get the computational complexity of Π for F to be $(4n-3)\cdot(\binom{n-1}{t}^2 + 2n\binom{n}{t})$.

Substituting the parameters $n=5$, $t=2$ (recall that t has to be at least 2), we get the total number of gates to be Π is 5712. Thus, substituting Π and $K = \log(\mathrm{poly}(\log(s)))$ in Proposition 3, we obtain a $(\mathbf{p}, \mathsf{negl}(s))$-secure composable circuit compiler for all circuits satisfying poly-efficiency (in particular, after compiling a circuit of size s, we get a circuit of size $s \cdot \mathrm{poly}(\log(s))$[11]), where $\mathbf{p} = \frac{1}{5712^2} = 3 \times 10^{-8}$. $\square$

Non-Boolean Basis. We present a construction of circuit compiler when the compiled circuit is over a non-boolean basis. As a consequence, we can prove the security of our construction under better leakage rate than the previous construction over boolean basis. For simplicity of analysis, we consider basis consisting of randomized functions. With a modification of the current analysis, the functions can be derandomized.

Proposition 5. *Let $\delta > 0$. Suppose there is a construction of composable circuit compiler $\mathsf{CC}_{\mathsf{Bool}}$ over $\mathbb{B}$ for $\mathcal{C}$ over $\mathbb{B}$ satisfying $(\mathbf{p}, \varepsilon)$-composable security. Then there is a construction of a composable circuit compiler $\mathsf{CC}_{\mathsf{NB}}$ over $\mathbb{B}'$ for $\mathcal{C}$ over $\mathbb{B}$ satisfying $(\mathbf{p}_{\mathsf{NB}}, \varepsilon)$-composable security, where (i) $\mathbb{B}'$ consists of all randomized functions mapping 2ℓ inputs to 2ℓ outputs and, (ii) $\mathbf{p}_{\mathsf{NB}} = \mathbf{p}^{1/\ell}$.*

Proof. We first present the construction of $\mathsf{CC}_{\mathsf{NB}}$.

$\mathsf{CC}_{\mathsf{NB}}.\mathsf{Compile}(C)$: On input circuit C, first compute $\widehat{C}_{\mathsf{Bool}} \leftarrow \mathsf{CC}_{\mathsf{Bool}}.\mathsf{Compile}(C)$. Construct a circuit $\widehat{C}_{\mathsf{Bool}}$ as follows: consider a gate G in $\widehat{C}$ with input wires w_1^{inp}, w_2^{inp} and output wires w_1^{out}, w_2^{out}. Replace every gate G in $\widehat{C}_{\mathsf{Bool}}$ with a function $f_G : \{0,1\}^{2\ell} \to \{0,1\}^{2\ell}$ defined as follows:

- f_G takes as input ℓ additive shares of values v_1 (carried by w_1) and v_2 (carried by w_2),
- reconstructs the values v_1, v_2,
- computes $G(v_1, v_2)$ and,
- computes two sets of ℓ additive shares of $G(v_1, v_2)$ (using fresh randomness) corresponding to the two output wires of G.

In particular, every wire w in $\widehat{C}_{\mathsf{Bool}}$ will be split into corresponding ℓ wires in $\widehat{C}_{\mathsf{NB}}$. We denote a function ϕ that maps w into a set of ℓ wires in $\widehat{C}_{\mathsf{NB}}$. If v_w is the value carried by w during the computation of $\widehat{C}$ then correspondingly the ℓ wires in $\widehat{C}_{\mathsf{NB}}$ will carry the additive shares of v_w. Note that the output of computation of $\widehat{C}_{\mathsf{NB}}$ is a secret sharing of the output of $\widehat{C}_{\mathsf{Bool}}$.

Output $\widehat{C}_{\mathsf{NB}}$.

$\mathsf{CC}_{\mathsf{NB}}.\mathsf{Decode}(\widehat{y})$: On input encoding $\widehat{y}$, first reconstruct the additive shares to obtain the output encoding of $\widehat{C}_{\mathsf{Bool}}$. By the XOR-encoding property, the output encoding of $\widehat{C}_{\mathsf{Bool}}$ is itself an additive sharing of y. Reconstruct y from the encoding. Output y.

The correctness and efficiency properties of $\mathsf{CC}_{\mathsf{NB}}$ follows from the correctness and efficiency properties of $\mathsf{CC}_{\mathsf{Bool}}$.

Lemma 17. *$(\mathbf{p}, \varepsilon)$-composable security of $\mathsf{CC}_{\mathsf{Bool}}$ implies the $(\mathbf{p}_{\mathsf{NB}}, \varepsilon)$-composable security of $\mathsf{CC}_{\mathsf{NB}}$.*

[11]Note that encoding of an input of length ℓ has size $\ell \cdot \mathrm{poly}(\log(s))$.

Proof. Let $\mathsf{Sim}_{\mathsf{Bool}} = (\mathsf{Sim}^1_{\mathsf{Bool}}, \mathsf{Sim}^2_{\mathsf{Bool}})$ be the partial simulator such that $\mathsf{CC}_{\mathsf{Bool}}$ satisfies $(\mathbf{p}, \varepsilon)$-composable security with respect to $\mathsf{Sim}_{\mathsf{Bool}}$. We construct a simulator $\mathsf{Sim}_{\mathsf{NB}} = (\mathsf{Sim}^1_{\mathsf{NB}}, \mathsf{Sim}^2_{\mathsf{NB}})$.

$\mathsf{Sim}_{\mathsf{NB}}(\widehat{C}_{\mathsf{NB}})$: On input circuit $\widehat{C}_{\mathsf{NB}}$, let $\mathcal{W}_{\mathsf{NB}}$ be the set of wires in $\widehat{C}_{\mathsf{NB}}$. Construct $\mathcal{W}^{\mathsf{NB}}_{lk}$ by including every wire $w \in \mathcal{W}_{\mathsf{NB}}$ with probability $\mathbf{p}$. Then compute the following.

$\mathsf{Sim}^1_{\mathsf{NB}}(\mathcal{W}^{\mathsf{NB}}_{lk})$: Construct a set $\mathcal{W}^{\mathsf{Bool}}_{lk}$. For every wire w in $\widehat{C}$, check if all the wires in $\phi(w)$ is included in $\mathcal{W}^{\mathsf{NB}}_{lk}$. If so, include $w \in \mathcal{W}^{\mathsf{Bool}}_{lk}$. Compute $\mathsf{Sim}^2_{\mathsf{Bool}}(\mathcal{W}^{\mathsf{Bool}}_{lk})$ to obtain $(\mathcal{W}^{\mathsf{Bool}}_{lk}, \mathcal{W}^{\mathsf{Bool}}_{inp}, \mathcal{W}^{\mathsf{Bool}}_{\mathsf{out}}, I)$. Compute $\mathcal{W}^{\mathsf{NB}}_{inp}$ and $\mathcal{W}^{\mathsf{NB}}_{out}$ as follows: for every wire $w \in \mathcal{W}^{\mathsf{Bool}}_{inp}$, include all the wires in $\phi(w)$ in $\mathcal{W}^{\mathsf{Bool}}_{inp}$. Similarly, for every wire $w \in \mathcal{W}^{\mathsf{Bool}}_{out}$, include all the wires in $\phi(w)$ in $\mathcal{W}^{\mathsf{Bool}}_{out}$.

Output $(\mathcal{W}^{\mathsf{NB}}_{lk}, \mathcal{W}^{\mathsf{NB}}_{inp}, \mathcal{W}^{\mathsf{NB}}_{out}, I)$.

Construct sets $\mathcal{S}^{\mathsf{NB}}_{inp}$ and $\mathcal{S}^{\mathsf{NB}}_{out}$. For every wire $w \in \mathcal{W}^{\mathsf{NB}}_{inp}$, include $(w, v_w) \in \mathcal{S}^{\mathsf{NB}}_{inp}$ for a bit v_w picked uniformly at random. For every wire $w \in \mathcal{W}^{\mathsf{NB}}_{out}$, include $(w, v_w) \in \mathcal{S}^{\mathsf{NB}}_{out}$ for a bit v_w picked uniformly at random.

$\mathsf{Sim}^2_{\mathsf{NB}}(\mathcal{W}^{\mathsf{NB}}_{lk}, \mathcal{W}^{\mathsf{NB}}_{inp}, \mathcal{S}^{\mathsf{NB}}_{inp}, \mathcal{W}^{\mathsf{NB}}_{out}, \mathcal{S}^{\mathsf{NB}}_{out}, I)$: Construct the sets $\mathcal{S}^{\mathsf{Bool}}_{inp}$ and $\mathcal{S}^{\mathsf{Bool}}_{out}$ as follows. First re-compute $\mathcal{W}^{\mathsf{Bool}}_{inp}$ and $\mathcal{W}^{\mathsf{Bool}}_{out}$ from $\mathcal{W}^{\mathsf{NB}}_{inp}$ and $\mathcal{W}^{\mathsf{NB}}_{out}$, respectively. For every wire $w \in \mathcal{W}^{\mathsf{Bool}}_{inp}$, perform the following: let $(v^1_w, \ldots, v^\ell_w)$ be the values assigned to the set $\phi(w)$ in $\mathcal{S}^{\mathsf{NB}}_{inp}$ and let $v_w = \oplus_{i=1}^{\ell} v^i_w$. Include $(w, v_w) \in \mathcal{S}^{\mathsf{Bool}}_{inp}$. Similarly, construct $\mathcal{S}^{\mathsf{Bool}}_{out}$. Compute $\mathsf{Sim}^2_{\mathsf{Bool}}(\mathcal{W}^{\mathsf{NB}}_{lk}, \mathcal{W}^{\mathsf{NB}}_{inp}, \mathcal{S}^{\mathsf{NB}}_{inp}, \mathcal{W}^{\mathsf{NB}}_{out}, \mathcal{S}^{\mathsf{NB}}_{out}, I)$ to obtain the set $\mathcal{S}^{\mathsf{Bool}}_{\mathsf{leak}}$. If $\mathsf{Sim}^2_{\mathsf{Bool}}$ then $\mathsf{Sim}^2_{\mathsf{NB}}$ also aborts.

Construct the set $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$ as follows. For every wire $w \in \mathcal{W}^{\mathsf{Bool}}_{lk}$,

- if all the wires in $\phi(w)$ are in $\mathcal{W}^{\mathsf{NB}}_{lk}$ then include all the pairs $(w_1, v^1_w), \ldots, (w_\ell, v^\ell_w)$ in $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$, where $\phi(w) = \{w_1, \ldots, w_\ell\}$ and $v^1_w, \ldots, v^\ell_w$ are sampled uniformly at random subject to the constraint that $v_w = \oplus_{i=1}^{\ell} v^i_w$ and $(w, v_w) \in \mathcal{S}^{\mathsf{Bool}}_{\mathsf{leak}}$.
- if all the wires in $\phi(w)$ are not in $\mathcal{W}^{\mathsf{NB}}_{lk}$ then let S be a proper subset of $\phi(w)$. For every $w_i \in S$, include $(w_i, v^i_w) \in \mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$, where v^i_w is sampled uniformly at random.

Output $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$.

Claim 6. *ε-simulation with abort property of $\mathsf{CC}_{\mathsf{Bool}}$ implies the ε-simulation with abort property of $\mathsf{CC}_{\mathsf{NB}}$.*

Proof. The probability that $\mathsf{Sim}_{\mathsf{NB}}$ aborts is the same as the probability that $\mathsf{Sim}_{\mathsf{Bool}}$ aborts. ☐

Claim 7. *The $\mathbf{p}$-partial simulation property of $\mathsf{CC}_{\mathsf{Bool}}$ implies the $\mathbf{p}_{\mathsf{NB}}$-partial simulation property of $\mathsf{CC}_{\mathsf{NB}}$.*

Proof. Consider a circuit C and input x. We argue that the leakage on the computation of $\widehat{C}_{\mathsf{NB}}$ on $\widehat{x}$ can be simulated by $\mathsf{Sim}_{\mathsf{NB}}$. Denote the output of $\mathsf{Sim}_{\mathsf{NB}}(\widehat{C})$ to be $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$. We consider the set $\mathsf{Marg}(\mathcal{S}_{\mathsf{leak}}) = \{w : \exists\, v_w \in \{0,1\}, (w, v_w \in \mathcal{S}_{\mathsf{leak}})\}$

To show this, we consider the following subset of wires $\mathsf{NotAllLk}$ in the circuit $\widehat{C}$. For every w in $\widehat{C}$, if $\phi(w) \not\subset \mathsf{Marg}(\mathcal{S}_{\mathsf{leak}})$ then include w in $\mathsf{NotAllLk}$.

for every wire $w \in \widehat{C}_{\mathsf{Bool}}$,

- Case 1: If every wire in $\phi(w)$ is also (along with associated values) included in $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$. The argument proceeds in two steps:
- Case 2: If only a proper subset S of wires in $\phi(w)$ is (along with associated values) included in $\mathcal{S}^{\mathsf{NB}}_{\mathsf{leak}}$ then the simulation of the values for the wires in S is perfect.

We prove this by hybrid argument.

Hyb_1: The output of this hybrid is the leakage on the computation of $\widehat{C}_{\mathsf{NB}}$ on $\widehat{x}$. Denote this set by $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},1}$.

Hyb_2: Let $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},1}$ be the output of the leakage on the computation of $\widehat{C}_{\mathsf{NB}}$ on $\widehat{x}$. For every wire $w \in \widehat{C}_{\mathsf{Bool}}$ such that $\phi(w) \not\subset \mathsf{Marg}(\mathcal{S}_{\mathsf{leak}})$, do the following: for every $w_i \in \phi(w)$ and $(w_i, v_w^i) \in \mathcal{S}_{\mathsf{leak}}$ for some v_w^i, remove (w_i, v_w^i) from $\mathcal{S}_{\mathsf{leak}}$ and include (w_i, v') in $\mathcal{S}_{\mathsf{leak}}$ for a freshly sampled random bit v'. Call the new set $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},2}$.

The new set $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},2}$ is distributed identically to $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},1}$ – this follows from the fact that any proper subset of additive shares is distributed identical to uniform distribution.

Hyb_3: The output of this hybrid is the output of $\mathsf{Sim}^{\mathsf{NB}}(\widehat{C})$, namely $\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},3}$.

The only difference between this hybrid and the previous hybrid is the following: (i) for every wire in $\widehat{C}$ such that the simulation of values for the wires in $\phi(w) \subseteq \mathsf{Marg}(\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},2})$ is performed using the leakage of $\widehat{C}$ on $\widehat{x}$, (ii) for every wire in $\widehat{C}_{\mathsf{Bool}}$ such that the simulation of values for the wires in $\phi(w) \subseteq \mathsf{Marg}(\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},2})$ is performed using $\mathsf{Sim}^{\mathsf{Bool}}$. In order to invoke the security of $\mathsf{CC}_{\mathsf{Bool}}$, we need to argue that the probability that $\phi(w) \subseteq \mathsf{Marg}(\mathcal{S}_{\mathsf{leak}}^{\mathsf{NB},2})$ is $\mathbf{p}(=\mathbf{p}_{\mathsf{NB}}^{\ell})$. This in turn follows from the fact that $\phi(w)$ consists of ℓ wires and all of them leak independently with probability $\mathbf{p}_{\mathsf{NB}}$.

□

□

□

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.

5.1 Construction

We prove the following proposition.

Proposition 6. *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 $\eta > 0$.*

Proof. We present the construction in Figure 3.

Consider the following claims.

Claim 8. *The correctness of CC_{comp} implies the correctness of CC_{LT}.*

Proof. We need to show that $\widehat{C}(x) = C(x)$, where $C \in \mathcal{C}$ and $\widehat{C} \leftarrow \mathsf{CC}_{comp}.\mathsf{Compile}(C)$. Note that $\widehat{C}(x) = \widehat{C}_{comp}(\widehat{x})$, where $\widehat{C}_{comp} \leftarrow \mathsf{CC}_{comp}.\mathsf{Compile}(C)$ and $\widehat{x}$ is the XOR secret sharing of x. Moreover, $\mathsf{CC}_{LT}.\mathsf{Decode} = \mathsf{CC}_{comp}.\mathsf{Decode}$.

From the correctness property of CC_{comp} we have that $\mathsf{CC}_{comp}.\mathsf{Decode}\left(\widehat{C}_{comp}(\widehat{x})\right) = C(x)$. This proves the claim. □

Claim 9. *The $(\mathbf{p},\varepsilon)$-composable security of CC_{comp} implies the $(\mathbf{p},\mathbf{p}',\varepsilon')$-leakage tolerance of CC_{LT}.*

Proof. We first present the description of the simulator.

$\mathsf{Sim}_{LT}(C, \mathcal{S}_{\mathsf{leak}}^{I})$: It takes as input circuit C, leaked set $\mathcal{S}_{\mathsf{leak}}^{I}$ of input wires. Let n be the input length of C.

Consider the following observation: the i^{th} bit of x_i is hidden if (i) the two wires carrying x_i are not leaked, (ii) the two wires carrying $r_{1,0}^i$ are not leaked and, (iii) two wires carrying $r_{1,1}^i$ are not leaked. This can be characterized as a binary string of length six. Define GoodSet $= \{000000\}$ – the first two bits of

Construction of CC_{LT}

- **Circuit compilation,** $\mathsf{CC}_{LT}.\mathsf{Compile}(C)$: On input a circuit C, it constructs a circuit $\widehat{C}$. On input x, the circuit $\widehat{C}$ does the following:
  - **Phase I**: For every i^{th} bit in x, it computes two sets of XOR secret shares of x_i. Set $\widehat{x}$ to be the concatenation of all the shares. In particular, the n shares of x_i is computed by first sampling bits $r^i_{1,b}, \ldots, r^i_{n-1,b}$ uniformly at random for $b \in \{0,1\}$ and then computing,
  $$r^i_{n,b} = \left(\left(\cdots (x^i_{1,b} \oplus r^i_{1,b}) \oplus r^i_{2,b} \cdots\right) \oplus r^i_{n-1,b}\right)$$
  Since there are two wires carrying x_i, there are two sets of XOR shares of x_i, namely $r^i_{1,0}, \ldots, r^i_{n,0}$ and $r^i_{1,1}, \ldots, r^i_{n,1}$.
  - **Phase II**: Generate $\widehat{C} \leftarrow \mathsf{CC}_{comp}.\mathsf{Compile}(C)$. Compute $\widehat{C}_{comp}(\widehat{x})$ to obtain $\widehat{y}$.
  - Output $\widehat{y}$.

  Output $\widehat{C}$.

- **Output encoding,** $\mathsf{CC}_{LT}.\mathsf{Decode}(\widehat{y})$: It reconstructs the XOR secret shares of every bit of y. Output y.

Figure 3: Construction of CC_{LT}

000000 indicates sub-case (i), third and forth bits indicates sub-case (ii) and fifth and sixth bits indicate sub-case (iii) defined above. More generally, we can define a binary string $b_1 \cdots b_6$ of length six to be one, where $b_1 = 1$ only if first input wire carrying x_i is leaked, $b_2 = 1$ indicates that the second bit is leaked only if the second input wire of x_i is leaked and so on. Let ℓ be the input length of x. Sample ℓ times, with repetition, from the distribution $\mathcal{D}$ defined on set of all strings $\{0,1\}^6$. In more detail, the sampling of a string in $\{0,1\}^6$ proceeds by running six independent trials, where in each trial 0 (denoting not leaked) is sampled with probability $1 - \mathbf{p}$ and 1 (denoting leaked) is sampled with probability $\mathbf{p}$. The resulting sampled strings are denoted by $s_1, \ldots, s_\ell$. We emphasize that the strings $s_1, \ldots, s_\ell$ need not be distinct. If $|\{s_1, \ldots, s_\ell\} \cap \text{GoodSet}| \leq 2\ell - |\mathcal{S}^I_{\mathsf{leak}}|$ then abort, where $\{s_1, \ldots, s_\ell\}$ is a multi-set. Otherwise, let ϕ be a random permutation on $[\ell]$ subject to the constraint $s_{\phi(i)} \notin \text{GoodSet}$ if and only if $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$, where w is the wire carrying the i^{th} input bit.

The simulation proceeds in two steps: in the first step, Phase I is simulated, i.e., the leakage on the encoding of the input bit is simulated. We sub-divide the set of the wires in Phase I into sets $\mathcal{W}_1$ and $\mathcal{W}_2$. The set $\mathcal{W}_1$ consists of all wires w such that w carries either an input bit x_i or it carries a random bit $r^i_{1,b}$, for some $i \in [\ell]$ and $b \in \{0,1\}$. The set $\mathcal{W}_2$ is the complement set of $\mathcal{W}_1$, i.e., it consists of all the wires in Phase I that are already not present in $\mathcal{W}_1$.

Construct the set $\mathcal{S}^1_{\mathsf{leak}}$ consisting of simulated wire values in Phase I. But first we assign values to the wires in Phase I. There are two cases:

- Case 1: Assigning values for wires in $\mathcal{W}_1$. For every $i \in [\ell]$, if $s_{\phi(i)} \notin$ GoodSet, assign the value v_w to the wire w carrying the i^{th} input bit, where $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$. In this case, also assign a value $v^i_{1,b}$ to the wire carrying the random bit $r^i_{1,b}$ for $b \in \{0,1\}$, where $v^i_{1,b}$ is a bit sampled uniformly at random.

- Case 2: Assigning values for wires in $\mathcal{W}_2$. For every wire $w \in \mathcal{W}_2$, assign v_w, where v_w is computed as follows: (i) if w is either an input wire, v_w is sampled uniformly at random, (ii) if w is the output wire of a gate whose both input wires are unassigned then v_w is sampled uniformly at random, (iii) otherwise, set v_w to be the output of G on the values assigned to both the input wires.

Now, we construct $\mathcal{S}^1_{\mathsf{leak}}$ according to the two cases: for every wire w in Phase I,

- Case 1: $w \in \mathcal{W}_1$. We are only concerned with the case when w is assigned a value v_w in the above process. Let $i \in [\ell]$ be such that w carries one of the following variables: x_i, $r^i_{1,0}$ or $r^i_{1,1}$. If w carries the variable x_i and if the corresponding bit in $s_{\phi(i)}$ is set to 1, then include $(w, v_w) \in \mathcal{S}^1_{\mathsf{leak}}$. If the corresponding bit is 0, don't include. To illustrate, if w is the first wire that carries the variable x_i and if $s_{\phi(i)}$ is of the form $1 \star\star\star\star\star$ then include (w, v_w) in $\mathcal{S}^1_{\mathsf{leak}}$. Similarly, if w is the second input wire that carries the variable x_i and $s_{\phi(i)}$ is of the form $\star 1 \star\star\star\star$ then include (w, v_w) in $\mathcal{S}^1_{\mathsf{leak}}$, and so on. Note that if w is unassigned by the above process then it will be, by definition, not included in $\mathcal{S}^1_{\mathsf{leak}}$.
- Case 2: $w \in \mathcal{W}_2$. Include (w, v_w) in $\mathcal{S}^1_{\mathsf{leak}}$ with probability $\mathbf{p}$, where v_w is picked uniformly at random.

This concludes the simulation of wires in Phase I.

In the second step of the simulation, simulate the leakage on the computation of $\widehat{C}$. Let the partial simulator of CC_{comp} be $\mathsf{Sim}_{comp} = (\mathsf{Sim}^{SC}_1, \mathsf{Sim}^{SC}_2)$. Include every internal or output wire w of $\widehat{C}$ in $\mathcal{W}_{lk}$ with probability $\mathbf{p}$. For every input wire w of $\widehat{C}$, include w in $\mathcal{W}_{lk}$ if and only if $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$ for some bit v_w.

Compute $\mathsf{Sim}^{SC}_1(\widehat{C}_{comp}, \mathcal{W}_{lk})$ to obtain $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. Construct the set S^{inp} as follows. For every $w \in \mathcal{W}^{inp}$, include (w, v_w) in S^{inp} where $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$, if not v_w is sampled at random subject to the condition that it is consistent with the other leaked values[12]. The set S^{out} is constructed by including $(w, v_w) \in S^{out}$ for every $w \in \mathcal{W}^{out}$ and v_w is picked uniformly at random. Compute $\mathsf{Sim}_2(\widehat{C}, \mathcal{W}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)$ to obtain the set $\mathcal{S}^{SC}_{\mathsf{leak}}$. If Sim_2 aborts then Sim also aborts.

Output of Sim is $\mathcal{S}_{\mathsf{leak}} \cup \mathcal{S}^{SC}_{\mathsf{leak}}$.

Conditioned on the event that Sim does not abort, the output distribution of $\mathsf{Sim}(\widehat{C}, L_{inp}(x))$ is identically distributed to the leakage of $\widehat{C}$ on $\widehat{x}$. This follows from the perfect simulation of the wires in the input encoding sub-circuit and the $(\mathbf{p}, \varepsilon)$-simulation with abort property of CC_{comp} that guarantees that the output of Sim_2 is identically distributed to the real leakage conditioned on Sim_2 not aborting.

Claim 10. *Suppose* $\mathbf{p}' = (1+\eta)^2(1-(1-\mathbf{p})^6)$*, for some arbitrarily small constant* $\eta > 0$*. The probability that* Sim *aborts is* $\varepsilon' \leq \varepsilon + \frac{1}{e^{c \cdot n}}$*, for some constant* c*.*

Proof. We note that Sim aborts under the following conditions:

- The simulator of CC_{comp} aborts.
- If $|\{s_1, \ldots, s_n\} \cap \text{GoodSet}| \leq 2n - |\mathcal{S}^I_{\mathsf{leak}}|$.

Moreover, the above two events are independent. From the security of CC_{comp}, the probability that the simulator of CC_{comp} aborts is ε. Thus, we need to calculate the probability that $|\{s_1, \ldots, s_n\} \cap \text{GoodSet}| \leq 2n - |\mathcal{S}^I_{\mathsf{leak}}|$. Rephrasing this, we need to calculate the probability that the cardinality of subset of $\{s_1, \ldots, s_n\}$, that do not belong to GoodSet, is greater than the number of leaked inputs.

Define a random variable $\mathbf{X}_i$ for every $i \in [n]$ such that $\mathbf{X}_i = 1$ if there exists $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$ such that the wire w carries the i^{th} bit of the input and for some bit v_w. Otherwise, $\mathbf{X}_i = 0$. Note that $\Pr[\mathbf{X}_i = 1] = \mathbf{p}'$. Define a random variable $\mathbf{Y}_i$ for every $i \in [n]$ such that $\mathbf{Y}_i = 1$ if $s_{\phi(i)} \notin \text{GoodSet}$. Otherwise, $\mathbf{Y}_i = 0$. Note that $\Pr[\mathbf{Y}_i = 1] = 1 - (1-\mathbf{p})^6$.

Denote $\mathbf{X} = \sum_{i=1}^n \mathbf{X}_i$ and $\mathbf{Y} = \sum_{i=1}^n \mathbf{Y}_i$. Set $t = n(1+\eta)(1-(1-\mathbf{p})^6)$. Set $\delta_1 = \eta$ and $\delta_2 = 1 - \frac{1}{(1+\eta)}$.

[12]For instance, if w is the output wire of G and if the values to both the input wires of G are already assigned, then assign the value to w to be the output of G.

$$\begin{aligned}
\mathsf{Pr}[\mathbf{X}-\mathbf{Y}\geq 0] &\geq \mathsf{Pr}[\mathbf{X}<t \text{ and } \mathbf{Y}>t]\\
&= \mathsf{Pr}[\mathbf{X}<t]\cdot\mathsf{Pr}[\mathbf{Y}>t]\\
&= \mathsf{Pr}[\mathbf{X}<(1+\eta)\mathbb{E}[\mathbf{X}]]\cdot\mathsf{Pr}[\mathbf{Y}>\frac{1}{(1+\eta)}\mathbb{E}[\mathbf{Y}]]\\
&= \mathsf{Pr}[\mathbf{X}<(1+\delta_1)\mathbb{E}[\mathbf{X}]]\cdot\mathsf{Pr}[\mathbf{Y}>(1-\delta_2)\mathbb{E}[\mathbf{Y}]]\\
&\geq \left(1-\frac{1}{e^{\frac{\delta_1^2\mathbb{E}[\mathbf{X}]}{3}}}\right)\cdot\left(1-\frac{1}{e^{\frac{\delta_2^2\mathbb{E}[\mathbf{X}]}{2}}}\right) \text{ (by Chernoff Bounds)}\\
&\geq \left(1-\frac{1}{e^{\frac{c_1\cdot n}{3}}}\right)\cdot\left(1-\frac{1}{e^{\frac{c_2\cdot n}{2}}}\right) \text{ (for some constants } c_1,c_2)\\
&\geq 1-\frac{1}{e^{c\cdot n}} \text{ (for some constant } c)
\end{aligned}$$

□

□

□

We combine Propositions 4 and 6 to obtain the following proposition.
Combining with Proposition 4 obtain the following proposition.

Proposition 7. *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 $\mathbf{p}=3\times 10^{-8}$ and $\mathbf{p}'=2\times 10^{-7}$.*

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 8. *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.*

Proof. The proof of this theorem follows the same template as Theorem 6. We describe the construction in Figure 4.

Consider the following claims.

Claim 11. *The correctness of CC_{comp} implies the correctness of CC_{LT}.*

The proof of the above claim is identical to the proof of Claim 8.

Claim 12. *The $(\mathbf{p},\varepsilon)$-composable security of CC_{LT} implies the $(\mathbf{p},\mathbf{p}',\varepsilon')$-leakage tolerance of CC_{LT}.*

Proof. We first present the description of the simulator.

$\mathsf{Sim}_{LT}(C,\mathcal{S}^I_{\mathsf{leak}})$: It takes as input circuit C, leaked set $\mathcal{S}^I_{\mathsf{leak}}$ of input wires. Let n be the input length of C.

Consider the following observation: the i^{th} bit of x_i is hidden if all of the following conditions hold: (i) the two wires carrying x_i are not leaked, (ii) $\exists j\in[n]$ such that the wire carrying $r^i_{j,0}$ is not leaked, (iii) $\exists j\in[n]$ such that the wire carrying $r^i_{j,1}$ is not leaked. As before, this can be characterized as binary strings of length $2n+2$. Define GoodSet to consist of all strings of the following form: the first two bits is 00, followed by a n-bit string containing at least one 0, which is followed by a n-bit string that also contains at least one 0. Let ℓ be the input length of x. Sample ℓ times, with repetition, from the distribution $\mathcal{D}$ defined on set of

Construction of CC_{LT}

- **Circuit compilation,** $\mathsf{CC}_{LT}.\mathsf{Compile}(C)$: On input a circuit C, it constructs a circuit $\widehat{C}$. On input x, the circuit $\widehat{C}$ does the following:
  - **Phase I**: For every i^{th} bit in x, it computes two sets of XOR secret shares of x_i. Set $\widehat{x}$ to be the concatenation of all the shares. In particular, a pair of n shares of x_i is denoted by $(r^i_{1,0}, \ldots, r^i_{n,0})$ and $(r^i_{1,1}, \ldots, r^i_{n,1})$ subject to the constraint that $x_i = \oplus_{j=1}^n r^i_{j,0}$ and $x_i = \oplus_{j=1}^n r^i_{j,1}$. This can be computed by two randomized functions in $\mathbb{B}'$ mapping 1 bit to n bits.
  - **Phase II**: Generate $\widehat{C} \leftarrow \mathsf{CC}_{comp}.\mathsf{Compile}(C)$. Compute $\widehat{C}_{comp}(\widehat{x})$ to obtain $\widehat{y}$.
  - Output $\widehat{y}$.

  Output $\widehat{C}$.

- **Output encoding,** $\mathsf{CC}_{LT}.\mathsf{Decode}(\widehat{y})$: It reconstructs the XOR secret shares of every bit of y. Output y.

Figure 4: Construction of CC_{LT}

all strings $\{0,1\}^{2n+2}$. The sampling of a string in $\{0,1\}^{2n+2}$ proceeds by running $2n+2$ independent trials, where in each trial 0 (denoting not leaked) is sampled with probability $1 - \mathbf{p}$ and 1 (denoting leaked) is sampled with probability $\mathbf{p}$. The resulting sampled strings are denoted by $s_1, \ldots, s_\ell$. We emphasize that the strings $s_1, \ldots, s_\ell$ need not be distinct. If $|\{s_1, \ldots, s_\ell\} \cap \text{GoodSet}| \leq 2\ell - |\mathcal{S}^I_{\mathsf{leak}}|$ then abort, where $\{s_1, \ldots, s_\ell\}$ is a multi-set. Otherwise, let ϕ be a random permutation on $[\ell]$ subject to the constraint $s_{\phi(i)} \notin \text{GoodSet}$ if and only if $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$, where w is the wire carrying the i^{th} input bit.

The simulation proceeds in two steps: in the first step, Phase I is simulated, i.e., the leakage on the encoding of the input bit is simulated. Construct the set $\mathcal{S}^1_{\mathsf{leak}}$ as follows.

- For every wire w carrying the variable x_i, include $(w, v_w) \in \mathcal{S}^1_{\mathsf{leak}}$, if it holds that (i) $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$ and, (ii) $s_{\phi(i)} = 11\star\cdots\star$.
- For every $i \in [\ell]$ and $s_{\phi(i)} \notin \text{GoodSet}$, consider the following scenarios: (i) if $s_{\phi(i)} = \star\star1\cdots1\star\cdots\star$, i.e., every bit in the third position through the $(n+2)^{th}$ position of $s_{\phi(i)}$ is 1. Include $(w^i_{j,0}, v^i_{j,0}) \in \mathcal{S}^1_{\mathsf{leak}}$, where w^i_j is the wire carrying the variable $r^i_{j,0}$ and $v^i_{j,0}$ is sampled uniformly at random subject to the condition that $\oplus_{i=1}^n v^i_{j,0} = x_i$, (ii) if $s_{\phi(i)} = \star\star1\cdots1$, i.e., every bit in the $(n+3)^{th}$ position through the $(2n+2)^{th}$ position of $s_{\phi(i)}$ is 1 and, (iii) otherwise, for every wire $w^i_{j,0}$ carrying the variable $r^i_{j,b}$, if the $(2+b\cdot n+j)^{th}$ bit of $s_{\phi(i)}$ is set to 1 then include $(w^i_{j,b}, v) \in \mathcal{S}^1_{\mathsf{leak}}$ for a randomly sampled bit v.
- For every $i \in [\ell]$ and $s_{\phi(i)} \in \text{GoodSet}$, for any wire $w^i_{j,b}$ carrying the variable $r^i_{j,b}$, if the $(2+b\cdot n+j)^{th}$ bit of $s_{\phi(i)}$ is set to 1 then include $(w^i_{j,b}, v) \in \mathcal{S}^1_{\mathsf{leak}}$ for a randomly sampled bit v.

This concludes the simulation of wires in Phase I.

In the second step of the simulation, simulate the leakage on the computation of $\widehat{C}_{comp}$. Let the partial simulator of CC_{comp} be $\mathsf{Sim}_{comp} = (\mathsf{Sim}^{SC}_1, \mathsf{Sim}^{SC}_2)$. Include every internal or output wire w of $\widehat{C}$ in $\mathcal{W}_{lk}$ with probability $\mathbf{p}$. For every input wire w of $\widehat{C}$, include w in $\mathcal{W}_{lk}$ if and only if $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$ for some bit v_w.

Compute $\mathsf{Sim}^{SC}_1(\widehat{C}_{comp}, \mathcal{W}_{lk})$ to obtain $(\mathcal{W}_{lk}, \mathcal{W}^{inp}, \mathcal{W}^{out}, I)$. Construct the set S^{inp} as follows. For every $w \in \mathcal{W}^{inp}$, include (w, v_w) in S^{inp} where $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$, if not v_w is sampled at random subject to the condition that it is consistent with the other leaked values[13]. The set S^{out} is constructed by including $(w, v_w) \in S^{out}$ for every $w \in \mathcal{W}^{out}$ and v_w is picked uniformly at random. Compute $\mathsf{Sim}_2(\widehat{C}, \mathcal{W}, \mathcal{W}^{inp}, S^{inp}, \mathcal{W}^{out}, S^{out}, I)$ to obtain the set $\mathcal{S}^{SC}_{\mathsf{leak}}$. If Sim_2 aborts then Sim also aborts.

Output of Sim is $\mathcal{S}_{\mathsf{leak}} \cup \mathcal{S}^{SC}_{\mathsf{leak}}$.

Conditioned on the event that Sim does not abort, the output distribution of $\mathsf{Sim}(\widehat{C}, L_{inp}(x))$ is identically distributed to the leakage of $\widehat{C}$ on $\widehat{x}$. This follows from the perfect simulation of the wires in the input encoding sub-circuit and the $(\mathbf{p}, \varepsilon)$-simulation with abort property of CC_{comp} that guarantees that the output of Sim_2 is identically distributed to the real leakage conditioned on Sim_2 not aborting.

Claim 13. *Suppose* $\mathbf{p}' = (1+\eta)^2(1-((1-\mathbf{p})^2)\cdot(1-\mathbf{p}^n)^2))$, *for some arbitrarily small constant* $\eta > 0$. *The probability that* Sim *aborts is* $\varepsilon' \leq \varepsilon + \frac{1}{e^{c\cdot n}}$, *for some constant c.*

Proof. We note that Sim aborts under the following conditions:

- The simulator of CC_{comp} aborts.
- If $|\{s_1, \ldots, s_n\} \cap \text{GoodSet}| \leq 2n - |\mathcal{S}^I_{\mathsf{leak}}|$.

Moreover, the above two events are independent. From the security of CC_{comp}, the probability that the simulator of CC_{comp} aborts is ε. Thus, we need to calculate the probability that $|\{s_1, \ldots, s_n\} \cap \text{GoodSet}| \leq 2n - |\mathcal{S}^I_{\mathsf{leak}}|$. Rephrasing this, we need to calculate the probability that the cardinality of subset of $\{s_1, \ldots, s_n\}$, that do not belong to GoodSet, is greater than the number of leaked inputs.

Define a random variable $\mathbf{X}_i$ for every $i \in [n]$ such that $\mathbf{X}_i = 1$ if there exists $(w, v_w) \in \mathcal{S}^I_{\mathsf{leak}}$ such that the wire w carries the i^{th} bit of the input and for some bit v_w. Otherwise, $\mathbf{X}_i = 0$. Note that $\mathsf{Pr}[\mathbf{X}_i = 1] = \mathbf{p}'$. Define a random variable $\mathbf{Y}_i$ for every $i \in [n]$ such that $\mathbf{Y}_i = 1$ if $s_{\phi(i)} \notin \text{GoodSet}$. Otherwise, $\mathbf{Y}_i = 0$. Note that $\mathsf{Pr}[\mathbf{Y}_i = 1] = (1-(1-\mathbf{p})^2)(1-\mathbf{p}^n) + \mathbf{p}^n$. Also, define the following events:

- $\mathsf{OneWire}_i$: one of the wires carrying x_i is leaked.
- $\mathsf{NotAllZero}_i$: Not all the wires carrying $r^i_{j,0}$ are leaked.
- $\mathsf{NotAllOne}_i$: Not all the wires carrying $r^i_{j,1}$ are leaked.
- All_i: For every $j \in [\ell]$, all the wires carrying $r^i_{j,0}$ is leaked OR for every $j \in [\ell]$, all the wires carrying $r^i_{j,1}$ is leaked.

Consider the following quantity:

$$\begin{aligned}
\mathsf{Pr}[\mathbf{Y}_i = 1] &= \mathsf{Pr}\left[(\mathsf{OneWire}_i \wedge \mathsf{NotAllZero}_i \wedge \mathsf{NotAllOne}_i) \vee (\mathsf{All}_i)\right] \\
&= \mathsf{Pr}[(\mathsf{OneWire}_i \wedge \mathsf{NotAllZero}_i \wedge \mathsf{NotAllOne}_i)] + \mathsf{Pr}[\mathsf{All}_i] \\
&= \mathsf{Pr}[\mathsf{OneWire}_i] \cdot \mathsf{Pr}[\mathsf{NotAllZero}_i] \cdot \mathsf{Pr}[\mathsf{NotAllOne}_i] + \mathsf{Pr}[\mathsf{All}_i] \\
&= (1-(1-\mathbf{p})^2)\cdot(1-\mathbf{p}^n)\cdot(1-\mathbf{p}^n) + (1-(1-\mathbf{p}^n)^2) \\
&= 1-((1-\mathbf{p})^2)\cdot(1-\mathbf{p}^n)^2)
\end{aligned}$$

Denote $\mathbf{X} = \sum_{i=1}^n \mathbf{X}_i$ and $\mathbf{Y} = \sum_{i=1}^n \mathbf{Y}_i$. Set $t = n(1+\eta)\left(1-((1-\mathbf{p})^2)\cdot(1-\mathbf{p}^n)^2)\right)$. Set $\delta_1 = \eta$ and $\delta_2 = 1 - \frac{1}{(1+\eta)}$.

[13]For instance, if w is the output wire of G and if the values to both the input wires of G are already assigned, then assign the value to w to be the output of G.

$$\begin{aligned}
\mathsf{Pr}[\mathbf{X}-\mathbf{Y}\geq 0] &\geq \mathsf{Pr}[\mathbf{X}<t \text{ and } \mathbf{Y}>t]\\
&= \mathsf{Pr}[\mathbf{X}<t]\cdot\mathsf{Pr}[\mathbf{Y}>t]\\
&= \mathsf{Pr}[\mathbf{X}<(1+\eta)\mathbb{E}[\mathbf{X}]]\cdot\mathsf{Pr}[\mathbf{Y}>\frac{1}{(1+\eta)}\mathbb{E}[\mathbf{Y}]]\\
&= \mathsf{Pr}[\mathbf{X}<(1+\delta_1)\mathbb{E}[\mathbf{X}]]\cdot\mathsf{Pr}[\mathbf{Y}>(1-\delta_2)\mathbb{E}[\mathbf{Y}]]\\
&\geq \left(1-\frac{1}{e^{\frac{\delta_1^2\mathbb{E}[\mathbf{X}]}{3}}}\right)\cdot\left(1-\frac{1}{e^{\frac{\delta_2^2\mathbb{E}[\mathbf{X}]}{2}}}\right) \text{ (by Chernoff Bounds)}\\
&\geq \left(1-\frac{1}{e^{\frac{c_1\cdot n}{3}}}\right)\cdot\left(1-\frac{1}{e^{\frac{c_2\cdot n}{2}}}\right) \text{ (for some constants } c_1,c_2)\\
&\geq 1-\frac{1}{e^{c\cdot n}} \text{ (for some constant } c)
\end{aligned}$$

□

□

□

From the above proposition, we have the following theorem. As remarked earlier, we can achieve the above theorem with deterministic basis with a simple modification of the above analysis [14].

Theorem 8. *Consider any constant $0<\mathbf{p}<\mathbf{p}'<1$ and let $\mathbb{B}$ denote a basis. For some constant δ, there is a construction of $(\mathbf{p},\mathbf{p}',\exp(-s))$-leakage tolerant circuit compiler over basis $\mathbb{B}'$ for all circuits of size s over basis $\mathbb{B}$, where $\mathbb{B}'$ consists of all functions mapping $2\cdot\min(\lceil\frac{\log(\delta)}{\log(\mathbf{p})}\rceil,2)$ bits to $2\cdot\min(\lceil\frac{\log(\delta)}{\log(\mathbf{p})}\rceil,2)$ bits.*

5.2 Negative Result

We present a negative result on the leakage rate of a leakage tolerant circuit compiler. Before that we consider an alternative definition, where the gates are leaked instead of wire values. That is, for every gate with probability $\mathbf{p}$, both its input wire values and its output wire values are leaked. We term this as gate probing attacks, which we formally define this below.

Step I: Gate Probing Attacks. Every gate in the computation of the compiled circuit $\widehat{C}$ on input encodings $\{\widehat{x}\}$ is leaked independently with probability $\mathbf{p}$.

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

$L_{comp}(\widehat{C},\widehat{x})$: construct the set of leaked values $\mathcal{S}^{C}_{\mathsf{leak}}$ as follows. For every gate G in $\widehat{C}$ and values $(v_{w_1},v_{w_2},v_{w_3})$ assigned to the input and output wires of G, include $(G,v_{w_1},v_{w_2},v_{w_3})$ in $\mathcal{S}^{C}_{\mathsf{leak}}$ with probability $\mathbf{p}$. Output $\mathcal{S}^{C}_{\mathsf{leak}}$.

$L_{inp}(x)$: 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 $\mathbf{p}'$. Also, include (w',x_i) in $\mathcal{S}^{I}_{\mathsf{leak}}$, where w' is an input wire carrying x_i. Output $\mathcal{S}^{I}_{\mathsf{leak}}$.

We define leakage tolerance against random probing attacks below.

[14] In particular, instead of having the function producing the secret shares, we can require that the function takes as input all the random bits and outputs the XORed value.

Definition 13 (Leakage Tolerance Against Random Gate Probing Attacks)**.** *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *for a family of circuits* $\mathcal{C}$ *is said to be* $(\mathbf{p}, \mathbf{p}', \varepsilon)$*-leakage tolerant against random gate probing attacks if* CC *is* ε*-leakage tolerant against* $\mathcal{L}^{G}_{\mathbf{p},\mathbf{p}'}$*.*

Step II: From Wire to Gate Leakage Security. We show that any circuit compiler that is secure against $\mathbf{p}$-random wire probing attacks, is also secure against $\mathbf{p}^*$-random gate probing attacks for some $\mathbf{p}^*$.

Proposition 9. *Consider a circuit compiler* CC *for* $\mathcal{C}$ *over boolean basis* $\mathbb{B}$ *that is* $(\mathbf{p}, \mathbf{p}', \varepsilon)$*-leakage tolerant against random (wire) probing attacks. Then,* CC *is* $(\mathbf{p}^*, \mathbf{p}', \varepsilon)$*-leakage tolerant against random gate probing attacks for* $\mathcal{C}$ *over* $\mathbb{B}$*, where* $\mathbf{p}^* = \mathbf{p}^2(1 - (1 - \mathbf{p})^2)$*.*

Proof. To prove this proposition, we first introduce some notation. We define the leakage distribution on the computation of $\widehat{C}$ on $\widehat{x}$ to be $\mathsf{RPDistr}^{g}_{\mathbf{p}}$.

Sampler $\mathsf{RPDistr}^{g}_{\mathbf{p}^*}(\widehat{C}, \widehat{x})$: Denote the set of gates in $\widehat{C}$ as $\mathcal{G}$. Consider the computation of $\widehat{C}$ on input encoding $\widehat{x}$. For every gate $G \in \mathcal{G}$, denote $\mathbf{val}(G)$ to be the set of values assigned to the input wires and the output wires of G during the evaluation of $\widehat{C}$ on $\widehat{x}$.

We construct set $\mathcal{S}_{\mathsf{leak}}$ as follows: initially $\mathcal{S}_{\mathsf{leak}}$ is assigned to be $\{\}$. For every $G \in \mathcal{G}$, with probability $\mathbf{p}^*$, include $(G, \mathbf{val}(G)$ in $\mathcal{S}_{\mathsf{leak}}$. Output $\mathcal{S}_{\mathsf{leak}}$.

We also consider a hybrid distribution the following distribution that will be useful for the proof.

Sampler $\mathcal{D}^{w}_{\mathbf{p}}(\widehat{C}, \widehat{x})$: Denote the set of wires in $\widehat{C}$ as $\mathcal{W}$[15]. 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 set S as follows: initially S is assigned to be $\{\}$. For every $w \in \mathcal{W}$, with probability $\mathbf{p}$, include $(w, \mathbf{val}(w))$ in S (i.e., with probability $(1-\mathbf{p})$ the pair $(w, \mathbf{val}(w))$ is not included). Construct the set of leaked wire values $\mathcal{S}_{\mathsf{leak}}$ as follows: for every gate $G \in C$ with input wires w_1^{inp}, w_2^{inp} and one of the two output wires w^{out}, include $(w_1^{inp}, b_1^{inp}), (w_2^{inp}, b_2^{inp}), (w^{out}, b^{out}) \in \mathcal{S}_{\mathsf{leak}}$ if and only if $(w_1^{inp}, b_1^{inp}), (w_2^{inp}, b_2^{inp}), (w^{out}, b^{out}) \in S$ for some $b_1^{inp}, b_2^{inp}, b^{out} \in \{0, 1\}$. Furthermore, if there exists wire w' such that w' carries the same value as w (for instance, w' and w are two output wires of the same gate) and if $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$, then also include (w', v_w) in $\mathcal{S}_{\mathsf{leak}}$.

Output $\mathcal{S}_{\mathsf{leak}}$.

It immediately follows that the distributions $\mathcal{D}^{w}_{\mathbf{p}}$ and $\mathsf{RPDistr}^{g}_{\mathbf{p}^*}$ are identical: the probability $\mathbf{p}^*$ that any given gate is leaked is the same as the probability that both its input wires and one of its output wire is leaked. Since, every wire is leaked independently, we have $\mathbf{p}^* = 2\mathbf{p}^3(1 - \mathbf{p}) + \mathbf{p}^4$.

$$\begin{aligned}
\mathbf{p}^* &= \mathsf{Pr}\left[\ell_{in} \text{ input wires of } G \text{ are leaked } \wedge \text{ one of } \textit{two} \text{ output wires of } G \text{ is leaked}\right] \\
&= \mathsf{Pr}\left[\ell_{in} \text{ input wires of } G \text{ are leaked}\right] \cdot \mathsf{Pr}\left[\text{one of output wires of } G \text{ is leaked}\right] \\
&= \mathbf{p}^2 \cdot (1 - \mathsf{Pr}\left[\text{both the output wires of } G \text{ are not leaked}\right]) \\
&= \mathbf{p}^2 \cdot (1 - (1 - \mathbf{p})^2)
\end{aligned}$$

It remains to show that CC is secure with respect to the distribution $\mathcal{D}^{w}_{\mathbf{p}}$ of wire probing attacks. Suppose $\mathsf{Sim}_{\mathbf{p}}$ is a PPT simulator that simulates the leakage $\mathcal{L}_{\mathbf{p},\mathbf{p}'}$ (Section 3.2). We construct a PPT simulator $\mathsf{Sim}^{g}_{\mathbf{p}}$ as follows: on input circuit C, it executes $\mathsf{Sim}_{\mathbf{p}}$ to obtain the set of leaked wire values S. Output a subset $\mathcal{S}_{\mathsf{leak}} \subseteq S$ such that for every gate G with input wires w_1^{inp}, w_2^{inp} and w^{out}, include $(w_1^{inp}, b_1^{inp}), (w_2^{inp}, b_2^{inp}), (w^{out}, b^{out})$ in $\mathcal{S}_{\mathsf{leak}}$ if and only if $(w_1^{inp}, b_1^{inp}), (w_2^{inp}, b_2^{inp}), (w^{out}, b^{out}) \in S$ for some $b_1^{inp}, b_2^{inp}, b^{out} \in \{0, 1\}$. As before, include (w', v_w) in $\mathcal{S}_{\mathsf{leak}}$ if $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$ and if w and w' carry the same value in $\widehat{C}$. The statistical distance between the output distributions of $\mathsf{Sim}^{g}_{\mathbf{p}}$ and $\mathcal{D}^{w}_{\mathbf{p}}$ is at most ε; this

[15]Suppose a gate has two output wires, then including one of the output wires in $\mathcal{W}$ means including also the other one.

follows from the security of CC against $\mathbf{p}$-random wire probing attacks. And thus, the statistical distance between the output distributions of $\mathsf{Sim}^g_{\mathbf{p}}$ and $\mathsf{RPDistr}^g_{\mathbf{p}'}$ is at most ε. This completes the proof. $\square$

We also consider a generalization of the above proposition for circuits over arbitrary basis (not necessarily boolean).

Proposition 10. *Consider a basis $\mathbb{B}$ such that every gate in this basis maps ℓ_{in} input bits to ℓ_{out} output bits. Consider a circuit compiler CC for $\mathcal{C}$ over $\mathbb{B}$ that is $(\mathbf{p}, \mathbf{p}', \varepsilon)$-leakage tolerant against random probing attacks. Then, CC is $(\mathbf{p}^*, \mathbf{p}', \varepsilon)$-leakage tolerant against random gate probing attacks for $\mathcal{C}$ over $\mathbb{B}$, where $\mathbf{p}^* = \mathbf{p}^{\ell_{in}} \cdot (1 - (1 - \mathbf{p})^{\ell_{out}})$.*

Proof. The proof of this proposition follows closely along the lines of Proposition 9. As before, we define the following hybrid distribution.

Sampler $\mathcal{D}^w_{\mathbf{p}}(\widehat{C}, \widehat{x})$: Denote the set of wires in $\widehat{C}$ as $\mathcal{W}$[16]. 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 set S as follows: initially S is assigned to be $\{\}$. For every $w \in \mathcal{W}$, with probability $\mathbf{p}$, include $(w, \mathbf{val}(w))$ in S (i.e., with probability $(1 - \mathbf{p})$ the pair $(w, \mathbf{val}(w))$ is not included). Construct the set of leaked wire values $\mathcal{S}_{\mathsf{leak}}$ as follows: for every gate $G \in C$ with input wires $w^{inp}_1, \ldots, w^{inp}_{\ell_{in}}$ and one of the ℓ_{out} output wires w^{out},

$$\text{include } (w^{inp}_1, b^{inp}_1), \ldots, (w^{inp}_{\ell_{in}}, b^{inp}_{\ell_{in}}), (w^{out}, b^{out}) \text{ in } \mathcal{S}_{\mathsf{leak}}$$

$$\Leftrightarrow (w^{inp}_1, b^{inp}_1), \ldots, (w^{inp}_{\ell_{in}}, b^{inp}_{\ell_{in}}), (w^{out}, b^{out}) \in S$$

Furthermore, if there exists wire w' such that w' carries the same value as w (for instance, w' and w are the output wires of the same gate) and if $(w, v_w) \in \mathcal{S}_{\mathsf{leak}}$, then also include (w', v_w) in $\mathcal{S}_{\mathsf{leak}}$.

Output $\mathcal{S}_{\mathsf{leak}}$.

It immediately follows that the distributions $\mathcal{D}^w_{\mathbf{p}}$ and $\mathsf{RPDistr}^g_{\mathbf{p}^*}$ (same as defined in the proof of the Proposition 9) are identical: the probability $\mathbf{p}^*$ that any given gate G is leaked is the same as the probability that both its input wires and one of its output wires are leaked. Since, every wire is leaked independently, we have

$$\begin{aligned}
\mathbf{p}^* &= \mathsf{Pr}\left[\ell_{in} \text{ input wires of } G \text{ are leaked } \wedge \text{ one of } \ell_{out} \text{ output wires of } G \text{ is leaked}\right] \\
&= \mathsf{Pr}\left[\ell_{in} \text{ input wires of } G \text{ are leaked}\right] \cdot \mathsf{Pr}\left[\text{all the output wires of } G \text{ are not leaked}\right] \\
&= \mathbf{p}^{\ell_{in}} \cdot (1 - \mathsf{Pr}\left[\text{all the output wires of } G \text{ are not leaked}\right]) \\
&= \mathbf{p}^{\ell_{in}} \cdot (1 - (1 - \mathbf{p})^{\ell_{out}})
\end{aligned}$$

It remains to show that CC is secure with respect to the distribution $\mathcal{D}^w_{\mathbf{p}}$ of wire probing attacks. This part of the argument proceeds along the same lines as in the proof of Proposition 9. $\square$

Proposition 11. *For any basis $\mathbb{B}$, any constant ε, there does not exist any circuit compiler that is $(\mathbf{p}, \varepsilon)$-leakage tolerant against random gate probing attacks over basis $\mathbb{B}$, where $\mathbf{p} \geq \frac{1}{2}$.*

Proof. Suppose the proposition statement is true, then the following holds: there exists a circuit compiler CC for a circuit C (defined below) that is $(\mathbf{p}, \varepsilon)$-leakage tolerant against random gate probing attacks with $\mathbf{p}$and ε as defined in the proposition statement. Using this, we construct an information theoretically secure two party computation protocol Π for two-party functionality $\mathcal{F}$ (which will correspond to the function computed by C). By choosing F appropriately, we arrive at a contradiction by invoking the impossibility result of information theoretically secure two party computation protocol for F by Chor and Kushilevitz [CK91].

[16]Suppose a gate has two output wires, then including one of the output wires in $\mathcal{W}$ means including also the other one.

We define the two-party functionality $\mathcal{F}$ and the protocol Π for $\mathcal{F}$ next. To do that, first consider the following: let $\widehat{C} \leftarrow \mathsf{Compile}(C)$. Since $\mathsf{Compile}$ is deterministic, $\widehat{C}$ is uniquely defined given C. Let $\mathcal{G}$ be the set of gates in $\widehat{C}$. Construct $\mathcal{G}'$ by including in $\mathcal{G}'$ every gate $G \in \mathcal{G}$ with probability $\mathbf{p}$. Define $\mathsf{Inp}(G)$ to be the set of input wires of gate G.

Define $I \subseteq [n]$ as consisting of all indices $i \in [n]$ such that there exists at least one wire $w \in \mathsf{Inp}(G')$ for some $G \in \mathcal{G}'$ and also w carries the i^{th} input bit.

Defining $\mathcal{F}$. The two-party functionality $\mathcal{F}$ computes the same function as that represented by C. The joint input length of $\mathcal{F}$ is the same as the input length of C. In more detail, $\mathcal{F}(y_1, y_2) = C(x)$, where $y_1||y_2$ is a permutation of bits of x. This permutation is specified by the index set I. Let $I = \{i_1, \dots, i_L\}$ and let $\overline{I} = \{j_1, \dots, j_{n-L}\}$. Define $y_1 = x_{i_1}|| \cdots ||x_{i_L}$ and $y_2 = x_{j_1}|| \cdots ||x_{j_{n-L}}$.

Construction of Π. We now construct a two party computation protocol Π for $\mathcal{F}$. Then we reduce the security of Π to the security of CC.

Denote the two parties in Π to be P_1 and P_2. That is, they compute $\mathcal{F}(y_1, y_2)$, where x_i is the input of party P_i. The main idea behind the construction is to divide $\widehat{C}$ (encoding of C w.r.t CC) into two circuits that compute P_1 and P_2.

To do this we define the following partition function, $\mathsf{Partition}(\widehat{C}, \mathcal{G}')$. It takes as input $\widehat{C}$, subset of gates $\mathcal{G}'$ and outputs the description of the protocol $\Pi = (P_1, P_2)$. For every gate $G \in \mathcal{G}'$, assign G to P_1 and for every gate $G \notin \mathcal{G}'$, assign it to P_2. Since $\widehat{C}$ is a graph, this performs a partition of the vertices of G. Observe that if $G, G' \in \mathcal{G}'$ and if the output wire of G is fed into G' then this wire remains inside the circuit computing P_1. If there is $G \in \mathcal{G}', G' \notin \mathcal{G}'$ and if the output wire of G is fed into G' then this wire connects P_1 and P_2.

It can be seen that the correctness of CC implies the correctness of Π. We prove the security below.

Lemma 18. *The $(\mathbf{p}, \varepsilon)$-leakage tolerance of* CC *against random gate probing attacks implies that* Π *satisfies ε-statistical security against semi-honest adversaries.*

Proof. We introduce some notation. Consider two sets A and B. Consider a set $S \subseteq A \times B$. We define $\mathsf{Marg}(S) = \{a \ : \ \exists \, b \in B, \ (a, b) \in S\}$. Consider a circuit C and let $\mathcal{G}$ be the set of gates in C. We write this as $\mathcal{G} \subseteq C$.

We prove the following claim.

Claim 14. *Consider a circuit $C \in \mathcal{C}$ and an input x. Let $\widehat{C} \leftarrow$* $\mathsf{Compile}(C)$ *and let $\mathcal{G}^*$ be any subset of the gates in $\widehat{C}$. Let* Sim_{LT} *be the PPT simulator associated with the leakage tolerant circuit compiler* CC. *We have,*

$$\sum_{\mathcal{S}_{\mathsf{leak}}:\mathsf{Marg}(\mathcal{S}_{\mathsf{leak}})=\mathcal{G}^*} \left| \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{RPDistr}^g_{\mathbf{p}}(\widehat{C}, \widehat{x})\right] - \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{Sim}_{LT}(\widehat{C})\right] \right| \leq \varepsilon$$

Proof. From the $(\mathbf{p}, \varepsilon)$-leakage tolerance of CC, we have the following:

$$\begin{aligned}
\sum_{\mathcal{S}_{\mathsf{leak}}} \left| \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{RPDistr}^g_{\mathbf{p}}(\widehat{C}, \widehat{x})\right] - \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{Sim}_{LT}(\widehat{C})\right] \right| &\leq \varepsilon \\
\sum_{\mathcal{G}' \subseteq \widehat{C}} \left(\sum_{\mathcal{S}_{\mathsf{leak}}:\mathsf{Marg}(\mathcal{S}_{\mathsf{leak}})=\mathcal{G}'} \left| \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{RPDistr}^g_{\mathbf{p}}(\widehat{C}, \widehat{x})\right] - \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{Sim}_{LT}(\widehat{C})\right] \right| \right) &\leq \varepsilon
\end{aligned}$$

Thus, for any $\mathcal{G}' \subseteq \widehat{C}$, it holds that,

$$\sum_{\mathcal{S}_{\mathsf{leak}}:\mathsf{Marg}(\mathcal{S}_{\mathsf{leak}})=\mathcal{G}'} \left| \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{RPDistr}^g_{\mathbf{p}}(\widehat{C}, \widehat{x})\right] - \mathsf{Pr}\left[\mathcal{S}_{\mathsf{leak}} \leftarrow \mathsf{Sim}_{LT}(\widehat{C})\right] \right| \leq \varepsilon$$

This proves the claim. □

Consider a circuit $C \in \mathcal{C}$. Let $\widehat{C} \leftarrow \mathsf{Compile}(C)$ and let $\mathcal{G}$ be the set of gates in $\widehat{C}$. Construct $\mathcal{G}'$ by including every gate $G \in \mathcal{G}$ in $\mathcal{G}'$ with probability $\mathbf{p}$. The protocol $\Pi = (P_1, P_2)$ and two-party functionality F is as computed by $\mathsf{Partition}(\widehat{C}, \mathcal{G}')$. Define the following classes of simulators:

- $\mathcal{SIM}_A^{\widehat{C},\mathcal{G}'}$: it consists of all PPT simulators Sim such that $\mathcal{G}' \leftarrow \mathsf{Marg}(\mathsf{Sim}(\widehat{C}))$. That is, the marginal distribution of the output of $\mathsf{Sim}(\widehat{C})$ is always $\mathcal{G}'$.
- $\mathcal{SIM}_B^{\widehat{C},\mathcal{G}'}$: it consists of all PPT simulators Sim such that $(\mathcal{G}\backslash\mathcal{G}') \leftarrow \mathsf{Marg}(\mathsf{Sim}(\widehat{C}))$. That is, the marginal distribution of the output of $\mathsf{Sim}(\widehat{C})$ is always $\mathcal{G}\backslash\mathcal{G}'$.

Consider the following claims.

Claim 15. *Consider a circuit $C \in \mathcal{C}$. Suppose $\widehat{C} \leftarrow \mathsf{CC}$ and let $\mathcal{G}' \subseteq \widehat{C}$. Let F be a two-party functionality as computed above. Let Π be a two-party computation protocol for F constructed from C and CC. Let (x_1, x_2) be a pair of inputs in the input domain of F. Then the following holds:*

- *Let $\mathsf{Sim} \in \mathcal{SIM}_A^{\widehat{C},\mathcal{G}'}$.*
$$\mathsf{Sim}(F(x_1,x_2),x_1) \approx_\varepsilon \mathsf{Real}_{F,\{1\}}(x_1,x_2),$$
- *Let $\mathsf{Sim} \in \mathcal{SIM}_B^{\widehat{C},\mathcal{G}'}$.*
$$\mathsf{Sim}(F(x_1,x_2),x_1) \approx_\varepsilon \mathsf{Real}_{F,\{2\}}(x_1,x_2),$$

where $\mathsf{Real}_{F,\{1\}}$ is as defined in Definition 1.

The proof of the above claims follows from Claim 14. Moreover the above two claim prove the lemma.

$\square$

$\square$

We now state the main negative result.

Proposition 12. *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 $\mathbb{B}$.*

The proof of the above proposition follows from Propositions 10 and Proposition 11. In particular, for any basis mapping ℓ_{in} bits to ℓ_{out} bits, we can choose the appropriate $\mathbf{p}$ such that $(\mathbf{p})^{\ell_{in}} \cdot (1-(1-\mathbf{p})^{\ell_{out}}) = \frac{1}{2}$. For this choosing of $\mathbf{p}$, the above theorem is satisfied.

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 9. *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 4.

Theorem 10. *Consider any constant $0 < \mathbf{p} < 1$ and let $\mathbb{B}$ be a basis. For some constant $1 > \delta > 0$, there is a construction of $(\mathbf{p}, \exp(-s))$-leakage resilient circuit compiler over $\mathbb{B}'$ for all circuits over $\mathbb{B}$ of size s, secure against random probing attacks, where $\mathbb{B}'$ consists of all functions mapping $2\min(\lceil\frac{\log(\delta)}{\log(\mathbf{p})}\rceil, 2)$ bits to $2\min(\lceil\frac{\log(\delta)}{\log(\mathbf{p})}\rceil, 2)$ bits.*

The proof of the above theorem follows from Proposition 5.

7 Randomness Encoders

We show that we can construct leakage resilient circuit compilers with rate $\mathbf{p}$, where $\mathbf{p}$ tends to 1. To achieve this, we relax the definition of circuit compilers and allow a randomness encoder that produces freshly computed correlated distribution for every input encoding. We present the definition below.

Definition 14 (Randomness Encoder). *A circuit compiler* $\mathsf{CC} = (\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ *is said to be a circuit compiler with randomness encoder if it has an additional PPT algorithm:*

- $\mathsf{REncoder}(1^n)$*: On input* 1^n*, it produces a correlated distribution* μ*.*

such that the following holds: for every circuit C*, input* x*,*

$$\mathsf{Decode}\left(\mathsf{Compile}(C), \mathsf{Encode}(x), \mathsf{REncoder}(1^{|C|})\right) = C(x)$$

Remark 4. *We remark that we don't place any requirement on the size of the output produced by the randomness encoder. In fact, the size of the correlated distribution produced by the randomness encoder could be as large as the size of the circuit being compiled.*

We prove the following proposition.

Proposition 13. *For any constant* $0 < \mathbf{p} < 1$*, there is a construction of* $(\mathbf{p}, \varepsilon)$*-secure leakage resilient circuit compiler, where* ε *is negligible in the circuit size.*

Proof Sketch. Consider a constant $0 < \mathbf{p} < 1$.
To compile a circuit C of size s, we proceed in the following steps.

1. $(\mathbf{p}, \varepsilon)$**-secure LRCC for AND with rand. encoder, for some constant** $0 < \varepsilon < 1$**.** We start with the following MPC protocol for AND by Beaver [Bea91] in the correlated randomness model.

- Inputs: Additive shares $[a] = ([a]_1, \ldots, [a]_m)$ and $[b] = ([b]_1, \ldots, [b]_m)$ of secrets $a, b \in \mathbb{F}_2$.
- Outputs: Additive shares $[c] = ([c]_1, \ldots, [c]_m)$ of $c = ab$.
- Correlated randomness: Random additive shares $[a'], [b']$ of random and independent secrets $a', b' \in \mathbb{F}_2$, and random additive shares $[c']$ of $c' = a'b'$.
- Communication: Party i locally computes $[\Delta a]_i = [a]_i - [a']_i$ and $[\Delta b]_i = [b]_i - [b']_i$ and sends $[\Delta a]_i$ and $[\Delta b]_i$ to all other parties.
- Computing output: Party i computes $\Delta a = \sum_{j=1}^{m} [\Delta a]_j$ and
  $\Delta b = \sum_{j=1}^{m} [\Delta b]_j$, and outputs $[c]_i = \Delta b [a]_i + \Delta a [b]_i + [c']_i - \Delta a \Delta b$

We claim that the circuit representing the above protocol is a leakage resilient circuit compiler secure against $(\mathbf{p}, \varepsilon)$-random probing attacks.

2. $(\mathbf{p}, \varepsilon)$**-secure LRCC for AND with rand. encoder, where** $\varepsilon = \exp(-s)$**.** This follows by repeatedly composing the AND gadget with itself, along the same lines as done in the previous sections. In particular, the composition step works even on circuit compilers augmented with randomness encoder.

3. $(\mathbf{p}, s \cdot \varepsilon)$**-secure LRCC for** C **with rand. encoder, where** $\varepsilon = \exp(-s)$**.** Note that we can similarly obtain a $(\mathbf{p}, \varepsilon)$-secure LRCC for XOR with rand. encoder, where $\varepsilon = \exp(-s)$. We can then stitch the gadgets for all the AND and XOR gates in C to obtain the leakage resilient circuit compiler for C. If the simulation error in each gadget is at most ε then the error incurred in simulating the whole compiled circuit is at most $s \cdot \varepsilon$.

□

Acknowledgements. We thank Jean-Sébastien Coron, Stefan Dziembowski, and Sebastian Faust for helpful discussions. Special thanks to Jean-Sébastien Coron for pointing out an error in our result on the randomness complexity of private circuits; we have retracted this result from the full version.

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 was 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 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] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with O(1/\log (n)) leakage rate. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques II, Vienna, Austria, May 8-12, 2016*, pages 586–615, 2016.

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In *Annual International Cryptology Conference*, pages 427–455. Springer, 2018.

[Ajt11] Miklós Ajtai. Secure computation with information leaking to an adversary. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 715–724. ACM, 2011.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129. ACM, 2016.

[BBP$^+$16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 616–648. Springer, 2016.

[BBP$^+$17] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In *Annual International Cryptology Conference*, pages 397–426. Springer, 2017.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

[CDI$^+$13] Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D Rothblum. Efficient multiparty protocols via log-depth threshold formulae. In *Advances in Cryptology–CRYPTO 2013*, pages 185–202. Springer, 2013.

[CK91] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM Journal on Discrete Mathematics*, 4(1):36–47, 1991.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 423–440, 2014.

[GIK$^+$15] Sanjam Garg, Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with one-way communication. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 191–208, 2015.

[GIM$^+$16] Vipul Goyal, Yuval Ishai, Hemanta K Maji, Amit Sahai, and Alexander A Sherstov. Bounded-communication leakage resilience via parity-resilient circuits. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 1–10. IEEE, 2016.

[HM00] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of cryptology*, 13(1):31–60, 2000.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[Mau02] Ueli Maurer. Secure multi-party computation made simple. In *International Conference on Security in Communication Networks*, pages 14–28. Springer, 2002.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge university press, 2005.

[Pip85] Nicholas Pippenger. On networks of noisy gates. In *FOCS*, pages 30–38, 1985.

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