Protecting Computations Against Continuous Bounded-Communication Leakage

Yuval Ishai
Technion, Israel
Amazon Web Services, USA*
yuvali@cs.technion.ac.il

Yifan Song
Tsinghua University, China
Shanghai Qi Zhi Institute, China
yfsong@mail.tsinghua.edu.cn

Abstract

We consider the question of protecting a general computation device, modeled by a stateful Boolean circuit, against leakage of partial information about its internal wires. Goyal et al. (FOCS 2016) obtained a solution for the case of *bounded-communication leakage*, where the wires are partitioned into two parts and the leakage can be any function computed using t bits of communication between the parts. However, this solution suffers from two major limitations: (1) it only applies to a *one-shot* (stateless) computation, mapping an encoded input to an encoded output, and (2) the leakage-resilient circuit consumes fresh random bits, whose number scales linearly with the circuit complexity of the computed function.

In this work, we eliminate the first limitation and make progress on the second. Concretely:

- We present the first construction of *stateful* circuits that offer information-theoretic protection against *continuous* bounded-communication leakage. As an application, we extend a two-party "malware-resilient" protocol of Goyal et al. to the continuous-leakage case.
- For simple types of bounded-communication leakage, which leak t parities or t disjunctions of circuit wires or their negations, we obtain a *deterministic* variant that does not require any fresh randomness beyond the randomness in the initial state. Here we get computational security based on a subexponentially secure one-way function. This is the first deterministic leakage-resilient circuit construction for any nontrivial class of *global* leakage.

*This paper describes work performed at the Technion and is not associated with Amazon.

Contents

1 Introduction

Protecting secrets against partial information leakage is a central theme in cryptography. In the context of securely storing and communicating secrets, this has led to a rich body of work on *secret sharing* [Sha79] and *privacy amplification* [BBCM95] that protect secrets against local and global leakage, respectively. A more general goal is protecting secrets in the context of *computation,* where even intermediate results may be subject to leakage. This question too has been the subject of a large body of work on *secure multiparty computation* [Yao86, GMW87, BGW88, CCD88] and *leakage-resilient circuits* [ISW03, FRR+14] that protect general computations against local and global leakage, respectively.

From the four combinations of "communications vs. computation" and "local vs. global leakage," three are well understood in the sense that the basic feasibility questions have been long settled. The most challenging remaining frontier is protecting general *computations* against *global* leakage, which may apply jointly to the entire computation transcript. This type of leakage may arise in realistic side-channels attacks such as timing or power analysis that measure global information about the computation.

A clean theoretical framework for capturing this problem is given by the notion of a *leakage-resilient circuit* (LRC) [ISW03, FRR+14]. We consider a class $\mathcal{L}$ of allowable leakage functions, where each $L \in \mathcal{L}$ takes the values of all wires in a Boolean circuit as an input, and outputs t bits of partial information about these wires. LRCs come in two flavors: a simpler *stateless* LRC, which captures the case of a single computation with a "one-shot" leakage, and a *stateful* LRC that aims to capture a general computation device with updatable memory that is subject to *continuous* leakage.

In more detail, a *stateless* LRC for a function $f : \{0,1\}^n \to \{0,1\}^m$ is a randomized Boolean circuit $C : \{0,1\}^{\hat{n}} \to \{0,1\}^{\hat{m}}$, mapping an encoded input $\hat{\boldsymbol{x}}$ to an encoded output $\hat{\boldsymbol{y}}$, together with a trusted (leak-free) randomized input encoder $I : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ and output decoder $O : \{0,1\}^{\hat{m}} \to \{0,1\}^m$. The correctness requirement is that for any input $\boldsymbol{x}$, we have $O(C(I(\boldsymbol{x}))) = f(\boldsymbol{x})$ (with probability 1). The security requirement is that applying any leakage function $L \in \mathcal{L}$ to the wires of C reveals essentially nothing about $\boldsymbol{x}$. To rule out trivial solutions where f is computed by I or O, we require that I and O be universal, and only C can depend on f; we will in fact aim to construct efficient *circuit compilers* that convert a circuit for f into C.

Stateful Leakage-Resilient Circuits. An inherent disadvantage of stateless LRCs is the need to rely on ideal implementations of I and O. The *stateful* LRC model reduces this trust to a one-time setup, extends the functionality to capture a stateful computation, and extends the one-shot leakage to a continuous leakage. Concretely, the ideal functionality $f[\boldsymbol{s}]$ is a reactive functionality that starts with a secret initial state $\boldsymbol{s} = \boldsymbol{s}_0$. In each invocation, there is a public input $\boldsymbol{x}_i$ and a public output $\boldsymbol{y}_i$, and the state may be updated. For example, consider a stateful AES circuit in which the internal state is the AES secret key, and in each invocation the circuit takes a plaintext $\boldsymbol{x}_i$ as input and outputs the corresponding ciphertext $\boldsymbol{y}_i$. In this example, the state remains the same. An LRC for f consists of a randomized stateful circuit $C[\hat{\boldsymbol{s}}]$, along with a randomized mapping from an initial secret state $\boldsymbol{s}_0$ of f to the initial state $\hat{\boldsymbol{s}}_0$ of C. The correctness requirement is that C, when initialized with $\hat{\boldsymbol{s}}_0$, must have the same reactive functionality as f when initialized with $\boldsymbol{s}_0$. For security, we require that the view of any adversary, who interacts with C by adaptively choosing an input $\boldsymbol{x}_i$ and leakage query $L_i \in \mathcal{L}$ for each invocation, can be efficiently simulated given the public input $\boldsymbol{x}_i$ and public output $\boldsymbol{y}_i$ of f alone, without knowing $\boldsymbol{s}_0$.

Originating from the work of Ishai, Sahai, and Wagner [ISW03], there has been a long line of theoretical and applied works on efficient constructions of stateful LRCs against local "probing" leakage, where each L_i outputs the values of t wires in C. The study of stateful LRCs against *global* leakage was initiated[1] by Faust, Rabin, Reyzin, Tromer, and Vaikuntanathan [FRR+14], who constructed stateful LRCs with information-theoretic security against $\mathcal{AC}0$ leakage by relying on leak-free components whose size depends only on a statistical security parameter. Rothblum [Rot12] eliminated the trusted components at the expense of relying on a complexity theoretic conjecture, and Bogdanov, Ishai, and Srinivasan [BIS21] finally settled this feasibility question by obtaining an unconditional variant. Beyond $\mathcal{AC}0$, Goldwasser and Rothblum [GR15] established the feasibility of stateful LRCs for a variant of *"only computation leaks"* (OCL) leakage [MR04], where t bits can leak from each sequential sub-computation. See Appendix A for further discussion of this model.

Bounded-Communication Leakage. In this work, we are interested in the case of *bounded-communication leakage* (BCL) [GIM+16, GIW17]. A BCL class $\mathcal{L}$ is parameterized by a communication bound t and a partition of the wires of C into two parts. (The choice of the partition often does not matter, as some applications just rely on the *existence* of such a partition.) The induced BCL class $\mathcal{L}$ includes every function L that can be computed by an interactive two-party protocol between the two parts, where the protocol involves t bits of per-party communication. Note that this class strictly contains the class of t-parities, and can capture several realistic types of leakage. In particular, BCL with an arbitrary partition can compute (weighted) sums over the integers of any number of wire values. This captures toy versions of attacks such as power analysis, timing attacks, or electromagnetic probes, which can be approximated by linear combinations of wire values [GIW17]. Finally, as we discuss below, the BCL model tightly captures an application from [GIM+16] to protecting two-party computations against malware. The related OCL model does not suffice for this purpose (see Appendix A).

The study of leakage-resilient computation against BCL was initiated by Goyal, Ishai, Maji, Sahai, and Sherstov [GIM+16], who considered a two-party model and assumed either ideal oblivious transfer or standard cryptographic assumptions. It was later observed by Genkin, Ishai, and Weiss [GIW17] that essentially the same construction can be cast in the standard LRC model.

Unfortunately, the LRC constructions from [GIM+16, GIW17] suffer from two major limitations. First, and most importantly, they are limited to the easier *stateless* case, leaving a stateful variant as an explicit open problem. Second, the randomness complexity of these constructions scales linearly with the circuit complexity of f. High randomness complexity is undesirable, since randomness is an expensive and fragile resource, especially for embedded devices. This motivated a line of works, initiated by [IKL+13], on LRCs with sublinear randomness. However, all of these works are restricted to the case of *probing* leakage. The main source of difficulty in going beyond probing leakage is that standard pseudorandom generators do not suffice, since the randomness generation itself is subject to leakage.

1.1 Our Contribution

In this work, we eliminate the first of the above two limitations and make progress on the second. Concretely, we obtain the following results.

[1] The work of Micali and Reyzin [MR04] studied a similar problem, but in a different and more abstract setting.

Stateful LRCs for General Bounded-Communication Leakage. We present the first construction of *stateful* BCL-resilient circuits (with security in the presence of continuous leakage), settling an open question from [GIM+16, GIW17]. Among other tools, this result combines a recent construction of stateless *parity-tolerant* circuits [IS24] with a communication complexity lifting lemma from [GIM+16].

Deterministic LRCs for Special Bounded-Communication Leakage. For simple types of bounded-communication leakage, such as leaking t parities or t disjunctions/conjunctions of circuit wires or their negations, we obtain a *deterministic* stateful LRC that does not require any fresh randomness beyond the randomness embedded in the encoded initial state. Here we need to settle for computational security, assuming the existence of a (subexponentially secure) one-way function. This is the first construction of a deterministic stateful LRC for any nontrivial class of *global* leakage. Previously, deterministic constructions were only known for (local) probing leakage [ISW03], building on a distributed pseudorandomness generation technique from [CH94]. Our deterministic LRC construction is based on a general technique that combines a *partially derandomized* stateless LRC with a cryptographic pseudorandom generator. The technique is limited to leakage classes with a strong "decomposability" property. We then show how to partially derandomize stateless LRCs from the literature.

Application: Protecting Computation Against Malware. Our results for general BCL are motivated by the following application. It is typically the case that once a computer is infected by malware and is connected with the outside world, cryptography is of no help. Indeed, if the malware has full access to secrets, it can reveal sensitive information even with just one bit of communication. A possible way around this is to allow the secrets to be *distributed* and assume the malware has a limited *communication* budget. The latter assumption, which underlies cryptography in the "bounded-retrieval model" [Dzi06, CLW06], may capture situations where the malware does not alter the legitimate messages sent by the infected computer, in the fear of being detected; instead, the malware can resort to communicating at a very slow rate, e.g., via subtle message delays. Initial work in this direction focused on basic primitives such as secret sharing or encryption (see [DP07, BKKV10, ADN+10] and references therein). Can cryptography help protect *general computations* against malware in this bounded-communication setting?

The work of Goyal et al. [GIM+16] suggested the following approach. Suppose we have two computers, which may be *both* infected by malware, but where each instance of the malware has a budget of t bits of communication with the outside world. The idea of [GIM+16] is to combine an information-theoretic BCL-resilient LRC with a strong flavor of adaptively secure oblivious transfer (which can be based on a variety of standard cryptographic assumptions), obtaining a two-party protocol with the following guarantees. Starting with an input $\boldsymbol{x}$ which is split between the two computers using a BCL-resilient secret-sharing scheme, the protocol outputs a BCL-resilient secret sharing of $\boldsymbol{y} = f(\boldsymbol{x})$. The security guarantee is that even if *both* computers are corrupted by t-BCL bounded malware, the input $\boldsymbol{x}$ remains computationally hidden.

The main limitation of the solution from [GIM+16] is that it only applies to the case of a one-shot computation, and relies on trusted (leak-free) encoding of the input $\boldsymbol{x}$ and decoding of the output $\boldsymbol{y}$. These are assumed to be done by an external, uncorrupted party. Extending the solution to the case of continuous leakage, or even just eliminating the need for trusted encoding and decoding in the one-shot case, was left as an open problem. Our stateful BCL-resilient construction implies a

solution to this problem under standard cryptographic assumptions (hardness of factoring or DDH).

Note that unlike the LRC application, where BCL captures a wide and natural but not necessarily the "ultimate" class of attacks, here BCL tightly matches the problem requirements and enables solutions with a *minimal number of parties*. In particular, the wire partition is induced by the interaction between the two parties.[2] Another difference is that in the context of the current application, the information leaked by BCL is not sensitive at all to the computational model; indeed, the leakage returns a t-bounded-communication function of two views, irrespectively of how the views were computed or how they are represented. We view the application to two-party BCL-resilient computation as a central motivation behind this work. See Section 4.4 for a formal treatment.

2 Technical Overview

In this section we give an overview of our techniques, starting with some relevant background.

2.1 Background

Stateful and Stateless LRCs. Recall that a stateful leakage-resilient circuit (LRC) aims to protect a general-purpose computing device, which can be invoked any number of times and possibly update its memory in each invocation, against a prescribe class of leakage functions. For an ideal stateful circuit $C[\boldsymbol{s}]$, an LRC consists of a randomized stateful circuit $C'[\boldsymbol{s}']$ along with a randomized mapping from an initial secret state $\boldsymbol{s}_0$ of C to the initial state $\boldsymbol{s}'_0$ of C', satisfying the following correctness and security requirements: (1) $C'[\boldsymbol{s}'_0]$ has the same (reactive) functionality as $C[\boldsymbol{s}_0]$, and (2) any adversary who interacts with the stateful circuit $C'[\boldsymbol{s}'_0]$, adaptively choosing inputs $\boldsymbol{x}_i$ and leakage queries $L_i \in \mathcal{L}$, can be efficiently simulated given only the public inputs $\boldsymbol{x}_i$ and outputs $\boldsymbol{y}_i$ alone, without knowing the initial secret state $\boldsymbol{s}_0$.

The simpler notion of a *stateless* LRC considers a one-shot computation of a function f on a secret input $\boldsymbol{x}$. Concretely, a stateless LRC is defined by a triple (I, C, O), where I is a randomized input encoder, C is a randomized circuit mapping the encoded input to an encoded output, and O is an output decoder. Here we assume that the (single) leakage function $L \in \mathcal{L}$ applies only to the wires in C but not to the internal computation of I or O, and require that the output of L can be simulated without knowing the private input.

Starting from a stateless LRC, a natural attempt for constructing a stateful LRC is to run a stateless LRC in each invocation, taking an encoded state and a public input and outputting an encoded updated state and a public output. However, such a construction does not work in general. A simple counterexample is the case where we just want to keep the secret state alive, namely we apply the identity function to the secret state. A trivial stateless LRC for the identity function can simply output the encoded input state. But this does not yield a stateful LRC since the same encoding is under attack in each invocation of the stateful circuit. For example, an attacker may choose to learn the i-th bit in the i-th invocation. After a sufficient number of invocations, the entire encoding of the state is compromised. The problem is that stateless LRCs may not separate the randomness used in the input encoding from the randomness used in the output encoding. In the above example, we use the same randomness for both encodings. Partially motivated by practical constructions, a variety of composable notions of LRCs that achieve such a separation in the case of

[2] Unlike the BCL case, where the wires are perfectly partitioned, here we must have an overlap between the views of the two parties; this is the reason for requiring the use of oblivious transfer.

probing leakage have been proposed; see, e.g., [RP10, CPRR13, BBD+16, BCRT23] and references therein. However, it is not clear a-priori how to extend these approaches even to very simple types of global leakage without relying on trusted leak-free components.

Leakage-Tolerant Circuits. Motivated by this gap, a recent work of Ishai and Song [IS24] initiated the study of *leakage-tolerant* circuits (LTCs). An LTC is a stateless randomized circuit C that directly maps an unprotected input $\boldsymbol{x}$ to an unprotected output $\boldsymbol{y} = f(\boldsymbol{x})$ while providing the following best-possible security guarantee: the output of any leakage function $L \in \mathcal{L}$ on the internal wires of C can be *simulated* (up to a negligible statistical distance) by applying a similar leakage $L' \in \mathcal{L}$ to $(\boldsymbol{x}, \boldsymbol{y})$. Thus, C is "as secure" as an ideal (leak-free) implementation of f. Note that an LTC against $\mathcal{L}$ can be readily converted into an LRC against $\mathcal{L}$ by using an arbitrary $\mathcal{L}$-resilient encoding to protect the input $\boldsymbol{x}$ and the output $\boldsymbol{y}$. More interestingly, the results of [IS24] also show that an LTC can be used for realizing a *stateful* LRC for the same leakage class.

However, building LTCs appears to be much more difficult than building LRCs. Before [IS24], the only leakage class for which LTCs were known to exist was the probing class, which outputs the values of t wires [IKL+13, AIS18]. The main contributions of [IS24] are constructions of LTCs against depth-1 $\mathcal{AC}0$ leakage and parity leakage. In contrast, there were known solutions for stateless LRCs against much stronger leakage classes, including bounded-communication leakage [GIM+16].

2.2 Stateful LRCs for Bounded-Communication Leakage

We start by formally defining the notion of t-bounded-communication leakage.

Definition 1 (t-BCL [GIM+16, GIW17])**.** *Let $t \in \mathbb{N}$ be a leakage bound parameter. We say that a deterministic 2-party protocol is t-bounded if its communication complexity per party is at most t bits (so the total communication is bounded by $2t$ bits). Given a partitioned input $x = (x_1, x_2)$, the t-bounded-communication leakage (t-BCL) class $\mathcal{L}^t_{\mathtt{BCL}}$ contains all leakage function $L : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^{2t}$ such that $L(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is the transcript of a t-bounded protocol $\Pi(\boldsymbol{x}_1, \boldsymbol{x}_2)$.*

When we refer to t-BCL-resilience of an implementation, we mean that there is *some* partition of the wires for which the implementation is resilient to the t-BCL class induced by this partition. More concretely, we say a tuple (I, C, O) is a t-BCL-resilient implementation of a function f if there is a partition of the wires in C, denoted by $(\tau_1(C), \tau_2(C))$, such that any t-BCL leakage function L applied to $(\tau_1(C), \tau_2(C))$ reveals essentially nothing about the input and output of f. We similarly define t-BCL resilience for stateful circuits.

Challenges in Building Stateful LRCs for BCL. Goyal et al. [GIM+16] constructed *stateless* LRCs for BCL, and left the stateful case as an open question. As argued above, the direct approach of using a stateless LRC to emulate each invocation of a stateful LRC does not work. On the other hand, while [IS24] gives a general compiler for stateful LRCs from leakage *tolerant* circuits, constructing LTCs for BCL remains a challenging open problem.

Before explaining our alternative strategy, we first review the techniques underlying the stateless LRC for BCL from [GIM+16]. For a function f, the first step is to construct a *stateless* LRC $(\hat{I}, \hat{C}, \hat{O})$ for *parity* leakage. Then, parity resilience is lifted to BCL resilience using the following communication complexity lemma, which generalizes the "small-bias masking lemma" from [DS05]: If two distributions X, Y over $\{0,1\}^n$ are indistinguishable by parities, then $(R, X \oplus R)$ and $(R, Y \oplus$

R) are indistinguishable by t-BCL, where R is uniformly distributed. (More precisely, ϵ-parity resilience implies $\epsilon \cdot 2^t$ resilience to t-BCL.) Based on this lemma, an LRC C against BCL can be obtained by splitting the transcript of $\hat{C}$, denoted by $\tau(\hat{C})$, into two parts $(R, R \oplus \tau(\hat{C}))$ (viewed as 2-out-of-2 shares of the transcript), where R is uniformly distributed. The splitting is done using secure computation techniques that are mostly irrelevant to the current discussion.

Note that to compute $(R, R \oplus \tau(\hat{C}))$, for each gate in $\hat{C}$ with input wires w_i, w_j and output wire w_k, the circuit C needs to compute an additive sharing of w_k from additive sharings of w_i and w_j. Genkin et al. [GIW17] observe that this can be done by building a sub-circuit in which each inner wire is either a function of R or a function of $R \oplus \tau(\hat{C})$. Note that under BCL, having access to these inner wires does *not* help the adversary since the leakage protocol Π can compute these wires on its own. For simplicity, we just assume that the transcript of C is $(R, R \oplus \tau(\hat{C}))$.

Starting Point. Our initial idea is to use a *parity-tolerant* circuit (recently constructed in [IS24]) instead of a parity-resilient circuit. Let $C_{\mathtt{pr}}$ be a parity-tolerant implementation of f. We observe that parity tolerance implies the following useful property.

- Suppose there is a fixed value y and two random variables X, X' such that (1) $y = f(X) = f(X')$ and (2) X, X' are indistinguishable by parities. Then, $\tau(C_{\mathtt{pr}}, X)$ and $\tau(C_{\mathtt{pr}}, X')$ are indistinguishable by parities, where $\tau(C_{\mathtt{pr}}, X)$ refers to the transcript of $C_{\mathtt{pr}}$ with input X.

At a first glance the required condition may look odd, as X and X' are two random variables but we require $f(X)$ and $f(X')$ to be the same fixed value. This can be the case where the function f takes as input a parity-resilient encoding of m, decodes the message m, and computes some function f' on m. Then for any m, m' such that $y = f'(m) = f'(m')$, we can choose X as a random parity-resilient encoding of m and X' as a random parity-resilient encoding of m'.

Now, to apply the generalized small-bias masking lemma, we split the transcript of $C_{\mathtt{pr}}$ to $(R, R \oplus \tau(C_{\mathtt{pr}}))$ and denote the resulting circuit by $C_{\mathtt{BCL}}$. Note that $C_{\mathtt{BCL}}$ no longer achieves the same functionality as $C_{\mathtt{pr}}$ since the input and output of $C_{\mathtt{BCL}}$ are random 2-out-of-2 additive sharings of the input and output of $C_{\mathtt{pr}}$ respectively.

Let R_y be the sub-string of R that is used to mask the output y of $C_{\mathtt{pr}}$. Then, given $(R_y, R_y \oplus y)$, $(R, R \oplus \tau(C_{\mathtt{pr}}, X))$ and $(R, R \oplus \tau(C_{\mathtt{pr}}, X'))$ are indistinguishable by BCL. This is because we effectively apply the generalized small-bias masking lemma to the transcript of $C_{\mathtt{pr}}$ except its output. Note that although the obtained circuit $C_{\mathtt{BCL}}$ is not an LTC against BCL, it does provide some separation between the randomness used for the input and the randomness used for the output.

Letting $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$ be a parity-resilient encoding, our stateful LRC proceeds as follows.

1. The input encoder computes a parity-resilient encoding $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_0)$ of the initial state $\boldsymbol{s}_0$, and splits it into two parts via an additive sharing over $\mathbb{F}_2$ (namely, XOR-sharing). The output can be viewed as an encoding of $\boldsymbol{s}_0$ that is secure against BCL, denoted by $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_0)$.

2. In each invocation with encoded state $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s})$ and a public input $\boldsymbol{x}$, we maintain the invariant that $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s})$ is an additive sharing of a parity-resilient encoding of $\boldsymbol{s}$, denoted by $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$. Let f be a function that takes $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$ and $\boldsymbol{x}$ as inputs and computes the following.

   - f decodes $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$ and obtains $\boldsymbol{s}$.
   - f computes $C[\boldsymbol{s}](\boldsymbol{x})$ and obtains the updated state $\tilde{\boldsymbol{s}}$ and public output $\boldsymbol{y}$.
   - Finally, f outputs $(\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}), \boldsymbol{y})$.

Let $C_{\mathtt{pr}}$ be a parity-tolerant circuit implementation of f and $C_{\mathtt{BCL}}$ be the circuit that computes additive shares of the transcript of $C_{\mathtt{pr}}$. Then $C_{\mathtt{BCL}}$ takes additive shares of $(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}), \boldsymbol{x})$ as input and outputs additive shares of $(\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}), \boldsymbol{y})$.

3. Note that $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s})$ is already an additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$. To use $C_{\mathtt{BCL}}$ to carry out the computation in each invocation, we randomly sample $\boldsymbol{r}_0$ and compute $\boldsymbol{x} \oplus \boldsymbol{r}_0$. Then we use $C_{\mathtt{BCL}}$ with input $(\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}), (\boldsymbol{r}_0, \boldsymbol{x} \oplus \boldsymbol{r}_0))$ to compute additive shares of $(\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}), \boldsymbol{y})$. Here the additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}})$ are viewed as an output of $\mathtt{Enc}_{\mathtt{BCL}}(\tilde{\boldsymbol{s}})$. To obtain $\boldsymbol{y}$, we simply add up (XOR) the two additive shares of $\boldsymbol{y}$.

To prove the security of this construction, we consider a sequence of hybrids where in the i-th hybrid, we want to replace the i-th invocation by a fake execution of $C_{\mathtt{BCL}}$ without relying on its encoded input state $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_{i-1})$. Intuitively, this is done by fixing the output in the i-th invocation and replacing the encoded input state $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_{i-1})$ by a fake input state, say $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0})$. By our observation on $C_{\mathtt{BCL}}$, $\mathbf{Hybrid}_i$ and $\mathbf{Hybrid}_{i-1}$ are indistinguishable by BCL. However, simply replacing $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_{i-1})$ by $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0})$ would not work, since this changes the functionality.

Rebuilding f to Support Fake Execution. To support fake execution transitions, we apply the following high-level idea from [IS24] (which can be viewed as an information-theoretic variant of a technique from [FLS99]): Let f take an additional control bit b attached to the state as well as an encoded auxiliary input. If $b = 0$, then f faithfully emulates the target stateful circuit. Otherwise, f simply outputs the decoding of the auxiliary input. More concretely:

1. f takes as input $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$, $\mathtt{Enc}_{\mathtt{pr}}(b)$, $\boldsymbol{x}$, and an encoded auxiliary input $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$. Here $b \in \{0, 1\}$ is the control bit and $\boldsymbol{r}$ is a random string. We require that $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$ be uniform.

2. f decodes the encodings and obtains $\boldsymbol{s}, b, \boldsymbol{r}$. If b is 0, then f faithfully computes the output as described above. Otherwise, f outputs $\boldsymbol{r}$.

In the real invocation, b is always 0 and the auxiliary input does not affect the output. In the simulation, we switch b to 1 and program $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$ to be an encoding of the desired output. This way, the encoded input state $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$ does not affect the function output and it is safe to replace $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$ by $\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0})$. As a result, the fake execution preserves the same functionality as the real execution but no longer requires the input state $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s})$.

Our construction is obtained by using this function f in our previous attempt. Now the input of $C_{\mathtt{BCL}}$ in our construction becomes additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i-1}), \mathtt{Enc}_{\mathtt{pr}}(b), \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r}), \boldsymbol{x}$ and the output of $C_{\mathtt{BCL}}$ is additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i), \mathtt{Enc}_{\mathtt{pr}}(b), \boldsymbol{y}$, where $b = 0$ by default.

Subtleties in the Security Analysis. While the above idea can be applied directly in [IS24], our case is more involved. Whereas in [IS24] the function f is implemented by a *leakage-tolerant circuit* (against parity leakage), in our case we can only implement f by a *leakage-resilient circuit* (against BCL). The main benefit of an LTC is that the leakage on inner wires of C can be simulated by some leakage on its input and output. In our case, we can only simulate the leakage by first simulating all inner wires of C and then applying the leakage function to the simulated inner wires. This requires us to always maintain the correct functionality when switching from encoded states to fake states in the security analysis.

Consider $\mathbf{Hybrid}_i$ where we want to replace the encoded input state of the i-th invocation by a fake state. The circuit $C_{\mathtt{BCL}}$ takes as input additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i-1}), \mathtt{Enc}_{\mathtt{pr}}(0), \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r}), \boldsymbol{x}$ and

outputs additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i), \mathtt{Enc}_{\mathtt{pr}}(0), \boldsymbol{y}$. To maintain the same functionality, we replace the input to be additive shares of

$$\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}), \mathtt{Enc}_{\mathtt{pr}}(1), \mathtt{Enc}_{\mathtt{pr}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i) \| \mathtt{Enc}_{\mathtt{pr}}(0) \| \boldsymbol{y}), \boldsymbol{x}.$$

By construction, the function f will just output $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i) \| \mathtt{Enc}_{\mathtt{pr}}(0) \| \boldsymbol{y}$. As a result, the outputs of $C_{\mathtt{BCL}}$ are additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i), \mathtt{Enc}_{\mathtt{pr}}(0), \boldsymbol{y}$, the same as $\mathbf{Hybrid}_{i-1}$. Also by the property of $C_{\mathtt{BCL}}$, we can show that the distributions of the leakage result in the i-th invocation are statistically close in $\mathbf{Hybrid}_{i-1}$ and $\mathbf{Hybrid}_i$. However, several difficulties remain.

First, we need to prove that the replacement does not influence the leakage results in future invocations. Since all future invocations only depend on the output in the i-th invocation, our idea is to argue that given the output in the i-th invocation, the outputs of $\mathbf{Hybrid}_{i-1}$ and $\mathbf{Hybrid}_i$ are indistinguishable. To this end, we convert f to be a deterministic function by letting f take another auxiliary encoding of randomness that is used as random tapes for the stateful circuit C and the parity-resilient encoding scheme.

Second, we also need to maintain the correct functionality in the $(i-1)$-th invocation, since a part of the input of the i-th invocation comes from the output of the $(i-1)$-th invocation. This requires us to also change the (auxiliary) input of the $(i-1)$-th invocation and argue the security of two consecutive executions. Recall that the circuit $C_{\mathtt{BCL}}$ is obtained by splitting the transcript of a parity-tolerant circuit $C_{\mathtt{pr}}$. We observe that the concatenation of two parity-tolerant circuits can be viewed as a single circuit with a relaxed variant of parity tolerance, which suffices for the concatenation to remain secure against BCL.

Finally, since replacing the input of the i-th invocation would affect both the $(i-1)$-th invocation and the i-th invocation, we need to argue the security against two BCL queries, one in each invocation. What makes this more challenging is that these two queries are made adaptively. To address this issue, we use a statistical security leveraging technique from [FRR+14], reducing security against 2-adaptive t-BCL to security against standard (non-adaptive) $2t$-BCL.

We refer the readers to Section 4 for the full details of our stateful BCL-resilient circuit construction.

2.3 Application: Protecting Computation Against Malware

In this section, we discuss the application of our main result to obtain a stateful variant of the two-party protocol of Goyal et al. [GIM+16]. As discussed in the introduction, this is motivated by the goal of protecting computations in a scenario where *all* computers may be corrupted by malware, but the malware has a bounded communication rate.

Concretely, we want two parties to jointly emulate a stateful circuit C, given a leakage-resilient secret sharing of the initial secret state $\boldsymbol{s}_0$. In the i-th epoch, the parties may obtain an external public input $\boldsymbol{x}_i$ and compute $(\boldsymbol{s}_i, \boldsymbol{y}_i) \leftarrow C[\boldsymbol{s}_{i-1}](\boldsymbol{x}_i)$ where $\boldsymbol{s}_i$ is the updated secret state and $\boldsymbol{y}_i$ is the public output.

The adversary $\mathcal{A}$ is modeled as follows: For a leakage bound parameter t, we consider $\mathcal{A}$ to be a tuple of three PPT algorithms $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}^\star)$. Here $\mathcal{A}_0, \mathcal{A}_1$ are the actual malicious applications running on the two parties and $\mathcal{A}^\star$ is the central adversary to which they report. Concretely,

- For both $i \in \{0,1\}$, $\mathcal{A}_i$ can read the entire view of P_i at any time but not modify the actions of P_i. In addition, $\mathcal{A}_i$ has access to a local memory of size t, which can be used, for example, to

store a state of size t for the next epoch.[3] In each epoch, $\mathcal{A}_i$ can perform any polynomial-time computation (regardless of space) and send at most t bits to $\mathcal{A}_{1-i}$.

- $\mathcal{A}^\star$ can see the communication between $\mathcal{A}_0$ and $\mathcal{A}_1$ and that between P_0 and P_1 but has no access to the memory or the view of $\mathcal{A}_0, \mathcal{A}_1, P_0, P_1$. In each epoch, $\mathcal{A}^\star$ may decide the public input depending on its view and learn the public output. In the context of the motivating application, we can think of $\mathcal{A}_0$ and $\mathcal{A}_1$ as two malicious programs running on the two servers and $\mathcal{A}^\star$ as a central adversary controlling them.

Very informally, the security requirement is that any adversary $\mathcal{A}$, who can invoke the two-party protocol any polynomial number of times (in the security parameter), cannot learn any further information beyond the queried inputs and their associated outputs. We refer to a two-party protocol satisfying the above security guarantee as a *stateful t-BCL resilient protocol*.

Sketch of Our Construction. To build a stateful t-BCL resilient protocol, the idea is to transform our stateful LRC for t-BCL to a two-party protocol. Recall that in our construction, we use a circuit $C_{\mathtt{BCL}}$ to compute 2-out-of-2 additive sharings of the wire values in $C_{\mathtt{pr}}$. To transform it to a 2-party protocol, we simply let each party hold one share of each additive sharing.

A technical issue that arises is handling AND gates in $C_{\mathtt{BCL}}$ that take inputs from both parts. Translating to the 2-party setting, we need to compute the AND of a bit from the first party and a bit from the second party. This is not a problem for t-BCL LRC since we only partition the *wires* into two disjoint parts and allow an AND gate to take inputs from both parts. In the 2-party setting, however, no single party can compute this AND gate locally. To address this issue, we follow [GIM+16] by using a $(2,1)$-OT with *joint-simulation security* to compute the AND gate on distributed inputs.

Intuitively, joint-simulation security requires the existence of a pair of simulators $(\mathtt{Sim}_1, \mathtt{Sim}_2)$ which can simulate both parties' views *jointly* by only taking the input and output of each party respectively, assuming that the simulators can share common randomness. This is different from standard security that only requires *individual simulation*, but the joint output of the two simulators can be distinguishable from the real execution due to inconsistent transcripts. Joint-simulation security is needed in our case since both parties are simultaneously corrupted by malware, and the adversary can easily detect inconsistency between the transcripts via low communication.

Jumping ahead, relying on joint-simulation security, we can simulate each party's view of the OT protocols by only using its own additive shares from $C_{\mathtt{BCL}}$. Under BCL, these additional wires do not help the adversary since each of $\mathcal{A}_0$ and $\mathcal{A}_1$ can compute these wires by themselves separately. We use the construction from [GIM+16] of oblivious transfer with joint-simulation security, which necessarily requires computational assumptions as it implies standard oblivious transfer. We refer readers to Section 4.4 for the formal definitions of the notion of joint-simulation security.

Now, our construction of the 2-party stateful t-BCL resilient protocol works as follows:

1. In the beginning, both parties take as input $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_0 \| 0)$, which is a 2-out-of-2 additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_0 \| 0)$, where $\boldsymbol{s}_0$ is the initial state, and $b = 0$ is a control bit which is set to be 0 by default.

[3] In our construction, each party will erase all data except the public output and the most updated state at the end of each epoch. Assuming erasures is necessary, since otherwise the adversary may only attack the initial state and eventually leak it in its entirety. The limited memory allows each adversary $\mathcal{A}_i$ to be stateful as well. The restriction on the adversary's state size, similarly to the restriction on the communication rate, can be motivated by a malware's goal to avoid being detected.

2. In each epoch, both parties hold additive shares of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}\|b)$ and receive a public input $\boldsymbol{x}$.

3. P_0 samples a random string $\boldsymbol{r}_0$ and sends $\boldsymbol{r}_0$ to P_1. Then P_1 computes $\boldsymbol{x} \oplus \boldsymbol{r}_0$. This step is to convert $\boldsymbol{x}$ to a random additive sharing of $\boldsymbol{x}$.

4. For the auxiliary random additive sharings of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$ and $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape})$, where recall that the latter is to make the function f deterministic (see Section 2.2), both parties simply sample random values as their shares.

5. Both parties compute an additive sharing for each wire in $C_{\mathtt{pr}}$ following $C_{\mathtt{BCL}}$. In particular, for each AND gate in $C_{\mathtt{BCL}}$ that takes inputs from both parties, they use a $(2,1)$-OT with joint simulation security to compute the AND result.

6. After evaluating $C_{\mathtt{pr}}$, both parties together hold an additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b)$ and an additive sharing of $\boldsymbol{y}$, denoted by $(\boldsymbol{r}_1, \boldsymbol{y} \oplus \boldsymbol{r}_1)$. Both parties exchange their shares of the additive sharing of $\boldsymbol{y}$ and compute the public output $\boldsymbol{y}$.

7. Both parties erase all data except the public output $\boldsymbol{y}$ and their shares of $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b)$.

We prove the security of our construction by converting any adversary $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}^\star)$ attacking our 2-party stateful t-BCL resilient protocol to an adversary $\mathcal{B}$ attacking the underlying stateful t-BCL LRC. We refer the readers to Section 4.4 for more details.

2.4 Deterministic LRCs for Simple Leakage Classes

We turn to describe our compilers for *deterministic* leakage-resilient circuits. Our compiler requires that the leakage class $\mathcal{L}$ be *decomposable*, in the following sense.

Definition 2 (Decomposable Leakage Functions)**.** *We say $\mathcal{L}$ is a decomposable leakage function class if for every $L : \{0,1\}^n \to \{0,1\}^m \in \mathcal{L}$ and every partition $S_1, S_2, \ldots, S_k$ of $[n]$, there exist leakage functions $L_1, L_2, \ldots, L_k \in \mathcal{L}$ and a function L_0 such that $L(\boldsymbol{x}) = L_0(L_1(\boldsymbol{x}_{S_1}), \ldots, L_k(\boldsymbol{x}_{S_k}))$. Here $\boldsymbol{x}_{S_i}$ denotes the vector $(x_i)_{i \in S_i}$.*

The need for this decomposition requirement will become apparent later. Our initial idea is to use the recent reduction from stateful LRCs to stateless LTCs given in [IS24], which we briefly describe below. Let $C[\boldsymbol{s}_0]$ be an unprotected stateful circuit. The LRC for C proceeds as follows:

- In the trusted setup, the initial state $\boldsymbol{s}_0$ is encoded into $\boldsymbol{s}'_0$ by an $\mathcal{L}$-leakage-resilient encoding.
- In each invocation, use an $\mathcal{L}$-LTC to compute the following function f:
  - Given the encoded state $\boldsymbol{s}'$ and the public input $\boldsymbol{x}$, decode $\boldsymbol{s}'$ to obtain $\boldsymbol{s}$.
  - Then f computes $C[\boldsymbol{s}](\boldsymbol{x})$ to obtain updated state $\tilde{\boldsymbol{s}}$ and public output $\boldsymbol{y}$.
  - Finally, f computes an encoding $\tilde{\boldsymbol{s}}'$ of the updated state, and outputs $(\tilde{\boldsymbol{s}}', \boldsymbol{y})$.

Relying on $\mathcal{L}$-leakage tolerance, the leakage query made by the adversary to the inner wires in each invocation can be reduced to a leakage query to the encoded (input and output) states. Since the states are protected by $\mathcal{L}$-leakage-resilient encodings, the leakage queries to the encoded states can be simulated without knowing the states.

We note that for most nontrivial functions, $\mathcal{L}$-LTCs require fresh randomness. Indeed, if an $\mathcal{L}$-LTC is deterministic, then every inner wire should be computable by some function in $\mathcal{L}$ on input and output of the circuit, which severely restricts the possible functions that can be computed. As a result, in the above construction, each invocation requires an additional random input to serve as the internal randomness for the underlying $\mathcal{L}$-LTC.

To eliminate the need for fresh randomness, a natural idea is to generate in each invocation a pseudorandom string that will be used as the internal randomness for the next invocation. This can be done as follows. First, add a short random seed $\mathtt{Seed}$ to the initial state. Then, in each invocation, f not only computes $C[\boldsymbol{s}](\boldsymbol{x})$ as described above, but also computes $\mathtt{PRG}(\mathtt{Seed}) = (\widetilde{\mathtt{Seed}}, \widetilde{\mathtt{Rtape}})$, where $\widetilde{\mathtt{Rtape}}$ will be used as the internal randomness of the $\mathcal{L}$-LTC in the next invocation. Then $\tilde{\boldsymbol{s}} \| \widetilde{\mathtt{Seed}}$ is viewed as the updated state. With this approach, we only need fresh randomness in the initial encoded state.

Security Issue. This seemingly straightforward idea unfortunately does not work. An immediate issue is that the randomness complexity of existing $\mathcal{L}$-LTC constructions grows linearly with the circuit size. This means that the pseudorandom string generated in the current invocation, which is bounded by the circuit size, is too short to be useful for the next invocation. This calls for the use of *randomness-efficient* $\mathcal{L}$-LTCs, which are the main technical ingredient in our solution.

However, even with such randomness-efficient $\mathcal{L}$-LTCs, a more severe issue is that a single leakage query may affect all later invocations. Consider an $\mathcal{L}$-leakage query in the first invocation. By the property of leakage tolerance, we can reduce it to an $\mathcal{L}$-leakage query to the input and output of the first invocation. Note that a part of the output of the first invocation is used as the internal randomness in the second invocation. In the worst case, the $\mathcal{L}$-leakage query to the output of the first invocation becomes an $\mathcal{L}$-leakage query to the internal randomness in the second invocation. Since the adversary can make another $\mathcal{L}$-leakage query in the second invocations, it can effectively make two $\mathcal{L}$-leakage queries in the second invocation, three $\mathcal{L}$-leakage queries in the third invocation, and so on. This accumulated leakage would eventually break the security of the above construction.

An Attempt to Solve the Second Issue. To address the second issue, we have to find a way to cut off the influence of a leakage query on future invocations. We note that $\mathtt{Rtape}$ can be viewed as an imperfect randomness. A natural idea is to apply a deterministic *randomness extractor* on $\mathtt{Rtape}$ so that the output is uniformly random and not affected by the leakage in the previous invocation. To be more concrete, in each invocation, we first apply a deterministic randomness extractor on $\mathtt{Rtape}$ and let $\mathtt{Rtape}'$ denote the output. Then we use $\mathtt{Rtape}'$ as the internal randomness of the $\mathcal{L}$-LTC in the current invocation. In this way the leakage on $\mathtt{Rtape}$ from the previous invocation is independent of $\mathtt{Rtape}'$.

To argue security, we also need to consider the leakage query in the current invocation. A security issue is that the adversary can apply an $\mathcal{L}$-leakage query on both the input of the extractor and the wires of the $\mathcal{L}$-LTC (recall that the output of the extractor is used as the internal randomness of the $\mathcal{L}$-LTC). Ideally, we only want the adversary to apply an $\mathcal{L}$-leakage query on the wires of the $\mathcal{L}$-LTC. To address this issue, we rely on the assumption that the leakage class $\mathcal{L}$ is decomposable: For an $\mathcal{L}$-leakage query L in the current invocation, we may decompose L into two different queries $L_1, L_2 \in \mathcal{L}$ where L_1 is applied on $\mathtt{Rtape}$ and L_2 is applied on the wires of the $\mathcal{L}$-LTC. By the property of the randomness extractor, $L_1(\mathtt{Rtape})$ is independent of $\mathtt{Rtape}'$. Thus, we can rely on the leakage tolerance to simulate the result of the leakage query L_2.

Unfortunately, the above discussion assumes a black-box access to the extractor. In the actual construction, however, the adversary can attack the wires of the extractor as well. To preserve the security, we have to use an $\mathcal{L}$-LTC for the extractor. Although the extractor itself is deterministic, the $\mathcal{L}$-LTC for the extractor may be randomized. Then we again face the problem of generating the internal randomness for the extractor circuit.

Our Solution. Our next attempt is to directly compute the deterministic extractor without any protection. At a first glance, this idea sounds problematic since allowing the adversary to attack the inner wires of the extractor may break the security entirely. We make the following observations.

- Due to the use of the extractor, the leakage on $\mathtt{Rtape}$ in the previous invocation is already independent of the randomness $\mathtt{Rtape}'$ (the output of the extractor) in the current invocation.
- We still need to address leakage from the inner wires of the extractor. However, if each such inner wire can be simulated based on a *single* bit of $\mathtt{Rtape}'$, then this leakage can be simulated by leakage from the output wires of the extractor.

To realize the above local simulation property, our idea is to divide $\mathtt{Rtape}$ into several pieces and only extract a single random bit from each piece, so that the inner wires only depend on this single output bit. To be more concrete, suppose ℓ is the number of random bits we need for the $\mathcal{L}$-LTC. We first divide $\mathtt{Rtape}$ into ℓ pieces, denoted by $\{\mathtt{Rtape}_i\}_{i=1}^{\ell}$. Then we apply a deterministic randomness extractor $\mathtt{Ext}(\mathtt{Rtape}_i)$ which outputs a single bit b_i. The concatenation of all ℓ bits forms $\mathtt{Rtape}' = (b_1, \ldots, b_\ell)$.

For each $i \in \{1, \ldots, \ell\}$, to represent each wire of the extractor as a variable that only depends on a single output bit, we reverse-sample $\mathtt{Rtape}_{i,0}$ and $\mathtt{Rtape}_{i,1}$ given the $\mathcal{L}$-leakage from the previous invocation such that $\mathtt{Ext}(\mathtt{Rtape}_{i,0}) = 0$ and $\mathtt{Ext}(\mathtt{Rtape}_{i,1}) = 1$. Then we may set $\mathtt{Rtape}_i = \mathtt{Rtape}_{i,0} \cdot (1 - b_i) + \mathtt{Rtape}_{i,1} \cdot b_i$. This ensures that each input bit of the extractor (hence also its inner wires) only depends on a single output bit b_i. Now the $\mathcal{L}$-leakage query in the current invocation can be reduced to an $\mathcal{L}$-leakage query on the wires of the $\mathcal{L}$-LTC.

One issue we omitted above is that the $\mathcal{L}$-leakage from the previous invocation is performed on the entire $\mathtt{Rtape}$ rather than each single piece $\mathtt{Rtape}_i$ separately. It is not clear how to reverse-sample $\mathtt{Rtape}_{i,0}$ and $\mathtt{Rtape}_{i,1}$ while matching the global leakage. Here we again rely on the property of decomposable leakage function classes. The leakage query L' in the previous invocation is first decomposed to $L'_1, \ldots, L'_{\ell+1}$ where L'_i is applied on $\mathtt{Rtape}_i$ for all $i \leq \ell$ and $L'_{\ell+1}$ is applied on the rest of wires. Now we can reverse-sample $\mathtt{Rtape}_{i,0}$ and $\mathtt{Rtape}_{i,1}$ given the leakage $L'_i(\mathtt{Rtape}_i)$. Finally, to ensure that this reverse sampling can be carried out efficiently, we rely on simple specific constructions of deterministic extractors: the XOR extractor for depth-1 $\mathcal{AC}0$-leakage, and the inner-product extractor for parity leakage.

2.5 Randomness-Efficient Leakage-Tolerant Circuits

The above overview shows how to reduce the construction of deterministic stateful LRCs for simple leakage classes to the construction of LTCs *with sublinear randomness* for the same classes. The final and most technical ingredient in our solution is a derandomization of recent LTC constructions from [IS24] for two simple types of global leakage: $\mathcal{AC}0$ leakage (or *D1 leakage* for short), outputting t disjunctions or conjunctions of any subset of wires or their negations, and parity leakage, outputting t parities of subsets of wires.

Derandomization for D1 Leakage. At a high level, the construction in [IS24] can be divided into two parts: (1) the outer construction for computing the target function f assuming the existence of D1-tolerant circuits for linear functions, and (2) the inner construction for computing any linear function with D1-tolerance.

Outer Construction. For the outer construction, the main observation is that a D1-leakage query cannot distinguish whether the secret of a random additive sharing with $\kappa + 1$ shares is 0 or 1. Indeed, a D1-leakage query just computes `OR` of a subset of shares or their flips. If the subset size is bounded by κ, then all shares in the subset are independent of the secret. Otherwise, the leakage query would compute `OR` of κ random bits, which is 1 with overwhelming probability. Utilizing this observation, the idea in [IS24] is to compute an additive sharing for each wire value in the circuit implementation of f.

To derandomize the outer construction, our solution is to use the output of a strong κ-wise independent PRG to mask the wire values in the circuit. To be more concrete, suppose C is a circuit implementation of f. Let $G : \{0,1\}^\lambda \to \{0,1\}^{|C|}$ be a strong κ-wise independent PRG. Instead of computing an additive sharing for each wire value in C, we choose to compute

$$(\boldsymbol{r}, G(\boldsymbol{r}) \oplus \boldsymbol{w}),$$

where $\boldsymbol{w}$ is the vector of all wire values in C. This allows us to reduce the randomness complexity in the outer construction. As for the security, if a D1-leakage query touches at most κ wires, by the property of G, all these wires are uniform bits and independent of $\boldsymbol{w}$. On the other hand, if a D1-leakage query touches more than κ wires, then at least κ wires of them are uniform bits and the query result is 1 with overwhelming probability.

In our construction, we actually use a strong and linear κ-wise independent PRG. And there are some other changes we need to make to derandomize the outer construction in [IS24].

Inner Construction. As for the inner construction, the authors in [IS24] proved that the probing-tolerant variant of the ISW construction has already achieved D1 tolerance. We note that the inner construction is used to compute every gate in the circuit implementation of f. Thus, instead of directly derandomizing the inner construction, we choose to derandomize the *random tapes* used by the inner construction: We will generate the random tapes such that the random tapes of any κ invocations of the inner construction are uniformly random. This can be done by using a κ-wise independent pseudo-random source. By the property of the ISW construction, we manage to show that: If the D1-leakage query touches at most κ invocations, it is the same as using uniform random tape for each invocation as [IS24]. Otherwise, it must contain a subset of κ wires which are uniformly random and the query result is 1 with overwhelming probability.

One issue we omitted above is that the adversary can also attack the inner wires when generating a κ-wise independent pseudo-random source. Our construction addresses this issue by following the idea in [ISW03, IKL$^+$13] that is used to build randomness-efficient probing-tolerant circuits: We first compute $\kappa + 1$ copies of a κ-wise independent PRG and compute the XOR of the outputs as the random source. Intuitively, if the D1-leakage query touches at most κ copies of the PRG circuit, then at least one copy is not touched and the random source is κ-wise independent even given the query result. Otherwise, the D1-leakage query would touch at least κ random bits and the query result is 1 with overwhelming probability. See Section 6.1 for a more detailed overview.

Derandomization for Parity Leakage. The construction of parity-tolerant circuits in [IS24] is also obtained in two steps.

In the first step, for an unprotected circuit $\tilde{C}$, the authors in [IS24] introduce the notion of parity-to-probing circuit implementation of $\tilde{C}$, which is a tuple (I, C, O) where I is an input encoder and O is an output decoder. Intuitively, it satisfies that any parity query to the inner wires of C can be reduced to probing queries to the inner wires of $\tilde{C}$. The authors in [IS24] show that a parity-tolerant circuit can be constructed from a parity-to-probing circuit in a black-box way.

Then in the second step, the authors construct a circuit compiler that transforms any unprotected circuit $\tilde{C}$ to a parity-to-probing circuit implementation of $\tilde{C}$. For simplicity, in this overview, we only give intuitions about derandomization of parity-to-probing circuits.

For an unprotected circuit $\tilde{C}$ with inner wires $\boldsymbol{w} \in \{0,1\}^{|\tilde{C}|}$, the construction of a parity-to-probing circuit implementation is boiled down to use a circuit C to compute the following values:

$$(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}),$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ are two random vectors of size $|\tilde{C}|$, and $*$ denotes the coordinate-wise multiplication operation. To see why this is secure, note that a parity query is to compute the parity of a subset of wires in C.

- If the parity query touches at most k wires in $\boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}$, then the query result can be perfectly simulated by learning the corresponding wires in $\boldsymbol{w}$, thus reducing to probing queries to the underlying circuit $\tilde{C}$.

- Otherwise, the query result is masked by an inner product between two random vectors of size k. We can show that with probability $1 - 2^{-k}$, the query result is a uniform bit. Thus, in this case the query result can be directly simulated by a random bit without making any probing queries to the underlying circuit $\tilde{C}$.

For simplicity, we omit the concrete construction of C but focus on how to derandomize $\boldsymbol{u}$ and $\boldsymbol{v}$.

We first observe that all the inner wires in C is linear in $\boldsymbol{u}$ and $\boldsymbol{v}$. Thus, it is sufficient to use *parity-resilient* pseudo-random sources for $\boldsymbol{u}$ and $\boldsymbol{v}$. However, there are two difficulties:

- Since the adversary can also attack the inner wires of the generation process, we need to use a parity-tolerant circuit to generate the parity-resilient pseudo-random source.

- We also need to use a *randomness-efficient* parity-tolerant circuit. Recall that our goal is to generate a pseudo-random source with small amount of random bits. If the randomness complexity of the generation circuit is proportional to the circuit size, then we do not achieve any saving.

Note that the purpose of derandomizing parity-to-probing circuits is to construct randomness-efficient parity-tolerant circuits. And the above idea has already required a randomness-efficient parity-tolerant circuit, which appears to create a circularity.

Our idea is to analyze the concrete requirements for derandomizing $\boldsymbol{u}$ and $\boldsymbol{v}$. We manage to generating each source without using parity-tolerant circuits, thus avoiding the circularity.

Derandomization of $\boldsymbol{v}$. We observe that the security analysis about the construction in [IS24] also holds when $\boldsymbol{v}$ is given. Intuitively, this is because after $\boldsymbol{v}$ is given, the parity query is masked by a random linear combination of $\boldsymbol{u}$. Since each bit of $\boldsymbol{u}$ is uniformly random, as long as the coefficient of one bit in $\boldsymbol{u}$ is not 0, the query result is a random bit. When the query result touches at least k wires in $\boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}$ and $\boldsymbol{v}$ is randomly sampled and given, with probability $1 - 2^{-k}$, the coefficients of $\boldsymbol{u}$ are not all 0. Thus, we can continue to use a random bit to simulate the query result.

Following this observation, we show that we can use an *unprotected* parity-resilient PRG to generate $\boldsymbol{v}$. Intuitively, this is because after $\boldsymbol{v}$ is given, all the inner wires of the PRG are fixed, which do not affect the distribution of the query result.

Derandomization of $\boldsymbol{u}$. Unfortunately the same analysis does not work for $\boldsymbol{u}$. Our idea is to find a pseudo-random source for $\boldsymbol{u}$ that can be generated by a *linear circuit*. In this way, each inner wire in the circuit can be computed by a linear combination of bits in $\boldsymbol{u}$. Thus, under parity query, being able to attack the inner wires of the generation circuit does not give any additional advantage to the adversary.

Our core lemma shows that, when $\boldsymbol{v}$ is a public and parity-resilient pseudo-random source, it is sufficient to use a k-wise independent pseudo-random source for $\boldsymbol{u}$. Note that we can generate such a pseudo-random source by using a linear k-wise independent PRG.

In summary, our construction uses an unprotected parity-resilient PRG to prepare $\boldsymbol{v}$ and uses a linear k-wise independent PRG to prepare $\boldsymbol{u}$. This allows us to achieve poly-logarithmic randomness complexity for the parity-to-probing circuits. We refer the readers to Section 7.1 for more details.

2.6 Discussion and Open Questions

Our work leaves several interesting directions for future research. We discuss some of the open questions and natural barriers one encounters when trying to solve them.

Deterministic LRCs Beyond D1 Leakage and Parity Leakage. We obtained the first stateful LRC against bounded-communication leakage. However, this construction is inherently randomized, where the randomness complexity is proportional to the circuit size. In contrast, for D1 leakage and parity leakage, we were able to construct *deterministic* LRCs that only require randomness in the initial state. Our attempt to extend our approach to richer leakage classes encountered the following two difficulties. First, our general compiler relies on a randomness-efficient LTC. Obtaining LTCs for leakage classes beyond D1 and parity leakage is a challenging open problem even without any restrictions on the randomness complexity.

Second, our general compiler only works for *decomposable* leakage classes. In particular, this means that even a randomness-efficient LTC for BCL would not suffice. Recall that in our construction, each invocation outputs a pseudorandom string that is used as the random tape in the next invocation. To cut off the influence of a leakage query on future invocations, instead of directly using the pseudo-random string as the random tape, we apply a randomness extractor and use the output as the random tape. To avoid using an LTC for the extractor, the pseudorandom string is divided into several pieces, where a deterministic extractor is used to extract a single bit from each piece. In this way, each inner wire of the extractor only depends on its output bit and, consequently, attacking the inner wires of the extractor does not give the adversary any additional advantage. To make this idea work, we require that the leakage class is decomposable so that we can reverse-sample each piece of the pseudorandom string independently given the leakage on this piece.

One potential solution would be using a different approach that does not require decomposability of the leakage class to prepare the random tape for the next invocation. For example, what if we directly use the pseudorandom string from the last invocation as the random tape in the current invocation? As discussed above, in this case, an adversary effectively can make two leakage queries in the current invocation. And the leakage results may further influence all future invocations. However, the hope is that the first query is restricted to the random tape, which is independent of the secret state. Thus, a potential avenue for making progress is to design an LTC such that any two leakage queries, where the first query is restricted to the random tape, can be reduced to a single leakage query to its input and output.

Derandomization of Stateless LRCs for BCL. We note that a deterministic LRC implies a randomness-efficient stateless LRC, and constructing randomness-efficient stateless LRCs itself is already an interesting question. Thus, a weaker goal would be constructing randomness-efficient stateless LRCs for BCL.

A natural approach is to derandomize the stateless LRC from [GIM+16]. This task can be divided into two parts: (1) what kind of pseudorandom source is sufficient to replace the random tape in this construction? (2) how can such a pseudorandom source be sampled securely? We note that in [GIM+16], to obtain leakage resilience against BCL, the authors use a parity-resilient circuit and rely on the generalized small-bias masking lemma. Although we may use our techniques to obtain a randomness-efficient parity-resilient circuit, (the proof of) the generalized small-bias masking lemma crucially relies on the fact that the mask string is uniformly random. It is not even clear what kind of pseudorandom source one should use to derandomize the mask string. Having a closer look at this problem, the issue is that the lamma allows the leakage function to apply any unbounded function on the mask string (which belongs to the same part in the partition). We cannot hope for a pseudorandom source to fool an arbitrary unbounded function.

A possible way around this barrier is to focus on an even weaker goal, say protecting against a sub-class $\mathcal{L}'$ of BCL that is decomposable. A natural example is leakage functions that compute sums (or weighted sums) over the integers of subsets of wires. For such leakage classes, it suffices to use a pseudorandom source that fools any leakage function in $\mathcal{L}'$. We note that if A is a parity-resilient pseudorandom source and R is a random string, then according to the generalized small-bias masking lemma $(R, R \oplus A)$ is indistinguishable from a uniform string by BCL. Thus, we can use $(R, R \oplus A)$ as a pseudorandom source for the mask string. This reduces the randomness complexity by a factor of 2. Now we can recursively apply this idea so that eventually we obtain a pseudorandom source that can be generated from a polylogarithmic amount of random bits. This solves the first part of the difficulty.

However the second part of the difficulty stops us from making further progress: How can we generate such a pseudorandom source while ensuring that attacking the inner wires does not give any additional advantage to the adversary?

Deterministic LRCs with Efficient Simulation. One limitation of our general compiler for deterministic LRCs is that the simulator needs to invoke the simulator of an underlying LTC. Indeed, in the security proof, the simulator is required to reduce the leakage query on the inner wires of the stateful circuit to a leakage query on the circuit input and output. This is in contrast to the general compiler for LRCs in [IS24] which has efficient simulation even if the underlying LTCs do not.

In [IS24], the authors achieve efficient simulation by allowing a fake execution of the stateful

circuit so that the leakage result can be simulated by directly applying the leakage function on the wire values in the fake execution. However, to support a fake execution, the construction in [IS24] requires to take an auxiliary input in each invocation: In the real world, the auxiliary input is just a random string from the random tape. In the ideal world, the simulator can program the auxiliary input, depending on the public output in this invocation, to achieve the same functionality.

In our case, the main difference is that the resulting stateful circuit does not take any random tape in each invocation except the first invocation. This means that the functionality is fixed once we fix the initial state and the random tape in the first execution. It is not clear how to support a fake execution that maintains the same functionality when the input is adaptively chosen by the adversary in each invocation.

3 Preliminaries

3.1 Leakage-Resilient and Leakage-Tolerant Circuits

In this section, we define the notion of leakage-resilient circuits and leakage-tolerant circuits for general leakage classes.

We assume that the leakage classes considered in this work are closed under restrictions, i.e., for all $L : \{0,1\}^n \to \{0,1\}^m$ in $\mathcal{L}$, for all $S \subset \{1,\ldots,n\}$, and for all $\boldsymbol{x}_S \in \{0,1\}^{|S|}$, we require that $L(\boldsymbol{x})$ given $\boldsymbol{x}_S$ is also in $\mathcal{L}$. We also assume that the leakage classes are closed under negation of input bits. I.e., for all $L : \{0,1\}^n \to \{0,1\}^m$ in $\mathcal{L}$ and for all $\boldsymbol{c} \in \{0,1\}^n$, $L'(\boldsymbol{x}) = L(\boldsymbol{x} \oplus \boldsymbol{c})$ is also in $\mathcal{L}$. For a leakage bound parameter t, we say $\mathcal{L}$ is the t-leakage class of $\mathcal{L}'$ if $\mathcal{L}$ contains all leakage functions L such that there exists $L'_1, \ldots, L'_t \in \mathcal{L}'$ with the same input length as L and $L(\boldsymbol{x}) = (L'_1(\boldsymbol{x}), \ldots, L'_t(\boldsymbol{x}))$.

We borrow the definition of leakage-resilient circuits from [GIM+16] and the definition of leakage-tolerant circuits from [IS24] as follows.

Definition 3 ([GIM+16])**.** *For a (possibly randomized) function $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$, a leakage-resilient circuit for f is defined by (I, C, O), where*

- *$I : \{0,1\}^{n_i} \to \{0,1\}^{\hat{n}_i}$ is a randomized input encoder, which maps an input $\boldsymbol{x}$ to an encoded input $\hat{\boldsymbol{x}}$,*
- *C is a randomized circuit, mapping an encoded input $\hat{\boldsymbol{x}} \in \{0,1\}^{\hat{n}_i}$ to an encoded output $\hat{\boldsymbol{y}} \in \{0,1\}^{\hat{n}_o}$,*
- *$O : \{0,1\}^{\hat{n}_o} \to \{0,1\}^{n_o}$ is a deterministic output decoder, mapping $\hat{\boldsymbol{y}}$ to an output $\boldsymbol{y}$.*

Let $\mathcal{L}$ be a class of leakage functions. We say C is an $(\mathcal{L}, \epsilon)$-leakage-resilient implementation of f if

- *Correctness: For any input $\boldsymbol{x} \in \{0,1\}^{n_i}$, the following two distributions are identical:*

$$f(\boldsymbol{x}) \equiv O(C(I(\boldsymbol{x}))).$$

- *Leakage Resilience: For all $L \in \mathcal{L}$ with input size $|C|$ and for all $\boldsymbol{x}, \boldsymbol{x}' \in \{0,1\}^{n_i}$, the statistical distance between the following two distributions is at most ϵ:*

$$(L(\tau(C, I(\boldsymbol{x}))), O(C(I(\boldsymbol{x})))) \approx_\epsilon (L(\tau(C, I(\boldsymbol{x}'))), O(C(I(\boldsymbol{x}')))),$$

  where $\tau(C, \hat{\boldsymbol{x}})$ denotes the random variables of the wire values of C when taking $\hat{\boldsymbol{x}}$ as input.

Definition 4 (Leakage-Tolerant Circuit)**.** *For a (possibly randomized) function $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$, a leakage-tolerant circuit for f is defined by a randomized circuit C mapping an input $\boldsymbol{x} \in \{0,1\}^{n_i}$ to an output $\boldsymbol{y} \in \{0,1\}^{n_o}$. Let $\mathcal{L}$ be a class of leakage functions. We say C is an $(\mathcal{L}, \epsilon)$-leakage-tolerant implementation of f if*

- *Correctness: For any input $\boldsymbol{x} \in \{0,1\}^{n_i}$, the following two distributions are identical:*

$$f(\boldsymbol{x}) \equiv C(\boldsymbol{x}).$$

- *Leakage Tolerance: There exists a simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ with the following syntax:*
  - *$\mathtt{Sim}_1$ takes as input a leakage function $L \in \mathcal{L}$ with input size $|C|$ and outputs a state $\mathtt{st}$ as well as a leakage function $L' \in \mathcal{L}$ with input size $n_i + n_o$,*
  - *$\mathtt{Sim}_2$ takes as input a state $\mathtt{st}$ and the output of the ideal leakage $L'(\cdot)$ on the input and output of f, and outputs a string $\boldsymbol{b}$,*

  such that for all input $\boldsymbol{x} \in \{0,1\}^{n_i}$ and for all $L \in \mathcal{L}$, the following two distributions are ϵ-close (in statistical distance):

$$(L(\tau(C, \boldsymbol{x})), C(\boldsymbol{x})) \approx_\epsilon (\mathtt{Sim}_2(\mathtt{st}, L'(\boldsymbol{x}, \boldsymbol{y})), \boldsymbol{y}) : \boldsymbol{y} \leftarrow f(\boldsymbol{x}), (\mathtt{st}, L') \leftarrow \mathtt{Sim}_1(L)$$

  where $\tau(C, \boldsymbol{x})$ denotes the random variables of the wire values of C when the input is $\boldsymbol{x}$.

3.2 Stateful Leakage-Resilient Circuits

We follow the definition of stateful circuits in [ISW03] and extend the definition of stateful $\mathcal{L}$-leakage-resilient circuits in [FRR+10] to the case of computational security.

Definition 5 (Memory Cells [ISW03])**.** *A memory cell is a stateful gate with fan-in 1. On any invocation of the circuit, it outputs the previous input to the gate, and stores the current input for the next invocation.*

Definition 6 (Stateful Circuits [ISW03])**.** *A stateful circuit C is a circuit with the extensions that (1) C may contain memory cells, and (2) C may contain cycles as long as every cycle traverses at least one memory cell. Let $\boldsymbol{s}_0$ be the initial state for the memory cells. We write $C[\boldsymbol{s}_0]$ for the circuit C with memory cells initially filled with $\boldsymbol{s}_0$.*

Remark 1. *Stateful circuits can have external input and output wires. One example given in [ISW03] is an AES circuit, where the internal memory cells contain the secret key, the input wires are the plaintext, and the output wires are the corresponding ciphertext.*

Definition of Leakage-Resilient Circuits. Let $\mathcal{L}$ be a leakage class. In the following, we will use the notation $C[\boldsymbol{s}_0]$ to refer to an unprotected stateful circuit with initial state $\boldsymbol{s}_0$ and $C'[\boldsymbol{s}'_0]$ to refer to a stateful circuit with initial state $\boldsymbol{s}'_0$ that is an $\mathcal{L}$-leakage-resilient implementation of $C[\boldsymbol{s}_0]$. To define $\mathcal{L}$-leakage resilience of $C'[\boldsymbol{s}'_0]$, we consider the real-world execution and the ideal-world execution.

Real-world execution. In the real world, we consider an $(S, q, \mathcal{L})$-adversary $\mathcal{A}$ defined as follows. The adversary $\mathcal{A}$ is a circuit of size at most S with access to $C'[\boldsymbol{s}'_0]$. $\mathcal{A}$ may invoke C' at most q

times. For each $i \in \{1, \ldots, q\}$, in the i-th invocation, $\mathcal{A}$ may adaptively choose an input $\boldsymbol{x}_i$ and a leakage function $L_i \in \mathcal{L}$ with input size $|C'|$ based on the observed output values and answers to the leakage queries in previous invocations, and learns the output $\boldsymbol{y}_i$ corresponding to the chosen input and the answer to the leakage query L_i. The transcript of the real-world experiment is the view of $\mathcal{A}$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{q}$.

Ideal-world execution. In the ideal-world execution, we consider an (S_0, q)-adversary $\mathtt{Sim}$ defined as follows. $\mathtt{Sim}$ is a circuit of size at most S_0 with oracle access to both the real world adversary $\mathcal{A}$ and $C[\boldsymbol{s}_0]$. $\mathtt{Sim}$ may invoke C at most q times. For each $i \in \{1, \ldots, q\}$, in the i-th invocation, $\mathtt{Sim}$ may adaptively choose an input $\boldsymbol{x}_i$ based on the observed output values in previous invocations and learns the output $\boldsymbol{y}_i$ corresponding to the chosen input. The transcript of the ideal-world experiment is the output of $\mathtt{Sim}$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{q}$.

Now we are ready to define stateful $\mathcal{L}$-leakage-resilient circuits.

Definition 7 (Stateful LRC: Concrete Version)**.** *Let $\mathcal{L}$ be a class of leakage functions, C, C' stateful circuits, and $\mathcal{T}$ a randomized state encoder mapping an initial state $\boldsymbol{s}_0$ to an encoded initial state $\boldsymbol{s}'_0$. We say that $(C', \mathcal{T})$ is an $(S, S_0, S_1, q, \epsilon)$-$\mathcal{L}$-leakage-resilient implementation of C, if for every $(S, q, \mathcal{L})$-adversary $\mathcal{A}$ there exists an ideal (S_0, q)-adversary $\mathtt{Sim}$, such that for every initial state $\boldsymbol{s}_0$, every size-S_1 distinguisher has at most an ϵ advantage distinguishing between the following two experiments:*

- *The real-world transcript of $\mathcal{A}$ interacting with $C[\boldsymbol{s}_0]$;*
- *The ideal-world transcript of $\mathtt{Sim}$ interacting with $C'[\mathcal{T}(\boldsymbol{s}_0)]$.*

Definition 8 (Stateful LRC Compiler)**.** *Let $\mathcal{T}_C$ be a polynomial-time algorithm mapping a stateful circuit C and a security parameter κ to a stateful circuit C', and and $\mathcal{T}_s$ be a PPT algorithm mapping an initial state $\boldsymbol{s}_0$ and a security parameter κ to an encoded initial state $\boldsymbol{s}'_0$. For a leakage class $\mathcal{L}$, size bounds S, S_0 and query bound q (all functions of κ), we say that $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ is an $(\mathcal{L}, S, S_0, q)$-LRC compiler if for every polynomial $p(\cdot)$ there is a negligible $\epsilon(\cdot)$ such that for every $\kappa \in \mathbb{N}$ and and for every size-$S(\kappa)$ circuit C, the pair $(C', \mathcal{T})$ is an $(S, S_0, S_1, q, \epsilon)$-$\mathcal{L}$-leakage-resilient implementation of C, where $C' = \mathcal{T}_C(1^\kappa, C)$, $\mathcal{T}(\boldsymbol{s}_0) = \mathcal{T}_s(1^\kappa, \boldsymbol{s}_0)$, $\mathcal{L} = \mathcal{L}(\kappa)$, $S = S(\kappa)$, $S_0 = S_0(\kappa)$, $S_1 = p(S_0)$, $q = q(\kappa)$, and $\epsilon = \epsilon(\kappa)$. Furthermore, the simulator circuit $\mathtt{Sim}$ (guaranteed by Definition 7) can be computed from the $(S, q, \mathcal{L})$-adversary circuit $\mathcal{A}$ (in Definition 7) in time* $\mathrm{poly}(S_0(\kappa))$.

- *We say that $\mathtt{Comp}$ is a (computational) $\mathcal{L}$-LRC compiler if for every polynomial adversary bound S there is a polynomial simulator bound S_0 such that $\mathtt{Comp}$ is an $(\mathcal{L}, S, S_0, S)$-LRC compiler;*
- *We say that $\mathtt{Comp}$ is a* statistical *$\mathcal{L}$-LRC compiler if for every polynomial query bound q and (possibly exponential) adversary bound $S = S(\kappa)$, there is a simulator bound $S_0 = \mathrm{poly}(S, \kappa)$, such that $\mathtt{Comp}$ satisfies a stronger variant of $(\mathcal{L}, S, S_0, q)$-LRC compiler where the distinguisher size S_1 is unbounded.*

Remark 2 (Black-box and super-polynomial simulation)**.** *Our constructions will all achieve the stronger notion of* black-box *simulation, where there is a universal $\mathtt{Sim}$ that makes a black-box use of $\mathcal{A}$. In some cases, we will need to relax the above definitions of computational or statistical LRC compiler by allowing the simulator size S_0 to grow exponentially with the number of bits returned by the leakage function.*

4 Stateful LRC for Bounded-Communication Leakage

In this section, we give the formal description of our construction of stateful leakage-resilient circuits for bounded-communication leakage.

4.1 Preliminaries

We recall the definition of the t-bounded-communication leakage class from [GIM+16, GIW17] and the generalized small-bias masking lemma from [GIM+16].

Definition 1 (t-BCL [GIM+16, GIW17])**.** *Let $t \in \mathbb{N}$ be a leakage bound parameter. We say that a deterministic 2-party protocol is t-bounded if its communication complexity per party is at most t bits (so the total communication is bounded by $2t$ bits). Given a partitioned input $x = (x_1, x_2)$, the t-bounded-communication leakage (t-BCL) class $\mathcal{L}^t_{\mathtt{BCL}}$ contains all leakage function $L : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^{2t}$ such that $L(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is the transcript of a t-bounded protocol $\Pi(\boldsymbol{x}_1, \boldsymbol{x}_2)$.*

Lemma 1 (Generalized Small-Bias Masking Lemma [GIM+16])**.** *Let t be a positive integer. Let X, Y be random variables of size n such that X and Y are ϵ-indistinguishable by parity. Then for all t-bounded protocol $\Pi : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $\Pi(R, X \oplus R)$ and $\Pi(R, Y \oplus R)$ are statistically close with distance $2^t\epsilon$.*

We note that Lemma 1 focus on the *output* of Π rather than the *transcript* of Π, which does not capture the leakage class $\mathcal{L}^t_{\mathtt{BCL}}$. Thus, we generalize Lemma 1 to the transcript of Π in Lemma 2.

Lemma 2. *Let t be a positive integer. Let X, Y be random variables of size n such that X and Y are ϵ-indistinguishable by parity. Then for all $L \in \mathcal{L}^t_{\mathtt{BCL}}$, $L(R, X \oplus R)$ and $L(R, Y \oplus R)$ are statistically close with distance $2^t\epsilon$.*

Proof. By the definition of statistical distance, it is sufficient to show that for any distinguisher $D : \{0,1\}^{2t} \to \{0,1\}$,

$$|\Pr[D(L(R, X \oplus R)) = 0] - \Pr[D(L(R, Y \oplus R)) = 0]| \leq 2^t\epsilon$$

Suppose Π is the t-bounded 2-party protocol associated with L. We construct a 2-party protocol Π' as follows:

- Π' runs the 2-party protocol Π and outputs the result of applying D on the transcript.

Then Π' is also a t-bounded 2-party protocol. By Lemma 1, $\Pi(R, X \oplus R)$ and $\Pi(R, Y \oplus R)$ are statistically close with distance $2^t\epsilon$, which implies that $|\Pr[D(L(R, X \oplus R)) = 0] - \Pr[D(L(R, Y \oplus R)) = 0]| \leq 2^t\epsilon$. □

Our construction requires a compiler that transforms a circuit C into a 1-parity-tolerant circuit. We borrow the following result from [IS24].

Theorem 1 ([IS24])**.** *There exists a polynomial-time circuit compiler that takes as input $(C_f, 1^t, 1^\kappa)$, where C_f is a circuit of size s that computes f, and outputs a $(t, 2^{-\kappa})$-parity-tolerant implementation of f with circuit size poly(t, κ, s).*

4.2 Formal Description

Theorem 2. *For all $t \in \mathbb{N}$, there exists a statistical t-BCL-LRC compiler* $\mathtt{Comp}$.

Let $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$ be a parity-resilient encoding scheme $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$ with statistical error ϵ such that when r is a random bit, $\mathtt{Enc}_{\mathtt{pr}}(r)$ is uniformly distributed over $\{0,1\}^{|\mathtt{Enc}_{\mathtt{pr}}(r)|}$. Such an encoding scheme is known in [IS24] with encoding size linear in $\log \frac{1}{\epsilon}$. For a vector $\boldsymbol{x}$, we use $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{x})$ to denote the concatenation of the encodings of every bit in $\boldsymbol{x}$. The encoding scheme in [IS24] satisfies that for all $\boldsymbol{x}$ and $\boldsymbol{y}$ of the same length, the statistical distance between $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{x})$ and $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{y})$ is also bounded by ϵ.

Let $(\mathtt{Enc}_{\mathtt{BCL}}, \mathtt{Dec}_{\mathtt{BCL}})$ be an encoding scheme where $\mathtt{Enc}_{\mathtt{BCL}}(m)$ outputs a random 2-out-of-2 additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(m)$, and $\mathtt{Dec}_{\mathtt{BCL}}(c)$ views $c = (c_1, c_2)$ as a 2-out-of-2 additive sharing and outputs $\mathtt{Dec}_{\mathtt{pr}}(c_1 \oplus c_2)$. By the property of $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$, when r is a random bit, $\mathtt{Enc}_{\mathtt{BCL}}(r)$ is uniformly distributed over $\{0,1\}^{|\mathtt{Enc}_{\mathtt{BCL}}(r)|}$. Similarly, for a vector $\boldsymbol{x}$, we use $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{x})$ to denote the concatenation of the encodings of every bit in $\boldsymbol{x}$.

Our construction of $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ works as follows. $\mathcal{T}_s$ takes the initial state $\boldsymbol{s}_0$ as input. Then $\mathcal{T}_s$ outputs $\boldsymbol{s}'_0 = \mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_0 \| 0)$.

Let $C[\boldsymbol{s}_0]$ be the input stateful circuit. Let f be a deterministic function defined as follows.

1. f takes as input $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}\|b)$, $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$, $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape})$, $\boldsymbol{x}$.
2. f decodes the encodings and obtains $\boldsymbol{s}, b, \boldsymbol{r}, \mathtt{Rtape}$. Then $\mathtt{Rtape}$ is partitioned into two parts $\mathtt{Rtape} = (\mathtt{Rtape}_1, \mathtt{Rtape}_2)$.
3. f computes $C[\boldsymbol{s}](\boldsymbol{x})$ with $\mathtt{Rtape}_1$ as the random tape. Let $\boldsymbol{y}$ be the public output and $\tilde{\boldsymbol{s}}$ be the updated state.
4. f computes $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b)$ with $\mathtt{Rtape}_2$ as the random tape.
5. If $b = 0$, f outputs $(\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b), \boldsymbol{y})$. Otherwise, f outputs $\boldsymbol{r}$.

Let $C_{\mathtt{pr}}$ be a $(1,\epsilon)$-parity-tolerant circuit implementation of f. Now we build a circuit $C_{\mathtt{BCL}}$ which takes as input a 2-out-of-2 additive sharing of the input of $C_{\mathtt{pr}}$ and outputs a random 2-out-of-2 additive sharing of the output of $C_{\mathtt{pr}}$. The description of $C_{\mathtt{BCL}}$ appears in Figure 1. The high level idea is to follow the circuit $C_{\mathtt{pr}}$ and computes a random 2-out-of-2 additive sharing of each wire in $C_{\mathtt{pr}}$. To compute each gate g in $C_{\mathtt{pr}}$, the goal is to compute a random additive sharing of the output wire from the additive sharings of the two input wires. Notice that for all $a_1, a_2 \in \{0,1\}$, α_{a_1,a_2} can be computed from (u_1, u_2, u_3), β_{a_1,a_2} can be computed from (v_1, v_2), and

$$\begin{aligned}
\gamma_{a_1,a_2} &= (f_g(u_1 \oplus a_1, u_2 \oplus a_2) \oplus u_3) \cdot (v_1 \oplus a_1 \oplus 1)(v_2 \oplus a_2 \oplus 1) \\
&= (f_g(u_1 \oplus v_1, u_2 \oplus v_2) \oplus u_3) \cdot (v_1 \oplus a_1 \oplus 1)(v_2 \oplus a_2 \oplus 1) \\
&= v_3 \cdot (v_1 \oplus a_1 \oplus 1)(v_2 \oplus a_2 \oplus 1),
\end{aligned}$$

where the second equality follows from the fact that both sides are equal to $f_g(u_1 \oplus v_1, u_2 \oplus v_2) \oplus u_3$ if $v_1 = a_1$ and $v_2 = a_2$, and 0 otherwise. Thus, γ_{a_1,a_2} can be computed from (v_1, v_2, v_3). Later on we will use $C_{\mathtt{BCL}}$ to build a stateful LTC against t-BCL. Under t-BCL, having access to $\alpha_{a_1,a_2}, \beta_{a_1,a_2}, \gamma_{a_1,a_2}$ does not give any additional advantage to the adversary since all these values can be locally computed in a 2-party protocol where parties take (u_1, u_2, u_3) and (v_1, v_2, v_3) as input respectively.

Circuit $C_{\mathtt{BCL}}$

1. Input: The circuit $C_{\mathtt{BCL}}$ takes 2-out-of-2 additive sharings of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}\|b)$, $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$, $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape})$, $\boldsymbol{x}$ as input.

2. Evaluation: $C_{\mathtt{BCL}}$ follows the circuit $C_{\mathtt{pr}}$ and computes a random 2-out-of-2 additive sharing of each wire in $C_{\mathtt{pr}}$. For each gate g with input wires w_1, w_2 and output wire w_3, suppose $C_{\mathtt{BCL}}$ has computed additive sharings $(u_1, v_1), (u_2, v_2)$ of w_1, w_2 respectively. Let f_g be the function that maps (w_1, w_2) to w_3.

   $C_{\mathtt{BCL}}$ sets u_3 to be a random bit. Then for all $a_1, a_2 \in \{0,1\}$, $C_{\mathtt{BCL}}$ computes

   - $\alpha_{a_1,a_2} = f_g(u_1 \oplus a_1, u_2 \oplus a_2) \oplus u_3$,
   - $\beta_{a_1,a_2} = (v_1 \oplus a_1 \oplus 1)(v_2 \oplus a_2 \oplus 1)$,
   - $\gamma_{a_1,a_2} = \alpha_{a_1,a_2} \cdot \beta_{b_1,b_2}$.

   Then $C_{\mathtt{BCL}}$ computes $v_3 = \sum_{a_1,a_2 \in \{0,1\}} \gamma_{a_1,a_2}$.

3. Output: The circuit $C_{\mathtt{BCL}}$ outputs the obtained 2-out-of-2 additive sharings of the output wires of $C_{\mathtt{pr}}$.

Figure 1: Construction of $C_{\mathtt{BCL}}$ that computes additive sharings of the wire values in $C_{\mathtt{pr}}$

Now we are ready to present our construction of $\mathcal{T}_C$. $\mathcal{T}_C$ takes as input a stateful circuit C and outputs the following stateful circuit C'.

1. In each invocation, C' takes as input an internal state $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}\|b)$ and a public input $\boldsymbol{x}$.
2. C' samples a random string $\boldsymbol{r}_0$ of length $|\boldsymbol{x}|$ and computes $\boldsymbol{x} \oplus \boldsymbol{r}_0$.
3. C' samples random strings as $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape})$.
4. C' computes $C_{\mathtt{BCL}}$ with input $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}\|b), \mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}), \mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}), (\boldsymbol{r}_0, \boldsymbol{x} \oplus \boldsymbol{r}_0)$. The output is denoted by $\mathtt{Enc}_{\mathtt{BCL}}(\tilde{\boldsymbol{s}}\|b), (\boldsymbol{r}_1, \boldsymbol{y} \oplus \boldsymbol{r}_1)$.
5. C' computes $\boldsymbol{y} = \boldsymbol{r}_1 \oplus (\boldsymbol{y} \oplus \boldsymbol{r}_1)$. Then $\boldsymbol{y}$ is the public output in this invocation and $\mathtt{Enc}_{\mathtt{BCL}}(\tilde{\boldsymbol{s}}\|b)$ is the updated internal state.

4.3 Security Analysis

In this part, we prove the security of our construction according to Definition 8. Let C be the (unprotected) stateful circuit and $(C', \mathcal{T})$ be the stateful circuit obtained from our compiler. For all polynomial query bound q and adversary bound S, and for all $(S, q, \mathcal{L}^t_{\mathtt{BCL}})$-adversary circuit $\mathcal{A}$, we first give the construction of the simulator $\mathtt{Sim}$.

$\mathtt{Sim}$ starts with invoking $\mathcal{A}$. In the beginning, $\mathtt{Sim}$ sets $\boldsymbol{s}'_0 = \mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0}\|1)$, i.e., the initial state $\boldsymbol{s}_0$ is set to be $\mathbf{0}$ and the control bit is set to be 1. $\mathtt{Sim}$ does the following for the i-th invocation.

1. $\mathtt{Sim}$ takes as input the public input $\boldsymbol{x}_i$ and a leakage query $L_i \in \mathcal{L}^t_{\mathtt{BCL}}$.
2. $\mathtt{Sim}$ invokes $C[\boldsymbol{s}_{i-1}]$ with public input $\boldsymbol{x}_i$ and obtains the public output $\boldsymbol{y}_i$.
3. $\mathtt{Sim}$ sets $\boldsymbol{r}^{(i)} = \mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1)\|\boldsymbol{y}_i$ and $\mathtt{Rtape}^{(i)}$ to be all-0 string. Then $\mathtt{Sim}$ generates $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(i)})$.
4. $\mathtt{Sim}$ computes C' with input $\boldsymbol{s}'_{i-1}$ and $\boldsymbol{x}_i$. When C' samples random strings for $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(i)})$, use the encodings generated by $\mathtt{Sim}$ instead. Then $\mathtt{Sim}$ sets $\boldsymbol{s}'_i$ to be the updated internal state output by C'.
5. $\mathtt{Sim}$ computes the leakage result by applying L_i on the wire values in C'.
6. $\mathtt{Sim}$ outputs $\boldsymbol{y}_i$ and the leakage result to $\mathcal{A}$.

We argue the security by hybrid arguments. Recall that q is the number of invocations made by the adversary $\mathcal{A}$.

$\mathbf{Hybrid}_0$: This hybrid is the real world execution. The output is the transcript of the real-world experiment, which includes the view of $\mathcal{A}$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^q$.

$\mathbf{Hybrid}_1$: In this hybrid, we change the way of generating $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(i)})$ in each invocation. Instead of directly picking random strings as $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(i)})$, we first randomly sample $\boldsymbol{r}^{(i)}$ and $\mathtt{Rtape}^{(i)}$ and then compute random encodings of $\boldsymbol{r}^{(i)}$ and $\mathtt{Rtape}^{(i)}$. By the property of $\mathtt{Enc}_{\mathtt{BCL}}$, $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_0$ are identically distributed.

$\mathbf{Hybrid}_{2,0}$: In the first invocation, we make the following changes:

- We first randomly sample $\mathtt{Rtape}^{(1)} = (\mathtt{Rtape}^{(1)}_1, \mathtt{Rtape}^{(1)}_2)$. Then we compute $C[\boldsymbol{s}_0](\boldsymbol{x}_1)$ with random tape $\mathtt{Rtape}^{(1)}_1$ and obtain $\boldsymbol{s}_1, \boldsymbol{y}_1$. We compute $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_1\|0)$ with random tape $\mathtt{Rtape}^{(1)}_2$.
- We set the initial encoded state $\boldsymbol{s}'_0$ to be $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0}\|1)$. Then we replace $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(1)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(1)})$ by $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_1\|0)\|\boldsymbol{y}_1)$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0})$ respectively.

We argue that $\mathbf{Hybrid}_{2,0}$ and $\mathbf{Hybrid}_1$ are statistically close.

Suppose $\boldsymbol{r}_o$ is the random string that is used to mask the output of $C_{\mathtt{BCL}}$ in the first invocation. We prove a stronger statement: $\mathbf{Hybrid}_{2,0}$ and $\mathbf{Hybrid}_1$ are statistically close given the following values.

- The initial unprotected state $\boldsymbol{s}_0$, public input $\boldsymbol{x}_1$, random tape $\mathtt{Rtape}^{(1)}$. Then $\boldsymbol{s}_1, \boldsymbol{y}_1, \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_1\|0)$ are also fixed.
- The random string $\boldsymbol{r}_o$. Then the output of $C_{\mathtt{BCL}}$ (including $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_1\|0)$ and the additive sharing of $\boldsymbol{y}_1$) is also fixed.

Since future invocations only depend on $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_1\|0)$, it is sufficient to focus on the public output and the leakage result in the first invocation.

First note that by construction the output of $C_{\mathtt{BCL}}$ in both hybrids are identical. This implies that the public output $\boldsymbol{y}_1$ in both invocations are identical. Given $\boldsymbol{x}_1, \boldsymbol{y}_1$ and the output of $C_{\mathtt{BCL}}$, we can reduce L to a t-BCL query L' that is applied on the wires of $C_{\mathtt{BCL}}$ excluding its output wires. By construction, $C_{\mathtt{BCL}}$ computes a 2-out-of-2 additive sharing of each wire in $C_{\mathtt{pr}}$, and all other wires in $C_{\mathtt{BCL}}$ only depends on one part of $C_{\mathtt{BCL}}$. Without loss of generality, we assume that L' is applied on the 2-out-of-2 additive sharings of the wires in $C_{\mathtt{pr}}$ (except its output wires).

Note that in $\mathbf{Hybrid}_1$, the effective input of $C_{\mathtt{pr}}$ is $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_0\|0), \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r}^{(1)}), \mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape}^{(1)}), \boldsymbol{x}_1$. In $\mathbf{Hybrid}_{2,0}$, the effective input of $C_{\mathtt{pr}}$ is $\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1), \mathtt{Enc}_{\mathtt{pr}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_1\|0)\|\boldsymbol{y}_1), \mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}), \boldsymbol{x}_1$. Since $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$ is a parity-resilient encoding scheme, the inputs of $C_{\mathtt{pr}}$ in both two hybrids are indistinguishable by parity with statistical distance ϵ. Since the outputs of $C_{\mathtt{pr}}$ in both hybrids are identical, by the property of parity-tolerant circuits, the transcripts of $C_{\mathtt{pr}}$ in both hybrids are indistinguishable by parity with statistical distance 3ϵ (which can be proved by a standard hybrid argument). Since $C_{\mathtt{BCL}}$ computes a random 2-out-of-2 additive sharing for each inner wire in $C_{\mathtt{pr}}$, and the input of $C_{\mathtt{BCL}}$ is a random 2-out-of-2 additive sharing of the input of $C_{\mathtt{pr}}$, by Lemma 2, the leakage result of L' in both hybrids are statistically close with error $3 \cdot 2^t\epsilon$.

Thus, $\mathbf{Hybrid}_{2,0}$ and $\mathbf{Hybrid}_1$ are statistically close. In the following, we define hybrids for $i \in \{1, \ldots, q-1\}$.

$\mathbf{Hybrid}_{2,i}$: In this hybrid, we make the following changes to the i-th invocation and the $(i+1)$-th invocation:

- In the i-th execution, we first randomly sample $\mathtt{Rtape}^{(i)} = (\mathtt{Rtape}_1^{(i)}, \mathtt{Rtape}_2^{(i)})$ and compute $\boldsymbol{s}_i, \boldsymbol{y}_i, \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i\|0)$ as that in $\mathbf{Hybrid}_{2,i-1}$.
- Recall that in $\mathbf{Hybrid}_{2,i-1}$, $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i)})$ has been replaced by $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_i\|0)\|\boldsymbol{y}_i)$. In this hybrid, we further change it to be $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1)\|\boldsymbol{y}_i)$. In this way, the output of C' in the i-th invocation becomes $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0}\|1), \boldsymbol{y}_i$. The leakage result is computed by applying the leakage function on the wire values in C'. Note that $\mathtt{Rtape}_2^{(i)}$ is not used in $\mathbf{Hybrid}_{2,i}$.
- In the $(i+1)$-th execution, we first randomly sample $\mathtt{Rtape}^{(i+1)} = (\mathtt{Rtape}_1^{(i+1)}, \mathtt{Rtape}_2^{(i+1)})$. Then we compute $C[\boldsymbol{s}_i](\boldsymbol{x}_{i+1})$ with random tape $\mathtt{Rtape}_1^{(i+1)}$ and obtain $\boldsymbol{s}_{i+1}, \boldsymbol{y}_{i+1}$. We compute $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i+1}\|0)$ with random tape $\mathtt{Rtape}_2^{(i+1)}$.
- We replace $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(i+1)})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Rtape}^{(i+1)})$ by $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i+1}\|0)\|\boldsymbol{y}_{i+1})$ and $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0})$ respectively.

We argue that $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ are statistically close.

We first construct an adversary $\mathcal{B}$ such that if the adversary $\mathcal{A}$ can distinguish $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ with advantage η. Then $\mathcal{B}$ can distinguish these two hybrids with advantage $\eta/2^{2t}$. The construction of $\mathcal{B}$ is as follows.

1. $\mathcal{B}$ starts with invoking $\mathcal{A}$. In the first $i-1$ invocations, $\mathcal{B}$ behaves the same as $\mathcal{A}$.
2. In the i-th invocation, $\mathcal{B}$ receives $\boldsymbol{x}_i, L_i$ from $\mathcal{A}$. Then $\mathcal{B}$ invokes C' with input $\boldsymbol{x}_i$ and learns $\boldsymbol{y}_i$. $\mathcal{B}$ randomly samples $\ell_i' \in \{0,1\}^{2t}$ as the output of L_i and returns $\boldsymbol{y}_i, \ell_i'$ to $\mathcal{A}$.
3. In the $(i+1)$-th invocation, $\mathcal{B}$ receives $\boldsymbol{x}_{i+1}, L_{i+1}$ from $\mathcal{A}$. Then $\mathcal{B}$ invokes C' with input $\boldsymbol{x}_{i+1}$ and applies L_i on wire values in the i-th invocation and L_{i+1} on wire values in the $(i+1)$-th invocation. Then $\mathcal{B}$ receives $\boldsymbol{y}_{i+1}, \ell_i, \ell_{i+1}$. If $\ell_i = \ell_i'$, $\mathcal{B}$ returns $\boldsymbol{y}_{i+1}, \ell_{i+1}$ to $\mathcal{A}$. Otherwise $\mathcal{B}$ aborts.
4. If $\mathcal{B}$ does not abort, $\mathcal{B}$ follows $\mathcal{A}$ in the rest of executions.

The only difference between $\mathcal{A}$ and $\mathcal{B}$ is that $\mathcal{B}$ makes a random guess as the leakage in the i-th invocation and later verifies his guess in the $(i+1)$-th invocation. If the guess is correct, then the

output of $\mathcal{B}$ is identically distributed to that of $\mathcal{A}$. If the guess is incorrect, then the output of $\mathcal{B}$ would be $\texttt{fail}$. Note that $\mathcal{B}$ outputs $\texttt{fail}$ with probability $1 - 1/2^{2t}$ and it is independent of the leakage results. Thus, the advantage of $\mathcal{B}$ of distinguishing $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ is $\eta/2^{2t}$.

Note that, $\mathcal{B}$ makes two non-adaptive leakage queries for the i-th execution and the $(i+1)$-th execution. And it is sufficient to argue that any adversary $\mathcal{B}$ cannot distinguish $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ with non-negligible probability.

First note that the first $(i-1)$ invocations are identical in both $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$. For the i-th invocation, by construction, the public outputs $\boldsymbol{y}_i$ in both hybrids are identically distributed. Thus the leakage queries L_i, L_{i+1} made by $\mathcal{B}$ are identically distributed in both hybrids.

Suppose $\boldsymbol{r}_o$ is the random string that is used to mask the output of $C_{\texttt{BCL}}$ in the $(i+1)$-th invocation. We prove a stronger statement: $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ are statistically close against $\mathcal{B}$ given the following values.

- The initial unprotected state $\boldsymbol{s}_0$, $\{\texttt{Rtape}^{(j)}\}_{j\le i-1}$, $\texttt{Rtape}_1^{(i)}$, $\texttt{Rtape}^{(i+1)}$, $\{\boldsymbol{x}_j\}_{j\le i+1}$. These also fix $\boldsymbol{s}_i$, $\boldsymbol{s}_{i+1}$, $\{\boldsymbol{y}_j\}_{j\le i+1}$ and $\texttt{Enc}_{\texttt{pr}}(\boldsymbol{s}_{i+1}\|0)$.
- The wire values in the first $(i-1)$ invocations. These also fix the input state of C' in the i-th invocation $\boldsymbol{s}'_{i-1}$, which is an encoding of $\mathbf{0}\|1$, denoted by $\widetilde{\texttt{Enc}}_{\texttt{BCL}}(\mathbf{0}\|1)$.
- The random string $\boldsymbol{r}_o$. Then the output of $C_{\texttt{BCL}}$ (including $\texttt{Enc}_{\texttt{BCL}}(\boldsymbol{s}_{i+1}\|0)$ and the additive sharing of $\boldsymbol{y}_{i+1}$) is also fixed.

Since future invocations only depend on $\texttt{Enc}_{\texttt{BCL}}(\boldsymbol{s}_{i+1}\|0)$, it is sufficient to focus on the leakage results in the i-th invocation and the $(i+1)$-th invocation.

Given $\boldsymbol{x}_i, \boldsymbol{y}_i$ and the input internal state of $C_{\texttt{BCL}}$ in the i-th invocation, we can reduce L_i to a t-BCL query L'_i that is applied on the wires of $C_{\texttt{BCL}}$ excluding its input internal state in the i-th invocation. Given $\boldsymbol{x}_{i+1}, \boldsymbol{y}_{i+1}$ and the output of $C_{\texttt{BCL}}$ in the $(i+1)$-th invocation, we can reduce L_{i+1} to a t-BCL query L'_{i+1} that is applied on the wires of $C_{\texttt{BCL}}$ excluding its output wires in the $(i+1)$-th invocation. Note that $L' = (L'_i, L'_{i+1})$ is a single $2t$-BCL query. By construction, $C_{\texttt{BCL}}$ computes a 2-out-of-2 additive sharing of each wire in $C_{\texttt{pr}}$ and all other wires in $C_{\texttt{BCL}}$ only depends on one part of $C_{\texttt{BCL}}$. Without loss of generality, we assume that L' is applied on the 2-out-of-2 additive sharings of the wires in $C_{\texttt{pr}}$ in the i-th invocation and the $(i+1)$-th invocation.

We note that the circuit $C_{\texttt{pr}}$ in the i-th invocation and the $(i+1)$-th invocation can be viewed as a single circuit, denoted by $C_{\texttt{pr}}[i, i+1]$. In $\mathbf{Hybrid}_{2,i-1}$, the effective input of $C_{\texttt{pr}}[i, i+1]$ is the following.

- $\widetilde{\texttt{Enc}}_{\texttt{pr}}(\mathbf{0}\|1)$, $\texttt{Enc}_{\texttt{pr}}(\texttt{Enc}_{\texttt{pr}}(\boldsymbol{s}_i\|0)\|\boldsymbol{y}_i)$, $\texttt{Enc}_{\texttt{pr}}(\mathbf{0})$, $\boldsymbol{x}_i$ for $C_{\texttt{pr}}$ in the i-th invocation. Here $\widetilde{\texttt{Enc}}_{\texttt{pr}}(\mathbf{0}\|1)$, $\boldsymbol{y}_i, \boldsymbol{x}_i$ are fixed.
- $\texttt{Enc}_{\texttt{pr}}(\boldsymbol{r}^{(i+1)})$, $\texttt{Enc}_{\texttt{pr}}(\texttt{Rtape}^{(i+1)})$, $\boldsymbol{x}_{i+1}$ for $C_{\texttt{pr}}$ in the $(i+1)$-th invocation. Here $\boldsymbol{x}_{i+1}$ is fixed.

The effective output of $C_{\texttt{pr}}[i, i+1]$ is the following.

- $\texttt{Enc}_{\texttt{pr}}(\boldsymbol{s}_i\|0)$, $\boldsymbol{y}_i$ for $C_{\texttt{pr}}$ in the i-th invocation. Here $\boldsymbol{y}_i$ is fixed.
- $\texttt{Enc}_{\texttt{pr}}(\boldsymbol{s}_{i+1}\|0)$, $\boldsymbol{y}_{i+1}$ for $C_{\texttt{pr}}$ in the $(i+1)$-th invocation. Here both $\texttt{Enc}_{\texttt{pr}}(\boldsymbol{s}_{i+1}\|0)$ and $\boldsymbol{y}_{i+1}$ are fixed.

Similarly, In $\mathbf{Hybrid}_{2,i}$, the effective input of $C_{\texttt{pr}}[i, i+1]$ is the following.

- $\widetilde{\mathtt{Enc}_{\mathtt{pr}}}(\mathbf{0}\|1)$, $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1)\|\boldsymbol{y}_i)$, $\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0})$, $\boldsymbol{x}_i$ for $C_{\mathtt{pr}}$ in the i-th invocation. Here $\widetilde{\mathtt{Enc}_{\mathtt{pr}}}(\mathbf{0}\|1)$, $\boldsymbol{y}_i, \boldsymbol{x}_i$ are fixed.
- $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i+1}\|0)\|\boldsymbol{y}_{i+1})$, $\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0})$, $\boldsymbol{x}_{i+1}$ for $C_{\mathtt{pr}}$ in the $(i+1)$-th invocation. Here $\boldsymbol{x}_{i+1}$ is fixed.

The effective output of $C_{\mathtt{pr}}[i, i+1]$ is the following.

- $\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1)$, $\boldsymbol{y}_i$ for $C_{\mathtt{pr}}$ in the i-th invocation. Here $\boldsymbol{y}_i$ is fixed.
- $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i+1}\|0)$, $\boldsymbol{y}_{i+1}$ for $C_{\mathtt{pr}}$ in the $(i+1)$-th invocation. Here both $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_{i+1}\|0)$ and $\boldsymbol{y}_{i+1}$ are fixed.

We first show that $C_{\mathtt{pr}}[i, i+1]$ satisfies a relaxed variant of parity tolerance: A parity query on wires in $C_{\mathtt{pr}}[i, i+1]$ can be simulated by two parity queries on input and output of $C_{\mathtt{pr}}[i, i+1]$. This is because a parity query on wires in $C_{\mathtt{pr}}[i, i+1]$ can be computed by two parity queries on wires in $C_{\mathtt{pr}}$ in the i-th invocation and the $(i+1)$-th invocation respectively. Then relying on the parity tolerance of $C_{\mathtt{pr}}$, these two parity queries can be simulated by two parity queries on the input and output of $C_{\mathtt{pr}}$ in the i-th invocation and the $(i+1)$-th invocation respectively, which are just the input and output of $C_{\mathtt{pr}}[i, i+1]$.

Note that the inputs and outputs of $C_{\mathtt{pr}}[i, i+1]$ in both hybrids are indistinguishable by parity with statistical distance ϵ. According to the XOR lemma (Lemma 3 stated below), they are indistinguishable by two parities with statistical distance 2ϵ.

Lemma 3 (XOR Lemma [Gol11])**.** *Let $\boldsymbol{X} = (X_1, \ldots, X_n)$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$ be two random variables over $\{0,1\}^n$. If for all $S \subset \{1, \ldots, n\}$, the statistical distance between $\oplus_{i \in S} X_i$ and $\oplus_{i \in S} Y_i$ is at most ϵ, then the statistical distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$ is at most $2^{n/2} \cdot \epsilon$.*

By the property $C_{\mathtt{pr}}[i, i+1]$, the transcripts of $C_{\mathtt{pr}}[i, i+1]$ in both hybrids are indistinguishable by parity with statistical distance 8ϵ (which can be proved by a standard hybrid argument). Since $C_{\mathtt{BCL}}[i, i+1]$ computes a random 2-out-of-2 additive sharing for each inner wire and output wire in $C_{\mathtt{pr}}[i, i+1]$ (except $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_{i+1}\|0)$ and the additive sharing of $\boldsymbol{y}_{i+1}$, which are fixed), and the input of $C_{\mathtt{BCL}}[i, i+1]$ (except $\widetilde{\mathtt{Enc}_{\mathtt{BCL}}}(\mathbf{0}\|1)$, which is fixed) is a random 2-out-of-2 additive sharing of the input of $C_{\mathtt{pr}}[i, i+1]$ (except $\widetilde{\mathtt{Enc}_{\mathtt{pr}}}(\mathbf{0}\|1)$), by lemma 2, the leakage result of any $2t$-BCL query in both hybrids are statistically close with error $8 \cdot 2^{2t}\epsilon$. Thus, $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ are statistically close with statistical error $8 \cdot 2^{2t}\epsilon$ against $\mathcal{B}$, which also implies that $\mathbf{Hybrid}_{2,i}$ and $\mathbf{Hybrid}_{2,i-1}$ are statistically close with statistical error $8 \cdot 2^{4t}\epsilon$ against $\mathcal{A}$.

$\mathbf{Hybrid}_{2,q}$: In this hybrid, we make the following changes to the q-th invocation:

- In the q-th execution, we first randomly sample $\mathtt{Rtape}^{(q)} = (\mathtt{Rtape}_1^{(q)}, \mathtt{Rtape}_2^{(q)})$ and compute $\boldsymbol{s}_q, \boldsymbol{y}_q, \mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_q\|0)$ as that in $\mathbf{Hybrid}_{2,q-1}$.
- Recall that in $\mathbf{Hybrid}_{2,q-1}$, $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{r}^{(q)})$ has been replaced by $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_q\|0)\|\boldsymbol{y}_q)$. In this hybrid, we further change it to be $\mathtt{Enc}_{\mathtt{BCL}}(\mathtt{Enc}_{\mathtt{pr}}(\mathbf{0}\|1)\|\boldsymbol{y}_q)$. In this way, the output of C' in the q-th invocation becomes $\mathtt{Enc}_{\mathtt{BCL}}(\mathbf{0}\|1), \boldsymbol{y}_q$. The leakage result is computed by applying the leakage function on the wire values in C'. Note that $\mathtt{Rtape}_2^{(q)}$ is not used in $\mathbf{Hybrid}_{2,q}$.

Following a similar argument, $\mathbf{Hybrid}_{2,q}$ and $\mathbf{Hybrid}_{2,q-1}$ are statistically close.

$\mathbf{Hybrid}_3$: In this hybrid, we switch to use $\mathtt{Sim}$ to simulate the public output and leakage result in each invocation. Note that the only difference between $\mathbf{Hybrid}_{2,q}$ and $\mathbf{Hybrid}_3$ is that in $\mathbf{Hybrid}_{2,q}$, we faithfully compute $C[\boldsymbol{s}_{i-1}]$ with input $\boldsymbol{x}_i$ and obtain $\boldsymbol{s}_i, \boldsymbol{y}_i$, whereas in $\mathbf{Hybrid}_3$, the computation is carried out by the ideal functionality. Since the simulation only requires $\boldsymbol{y}_i$ and is independent of $\boldsymbol{s}_{i-1}, \boldsymbol{s}_i$, $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_{2,q}$ are identically distributed. Note that $\mathbf{Hybrid}_3$ is the ideal world, and $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_0$ are statistically close with error bounded by $q \cdot 8 \cdot 2^{4t}\epsilon$. For a target statistical error $\epsilon^\star$, when instantiating $(\mathtt{Enc}_{\mathtt{pr}}, \mathtt{Dec}_{\mathtt{pr}})$ and the compiler for $(1, \epsilon)$-parity-tolerant circuits, we may set $\epsilon = \epsilon^\star/(8 \cdot 2^{4t})$.

4.4 Application: Protecting Computation Against Malware

In this section, we discuss the application of our main result to obtain a stateful variant of the two-party protocol of Goyal et al. [GIM+16]. As discussed in the introduction, this is motivated by the goal of protecting computations in a scenario where *all* computers may be corrupted by malware, but the malware has a bounded communication rate.

Concretely, we want two parties to jointly emulate a stateful circuit C, given a leakage-resilient secret sharing of the initial secret state $\boldsymbol{s}_0$. In the i-th epoch, the parties may obtain an external public input $\boldsymbol{x}_i$ and compute $(\boldsymbol{s}_i, \boldsymbol{y}_i) \leftarrow C[\boldsymbol{s}_{i-1}](\boldsymbol{x}_i)$ where $\boldsymbol{s}_i$ is the updated secret state and $\boldsymbol{y}_i$ is the public output.

For a leakage bound parameter t and a query bound parameter q, a (t, q)-bounded adversary $\mathcal{A}$ is a tuple of three PPT algorithms $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}^\star)$ with the following syntax:

- The adversary $\mathcal{A}$ may invoke the 2-party protocol between P_0 and P_1 for at most q epochs.
- For both $i \in \{0, 1\}$, $\mathcal{A}_i$ can read the entire view of P_i at any time but not modify the actions of P_i. In addition, $\mathcal{A}_i$ has access to a local memory of size t, which can be used, for example, to store a state of size t for the next epoch. In each epoch, $\mathcal{A}_i$ can perform any polynomial-time computation (regardless of space) and send at most t bits to $\mathcal{A}_{1-i}$.
- $\mathcal{A}^\star$ can see the communication between $\mathcal{A}_0$ and $\mathcal{A}_1$ and that between P_0 and P_1 but has no access to the memory or the view of $\mathcal{A}_0, \mathcal{A}_1, P_0, P_1$. In each epoch, $\mathcal{A}^\star$ may decide the public input depending on its view and learn the public output.

Real-world execution. In the real world, let I be an input encoder that takes as input an initial state $\boldsymbol{s}_0$ and outputs $(\tilde{\boldsymbol{s}}_{0,0}, \tilde{\boldsymbol{s}}_{0,1})$. The 2-party protocol Π starts with the two parties holding $(\tilde{\boldsymbol{s}}_{0,0}, \tilde{\boldsymbol{s}}_{0,1})$ respectively. The transcript of the real-world experiment is the view of $\mathcal{A}^\star$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^q$.

Ideal-world execution. In the ideal world, we consider a q-bounded ideal adversary with oracle access to both the real world adversary $\mathcal{A}$ and $C[\boldsymbol{s}_0]$. $\mathtt{Sim}$ may invoke C at most q times. For each $i \in \{1, \ldots, q\}$, in the i-th invocation, $\mathtt{Sim}$ may adaptively choose an input $\boldsymbol{x}_i$ based on the observed output values in previous invocations and learns the output $\boldsymbol{y}_i$ corresponding to the chosen input. The transcript of the ideal-world experiment is the output of $\mathtt{Sim}$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^q$.

Definition 9 (Stateful t-BCL Resilient Protocol)**.** *Let I be an input encoder and Π be a 2-party protocol. For a stateful circuit C with initial state $\boldsymbol{s}_0$, we say (I, Π) is a t-bounded-communication leakage resilient protocol for $C[\boldsymbol{s}_0]$ if for all polynomial $q(\kappa)$ and for all (t, q)-bounded P.P.T. adversary $\mathcal{A}$, there exists a q-bounded simulator $\mathtt{Sim}$ such that the transcript in the real world is computationally indistinguishable from the transcript in the ideal world.*

4.4.1 Oblivious Transfer with Joint Simulation Security

To build a stateful t-BCL resilient protocol, our idea is to transform our stateful LRC for $2t$-BCL to a 2-party protocol. We will explain later why we need to use a stateful LRC for $2t$-BCL.

Recall that in our construction, we use a circuit $C_{\mathtt{BCL}}$ to compute 2-out-of-2 additive sharings of the wire values in $C_{\mathtt{pr}}$. To transform it to a 2-party protocol, we simply let each party hold one share for each additive sharing. One issue is that for each gate in $C_{\mathtt{pr}}$, when computing from additive sharings of the input wires to an additive sharing of the output wire, $C_{\mathtt{BCL}}$ would compute $\gamma_{a_1,a_2} = \alpha_{a_1,a_2} \cdot \beta_{a_1,a_2}$. While P_0 can compute α_{a_1,a_2} locally and P_1 can locally compute β_{a_1,a_2} locally, we cannot directly compute γ_{a_1,a_2} without revealing α_{a_1,a_2} to P_1 or revealing β_{a_1,a_2} to P_0. To solve this issue, we will use an oblivious transfer (OT) protocol to let P_1 obtain γ_{a_1,a_2}.

We borrow the construction of a $(2,1)$-OT with joint simulation security from [GIM+16].

Definition 10 (Joint Simulation Security [GIM+16])**.** *Let $f : X_1 \times X_2 \to Y_1 \times Y_2$ be a deterministic function such that $f(x_1, x_2) = (y_1, y_2)$. We say that a protocol $\pi(\cdot,\cdot)$ realizes f with joint simulation security if the following conditions hold:*

- *Correctness: For every inputs (x_1, x_2), we have $\Pr[\pi(x_1, x_2)$ outputs $f(x_1, x_2)] = 1$.*
- *Joint Simulation Security: There exists a pair of simulators $(\mathtt{Sim}_1, \mathtt{Sim}_2)$, and a randomness distribution T shared between the simulators such that for all inputs (x_1, x_2), $\mathtt{view}(x_1, x_2)$ and $(\mathtt{Sim}_1(x_1, y_1, T), \mathtt{Sim}_2(x_2, y_2, T))$ are computationally indistinguishable.*

Lemma 4 (Joint-Simulation OT from Standard Assumptions [GIM+16])**.** *Assuming either the hardness of factoring Blum integers or the Decisional Diffie-Hellman (DDH) assumption, there exists a $(2,1)$-OT protocol with joint simulation security.*

We will rely on the following property of OT with joint simulation security.

Lemma 5. *Suppose π is a 2-party computation protocol that realizes $(2,1)$-OT with joint simulation security. Then there exists a tuple of simulators $(\mathtt{Sim}_1, \mathtt{Sim}_2, \mathtt{Sim}_3)$, and a randomness distribution T shared among the simulators, such that for all inputs $x_1 = (m_0, m_1)$ and $x_2 = b$, the following two distributions are computationally indistinguishable:*

$$\mathtt{view}((m_0, m_1), b) \| \tau((m_0, m_1), b) \approx (\mathtt{Sim}_1((m_0, m_1), \bot, T), \mathtt{Sim}_2(b, m_b, T)) \| \mathtt{Sim}_3(T),$$

where $\tau((m_0, m_1), b)$ denotes the transcript of π.

Proof. Let $\mathtt{Sim}_1, \mathtt{Sim}_2$ be the simulators guaranteed by the joint simulation security of π. Without loss of generality, we assume that $\mathtt{Sim}_1$ and $\mathtt{Sim}_2$ are deterministic. This is because if $\mathtt{Sim}_1, \mathtt{Sim}_2$ are randomized, we may append random bits after T and let $\mathtt{Sim}_1$ and $\mathtt{Sim}_2$ use independent random bits from T as their random tapes.

Note that $\mathtt{Sim}_1$ is supposed to simulate the view of P_1 and $\mathtt{Sim}_2$ is supposed to simulate the view of P_2. Also note that the transcript of π is a part of the view of either party.

The construction of $\mathtt{Sim}_3$ is as follows: it runs $\mathtt{Sim}_1$ by replacing (m_0, m_1) by all-0 strings and outputs the simulated transcript obtained from the simulated view of P_1.

To prove that $(\mathtt{Sim}_1, \mathtt{Sim}_2, \mathtt{Sim}_3)$ satisfies the requirement, we build two algorithms A and B as follows:

- A takes as input $((m_0, m_1), \perp, T)$, runs $\mathtt{Sim}_1((m_0, m_1), \perp, T)$, and outputs the simulated transcript obtained from the simulated view of P_1.
- B takes as input (b, m_b, T), runs $\mathtt{Sim}_2(b, m_b, T)$, and outputs the simulated transcript obtained from the simulated view of P_2.

Then $\mathtt{Sim}_3(T) = A((0,0), \perp, T)$. We prove that for all m_0, m_1, with overwhelming probability over the choice of T, $A((m_0, m_1), \perp, T) = A((0,0), \perp, T)$.

According to the joint simulation security, for all $((m_0, m_1), b)$ we have

$$\mathtt{view}((m_0, m_1), b) \approx (\mathtt{Sim}_1((m_0, m_1), \perp, T), \mathtt{Sim}_2(b, m_b, T)).$$

Since $A((m_0, m_1), \perp, T)$ is the simulated transcript obtained from $\mathtt{Sim}_1((m_0, m_1), \perp, T)$ and $B(b, m_b, T)$ is the simulated transcript obtained from $\mathtt{Sim}_2(b, m_b, T)$, with overwhelming probability over the choice of T, $A((m_0, m_1), \perp, T) = B(b, m_b, T)$.

Then we have that

- For $((m_0, m_1), 0)$, with overwhelming probability over the choice of T, $A((m_0, m_1), \perp, T) = B(0, m_0, T)$.
- For $((m_0, 0), 0)$, with overwhelming probability over the choice of T, $A((m_0, 0), \perp, T) = B(0, m_0, T)$.
- For $((m_0, 0), 1)$, with overwhelming probability over the choice of T, $A((m_0, 0), \perp, T) = B(1, 0, T)$.
- For $((0,0), 1)$, with overwhelming probability over the choice of T, $A((0,0), \perp, T) = B(1, 0, T)$.

Thus, with overwhelming probability over the choice of T, $A((m_0, m_1), \perp, T) = A((0,0), \perp, T)$.

Finally, notice that

$$\begin{aligned}
& \mathtt{view}((m_0, m_1), b) \| \tau((m_0, m_1), b) \\
\approx\ & (\mathtt{Sim}_1((m_0, m_1), \perp, T), \mathtt{Sim}_2(b, m_b, T)) \| A((m_0, m_1), \perp, T) \\
\approx\ & (\mathtt{Sim}_1((m_0, m_1), \perp, T), \mathtt{Sim}_2(b, m_b, T)) \| A((0,0), \perp, T) \\
=\ & (\mathtt{Sim}_1((m_0, m_1), \perp, T), \mathtt{Sim}_2(b, m_b, T)) \| \mathtt{Sim}_3(T),
\end{aligned}$$

and the lemma follows. □

4.4.2 Construction of Stateful BCL-Resilient Two-Party Protocol

We are now ready to present our construction. Recall that in our construction of stateful LRCs for $2t$-BCL, the circuit wires are partitioned into two parts. At a high level, we will build a protocol such that the view of each party corresponds to one of the parts. We will show that a (t, q)-bounded adversary can be reduced to an adversary that makes q invocations to our stateful LRC for $2t$-BCL.

Theorem 3 (BCL-Resilient Two-Party Computation)**.** *Assume either the hardness of factoring Blum integers or the Decisional Diffie-Hellman (DDH) assumption. For all $t \in \mathbb{N}$, there exists an input encoder I and a polynomial-time compiler $\mathcal{T}_\pi$ which takes as input $(1^\kappa, t, C)$ and outputs a 2-party protocol Π such that for all stateful circuit C with initial state $\boldsymbol{s}_0$, $\Pi \leftarrow \mathcal{T}_\pi(1^\kappa, t, C)$ satisfies that (I, Π) is a t-bounded-communication leakage resilient protocol for $C[\boldsymbol{s}_0]$.*

Proof. We start by describing the construction, and then prove its security.

Construction. Suppose ρ is a $(2,1)$-OT protocol with joint simulation security. Let $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ be our construction of the compiler for stateful LRCs for $2t$-BCL.

The input encoder I takes as input $\boldsymbol{s}_0$ and runs $\mathcal{T}_s(\boldsymbol{s}_0)$, which outputs $\mathtt{Enc}_{\mathtt{BCL}}(\boldsymbol{s}_0\|0)$. Note that this is a 2-out-of-2 additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}_0\|0)$.

The compiler $\mathcal{T}_\pi$ takes as input C and outputs the following 2-party protocol.

1. In the beginning of each epoch, both parties hold an additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}\|b)$ and receive a public input $\boldsymbol{x}$.
2. P_0 samples a random string $\boldsymbol{r}_0$ and sends $\boldsymbol{r}_0$ to P_1. Then P_1 computes $\boldsymbol{x} \oplus \boldsymbol{r}_0$.
3. Both parties sample two random strings and view them as additive sharings of $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$ and $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape})$.
4. Both parties compute an additive sharing for each wire in $C_{\mathtt{pr}}$ as follows.
   (a) In the beginning, both parties hold additive sharings of the input of $C_{\mathtt{pr}}$, which are $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{s}\|b)$, $\mathtt{Enc}_{\mathtt{pr}}(\boldsymbol{r})$, $\mathtt{Enc}_{\mathtt{pr}}(\mathtt{Rtape})$, $\boldsymbol{x}$.
   (b) For each gate g in $C_{\mathtt{pr}}$ with input wires w_1, w_2 and output wire w_3, let f_g be the function that maps (w_1, w_2) to w_3. Suppose both parties hold additive sharings $(u_1, v_1), (u_2, v_2)$ of w_1, w_2 respectively. P_0 samples a random bit as u_3. Then for all $a_1, a_2 \in \{0,1\}$,
   - P_0 computes $\alpha_{a_1,a_2} = f_g(u_1 \oplus a_1, u_2 \oplus a_2) \oplus u_3$,
   - P_1 computes $\beta_{a_1,a_2} = (v_1 \oplus a_1 \oplus 1)(v_2 \oplus a_2 \oplus 1)$,
   - P_0 and P_1 invoke ρ with input $(0, \alpha_{a_1,a_2})$ and β_{b_1,b_2} respectively to let P_1 learn $\gamma_{a_1,a_2} = \alpha_{a_1,a_2} \cdot \beta_{b_1,b_2}$.

   Finally, P_1 computes $v_3 = \sum_{a_1,a_2 \in \{0,1\}} \gamma_{a_1,a_2}$.
5. After evaluating $C_{\mathtt{pr}}$, both parties together hold an additive sharing of $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b)$ and an additive sharing of $\boldsymbol{y}$, denoted by $(\boldsymbol{r}_1, \boldsymbol{y} \oplus \boldsymbol{r}_1)$. Both parties exchange their shares of the additive sharing of $\boldsymbol{y}$ and compute the public output $\boldsymbol{y}$.
6. Both parties erase all data except the public output $\boldsymbol{y}$ and their shares of $\mathtt{Enc}_{\mathtt{pr}}(\tilde{\boldsymbol{s}}\|b)$.

Security Analysis. To show the security of our construction, let C' be the stateful $2t$-BCL LRC of C obtained by applying $\mathcal{T}_C$ on C. We first note that the above construction can be transformed to a stateful circuit C'' by writing each party's computation as a circuit. In particular, the only difference between C'' and C' is that in each invocation, for each gate, we have additional wires corresponding to the views of the protocol ρ.

In the following, we show that for every (t,q)-bounded P.P.T. adversary $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}^\star)$ attacking Π, there is an $2t$-BCL adversary $\mathcal{B}$ attacking C' such that the transcript in the real-world execution can be simulated by $\mathcal{B}$. Then, following from the fact that that C' is a stateful $2t$-BCL LRC, the theorem follows. Consider the following hybrids.

Hybrid$_0$: In this hybrid, we consider the real-world execution and the output is the transcript of the real-world execution, i.e., the view of $\mathcal{A}^\star$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^q$.

Hybrid$_1$: In this hybrid, we modify the views of P_0 and P_1 as well as the transcript of each call of ρ in Π. Let $(\mathtt{Sim}_1, \mathtt{Sim}_2, \mathtt{Sim}_3)$ be the tuple of simulators and T be the randomness distribution guaranteed by Lemma 5. For each call of ρ with input $((m_0, m_1), b)$, we replace

$\mathtt{view}((m_0,m_1),b)\|\tau((m_0,m_1),b)$ by $(\mathtt{Sim}_1((m_0,m_1),\perp,T),\mathtt{Sim}_2(b,m_b,T))\|\mathtt{Sim}_3(T)$. By Lemma 5, the distribution of $\mathbf{Hybrid}_1$ is computationally indistinguishable from that of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, we construct an $2t$-BCL adversary $\mathcal{B}$ attacking C' that simulates the view of $\mathcal{A}^\star$.

1. $\mathcal{B}$ does the following for the first invocation.
   (a) $\mathcal{B}$ invokes $\mathcal{A}^\star$ and obtains the public input $\boldsymbol{x}_1$.
   (b) $\mathcal{B}$ constructs a 2-party protocol π between $\mathcal{A}'_0,\mathcal{A}'_1$ as follows. For each invocation of ρ with input $((m_0,m_1),b)$, B randomly samples T and computes $\mathtt{Sim}_3(T)$. Then $\mathcal{B}$ instructs $\mathcal{A}'_0,\mathcal{A}'_1$ to compute the views of P_0 and P_1 by $\mathtt{Sim}_1((m_0,m_1),\perp,T)$ and $\mathtt{Sim}_2(b,m_b,T)$ respectively. Finally, $\mathcal{A}'_0$ and $\mathcal{A}'_1$ run $\mathcal{A}_0$ and $\mathcal{A}_1$ with the change that both $\mathcal{A}'_0$ and $\mathcal{A}'_1$ will eventually send the memory of $\mathcal{A}_0$ and $\mathcal{A}_1$ to each other. $\mathcal{B}$ sets the leakage query L_1 to be the one corresponding to the protocol π. Then L_1 is a $2t$-BCL.
   (c) $\mathcal{B}$ queries $\boldsymbol{x}_1$ and L_1, and learns $\boldsymbol{y}_1$ and the leakage result. $\mathcal{B}$ sends $\boldsymbol{y}_1$, $\mathtt{Sim}_3(T)$ for each call of ρ, and the communication between $\mathcal{A}_0$ and $\mathcal{A}_1$ (without their memory) to $\mathcal{A}^\star$. $\mathcal{B}$ records the memory of both $\mathcal{A}_0,\mathcal{A}_1$. This finishes the first invocation.
2. For all $2\le i\le q$, in the i-th invocation, $\mathcal{B}$ does the following.
   (a) $\mathcal{B}$ invokes $\mathcal{A}^\star$ and obtains the public input $\boldsymbol{x}_i$.
   (b) $\mathcal{B}$ constructs a 2-party protocol π between $\mathcal{A}'_0,\mathcal{A}'_1$ as follows. For each invocation of ρ with input $((m_0,m_1),b)$, B randomly samples T and computes $\mathtt{Sim}_3(T)$. Then $\mathcal{B}$ instructs $\mathcal{A}'_0,\mathcal{A}'_1$ to compute the views of P_0 and P_1 by $\mathtt{Sim}_1((m_0,m_1),\perp,T)$ and $\mathtt{Sim}_2(b,m_b,T)$ respectively. Finally, $\mathcal{A}'_0$ and $\mathcal{A}'_1$ run $\mathcal{A}_0$ and $\mathcal{A}_1$ with their memory $\mathcal{B}$ records in the last invocation and ask both $\mathcal{A}_0$ and $\mathcal{A}_1$ to send their memory to each other at the end of the protocol. $\mathcal{B}$ sets the leakage query L_i to be the one corresponding to the protocol π. Note that in the real world, since each party has erased their data in all previous epochs. Each $\mathcal{A}_i$ can only access his own memory and the view of P_i in the current invocation. Thus, L_i is a $2t$-BCL for the current invocation.
   (c) $\mathcal{B}$ queries $\boldsymbol{x}_i$ and L_i, and learns $\boldsymbol{y}_i$ and the leakage result. $\mathcal{B}$ sends $\boldsymbol{y}_i$, $\mathtt{Sim}_3(T)$ for each call of ρ, and the communication between $\mathcal{A}_0$ and $\mathcal{A}_1$ (without their memory) to $\mathcal{A}^\star$. $\mathcal{B}$ records the memory of both $\mathcal{A}_0,\mathcal{A}_1$. This finishes the i-th invocation.
3. After all q executions, $\mathcal{B}$ outputs the view of $\mathcal{A}^\star$.

The output of $\mathbf{Hybrid}_2$ is the output of $\mathcal{B}$ together with $\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i=1}^q$.

We show that $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_1$ are identically distributed. Note that $\mathcal{B}$ can be viewed as first computing C' and then using $(\mathtt{Sim}_1,\mathtt{Sim}_2,\mathtt{Sim}_3)$ to accomplish the view of P_0 and P_1 and the transcript in $\mathbf{Hybrid}_1$. Then the leakage query $\mathcal{B}$ makes in each invocation faithfully emulates the actual interaction between $\mathcal{A}_0$ and $\mathcal{A}_1$. Thus, the view of $\mathcal{A}^\star$ is identically distributed to the output of $\mathcal{B}$. Since $\mathcal{B}$ queries the same input as $\mathcal{A}^\star$ does in each invocation, the joint distribution of the view of $\mathcal{A}^\star$ and $\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i=1}^q$ is identical to the joint distribution of the output of $\mathcal{B}$ and $\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i=1}^q$. $\square$

5 Deterministic Leakage-Resilient Circuits: A General Blueprint

In this section, for a given leakage class $\mathcal{L}$, our goal is to construct deterministic stateful $\mathcal{L}$-leakage-resilient circuits from randomness-efficient $\mathcal{L}$-leakage-tolerant circuits. I.e., the resulting stateful circuit does not require fresh randomness except the first invocation. We restrict ourselves to decomposable leakage function classes defined below.

Definition 2 (Decomposable Leakage Functions)**.** *We say $\mathcal{L}$ is a decomposable leakage function class if for every $L : \{0,1\}^n \to \{0,1\}^m \in \mathcal{L}$ and every partition $S_1, S_2, \ldots, S_k$ of $[n]$, there exist leakage functions $L_1, L_2, \ldots, L_k \in \mathcal{L}$ and a function L_0 such that $L(\boldsymbol{x}) = L_0(L_1(\boldsymbol{x}_{S_1}), \ldots, L_k(\boldsymbol{x}_{S_k}))$. Here $\boldsymbol{x}_{S_i}$ denotes the vector $(x_i)_{i \in S_i}$.*

In other words, for a decomposable leakage function class, $L(\boldsymbol{x})$ can be computed from individual leakage on each part of the input.

We first define the notions of 2-adaptive $\mathcal{L}$-leakage-resilient encoding and extractors for $\mathcal{L}$-leakage sources with reverse sampling.

Definition 11 (2-adaptive $\mathcal{L}$-leakage-resilient Encoding)**.** *For a pair of functions $(\mathtt{Enc}, \mathtt{Dec})$ (where $\mathtt{Enc}$ is randomized and $\mathtt{Dec}$ is deterministic), we say $(\mathtt{Enc}, \mathtt{Dec})$ is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding scheme if it satisfies the following two properties.*

- *For all $m \in \{0,1\}$,*
$$\Pr[\mathtt{Dec}(\mathtt{Enc}(m)) = m] = 1.$$
- *Any adversary cannot distinguish an encoding of 0 from that of 1 by making two adaptively chosen leakage queries in $\mathcal{L}$ on the codeword. Formally, for all (computationally unbounded) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following two distributions are statistically close with negligible distance:*
$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$
$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$

Definition 12 (Extractor for $\mathcal{L}$-leakage Source with Reverse Sampling)**.** *For a function $\mathtt{Ext} : \{0,1\}^I \to \{0,1\}$, we say $\mathtt{Ext}$ is a randomness extractor for $\mathcal{L}$-leakage source with reverse sampling if there exists a PPT simulator $\mathtt{Sim}$, where $\mathtt{Sim}$ takes $L \in \mathcal{L}$ with input size I as input and outputs a simulated leakage result ℓ together with two strings r_0, r_1, such that for all $L \in \mathcal{L}$ with input size I, the following two distributions are statistically close with negligible distance ϵ:*

$$(L(r), \mathtt{Ext}(r), r) : r \leftarrow \{0,1\}^I \approx_\epsilon (\ell, b, r_b) : (\ell, r_0, r_1) \leftarrow \mathtt{Sim}(L), b \leftarrow \{0,1\}$$

For a string $\boldsymbol{s}$, we use $\mathtt{Enc}(\boldsymbol{s})$ to denote the concatenation of the encoding of every bit in $\boldsymbol{s}$.

5.1 Formal Description

Theorem 4. *Let $\mathcal{L}$ be a decomposable leakage function class. Assume that for every polynomial adversary bound $S(\kappa)$, there exist the following tools:*

1. *A compiler that transforms a deterministic circuit C of size $S(\kappa)$ into an $\mathcal{L}$-leakage-tolerant circuit whose randomness complexity $\mathtt{RC}(S(\kappa), \kappa)$ is poly-logarithmic in the circuit size $S(\kappa)$, and whose simulator has circuit size $T(S(\kappa), \kappa)$;*

2. *There is a pseudorandom generator* $\mathtt{PRG}$ *with seed length* κ *that is secure against adversaries with circuit size poly*$(\kappa, T(S(\kappa), \kappa))$*;*

3. *A 2-adaptive* $\mathcal{L}$*-leakage-resilient encoding scheme* $(\mathtt{Enc}, \mathtt{Dec})$ *as defined in Definition 11;*

4. *A deterministic randomness extractor* $\mathtt{Ext} : \{0,1\}^I \to \{0,1\}$ *for* $\mathcal{L}$*-leakage sources with efficient reverse sampling as defined in Definition 12.*

Then there exists an $\mathcal{L}$*-LRC compiler* $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ *such that*

- *For every polynomial adversary bound* $S(\kappa)$*, there is a simulator bound* $S_0(\kappa) = poly(T(S(\kappa), \kappa), \kappa)$ *such that* $\mathtt{Comp}$ *is an* $(\mathcal{L}, S, S_0, S)$*-LRC compiler;*

- *For every stateful circuit* C*, the circuit* C' *generated by* $\mathcal{T}_C(1^\kappa, C)$ *does not require any fresh random bits beyond those embedded in the encoded initial state.*

Our construction of $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ works as follows. $\mathcal{T}_s$ takes the initial state $\boldsymbol{s}_0$ as input. Then $\mathcal{T}_s$ samples a random seed of length κ for $\mathtt{PRG}$ and outputs $\boldsymbol{s}'_0 = \mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0)$.

Let f be a deterministic binary function defined as follows.

1. f takes as input $\mathtt{Enc}(\boldsymbol{s} \| \mathtt{Seed})$ and $\boldsymbol{x}$.

2. f decodes the first input by $\mathtt{Dec}$ and obtains $\boldsymbol{s}$ and $\mathtt{Seed}$.

3. f computes $\mathtt{PRG}(\mathtt{Seed})$ and the output is partitioned into 4 parts $(\boldsymbol{r}_0, \widetilde{\mathtt{Seed}}, \boldsymbol{r}_1, \widetilde{\mathtt{Rtape}})$.

4. f computes $C[\boldsymbol{s}](\boldsymbol{x})$ by using $\boldsymbol{r}_0$ as internal randomness. Let $\boldsymbol{y}$ be the public output and $\tilde{\boldsymbol{s}}$ be the updated state.

5. f computes $\mathtt{Enc}(\tilde{\boldsymbol{s}} \| \widetilde{\mathtt{Seed}})$ by using $\boldsymbol{r}_1$ as internal randomness. Let $\tilde{\boldsymbol{s}}' = (\mathtt{Enc}(\tilde{\boldsymbol{s}} \| \widetilde{\mathtt{Seed}}), \widetilde{\mathtt{Rtape}})$.

6. f outputs $\tilde{\boldsymbol{s}}', \boldsymbol{y}$.

Let $\hat{C}$ be an $\mathcal{L}$-leakage-tolerant circuit for f. Let $|f|$ denote the circuit size of f. Then $\hat{C}$ requires $\mathtt{RC}(|f|, \kappa)$ random bits. The length of $\widetilde{\mathtt{Rtape}}$ is set to be (at least) $I \cdot \mathtt{RC}(|f|, \kappa)$. At a first look, there is a circular dependence between $|f|$ and $\widetilde{\mathtt{Rtape}}$. We note that $\mathtt{RC}(|f|, \kappa)$ is poly-logarithmic in $|f|$ and $|f|$ is polynomial in the length of $\widetilde{\mathtt{Rtape}}$. Thus, there exists a large enough length of $\widetilde{\mathtt{Rtape}}$ such that it is at least $I \cdot \mathtt{RC}(|f|, \kappa)$.

The construction of $C'[\boldsymbol{s}'_0]$ is as follows:

- In the first invocation with public input $\boldsymbol{x}_1$, $C'[\boldsymbol{s}'_0]$ views $\boldsymbol{s}'_0$ as $\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0)$ and takes as input a random tape $\mathtt{Rtape}_0$ of size $I \cdot \mathtt{RC}(|f|, \kappa)$. This random tape is only needed in the first execution. Then $\mathtt{Rtape}_0$ is partitioned into $\mathtt{RC}(f, \kappa)$ parts of length I. C' computes $b_j = \mathtt{Ext}(\mathtt{Rtape}_{0,j})$ for all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$ and sets $\mathtt{Rtape}'_0 = (b_1, b_2, \ldots, b_{\mathtt{RC}(f,\kappa)})$. Next, C' computes $\hat{C}(\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0), \boldsymbol{x}_1)$ by using $\mathtt{Rtape}'_0$ as internal randomness and obtains $(\boldsymbol{s}'_1, \boldsymbol{y}_1)$, which are regarded as the updated state and the public output respectively.

- In the i-th ($i \geq 2$) invocation with public input $\boldsymbol{x}_i$, $C'[\boldsymbol{s}'_{i-1}]$ parses $\boldsymbol{s}'_{i-1}$ as $(\mathtt{Enc}(\boldsymbol{s}_{i-1} \| \mathtt{Seed}_{i-1}), \mathtt{Rtape}_{i-1})$. Then $\mathtt{Rtape}_{i-1}$ is partitioned into $\mathtt{RC}(f, \kappa)$ parts of length I. C' computes $b_j = \mathtt{Ext}(\mathtt{Rtape}_{i-1,j})$ for all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$ and sets $\mathtt{Rtape}'_{i-1} = (b_1, b_2, \ldots, b_{\mathtt{RC}(f,\kappa)})$. Next, C' computes $\hat{C}(\mathtt{Enc}(\boldsymbol{s}_{i-1} \| \mathtt{Seed}_{i-1}), \boldsymbol{x}_i)$ by using $\mathtt{Rtape}'_{i-1}$ as internal randomness and obtains $(\boldsymbol{s}'_i, \boldsymbol{y}_i)$, which are regarded as the updated state and the public output respectively.

5.2 Security Analysis

In this part, we prove the security of our construction according to Definition 8. Let C be the (unprotected) stateful circuit and $(C', \mathcal{T})$ be the stateful circuit obtained from our compiler. For all $(S, S, \mathcal{L})$-adversary circuit $\mathcal{A}$, we first give the construction of the simulator $\mathcal{S}$.

$\mathcal{S}$ starts with invoking $\mathcal{A}$. Let $(\mathtt{Sim}_1, \mathtt{Sim}_2)$ be the simulator guaranteed by the $\mathcal{L}$-leakage tolerance of $\hat{C}$, and $\mathtt{Sim}'$ be the simulator guaranteed by the randomness extractor $\mathtt{Ext}$ for $\mathcal{L}$-leakage source with reverse sampling. In the beginning, $\mathcal{S}$ sets $\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0)$ to be a random encoding of an all-0 vector. As for $\mathtt{Rtape}_0$, let $L' \in \mathcal{L}$ be a constant function of input size I. For all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$, $\mathcal{S}$ invokes $\mathtt{Sim}'(L)$ to obtain (ℓ, r_0, r_1). Then $\mathcal{S}$ sets $\mathtt{Rtape}_{0,j} = r_0 \cdot (1 - b_j) + r_1 \cdot b_j$, where b_j is a random variable uniformly distributed in $\{0, 1\}$. $\mathcal{S}$ computes each bit of $\mathtt{Ext}(\mathtt{Rtape}_{0,j})$ as a random variable of b_j. Let $\mathtt{Rtape}'_0 = (b_1, \ldots, b_{\mathtt{RC}(f,\kappa)})$. $\mathcal{S}$ does the following for the i-th invocation.

1. $\mathcal{S}$ invokes $C[\boldsymbol{s}_{i-1}]$ with public input $\boldsymbol{x}_i$ and obtains the public output $\boldsymbol{y}_i$.
2. $\mathcal{S}$ sets $\mathtt{Enc}(\boldsymbol{s}_i \| \mathtt{Seed}_i)$ to be a random encoding of an all-0 vector.
3. Upon receiving the leakage query L_i from $\mathcal{A}$, $\mathcal{S}$ first computes an equivalent leakage query $\hat{L}_i$ on $\hat{C}$. This is done by replacing all wires in $\mathtt{Ext}(\mathtt{Rtape}_{i,j})$ by random variables of b_j, which is the j-th bit of $\mathtt{Rtape}'_{i-1}$.
4. $\mathcal{S}$ runs $\mathtt{Sim}_1(\hat{L}_i)$ and obtains $(\mathtt{st}_i, L'_i)$. Here L'_i is an $\mathcal{L}$-leakage query on the input and output of $\hat{C}$, which are $(\mathtt{Enc}(\boldsymbol{s}_{i-1} \| \mathtt{Seed}_{i-1}), \boldsymbol{x}_i)$ and $((\mathtt{Enc}(\boldsymbol{s}_i \| \mathtt{Seed}_i), \mathtt{Rtape}_i), \boldsymbol{y}_i)$ respectively.
5. $\mathcal{S}$ decomposes L'_i to $L'_{i,1}, \ldots, L'_{i,\mathtt{RC}(f,\kappa)}, L'_{i,\mathtt{RC}(f,\kappa)+1}, L'_{i,\mathtt{RC}(f,\kappa)+2}$, where $L'_{i,j}$ is applied on $\mathtt{Rtape}_{i,j}$ for all $j \leq \mathtt{RC}(f, \kappa)$, $L'_{i,\mathtt{RC}(f,\kappa)+1}$ is applied on $(\mathtt{Enc}(\boldsymbol{s}_{i-1} \| \mathtt{Seed}_{i-1}), \boldsymbol{x}_i)$, and $L'_{i,\mathtt{RC}(f,\kappa)+2}$ is applied on $((\mathtt{Enc}(\boldsymbol{s}_i \| \mathtt{Seed}_i), \mathtt{Rtape}_i), \boldsymbol{y}_i)$.

   $\mathcal{S}$ computes the leakage result of $L'_{i,\mathtt{RC}(f,\kappa)+1}, L'_{i,\mathtt{RC}(f,\kappa)+2}$ directly. For all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$, $\mathcal{S}$ invokes $\mathtt{Sim}'(L'_{i,j})$ to obtain (ℓ_j, r_0, r_1). $\mathcal{S}$ views ℓ_j as $L'_{i,j}(\mathtt{Rtape}_{i,j})$.

   $\mathcal{S}$ computes the leakage result of L'_i by using the leakage results of $\{L'_{i,j}\}_{j=1}^{\mathtt{RC}(f,\kappa)+2}$. $\mathcal{S}$ uses $\mathtt{Sim}_2$ to simulate the leakage result of L_i. Finally, $\mathcal{S}$ returns the simulated leakage and the function output $\boldsymbol{y}_i$ to $\mathcal{A}$.
6. For all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$, $\mathcal{S}$ sets $\mathtt{Rtape}_{i,j} = r_0 \cdot (1 - b_j) + r_1 \cdot b_j$, where b_j is a random variable uniformly distributed in $\{0, 1\}$. $\mathcal{S}$ computes each bit of $\mathtt{Ext}(\mathtt{Rtape}_{i,j})$ as a random variable of b_j. Let $\mathtt{Rtape}'_i = (b_1, \ldots, b_{\mathtt{RC}(f,\kappa)})$.

We argue the security by hybrid arguments. Let $q \leq S$ be the number of invocations made by the adversary $\mathcal{A}$.

Hybrid$_0$: This hybrid is the real world execution. The output is the transcript of the real-world experiment, which includes the view of $\mathcal{A}$ together with $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{q}$.

Hybrid$_1$: In this hybrid, we change the way of generating $\mathtt{Rtape}_0$ as follows: Let $L' \in \mathcal{L}$ be a constant function of input size I. For all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$, $\mathcal{S}$ invokes $\mathtt{Sim}'(L)$ to obtain (ℓ, r_0, r_1). Then $\mathcal{S}$ samples a random bit as b_j and sets $\mathtt{Rtape}_{0,j} = r_{b_j}$. By definition, for all $j \in \{1, \ldots, \mathtt{RC}(f, \kappa)\}$, $(\mathtt{Ext}(\mathtt{Rtape}_{0,j}), \mathtt{Rtape}_{0,j}) \approx_\epsilon (b_j, r_{b_j})$. Thus **Hybrid**$_1$ and **Hybrid**$_0$ are statistically close.

$\mathbf{Hybrid}_2$: In the following small hybrids, we change to simulate the leakage and function output in the first invocation as described above.

$\mathbf{Hybrid}_{2,0}$: This hybrid is identical to $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_{2,1}$: In this hybrid, $\mathcal{S}$ computes each bit of $\mathtt{Ext}(\mathtt{Rtape}_{0,j})$ as a random variable of b_j. Let $\mathtt{Rtape}'_0 = (b_1, \ldots, b_{\mathtt{RC}(f,\kappa)})$. After receiving L_1 from $\mathcal{A}$, $\mathcal{S}$ computes an equivalent leakage query $\hat{L}_1$ on $\hat{C}$. This is done by replacing all wires in $\mathtt{Ext}(\mathtt{Rtape}_{0,j})$ by random variables of b_j, which is the j-th bit of $\mathtt{Rtape}'_0$. Then the leakage is computed by applying $\hat{L}_1$ on $\hat{C}$. $\mathbf{Hybrid}_{2,1}$ and $\mathbf{Hybrid}_{2,0}$ are identically distributed.

$\mathbf{Hybrid}_{2,2}$: In this hybrid, $\mathcal{S}$ runs $\mathtt{Sim}_1(\hat{L}_1)$ and obtains $(\mathtt{st}_1, L'_1)$. Here L'_1 is an $\mathcal{L}$-leakage query on the input and output of $\hat{C}$, which are $(\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0), \boldsymbol{x}_1)$ and $((\mathtt{Enc}(\boldsymbol{s}_1 \| \mathtt{Seed}_1), \mathtt{Rtape}_1), \boldsymbol{y}_1)$ respectively. Then, the leakage result of L_1 is simulated by $\mathcal{S}$ by using $\mathtt{Sim}_2$ and the leakage result of L'_1 on the input and output of $\hat{C}$. By the security of $\mathcal{L}$-leakage-tolerant circuits, $\mathbf{Hybrid}_{2,2}$ and $\mathbf{Hybrid}_{2,1}$ are statistically close.

$\mathbf{Hybrid}_{2,3}$: In this hybrid, $\mathcal{S}$ computes f to obtain the input and output of $\hat{C}$. By the definition of $\mathcal{L}$-leakage-tolerant circuits, $\mathbf{Hybrid}_{2,3}$ and $\mathbf{Hybrid}_{2,2}$ are identically distributed.

$\mathbf{Hybrid}_{2,4}$: In this hybrid, the output of $\mathtt{PRG}(\mathtt{Seed}_0)$ is replaced by random strings. By the security of $\mathtt{PRG}$, $\mathbf{Hybrid}_{2,4}$ and $\mathbf{Hybrid}_{2,3}$ are computationally indistinguishable. Note that in $\mathbf{Hybrid}_{2,4}$, $(\boldsymbol{s}_1, \boldsymbol{y}_1)$ are identically distributed to those computed by $C[\boldsymbol{s}_0](\boldsymbol{x}_1)$, $\mathtt{Seed}_1$ and $\mathtt{Rtape}_1$ are uniformly random strings, and $\mathtt{Enc}(\boldsymbol{s}_1 \| \mathtt{Seed}_1)$ is a random encoding of $\boldsymbol{s}_1 \| \mathtt{Seed}_1$.

$\mathbf{Hybrid}_{2,5}$: In this hybrid, $\mathcal{S}$ computes $(\boldsymbol{s}_1, \boldsymbol{y}_1)$ by $C[\boldsymbol{s}_0](\boldsymbol{x}_1)$, samples random strings as $\mathtt{Seed}_1$ and $\mathtt{Rtape}_1$, and computes $(\mathtt{Enc}(\boldsymbol{s}_1 \| \mathtt{Seed}_1)$ as a random encoding of $\boldsymbol{s}_1 \| \mathtt{Seed}_1$. $\mathbf{Hybrid}_{2,5}$ and $\mathbf{Hybrid}_{2,4}$ are identically distributed.

$\mathbf{Hybrid}_{2,6}$: In this hybrid, $\mathcal{S}$ first decomposes L'_1 to $\{L'_{1,j}\}_{j=1}^{\mathtt{RC}(f,\kappa)+2}$ as described above. The leakage result of L'_1 is computed from the leakage results of $\{L'_{1,j}\}_{j=1}^{\mathtt{RC}(f,\kappa)+2}$. By the property of decomposable leakage function classes, $\mathbf{Hybrid}_{2,6}$ and $\mathbf{Hybrid}_{2,5}$ are identically distributed.

$\mathbf{Hybrid}_{2,7}$: In this hybrid, $\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0)$ is replaced by a random encoding of an all-0 vector. Note that $\mathtt{Enc}(\boldsymbol{s}_0 \| \mathtt{Seed}_0)$ is only used to obtain the leakage result of $L'_{1,\mathtt{RC}(f,\kappa)+1}$. By the security of 2-adaptive $\mathcal{L}$-leakage-resilient encoding schemes, $\mathbf{Hybrid}_{2,7}$ and $\mathbf{Hybrid}_{2,6}$ are statistically close.

$\mathbf{Hybrid}_{2,8}$: In this hybrid, for each $j \in \{1, \ldots, \mathtt{RC}(f,\kappa)\}$, $\mathcal{S}$ invokes $\mathtt{Sim}'(L'_{1,j})$ to obtain (ℓ_j, r_0, r_1). $\mathcal{S}$ uses ℓ_j as $L'_{1,j}(\mathtt{Rtape}_{1,j})$. Then $\mathcal{S}$ samples a random bit as b_j and sets $\mathtt{Rtape}_{0,j} = r_{b_j}$. By definition, $(L'_{1,j}(\mathtt{Rtape}_{1,j}), \mathtt{Ext}(\mathtt{Rtape}_{1,j}), \mathtt{Rtape}_{1,j}) \approx_\epsilon (\ell_j, b_j, r_{b_j})$. Thus $\mathbf{Hybrid}_{2,8}$ and $\mathbf{Hybrid}_{2,7}$ are statistically close.

We define $\mathbf{Hybrid}_{1+i,j}$ for all $i \in \{2, \ldots, q\}$ and $j \in \{0, \ldots, 8\}$ similarly to handle the rest of invocations. The only difference is that in $\mathbf{Hybrid}_{1+i,7}$, $\mathtt{Enc}(\boldsymbol{s}_{i-1} \| \mathtt{Seed}_{i-1})$ is used to obtain leakage results of both $L'_{i-1,\mathtt{RC}(f,\kappa)+2}$ and $L'_{i,\mathtt{RC}(f,\kappa)+1}$. By the security of 2-adaptive $\mathcal{L}$-leakage-resilient encoding schemes, $\mathbf{Hybrid}_{1+i,7}$ and $\mathbf{Hybrid}_{1+i,6}$ are statistically close.

$\mathbf{Hybrid}_{q+2}$: In this hybrid, we change $\mathtt{Enc}(\boldsymbol{s}_q \| \mathtt{Seed}_q)$ by a random encoding of an all-0 vector. Note that in $\mathbf{Hybrid}_{q+1,8}$, $\mathtt{Enc}(\boldsymbol{s}_q \| \mathtt{Seed}_q)$ is only used to obtain the leakage result of $L'_{q,\mathtt{RC}(f,\kappa)+2}$. By the security of 2-adaptive $\mathcal{L}$-leakage-resilient encoding schemes, $\mathbf{Hybrid}_{q+2}$ and $\mathbf{Hybrid}_{q+1,8}$ are statistically close.

$\mathbf{Hybrid}_{q+3}$: In this hybrid, the output $\boldsymbol{y}$ in each invocation is obtained by calling $C[\boldsymbol{s}](\boldsymbol{x})$ in a black box way. This does not change the distribution of $\boldsymbol{y}$. Note that $\mathbf{Hybrid}_{q+3}$ is the ideal world.

6 Instantiating Blueprint for Depth-1 $\mathcal{AC}0$ Leakage

In this section, we give a concrete instantiation of our compiler of deterministic stateful leakage-resilient circuits for depth-1 $\mathcal{AC}0$ leakage (or D1-leakage for short). Concretely, the D1 leakage class contains all leakage functions $L : \{0,1\}^n \to \{0,1\}$ defined by $S_0, S_1 \subset \{1,\ldots,n\}$ such that $L(\boldsymbol{x}) = (\bigvee_{i\in S_0} x_i) \vee (\bigvee_{i\in S_1}(x_i \oplus 1))$.

For a leakage bound parameter t, we use t-D1 leakage class to denote the t-leakage class of the depth-1 $\mathcal{AC}0$ leakage class. I.e., each leakage function corresponds to t D1-leakage queries. In the following we use (t,ϵ)-D1 tolerance to denote leakage tolerance against t-D1 leakage with error ϵ. We first construct a (t,ϵ)-D1-tolerant circuit whose randomness complexity is poly-logarithmic in the circuit size. In Section 6.1, we give an overview of our construction. Then we formally describe our construction and prove its security in Section 6.2 and Section 6.3. Finally, in Section 6.4, we construct a deterministic randomness extractor for t-D1-leakage source with efficient reverse sampling.

6.1 Overview of Our Construction

Our idea is to derandomize the t-D1-tolerant circuits constructed in [IS24]. For simplicity, we focus on a single D1-leakage query.

Review of Construction in [IS24]. At a high level, the construction in [IS24] can be divided into two parts: (1) the outer construction for computing the target function f assuming the existence of D1-tolerant circuits for linear functions, and (2) the inner construction for computing any linear functions with D1-tolerance.

For the outer construction, the main observation is that a D1-leakage query cannot distinguish whether the secret of a random additive sharing with $\kappa+1$ shares is 0 or 1. Recall that a D1-leakage query is a function $L : \{0,1\}^n \to \{0,1\}$ defined by $S_0, S_1 \subset \{1,\ldots,n\}$ such that $L(\boldsymbol{x}) = (\bigvee_{i\in S_0} x_i) \vee (\bigvee_{i\in S_1}(x_i \oplus 1))$. Consider the following two cases:

- If $|S_0 \cup S_1| \geq \kappa$, then with overwhelming probability, $L(\boldsymbol{x}) = 1$. This is because $L(\boldsymbol{x}) = 1$ as long as any bit in S_0 is 1 or any bit in S_1 is 0. Since any κ bits of $\boldsymbol{x}$ are uniformly random, the probability that all bits in S_0 are 0 and all bits in S_1 are 1 is negligible (no more than $2^{-\kappa}$) no matter whether the secret is 0 or 1.
- If $|S_0 \cup S_1| \leq \kappa$, then all bits in $S_0 \cup S_1$ are random and independent of the secret. Thus the leakage is independent of the secret.

Therefore, the idea is to compute an additive sharing for each wire value in the circuit. Let C be an unprotected circuit that computes the target function f. The outer construction is as follows:

1. For each input wire x of C, prepare κ random bits $(x_0,\ldots,x_{\kappa-1})$ and use the gadget for linear functions to compute $x_\kappa = x \oplus (\bigoplus_{i=0}^{\kappa-1} x_i)$. Then $(x_0,\ldots,x_\kappa)$ is an additive sharing of x, denoted by $\langle x\rangle$.
2. For each random wire of C, prepare $\kappa+1$ random bits and view them as an additive sharing of this random wire.

3. For each addition gate with input additive sharings $\langle x\rangle, \langle y\rangle$, the goal is to compute a random additive sharing $\langle x+y\rangle$. Prepare κ random bits $(z_0, \ldots, z_{\kappa-1})$ and use the gadget for linear functions to compute $z_\kappa = (\bigoplus_{i=0}^{\kappa} x_i) \oplus (\bigoplus_{i=0}^{\kappa} y_i) \oplus (\bigoplus_{i=0}^{\kappa-1} z_i)$.

4. For each multiplication gate with input additive sharings $\langle x\rangle, \langle y\rangle$, first compute $x_i \cdot y_j$ for all $i, j \in \{0, \ldots, \kappa\}$. Then prepare κ random bits $(z_0, \ldots, z_{\kappa-1})$ and use the gadget for linear functions to compute $z_\kappa = (\bigoplus_{i,j=0}^{\kappa} x_i \cdot y_j) \oplus (\bigoplus_{i=0}^{\kappa-1} z_i)$.

5. For each output with additive sharing $\langle x\rangle$, use the gadget for linear functions to compute $x = \bigoplus_{i=0}^{\kappa} x_i$.

By a standard hybrid argument, we may simulate all inner wires by random additive sharings of 0. Then a D1-leakage query on the wires in the above construction is naturally reduced to a D1-leakage query on the input and output of the circuit. This shows the security of the outer construction.

As for the inner construction, [IS24] proves that the probing-tolerant variant of the ISW construction is also D1-tolerant. Suppose the target function f is a linear function and let C be an unprotected circuit that computes f. The ISW construction works as follows:

1. For each input wire x of C, prepare κ random bits $(x_0, \ldots, x_{\kappa-1})$. Then compute $x_\kappa = x \oplus x_0 \oplus \ldots \oplus x_{\kappa-1}$ in order. Now $(x_0, \ldots, x_\kappa)$ is an additive sharing of x.

2. For each addition gate with input additive sharings $\langle x\rangle, \langle y\rangle$, computes $z_i = x_i \oplus y_i$ for all $i \in \{0, \ldots, \kappa\}$. Then $\langle z\rangle = (z_0, \ldots, z_\kappa)$ is viewed as an additive sharing of $z = x \oplus y$.

3. For each output with additive sharing $\langle x\rangle$, compute $x = x_\kappa \oplus x_{\kappa-1} \oplus \ldots \oplus x_0$ in order.

Consider a D1-leakage query L defined by S_0, S_1, which are subsets of the wires in the above circuit. For all $j \in \{0, \ldots, \kappa\}$, we say L touches j if for some additive sharing $\langle x\rangle$, the j-th share x_j is in $S_0 \cup S_1$. The main observation in [IS24] is that if L touches every $j \in \{0, \ldots, \kappa\}$, then there are at least κ wires in $S_0 \cup S_1$ that are uniformly random. Following the same argument as above, the output of L is 1 with overwhelming probability. On the other hand, if L touches at most κ indices, then every wire in the circuit can be represented as a variable that only depends on a single input or output bit. In this way, L is naturally reduced to a D1-leakage query applied on the input and output wires of the function, which shows the D1-tolerance of the construction.

To construct randomness-efficient D1-leakage-tolerant circuits, our idea is to derandomize the outer construction and the inner construction in [IS24] separately.

Derandomization of Outer Construction. Our starting idea is to use the output of a κ-wise independent PRG to mask the wire values in C rather than computing an additive sharing for each wire. In this way, we only need random bits for the input of the PRG.

This idea has been used in [GIS22] to construct randomness-efficient probing-resilient/tolerant circuits. In our case, we want to utilize the fact that after masking the wires by the output of a κ-wise independent PRG, any κ inner wires are uniformly random. Thus, for a D1-leakage defined by S_0, S_1,

- If $|S_0 \cup S_1| \geq \kappa$, then there are at least κ random bits in $S_0 \cup S_1$. Following the same argument as above, the output of L is 1 with overwhelming probability.

- Otherwise, $|S_0 \cup S_1| \leq \kappa$. Then all wires in $S_0 \cup S_1$ (except the input and output wires) are uniformly random. L is naturally reduced to a D1-leakage query applied on the input and output wires of the function.

To be more concrete, we use a strong and linear κ-wise independent PRG $G : \{0,1\}^\lambda \to \{0,1\}^{|C|}$. It satisfies that G can be computed by only using addition (XOR) gates and when $\boldsymbol{r}$ is randomly sampled from $\{0,1\}^\lambda$, any κ bits of $(\boldsymbol{r}, G(\boldsymbol{r}))$ are uniformly random. In [GIS22], the authors have shown that such a PRG exists with seed length $\lambda = O(\kappa \cdot \log |C|)$. Let $\boldsymbol{u} = G(\boldsymbol{r})$. Our goal is to compute $(\boldsymbol{r}, \boldsymbol{u} \oplus \boldsymbol{w})$ where $\boldsymbol{w}$ are all wires in C. Since G is linear, each u_i can be computed by a linear combination of the input bits $\boldsymbol{r}$. Thus, for each i, there exists a subset W_i of bits in $\boldsymbol{r}$ such that the parity of bits in W_i is u_i. Together with $u_i \oplus w_i$, we can view these $|W_i| + 1$ bits as an additive sharing of w_i and the number of shares is upper bounded by $\lambda + 1$. Now we try to use these additive sharings in the outer construction in [IS24].

We note that for an addition gate, we can continue to use the gadget for linear functions to compute the additive sharing for its output wire. On the other hand, for a multiplication gate with input sharings $\langle x \rangle, \langle y \rangle$, we have to compute $x_i \cdot y_j$ for all $i \in \{0, 1, \ldots, |W_i|\}, j \in \{0, 1, \ldots, |W_j|\}$. In [IS24], each additive sharing uses κ fresh random bits. In their case, they can argue the security by switching the secret of an additive sharing to 0 each time. When fixing all other additive sharings, an adversary essentially tries to distinguish whether a random additive sharing has secret 0 or 1. This analysis does not work in our case since the randomness of different additive sharings is correlated.

To address this issue, our idea is to prepare two copies of additive sharings for each wire with independent randomness. When computing a multiplication gate, we multiply shares of x from the first copy with shares of y from the second copy. In this way, we avoid doing multiplications between shares that use correlated randomness. We give a high-level description of our outer construction below.

1. Sample $\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)} \in \{0,1\}^\lambda$. Let $\boldsymbol{u}^{(0)} = G(\boldsymbol{r}^{(0)})$ and $\boldsymbol{u}^{(1)} = G(\boldsymbol{r}^{(1)})$. For all $i \in \{1, \ldots, |C|\}$, let W_i be the subset of $\{1, \ldots, \lambda\}$ such that $u_i^{(0)} = \bigoplus_{i \in W_i} r_i^{(0)}$ and $u_i^{(1)} = \bigoplus_{i \in W_i} r_i^{(1)}$.

2. For each input wire w_i, we will compute $u_i^{(0)} \oplus w_i$ and $u_i^{(1)} \oplus w_i$. This is done by using the gadget for linear functions to compute $w_i \oplus (\bigoplus_{j \in W_i} r_j^{(0)})$ and $w_i \oplus (\bigoplus_{j \in W_i} r_j^{(1)})$. We view $\{r_j^{(0)}\}_{j \in W_i} \cup \{u_i^{(0)} \oplus w_i\}$ as an additive sharing $\langle w_i \rangle_0$ and $\{r_j^{(1)}\}_{j \in W_i} \cup \{u_i^{(1)} \oplus w_i\}$ as an additive sharing $\langle w_i \rangle_1$.

3. For each random wire w_i, randomly sample a bit as $u_i^{(0)} \oplus w_i$. Then use the gadget for linear functions to compute $u_i^{(1)} \oplus w_i = (u_i^{(0)} \oplus w_i) \oplus (\bigoplus_{j \in W_i} r_j^{(0)}) \oplus (\bigoplus_{j \in W_i} r_j^{(1)})$. We view $\{r_j^{(0)}\}_{j \in W_i} \cup \{u_i^{(0)} \oplus w_i\}$ as an additive sharing $\langle w_i \rangle_0$ and $\{r_j^{(1)}\}_{j \in W_i} \cup \{u_i^{(1)} \oplus w_i\}$ as an additive sharing $\langle w_i \rangle_1$.

4. For each addition gate with input wires w_i, w_j and output wire w_k, suppose we have computed $\langle w_i \rangle_0, \langle w_j \rangle_0$ and $\langle w_i \rangle_1, \langle w_j \rangle_1$. We use the gadget for linear functions to compute $u_k^{(0)} \oplus w_k$ and $u_k^{(1)} \oplus w_k$ to obtain $\langle w_k \rangle_0$ and $\langle w_k \rangle_1$.

5. For each multiplication gate with input wires w_i, w_j and output wire w_k, suppose we have computed $\langle w_i \rangle_0, \langle w_j \rangle_0$ and $\langle w_i \rangle_1, \langle w_j \rangle_1$. We first compute $w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$ for all $\ell_1 \in \{0, 1, \ldots |W_i|\}$,

$\ell_2 \in \{0, 1, \ldots, |W_j|\}$. Here $w_{i,0,\ell_1}$ is the ℓ_1-th share of $\langle w_i \rangle_0$ and $w_{j,1,\ell_2}$ is the ℓ_2-th share of $\langle w_j \rangle_1$. Note that $w_k = \bigoplus_{\ell_1,\ell_2} w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$. We use the gadget for linear functions to compute $u_k^{(0)} \oplus w_k$ and $u_k^{(1)} \oplus w_k$ to obtain $\langle w_k \rangle_0$ and $\langle w_k \rangle_1$.

6. For each output wire with additive sharing $\langle w_i \rangle_0$ and $\langle w_i \rangle_1$. We use the gadget for linear functions to compute w_i.

To argue the security of our construction, note that the inner wires of the outer construction can be computed from $(\boldsymbol{r}^{(0)}, \boldsymbol{u}^{(0)} \oplus \boldsymbol{w}), (\boldsymbol{r}^{(1)}, \boldsymbol{u}^{(1)} \oplus \boldsymbol{w})$. Consider the following two hybrids:

- In the first hybrid, we compute the inner wires by using $(\boldsymbol{r}^{(0)}, \boldsymbol{u}^{(0)} \oplus \boldsymbol{w}), (\boldsymbol{r}^{(1)}, \boldsymbol{u}^{(1)} \oplus \mathbf{0})$. By fixing $\boldsymbol{r}^{(0)}$ and $\boldsymbol{w}$, an adversary essentially tries to distinguish $(\boldsymbol{r}^{(1)}, \boldsymbol{u}^{(1)} \oplus \boldsymbol{w})$ and $(\boldsymbol{r}^{(1)}, \boldsymbol{u}^{(1)} \oplus \mathbf{0})$. As we have argued above, the adversary cannot distinguish theses two distributions by D1-leakage.

- In the second hybrid, we compute the inner wires by using $(\boldsymbol{r}^{(0)}, \boldsymbol{u}^{(0)} \oplus \mathbf{0}), (\boldsymbol{r}^{(1)}, \boldsymbol{u}^{(1)} \oplus \mathbf{0})$. Now all inner wires can be generated and independent of the input and output of the circuit. Thus, the D1-leakage query made by the adversary is naturally reduced to a D1-leakage query on the input and output of the circuit.

Derandomization of Inner Construction. Now we move to derandomize the inner construction. We note that the inner construction is used to compute addition among at most $(\lambda+1)^2+\lambda = \Theta(\lambda^2)$ wires for each gate in the outer construction, where λ is the seed length of the κ-wise independent PRG in the outer construction. Recall that $\lambda = O(\kappa \cdot \log |C|)$, which is logarithmic in the circuit size. Thus, it is affordable to use fresh randomness for a bounded number of gates, but not for each gate in the circuit.

Let $\mathtt{RC} = ((\lambda+1)^2 + \lambda)\kappa$, which is the number of random bits we need to add $(\lambda+1)^2 + \lambda$ bits when using the ISW construction. Our idea is to reuse randomness across different addition circuits. To be more concrete, we want to make sure that the internal randomness of any κ addition circuits are uniformly random. We can achieve this by using $\mathtt{RC}$ copies of κ-wise independent sources. Then the i-th bit of each copy, which are $\mathtt{RC}$ bits in total, are used as the internal randomness for the i-th addition circuit. Now consider a D1-leakage query L defined by S_0, S_1. We say a wire is touched by L if this wire is in $S_0 \cup S_1$. There are two cases:

- If L touches wires in at most κ addition circuits, by construction, those κ addition circuits use fresh randomness. Thus we can rely on D1-leakage-tolerance of the ISW construction directly.

- Otherwise, L touches wires in at least κ addition circuits. We observe that in the ISW construction, each wire that is not the input or output wire is uniformly distributed. Thus, at least κ wires touched by L are uniformly distributed. Now applying the same analysis as above, the output of L is 1 with overwhelming probability.

Now the question is reduced to generating κ-wise independent source. The key problem here is that the adversary can also attack the wires in the generation circuit. We follow the idea of generating κ-wise independent sources with probing tolerance in [ISW03, IKL$^+$13] but change to use a linear κ-wise independent PRG. We first generate $\kappa+1$ copies of such random sources by using the unprotected circuit of a linear κ-wise independent PRG. Then we compute the XOR of these $\kappa+1$ sources as the final output. Intuitively, the security follows from the fact if the D1-leakage query

touches the generation of at most κ random sources, then the output remains κ-wise independent and independent of the leakage. If the query touches the generation of all $\kappa+1$ sources, since we use a linear κ-wise independent PRG, any $\kappa+1$ wires, one from the generation of each source, are uniformly distributed, which means that the leakage result is 1 with overwhelming probability.

In summary, our inner construction is as follows.

1. For each $i \in \{1, \ldots, \mathtt{RC}\}$, we first prepares $\kappa+1$ random seeds for a linear κ-wise independent PRG and computes the outputs by using linear circuits individually. Then we compute the XOR of the $\kappa+1$ output sources to obtain a single κ-wise independent source.

2. After obtaining $\mathtt{RC}$ copies of κ-wise independent sources, we use the i-th bit of each copy as the internal randomness for the i-th addition circuit.

The randomness complexity is $\mathtt{RC} \cdot (\kappa+1) \cdot O(\kappa \cdot \log |C|) = \mathrm{poly}(\kappa, \log |C|)$, which is poly-logarithmic in the circuit size.

Handling t-D1 Leakage. So far, the above overview focuses on a single D1-leakage. Very informally, for our construction, it satisfies that

- If the D1-leakage query touches at least κ wires, then the leakage result is 1 with overwhelming probability.
- If the D1-leakage query touches at most κ wires, then we can reduce it to a D1-leakage query that is only applied on the input and output of the circuit.

For t D1-leakage queries, we follow the same strategy as that in [IS24]. First, we change to use $t\kappa$-wise independent PRG and $t\kappa$-probing-tolerant circuits in the above construction. Now consider t D1-leakage queries made by the adversary. At a high level,

- For each query that touches at least κ wires, following the same analysis, the leakage result is 1 with overwhelming probability.
- Consider the rest of queries where each query touches at most κ wires, those queries in total touch at most $t\kappa$ wires. Following the same strategy as that for a single D1-leakage query, we can reduce them to the same amount of D1-leakage queries that are only applied on the input and output of the circuit.

6.2 Formal Description

In this subsection, we give the formal description of our construction and defer the security proof to the next subsection. We first define the notion of strong and linear k-wise independent PRGs.

Definition 13. *We say $G : \{0,1\}^\lambda \to \{0,1\}^O$ is a strong and linear k-wise independent PRG if it satisfies that*

- ***Linearity**: Each output bit is a linear combination of the input bits.*
- ***Strong k-wise Independence**: When $\boldsymbol{r}$ is randomly sampled from $\{0,1\}^\lambda$, any k bits of $(\boldsymbol{r}, G(\boldsymbol{r}))$ are uniformly distributed.*

It is known [GIS22] that there exists an explicit construction of a strong and linear k-wise independent PRG with seed length $\lambda = O(k \cdot \log O)$.

Theorem 5. *For all circuit C, there exists an explicit construction of (t, ϵ)-D1-tolerant implementation of C that uses poly$(t, \log(1/\epsilon), \log |C|)$ additional random bits (except the random bits required by C), where $|C|$ is the circuit size of C.*

Let κ be the security parameter. Later we will show how to choose κ so that the statistical error of our construction is ϵ. Our construction can be split into two parts: (1) An outer construction that computes C with t-D1 tolerance assuming a black-box gadget for computing linear functions, and (2) an inner construction that computes linear functions with t-D1 tolerance. We first introduce our inner construction.

Inner Construction. Let $m, n_i > 0$ be integers. We consider the task of computing m copies of the addition function $(x_1, \ldots, x_{n_i}) \to x_1 \oplus \ldots \oplus x_{n_i}$. As we mentioned in the overview, for each addition function, we will directly use the construction in [IS24]. To reduce the randomness complexity, we use correlated randomness among different copies.

Let $\mathtt{RC} = n_i \cdot t \cdot \kappa$. As we will see later, the construction in [IS24] for each copy requires $\mathtt{RC}$ random bits. Let $G_0 : \{0,1\}^\lambda \to \{0,1\}^m$ be a strong and linear $t\kappa$-wise independent PRG. We will specify the size of λ later.

1. **Preparing Correlated Random Tapes**: In the first step, we prepare the random tape for each of these m addition functions. For all $i \in \{1, \ldots, \mathtt{RC}\}$, do the following.
   (a) Randomly sample $\boldsymbol{\gamma}^{(i,j)} \in \{0,1\}^\lambda$ for all $j \in \{0, \ldots, t\kappa\}$.
   (b) Compute $\boldsymbol{\nu}^{(i,j)} = G_0(\boldsymbol{\gamma}^{(i,j)})$ by using a linear circuit for all $j \in \{0, \ldots, t\kappa\}$.
   (c) Compute $\boldsymbol{\nu}^{(i)} = \bigoplus_{j=0}^{t\kappa} \boldsymbol{\nu}^{(i,j)}$.

   For all $j \in \{1, \ldots, m\}$, let $\boldsymbol{v}^{(j)} = (\nu_j^{(1)}, \ldots, \nu_j^{(\mathtt{RC})})$. I.e., we pick the j-th bit of each $\boldsymbol{\nu}^{(i)}$ to form $\boldsymbol{v}^{(j)}$. $\boldsymbol{v}^{(j)}$ is used as the random tape for computing the j-th addition function.

2. **Computing Each Addition Function**: For each addition function with input $x_1, \ldots, x_{n_i}$, let $\boldsymbol{v}$ be the random tape associated with this addition function. Do the following to compute $y = x_1 \oplus \ldots \oplus x_{n_i}$.
   (a) For each input wire x_i, let $(x_{i,0}, \ldots, x_{i,t\kappa-1})$ be the first $t\kappa$ unused random bits in $\boldsymbol{v}$. Compute $x_{i,t\kappa} = x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,t\kappa-1}$ in order. View $(x_{i,0}, \ldots, x_{i,t\kappa})$ as an additive sharing of x_i, denoted by $\langle x_i \rangle$.
   (b) Compute $\langle y \rangle = \langle x_1 \rangle \oplus \ldots \oplus \langle x_{n_i} \rangle$ in order. Concretely, when adding two additive sharings, we add the shares coordinate by coordinate.
   (c) Compute $y = y_{t\kappa} \oplus \ldots \oplus y_0$ in order.

The randomness complexity of the inner construction is $O(n_i t^2 \kappa^2 \lambda)$ bits.

Outer Construction. Let $G_1 : \{0,1\}^\lambda \to \{0,1\}^{|C|}$ be a strong and linear $t\kappa$-wise independent PRG. We will specify the size of λ later. Let $n_i = (\lambda+1)^2 + \lambda$. Each time we will use the inner construction to add at most n_i wires. Note that when the number of wires we need to add is smaller than n_i, we can always pad 0.

1. Randomly sample $\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)} \in \{0,1\}^\lambda$. Let $\boldsymbol{u}^{(0)}$ and $\boldsymbol{u}^{(1)}$ denote $G_1(\boldsymbol{r}^{(0)})$ and $G_1(\boldsymbol{r}^{(1)})$ respectively. For all $i \in \{1,\ldots,|C|\}$, let W_i be the subset of $\{1,\ldots,\lambda\}$ such that $r_i^{(0)} = \bigoplus_{j\in W_i} u_j^{(0)}$ (and $r_i^{(1)} = \bigoplus_{j\in W_i} u_j^{(1)}$). Let $\boldsymbol{w} = (w_1,\ldots,w_{|C|})$ denote all wires in C.

2. For each input wire w_i, use the inner construction to compute $u_i^{(0)} \oplus w_i$ by $w_i \oplus (\bigoplus_{j\in W_i} r_j^{(0)})$ and $u_i^{(1)} \oplus w_i$ by $w_i \oplus (\bigoplus_{j\in W_i} r_j^{(1)})$. We view $\{r_j^{(0)}\}_{j\in W_i} \cup \{u_i^{(0)} \oplus w_i\}$ as an additive sharing $\langle w_i\rangle_0$ and $\{r_j^{(1)}\}_{j\in W_i} \cup \{u_i^{(1)} \oplus w_i\}$ as an additive sharing $\langle w_i\rangle_1$.

3. For each random wire w_i, randomly sample a bit as $u_i^{(0)} \oplus w_i$. Then use the inner construction to compute $u_i^{(1)} \oplus w_i = (u_i^{(0)} \oplus w_i) \oplus (\bigoplus_{j\in W_i} r_j^{(0)}) \oplus (\bigoplus_{j\in W_i} r_j^{(1)})$. We view $\{r_j^{(0)}\}_{j\in W_i} \cup \{u_i^{(0)} \oplus w_i\}$ as an additive sharing $\langle w_i\rangle_0$ and $\{r_j^{(1)}\}_{j\in W_i} \cup \{u_i^{(1)} \oplus w_i\}$ as an additive sharing $\langle w_i\rangle_1$.

4. For each addition gate with input wires w_i, w_j and output wire w_k, suppose we have computed $\langle w_i\rangle_0, \langle w_j\rangle_0$ and $\langle w_i\rangle_1, \langle w_j\rangle_1$. Use the inner construction to compute $u_k^{(0)} \oplus w_k$ and $u_k^{(1)} \oplus w_k$ and obtain $\langle w_k\rangle_0$ and $\langle w_k\rangle_1$.

5. For each multiplication gate with input wires w_i, w_j and output wire w_k, suppose we have computed $\langle w_i\rangle_0, \langle w_j\rangle_0$ and $\langle w_i\rangle_1, \langle w_j\rangle_1$. Compute $w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$ for all $\ell_1 \in \{0,1,\ldots|W_i|\}, \ell_2 \in \{0,1,\ldots,|W_j|\}$. Here $w_{i,0,\ell_1}$ is the ℓ_1-th share of $\langle w_i\rangle_0$ and $w_{j,1,\ell_2}$ is the ℓ_2-th share of $\langle w_j\rangle_1$. Note that $w_k = \bigoplus_{\ell_1,\ell_2} w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$. Use the inner construction to compute $u_k^{(0)} \oplus w_k$ and $u_k^{(1)} \oplus w_k$ to obtain $\langle w_k\rangle_0$ and $\langle w_k\rangle_1$.

6. For each output wire with additive sharing $\langle w_i\rangle_0$ and $\langle w_i\rangle_1$. Use the inner construction to compute w_i.

The randomness complexity of the outer construction is $O(\lambda)$ plus the randomness complexity of the inner construction and the randomness complexity of the unprotected circuit C. Note that the outer construction uses the inner construction to compute $m = O(|C|)$ additions, each with input size $n_i = O(\lambda^2)$. As for the size of λ, the output sizes of G_0 and G_1 are both $O(|C|)$. Therefore, we may set $\lambda = O(t\kappa \log |C|)$. Thus, the total randomness complexity is $O(t^5\kappa^5 \log^3 |C|) = \mathrm{poly}(\kappa, \log |C|)$ plus the randomness complexity of the unprotected circuit C.

6.3 Security Analysis

Recall that a D1-leakage query L is defined by two subsets S_1, S_2 of the input bits such that $L(\boldsymbol{x}) = (\bigvee_{i\in S_0} x_i) \vee (\bigvee_{i\in S_1}(x_i \oplus 1))$. We say a wire w is touched by L if $w \in S_1 \cup S_2$.

We first give the construction of the simulator $(\mathtt{Sim}_1, \mathtt{Sim}_2)$. Let $L_1,\ldots,L_t$ be the input t D1-leakage queries. Note that all wires in the outer construction are either input or output wires of the inner construction. We first reduce these t D1-leakage queries to those applied only on the wires in the outer construction.

Construction of $\mathtt{Sim}_1$. For each D1-leakage query $L \in \{L_1, \ldots, L_t\}$, if L touches wires in the computation of $\boldsymbol{\nu}^{(i,j)} = G_0(\boldsymbol{r}^{(i,j)})$ for at least κ different indices j, replace L by a constant function that always output 1. After the replacement, there exists an index $j^\star$ such that any wire in the computation of $\boldsymbol{\nu}^{(i,j^\star)} = G_0(\boldsymbol{r}^{(i,j^\star)})$ for all $i \in \{1, \ldots, \mathtt{RC}\}$ is not touched by any leakage function in $\{L_1, \ldots, L_t\}$.

$\mathtt{Sim}_1$ randomly samples $\boldsymbol{r}^{(i,j)}$ for all i and $j \neq j^\star$ and computes all wires in the computation of $\boldsymbol{\nu}^{(i,j)} = G_0(\boldsymbol{r}^{(i,j)})$. Now for all $i \in \{1, \ldots, \mathtt{RC}\}$, each wire in the computation of $\boldsymbol{\nu}^{(i)}$ is a random variable that only depends on a single bit of $\boldsymbol{\nu}^{(i)}$. $\mathtt{Sim}_1$ transforms $L_1, \ldots, L_t$ to t D1-leakage queries applied only on the t-D1-tolerant circuits for addition functions by representing each wire in the computation of $\boldsymbol{\nu}^{(i)}$ as a random variable that only depends on a single bit of $\boldsymbol{\nu}^{(\boldsymbol{i})}$. The new leakage functions are denoted by $L'_1, \ldots, L'_t$.

For each D1-leakage query $L' \in \{L'_1, \ldots, L'_t\}$, if L' touches *the inner wires* (i.e., wires that are not input and output wires) of at least κ t-D1-tolerant circuits for addition functions, replace L' by a constant function that always output 1. After the replacement, each leakage function touches the inner wires of at most κ t-D1-tolerant circuits. Without loss of generality, assume these circuits are just the first $t\kappa$ circuits $\hat{C}_1, \ldots, \hat{C}_{t\kappa}$.

For each D1-leakage query $L' \in \{L'_1, \ldots, L'_t\}$ and for each circuit $\hat{C} \in \{\hat{C}_1, \ldots, \hat{C}_{t\kappa}\}$, if L' touches the j-th share of some additive sharing $\langle x \rangle$ for at least κ different indices $j \in \{0, \ldots, t\kappa\}$, replace L' by a constant function that always output 1. After the replacement, for each $\hat{C}_\ell$, there exists an index $j^\star_\ell$ such that for any additive sharing $\langle x \rangle$ in $\hat{C}_\ell$, the $j^\star_\ell$-th share is not touched by any leakage function. Now we follow the strategy in [IS24] by representing every wire (except the $j^\star_\ell$-th share of each additive sharing) in $\hat{C}_\ell$ as a variable that only depends on a single input or output wire:

- For each input x_i, for all $j \neq j^\star_\ell$, sample a random bit as the j-th share of $\langle x_i \rangle$. Then for each additive sharing $\langle x \rangle$, compute the j-th share following the circuit.
- For each input x_i, we have generated all shares of $\langle x_i \rangle$ except the $j^\star_\ell$-th share. Now we generate the inner wire when computing $x_{i,t\kappa}$, which is in the form of $x_i \oplus (\bigoplus_{k=0}^{j} x_{i,k})$ for all $j \in \{0, \ldots, t\kappa - 1\}$.

  For $j < j^\star_\ell$, we have generated $x_{i,k}$ for all $k \leq j$. Then the inner wire $x_i \oplus (\bigoplus_{k=0}^{j} x_{i,k})$ is a variable that only depends on x_i.

  For $j \geq j^\star_\ell$, we have generated $x_{i,k}$ for all $k > j$. Compute $x_i \oplus (\bigoplus_{k=0}^{j} x_{i,k})$ by $x_{i,t\kappa} \oplus \ldots \oplus x_{i,j+1}$.
- For the output y, we have generated all shares of $\langle y \rangle$ except the $j^\star_\ell$-th share. Now we generate the inner wire when computing y, which is in the form of $y_{t\kappa} \oplus \ldots \oplus y_j$ for all $j \in \{0, \ldots, t\kappa - 1\}$.

  For $j \leq j^\star_\ell$, we have generated y_k for all $k < j$. Compute $y_{t\kappa} \oplus \ldots \oplus y_j$ by $y \oplus (\bigoplus_{k=0}^{j} y_k)$, which is a variable that only depends on y.

  For $j > j^\star_\ell$, we have generated y_k for all $k \geq j$. Directly compute $y_{t\kappa} \oplus \ldots \oplus y_j$.

$\mathtt{Sim}_1$ transforms $L'_1, \ldots, L'_t$ to t D1-leakage queries applied only on the input and output of $\hat{C}_1, \ldots, \hat{C}_{t\kappa}$ by representing each inner wire touched by any of $L' \in \{L'_1, \ldots, L'_t\}$ as a random variable that only depends on a single input or output wire as described above. The new leakage functions are denoted by $L''_1, \ldots, L''_t$. Note that the wires touched by $L''_1, \ldots, L''_t$ are also wires in the outer construction.

In the outer construction, $\mathtt{Sim}_1$ sets $\boldsymbol{w}$ to be an all-0 vector of size $|C|$. $\mathtt{Sim}_1$ randomly samples $\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)}$ and computes $\boldsymbol{u}^{(0)} = G_1(\boldsymbol{r}^{(0)}), \boldsymbol{u}^{(1)} = G_1(\boldsymbol{r}^{(1)})$. Then $\mathtt{Sim}_1$ computes $\langle w_i \rangle_0, \langle w_i \rangle_1$ for all

i. For each multiplication gate, $\mathtt{Sim}_1$ also computes $w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$ for all $\ell_1 \in \{0, \ldots, |W_i|\}$ and $\ell_2 \in \{0, \ldots, |W_j|\}$ in clear. Note that we have computed all wires except the input and output wires in the outer construction in clear. $\mathtt{Sim}_1$ transforms $L''_1, \ldots, L''_t$ to t D1-leakage queries applied only on the input and output of C. The new leakage functions are denoted by $\tilde{L}_1, \ldots, \tilde{L}_t$.

$\mathtt{Sim}_1$ outputs $\tilde{L}_1, \ldots, \tilde{L}_t$ and $\mathtt{st} = \bot$.

Construction of $\mathtt{Sim}_2$. For $\mathtt{Sim}_2$, it obtains the t leakage results of $\tilde{L}_1, \ldots, \tilde{L}_t$ applied on the input and output of C and output the leakage results directly.

Hybrid Analysis. Consider the following hybrids.

$\mathbf{Hybrid}_0$: In the initial hybrid, we consider the leakage in the real world. I.e., we first evaluate the t-D1-tolerant circuit faithfully and compute the result of the t D1-leakage queries $L_1, \ldots, L_t$. The output is the joint distribution of the leakage results together with the circuit output.

$\mathbf{Hybrid}_1$: In this hybrid, we apply the first replacement of leakage functions. I.e., for each $L \in \{L_1, \ldots, L_t\}$, if L touches wires in the computation of $\boldsymbol{\nu}^{(i,j)} = G_0(\boldsymbol{r}^{(i,j)})$ for at least κ different indices j, replace L by a constant function that always output 1. We argue that $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_0$ are statistically close.

First note that G_0 is implemented by a linear circuit. Then each wire in the computation of G_0 is a linear combination of its input bits. Since the input of G_0 is a random string, each wire is uniformly distributed. If L touches wires in the computation of $\boldsymbol{\nu}^{(i,j)} = G_0(\boldsymbol{r}^{(i,j)})$ for at least κ different indices j, then the κ wires, one from each circuit of $G_0(\boldsymbol{r}^{(i,j)})$, are uniformly random. The probability that the output of L is 0 is at most $2^{-\kappa}$. Since this holds for every L that is replaced, the statistical distance between $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_0$ is at most $t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_2$: In this hybrid, we transform $L_1, \ldots, L_t$ to $L'_1, \ldots, L'_t$ as described above. This does not change the distribution.

$\mathbf{Hybrid}_3$: In this hybrid, we apply the second replacement of leakage functions. I.e., for each $L' \in \{L'_1, \ldots, L'_t\}$, if L' touches *the inner wires* (i.e., wires that are not input and output wires) of at least κ t-D1-tolerant circuits for addition functions, replace L' by a constant function that always output 1.

We first argue that when the random tape is uniformly distributed, each inner wire in the t-D1-tolerant circuit for addition, denoted by $\hat{C}$, is uniformly distributed. By construction $\hat{C}$ is a linear circuit. Thus each inner wire is a linear combination of its input bits and internal random bits. Also note that, except the input and output bits, each wire is masked by at least one bit of the internal randomness. Thus, when the random tape is uniformly distributed, each inner wire is uniformly distributed.

Since we use $t\kappa$-wise independent PRG to generate the internal randomness of all t-D1-tolerant circuits for addition functions, any κ circuits use uniform random tape. Then L' touches at least κ random wires. The probability that the output of L' is 0 is at most $2^{-\kappa}$. Since this holds for every L' that is replaced, the statistical distance between $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_2$ is at most $t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_4$: In this hybrid, for each t-D1-tolerant circuit for addition, we change to use uniform random tape rather than computing it from $t\kappa$-wise independent PRG. First note that the circuit output is independent of the random tapes of the t-D1-tolerant circuits for addition functions. In $\mathbf{Hybrid}_3$, after replacement, the number of t-D1-tolerant circuits such that some wire of each circuit is touched by some leakage in $\{L'_1, \ldots, L'_t\}$ is bounded by $t\kappa$. Since in $\mathbf{Hybrid}_3$, the random tape

is generated from $t\kappa$-wise independent PRG, the random tapes of these $t\kappa$ t-D1-tolerant circuits are uniformly distributed. Thus $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_3$ are identically distributed.

$\mathbf{Hybrid}_5$: In this hybrid, we apply the third replacement of leakage functions. I.e., for each $L' \in \{L'_1, \ldots, L'_t\}$ and for each circuit $\hat{C} \in \{\hat{C}_1, \ldots, \hat{C}_{t\kappa}\}$, if L' touches the j-th share of some additive sharing $\langle x \rangle$ for at least κ different indices $j \in \{0, \ldots, t\kappa\}$, replace L' by a constant function that always output 1.

By the property of additive sharings, in the above case, L' touches at least κ random wires. The probability that the output of L' is 0 is at most $2^{-\kappa}$. Since this holds for every L' that is replaced, the statistical distance between $\mathbf{Hybrid}_5$ and $\mathbf{Hybrid}_4$ is at most $t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_6$: In this hybrid, we change the way of computing the wires in the inner construction. We first compute all wires in the outer construction. Then for each $\hat{C}_\ell$, we compute the inner wires except the $j^\star_\ell$-th share of each additive sharing in$\hat{C}_\ell$ in the way as described above. Note that the only difference is that for each additive sharing of an input wire in $\hat{C}_\ell$, instead of generating the first $t\kappa$ shares randomly and computing the last share, we generate all shares randomly except the $j^\star_\ell$-th share. By the property of additive sharings, $\mathbf{Hybrid}_6$ and $\mathbf{Hybrid}_5$ are identically distributed.

$\mathbf{Hybrid}_7$: In this hybrid, we transform $L'_1, \ldots, L'_t$ to $L''_1, \ldots, L''_t$ as described above. This does not change the distribution. Note that $L''_1, \ldots, L''_t$ are applied only on the wires in the outer construction.

$\mathbf{Hybrid}_8$: In this hybrid, we change the way of computing the wires in the outer construction. We first compute all wires in the original circuit C, denoted by $\boldsymbol{w}$. Then we compute $\boldsymbol{u}^{(0)} = G_1(\boldsymbol{r}^{(0)}), \boldsymbol{u}^{(1)} = G_1(\boldsymbol{r}^{(1)})$ and $\boldsymbol{u^{(0)}} \oplus \boldsymbol{w}, \boldsymbol{u^{(1)}} \oplus \boldsymbol{w}$. Finally, for each multiplication gate, we compute $w_{i,0,\ell_1} \cdot w_{j,1,\ell_2}$ for all $\ell_1 \in \{0, \ldots, |W_i|\}$ and $\ell_2 \in \{0, \ldots, |W_j|\}$. $\mathbf{Hybrid}_8$ and $\mathbf{Hybrid}_7$ are identically distributed.

Note that if we fix $\boldsymbol{w}, \boldsymbol{u}^{(1)}$, then $L''_1, \ldots, L''_t$ can be viewed as leakage queries applied on $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \boldsymbol{w})$.

$\mathbf{Hybrid}_9$: In this hybrid, we change $\boldsymbol{u^{(0)}} \oplus \boldsymbol{w}$ by $\boldsymbol{u^{(0)}} \oplus \mathbf{0}$ and compute the rest of wires accordingly. Given $\boldsymbol{w}, \boldsymbol{u}^{(1)}$, the only difference between $\mathbf{Hybrid}_8$ and $\mathbf{Hybrid}_9$ is that $L''_1, \ldots, L''_t$ are applied on $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \boldsymbol{w})$ in $\mathbf{Hybrid}_8$ while they are applied on $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \mathbf{0})$ in $\mathbf{Hybrid}_9$.

We show that $\mathbf{Hybrid}_9$ is statistically close to $\mathbf{Hybrid}_8$. Consider two intermediate hybrids:

$\mathbf{Hybrid}_{8,1}$: In this hybrid, for each $L'' \in \{L''_1, \ldots, L''_t\}$, if L'' touches at least κ wires in $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \boldsymbol{w})$, we replace L'' by a constant function that always output 1. Since G_1 is a $t\kappa$-wise independent PRG, the output of L'' is 1 with probability at least $1 - 2^{-\kappa}$. Then the statistical distance between $\mathbf{Hybrid}_{8,1}$ and $\mathbf{Hybrid}_8$ is at most $t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_{8,2}$: In this hybrid, we change from $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \boldsymbol{w})$ to $(\boldsymbol{r}^{(0)}, \boldsymbol{u^{(0)}} \oplus \mathbf{0})$. Note that all leakage functions after replacement touch at most $t\kappa$ wires. Since G_1 is a strong $t\kappa$-wise independent PRG, any $t\kappa$ wires are uniformly distributed. Thus, $\mathbf{Hybrid}_{8,2}$ and $\mathbf{Hybrid}_{8,1}$ are identically distributed.

Also note that the difference between $\mathbf{Hybrid}_{8,2}$ and $\mathbf{Hybrid}_9$ is that we use the original leakage functions before replacement in $\mathbf{Hybrid}_{8,1}$. By the same analysis, the statistical distance between $\mathbf{Hybrid}_9$ and $\mathbf{Hybrid}_{8,2}$ is at most $t \cdot 2^{-\kappa}$.

In summary, the statistical distance between $\mathbf{Hybrid}_9$ and $\mathbf{Hybrid}_8$ is at most $2t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_{10}$: In this hybrid, we change $\boldsymbol{u^{(1)}} \oplus \boldsymbol{w}$ by $\boldsymbol{u^{(1)}} \oplus \mathbf{0}$. By the same analysis, the statistical distance between $\mathbf{Hybrid}_{10}$ and $\mathbf{Hybrid}_9$ is at most $2t \cdot 2^{-\kappa}$.

$\mathbf{Hybrid}_{11}$: In this hybrid, we transform $L''_1, \ldots, L''_t$ to $\tilde{L}_1, \ldots, \tilde{L}_t$ as described above. This does not change the distribution. Note that $\tilde{L}_1, \ldots, \tilde{L}_t$ are applied only on the wires of the original

circuit. This corresponds to the ideal scenario.

Note that the statistical distance between the real scenario and the ideal scenario is bounded by $7t \cdot 2^{-\kappa}$. To achieve the statistical error ϵ, we set $\kappa = \log \frac{7t}{\epsilon} = \mathrm{poly}(t, \log(1/\epsilon))$. Recall that the randomness complexity of our construction is $\mathrm{poly}(\kappa, \log|C|)$ plus the randomness complexity of the unprotected circuit C. Taking $\kappa = \mathrm{poly}(t, \log(1/\epsilon))$, the randomness of our construction is $\mathrm{poly}(t, \log(1/\epsilon), \log|C|)$ plus the randomness complexity of the unprotected circuit C.

6.4 Deterministic Randomness Extractor for t-D1-Leakage Source

Finally, we give a concrete instantiation of Definition 12 for t-D1-leakage source. The construction is as follows: Let $\kappa = \log \frac{t}{\epsilon}$. The extractor $\mathtt{Ext}$ takes as input $I = t\kappa + 1$ bits and outputs the parity of the input bits. We show that this simple construction suffices.

To prove the security of $\mathtt{Ext}$, we first construct the simulator $\mathtt{Sim}$ required by Definition 12. Let $L_1, \ldots, L_t$ be the input t D1-leakage queries. For each $L \in \{L_1, \ldots, L_t\}$, if L touches at least κ input bits, we replace L by a constant function that always outputs 1. After the replacement, there exists an index $j \in \{0, 1, \ldots, t\kappa\}$ such that the j-th input bit is not touched by any leakage function. $\mathtt{Sim}$ randomly samples a random string as $r_0 \in \{0,1\}^I$ such that the parity of all bits of r_0 is 0. Then $\mathtt{Sim}$ sets r_1 to be r_0 except that the j-th bit of r_0 is flipped. $\mathtt{Sim}$ computes ℓ to be the leakage result by applying the leakage functions (after replacement) on r_0 and outputs (ℓ, r_0, r_1).

Now consider the following hybrids.

$\mathbf{Hybrid}_0$: In the first hybrid, we randomly sample $r \leftarrow \{0,1\}^I$ and outputs $(\{L_i(r)\}_{i=1}^t, \mathtt{Ext}(r), r)$. This corresponds to the LHS in Definition 12.

$\mathbf{Hybrid}_1$: In this hybrid, for each $L \in \{L_1, \ldots, L_t\}$, if L touches at least κ input bits, we replace L by a constant function that always outputs 1. Then we compute the output as $\mathbf{Hybrid}_0$ with the replaced leakage functions.

Note that if L touches at least κ input bits, when r is a random string of length I, the probability that the output of L is 0 is at most $2^{-\kappa}$. Thus the statistical distance between $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ is at most $t \cdot 2^{-\kappa}$.

Note that after replacement, there exists $j \in \{0, \ldots, t\kappa\}$ such that the j-th input bit is not touched by any leakage function.

$\mathbf{Hybrid}_2$: In this hybrid, for all $i \neq j$, we first sample a random bit as the i-th bit of the input bit. These $t\kappa$ input bits are sufficient to compute the leakage results $\{L_i(r)\}_{i=1}^t$. Then we randomly sample a bit b as the extraction result $\mathtt{Ext}(r)$. Since $\mathtt{Ext}$ simply computes the parity of all input bits, we compute the j-th bit of r such that $\mathtt{Ext}(r) = b$. The distribution of $\mathbf{Hybrid}_2$ is identical to that of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, after sampling a random bit as the i-th bit of the input bit for each $i \neq j$, we compute r_0 such that $\mathtt{Ext}(r_0) = 0$ and r_1 such that $\mathtt{Ext}(r_1) = 1$. Note that r_0 and r_1 only differ in the j-th bit. Then we randomly sample b and sets $r = r_b$. The distribution of $\mathbf{Hybrid}_3$ is identical to that of $\mathbf{Hybrid}_2$. Note that $\mathbf{Hybrid}_3$ is the RHS in Definition 12.

By setting $\kappa = \log \frac{t}{\epsilon}$, $\mathtt{Ext}$ is a deterministic randomness extractor for t-D1-leakage source with efficient reverse sampling and statistical error ϵ.

Now from Theorem 6, we have the following corollary.

Corollary 1. *Let κ denote the security parameter. Assume there is a pseudo-random generator $\mathtt{PRG}$ with seed length κ that is secure against adversaries with circuit size poly(κ). For every polynomial leakage bound t = poly(κ), there exists a computational t-D1-LRC compiler $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ such*

that for every stateful circuit C, the circuit C' generated by $\mathcal{T}_C(1^\kappa, C)$ does not require any fresh random bits beyond those embedded in the encoded initial state.

7 Instantiating Blueprint for Parity Leakage

In this section, we give a concrete instantiation for our compiler of deterministic stateful leakage resilient circuits for parity leakage. Concretely, the parity leakage class contains all leakage functions $L : \{0,1\}^n \to \{0,1\}$ defined by $S \subset \{1,\ldots,n\}$ such that $L(\boldsymbol{x}) = \bigoplus_{i\in S} x_i$.

For a leakage bound parameter t, we use t-parity leakage class to denote the t-leakage class of the parity leakage class. I.e., each leakage function corresponds to t parity queries. In the following, we use (t,ϵ)-parity tolerance to denote leakage tolerance against t-parity leakage with error ϵ. We first construct a (t,ϵ)-parity-tolerant circuit whose randomness complexity is poly-logarithmic in the circuit size. In Section 7.1, we give an overview of our construction. As [IS24], our construction is split in to two parts which are formally described in Section 7.2 and Section 7.3 respectively with security proof in Section 7.4. Finally, in Section 7.6, we construct a deterministic randomness extractor for t-parity-leakage source with efficient reverse sampling.

7.1 Overview of Our Construction

Our idea is to derandomize the t-parity-tolerant circuits constructed in [IS24]. At a high level, the construction in [IS24] is obtained by two steps. First, for an unprotected circuit $\tilde{C}$, the authors in [IS24] introduce the notion of (t,k,ϵ)-parity-to-probing circuit implementation of $\tilde{C}$, which is a tuple (I,C,O) satisfying the following requirements:

- Correctness: (I,C,O) computes the same function as $\tilde{C}$. I.e., $O(C(I(x))) = \tilde{C}(x)$.
- Trivial Parity Tolerance: The input encoder I and the output decoder O admit circuit implementations in which any inner wire is a linear combination of the input and output wires.
- Parity-to-Probing Security: Any t parities of wires in C can be simulated by k wires in $\tilde{C}$ (with negligible statistical error ϵ).

Hereafter, we omit the statistical error ϵ for simplicity. At a first glance, one may feel that a (t,k)-parity-to-probing circuit is weaker than a t-parity-resilient circuit since the latter requires that any t parities of wires in C should be independent of any wire in $\tilde{C}$. However, note that a (t,k)-parity-to-probing circuit requires that the input encoder and the output decoder to be trivially parity-tolerant whereas a t-parity-resilient circuit does not put any restriction on the input encoder and the output decoder. The authors in [IS24] show that a (t,k)-parity-to-probing circuit can be used to construct a t-parity-tolerant circuit in a black-box way. Such a result is not known from t-parity-resilient circuits.

Second, the authors in [IS24] construct a circuit compiler that transforms any unprotected circuit $\tilde{C}$ to a (t,k)-parity-to-probing circuit implementation of $\tilde{C}$.

Our construction is obtained by derandomizing each step in [IS24] separately. For simplicity, we focus on a single parity-leakage query in the overview.

7.1.1 Derandomization of Step 1

We first review how this is achieved in [IS24]. Let f be the function we want to compute. The function inputs are denoted by $\boldsymbol{x}$. Intuitively, the idea in [IS24] is to make use of a probing-tolerant circuit $\tilde{C}$ for f and then compile $\tilde{C}$ to a parity-to-probing circuit (I, C, O). In this way, any parity query to C is reduced to probing queries to $\tilde{C}$, which is then reduced to probing queries to the input and output of f. However, to build a parity-tolerant circuit, we need to reduce any parity query to C to a parity query, rather than probing queries, to the input and output of f. To achieve this, the authors in [IS24] first encode the input of f and then apply the above idea to compute an encoding of the output of f as follows.

1. *Input Encoding Phase:* $C^\star$ first encodes each bit of the input $\boldsymbol{x}$ of f by a random additive sharing with $k+1$ shares. Let $\hat{\boldsymbol{x}}$ denotes the encoding of $\boldsymbol{x}$.

2. *Computation Phase:* Then let $\hat{f}$ be the (randomized) function that maps an encoded input $\hat{\boldsymbol{x}}$ to a freshly encoded output $\hat{\boldsymbol{y}}$. Let $\tilde{C}$ be a k-probing-tolerant implementation of $\hat{f}$, and let (I, C, O) be a $(1, k)$-parity-to-probing circuit implementation of $\tilde{C}$. Then $C^\star$ runs (I, C, O) on $\hat{\boldsymbol{x}}$ and outputs $\hat{\boldsymbol{y}}$.

3. *Output Decoding Phase:* Finally, the circuit $C^\star$ decodes $\hat{\boldsymbol{y}}$ by computing the parity of each additive sharing and obtains $\boldsymbol{y}$.

To see why this construction works, consider the parity of a set W of wires in $C^\star$, denoted by $\texttt{parity}(W)$. We may divide W into three disjoint subsets $W = W_1 \cup W_2 \cup W_3$, where W_1 only contain wires in the Input Encoding Phase, W_2 only contain wires in the Computation Phase, and W_3 only contain wires in the Output Decoding Phase. First note that:

- For the Input Encoding Phase and the Output Decoding Phase, all operations are XOR operations. Each wire can be computed by parity of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$. Thus, it is sufficient to only consider the case where W_1 only contains wires in $\hat{\boldsymbol{x}}$ and W_3 only contains wires in $\hat{\boldsymbol{y}}$.

- For the Computation Phase, since I and O are trivially parity-tolerant, each inner wire can be computed by parity of the input and output of I and O. It is sufficient to only consider the case where W_2 does not contain any inner wires in I and O. Furthermore, since the input of I is $\hat{\boldsymbol{x}}$, which is covered by the Input Encoding Phase, and the output of O is $\hat{\boldsymbol{y}}$, which is covered by the Output Decoding Phase. We may further assume that W_2 only contain wires in C.

By the property of $(1, k)$-parity-to-probing circuits, $\texttt{parity}(W_2)$ can be simulated by k probings of $\tilde{C}$. Then by the property of k-probing-tolerant circuits, $\texttt{parity}(W_2)$ is further reduced to k probings of the input of $\tilde{C}$, i.e., $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$. Recall that $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ are additive sharings of $\boldsymbol{x}$ and $\boldsymbol{y}$ with $k+1$ shares. Thus, any k probings of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ are uniformly random. We can first sample random values for these k probings and use those random values to simulate $\texttt{parity}(W_2)$. Then we can represent each wire of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ as a linear combination of $\boldsymbol{x}$ and $\boldsymbol{y}$. In this way $\texttt{parity}(W_1)$ and $\texttt{parity}(W_3)$ are naturally parity queries to $\boldsymbol{x}$ and $\boldsymbol{y}$. As a result, we can reduce $\texttt{parity}(W)$ to a parity query to the input and output of f.

Our Solutions. We note that the above construction requires randomness in three places: (1) Encoding the input $\boldsymbol{x}$ (and output $\boldsymbol{y}$), (2) Constructing k-probing-tolerant circuit $\tilde{C}$, and (3) Constructing $(1,k)$-parity-to-probing circuit implementation of $\tilde{C}$. We will show how to derandomize (1) and (2). Derandomization of (3) will be the main task of Step 2.

For (2), we simply use a randomness-efficient k-probing tolerant circuit such as the constructions in [IKL$^+$13, CGZ20]. For (1), our idea is to use a strong and linear k-wise independent PRG G to encode $\boldsymbol{x}$ (and $\boldsymbol{y}$). To be more concrete, we set $\hat{\boldsymbol{x}}$ to be $(\boldsymbol{r}, G(\boldsymbol{r}) \oplus \boldsymbol{x})$, where $\boldsymbol{r}$ is a uniform string. The intuition of the security is as follows:

- Since G is linear, any wire in the Input Encoding Phase can be computed by a parity of $\hat{\boldsymbol{x}}$. Thus, similarly to the above analysis, it is sufficient to only consider the case where W_1 only contains wires in $\hat{\boldsymbol{x}}$.
- Since G is a strong k-wise independent PRG, after reducing $\texttt{parity}(W_2)$ to k probings of $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$, we may continue to sample random values as these k probings due to the k-wise independence of $(\boldsymbol{r}, G(\boldsymbol{r}) \oplus \boldsymbol{x})$. After fixing k bits in $\hat{\boldsymbol{x}}$, we still can represent each bit of $\hat{\boldsymbol{x}}$ as a linear combination of $\boldsymbol{x}$ just as the case of using additive sharings. Thus, following the same argument as above, $\texttt{parity}(W)$ can be reduced to a parity query to the input and output of f.

7.1.2 Derandomization of Step 2

In [IS24], the second step is to construct (t,k)-parity-to-probing circuits. The authors in [IS24] observe that the construction in [GIM$^+$16], which targets for parity-resilient circuits, implicitly gives a (t,k)-parity-to-probing circuit with the aid of a secure `NAND` gadget. The main contribution of [IS24] lies in realizing `NAND` gadgets.

To better expose our idea, in the overview, we focus on the parity-to-probing circuit from [GIM$^+$16] that requires secure `NAND` gadgets and we focus on a single parity query. Let $\tilde{C}$ be the unprotected circuit. The high-level idea of [GIM$^+$16] is to compute a *small-bias encoding* for each wire in $\tilde{C}$. Very informally, an ϵ-bias encoding satisfies that the parity of any non-empty subset of the encodings has bias (towards a uniform bit) at most ϵ. We will use the following concrete 1/2-bias encoding scheme (Figure 2). The construction of (I, C, O) is as follows:

- I takes $\boldsymbol{x} \in \{0,1\}^{n_i}$ as input and computes a small-bias encoding for each input bit: $\{\texttt{Enc}(x_i)\}_{i=1}^{n_i}$.
- C takes $\{\texttt{Enc}(x_i)\}_{i=1}^{n_i}$ as input. Then C computes a small-bias encoding for each wire value in $\tilde{C}$. Concretely, for each `NAND` gate in $\tilde{C}$ with input x_0, x_1, C uses a secure `NAND` gadget to compute $\texttt{Enc}(x_0 \texttt{ NAND } x_1)$ from $\texttt{Enc}(x_0), \texttt{Enc}(x_1)$.
- O takes $\{\texttt{Enc}(y_i)\}_{i=1}^{n_o}$ as input and outputs $\{y_i = \texttt{Dec}(\texttt{Enc}(y_i))\}_{i=1}^{n_o}$.

One can verify that I and O are trivially parity-tolerant (See more details in Section 7.4.1 or [IS24]).

Now consider a parity query of the wires in C, which consists of small-bias encoding of each wire in $\tilde{C}$. If the parity query touches at most k encodings, then the parity result can be simulated by learning k wires in $\tilde{C}$. Otherwise, the parity result is statistically close to a uniform bit with bias at least 2^{-k}. In this case, the parity result can be simulated by a random bit. This shows that the above construction is a $(1,k)$-parity-to-probing circuit.

To be more concrete, let $\boldsymbol{w}$ denote all wire values in $\tilde{C}$. Let $\boldsymbol{u}, \boldsymbol{v}$ be random strings of length $|\tilde{C}|$ that are used to compute small-bias encodings for $\boldsymbol{w}$. Then all wires in C can be represented as

1/2-**Bias Encoding Scheme** $(\mathtt{Enc}, \mathtt{Dec})$

$$\begin{aligned} \mathtt{Enc}(x; r_0, r_1) &= (r_0, r_1, r_0 \cdot r_1 \oplus x) \\ \mathtt{Dec}(\hat{x}_0, \hat{x}_1, \hat{x_2}) &= \hat{x}_0 \cdot \hat{x}_1 \oplus \hat{x}_2. \end{aligned}$$

Figure 2: An Example of Small-bias Encoding Scheme

$(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w})$, where $*$ denotes the coordinate-wise multiplication operation. Consider the parity of a subset W of wires in C, i.e., $\mathtt{parity}(W)$. Then there exists $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3 \subset \{1, \ldots, |\tilde{C}|\}$ such that

$$\mathtt{parity}(W) = (\bigoplus_{i \in \mathcal{I}_1} u_i) \oplus (\bigoplus_{i \in \mathcal{I}_2} v_i) \oplus (\bigoplus_{i \in \mathcal{I}_3} (u_i v_i \oplus w_i)).$$

Then the security of C essentially says the following:

- If $|\mathcal{I}_3| \leq k$, $\mathtt{parity}(W)$ can be perfectly simulated by $\{w_i\}_{i \in \mathcal{I}_3}$.
- If $|\mathcal{I}_3| \geq k$, $\mathtt{parity}(W)$ is statistically close to a uniform bit.

Our Solutions. To derandomize the above construction, we want to use pseudo-random sources to replace $\boldsymbol{u}$ and $\boldsymbol{v}$. When $|\mathcal{I}_3| \leq k$, it is always possible to simulate $\mathtt{parity}(W)$ with at most k probings of $\tilde{C}$. In the following, we only focus on the case $|\mathcal{I}_3| \geq k$. We first observe that $\mathtt{parity}(W)$ is linear in $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively. Thus, it is sufficient to use parity-resilient pseudo-random sources for $\boldsymbol{u}$ and $\boldsymbol{v}$. However, there are two difficulties:

- We need the circuit that computes a parity-resilient pseudorandom source to be parity-tolerant. This is because an adversary can also attack the inner wires of the circuit.
- We need this parity-tolerant circuit to be randomness-efficient. Recall that the goal is to generate a pseudo-random source with small amount of random bits. If the randomness complexity of the circuit is proportional to the circuit size, then we do not achieve any saving.

Derandomization of $\boldsymbol{v}$. In [IS24], the authors observe that when $|\mathcal{I}_3| \geq k$, $\mathtt{parity}(W)$ is statistically close to a uniform bit even given $\boldsymbol{v}$. Intuitively, this is because for a fixed $\boldsymbol{v}$, $\bigoplus_{i \in \mathcal{I}_3} w_i$ is masked by a linear combination of bits in $\boldsymbol{u}$. As long as one coefficient of $\boldsymbol{u}$ is 1, $\mathtt{parity}(W)$ is a uniform bit. When $\boldsymbol{v}$ is randomly sampled, $\bigoplus_{i \in \mathcal{I}_3} u_i v_i$ is a random linear combination of $\{u_i\}_{i \in \mathcal{I}_3}$. $\mathtt{parity}(W)$ is not uniformly random if and only if $(\bigoplus_{i \in \mathcal{I}_1} u_i) \oplus (\bigoplus_{i \in \mathcal{I}_3} u_i v_i) = 0$, which happens with probability at most 2^{-k}.

We note that this observation also holds when $\boldsymbol{v}$ is a parity-resilient source: In the above analysis, we only utilize the property that any k bits of $\boldsymbol{v}$ are uniformly random. For a parity-resilient source, one can prove that any k bits are almost uniformly random, which is sufficient for the above analysis. Thus, for a parity-resilient source $\boldsymbol{v}$, $\mathtt{parity}(W)$ is statistically close to a uniform bit even given $\boldsymbol{v}$. An immediate corollary of this observation is that, for any function $f : \{0,1\}^{|\tilde{C}|} \to \{0,1\}$, $\mathtt{parity}(W) + f(\boldsymbol{v})$ is statistically close to a uniform bit.

Our idea is to compute an unprotected parity-resilient PRG G and use its output as $\boldsymbol{v}$. Note that the parity of any subset of wires in G can be computed by some function f on $\boldsymbol{v}$. By the above corollary, as long as $|\mathcal{I}_3| \geq k$, $\texttt{parity}(W) + f(\boldsymbol{v})$ is statistically close to a uniform bit. Thus, being able to attack the inner wires of G does not give any additional advantage to the adversary. In this way, we can generate $\boldsymbol{v}$ from a short random seed of G.

Derandomization of $\boldsymbol{u}$. A similar trick unfortunately does not work for $\boldsymbol{u}$. Our idea is to find a pseudo-random source for $\boldsymbol{u}$ that can be generated by a *linear circuit*. Since for a linear circuit, any inner wire can be computed by a linear combination of its output[4], being able to attack inner wires does not give the adversary any additional advantage. This allows us to not worry about the parity tolerance of the circuit. Unfortunately, a parity-resilient pseudo-random source cannot be generated by a linear circuit. In the following, we try to determine a sufficient condition for $\boldsymbol{u}$.

Recall the core property we need for $\boldsymbol{u}$: for all set $\mathcal{I}_1, \mathcal{I}_3$ such that $|\mathcal{I}_3| \geq k$, when $\boldsymbol{v}$ is generated by a parity-resilient PRG, $(\bigoplus_{i\in\mathcal{I}_1} u_i) \oplus (\bigoplus_{i\in\mathcal{I}_3} u_i v_i)$ is statistically close to a uniform bit given $\boldsymbol{v}$. Now suppose $\boldsymbol{u}$ is computed by a linear circuit G' with input $\boldsymbol{r}$ of size λ. Then every u_i is a linear function on $\boldsymbol{r}$. Let $\boldsymbol{A}$ be a matrix of size $|\mathcal{I}_3| \times \lambda$ such that $(u_i)_{i\in\mathcal{I}_3} = \boldsymbol{A} \cdot \boldsymbol{r}$, and let $\boldsymbol{b}$ be a vector of size λ such that $\bigoplus_{i\in\mathcal{I}_1} u_i = \langle \boldsymbol{b}, \boldsymbol{r}\rangle$. Then we have

$$\begin{aligned}
& (\bigoplus_{i\in\mathcal{I}_1} u_i) \oplus (\bigoplus_{i\in\mathcal{I}_3} u_i v_i) \\
= \;& \langle \boldsymbol{b}, \boldsymbol{r}\rangle \oplus \langle (u_i)_{i\in\mathcal{I}_3}, (v_i)_{i\in\mathcal{I}_3}\rangle \\
= \;& \boldsymbol{r}^{\mathrm{T}} \cdot \boldsymbol{b} \oplus \boldsymbol{r}^{\mathrm{T}} \cdot \boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}_3} \\
= \;& \boldsymbol{r}^{\mathrm{T}} \cdot (\boldsymbol{b} \oplus \boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}_3}).
\end{aligned}$$

We note that this is a linear combination of $\boldsymbol{r}$ and for each $j \in \{1, \ldots, \lambda\}$, the coefficient of r_j is $b_j \oplus \boldsymbol{A}_j^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}_3}$. We observe that

- $\boldsymbol{v}$ is computed by a parity-resilient PRG. Thus any (non-empty) parity of bits in $\boldsymbol{v}$ is statistically close to a uniform bit.
- $\boldsymbol{r}$ is a uniform string. Thus any non-zero linear combination of $\boldsymbol{r}$ is a uniform bit.

Thus, if we can ensure that when $|\mathcal{I}_3| \geq k$, with overwhelming probability, not all coefficients of $\boldsymbol{r}$ are 0, then we are done.

We note that a sufficient condition is to require that the column rank of $\boldsymbol{A}$ is at least k. This is because the j-th column of $\boldsymbol{A}$ corresponds to a linear query to $\boldsymbol{v}$ and its result XOR with b_j is used as the coefficient of r_j. If we have k linearly independent columns in $\boldsymbol{A}$, then these k query results would be statistically close to k random bits. Therefore, the probability that the coefficients of all r_j are 0 is negligible in k. Thus, we require that when $|\mathcal{I}_3| \geq k$, the corresponding matrix $\boldsymbol{A}$ should have column rank at least k, or equivalently, row rank at least k. This implies that at least k bits of $(u_i)_{i\in\mathcal{I}_3} = \boldsymbol{A} \cdot \boldsymbol{r}$ are linearly independent. Note that a k-wise independent random source is sufficient! Thus, we will use a linear k-wise independent PRG to generate $\boldsymbol{u}$.

[4]If this is not the case, it implies that some input bit is independent of the output, which means that this input bit is redundant. We may simply set this input wire to be 0 without affecting the output distribution.

Summary. In summary, to derandomize Step 2, our idea is to use an unprotected linear k-wise independent PRG to generate $\boldsymbol{u}$ and use an unprotected parity-resilient PRG to generate $\boldsymbol{v}$. Intuitively, the security follows from the following three facts:

- The inner-product between a k-wise independent source $\boldsymbol{u}$ and a parity-resilient source $\boldsymbol{v}$ is statistically close to a uniform bit.
- The above fact holds even if $\boldsymbol{v}$ is given. Thus, being able to attack the inner wires of the generation of $\boldsymbol{v}$ does not give the adversary any additional advantage.
- When using a linear k-wise independent PRG to generate $\boldsymbol{u}$, any inner wire is a linear combination of $\boldsymbol{u}$. Under parity attacks, being able to attack the inner wires of the generation of $\boldsymbol{u}$ does not give the adversary any additional advantage.

However, this is not the end of the story. As we mentioned in the beginning, the main contribution of [IS24] is to instantiate the secure `NAND` gadgets for the construction in [GIM+16]. We have to also derandomize the construction for `NAND` gadgets in [IS24].

In a nutshell, we continue to rely on the above three facts to generate random bits required by the construction in [IS24]. We manage to show that any parity query to the construction can be reduced to the case where

- Either the parity query touches at most k wires of the underlying circuit $\tilde{C}$, in which case it can be perfectly simulated by k probings of $\tilde{C}$,
- Or the parity query is masked by an inner-product between a k-wise independent source $\boldsymbol{u}$ and a parity-resilient source $\boldsymbol{v}$ so that the leakage result is statistically close to a uniform bit.

7.2 Randomness-Efficient Compiler from Parity-to-Probing to Parity-Tolerance

We follow the definitions of trivial parity tolerance and parity-to-probing circuits in [IS24].

Definition 14 (Trivial Parity Tolerance [IS24])**.** *We say a randomized circuit C is trivially parity-tolerant if every wire of C can be expressed as a linear combination of its inputs and outputs. We say a function $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$ is trivially parity-tolerant if it admits a trivially parity-tolerant implementation.*

Definition 15 (Parity-to-Probing Circuits [IS24])**.** *For a (possibly randomized) circuit $\tilde{C} : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$, a parity-to-probing circuit implementation of $\tilde{C}$ is a tuple (I, C, O) with the same syntax as leakage-resilient circuits defined in Definition 3 such that*

- *Correctness: For all input $\boldsymbol{x} \in \{0,1\}^{n_i}$, the following distributions are identically distributed:*

$$\tilde{C}(\boldsymbol{x}) \equiv O(C(I(\boldsymbol{x}))).$$

- *Trivial Parity Tolerance: The input encoder I and the output decoder O are trivially parity-tolerant.*
- *Parity-to-Probing Security: There exists a simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ with the following syntax:*

– $\mathtt{Sim}_1$ *takes as input* t *sets* $W_1, \ldots, W_t$ *of wires in* C *and outputs a state* $\mathtt{st}$ *as well as a set* V *of at most* k *wires in* $\tilde{C}$*,*

– $\mathtt{Sim}_2$ *takes as input a state* $\mathtt{st}$ *and the wire values in* V *and outputs* t *bits* $(b_1, \ldots, b_t)$*,*

such that for all input $\boldsymbol{x} \in \{0,1\}^{n_i}$ *and for all sets* $W_1, \ldots, W_t$ *of wires in* C*, the following two distributions are statistically close with error* ϵ*:*

$$\begin{aligned} & (\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), O(C(I(\boldsymbol{x})))) \\ \approx_\epsilon \quad & (\mathtt{Sim}_2(\mathtt{st}, \mathsf{val}(V)), \tilde{C}(\boldsymbol{x})) : (\mathtt{st}, V) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t). \end{aligned}$$

Here $\mathsf{val}(V)$ *denotes the values of the wires of* $\tilde{C}$ *in* V*.*

Let $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$ be the input function. Let $n = \max\{n_i, n_o\}$. The compiler in [IS24] relies on the following tools:

- A compiler that transforms any function g to a k-probing-tolerant implementation of g.
- A compiler that transforms any circuit $\tilde{C}$ to a (t, k, ϵ)-parity-to-probing implementation of $\tilde{C}$.
- An encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ with the following properties:
  - $\mathtt{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ is a randomized function such that (1) $\mathtt{Enc}$ outputs its random tape, (2) each output bit is a linear combination of its input and random tape, and (3) when the random tape of $\mathtt{Enc}$ are sampled uniformly, any k bits of the output of $\mathtt{Enc}$ are independent and uniformly distributed.
  - $\mathtt{Dec} : \{0,1\}^{\hat{n}} \to \{0,1\}^n$ is a deterministic function such that (1) for all $\boldsymbol{x} \in \{0,1\}^n$, $\Pr[\mathtt{Dec}(\mathtt{Enc}(\boldsymbol{x})) = \boldsymbol{x}] = 1$, where the probability is over the randomness of $\mathtt{Enc}$, and (2) each output bit of $\mathtt{Dec}$ is a linear combination of its input.

We give the construction from [IS24] in Figure 3 for completeness.

Lemma 6 ([IS24])**.** *The circuit* $C^\star$ *constructed in Figure 3 is a* (t, ϵ)*-parity-tolerant implementation of* f*.*

Derandomization of the Compiler in [IS24]. As discussed in Section 7.1, we will use randomness-efficient instantiations for the tools used in the compiler in [IS24].

For the compiler that transforms any function g to a k-probing-tolerant implementation of g, we may rely on known results in [IKL+13, CG20], which both achieve poly-logarithmic randomness complexity in the circuit size.

We will show how to construct randomness-efficient compiler that transforms any circuit $\tilde{C}$ to a (t, k, ϵ)-parity-to-probing implementation of $\tilde{C}$ in the next subsection.

As for $(\mathtt{Enc}, \mathtt{Dec})$, we will use a strong and linear k-wise independent PRG $G : \{0,1\}^\lambda \to \{0,1\}^n$. Recall that it satisfies that

- Strong k-wise Independence: When the input $\boldsymbol{r} \in \{0,1\}^\lambda$ is sampled uniformly, any k bits of $(\boldsymbol{r}, G(\boldsymbol{r}))$ are independent and uniformly distributed.
- Linearity: Each output bit of G is a linear combination of its input.

Parity-Tolerant Circuit for f

1. Encoding the Input: The circuit $C^\star$ samples random bits $\boldsymbol{r}$ for $\mathtt{Enc}$, pads the input $\boldsymbol{x}$ to be $\boldsymbol{x}\|\mathbf{0} \in \{0,1\}^n$, and computes $\hat{\boldsymbol{x}} := \mathtt{Enc}(\boldsymbol{x}\|\mathbf{0}; \boldsymbol{r})$ by only using $\mathtt{XOR}$ gates.
2. Computing Encoded Output from Encoded Input:
   (a) Let λ be the number of random bits used in $\mathtt{Enc}$. Let $f' : \{0,1\}^{\hat{n}} \times \{0,1\}^\lambda \to \{0,1\}^{\hat{n}}$ be the function defined by

$$\begin{aligned} f'(\hat{\boldsymbol{x}}, \boldsymbol{r}') \quad : \quad \boldsymbol{x}\|* &\leftarrow \mathtt{Dec}(\hat{\boldsymbol{x}}), \text{ where } \boldsymbol{x} \in \{0,1\}^{n_i} \\ \boldsymbol{y} &\leftarrow f(\boldsymbol{x}) \\ \hat{\boldsymbol{y}} &\leftarrow \mathtt{Enc}(\boldsymbol{y}\|\mathbf{0}; \boldsymbol{r}'), \text{ where } \boldsymbol{y}\|\mathbf{0} \in \{0,1\}^n. \end{aligned}$$

   (b) Let $\tilde{C}$ be a k-probing-tolerant implementation of f'. Note that $\tilde{C}$ is a randomized circuit. Let (I, C, O) be a (t, k, ϵ)-parity-to-probing circuit implementation of $\tilde{C}$.
   (c) The circuit $C^\star$ samples random bits $\boldsymbol{r}'$ and computes $\hat{\boldsymbol{y}} = O(C(I(\hat{\boldsymbol{x}}, \boldsymbol{r}')))$.
3. Decoding the Output: The circuit $C^\star$ computes $\mathtt{Dec}(\hat{\boldsymbol{y}})$ and outputs the first n_o bits.

Figure 3: Construction of Parity-Tolerant Circuits from Parity-to-Probing Circuits

An explicit construction of such a PRG G can be found in [GIS22] which requires $\lambda = O(k \cdot \log n)$ random bits.

Consider the following construction: $\mathtt{Enc}(\boldsymbol{x}; \boldsymbol{r}) = (\boldsymbol{r}, G(\boldsymbol{r}) \oplus \boldsymbol{x})$, and $\mathtt{Dec}(\hat{\boldsymbol{x}}) = \mathtt{Dec}(\boldsymbol{r}, G(\boldsymbol{r}) \oplus \boldsymbol{x}) = G(\boldsymbol{r}) \oplus (G(\boldsymbol{r}) \oplus \boldsymbol{x}) = \boldsymbol{x}$. For $\mathtt{Enc}$:

- $\mathtt{Enc}$ output its random tape $\boldsymbol{r}$.
- By the linearity of G, any output bit of $\mathtt{Enc}$ is a linear combination of $\boldsymbol{x}, \boldsymbol{r}$.
- When $\boldsymbol{r}$ is sampled uniformly, by the strong k-wise independence of G, any k bits of the output of $\mathtt{Enc}$ are independent and uniformly distributed.

As for $\mathtt{Dec}$, we always have $\mathtt{Dec}(\mathtt{Enc}(\boldsymbol{x})) = \boldsymbol{x}$. By the linearity of G, each output bit of $\mathtt{Dec}$ is a linear combination of its input. Thus $(\mathtt{Enc}, \mathtt{Dec})$ satisfies the above requirements.

7.3 Randomness-Efficient Compiler for Parity-to-Probing Circuits

We start with the derandomization of the construction in [GIM+16] that requires secure $\mathtt{NAND}$ gadgets. Let $\tilde{C}$ denote the input circuit and $\boldsymbol{w} = (w_1, \ldots, w_{|\tilde{C}|})$ denote the wire values in $\tilde{C}$. Recall that the idea is to compute the small-bias encoding (Figure 2) for each wire value in $\tilde{C}$. To be more concrete, the compiler in [GIM+16] takes $\tilde{C}$ as input and outputs (I, C, O) where the wire values in C are

$$(u_1, v_1, u_1 \cdot v_1 \oplus w_1), \ldots, (u_{|\tilde{C}|}, v_{|\tilde{C}|}, u_{|\tilde{C}|} \cdot v_{|\tilde{C}|} \oplus w_{|\tilde{C}|}),$$

or equivalently, $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w})$, where $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^{|\tilde{C}|}$ are random vectors.

Our idea is to generate $\boldsymbol{u}$ by an unprotected linear k-wise independent PRG and generate $\boldsymbol{v}$ by an unprotected parity-resilient PRG: Let $G_1 : \{0,1\}^{\lambda_1} \to \{0,1\}^{|\tilde{C}|}$ be a linear k-wise independent PRG, and $G_2 : \{0,1\}^{\lambda_2} \to \{0,1\}^{|\tilde{C}|}$ be a parity-resilient PRG. Then $\boldsymbol{u}, \boldsymbol{v}$ are obtained as followed

1. Randomly sample $\boldsymbol{r}_1 \in \{0,1\}^{\lambda_1}, \boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$.
2. Set $\boldsymbol{u} = G_1(\boldsymbol{r}_1)$ and $\boldsymbol{v} = G_2(\boldsymbol{r}_2)$.

Here we choose any linear circuit implementation for G_1 (i.e., only using `XOR` gates) and any circuit implementation for G_2.

Consider a subset W of wires in $\hat{C}$. Let $\mathcal{I} = \{i \mid u_i \cdot v_i \oplus w_i \in W\}$. Then $\texttt{parity}(W) = \oplus_{i\in\mathcal{I}}(u_i \cdot v_i \oplus w_i) \oplus f(\boldsymbol{r}_1) \oplus g(\boldsymbol{r}_2)$, where f is some *linear* function on its input $\boldsymbol{r}_1$ and g is some arbitrary function on its input $\boldsymbol{r}_2$. For an integer $k > 0$, if $|\mathcal{I}| \le k$, then $\texttt{parity}(W)$ can be simulated perfectly with the wire values in $\{w_i\}_{i\in\mathcal{I}}$. Now consider the case that $|\mathcal{I}| \ge k$. We have the following lemma.

Lemma 7. *Let $k > 0, n > 0$ be arbitrary integers. Let $G_1 : \{0,1\}^{\lambda_1} \to \{0,1\}^n$ be a linear k-wise independent PRG and $G_2 : \{0,1\}^{\lambda_2} \to \{0,1\}^n$ be a parity-resilient PRG with error $\epsilon^\star$. Let $\boldsymbol{r}_1 \in \{0,1\}^{\lambda_1}, \boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$ be uniform strings, and $(\boldsymbol{u}, \boldsymbol{v}) = (G_1(\boldsymbol{r}_1), G_2(\boldsymbol{r}_2))$. For all $\mathcal{I} \subset \{1, \ldots, n\}$ of size at least k and for all $\mathcal{J} \subset \{1, \ldots, \lambda_1\}$, the joint distribution of $((\oplus_{i\in\mathcal{I}} u_i \cdot v_i) \oplus (\oplus_{j\in\mathcal{J}} r_{1,j}), \boldsymbol{r}_2)$ is statistically close to $(r', \boldsymbol{r}_2)$, where r' is a uniform bit. The statistical error is equal to $2^{-k-1} + \epsilon^\star$.*

Proof. Recall that G_1 is a linear k-wise independent PRG. Then each bit of $\boldsymbol{u}$ is a linear combination of $\boldsymbol{r}_1$. Let $\boldsymbol{A}$ be the matrix of size $|\mathcal{I}| \times \lambda_1$ such that $(u_i)_{i\in\mathcal{I}} = \boldsymbol{A} \cdot \boldsymbol{r}_1$. Then

$$\oplus_{i\in\mathcal{I}} u_i \cdot v_i = \langle (u_i)_{i\in\mathcal{I}}, (v_i)_{i\in\mathcal{I}} \rangle = (\boldsymbol{r}_1)^{\mathrm{T}} \cdot \boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}}.$$

Let $\mathbf{1}_{\mathcal{J}}$ denote the vector of dimension λ_1 where the j-th entry is 1 if $j \in \mathcal{J}$, and 0 otherwise. Then

$$\oplus_{j\in\mathcal{J}} r_{1,j} = (\boldsymbol{r}_1)^{\mathrm{T}} \cdot \mathbf{1}_{\mathcal{J}}.$$

Now we have that,

$$(\oplus_{i\in\mathcal{I}} u_i \cdot v_i) \oplus (\oplus_{j\in\mathcal{J}} r_{1,j}) = (\boldsymbol{r}_1)^{\mathrm{T}} \cdot (\boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}} \oplus \mathbf{1}_{\mathcal{J}}).$$

Note that if $\boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}} \oplus \mathbf{1}_{\mathcal{J}}$ is not an all-zero vector, then the above result is a uniform bit even given $\boldsymbol{r}_2$, since it is equal to a linear combination of bits in $\boldsymbol{r}_1$, which are a batch of random bits. Therefore, it is sufficient to show that the probability that $\boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}} \oplus \mathbf{1}_{\mathcal{J}} = \mathbf{0}$ is negligible. Or equivalently, with overwhelming probability,

$$\boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}} \neq \mathbf{1}_{\mathcal{J}}.$$

Since G_1 is a linear k-wise independent PRG, any k rows of $\boldsymbol{A}$ are linearly independent. It also implies that $\boldsymbol{A}$ has rank at least k. Without loss of generality, assume that the first k columns of $\boldsymbol{A}$ are linearly independent. For each $j \in \{1, 2, \ldots, k\}$, $\boldsymbol{A}_j^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}}$ corresponds to a linear query to the output of G_2. In particular, these k queries are linearly independent. Let $Y_j = \boldsymbol{A}_j^{\mathrm{T}} \cdot (v_i)_{i\in\mathcal{I}}$. Then for all $S \subset \{1, \ldots, k\}$ and $S \neq \emptyset$, $\oplus_{j\in S} Y_j$ is statistically close to a uniform bit with error $\epsilon^\star$.

Claim 1. *Let $k > 0$ be an integer. Suppose $Y_1, \ldots, Y_k$ are k random variables over $\{0,1\}$ such that for all $S \subset \{1, \ldots, k\}$ and $S \neq \emptyset$, $\oplus_{j \in S} Y_j$ is statistically close to a uniform bit with error $\epsilon^\star$. Then for all k-bit string $\boldsymbol{y}$, $\Pr[\boldsymbol{Y} = \boldsymbol{y}] \leq 1/2^k + 2\epsilon^\star$.*

Proof. Let $\boldsymbol{Y'} = (Y'_1, \ldots, Y'_k)$ be k random variables over $\{0,1\}$ that are independent and uniformly random. Then for all $\boldsymbol{y} \in \{0,1\}^k$, $\Pr[\boldsymbol{Y'} = \boldsymbol{y}] = 1/2^k$. It is sufficient to show that $|\Pr[\boldsymbol{Y} = \boldsymbol{y}] - \Pr[\boldsymbol{Y'} = \boldsymbol{y}]| \leq 2\epsilon^\star$.

Let $f(\boldsymbol{y}) = \Pr[\boldsymbol{Y} = \boldsymbol{y}] - \Pr[\boldsymbol{Y'} = \boldsymbol{y}]$. Then the Fourier coefficients of $f(\cdot)$ are $\hat{f}(S) = \frac{1}{2^k} \sum_{\boldsymbol{y} \in \{0,1\}^k} f(\boldsymbol{y}) \chi_S(\boldsymbol{y})$. For all $S \neq \emptyset$, we have

$$\begin{aligned}
\frac{1}{2^k} \sum_{\boldsymbol{y} \in \{0,1\}^k} f(\boldsymbol{y}) \chi_S(\boldsymbol{y}) &= \frac{1}{2^k} \sum_{\boldsymbol{y} \in \{0,1\}^k} (\Pr[\boldsymbol{Y} = \boldsymbol{y}] - \Pr[\boldsymbol{Y'} = \boldsymbol{y}]) \chi_S(\boldsymbol{y}) \\
&= \frac{1}{2^k} \sum_{\boldsymbol{y} \in \{0,1\}^k} \Pr[\boldsymbol{Y} = \boldsymbol{y}] \chi_S(\boldsymbol{y}) \\
&= \frac{1}{2^k} \cdot (\Pr[\chi_S(\boldsymbol{Y}) = 1] - \Pr[\chi_S(\boldsymbol{Y}) = -1])
\end{aligned}$$

Here the second step relies on the fact that (1) $\Pr[\boldsymbol{Y'} = \boldsymbol{y}] = 1/2^k$ and (2) $\sum_{\boldsymbol{y} \in \{0,1\}^k} \chi_S(\boldsymbol{y}) = 0$ for all $S \neq \emptyset$. Note that for all $S \neq \emptyset$, $\oplus_{j \in S} Y_j$ is statistically close to a uniform bit with error at most $\epsilon^\star$. Thus, $|(\Pr[\chi_S(\boldsymbol{Y}) = 1] - \Pr[\chi_S(\boldsymbol{Y}) = -1])| \leq 2\epsilon^\star$. Thus, $|\hat{f}(S)| \leq 2\epsilon^\star / 2^k$. When $S = \emptyset$, we have $\hat{f}(S) = 0$.

Thus, for all $\boldsymbol{y} \in \{0,1\}^k$,

$$\begin{aligned}
|f(\boldsymbol{y})| &= |\sum_{S \subset \{1,\ldots,k\}} \hat{f}(S) \cdot \chi_S(\boldsymbol{y})| \\
&\leq \sum_{S \subset \{1,\ldots,k\}} |\hat{f}(S)| \\
&\leq 2\epsilon^\star.
\end{aligned}$$

□

Thus, with probability $1 - 1/2^k - 2\epsilon^\star$, $\boldsymbol{A}^{\mathrm{T}} \cdot (v_i)_{i \in \mathcal{I}} \neq \mathbf{1}_{\mathcal{J}}$, indicating that $(\oplus_{i \in \mathcal{I}} u_i \cdot v_i) \oplus (\oplus_{j \in \mathcal{J}} r_{1,j})$ is a uniform bit given $\boldsymbol{r}_2$. This implies that $((\oplus_{i \in \mathcal{I}} u_i \cdot v_i) \oplus (\oplus_{j \in \mathcal{J}} r_{1,j}), \boldsymbol{r}_2)$ is statistically close to $(r', \boldsymbol{r}_2)$ with error $2^{-k-1} + \epsilon^\star$. □

We have the following corollary directly from Lemma 7.

Corollary 2. *Let $k > 0, n > 0$ be arbitrary integers. Let $G_1 : \{0,1\}^{\lambda_1} \to \{0,1\}^n$ be a linear k-wise independent PRG and $G_2 : \{0,1\}^{\lambda_2} \to \{0,1\}^n$ be a parity-resilient PRG with error $\epsilon^\star$. Let $\boldsymbol{r}_1 \in \{0,1\}^{\lambda_1}, \boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$ be uniform strings, and $(\boldsymbol{u}, \boldsymbol{v}) = (G_1(\boldsymbol{r}_1), G_2(\boldsymbol{r}_2))$. For all $\mathcal{I} \subset \{1, \ldots, n\}$ of size at least k, for all $\mathcal{J} \subset \{1, \ldots, \lambda_1\}$, and for all function $f : \{0,1\}^{\lambda_1} \to \{0,1\}$, the distribution of $(\oplus_{i \in \mathcal{I}} u_i \cdot v_i) \oplus (\oplus_{j \in \mathcal{J}} r_{1,j}) \oplus f(\boldsymbol{r}_2)$ is statistically close to r', where r' is a uniform bit. The statistical error is bounded by $2^{-k-1} + \epsilon^\star$.*

Thus, from Corollary 2, when $|\mathcal{I}| \geq k$, $\texttt{parity}(W)$ is statistically close to a uniform bit. In the next subsection, we move to derandomize our construction in [IS24].

7.3.1 Review of NAND Gadgets in [IS24]

For each NAND gate in $\tilde{C}$ with input wires w_1, w_2 and output wire $w_0 = w_1$ NAND w_2, the NAND gadget takes the small-bias enodings of w_1, w_2 as input and computes the small-bias encoding of w_0. Let $(u_1, v_1, z_1), (u_2, v_2, z_2)$ denote the encodings of w_1, w_2 respectively and let u_0, v_0 denote the first two random bits of the small-bias encoding of w_0. Then the goal is to compute $z_0 = u_0 \cdot v_0 \oplus w_0$.

In [IS24], the circuit first computes new small-bias encodings for w_1 and w_2 with fresh randomness, denoted by (u_1', v_1', z_1') and (u_2', v_2', z_2'), and then computes z_0 from these two new small-bias encodings. We give the realization of the NAND gadget in Figure 4.

An important observation used in [IS24] is that, for all function $f : \{0,1\}^n \to \{0,1\}$ on input $\boldsymbol{x} \in \{0,1\}^n$ and for all possible assignment $\boldsymbol{a} \in \{0,1\}^n$, $f(\boldsymbol{x}) \cdot \prod_{i=1}^n (x_i \oplus a_i \oplus 1) = f(\boldsymbol{a}) \cdot \prod_{i=1}^n (x_i \oplus a_i \oplus 1)$. This is because when $\boldsymbol{x} = \boldsymbol{a}$, both sides are equal to $f(\boldsymbol{a})$. Otherwise, both sides are equal to 0. Thus, in Step 1(c), $\gamma_{1,\boldsymbol{a}} = (z_1' \oplus \rho_1) \cdot \beta_{1,\boldsymbol{a}}$ and $\gamma_{2,\boldsymbol{a}} = (z_2' \oplus \rho_2) \cdot \beta_{2,\boldsymbol{a}}$. Similarly, in Step 2(d), $\gamma_{3,\boldsymbol{a}} = (z_0 \oplus \rho_3) \cdot \beta_{3,\boldsymbol{a}}$.

The security of Figure 4 follows from the following facts:

- First, computing new small-bias encodings of w_1, w_2 does not break the parity-to-probing property.
- Let $\rho_1' = z_1' \oplus \rho_1$ and $\rho_2' = z_2' \oplus \rho_2$. Then each intermediate wire in Step 1 can be written in the form of a linear combination of $u_1, z_1, u_1', z_1', u_2, z_2, u_2', z_2'$ plus some function on $v_1, v_1', v_2, v_2', \rho_1', \rho_2'$. By the observation that the parity-to-probing property holds even if v_1, v_1', v_2, v_2' are given, and that ρ_1', ρ_2' are just random bits, being able to attack the intermediate wires in Step 1 does not give any further advantage to the adversary.
- Similarly, let $\rho_3' = z_0 \oplus \rho_3$. Then each intermediate wire in Step 2 can be written in the form of a linear combination of u_1', z_1', u_2', z_2' and $z_1' \cdot z_2', z_1' \cdot u_2', u_1' \cdot z_2', u_1' \cdot u_2'$ plus some function on v_0, v_1', v_2', ρ_3'. The authors in [IS24] show that in this case, being able to attack the intermediate wires in Step 2 only give limited advantage to the adversary.

  The idea is to show that if the adversary touches enough number of wires related to z_1' and z_2', then the parity result should be statistically close to a random bit. This can be argued by first fixing the encoding (u_1', v_1', z_1') in all NAND gadgets. Then all intermediate wires become linear in u_2', z_2' and we can show that if the adversary touches enough number of wires related to z_2' then the parity result is statistically close to a random bit. Similar analysis holds for wires related to z_1'.

  To make the above argument work, it is important that (u_2', v_2', z_2') is not fixed when fixing (u_1', v_1', z_1'), which is the reason of computing new encodings for w_1 and w_2 in Step 1.

7.3.2 Derandomization of NAND Gadgets

Following Lemma 7, we will generate u, u_1', u_2' of each NAND gadget by a linear k-wise independent PRG with independent random seeds respectively, and generate v, v_1', v_2' by a parity-resilient PRG with independent random seeds respectively.

Now the remaining problem is how to derandomize ρ_1, ρ_2, ρ_3. Recall that in the above analysis, we utilize that ρ_1, ρ_2, ρ_3 are uniformly random so that $\rho_1' = z_1' \oplus \rho_1, \rho_2' = z_2' \oplus \rho_2, \rho_3' = z_0 \oplus \rho_3$ are also uniformly random. However, this is no longer true when we derandomize ρ_1, ρ_2, ρ_3, which

Circuit for NAND

- Input: $(u_1, v_1, z_1), (u_2, v_2, z_2), u_0, v_0$
- Auxiliary Input: $(u'_1, v'_1), (u'_2, v'_2), \rho_1, \rho_2, \rho_3$
- Output: $z_0 := u_0 \cdot v_0 \oplus (z_1 \oplus u_1 \cdot v_1) \cdot (z_2 \oplus u_2 \cdot v_2) \oplus 1$.

1. Refresh Input Encoding: The goal of the first step is to compute

$$z'_1 = u'_1 \cdot v'_1 \oplus (z_1 \oplus u_1 \cdot v_1), \quad z'_2 = u'_2 \cdot v'_2 \oplus (z_2 \oplus u_2 \cdot v_2).$$

   The circuit is constructed as follows.

   (a) For all $\boldsymbol{a} \in \{0,1\}^2$, the circuit computes $\alpha_{1,\boldsymbol{a}} := u'_1 \cdot a_1 \oplus (z_1 \oplus u_1 \cdot a_2) \oplus \rho_1$ and $\alpha_{2,\boldsymbol{a}} := u'_2 \cdot a_1 \oplus (z_2 \oplus u_2 \cdot a_2) \oplus \rho_2$.

   (b) For all $\boldsymbol{a} \in \{0,1\}^2$, the circuit computes $\beta_{1,\boldsymbol{a}} := (v'_1 \oplus a_1 \oplus 1) \cdot (v_1 \oplus a_2 \oplus 1)$ and $\beta_{2,\boldsymbol{a}} := (v'_2 \oplus a_1 \oplus 1) \cdot (v_2 \oplus a_2 \oplus 1)$.

   (c) For all $\boldsymbol{a} \in \{0,1\}^2$, the circuit computes $\gamma_{1,\boldsymbol{a}} := \alpha_{1,\boldsymbol{a}} \cdot \beta_{1,\boldsymbol{a}}$ and $\gamma_{2,\boldsymbol{a}} := \alpha_{2,\boldsymbol{a}} \cdot \beta_{2,\boldsymbol{a}}$.

   (d) The circuit computes

$$z'_1 = \left(\oplus_{\boldsymbol{a} \in \{0,1\}^2} \gamma_{1,\boldsymbol{a}}\right) \oplus \rho_1, \quad z'_2 = \left(\oplus_{\boldsymbol{a} \in \{0,1\}^2} \gamma_{2,\boldsymbol{a}}\right) \oplus \rho_2.$$

2. Computing NAND on Refreshed Input Encodings: The goal of the second step is to compute z_0 from $(u'_1, v'_1, z'_1), (u'_2, v'_2, z'_2), u_0, v_0$:

$$z_0 = u_0 \cdot v_0 \oplus (z'_1 \oplus u'_1 \cdot v'_1) \cdot (z'_2 \oplus u'_2 \cdot v'_2) \oplus 1.$$

   The circuit is constructed as follows.

   (a) The circuit computes $z'_1 \cdot z'_2, z'_1 \cdot u'_2, u'_1 \cdot z'_2, u'_1 \cdot u'_2$.

   (b) For all $\boldsymbol{a} \in \{0,1\}^3$, the circuit computes

$$\begin{aligned} \alpha_{3,\boldsymbol{a}} &:= u_0 \cdot a_0 \oplus (z'_1 \oplus u'_1 \cdot a_1) \cdot (z'_2 \oplus u'_2 \cdot a_2) \oplus 1 \oplus \rho_3 \\ &= u_0 \cdot a_0 \oplus (z'_1 \cdot z'_2) \oplus a_2 \cdot (z'_1 \cdot u'_2) \oplus a_1 \cdot (u'_1 \cdot z'_2) \oplus \ a_1 a_2 \cdot (u'_1 \cdot u'_2) \oplus 1 \oplus \rho_3. \end{aligned}$$

   by a proper linear combination of $u_0, z'_1 \cdot z'_2, z'_1 \cdot u'_2, u'_1 \cdot z'_2, u'_1 \cdot u'_2, \rho_3$.

   (c) For all $\boldsymbol{a} \in \{0,1\}^3$, the circuit computes $\beta_{3,\boldsymbol{a}} := (v_0 \oplus a_0 \oplus 1) \cdot (v'_1 \oplus a_1 \oplus 1) \cdot (v'_2 \oplus a_2 \oplus 1)$.

   (d) For all $\boldsymbol{a} \in \{0,1\}^3$, the circuit computes $\gamma_{3,\boldsymbol{a}} := \alpha_{3,\boldsymbol{a}} \cdot \beta_{3,\boldsymbol{a}}$.

   (e) The circuit computes $z_0 = \left(\oplus_{\boldsymbol{a} \in \{0,1\}^3} \gamma_{3,\boldsymbol{a}}\right) \oplus \rho_3$.

Figure 4: Circuit Implementation of NAND Gadget in [IS24]

forces us to consider the following intermediate wires

$$\{\gamma_{1,\boldsymbol{a}} = (z'_1 \oplus \rho_1) \cdot (v'_1 \oplus a_1)(v_1 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^2},$$
$$\{\gamma_{2,\boldsymbol{a}} = (z'_2 \oplus \rho_2) \cdot (v'_2 \oplus a_1)(v_2 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^2},$$
$$\{\gamma_{3,\boldsymbol{a}} = (z_0 \oplus \rho_3) \cdot (v_0 \oplus a_0)(v'_1 \oplus a_1)(v'_2 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^3}.$$

Our idea is to derandomize ρ_1, ρ_2, ρ_3 of each `NAND` gadget by a linear k-wise independent PRG with independent random seeds. The intuition is as follows:

- For $\{\gamma_{1,\boldsymbol{a}} = (z'_1 \oplus \rho_1) \cdot (v'_1 \oplus a_1)(v_1 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^2}$, if we fix v_1, then this set becomes $\{(z'_1 \oplus \rho_1) \cdot v'_1, (z'_1 \oplus \rho_1) \cdot (v'_1 \oplus 1)\}$. Here ρ_1 is from the output of a linear k-wise independent PRG, and v'_1 is from the output of a parity-resilient PRG. We may use Lemma 7 to argue the parity-to-probing security.

  Similar argument works for $\{\gamma_{2,\boldsymbol{a}} = (z'_2 \oplus \rho_2) \cdot (v'_2 \oplus a_1)(v_2 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^2}$.

- For $\{\gamma_{3,\boldsymbol{a}} = (z_0 \oplus \rho_3) \cdot (v_0 \oplus a_0)(v'_1 \oplus a_1)(v'_2 \oplus a_2)\}_{\boldsymbol{a}\in\{0,1\}^3}$, if we fix v'_1, v'_2, then this set becomes $\{(z_0 \oplus \rho_3) \cdot v_0, (z_0 \oplus \rho_3) \cdot (v_0 \oplus 1)\}$. Here ρ_3 is from the output of a linear k-wise independent PRG, and v_0 is from the output of a parity-resilient PRG. We may use Lemma 7 to argue the parity-to-probing security.

We describe our construction in Figure 5 and Figure 6.

Derandomized (t, k, ϵ)-Parity-to-Probing Circuit (I, C, O)

- Input: A randomized circuit $\tilde{C}$ with input size $\{0,1\}^{n_i}$ and output size $\{0,1\}^{n_o}$
- Output: A tuple (I, C, O)
- Auxiliary Input: A parameter $k > 0$, a linear circuit implementation of a k-wise independent PRG $G_1 : \{0,1\}^{\lambda_1} \to \{0,1\}^{|\tilde{C}|}$, and a circuit implementation of a parity-resilient PRG $G_2 : \{0,1\}^{\lambda_2} \to \{0,1\}^{|\tilde{C}|}$ with error $\epsilon^\star$.

Input Encoder I: For input $\boldsymbol{x} \in \{0,1\}^{n_i}$, $I(\boldsymbol{x})$ does the following.

1. I randomly samples $\boldsymbol{r}_1 \in \{0,1\}^{\lambda_1}, \boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$. I computes $\boldsymbol{u} = G_1(\boldsymbol{r}_1)$ and $\boldsymbol{v} = G_2(\boldsymbol{r}_2)$.
2. I outputs $(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{u}, \boldsymbol{v})$, all inner wires of $G_2(\boldsymbol{r}_2)$, and $\{u_i \cdot v_i \oplus x_i\}_{i=1}^{n_i}$.

Figure 5: Derandomized (t, k, ϵ)-Parity-to-Probing Circuit (Part 1)

Theorem 6. *For all circuit $\tilde{C}$, the construction in Figure 5 is a (t, k, ϵ)-parity-to-probing circuit implementation of $\tilde{C}$, where*

$$\epsilon = 2^{t/2} \cdot (e^{-k'/168} + 2^{-k'/42} + 2(2^{2k'/21} + 1)\epsilon^\star),$$

$k' = k/t$, and $\epsilon^\star$ is the statistical error of G_2 in Figure 5.

Derandomized (t, k, ϵ)-Parity-to-Probing Circuit (I, C, O)

Circuit C:

1. Suppose the wires in $\tilde{C}$ are denoted by $w_1, \ldots, w_{|\tilde{C}|}$. For simplicity, assume that the first n_i wires are the input wires, and the last n_o wires are the output wires. The circuit C takes $I(\boldsymbol{x})$ as input. Then C generates the following random bits:
   (a) C randomly samples $\boldsymbol{r}_3, \boldsymbol{r}_1^{(1)}, \boldsymbol{r}_3^{(1)}, \boldsymbol{r}_1^{(2)}, \boldsymbol{r}_3^{(2)} \in \{0,1\}^{\lambda_1}$ and $\boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)} \in \{0,1\}^{\lambda_2}$.
   (b) C computes $\boldsymbol{\rho} = G_1(\boldsymbol{r}_3)$, $(\boldsymbol{u}^{(1)}, \boldsymbol{v}^{(1)}, \boldsymbol{\rho}^{(1)}) = (G_1(\boldsymbol{r}_1^{(1)}), G_2(\boldsymbol{r}_2^{(1)}), G_1(\boldsymbol{r}_3^{(1)}))$, and $(\boldsymbol{u}^{(2)}, \boldsymbol{v}^{(2)}, \boldsymbol{\rho}^{(2)}) = (G_1(\boldsymbol{r}_1^{(2)}), G_2(\boldsymbol{r}_2^{(2)}), G_1(\boldsymbol{r}_3^{(2)}))$.
2. For each random wire w_i in $\tilde{C}$, the circuit C samples a random bit as z_i and sets the encoding of w_i as (u_i, v_i, z_i).
3. We label the `NAND` gates in $\tilde{C}$ by the indices of their output wires. I.e., the output of the ℓ-th `NAND` gate is the ℓ-th wire w_ℓ. Let $\pi_1(\cdot)$ be the function that maps ℓ to the index of the first input of the ℓ-th `NAND` gate, and $\pi_2(\cdot)$ be the function that maps ℓ to the index of the second input of the ℓ-th `NAND` gate. Then the ℓ-th `NAND` gate takes as input $w_{\pi_1(\ell)}, w_{\pi_2(\ell)}$ and outputs $w_\ell = w_{\pi_1(\ell)}$ `NAND` $w_{\pi_2(\ell)}$.
4. For each `NAND` gate in $\tilde{C}$, suppose this is the ℓ-th `NAND` gate and the input encodings are $(u_{\pi_1(\ell)}, v_{\pi_1(\ell)}, z_{\pi_1(\ell)})$ and $(u_{\pi_2(\ell)}, v_{\pi_2(\ell)}, z_{\pi_2(\ell)})$. The circuit C computes the subcircuit for the `NAND` gadget described in Figure 4 with
   - Input: $(u_{\pi_1(\ell)}, v_{\pi_1(\ell)}, z_{\pi_1(\ell)}), (u_{\pi_2(\ell)}, v_{\pi_2(\ell)}, z_{\pi_2(\ell)}), u_\ell, v_\ell$
   - Auxiliary Input: $(u_\ell^{(1)}, v_\ell^{(1)}), (u_\ell^{(2)}, v_\ell^{(2)}), \rho_\ell^{(1)}, \rho_\ell^{(2)}, \rho_\ell$

   We set z_ℓ to be the output of the subcircuit.
5. The output of C is $\{(u_i, v_i, z_i)\}_{i=|\tilde{C}|-n_o+1}^{|\tilde{C}|}$

Output Decoder O: O takes the output of C, denoted by $\{(u_i, v_i, z_i)\}_{i=|\tilde{C}|-n_o+1}^{|\tilde{C}|}$, as input, and outputs $\boldsymbol{y} = (u_{|\tilde{C}|-n_o+1} \cdot v_{|\tilde{C}|-n_o+1} \oplus z_{|\tilde{C}|-n_o+1}, \ldots, u_{|\tilde{C}|} \cdot v_{|\tilde{C}|} \oplus z_{|\tilde{C}|})$.

Figure 6: Derandomized (t, k, ϵ)-Parity-to-Probing Circuit (Part 2)

7.4 Proof of Theorem 6

In this section, we formally prove Theorem 6. Let $\tilde{C}$ denote the input circuit, and (I, C, O) denote the construction from Figure 5. We use $\boldsymbol{w} = (w_1, \ldots, w_{|\tilde{C}|})$ to denote the wires in $\tilde{C}$. We first show that the input encoder I and the output decoder O are trivially parity-tolerant.

7.4.1 Trivial Parity Tolerance.

For the input encoder, consider the following circuit implementation of I:

1. The circuit takes $\boldsymbol{x}$ as input.
2. The circuit randomly samples $\boldsymbol{r}_1 \in \{0,1\}^{\lambda_1}, \boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$ and computes $\boldsymbol{u} = G_1(\boldsymbol{r}_1)$ by a linear circuit implementation of G_1 and $\boldsymbol{v} = G_2(\boldsymbol{r}_2)$ by a circuit implementation of G_2.
3. For all $i \in \{1, \ldots, n_i\}$, the circuit computes $u_i \cdot v_i$ and then computes $z_i := (u_i \cdot v_i) \oplus x_i$.
4. The circuit outputs $(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{u}, \boldsymbol{v})$, all inner wires of $G_2(\boldsymbol{r}_2)$, $\{u_i, v_i, z_i\}_{i=1}^{n_i}$.

Note that the sampled randomness $\boldsymbol{r}_1, \boldsymbol{r}_2$ are a part of the output. Since the circuit implementation of G_1 is linear, each inner wire can be computed as a linear combination of $\boldsymbol{r}_1$. As for G_2, the inner wires are a part of the output. For all $i \in \{1, \ldots, n_i\}$, the intermediate result $u_i \cdot v_i$ can be written as $(u_i \cdot v_i \oplus x_i) \oplus x_i = z_i \oplus x_i$, where z_i is an output bit and x_i is an input bit. Thus, I is trivially parity-tolerant.

As for the output decoder, O takes as input $\{u_i, v_i, z_i\}_{i=1}^{n_o}$ and outputs $\boldsymbol{y} = (u_i \cdot v_i \oplus z_i)_{i=1}^{n_o}$. Consider the following circuit implementation of O:

1. The circuit takes $\{u_i, v_i, z_i\}_{i=1}^{n_o}$ as input.
2. For all $i \in \{1, \ldots, n_i\}$, the circuit computes $u_i \cdot v_i$ and then computes $y_i := u_i \cdot v_i \oplus z_i$.
3. The circuit outputs $\boldsymbol{y}$.

Note that for all $i \in \{1, \ldots, n_i\}$, the intermediate result $u_i \cdot v_i$ can be written as $(u_i \cdot v_i \oplus z_i) \oplus z_i = y_i \oplus z_i$, where y_i is an output bit and z_i is an input bit. Thus, O is also trivially parity-tolerant.

Now we move to show that C achieves parity-to-probing security. We first analyse the wires in C and then prove the parity-to-probing security by hybrid arguments.

7.4.2 Analysis of Wires in C.

Let $\boldsymbol{z} := \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}$, $\boldsymbol{z}^{(1)} = (z_{\pi_1(\ell)})_{\ell=1}^{|\tilde{C}|}$, and $\boldsymbol{z}^{(2)} = (z_{\pi_2(\ell)})_{\ell=1}^{|\tilde{C}|}$. The wires we need to consider in C can be summarized as follows.

- Layer 0 — Randomness: $(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3), (\boldsymbol{r}_1^{(1)}, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_3^{(1)}), (\boldsymbol{r}_1^{(2)}, \boldsymbol{r}_2^{(2)}, \boldsymbol{r}_3^{(2)})$
- Layer 1 — Wires Only Depending on Randomness: Inner wires of $G_2(\boldsymbol{r}_2)$, $G_2(\boldsymbol{r}_2^{(1)})$, $G_2(\boldsymbol{r}_2^{(2)})$. Wires that are computation on $\boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}$ in the subcircuits for `NAND` gates.
- Layer 2 — Rest of Wires Related to $\boldsymbol{\rho}$: $\{(z_i \oplus \rho_i) \cdot (v_i \oplus a_0)(v_i^{(1)} \oplus a_1)(v_i^{(2)} \oplus a_2) \mid \boldsymbol{a} \in \{0,1\}^3\}_{i=1}^{|\tilde{C}|}$.
- Layer 3 — Rest of Wires Related to $\boldsymbol{u}$: $\boldsymbol{z}$.
- Layer 4 — Rest of Wires Related to $\boldsymbol{\rho}^{(1)}$: $\{(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot (v_{\pi_1(i)} \oplus a_0)(v_i^{(1)} \oplus a_1) \mid \boldsymbol{a} \in \{0,1\}^2\}_{i=1}^{|\tilde{C}|}$.
- Layer 5 — Rest of Wires Related to $\boldsymbol{\rho}^{(2)}$: $\{(z_i^{(2)} \oplus \rho_i^{(2)}) \cdot (v_{\pi_2(i)} \oplus a_0)(v_i^{(2)} \oplus a_1) \mid \boldsymbol{a} \in \{0,1\}^2\}_{i=1}^{|\tilde{C}|}$.

- Layer 6 — Rest of Wires Related to $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}$: $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{z}^{(1)} * \boldsymbol{z}^{(2)}, \boldsymbol{z}^{(1)} * \boldsymbol{u}^{(2)}, \boldsymbol{u}^{(1)} * \boldsymbol{z}^{(2)}, \boldsymbol{u}^{(1)} * \boldsymbol{u}^{(2)}$.

For a subset W of wires in C and for all $i \in \{1, \ldots, |\tilde{C}|\}$, the parity of wires in $W \cap \{(z_i \oplus \rho_i) \cdot (v_i \oplus a_0)(v_i^{(1)} \oplus a_1)(v_i^{(2)} \oplus a_2) \mid \boldsymbol{a} \in \{0,1\}^3\}$ can be written as $(z_i \oplus \rho_i) \cdot \phi_i(v_i, v_i^{(1)}, v_i^{(2)})$ for some function $\phi_i : \{0,1\}^3 \to \{0,1\}$. Note that if $\phi_i \equiv 1$, then we may replace $z_i \oplus \rho_i$ by z_i and ρ_i in W. Without loss of generality, we assume that $\phi_i \not\equiv 1$.

Given $\boldsymbol{r}_2$, $\boldsymbol{v} = G_2(\boldsymbol{r}_2)$ is also fixed. For all $i \in \{1, \ldots, |\tilde{C}|\}$,

$$\{(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot (v_{\pi_1(i)} \oplus a_0)(v_i^{(1)} \oplus a_1) \mid \boldsymbol{a} \in \{0,1\}^2\}$$

is reduced to

$$\{(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)}, (z_i^{(1)} \oplus \rho_i^{(1)}) \cdot (v_i^{(1)} \oplus 1)\}.$$

Note that $(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot (v_i^{(1)} \oplus 1) = (z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)} \oplus z_i^{(1)} \oplus \rho_i^{(1)}$. We only need to consider $(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)}$.

Thus, given $\boldsymbol{r}_2$, the Layer-4 wires are reduced to $\{(z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)}\}_{i=1}^{|\tilde{C}|}$. Similarly, given $\boldsymbol{r}_2$, the Layer-5 wires are reduced to $\{(z_i^{(2)} \oplus \rho_i^{(2)}) \cdot v_i^{(2)}\}_{i=1}^{|\tilde{C}|}$.

We use $W|_{\boldsymbol{r}_2}$ to denote the set of wires in W given $\boldsymbol{r}_2$. For all subset W of wires in C and $\boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$, we define the following subsets of wires in $\tilde{C}$:

$$\begin{aligned}
\mathcal{I}_2 &= \{w_i \mid \phi_i \not\equiv 0\} \\
\mathcal{I}_3 &= \{w_i \mid z_i \in W\} \\
\mathcal{I}_4(\boldsymbol{r}_2) &= \{w_{\pi_1(i)} \mid (z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)} \in W|_{\boldsymbol{r}_2}\} \\
\mathcal{I}_5(\boldsymbol{r}_2) &= \{w_{\pi_2(i)} \mid (z_i^{(2)} \oplus \rho_i^{(2)}) \cdot v_i^{(2)} \in W|_{\boldsymbol{r}_2}\} \\
\mathcal{I}_6^{(1)} &= \{w_{\pi_1(i)} \mid \text{At least one of } z_i^{(1)}, z_i^{(1)} \cdot u_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in W}\} \\
\mathcal{I}_6^{(2)} &= \{w_{\pi_2(i)} \mid \text{At least one of } z_i^{(2)}, u_i^{(1)} \cdot z_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in W}\}
\end{aligned}$$

Finally, let $k' = k/t$. We set

$$\mathcal{V}(W, \boldsymbol{r}_2) = \begin{cases} \mathcal{I}_2 \cup \mathcal{I}_3 \cup \mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2) \cup \mathcal{I}_6^{(1)} \cup \mathcal{I}_6^{(2)}, & \text{If its size is no more than } k' \\ \emptyset, & \text{Otherwise} \end{cases}$$

For two sets W_1, W_2, we define $W_1 \oplus W_2$ to be $(W_1 \cup W_2) \backslash (W_1 \cap W_2)$. Then $\texttt{parity}(W_1) \oplus \texttt{parity}(W_2) = \texttt{parity}(W_1 \oplus W_2)$. For t sets $W_1, \ldots, W_t$ and a subset $S\{1, \ldots, t\}$, we use W_S to denote $\bigoplus_{i \in S} W_i$. Then we have the following claim.

Claim 2. *For all $W_1, \ldots, W_t$ and $\boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$, $|\cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S, \boldsymbol{r}_2)| \leq k' \cdot t = k$.*

Proof. We first show that for all $W_1, \ldots, W_t$ and $\boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$, if $\mathcal{V}(W_1, \boldsymbol{r}_2), \ldots, \mathcal{V}(W_t, \boldsymbol{r}_2)$ are not empty sets, then for $W_0 := \bigoplus_{i=1}^t W_i$, $\mathcal{V}(W_0, \boldsymbol{r}_2) \subset \cup_{i=1}^t \mathcal{V}(W_i, \boldsymbol{r}_2)$.

Let $(\mathcal{I}_{2,i}, \mathcal{I}_{3,i}, \mathcal{I}_{4,i}(\boldsymbol{r}_2), \mathcal{I}_{5,i}(\boldsymbol{r}_2), \mathcal{I}_{6,i}^{(1)}, \mathcal{I}_{6,i}^{(2)})$ be the sets defined above for W_i for all $i \in \{0, \ldots, t\}$. Since for each wire in W_0, it is also in W_i for some $i \in \{1, \ldots, t\}$, by definition, we have $\mathcal{I}_{2,0} \subset \cup_{i=1}^t \mathcal{I}_{2,i}$. Since $\mathcal{V}(W_i, \boldsymbol{r}_2) \neq \emptyset$ for all $i \in \{1, \ldots, t\}$, we have $\cup_{i=1}^t \mathcal{I}_{2,i} \subset \cup_{i=1}^t \mathcal{V}(W_i, \boldsymbol{r}_2)$, which

implies that $\mathcal{I}_{2,0} \subset \cup_{i=1}^{t} \mathcal{V}(W_i, \boldsymbol{r}_2)$. The same analysis holds for $\mathcal{I}_{3,0}, \mathcal{I}_{4,0}(\boldsymbol{r}_2), \mathcal{I}_{5,0}(\boldsymbol{r}_2), \mathcal{I}_{6,0}^{(1)}, \mathcal{I}_{6,0}^{(2)}$. By definition, $\mathcal{V}(W_0, \boldsymbol{r}_2) \subset \mathcal{I}_{2,0} \cup \mathcal{I}_{3,0} \cup \mathcal{I}_{4,0}(\boldsymbol{r}_2) \cup \mathcal{I}_{5,0}(\boldsymbol{r}_2) \cup \mathcal{I}_{6,0}^{(1)} \cup \mathcal{I}_{6,0}^{(2)}$. Thus, $\mathcal{V}(W_0, \boldsymbol{r}_2) \subset \cup_{i=1}^{t} \mathcal{V}(W_i, \boldsymbol{r}_2)$.

Now come back to the original statement. Let $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S, \boldsymbol{r}_2)$ and $\mathcal{W} = \{W_S \mid |\mathcal{V}(W_S, \boldsymbol{r}_2)| \neq \emptyset\}$. Then $V = \cup_{W_S \in \mathcal{W}} \mathcal{V}(W_S, \boldsymbol{r}_2)$. Let $W'_1, \ldots, W'_{t'}$ be a basis of $\mathcal{W}$. That is for all $W_S \in \mathcal{W}$, W_S is a linear combination of $W'_1, \ldots, W'_{t'}$. Note that $t' \leq t$.

By the above statement, for all $W_S \in \mathcal{W}$, $\mathcal{V}(W_S, \boldsymbol{r}_2) \subset \cup_{i=1}^{t'} \mathcal{V}(W'_i, \boldsymbol{r}_2)$. Thus $V = \cup_{i=1}^{t'} \mathcal{V}(W'_i, \boldsymbol{r}_2)$. Then $|V| = |\cup_{i=1}^{t'} \mathcal{V}(W'_i, \boldsymbol{r}_2)| \leq \sum_{i=1}^{t'} |\mathcal{V}(W'_i, \boldsymbol{r}_2)| \leq t' \cdot k' \leq t \cdot k' = k$. □

7.4.3 Parity-to-Probing Security.

Construction of the Simulator. We first construct the simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$.

- $\mathtt{Sim}_1$ takes $W_1, \ldots, W_t$ as input. $\mathtt{Sim}_1$ randomly samples $\boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$. Then $\mathtt{Sim}_1$ sets $\mathtt{st} = (\boldsymbol{r}_2, W_1, \ldots, W_t)$ and $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S, \boldsymbol{r}_2)$.
- $\mathtt{Sim}_2$ takes $(\mathtt{st}, \mathsf{val}(V))$ as input. For all $w_i \in V$, $\mathtt{Sim}_2$ sets $w'_i = w_i$. For all $w_i \notin V$, $\mathtt{Sim}_2$ sets $w'_i = 0$. Then $\mathtt{Sim}_2$ uses $\boldsymbol{r}_2$ from $\mathtt{st}$, randomly samples $\boldsymbol{r}_1, \boldsymbol{r}_3, \boldsymbol{r}_1^{(1)}, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_3^{(1)}, \boldsymbol{r}_1^{(2)}, \boldsymbol{r}_2^{(2)}, \boldsymbol{r}_3^{(2)}$, and computes all wires in C by viewing $\boldsymbol{w}'$ as the wires in $\tilde{C}$.

  $\mathtt{Sim}_2$ outputs $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$.

Hybrid Arguments. Consider the following hybrids.

Hybrid$_0$: In this hybrid, consider the following generation process.

1. Given the input $\boldsymbol{x}$, we compute $\boldsymbol{y} = O(C(I(\boldsymbol{x})))$.
2. We use the wire values in C to compute $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$.
3. Finally, we output
$$(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), \boldsymbol{y})$$

The output distribution is identical to $(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), O(C(I(\boldsymbol{x}))))$.

Hybrid$_1$: In this hybrid, we compute the wire values in $\tilde{C}$ and use them to compute the wire values in C:

1. Given the input $\boldsymbol{x}$, we randomly sample the random tape for $\tilde{C}$, denoted by $\tilde{\boldsymbol{r}}$. Then we compute $\boldsymbol{y} = \tilde{C}(\boldsymbol{x}; \tilde{\boldsymbol{r}})$. Let $\boldsymbol{w}$ denote the wire values of $\tilde{C}$.
2. We generate the wire values in all layers (see Section 7.4.2) using $\boldsymbol{w}$. Concretely, we faithfully generate the Layer-0 and Layer-1 wire values. Then, we compute $\boldsymbol{z} = \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}$ and $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$ accordingly. Finally, we compute wire values in the rest of layers.
3. We use the wire values in C to compute $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$.
4. Finally, we output
$$(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), \boldsymbol{y})$$

The output distribution of **Hybrid**$_1$ is identical to that of **Hybrid**$_0$.

Hybrid$_2$: In this hybrid, we switch to use $\mathtt{Sim}$ to compute $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$. Concretely,

1. Given the input $\boldsymbol{x}$, we randomly sample the random tape for $\tilde{C}$, denoted by $\tilde{\boldsymbol{r}}$. Then we compute $\boldsymbol{y} = \tilde{C}(\boldsymbol{x}; \tilde{\boldsymbol{r}})$. Let $\boldsymbol{w}$ denote the wire values of $\tilde{C}$.

2. We follow $\mathtt{Sim}_1$ to randomly sample $\boldsymbol{r}_2 \in \{0,1\}^{\lambda_2}$. Then we set $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S, \boldsymbol{r}_2)$. For all $w_i \in V$, we set $w'_i = w_i$. For all $w_i \notin V$, we set $w'_i = 0$. Then we generate the wire values in all layers (see Section 7.4.2) using $\boldsymbol{w}'$. Concretely, we faithfully generate the rest of Layer-0 and Layer-1 wire values. Then, we compute $\boldsymbol{z} = \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}'$ and $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$ accordingly. Finally, we compute wire values in the rest of layers.

3. We use the wire values in C to compute $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$.

4. Finally, we output
$$(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), \boldsymbol{y})$$

Now we argue that the output of $\mathbf{Hybrid}_2$ and the output of $\mathbf{Hybrid}_1$ are statistically close. We prove a stronger statement: we will show that for all $\boldsymbol{w} \in \{0,1\}^{|\tilde{C}|}$ (which may not be a valid transcript of $\tilde{C}(\boldsymbol{x})$), the following two distributions are statistically close:

- $\mathbf{Hybrid}'_1$: The first distribution is obtained by running Step 2, Step 3 in $\mathbf{Hybrid}_1$ and outputting $(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t))$.
- $\mathbf{Hybrid}'_2$: The second distribution is obtained by running Step 2, Step 3 in $\mathbf{Hybrid}_2$ and outputting $(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t))$.

Note that this statement implies that the above two distributions are statistically close given $\boldsymbol{w}$ for all valid transcript $\boldsymbol{w}$ of $\tilde{C}(\boldsymbol{x})$. Since $\boldsymbol{y}$ is a part of the transcript, it implies that the output of $\mathbf{Hybrid}_2$ is statistically close to that of $\mathbf{Hybrid}_1$.

Note that the only difference between $\mathbf{Hybrid}'_1$ and $\mathbf{Hybrid}'_2$ is that $\mathbf{Hybrid}'_1$ uses $\boldsymbol{w}$ to compute all layer wires while $\mathbf{Hybrid}'_2$ uses $\boldsymbol{w}'$ which depends on $\boldsymbol{r}_2$. Let $\boldsymbol{X} = (X_1, \ldots, X_t)$ denote the random variables of the output of $\mathbf{Hybrid}'_1$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_t)$ denote the random variables of the output of $\mathbf{Hybrid}'_2$. By the XOR lemma [Gol11], it is sufficient to show that for all $S \subset \{1, \ldots, t\}$, the statistical distance between $\oplus_{i \in S} X_i$ and $\oplus_{i \in S} Y_i$ is $\epsilon' = \epsilon / 2^{t/2} = e^{-k'/168} + 2^{-k'/42} + 2(2^{2k'/21} + 1)\epsilon^\star$.

To prove this statement, let $\mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_4(\boldsymbol{r}_2), \mathcal{I}_5(\boldsymbol{r}_2), \mathcal{I}_6^{(1)}, \mathcal{I}_6^{(2)}$ be the sets defined above for W_S. Let $k_1 = 25k'/42, k_2 = 17k'/42$. Then $k_1 + k_2 = k'$. We consider two cases:

Case 1: $|\mathcal{I}_2 \cup \mathcal{I}_3| \leq k_1$. In this case, we show that for all $\boldsymbol{r}_2$, the statistical distance between X_S and Y_S is ϵ'. Note that given $\boldsymbol{r}_2$, $\boldsymbol{w}'$ used in $\mathbf{Hybrid}'_2$ is fixed. We consider the following two subcases:

- If $|\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2) \cup \mathcal{I}_6^{(1)} \cup \mathcal{I}_6^{(2)}| \leq k_2$, then $|\mathcal{I}_2 \cup \mathcal{I}_3 \cup \mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2) \cup \mathcal{I}_6^{(1)} \cup \mathcal{I}_6^{(2)}| \leq k'$. We have $\mathcal{V}(W_S, \boldsymbol{r}_2) = \mathcal{I}_2 \cup \mathcal{I}_3 \cup \mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2) \cup \mathcal{I}_6^{(1)} \cup \mathcal{I}_6^{(2)}$. In this case, $\mathtt{parity}(W_S)$ only depends on wires in $\mathcal{V}(W_S, \boldsymbol{r}_2)$ in $\tilde{C}$, and $w_i = w'_i$ for all $w_i \in \mathcal{V}(W_S, \boldsymbol{r}_2)$. Thus, X_S and Y_S are identically distributed.

- Otherwise, let $k_{2,1} = k_2/17, k_{2,2} = 8k_2/17, k_{2,3} = 8k_2/17$. Then at least one of the following three cases happen: (1) $|\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2)| \geq k_{2,1}$, (2) $|\mathcal{I}_6^{(1)} \backslash (\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2))| \geq k_{2,2}$, (3) $|\mathcal{I}_6^{(2)} \backslash (\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2))| \geq k_{2,3}$.

– If (1) happens, we fix $\boldsymbol{r}_1, \boldsymbol{r}_3, \boldsymbol{r}_1^{(1)}, \boldsymbol{r}_1^{(2)}$. Let $\mathcal{J}^{(1)} = \{i \mid (z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)} \in W_S|_{\boldsymbol{r}_2}\}$ and $\mathcal{J}^{(2)} = \{i \mid (z_i^{(2)} \oplus \rho_i^{(2)}) \cdot v_i^{(2)} \in W_S|_{\boldsymbol{r}_2}\}$. Then $\texttt{parity}(W_S)$ can be written as

$$(\oplus_{i\in\mathcal{J}^{(1)}}\rho_i^{(1)} \cdot v_i^{(1)}) \oplus (\oplus_{i\in\mathcal{J}^{(2)}}\rho_i^{(2)} \cdot v_i^{(2)}) \oplus f(\boldsymbol{r}_3^{(1)}, \boldsymbol{r}_3^{(2)}) \oplus g(\boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)})$$

for some linear function f and some arbitrary function g. In particular, $|\mathcal{J}^{(1)}| + |\mathcal{J}^{(2)}| \geq |\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2)| \geq k_{2,1}$. By Corollary 2, $\texttt{parity}(W_S)$ is statistically close to uniform with error $2^{-k_{2,1}-1} + \epsilon^\star$. Note that this analysis works for all $\boldsymbol{w}$. Since $\mathbf{Hybrid}'_1$ and $\mathbf{Hybrid}'_2$ only differ in using $\boldsymbol{w}$ and $\boldsymbol{w}'$, the statistical distance between X_S and Y_S is at most $2^{-k_{2,1}} + 2\epsilon^\star \leq \epsilon'$.

– If (2) happens, we fix $\boldsymbol{r}_1, \boldsymbol{r}_3, \boldsymbol{r}_3^{(1)}, \boldsymbol{r}_1^{(2)}, \boldsymbol{r}_2^{(2)}, \boldsymbol{r}_3^{(2)}$. Then $\texttt{parity}(W_S)$ can be written as

$$(\oplus_{i\in\mathcal{J}}u_i^{(1)} \cdot v_i^{(1)}) \oplus f(\boldsymbol{r}_1^{(1)}) \oplus g(\boldsymbol{r}_2^{(1)})$$

for some set $\mathcal{J} \subset \{1, \ldots, |\tilde{C}|\}$, some linear function f, and some arbitrary function g. Consider

$$\mathcal{J}' = \{i \mid \text{At least one of } z_i^{(1)}, z_i^{(1)} \cdot u_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in } W_S \text{ but } (z_i^{(1)} \oplus \rho_i^{(1)}) \cdot v_i^{(1)} \notin W_S|_{\boldsymbol{r}_2}\}.$$

Note that $|\mathcal{J}'| \geq |\mathcal{I}_6^{(1)} \backslash (\mathcal{I}_4(\boldsymbol{r}_2) \cup \mathcal{I}_5(\boldsymbol{r}_2))| \geq k_{2,2}$.

For each $i \in \mathcal{J}'$, the coefficient of $u_i^{(1)} \cdot v_i^{(1)}$ is a function on $u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)}$, denoted by $f_i(u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)})$. One can verify that $f_i(u_i^{(2)}, v_i^{(2)}, 0) \not\equiv 0$ and $f_i(u_i^{(2)}, v_i^{(2)}, 1) \not\equiv 0$. We show that for all $\boldsymbol{w}$, when $\boldsymbol{r}_1^{(2)}, \boldsymbol{r}_2^{(2)}$ are randomly sampled, with overwhelming probability, $\sum_{i\in\mathcal{J}'} f_i(u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)}) \geq k_{2,2}/8$.

Claim 3. *Let $c > 0, k > 0$ be integers. Let $f_1, \ldots, f_k : \{0,1\}^c \to \{0,1\}$ be k non-zero functions. Suppose $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_c$ are independent random variables over $\{0,1\}^k$ such that $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_c)$ is statistically close to uniform with distance ϵ. Then*

$$\Pr[\sum_{i=1}^{k} f_i(X_{1,i}, \ldots, X_{c,i}) \geq \frac{k}{2^{c+1}}] \geq 1 - e^{-\frac{k}{2^{c+3}}} - \epsilon.$$

Proof. Let $\boldsymbol{X}'_1, \ldots, \boldsymbol{X}'_c$ be independent and uniform random variables over $\{0,1\}^k$. Since $f_i \not\equiv 0$, we have $\Pr[f_i(X'_{1,i}, \ldots, X'_{c,i}) = 1] \geq 1/2^c$. By Chernoff bound,

$$\Pr[\sum_{i=1}^{k} f_i(X'_{1,i}, \ldots, X'_{c,i}) \geq \frac{k}{2^{c+1}}] \geq 1 - e^{-\frac{k}{2^{c+3}}}.$$

Since $(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_c)$ is statistically close to $(\boldsymbol{X}'_1, \ldots, \boldsymbol{X}'_c)$ with distance ϵ, we have

$$\Pr[\sum_{i=1}^{k} f_i(X_{1,i}, \ldots, X_{c,i}) \geq \frac{k}{2^{c+1}}] \geq \Pr[\sum_{i=1}^{k} f_i(X'_{1,i}, \ldots, X'_{c,i}) \geq \frac{k}{2^{c+1}}] - \epsilon \geq 1 - e^{-\frac{k}{2^{c+3}}} - \epsilon.$$

□

Now we focus on the first $k_{2,2}$ entries in $\mathcal{J}'$, denoted by $\mathcal{J}'[k_{2,2}]$. Let $\boldsymbol{u}^{(2)}_{\mathcal{J}'[k_{2,2}]}, \boldsymbol{v}^{(2)}_{\mathcal{J}'[k_{2,2}]}$ denote the subvectors of $\boldsymbol{u}^{(2)}, \boldsymbol{v}^{(2)}$ with indices restricted in $\mathcal{J}'[k_{2,2}]$. By the XOR lemma [Gol11], $\boldsymbol{v}^{(2)}_{\mathcal{J}'[k_{2,2}]}$ is statistically close to uniform with distance $2^{k_{2,2}/2} \cdot \epsilon^\star$. Thus $(\boldsymbol{u}^{(2)}_{\mathcal{J}'[k_{2,2}]}, \boldsymbol{v}^{(2)}_{\mathcal{J}'[k_{2,2}]})$ is statistically close to uniform with distance $2^{k_{2,2}/2} \cdot \epsilon^\star$. By Claim 3, with probability $1 - e^{-k_{2,2}/32} - 2^{k_{2,2}/2} \cdot \epsilon^\star$, $\sum_{i \in \mathcal{J}'} f_i(u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)}) \geq k_{2,2}/8$. This implies that $|\mathcal{J}| \geq k_{2,2}/8$. In this case, by Corollary 2, $\mathtt{parity}(W_S)$ is statistically close to uniform with error $2^{-k_{2,2}/8-1} + \epsilon^\star$.

Thus, for all $\boldsymbol{w}$ and $\boldsymbol{r}_2$, $\mathtt{parity}(W_S)$ is statistically close to uniform with error $(e^{-k_{2,2}/32} + 2^{k_{2,2}/2} \cdot \epsilon^\star)/2 + 2^{-k_{2,2}/8-1} + \epsilon^\star$. Note that this analysis works for all $\boldsymbol{w}$. Since $\mathbf{Hybrid}'_1$ and $\mathbf{Hybrid}'_2$ only differ in using $\boldsymbol{w}$ and $\boldsymbol{w}'$, the statistical distance between X_S and Y_S is at most $e^{-k_{2,2}/32} + 2^{k_{2,2}/2} \cdot \epsilon^\star + 2^{-k_{2,2}/8} + 2\epsilon^\star \leq \epsilon'$.

– If (3) happens, following the same argument, the statistical distance between X_S and Y_S is at most $e^{-k_{2,3}/32} + 2^{k_{2,3}/2} \cdot \epsilon^\star + 2^{-k_{2,3}/8} + 2\epsilon^\star \leq \epsilon'$.

Case 2: $|\mathcal{I}_2 \cup \mathcal{I}_3| > k_1$. Let $k_{1,1} = 24k_1/25$ and $k_{1,2} = k_1/25$. In this case, at least one of the following subcases happen: (1) $|\mathcal{I}_2| \geq k_{1,1}$, (2) $|\mathcal{I}_3 \backslash \mathcal{I}_2| \geq k_{1,2}$.

- If (1) happens, we fix $\boldsymbol{r}_1, \boldsymbol{r}_1^{(1)}, \boldsymbol{r}_3^{(1)}, \boldsymbol{r}_1^{(2)}, \boldsymbol{r}_3^{(2)}$. Let $\mathcal{J}' = \{i \mid \phi_i \not\equiv 0\}$. Then $|\mathcal{J}'| = |\mathcal{I}_2|$, and $\mathtt{parity}(W_S)$ can be written as

$$(\oplus_{i \in \mathcal{J}'} \rho_i \cdot \phi_i(v_i, v_i^{(1)}, v_i^{(2)})) \oplus f(\boldsymbol{r}_3) \oplus g(\boldsymbol{r}_2, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)})$$

for some linear function f and some arbitrary function g.

For a function $\phi(x, y, z)$, we say ϕ depends on an input x if the coefficient of x is not 0. Note that if $\phi \not\equiv 0$, then ϕ depends on at least one input. Without loss of generality, suppose at least $1/3$ fraction of $\{\phi_i\}_{i \in \mathcal{J}'}$ depend on the first input. Let $\mathcal{J}'_1 \subset \mathcal{J}'$ denote the set of indices of such functions. We may write $\phi_i(v_i, v_i^{(1)}, v_i^{(2)}) = v_i \cdot \psi_i(v_i^{(1)}, v_i^{(2)}) \oplus h_i(v_i^{(1)}, v_i^{(2)})$, where $\psi_i \not\equiv 0$ for all $i \in \mathcal{J}'_1$. Then

$$\mathtt{parity}(W_S) = (\oplus_{i \in \mathcal{J}'} \psi_i(v_i^{(1)}, v_i^{(2)}) \cdot \rho_i \cdot v_i) \oplus (\oplus_{i \in \mathcal{J}'} h_i(v_i^{(1)}, v_i^{(2)}) \cdot \rho_i) \oplus f(\boldsymbol{r}_3) \oplus g(\boldsymbol{r}_2, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)}).$$

Given $\boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)}$, $(\oplus_{i \in \mathcal{J}'} h_i(v_i^{(1)}, v_i^{(2)}) \cdot \rho_i) \oplus f(\boldsymbol{r}_3)$ is a linear function on $\boldsymbol{r}_3$, denoted by $f'(\boldsymbol{r}_3)$, $g(\boldsymbol{r}_2, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_2^{(2)})$ is some arbitrary function on $\boldsymbol{r}_2$, denoted by $g'(\boldsymbol{r}_2)$, and there exists some subset $\mathcal{J} \subset \mathcal{J}'$ such that

$$\mathtt{parity}(W_S) = (\oplus_{i \in \mathcal{J}} \rho_i \cdot v_i) \oplus f'(\boldsymbol{r}_3) \oplus g'(\boldsymbol{r}_2).$$

Now we focus on the first $k_{1,1}/3$ entries in $\mathcal{J}'_1$, denoted by $\mathcal{J}'_1[k_{1,1}/3]$. Let $\boldsymbol{v}^{(1)}_{\mathcal{J}'_1[k_{1,1}/3]}, \boldsymbol{v}^{(2)}_{\mathcal{J}'_1[k_{1,1}/3]}$ denote the subvectors of $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}$ with indices restricted in $\mathcal{J}'_1[k_{1,1}/3]$. By the XOR lemma [Gol11], each of $\boldsymbol{v}^{(1)}_{\mathcal{J}'_1[k_{1,1}/3]}, \boldsymbol{v}^{(2)}_{\mathcal{J}'_1[k_{1,1}/3]}$ is statistically close to uniform with distance $2^{k_{1,1}/6} \cdot \epsilon^\star$. Thus $(\boldsymbol{v}^{(1)}_{\mathcal{J}'[k_{1,1}/3]}, \boldsymbol{v}^{(2)}_{\mathcal{J}'[k_{1,1}/3]})$ is statistically close to uniform with distance $2^{k_{1,1}/6+1} \cdot \epsilon^\star$. By Claim 3, with probability $1 - e^{-k_{1,1}/96} - 2^{k_{1,1}/6+1} \cdot \epsilon^\star$, $\sum_{i \in \mathcal{J}'_1[k_{1,1}/3]} \phi_i(v_i^{(1)}, v_i^{(2)}) \geq k_{1,1}/24$. This implies

that $|\mathcal{J}| \geq k_{1,1}/24$. In this case, by Corollary 2, $\mathtt{parity}(W_S)$ is statistically close to uniform with error $2^{-k_{1,1}/24-1} + \epsilon^\star$.

Thus, $\mathtt{parity}(W_S)$ is statistically close to uniform with error $(e^{-k_{1,1}/96} + 2^{k_{1,1}/6+1} \cdot \epsilon^\star)/2 + 2^{-k_{1,1}/24-1} + \epsilon^\star$. Note that this analysis works for all linear function f and all function g. Since $\mathbf{Hybrid}'_1$ and $\mathbf{Hybrid}'_2$ only differ in using $\boldsymbol{w}$ and $\boldsymbol{w}'$, and $\boldsymbol{w}'$ only depends on $\boldsymbol{w}, \boldsymbol{r}_2, W_S$, X_S and Y_S only differ in the function g. Thus, the statistical distance between X_S and Y_S is at most $e^{-k_{1,1}/96} + 2^{k_{1,1}/6+1} \cdot \epsilon^\star + 2^{-k_{1,1}/24} + 2\epsilon^\star \leq \epsilon'$.

- If (2) happens, we fix $\boldsymbol{r}_3, \boldsymbol{r}_1^{(1)}, \boldsymbol{r}_2^{(1)}, \boldsymbol{r}_3^{(1)}, \boldsymbol{r}_1^{(2)}, \boldsymbol{r}_2^{(2)}, \boldsymbol{r}_3^{(2)}$. Then $\mathtt{parity}(W_S)$ can be written as

$$(\oplus_{i \in \mathcal{J}} u_i \cdot v_i) \oplus f(\boldsymbol{r}_1) \oplus g(\boldsymbol{r}_2)$$

for some set $\mathcal{J} \subset \{1, \ldots, |\tilde{C}|\}$, some linear function f, and some arbitrary function g. Consider

$$\mathcal{J}' = \{i \mid z_i \in W_S \text{ and } \phi_i \equiv 0\}.$$

Note that $|\mathcal{J}'| \geq |\mathcal{I}_2 \backslash \mathcal{I}_3| \geq k_{1,2}$.

For each $i \in \mathcal{J}'$, the coefficient of $u_i \cdot v_i$ is 1. Thus $|\mathcal{J}| \geq |\mathcal{J}'| \geq k_{1,2}$. By Corollary 2, $\mathtt{parity}(W_S)$ is statistically close to uniform with error $2^{-k_{1,2}-1} + \epsilon^\star$. Note that this analysis works for all linear function f and all function g. Since $\mathbf{Hybrid}'_1$ and $\mathbf{Hybrid}'_2$ only differ in using $\boldsymbol{w}$ and $\boldsymbol{w}'$, and $\boldsymbol{w}'$ only depends on $\boldsymbol{w}, \boldsymbol{r}_2, W_S$, X_S and Y_S only differ in the function g. Thus, the statistical distance between X_S and Y_S is at most $2^{-k_{1,2}} + 2\epsilon^\star \leq \epsilon'$.

In summary, the statistical distance between X_S and Y_S in all cases is at most ϵ'. Therefore, the output of $\mathbf{Hybrid}_2$ is statistically close to the output of $\mathbf{Hybrid}_1$ with error ϵ.

Note that the distribution of the output of $\mathbf{Hybrid}_2$ is identical to

$$(\mathtt{Sim}_2(\mathtt{st}, \mathsf{val}(V)), \tilde{C}(\boldsymbol{x})) : (\mathtt{st}, V) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t).$$

Thus,

$$\begin{aligned} & (\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), O(C(I(\boldsymbol{x})))) \\ \approx_\epsilon \quad & (\mathtt{Sim}_2(\mathtt{st}, \mathsf{val}(V)), \tilde{C}(\boldsymbol{x})) : (\mathtt{st}, V) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t). \end{aligned}$$

This finishes the proof of Theorem 6.

7.5 Towards Derandomized t-Parity-Tolerant Circuits

We first analyse the randomness complexity of our construction of the derandomized parity-to-probing circuit compiler. Let $G_1 : \{0,1\}^{\lambda_1} \to \{0,1\}^{|\tilde{C}|}$ be the linear k-wise independent PRG and $G_2 : \{0,1\}^{\lambda_2} \to \{0,1\}^{|\tilde{C}|}$ be the parity-resilient PRG in Figure 5.

- The input encoder I uses $\lambda_1 + \lambda_2$ random bits.
- The circuit C uses $5\lambda_1 + 2\lambda_2$ random bits in Step 1. Further more, for each random wire in $\tilde{C}$, C generates a random bit. Let $\mathtt{RC}(\tilde{C})$ denote the randomness complexity of the underlying circuit $\tilde{C}$. Then C uses $5\lambda_1 + 2\lambda_2 + \mathtt{RC}(\tilde{C})$ random bits.

In total, (I, C, O) uses $6\lambda_1 + 3\lambda_2 + \mathtt{RC}(\tilde{C})$ random bits.

It is known that (strong) linear k-wise independent PRGs with output size $|\tilde{C}|$ exist when $\lambda_1 = O(k \cdot \log |\tilde{C}|)$ (See an example in [GIS22]). From [NN90], parity-resilient PRGs with error $\epsilon^\star$ exist when $\lambda_2 = O(\log |\tilde{C}| + \log 1/\epsilon^\star)$. Therefore, the randomness complexity of (I, C, O) is $O(k \cdot \log |\tilde{C}| + \log 1/\epsilon^\star)$.

Now we can use our derandomized (t, k, ϵ)-parity-to-probing circuit in Figure 3. As for k-probing-tolerant circuit, we borrow the following theorem from [IKL+13, CGZ20].

Theorem 7 ([IKL+13, CGZ20])**.** *There is a polynomial-time circuit compiler that takes as input $(C_f, 1^k)$, where C_f is a (possibly randomized) circuit of size s that computes f, and outputs a k-probing-tolerant implementation of f with circuit size poly(k, s) and randomness complexity poly$(k, \log s)$ plus the randomness complexity of C_f.*

Then from the construction in Figure 3, for all function f with circuit size s, there is a (t, ϵ)-parity-tolerant implementation of f, where

$$\epsilon = 2^{t/2}(e^{-k/168t} + 2^{-k/42t} + 2(2^{2k/21t} + 1)\epsilon^\star),$$

with randomness complexity $\mathrm{poly}(k, \log s, \log 1/\epsilon^\star)$ plus the randomness complexity of C_f. Let κ denote the security parameter. To achieve $\epsilon = 2^{-\kappa}$, the randomness complexity is $\mathrm{poly}(t, \kappa, \log s)$ plus the randomness complexity of C_f. We have the following corollary.

Corollary 3. *There is a polynomial-time circuit compiler that takes as input $(C_f, 1^t, 1^\kappa)$, where C_f is a (possibly randomized) circuit of size s that computes f, and outputs a $(t, 2^{-\kappa})$-parity-tolerant implementation of f with circuit size poly(t, κ, s) and randomness complexity poly$(t, \kappa, \log s)$ plus the randomness complexity of C_f.*

On the Computational Complexity of the Simulator. When proving the security of our derandomized parity-to-probing circuits, the computation complexity of the simulator is exponential in t, the number of parity queries. The simulator for the parity-tolerant circuits inherits the exponential computation complexity in t. We note that the same issue appears in [IS24] and the authors in [IS24] also provide evidence showing that such an inefficient simulation might be inherent.

7.6 Deterministic Randomness Extractor for t-Parity-Leakage Source

Finally, we give a concrete instantiation of Definition 12 for t-parity-leakage source. The construction is as follows: Let $\kappa = \log \frac{1}{\epsilon}$. The extractor $\mathtt{Ext}$ takes as input $(\boldsymbol{u}, \boldsymbol{v})$ where $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^{t+\kappa}$ and outputs the inner-product between $\boldsymbol{u}$ and $\boldsymbol{v}$. We show that this simple construction suffices.

To prove the security of $\mathtt{Ext}$, we first construct the simulator $\mathtt{Sim}$ required by Definition 12. Let $W_1, \ldots, W_t$ be the input t parity-leakage queries. Let $W'_i = W_i \cap \{u_1, \ldots, u_{t+\kappa}\}$ for all $i \in \{1, \ldots, t\}$. $\mathtt{Sim}$ first randomly samples $\boldsymbol{v} \in \{0,1\}^{t+\kappa}$. Given $\boldsymbol{v}$, the output of $\mathtt{Ext}$ is a linear combination of $\boldsymbol{u}$. Let $W(\boldsymbol{v})$ be a subset of $\{u_1, \ldots, u_{t+\kappa}\}$ such that given $\boldsymbol{v}$, the output of $\mathtt{Ext}$ is $\mathtt{parity}(W(\boldsymbol{v}))$.

Now if $W(\boldsymbol{v})$ is equal to $\oplus_{i \in S} W'_i$ for some $S \in \{1, \ldots, t\}$, $\mathtt{Sim}$ outputs $\mathtt{fail}$. Otherwise, $\mathtt{Sim}$ randomly samples $\boldsymbol{u}$ and computes $\mathtt{parity}(W'_1), \ldots, \mathtt{parity}(W'_t)$. Given $\mathtt{parity}(W'_1), \ldots, \mathtt{parity}(W'_t)$, $\mathtt{Sim}$ randomly samples $\boldsymbol{u}^{(0)}$ such that $\mathtt{parity}(W(\boldsymbol{v})) = 0$ and samples $\boldsymbol{u}^{(1)}$ such that $\mathtt{parity}(W(\boldsymbol{v})) = 1$. $\mathtt{Sim}$ sets $r_0 = (\boldsymbol{u}^{(0)}, \boldsymbol{v}), r_1 = (\boldsymbol{u}^{(1)}, \boldsymbol{v})$ and computes ℓ to be the leakage result by applying the leakage functions on r_0. Finally, $\mathtt{Sim}$ outputs (ℓ, r_0, r_1).

Now consider the following hybrids.

Hybrid$_0$: In the first hybrid, we randomly sample $r \leftarrow \{0,1\}^{2(t+\kappa)}$ and outputs $(\{\mathtt{parity}(W_i)\}_{i=1}^t, \mathtt{Ext}(r), r)$. This corresponds to the LHS in Definition 12.

Hybrid$_1$: In this hybrid, we first randomly sample $\boldsymbol{v} \in \{0,1\}^{t+\kappa}$. Let $W_0(\boldsymbol{v}), W_1', \ldots, W_t'$ be the sets defined above. If there exists $S \subset \{1,\ldots,t\}$ such that $W(\boldsymbol{v}) = \oplus_{i\in S} W_i'$, output $\mathtt{fail}$. Otherwise, randomly sample $\boldsymbol{u}$ and compute the output as **Hybrid**$_0$.

Note that $W(\boldsymbol{v})$ is a random subset of $\{u_1,\ldots,u_{t+\kappa}\}$ where as there are at most 2^t possible outcome from $\oplus_{i\in S} W_i'$. Thus, when $\boldsymbol{v}$ is randomly sampled, the probability that $W(\boldsymbol{v}) = \oplus_{i\in S} W_i'$ for some S is at most $2^t/2^{t+\kappa} = 1/2^\kappa$. Thus, the statistical distance between **Hybrid**$_0$ and **Hybrid**$_1$ is at most $1/2^\kappa$.

Hybrid$_2$: In this hybrid, when there is no $S \subset \{1,\ldots,t\}$ such that $W(\boldsymbol{v}) = \oplus_{i\in S} W_i'$, output $\mathtt{fail}$, $\mathtt{parity}(W(\boldsymbol{v}))$ is uniformly random given $\mathtt{parity}(W_1'),\ldots,\mathtt{parity}(W_t')$. After sampling $\boldsymbol{v}$, we randomly sample $\boldsymbol{u}$ and compute $\mathtt{parity}(W_1'),\ldots,\mathtt{parity}(W_t')$. Then we randomly sample $b \in \{0,1\}$ and set $\mathtt{parity}(W(\boldsymbol{v})) = b$. We randomly sample $\boldsymbol{u}^{(b)}$ given $\mathtt{parity}(W(\boldsymbol{v})), \mathtt{parity}(W_1'), \ldots, \mathtt{parity}(W_t')$. Finally, we compute the output as **Hybrid**$_0$. The distribution of **Hybrid**$_2$ is identical to that of **Hybrid**$_1$.

Hybrid$_3$: In this hybrid, we compute both $\boldsymbol{u}^{(0)}$ and $\boldsymbol{u}^{(1)}$. Then we randomly sample b and set $r = (\boldsymbol{u}^{(b)}, \boldsymbol{v})$. The distribution of **Hybrid**$_3$ is identical to that of **Hybrid**$_2$. This corresponds to the RHS in Definition 12.

By setting $\kappa = \log \frac{1}{\epsilon}$, $\mathtt{Ext}$ is a deterministic randomness extractor for t-D1-leakage source with efficient reverse sampling and statistical error ϵ.

Now from Theorem 6, we have the following corollary.

Corollary 4. *Let κ denote the security parameter. Let $t \in \mathtt{N}$ be the leakage bound parameter. Assume that there is a pseudo-random generator $\mathtt{PRG}$ with seed length κ that is secure against adversaries with circuit size poly$(2^t, \kappa)$. Then there exists a t-parity-LRC compiler $\mathtt{Comp} = (\mathcal{T}_C, \mathcal{T}_s)$ such that*

- *For every polynomial adversary bound $S(\kappa)$, there is a simulator bound $S_0(\kappa) =$ poly$(2^t, S(\kappa), \kappa)$ such that $\mathtt{Comp}$ is an $(\mathcal{L}, S, S_0, S)$-LRC compiler.*
- *For all stateful circuit C, $C' \leftarrow \mathcal{T}_C(1^\kappa, C)$ satisfies that C' does not require random tape except the first invocation.*

Acknowledgments

Y. Ishai was supported by ERC grant NTSC (742754), BSF grant 2022370, ISF grant 2774/20, and ISF-NSFC grant 3127/23. Y. Song was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003, and the Shanghai Qi Zhi Institute Innovation Program SQZ202313.

References

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *Advances*

in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.

[BBCM95] C.H. Bennett, G. Brassard, C. Crepeau, and U.M. Maurer. Generalized privacy amplification. *IEEE Transactions on Information Theory*, 41(6):1915–1923, 1995.

[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 Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BCRT23] Sonia Belaïd, Gaëtan Cassiers, Matthieu Rivain, and Abdul Rahman Taleb. Unifying freedom and separation for tight probing-secure composition. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 440–472, Cham, 2023. Springer Nature Switzerland.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). 1988.

[BIS21] Andrej Bogdanov, Yuval Ishai, and Akshayaram Srinivasan. Unconditionally secure computation against low-complexity leakage. *J. Cryptol.*, 34(4):38, 2021.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 501–510. IEEE Computer Society, 2010.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). 1988.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In *TCC 2020, Part II*, pages 652–682, 2020.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 342–375, Cham, 2020. Springer International Publishing.

[CH94] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 425–438. Springer, 1994.

[CLW06] Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 2006.

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

[DP07] Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 227–237. IEEE Computer Society, 2007.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 654–663, New York, NY, USA, 2005. Association for Computing Machinery.

[Dzi06] Stefan Dziembowski. On forward-secure storage. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 251–270. Springer, 2006.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 135–156, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[FRR+14] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from computationally bounded and noisy leakage. *SIAM J. Comput.*, 43(5):1564–1614, 2014.

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

[GIS22] Vipul Goyal, Yuval Ishai, and Yifan Song. Private circuits with quasilinear randomness. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 192–221, Cham, 2022. Springer International Publishing.

[GIW17] Daniel Genkin, Yuval Ishai, and Mor Weiss. How to construct a leakage-resilient (stateless) trusted party. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 209–244, Cham, 2017. Springer International Publishing.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.

[Gol11] Oded Goldreich. *Three XOR-Lemmas — An Exposition*, pages 248–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[GR15] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. *SIAM J. Comput.*, 44(5):1480–1549, 2015.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust Pseudorandom Generators. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 576–588, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[IS24] Yuval Ishai and Yifan Song. Leakage-tolerant circuits. EUROCRYPT, 2024.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[NN90] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 213–223, New York, NY, USA, 1990. Association for Computing Machinery.

[Rot12] Guy N. Rothblum. How to compute under ${\cal{AC}}^{\sf0}$ leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 552–569. Springer, 2012.

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

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

A Related Leakage Models

In this section, we compare our bounded-communication leakage (BCL) model with the related "only computation leaks" [MR04] (OCL) model and the feasibility result of Goldwasser and Rothblum [GR15] for this model.

Comparison with OCL. The BCL model and the OCL model share several similarities: They both partition the circuit into several parts and allow arbitrary bounded-length leakage from each part. However, the following differences make them technically incomparable.

- **The number of parts:** BCL can partition the computation into two parts, whereas in OCL the computation may be split into a bigger number of parts. Note that 2-part leakage with t bits per part is stronger than m-part leakage ($m \geq 2$) with $2t/m$ bits per part. But this is not clear when comparing with m-part leakage also with t bits per part.
- **Restrictions on partitions:** In BCL, the partition of *wires* of C can be arbitrary and in particular each part may not correspond to a valid sub-circuit. On the other hand, each part in OCL should be a valid sub-computation or sub-circuit.
- **Leakage from different parts:** In BCL, the leakage from the two parts can be arbitrarily interleaved and interactive, whereas in OCL, leakage from different components needs to follow *the computation order*.

All these differences make a (stateful) LRC for OCL unsuitable for our purposes. While one may attempt to convert an OCL LRC to a multiparty protocol where each party performs one sub-computation component, this does not work since the leakage in OCL needs to be done following the computation order and does not allow arbitrary interleaved and interactive leakage among different parts/parties. Also it is not clear whether there exists an OCL LRC that only has 2 parts.

In the other direction, a (stateful) LRC for BCL cannot be converted to a (stateful) LRC for OCL either, since each part in BCL may not correspond to a valid computation. An interesting open question is to study the leakage which partitions C into sub-computations (as OCL) but allows interleaved and interactive leakage among all components (as BCL). This would naturally lead to an information-theoretic multiparty computation protocol that allows interleaved and interactive leakage from each party.

Comparison with [GR15]. Given the discussion above, the stateful LRC for OCL constructed in [GR15] does not imply a stateful LRC for BCL, nor a solution towards our malware protection application. As for our second main result on deterministic stateful LRC for simple bounded-communication leakage (parities or disjunctions), although these leakage classes are covered by OCL, the LRC construction from [GR15] inherently requires fresh randomness in each invocation, and generating this randomness deterministically seems challenging.