Leakage-Tolerant Circuits

Yuval Ishai[1] and Yifan Song[2,3]

[1] Technion, Israel.
yuvali@cs.technion.ac.il
[2] Institute for Theoretical Computer Science, Institute for Interdisciplinary Information Sciences, Tsinghua University, P. R. China.
yfsong@mail.tsinghua.edu.cn
[3] Shanghai Qi Zhi Institute, P. R. China.

Abstract. A *leakage-resilient circuit* for $f : \{0,1\}^n \to \{0,1\}^m$ is a randomized Boolean circuit C mapping a randomized encoding of an input x to an encoding of $y = f(x)$, such that applying any leakage function $L \in \mathcal{L}$ to the wires of C reveals essentially nothing about x. A *leakage-tolerant circuit* achieves the stronger guarantee that even when x and y are not protected by any encoding, the output of L can be simulated by applying some $L' \in \mathcal{L}$ to x and y alone. Thus, C is as secure as an ideal hardware implementation of f with respect to leakage from $\mathcal{L}$.

Leakage-resilient circuits were constructed for low-complexity classes $\mathcal{L}$, including (length-t output) $\mathcal{AC}0$ functions, parities, and functions with bounded communication complexity. In contrast, leakage-*tolerant* circuits were only known for the simple case of *probing* leakage, where L outputs the values of t wires in C.

We initiate a systematic study of leakage-tolerant circuits for natural classes $\mathcal{L}$ of *global* leakage functions, obtaining the following main results.

- **Leakage-tolerant circuits for depth-1 leakage.** Every circuit C_f for f can be efficiently compiled into an $\mathcal{L}$-tolerant circuit C for f, where $\mathcal{L}$ includes all leakage functions L that output either t *parities* or t *disjunctions* (alternatively, conjunctions) of any number of wires or their negations. In the case of parities, our simulator runs in $2^{O(t)}$ time. We provide partial evidence that this may be inherent.
- **Application to stateful leakage-resilient circuits.** Using a general transformation from leakage-tolerant circuits, we obtain the first construction of *stateful* t-leakage-resilient circuits that tolerate a *continuous* parity leakage, and the first such construction for disjunction/conjunction leakage in which the circuit size grows sub-quadratically with t. Interestingly, here we can obtain poly(t)-time simulation even in the case of parities.

1 Introduction

A dream goal in cryptography is to design hardware that can perform general computations on secret inputs while resisting arbitrary side-channel attacks that reveal partial information about the computation. An extreme version of this goal requires ideal (and unrealizable) flavors of obfuscation. However, when settling for security against limited classes of side channels, this goal becomes realizable and has been the topic of a large body of theoretical and applied work.

A simple formal model for studying this problem is a *leakage-resilient circuit* (LRC) [ISW03]. Let $\mathcal{L}$ be a class of leakage functions with t-bit output. 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 a randomized encoding (or secret-sharing) $\hat{x}$ of an input x to an encoding $\hat{y}$ of $y = f(x)$, such that applying any leakage function $L \in \mathcal{L}$ to the wires of C reveals essentially nothing about x in an information-theoretic sense. Assuming the availability of an ideal (leak-free) input encoder and output decoder[4], this gives an end-to-end solution to the problem of protecting the computation of f against leakage class $\mathcal{L}$.

[4] To avoid trivial solutions in which the computation is carried out by the input encoder or the output decoder, it is required that the encoding of the inputs and outputs be *universal* in the sense that it depends only on their length and not on C.

There is a long line of works studying different flavors of the LRC question. The vast majority of these works focus on the simple class $\mathcal{L}$ of *probing* leakage, where L can only output the values of t (physical) wires in C. This baseline model has been extended in two orthogonal ways:

From local to global leakage. While the "local" leakage resilience captured by the probing model is practically relevant and implies resilience against certain kinds of noisy leakage [DDF19], it does not capture global leakage that applies *jointly* to an unbounded number of wires. This type of leakage may arise in realistic side-channels attacks such as timing or power analysis that measure global information about the computation. The study of such global leakage functions was initiated in [MR04,FRR+14]. Most relevant to the current work are constructions of LRCs against $\mathcal{AC}0$ leakage, consisting of constant-depth polynomial-size circuits over $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ gates with unbounded fan-in [FRR+14,Rot12,BIS21,BDF+22], and LRCs against *parity* leakage and (more broadly) bounded-communication leakage [GIM+16,GIW17]. See Section 1.2 for more work in this direction.

From stateless to stateful circuits. The simplicity of the LRC model makes it an attractive object of study. However, this model is limited in two significant ways. First, it assumes *leak-free* input encoder and output decoder. Second, it is limited to a *one-shot* scenario where a single computation is performed and is subject to a small amount of leakage. In contrast, real-world computers or embedded devices need to maintain a persistent secret state, and are subject to *continuous* leakage which may be unbounded over time. The *stateful* variant of LRCs [ISW03,DP08,FRR+14] addresses both limitations by considering circuits with memory that can be initialized with an encoded secret state, and can then "refresh" the encoded state in each invocation. The main challenge in converting stateless LRCs into stateful ones is to ensure sufficient independence between the encoded input and the encoded output conditioned on the leakage. While there is a variety of techniques for achieving this in the case of probing leakage [RP10,CPRR13,BBD+16,CGPZ16,GPRV21], it is not clear how to extend them even to very simple types of global leakage. Indeed, most stateless LRCs are not known to have stateful analogues without relying on leak-free components [FRR+14,DF12,MV13,GIM+16,BDF+22].

Leakage-Tolerant Circuits. In this work we focus on a different extension of the baseline LRC model, which is strongly motivated by the combination of the two extensions discussed above. Recall that the basic LRC model does not address the question of protecting the input encoding and output decoding steps, which are assumed to be leak-free. A more desirable goal is to obtain a *leakage-tolerant* circuit (LTC), where C maps an unprotected input x to an unprotected output $y = f(x)$ while providing the following best-possible security guarantee: the output of any leakage function $L \in \mathcal{L}$ can be *simulated* (up to a negligible statistical distance) by applying a similar leakage $L' \in \mathcal{L}$ to (x, y). Thus, C is "as secure" as an ideal (leak-free) hardware 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 x and the output y. More interestingly, the strong security guarantee of LTCs allows them to serve as a *general-purpose* replacement for ideal hardware with respect to leakage class $\mathcal{L}$. Jumping ahead, this may help bridge the gap between stateless and stateful LRCs for natural classes $\mathcal{L}$ of *global* leakage.

The starting point for the current work is the observation that we currently know virtually nothing about LTCs for general leakage classes. While LTCs are quite easy to realize in the baseline case of probing leakage [IKL+13,AIS18], this goal seems much more challenging even for simple classes of global leakage. Indeed, no such results are currently known. We thus ask:

Is leakage tolerance possible for any nontrivial class of global leakage?

1.1 Our Results

We initiate a systematic study of leakage-tolerant circuits for natural classes $\mathcal{L}$ of global leakage functions. Concretely, we focus on the case of *depth-1* leakage, which consists of two distinct classes $\mathcal{L}$ that strictly generalize t-probing:

– Depth-1 $\mathcal{AC}0$ leakage. This includes all leakage functions L that output t *disjunctions* (alternatively, conjunctions) of any number of wires or their negations. This simple kind of leakage is motivated by *selective failure* attacks (such as buffer overflows), where a system fails if at least one of several components fail. Existing LRCs for this class that do not rely on trusted hardware either incur a high asymptotic overhead, increasing the circuit size by a factor of $\Omega(t^2)$ [Rot12,BIS21] (in fact, this is known even for higher-depth $\mathcal{AC}0$ leakage), or alternatively are restricted to the stateless case [BDF+22].
– Parity leakage. Here L may output t *parities* of any number of wires. In theoretical computer science, parities often serve as the natural next step beyond probing: for instance, whereas a t-wise independent PRG fools t-probing distinguishers, a small-bias PRG [NN90] fools a parity distinguisher. From a more applied perspective, parity leakage can be viewed as going "half way" towards more realistic classes of global leakage, such as leaking the Hamming weight of subsets of wires. Finally, from a technical perspective this case seems challenging, because the standard LRC construction of [ISW03] completely breaks down in the presence of parity leakage. Nevertheless, efficient LRCs for this class (in fact, for the broader class of *bounded-communication leakage*) were obtained in [GIM+16,GIW17]. The question of obtaining *stateful* LRCs against parities was left open.

We note that the question of obtaining LTCs for these classes is nontrivial even for simple functions f, such as the mod-2 inner-product of two input vectors.

Feasibility of LTCs. Our main technical contribution is a general construction of LTCs for depth-1 leakage, establishing the first feasibility result for leakage-tolerance with respect to nontrivial global leakage classes. Concretely, we show that every circuit C_f computing a function f can be efficiently compiled into an LTC C for f in which any t disjunctions/conjunctions/parities of the wires of C or their negations can be simulated, with negligible statistical error, by making t queries of the same kind to the inputs and outputs of C.

A limitation of our parity-tolerant circuit compiler is that the associated *simulator* runs in time $2^{O(t)}$. We give a (weak) partial evidence for the necessity of inefficient simulation, showing that a related leakage-tolerant encoding scheme requires inefficient simulation assuming the hardness of the learning parity with noise (LPN) problem.

Application to stateful LRCs. Finally, we observe that LTCs can be used to bridge the gap between stateless and stateful LRCs. Extending a technique of Faust et al. [FRR+14], we show how to obtain a stateful LRC for $\mathcal{L}$ by combining two ingredients: (1) an LTC for $\mathcal{L}$, and (2) a leakage-resilient encoding for $\mathcal{L}$ that resists two *adaptive* leakage queries. Using this general transformation, we apply our LTC constructions to obtain the first construction of *stateful* t-leakage-resilient circuits that resist a *continuous* parity leakage, and the first such construction for disjunction/conjunction leakage in which the circuit size grows sub-quadratically with t. Somewhat surprisingly, we show how to ensure that the stateful LRC has an efficient simulator even if the underlying LTC does not. In particular, our stateful LRC for parities has a $\mathrm{poly}(t)$-time simulator.

Future directions. Several natural questions are left open for future work.

– Can we narrow the gap between LRCs and LTCs by obtaining LTC constructions for other classes? While our construction for parity leakage is closely related to the LRC against bounded-communication leakage from [GIM+16] (see Section 2 for an overview), we are unable to extend our results to this class, or even to the subclass of leakage functions that output the Hamming weight of a subset of wires. In particular, our LTC for the parity case crucially relies on a technique for reducing any parity query to a small number of probing queries, which we are not able to extend to the case of Hamming weight leakage. The LTC question is also still open for depth-2 $\mathcal{AC}0$ (capturing polynomial-size CNF/DNF formulas) and other classes for which LRCs are known to exist (see Section 1.2).
– Is there a t-parity-tolerant circuit with a $\mathrm{poly}(t)$-time simulator? Our simulator needs to find a short vector in a linear code defined by the parity queries, and we are only able to show that this is inherent for a related encoding problem. Note, however, that we can get around this limitation in the context of the application to stateful LRCs.

1.2 Related Work

We survey here some related works on leakage-resilient circuits. While in this work we focus on information-theoretic security, we are not aware of computationally secure constructions that achieve our goals. See [DLZ15] and references therein for constructions with computational security.

Global leakage. Simple classes $\mathcal{L}$ of global leakage considered in the literature include AC^0 leakage [FRR+14,Rot12,BIS21,BDF+22], "only computation leaks" leakage [MR04,DF12,GR15] and its extension to bounded-communication leakage [GIM+16,GIW17], and even NC^1 leakage [MV13]. Some of these constructions require trusted hardware components in the stateful case or input encoder which is as big as the circuit for f in the stateless case [FRR+14,DF12,MV13] and some rely on unproven conjectures [Rot12,MV13,BDF+22]. Another class of global leakage that was previously studied is captured by the *noisy leakage* model [PR13], where a small amount of information about every wire is independently leaked. However, the independence assumption makes this type of global leakage reducible to local leakage [DDF19].

Composable probing resilience. The notion of LTC provides general-purpose composition guarantees. However, more refined notions of composable security can suffice in specific application scenarios. Such notions for the relatively simple case of *probing* leakage have been considered in several previous works (see [BGR18,GPRV21,CGLS21,BCRT23] and references therein), with the *primary* goal of improving the practical efficiency of t-probing resilient circuits for small values of t.

2 Technical Overview

This work initiates a systematic study of *leakage-tolerant circuits* (LTCs). We start by describing the application of LTCs to building *stateful* leakage-resilient circuits (LRCs) that achieve security against *continuous leakage*. This application serves as a primary motivation behind this work.

2.1 Application: Stateful Leakage-Resilient Circuits

The notion of a *stateful* LRC aims to protect a general-purpose computing device, which can be invoked any number of times and possibly update the contents of its memory in each invocation. What makes this problem challenging is that the total amount of information leaked across multiple invocations is unbounded, and in particular is higher than the entropy of the secret state used to initialize this. This explains the fact that stateful LRCs are often much more challenging to construct than stateless ones.

We consider here the model of stateful LRCs that was introduced by Ishai et al. [ISW03] for the special case of "probing" leakage and later extended by Faust et al. [FRR+14] to general leakage classes. In this model, the ideal computation is defined by a stateful circuit $C[\boldsymbol{s}]$ that starts with a secret initial state $\boldsymbol{s} = \boldsymbol{s}_0$. In the i-th invocation, given its current secret state $\boldsymbol{s_{i-1}}$ and an external public input $\boldsymbol{x_i}$, the circuit C updates its secret state to $\boldsymbol{s_i}$ and produces an external public output $\boldsymbol{y}_i$, where $(\boldsymbol{s_i}, \boldsymbol{y_i})$ are a function of $(\boldsymbol{s_{i-1}}, \boldsymbol{x_i})$. 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 in different invocations remains the same.

For an ideal stateful circuit $C[\boldsymbol{s}]$ as above, a leakage-resilient implementation 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 $L_i \in \mathcal{L}$ to the circuit wires for each invocation, should learn essentially nothing about the secret state (in any invocation) of the stateful circuit $C[\boldsymbol{s}_0]$. In other words, the view of such an adversary should be simulatable given the public inputs $\boldsymbol{x_i}$ and public outputs $\boldsymbol{y_i}$ alone. Note that we do not protect external inputs and outputs and in fact the adversary may arbitrarily choose the external inputs in each invocation of $C'[\boldsymbol{s}'_0]$.

The simpler notion of a *stateless* LRC considers a one-shot computation of a function f on a secret input $\boldsymbol{x}$. To avoid a direct leakage on the input $\boldsymbol{x}$ or output $\boldsymbol{y} = f(\boldsymbol{x})$, the inputs and outputs are protected by an input encoder and an output decoder, respectively. Concretely, a stateless leakage-resilient circuit 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 wires in I or O, and require that the output of L reveals essentially nothing about the private input.

Note that LTCs are in a sense strictly stronger than (stateless) LRCs. Indeed, we can use LTCs to construct LRCs by having the input encoder I encode the input $\boldsymbol{x}$ using a leakage-resilient encoding (which is typically easy to construct), then use an LTC C to map the encoded input to a leakage-resilient encoding of the output, and finally let the output decoder O apply the decoding function of the leakage-resilient encoding. It turns out, however, that the stronger leakage-tolerance property plays a crucial role in the context of composition. One primary example is the construction of *stateful* LRCs. We start by discussing the challenges in converting stateless LRCs into stateful ones, and then explain how LTCs give rise to a conceptually simple transformation.

Transforming Stateless LRCs into Stateful LRCs? In [ISW03], the authors consider *t-probing attacks*, where the leakage class $\mathcal{L}$ includes all t-projection functions that output t fixed input bits. To construct a stateful LRC, they first augment a stateless LRC (I, C, O) to support additional public input and output, and then simply concatenate such LRCs together. In more detail: (1) The initial state is first encoded under the input encoder I. (2) Then in each invocation, an LRC C takes the encoded input and a public input, and computes the encoded secret output and the public output. (3) The encoded output corresponds to an encoding of the current state, which is used as the encoded input in the next invocation.[5]

It is important to highlight the fact that the security of the above transformation relies on specific properties of the underlying stateless LRC and does not work in general. To illustrate this, consider a stateful LRC that has no public input or output and simply keeps its state unchanged. That is, the ideal stateful circuit C simply computes the identity function on the state. A trivial stateless LRC for the identity function can simply output the encoded input. However, applying the ad-hoc transformation from [ISW03] does *not* yield a secure stateful LRC since in each invocation the attacker may probe t wires of the same encoding and eventually learn the whole encoding, which reveals the secret state.

This problem is inherent when using stateless LRCs because there is no separation between the randomness used in the input encoding and the randomness used in the output encoding. When using a general stateless LRC to construct a stateful LRC, the same randomness may be under attack in different invocations and eventually revealed to the attacker. In this case, the encoding no longer gives any protection to the secret state. While there are several formulations of sufficient "separation" requirements for the simple case of probing leakage [CPRR13,BBD+16,GPRV21], it is not clear how to meaningfully extend them to more powerful leakage classes. Indeed, most previous works on stateful LRCs for global leakage classes could only address this problem by assuming an ideal trusted hardware that helps perform "refreshing" operations to guarantee the required separation [FRR+14,DF12,MV13,BDF+22], or alternatively settling for computational security and relying on indistinguishability obfuscation [DLZ15].

From LTCs to Stateful LRCs. We now describe a general approach for obtaining stateful LRCs via a black-box combination of LTCs and a suitable type of leakage-resilient encoding, extending a previous approach from [FRR+14] for constructing stateful LRCs using small trusted hardware components.

Suppose we are given a randomized encoding function $\mathtt{Enc}$ which is secure against *two adaptive leakage queries* in the following sense: for all pairs of messages m_0, m_1, any (computationally unbounded) adversary who can adaptively choose $L_0, L_1 \in \mathcal{L}$ and learn $L_0(\mathtt{Enc}(m_b)), L_1(\mathtt{Enc}(m_b))$, where b is a random bit, cannot guess b with non-negligible advantage. Given such an encoding, the construction starts by letting $\boldsymbol{s}'_0 = \mathtt{Enc}(\boldsymbol{s}_0)$. Then, in the i-th invocation, we use an LTC $\tilde{C}'$ to decode the internal state $\boldsymbol{s}_{i-1}$ from $\boldsymbol{s}'_{i-1}$, invoke

[5] One can use an encoded output as an encoded input since I and O are compatible, in the sense that inputs and outputs are encoded in the same way.

the ideal stateful circuit $C[\boldsymbol{s}_{i-1}]$, and encode the new state $\boldsymbol{s}_i$ using a fresh invocation of $\mathtt{Enc}$. Relying on the leakage tolerance of $\tilde{C}'$, the leakage queries made by the adversary to the wires in $\tilde{C}'$ can be reduced to leakage queries to the encoded states. Since the states are protected by *fresh* leakage-resilient encodings, the leakage queries to the encoded states can be simulated without knowing the states. This concludes the stateful LRC construction.

Note that we need $\mathtt{Enc}$ to provide security against two adaptive leakage queries since an encoded state $\boldsymbol{s}'_i$ is under attack in both the $(i-1)$-th invocation and the i-th invocation, and moreover the leakage query made in the i-th invocation can depend on the result of the leakage query made in the $(i-1)$-th invocation. We summarize this transformation by the following informal theorem; see Appendix A for the formal version.

Theorem 1 (Stateful LRC via LTC, Informal). *Let $\mathcal{L}$ be a leakage class. Suppose there is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding, and an efficient compiler transforming any (stateless) circuit C to an equivalent $\mathcal{L}$-LTC C'. Then, there is an efficient compiler transforming any stateful circuit $C[\boldsymbol{s}_0]$ to an equivalent stateful $\mathcal{L}$-LRC $C'[\boldsymbol{s}'_0]$.*

Combining the above theorem with the LTC constructions we describe next, together with simple leakage-resilient encoding schemes, we obtain the first feasibility results for stateful LRCs against parity leakage, as well as the first *efficient* constructions of stateful LRCs against depth-1 $\mathcal{AC}0$ leakage.

On Efficient Simulation. We note that the simulation efficiency of the stateful LRC construction described above is inherited from the underlying LTC $\tilde{C}'$. If $\tilde{C}'$ requires inefficient simulation (in the output length of the leakage query), then the LRC $C'[\boldsymbol{s}'_0]$ also requires inefficient simulation. This will be the case when using our construction for t-parity-tolerant circuits to build a stateful t-parity-resilient circuit. In Appendix H we provide evidence that the inefficient simulation of our t-parity-tolerant circuits may be inherent; see Section 2.4 for an overview.

Fortunately, in the context of the stateful LRC application, we can still obtain efficient simulation regardless of the efficiency of the LTC simulator. At a high level, our idea is to include a control bit b in the initial state. We modify the functionality computed in each invocation by taking an auxiliary input: If $b = 0$, then we ignore the auxiliary input and compute the output by using the internal state and the public input as the original stateful circuit. Otherwise, we keep the state unchanged and output the decoding of the auxiliary input. To achieve the same functionality as the original stateful circuit, the control bit is set to be 0 by default and we just use random bits as the auxiliary input. Now the construction of the $\mathcal{L}$-leakage-resilient stateful circuit is applying the above compiler to the modified stateful circuit.

To obtain efficient simulation, the simulator will set the initial state to some default value and the control bit to 1. In each invocation, after learning the public input and output, the simulator will replace the auxiliary input by an encoding of the public output. In this way, the simulator can simply evaluate the $\mathcal{L}$-leakage-resilient stateful circuit with a fake initial state and compute the leakage queried by the adversary. This can be viewed as an information-theoretic analogue of the technique from [FLS99] for converting witness-indistinguishability to zero knowledge. See Appendix A.3 for a formal treatment.

2.2 Overview of Feasibility Results

Motivated in part by the application discussed above, our main technical contribution is establishing the feasibility of LTCs for simple classes of *global* leakage. We focus on the following two "depth-1" leakage classes:

- The first contains all depth-1 $\mathcal{AC}0$ functions. That is, every leakage function L has t output bits, each obtained by choosing a subset S of its inputs and computing the $\mathtt{OR}$ (alternatively $\mathtt{AND}$) of bits in S or their negations.
- The second leakage class contains all parity functions. That is, each of the t output bits of L is obtained by choosing a subset S of input bits and computing their $\mathtt{XOR}$.

We will give feasibility results of these two leakage classes separately in the following two subsections.

But to illustrate the non-triviality of the question, we start by showing a counter-intuitive result for the $\mathtt{OR}$ leakage class, where L outputs the $\mathtt{OR}$ of a subset of input bits without negating them.

Negative Result for the ISW Construction. The standard ISW construction [ISW03] implements an LRC against t-probing leakage (i.e., the adversary can learn any t chosen wires) using the following natural approach. Given a circuit C that computes the function $f : \{0,1\}^n \to \{0,1\}^m$, each wire w in C is transformed to a bundle of wires w_i whose parity is equal to value of w. One may interpret this as generating an *additive secret sharing* for each wire value. This ensures that learning t wires gives no information about the wire values in C.

It turns out that the same construction achieves leakage resilience against $\mathcal{AC}0$ leakage when setting t to be a security parameter κ [BIS21]. This intuitively follows from the well-known fact that $\mathcal{AC}0$ cannot compute parities (though the actual proof is much more subtle). Given this result, it is natural to conjecture that the κ-probing-tolerant variant of the ISW construction also achieves leakage *tolerance* against the $\mathtt{OR}$ leakage class, which is much weaker than the $\mathcal{AC}0$ leakage class. Surprisingly, we show that this is not the case even for a linear function f.

Counterexample. Consider the function $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. The probing-tolerant variant of the ISW construction for f works as follows.

1. Input Encoding Phase: For all $i \in \{1, 2, 3\}$, do the following.
   - Prepare κ random bits $x_{i,0}, \ldots, x_{i,\kappa-1}$.
   - Compute $x_{i,\kappa} = x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,\kappa-1}$ in order.
   - Set $[x_i] = (x_{i,0}, \ldots, x_{i,\kappa})$. Note that the parity of bits in $[x_i]$ is x_i.
2. Circuit Emulation Phase: Compute $[y] = [x_1] \oplus [x_2] \oplus [x_3]$ coordinate by coordinate in order.
3. Output Decoding Phase: Set $(y_0, \ldots, y_\kappa) = [y]$. Compute $y = y_\kappa \oplus y_{\kappa-1} \oplus \ldots \oplus y_0$ in order.

One can show that the above construction achieves κ-probing tolerance for f.

Consider the following attack: $(x_1 \oplus x_{1,0}) \vee (x_2 \oplus x_{2,0}) \vee (x_{1,0} \oplus x_{2,0})$, where $x_1 \oplus x_{1,0}, x_2 \oplus x_{2,0}$ are the inner wires in the input encoding phase when computing $x_{1,\kappa}$ and $x_{2,\kappa}$, and $x_{1,0} \oplus x_{2,0}$ is the inner wire in the circuit emulation phase when computing $[y]$. Using Boolean algebra identities, we have

$$(x_1 \oplus x_{1,0}) \vee (x_2 \oplus x_{2,0}) \vee (x_{1,0} \oplus x_{2,0}) = (x_1 \oplus x_{1,0}) \vee (x_2 \oplus x_{2,0}) \vee (x_1 \oplus x_2).$$

Let $r_1 = x_1 \oplus x_{1,0}$ and $r_2 = x_2 \oplus x_{2,0}$. Then the above attack reveals $r_1 \vee r_2 \vee (x_1 \oplus x_2)$, which is equal to 1 with probability $3/4$ and leaks $x_1 \oplus x_2$ with probability $1/4$. If the construction were $\mathtt{OR}$-tolerant, then such an attack could be simulated by computing the $\mathtt{OR}$ of a subset of $x_1, x_2, x_3, y = x_1 \oplus x_2 \oplus x_3$. However, intuitively, the $\mathtt{OR}$ operation is not able to compute $x_1 \oplus x_2$ from the above, which leads to a contradiction. See Appendix B.4 for the details.

2.3 Leakage Tolerance Against Depth-1 $\mathcal{AC}0$ Leakage

It turns out that if we extend the $\mathtt{OR}$ leakage class to depth-1 $\mathcal{AC}0$, by allowing both the real and the ideal leakage to apply negations, then the ISW construction becomes leakage-tolerant.

The reason for this unexpected phenomenon is that when we switch to a richer leakage class, we strengthen not only the "real" leakage L that applies to all wires, but also the "ideal" leakage L' that applies only to the input and output bits. Thus, for two leakage classes $\mathcal{L} \subset \hat{\mathcal{L}}$, an $\hat{\mathcal{L}}$-LTC is not necessarily an $\mathcal{L}$-LTC.

Single Bit of Depth-1 $\mathcal{AC}0$ Leakage. Consider a linear function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ where each y_j is a linear combination of $\{x_i\}_{i=1}^n$. Let κ be the security parameter. Recall the κ-probing-tolerant variant of the ISW construction for the linear function f:

1. Input Encoding Phase: For all $i \in \{1, \ldots, n\}$, do the following.
   - Prepare κ random bits $x_{i,0}, \ldots, x_{i,\kappa-1}$.
   - Compute $x_{i,\kappa} = x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,\kappa-1}$ in order.
   - Set $[x_i] = (x_{i,0}, \ldots, x_{i,\kappa})$.
2. Circuit Emulation Phase: The circuit computes $[y_j]$ via a proper linear combination of $[x_1], \ldots, [x_n]$.
3. Output Decoding Phase: For all $j \in \{1, \ldots, m\}$, set $(y_{j,0}, \ldots, y_{j,\kappa}) = [y]$ and compute $y_j = y_{j,\kappa} \oplus y_{j,\kappa-1} \oplus \ldots \oplus y_{j,0}$ in order.

For all $j \in \{0, \ldots, \kappa\}$, let $\mathcal{I}_j$ be the set of wires that are non-zero linear combinations of $\{x_{i,j}\}_{i=1}^{n}$ in the circuit emulation phase. We note that if we arbitrarily pick a variable $w_j \in \mathcal{I}_j$ for all $j \in \{0, \ldots, \kappa\}$, then any κ variables in $\{w_j \in \mathcal{I}_j\}_{j=0}^{\kappa}$ are uniformly random. This is because for all $S \subset \{0, \ldots, \kappa\}$ with $|S| = \kappa$, $\{x_{i,j}\}_{j \in S, i \in \{1,\ldots,n\}}$ are uniformly random bits. Then for all $j \in S$, each w_j is uniformly random given $\{x_{i,j'}\}_{j' \in S, j' \neq j, i \in \{1,\ldots,n\}}$.

Now consider a single bit of depth-1 $\mathcal{AC}0$ leakage, obtained as follows.

1. An adversary first chooses an arbitrary set W of wires in C.
2. Then the adversary may arbitrarily flip some wires in W. Denote the resulting set of literals (wire variables or their negations) by $\widetilde{W}$.
3. Finally, the leakage is defined to be $\mathtt{OR}(\widetilde{W})$, where $\mathtt{OR}(S)$ denotes the result of $\mathtt{OR}$ of all bits in S.

Our main observation is that, if for all $j \in \{0, \ldots, \kappa\}$, $W \cap \mathcal{I}_j \neq \emptyset$, then there are at least κ variables in $\widetilde{W}$ that are uniformly random. In this case $\mathtt{OR}(\widetilde{W}) = 1$ with probability at least $1 - 2^{-\kappa}$. Thus, we may simply output 0 as the simulation of $\mathtt{OR}(\widetilde{W})$.

It is therefore sufficient to focus on the case where there is $j^\star \in \{0, \ldots, \kappa\}$ such that $W \cap \mathcal{I}_{j^\star} = \emptyset$. We observe that in this case, every wire in W only depends on a single input or output bit. In this way, $\mathtt{OR}(\widetilde{W})$ naturally corresponds to a depth-1 $\mathcal{AC}0$ leakage on the input and output bits. At a high level, this can be achieved by the following two steps.

1. Simulate all wire values in $\bigcup_{j \neq j^\star} \mathcal{I}_j$. We note that this can be done without knowing the input and output bits.
2. Given wire values in $\bigcup_{j \neq j^\star} \mathcal{I}_j$, every intermediate wire in the input encoding phase and the output decoding phase only depends on a single input or output bit.

In Appendix B.2, we show how to extend the above idea to handle multiple bits of depth-1 $\mathcal{AC}0$ leakage.

Towards General Functions. Based on the positive result of the ISW construction for linear functions against depth-1 $\mathcal{AC}0$ leakage, we show how to construct leakage-tolerant circuits for general functions.

Let $f : \{0,1\}^n \to \{0,1\}^m$ be an arbitrary binary function with circuit implementation C. At a high level, for each wire w in C, we will compute an additive secret sharing of w. Then we will use the ISW construction to compute every $\mathtt{XOR}$ gate and every $\mathtt{AND}$ gate.

1. For each input bit x_i, we sample κ random bits $(x_{i,0}, \ldots, x_{i,\kappa-1})$. To obtain $[x_i]$, we want to compute $x_{i,\kappa} = x_i \oplus (\bigoplus_{\ell=0}^{\kappa-1} x_{i,\ell})$. We view the computation of $x_{i,\kappa}$ as a linear function and apply the ISW construction to it.
2. For each $\mathtt{XOR}$ gate with two input additive sharings $[a], [b]$, we sample random bits $c_0, \ldots, c_{\kappa-1}$. To obtain $[c] = [a + b]$, we want to compute $c_\kappa = (\bigoplus_{\ell=0}^{\kappa} a_\ell) \oplus (\bigoplus_{\ell=0}^{\kappa} b_\ell) \oplus (\bigoplus_{\ell=0}^{\kappa-1} c_\ell)$. We view the computation of c_κ as a linear function and apply the ISW construction to it.
3. For each $\mathtt{AND}$ gate with two input additive sharings $[a], [b]$, we sample random bits $c_0, \ldots, c_{\kappa-1}$. Then we compute $a_{\ell_1} \cdot b_{\ell_2}$ for all $\ell_1, \ell_2 \in \{0, \ldots, \kappa\}$. Our goal is to compute $c_\kappa = (\bigoplus_{\ell_1,\ell_2=0}^{\kappa} a_{\ell_1} \cdot b_{\ell_2}) \oplus (\bigoplus_{\ell=0}^{\kappa-1} c_\ell)$. We view the computation of c_κ as a linear function and apply the ISW construction to it.
4. After obtaining an additive secret sharing $[y_j]$ for each output y_j, we want to reconstruct $y_j = \bigoplus_{\ell=0}^{\kappa} y_{j,\ell}$. We view the computation of y_j as a linear function and apply the ISW construction to it.

Intuitively, by using the ISW construction for every linear function, it is sufficient to focus on an adversary which only has access to the input and output wires of the linear functions. These wires can be summarized as follows:

- The input and output bits of f: $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$
- A random additive secret sharing for every wire w: $[w] = (w_0, w_1, \ldots, w_\kappa)$.
- For each $\mathtt{AND}$ gate with two input additive sharings $[a], [b]$, the cross multiplications between shares: $\{a_{\ell_1} \cdot b_{\ell_2}\}_{\ell_1=0,\ell_2=0}^{\kappa}$.

Note that 1 bit of depth-1 $\mathcal{AC}0$ leakage on these wires can be viewed as the OR of two separated bits of depth-1 $\mathcal{AC}0$ leakage: the first part is depth-1 $\mathcal{AC}0$ leakage on the input and output bits, and the second part is depth-1 $\mathcal{AC}0$ leakage on the random additive secret sharings and the cross multiplications between shares. We note that the second part can be further viewed as a depth-2 $\mathcal{AC}0$ leakage on the random additive secret sharings. Since we know that an $\mathcal{AC}0$ leakage cannot distinguish a random additive secret sharing of 0 from a random additive secret sharing of 1, we may just replace random additive secret sharings for wire values in C by random additive secret sharings of 0 and compute the cross multiplications between shares accordingly. In this way, the second part can be computed directly while the first part is just a depth-1 $\mathcal{AC}0$ leakage on the input and output bits of the function. Thus, we have the following theorem.

Theorem 2 (LTC Against Depth-1 $\mathcal{AC}0$, Informal). *Let $\mathcal{L}$ denote the leakage class that contains all depth-1 $\mathcal{AC}0$ leakage functions. For all binary function $f : \{0,1\}^n \to \{0,1\}^m$ with circuit implementation C, there exists an $\mathcal{L}$-leakage-tolerant circuit for C.*

Using General Linear Secret Sharing Schemes. In Section 4, we show that our construction can be modified slightly to support any linear secret sharing scheme. In particular, we can use packed Shamir sharings to achieve a better circuit complexity for computing SIMD circuits. Relying on [DIK10], which transforms a general circuit to a SIMD circuit, we obtain the following corollary.

Corollary 1 (Informal). *Let $\mathcal{L}$ denote the leakage class that contains all depth-1 $\mathcal{AC}0$ leakage functions. For all binary function $f : \{0,1\}^n \to \{0,1\}^m$ with circuit implementation C of size s and depth h, there exists an $\mathcal{L}$-leakage-tolerant circuit for C with circuit size $\tilde{O}(\kappa s + \kappa^2 h + \kappa^3)$, where $\tilde{O}$ omits logarithmic factors and κ is the security parameter.*

2.4 Leakage Tolerance Against Parity Leakage

Unlike the depth-1 $\mathcal{AC}0$ leakage, one cannot hope the ISW construction to be parity-tolerant (i.e., leakage-tolerant against parity leakage) since we can easily recover an intermediate wire value by XOR. At a high level, our result is obtained in two steps.

Step 1. We first define a new notion which we refer to as *parity-to-probing* circuits. Very informally, given a circuit $\tilde{C}$ representing the function we want to compute, a (t,k)-parity-to-probing implementation of $\tilde{C}$ is a triple (I, C, O), where I is a randomized input encoder, C is a randomized circuit, and O is an output decoder, such that the following properties hold.

- For any input x, we have $O(C(I(x))) = \tilde{C}(x)$ (with probability 1).
- Any t parity queries to the wires of C can be simulated by k probing queries to the wires of $\tilde{C}$. We refer to this property as parity-to-probing security.
- I and O admit circuit implementations in which any inner wire is a linear combination of the input and output wires. We refer to this property as trivial parity tolerance. Note that this property can be satisfied even by nonlinear functions, such as $x_1 \cdot x_2$.

We observe that when $\tilde{C}$ is a k-probing-tolerant circuit, any t parity queries to the wires of C can be simulated by k probing queries to the inputs and outputs of $\tilde{C}$. We utilize this property and show that a t-parity-tolerant circuit can be constructed from a (t,k)-parity-to-probing circuit.

We remark that without the last property, a parity-to-probing circuit is strictly weaker than a parity-resilient circuit since the latter requires that any parity query to the wires in C should give essentially no information about the inputs and outputs of $\tilde{C}$. However, our construction relies on the property of trivial parity tolerance for the input encoder and the output decoder. On the other hand, we show that the input encoder and the output decoder of a parity-resilient circuit cannot be trivially parity-tolerant (see Claim 1).

Step 2. Then, we show how to construct a parity-to-probing circuit for any underlying circuit $\tilde{C}$. Our construction is inspired by the construction of [GIM+16]. We observe that the construction in [GIM+16] implies a (t,k)-parity-to-probing circuit with the aid of a secure NAND gadget. In this step, we construct a

circuit that realizes the secure NAND gadget and show that after replacing the NAND gadget by our construction in [GIM+16], the (t, k)-parity-to-probing security remains. Combining with the first step, we obtain a feasibility result for t-parity-tolerant circuits.

In the following we give a more detailed exposition of the above 2 steps.

Step 1: Parity-to-Probing Implies Parity Tolerance

Parity Tolerance vs. Parity Resilience. A natural attempt of constructing a parity-tolerant circuit is to use a parity-resilient circuit (I, C, O) and implement I and O by parity-tolerant circuits. However there are two issues with this attempt:

– *Constructing parity-tolerant circuits for I and O is not trivial.*

The only previous construction of parity-resilient circuits is from [GIM+16]. While the input encoder and the output decoder in this construction are not complicated, it is still not clear how to implement them in a parity-tolerant way. In fact, even constructing a parity-tolerant circuit for the mod-2 inner-product function $f(x_1, x_2, y_1, y_2) = x_1 \cdot y_1 + x_2 \cdot y_2$ is not an easy task. The naive implementation, which first computes $x_1 \cdot y_1, x_2 \cdot y_2$ and then computes their sum, is *not* parity-tolerant. This is because the inner wire $x_1 \cdot y_1$ cannot be simulated by one parity query to the input wires (x_1, x_2, y_1, y_2) and the output wire $x_1 \cdot y_1 + x_2 \cdot y_2$.

– *There is a quantitative loss when composing two parity-tolerant circuits.*

Even if we have constructed t-parity-tolerant circuits for I and O (meaning that any t parities of the wires can be simulated by t parities of the circuit inputs and outputs), the implementation of (I, C, O) may not achieve t-parity tolerance. Consider the following natural simulation strategy: For each parity query, we view it as the summation of three parity queries, the first one for the wires in I, the second one for the wires in C, and the last one for the wires in O. The t parity queries to the wires in C are not a problem, since (by parity resilience) they can be simulated independently of the function inputs and outputs. However, to simulate the t parity queries to the wires in I, we need to make t parity queries to the inputs and outputs of I, which translate to t parity queries to the function inputs and outputs. Similarly, to simulate the t parity queries to the wires in O, we need to make another t parity queries to the function inputs and outputs. These add up to $2t$ parity queries to the function inputs and outputs. In summary, to simulate t parity queries to the implementation of (I, C, O), we may have to make $2t$ parity queries to the function inputs and outputs.

Constructing Parity-Tolerant Circuits from Parity-to-Probing Circuits. Let f be a function. Our goal is to construct a parity-tolerant circuit implementation of f. We focus on a single parity query for simplicity.

Our first attempt is to start with a k-probing-tolerant implementation $\tilde{C}$ of f and then consider the $(1, k)$-parity-to-probing circuit (I, C, O) of $\tilde{C}$. Now consider the parity of a set W of wires of the circuit implementation of (I, C, O). Recall that for a parity-to-probing circuit, its input encoder and output decoder satisfies the trivial parity tolerance. Therefore, $\texttt{parity}(W)$ can be computed by some parity of (1) the function inputs and outputs, and (2) the wires in C. (Here and in the following, $\texttt{parity}(W)$ denotes the parity of all wires in W.) Now by the parity-to-probing security, the parity of wires in C can be simulated by k wires in $\tilde{C}$, which can be simulated by k inputs and outputs of $\tilde{C}$ due to the property of probing tolerance.

In summary, $\texttt{parity}(W)$ can be simulated by (1) a parity of the function inputs and outputs and (2) k wires of the function inputs and outputs. This requires us to not only make one parity query but also k probing queries.

To address this issue, for each bit x of the function inputs and outputs, we split it to $k + 1$ random additive shares with parity x. Then we apply the above idea to compute $\hat{f}$ which maps additive sharings of the inputs of f to additive sharings of the outputs of f. We note that

– The k probings of additive sharings of the function inputs and outputs can be simulated by choosing k random values.

– Given the k probings, any parity query to the additive sharings of the function inputs and outputs can be reduced to a parity query to the function inputs and outputs. Furthermore, the encoding and decoding processes of an additive sharing achieve parity tolerance for free, since every inner wire can be computed by a linear combination of the additive shares.

Thus, our construction of a parity-tolerant circuit $C^\star$ is as follows (See Figure 2 for the complete description):

1. The circuit $C^\star$ first encodes each bit of the input $\boldsymbol{x}$ of f by a random additive sharing. Let $\hat{\boldsymbol{x}}$ denotes the encoding of $\boldsymbol{x}$.
2. 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. Finally, the circuit $C^\star$ decodes $\hat{\boldsymbol{y}}$ by computing the parity of each additive sharing and obtains $\boldsymbol{y}$.

The above idea is naturally extended to constructing t-parity-tolerant circuits from (t, k)-parity-to-probing circuits and k-probing-tolerant circuits. We refer the readers to Section 5.1 for more details.

Step 2: Constructing Parity-to-Probing Circuits Now we turn our focus to parity-to-probing circuits. Our starting point is the construction in [GIM+16], which implicitly gives a (t, k)-parity-to-probing circuit with the aid of a secure `NAND` gadget. We first give an overview of the construction in [GIM+16].

Overview of [GIM+16]. Let $\tilde{C}$ denote the input circuit. The goal is to construct a (t, k)-parity-to-probing circuit implementation of $\tilde{C}$. Without loss of generality, we assume that $\tilde{C}$ only contains `NAND` gates. The idea in [GIM+16] is to encode each wire in $\tilde{C}$ by a small-bias encoding scheme. Very informally, an ϵ-bias encoding scheme ensures that the parity of any non-empty subset bits of the encoding has bias (i.e., difference from a uniform bit) at most ϵ. An example of a 1/2-bias encoding is presented in Figure 1. In the following, we will

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

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

Fig. 1: An Example of Small-bias Encoding Scheme

stick to the small-bias encoding scheme in Figure 1. The construction of $(\hat{I}, \hat{C}, \hat{O})$ is as follows:

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

We first verify that this construction achieves parity-to-probing security. Consider a parity query which chooses a set W of wires in $\hat{C}$. Intuitively, if W touches more than k different small-bias encodings in $\hat{C}$, then $\mathtt{parity}(W)$ would have bias at most 2^{-k}, indicating that $\mathtt{parity}(W)$ is statistically close to a uniform bit. On the other hand, if W touches at most k different small-bias encodings in $\hat{C}$, then $\mathtt{parity}(W)$ can be

perfectly simulated by probing the underlying k wires in C'. In Section 5.2, we show that the above analysis can be extended to t parity queries.

As for the trivial parity tolerance, one can easily verify that this property is satisfied in the following two circuit constructions for $\hat{I}$ and $\hat{O}$ respectively.

- For $\hat{I}$, which computes a 1/2-bias encoding of each bit x, the encoding is computed by first computing $r_0 \cdot r_1$, and then computing $(r_0 \cdot r_1) \oplus x$.
- For $\hat{O}$, which decodes a 1/2-bias encoding for each output bit, the decoding is computed by first computing $\hat{x}_0 \cdot \hat{x}_1$, and then computing $(\hat{x}_0 \cdot \hat{x}_1) \oplus \hat{x}_2$.

Instantiating `NAND` *Gadgets.* Our initial attempt is to utilize the instantiation in [GIM+16]. We note that the construction in [GIM+16] only focuses on parity-resilient circuits. Unfortunately, we do not know whether the (t, k)-parity-to-probing property remains after applying their instantiation for `NAND` gadgets. In Appendix D, we discuss their solution in details and show that:

- By using the parity-simulation lemma, the construction in [GIM+16] is a $(1, k)$-parity-to-probing circuit. Together with our main theorem in Step 1, we can compile their construction into a 1-parity-tolerant circuit. The parity-simulation lemma and its proof may be of independent interest.
- On the other hand, we show that the parity-simulation lemma cannot handle more than 1 bit of parity leakage. We also give a counter example which shows that in general, a 1-parity-tolerant circuit may not achieve 2-parity tolerance.

In the following, we discuss our solution that gives a (t, k)-parity-to-probing circuit. Our solution is specific to the use of the small-bias encoding presented in Figure 1. Let $(\hat{I}, \hat{C}, \hat{O})$ be the (t, k)-parity-to-probing circuit with the aid of a secure `NAND` gadget from [GIM+16]. Recall that $\hat{C}$ computes a small-bias encoding for every wire of the underlying circuit $\tilde{C}$. Suppose $\boldsymbol{w} = (w_1, \ldots, w_{|\tilde{C}|})$ are the wires in $\tilde{C}$ and $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^{|\tilde{C}|}$ are two vectors of random bits. Then, the wires in $\hat{C}$ 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, $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w})$.

Our main observation is that, the (t, k)-parity-to-probing property is preserved even if the adversary knows $\boldsymbol{v}$ and can do any computation on $\boldsymbol{v}$. To be more precise: For all subset W of wires in $\hat{C}$, the joint view of $(\texttt{parity}(W), \boldsymbol{v})$ can be simulated with statistical error negligible in k by probing k wires in $\tilde{C}$.

Intuitively, it follows the fact that $\texttt{parity}(W)$ is masked by $\oplus_{i \in I} u_i \cdot v_i$ for some set $\mathcal{I}$. When $|\mathcal{I}| \geq k$, as long as one of v_i is not 0, which happens with probability $1 - 2^{-k}$ the mask value is uniformly distributed. As long as $\boldsymbol{v}$ is sampled uniformly random and the adversary only learns $\boldsymbol{v}$ after choosing W, learning $\boldsymbol{v}$ does not help the adversary to learn any information about $\boldsymbol{w}$. This observation allows us to do any computation on $\boldsymbol{v}$ when instantiating `NAND` gadgets while preserving the parity-to-probing property.

Example: Using Our Observation to Instantiate `XOR` *Gadgets.* To explain our high level idea, we show how our observation allows us to instantiate `XOR` gadgets. Our instantiation for `NAND` gadgets follows a similar idea which is discussed later. Given two encodings $(u_1, v_1, z_1 = u_1 \cdot v_1 \oplus w_1), (u_2, v_2, z_2 = u_2 \cdot v_2 \oplus w_2)$, we want to compute an encoding $(u_0, v_0, z_0 = u_0 \cdot v_0 \oplus w_1 \oplus w_2)$. Since u_0, v_0 are random bits, we just need to compute $z_0 = u_0 \cdot v_0 \oplus u_1 \cdot v_1 \oplus u_2 \cdot v_2 \oplus z_1 \oplus z_2$. The main observation is that, given v_0, v_1, v_2, this equation becomes a linear combination of $(u_0, u_1, u_2, z_1, z_2)$, which can be implemented directly since the intermediate results can also be obtained under parity attack on $(u_0, u_1, u_2, z_1, z_2)$. Therefore, we first compute z_0 under all possible assignments of $(v_0, v_1, v_2) = (a_0, a_1, a_2)$. In the meantime, we also compute a bit indicating whether $(v_0, v_1, v_2) = (a_0, a_1, a_2)$. Note that the computation of indicating bits only depends on $\boldsymbol{v}$. As we argued above, these intermediate wires do not break the parity-to-probing security. Finally, we use the indicating bits to choose the proper z_0. In the actual construction, we will first prepare a random bit ρ and compute $z_0 + \rho$ using the above approach to protect the secrecy of $w_1 \oplus w_2$. The overall construction is as follows:

1. We first sample a random bit ρ. Let $f(v_0, v_1, v_2) = u_0 \cdot v_0 \oplus u_1 \cdot v_1 \oplus u_2 \cdot v_2 \oplus z_1 \oplus z_2 \oplus \rho$, which is just $z_0 \oplus \rho$.
2. For all $(a_0, a_1, a_2) \in \{0,1\}^3$, compute $f(a_0, a_1, a_2)$ by a linear combination of $(u_0, u_1, u_2, z_1, z_2, \rho)$.
3. For all $(a_0, a_1, a_2) \in \{0,1\}^3$, compute $\prod_{i=0}^{2}(v_i \oplus a_i \oplus 1)$, which indicates whether $(v_0, v_1, v_2) = (a_0, a_1, a_2)$.
4. Compute $z_0 \oplus \rho = \sum_{\boldsymbol{a} \in \{0,1\}^3} f(a_0, a_1, a_2) \cdot \prod_{i=0}^{2}(v_i \oplus a_i \oplus 1)$.
5. Compute $z_0 = (z_0 \oplus \rho) \oplus \rho$.

Note that the intermediate wires introduced in Step 2 are linear combination of the input and output bits. The intermediate wires introduced in Step 3 only depends on $\boldsymbol{v}$. As for the intermediate wires introduced in Step 4, let $\rho' = z_0 \oplus \rho$. We may sample a random bit as ρ' and compute $\rho = z_0 \oplus \rho'$. Then the intermediate wires introduced in Step 4 only depends on random bits $\boldsymbol{v}$ and ρ', which does not break the parity-to-probing security.

Moving to `NAND` *gadgets.* For `NAND` gadgets, what we need to compute becomes $z_0 = u_0 \cdot v_0 \oplus (u_1 \cdot v_1 \oplus z_1) \cdot (u_2 \cdot v_2 \oplus z_2)$. Even if given v_0, v_1, v_2, this equation is not a linear combination of $(u_0, u_1, u_2, z_1, z_2)$, which means that computing z_0 given (v_0, v_1, v_2) is no longer for free under parity attacks. To address this issue, our idea is to also fix (u_1, z_1) (or (u_2, z_2)) so that z_0 is a linear combination of (u_0, u_2, z_2) (or (u_0, u_1, z_1)). To be more concrete, we first refresh the encoding of w_1 (and w_2), i.e., computing $(u_1', v_1', z_1' = u_1' \cdot v_1' \oplus w_1)$ from (u_1, v_1, w_1). This can be viewed as a `Refresh` gadget and can be instantiated in a similar way to `XOR` gadgets. In this way, when analyzing the intermediate wires related to (u_2, z_2), we fix the refreshed encoding (u_1', v_1', z_1') and now the computation of z_0 is linear as that in `XOR` gadgets. We refer readers to Section 5.2 for more details.

In Appendix G, we show that our construction maintains the (t, k)-parity-to-probing security of $(\hat{I}, \hat{C}, \hat{O})$. Therefore, combining our result in the first step, we obtain a t-parity-tolerant circuit for any function f.

On the Need for Inefficient Simulation. A limitation of our construction of (t, k)-parity-to-probing circuits is that the running time of the simulation is exponential in t. This implies that our construction of t-parity tolerant circuits also has exponential-time simulation. We demonstrate this problem by taking the parity-to-probing circuit $(\hat{I}, \hat{C}, \hat{O})$ constructed in [GIM+16] as an example. To simplify the description, for two sets W_1, W_2, we define $W_1 \oplus W_2 = W_1 \cup W_2 \backslash (W_1 \cap W_2)$. Then $\texttt{parity}(W_1) \oplus \texttt{parity}(W_2) = \texttt{parity}(W_1 \oplus W_2)$.

Recall that for a set W of wires in $\hat{C}$, if W touches more than k small-bias encodings, then $\texttt{parity}(W)$ can be simulated by a random bit. Otherwise, we need to probes the set of wires of the underlying circuit $\hat{C}$ whose encodings are touched by W to simulate $\texttt{parity}(W)$. We denote this set by $\mathcal{V}(W)$. Given W, we can find $\mathcal{V}(W)$ efficiently.

When we have t sets $W_1, \ldots, W_t$, only finding the sets $\mathcal{V}(W_1), \ldots, \mathcal{V}(W_t)$ is not sufficient since we have to consider the joint distribution of all parities. For example, if W_1 and W_2 both touches more than k encodings, then $\mathcal{V}(W_1) = \mathcal{V}(W_2) = \emptyset$. On the other hand $W_1 \oplus W_2$ may only touch less than k encodings. In this case, we have to find the set $\mathcal{V}(W_1 \oplus W_2)$ to simulate $\texttt{parity}(W_1) \oplus \texttt{parity}(W_2)$. In general, to simulate t parities, we have to compute the set $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(\oplus_{i \in S} W_i)$ which takes time exponential in t. (A remark about the size of V: Although it is a union set of 2^t subsets, we show that the size of V is bounded by kt. See more details in Section 5.2 and the proof of Theorem 4 in Appendix G.)

In Appendix H, we consider a relaxed notion of t-parity tolerance which we refer to as (t, t')-parity-tolerant functions. Concretely, a (t, t')-parity-tolerant function requires that any t parities of the output of the function can be simulated by t' parities of the input of the function. Intuitively, one may view a parity-tolerant circuit as a special kind of parity-tolerant function where the function's input is the circuit input and output, and the function's output is the wires of the circuit.

We give an example of a (t, t')-parity-tolerant function that requires super-polynomial simulation time in t under the standard LPN assumption. This may serve as partial evidence that inefficient simulation may be inherent for parity tolerance as well.

3 Preliminaries

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 note that all natural leakage classes (or their simple variants[6]) considered in the literature satisfy this property. We borrow the definition of leakage-resilient circuits from [GIM+16] as follows.

Definition 1 ([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 distributions $L(\tau(C, I(\boldsymbol{x})))$ and $L(\tau(C, I(\boldsymbol{x}')))$ is at most ϵ, where $\tau(C, \hat{\boldsymbol{x}})$ denotes the wire values of C when taking $\hat{\boldsymbol{x}}$ as input.*

For leakage-tolerant circuits for general leakage classes, we generalize the notion of t-probing-tolerant circuits from [ISW03,AIS18,GIS22].

Definition 2 (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 $\boldsymbol{x}$.

In this work, we are interested in three different leakage classes.

[6] For example, for parity leakage, we may include functions that compute the flip of the parity of a subset of input bits. For probing leakage, we may include the constant functions that always output 0 or 1. These expansions of the leakage classes do not bring more power to the adversary.

- Probing Leakage Class: This leakage class contains all leakage functions $L : \{0,1\}^n \to \{0,1\}$ defined by $i \in \{1,\ldots,n\}$ such that $L(\boldsymbol{x}) = x_i$.
- Depth-1 $\mathcal{AC}0$ Leakage Class (D1 Leakage Class for short): This 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 $L(\boldsymbol{x}) = (\bigvee_{i\in S_0} x_i) \vee (\bigvee_{i\in S_1}(x_i \oplus 1))$.
- Parity Leakage Class: This leakage class contains all leakage functions $L : \{0,1\}^n \to \{0,1\}$ defined by $S \subset \{1,\ldots,n\}$ such that $L(\boldsymbol{x}) = \bigoplus_{i\in S} x_i$.

We say $\mathcal{L}$ is the t-leakage class of $\mathcal{L}'$ (denoted by $(\mathcal{L}')^{\otimes t}$) 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 $L(\boldsymbol{x}) = (L'_1(\boldsymbol{x}),\ldots,L'_t(\boldsymbol{x}))$. Thus, the above three leakage classes can be naturally extended to their t-leakage versions: t-probing leakage class, t-D1 leakage class, and t-parity leakage class.

In the following, we use (t,ϵ)-D1 tolerance to denote leakage tolerance against t-D1 leakage with error ϵ. Similarly, we use (t,ϵ)-parity tolerance to denote leakage tolerance against t-parity leakage with error ϵ. Since there are known results [ISW03,AIS18] that can achieve leakage tolerance against t-probing leakage without error, we simply write t-probing tolerance for probing leakage.

4 Sketch of Depth-1 $\mathcal{AC}0$ Leakage Tolerance

In this section, we give a sketch of our construction for leakage tolerant circuits against depth-1 $\mathcal{AC}0$ leakage (D1 leakage for short). We follow the overview in Section 2.3 and show that the ISW construction [ISW03] for linear functions achieves D1 tolerance. Then D1-tolerant circuits for general functions can be easily obtained following the compiler in Section 2.3. We give the formal description in Appendix B.

Recall that in Section 2.3, we show that the ISW construction for linear functions achieve 1-D1 tolerance. The high-level idea is to separate the wires in the computation phase to $\kappa+1$ disjoint sets $\mathcal{I}_0,\ldots,\mathcal{I}_\kappa$ such that if we arbitrarily pick a variable $w_j \in \mathcal{I}_j$ for all $j \in \{0,\ldots,\kappa\}$, then any κ variables in $\{w_j \in \mathcal{I}_j\}_{j=0}^{\kappa}$ are uniformly random. Based on this fact, if the leakage query touches wires in all $\kappa+1$ sets, then the output is 1 with overwhelming probability. Otherwise, say $\mathcal{I}_{j^\star}$ is not touched. Then every wire in C except those in $\mathcal{I}_{j^\star}$ only depends on a single input or output bit. So the D1-leakage query to the wires in the circuit is also a D1-leakage query to the input and output bits.

When considering t-D1 leakage, we use a $(t\kappa+1)$-probing tolerant construction. We may similarly separate the wires in the computation phase to $\{\mathcal{I}_j\}_{j=0}^{t\kappa}$. The observation is that for each D1-leakage query, if it touches wires in more than κ sets, then the output of this leakage bit is 1 with overwhelming probability. By a standard hybrid argument, it is sufficient to only focus on leakage queries that touch at most κ sets. Then at most $t\kappa$ sets are touched by all D1-leakage queries, and there is one set, say $\mathcal{I}_{j^\star}$, which is not touched by and D1-leakage query. Since every wire in C except those in $\mathcal{I}_{j^\star}$ only depends on a single input or output bit, the t D1-leakage queries to the wires in the circuit become t D1-leakage queries to the input and output bits.

Supporting General Linear Secret Sharings. We note that our construction for general functions can be modified slightly to support any linear secret sharing scheme that is $t\kappa$-wise independent. On one hand, it is known from [Bra11] that a $t\kappa$-wise independent secret sharing scheme is t-D1-resilient. On the other hand, reconstructing secrets and computing a fresh random secret sharing can be done by linear circuits for a linear secret sharing scheme. Note that for a general linear secret sharing scheme, the multiplication of two secret sharings can still be done by multiplying every two shares, one from each secret sharing. Reconstructing the secrets of the multiplication result can be done by a linear circuit.

When using (packed) Shamir sharings, the multiplication of two secret sharings can be done by multiplying two vector of shares coordinate-by-coordinate, which reduces the overhead from $O(k^2)$ to $O(k)$, where k is the number of shares. What's more, the packed Shamir secret sharing scheme allows to multiply a vector of $O(k)$ secrets in parallel. Thus, our construction with packed Shamir secret sharing scheme allows efficient evaluation of SIMD circuits that compute $O(k)$ copies of the same sub-circuits. In [DIK10], a general boolean circuit C with size s and depth h can be transformed to a SIMD circuit with size $\tilde{O}(s+kh+k^2)$, where the $\tilde{O}$ notation omits logarithmic factors. Since the ISW construction for linear circuits blows up the circuit size by $O(t\kappa)$, we have the following corollary.

Corollary 2. *Let κ denote the security parameter. For all positive integer t and for all $f : \{0,1\}^n \to \{0,1\}^m$ that admits a circuit of size s and depth h, there exists a $(t, t(s+1)2^{-\kappa})$-D1-tolerant implementation with size $\tilde{O}(t\kappa s + t^2\kappa^2 h + t^3\kappa^3)$.*

5 Parity Leakage Tolerance

5.1 Parity-to-Probing Implies Parity Tolerance

In this subsection, we focus on the first step discussed in Section 2.4. We first define the notions of trivial parity tolerance and parity-to-probing circuits. Then we give a general transformation from parity-to-probing circuits to parity-tolerant circuits.

In Appendix D, we review the construction in [GIM+16] and show that their construction is a parity-to-probing circuit against 1 parity query. We also give an example showing that in general 1-parity tolerance does not imply 2-parity tolerance in Appendix D.3.

Definition 3 (Trivial Parity Tolerance). *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 4 (Parity-to-Probing Circuits). *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 1 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:

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

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

As we mentioned in Section 2.4, without requiring I, O to be trivially parity-tolerant, the notion of parity-to-probing circuits is strictly weaker than parity-resilient circuits as the latter requires to hide all the wires of the input circuit $\tilde{C}$. On the other hand, a parity-resilient circuit usually requires more complicated input encoder and output decoder. In fact, we have the following claim stating that any encoding scheme $(\mathtt{Enc}, \mathtt{Dec})$ cannot be both trivially parity-tolerant and parity-resilient. To be more concrete, for a randomized function $\mathtt{Enc} : \{0,1\} \to \{0,1\}^n$,

- We say $\mathtt{Enc}$ is an encoding function if $\{\mathtt{Enc}(0; \boldsymbol{r})\}_{\boldsymbol{r}}$ and $\{\mathtt{Enc}(1; \boldsymbol{r})\}_{\boldsymbol{r}}$ have disjoint support sets.
- We say $\mathtt{Enc}$ is ϵ-parity-resilient if for all subset W of the output of $\mathtt{Enc}$, $\mathtt{parity}(W(\mathtt{Enc}(0)))$ and $\mathtt{parity}(W(\mathtt{Enc}(1)))$ are statistically close with distance ϵ.

Claim 1 *Let $\mathtt{Enc} : \{0,1\} \to \{0,1\}^n$ be an encoding function. If $\mathtt{Enc}$ admits a trivially parity-tolerant implementation C, then $\mathtt{Enc}$ is not ϵ-parity-resilient, where $\epsilon = \frac{1}{4|C|}$.*

We refer the readers to Appendix E for the proof of Claim 1.

Theorem 3. *Let t, k be integers. Assume that*

- *For all function f, there is a k-probing-tolerant implementation of f;*
- *For all circuit $\tilde{C}$, there is a (t, k, ϵ)-parity-to-probing circuit implementation of $\tilde{C}$.*

Then for all function $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$, there is a (t, ϵ)-parity-tolerant implementation of f.

Proof. Let $f : \{0,1\}^{n_i} \to \{0,1\}^{n_o}$ be the input function. Let $n = \max\{n_i, n_o\}$. Our construction works for all 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.

A direct instantiation of $(\mathtt{Enc}, \mathtt{Dec})$ is that

- $\mathtt{Enc} : \{0,1\}^n \to \{0,1\}^{(k+1)n}$ computes a random additive secret sharing with $k+1$ shares for each input bit.
- $\mathtt{Dec} : \{0,1\}^{(k+1)n} \to \{0,1\}^n$ parses the input as n additive secret sharings, each with $k+1$ shares, and computes each output bit by the parity of the $k+1$ shares.

We describe the parity-tolerant circuit for f (Figure 2).

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 ℓ be the number of random bits used in $\mathtt{Enc}$. Let $f' : \{0,1\}^{\hat{n}} \times \{0,1\}^\ell \to \{0,1\}^{\hat{n}}$ 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.

Fig. 2: Construction of Parity-Tolerant Circuits from Parity-to-Probing Circuits

Lemma 1. *The circuit $C^\star$ constructed in Figure 2 is a (t, ϵ)-parity-tolerant implementation of f.*

We refer the readers to Appendix F for the proof of Lemma 1.

5.2 Feasibility of Parity-Tolerant Circuits

According to Theorem 3, to obtain a (t, ϵ)-parity-tolerant circuit, it is sufficient to construct a (t, k, ϵ)-parity-to-probing circuit defined in Definition 4.

Our starting point is the construction in [GIM+16]. Let $\tilde{C}$ be a circuit that only consists of NAND gates. Recall that the idea is to encode each wire value in $\tilde{C}$ by a small-bias encoding scheme. In the following, we will focus on the 1/2-bias encoding scheme in Figure 1. Suppose

- $\boldsymbol{w} = (w_1, \ldots, w_{|\tilde{C}|})$ are the wire values in $\tilde{C}$.
- $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^{|\tilde{C}|}$ are two vectors of random bits.

Then, the compiler in [GIM+16] takes $\tilde{C}$ as input and outputs $(\hat{I}, \hat{C}, \hat{O})$ where the wire values in $\hat{C}$ 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, $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w})$.

We observe that $(\hat{I}, \hat{C}, \hat{O})$ is already a (t, k, ϵ)-parity-to-probing circuit assuming NAND gadget for some proper k and ϵ. The intuition is as follows.

Intuitions. First, for each subset W of wires in $\hat{C}$, there is a *deterministic* set $\mathcal{V}(W)$ of wires in $\tilde{C}$ of size at most k' such that $\mathtt{parity}(W)$ can be simulated with statistical error $2^{-k'-1}$ if we learn $\mathsf{val}(\mathcal{V}(W))$. More concretely, for a set W of wires in $\hat{C}$, we may define a set $\mathcal{V}(W)$ of wires in $\tilde{C}$ as follows:

- If W touches more than k' different encodings, then we set $\mathcal{V}(W) = \emptyset$.
- Otherwise, we set $\mathcal{V}(W) = \{w_i \mid \text{some bit of } \mathtt{Enc}(w_i) = (u_i, v_i, u_i \cdot v_i \oplus w_i) \text{ is in } W\}$.

Then, if $|\mathcal{V}(W)| \neq \emptyset$, $\mathtt{parity}(W)$ can be perfectly simulated if we learn $\mathsf{val}(\mathcal{V}(W))$. If $|\mathcal{V}(W)| = \emptyset$, then either W is an empty set, indicating that $\mathtt{parity}(W) = 0$, or W touches more than k' different encodings, indicating that $\mathtt{parity}(W)$ is statistically close to a uniform bit with error $2^{-k'-1}$ (See the proof of Theorem 9). In any case, $\mathtt{parity}(W)$ can be simulated with statistical error $2^{-k'-1}$ when learning wire values in $\mathcal{V}(W)$.

Now, for two sets W and W', we define $W \oplus W' = (W \cup W') \backslash (W \cap W')$. Then we have $\mathtt{parity}(W) \oplus \mathtt{parity}(W') = \mathtt{parity}(W \oplus W')$. Now for t subsets $W_1, \ldots, W_t$, we want to simulate the joint distribution of $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$. Let $V = \cup_{S \subset \{1,\ldots,t\}} (\mathcal{V}(\oplus_{i \in S} W_i))$. Then if we learn all wire values in V, we can simulate $\mathtt{parity}(\oplus_{i \in S} W_i)$ with statistical error $2^{-k'-1}$ for all subset $S \subset \{1, \ldots, t\}$. By the XOR lemma (See Appendix C), we can simulate the joint distribution of $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$ with statistical error $2^{t/2-k'-1}$ from the wire values in V.

Regarding the size of V, we note that $\mathcal{V}(\cdot)$ satisfies that for all subsets W, W', if $\mathcal{V}(W), \mathcal{V}(W')$ are not the empty set, then $\mathcal{V}(W \oplus W') \subset \mathcal{V}(W) \cup \mathcal{V}(W')$. Indeed,

- If $\mathcal{V}(W \oplus W') = \emptyset$, then $\mathcal{V}(W \oplus W') \subset \mathcal{V}(W) \cup \mathcal{V}(W')$.
- If $\mathcal{V}(W \oplus W') \neq \emptyset$, then for all $w_i \in \mathcal{V}(W \oplus W')$, some bit of $\mathtt{Enc}(w_i)$ is in $W \oplus W'$. This implies that at least one of W and W' contains some bit of $\mathtt{Enc}(w_i)$, indicating that $w_i \in \mathcal{V}(W) \cup \mathcal{V}(W')$. Thus, $\mathcal{V}(W \oplus W') \subset \mathcal{V}(W) \cup \mathcal{V}(W')$.

With this property, we can prove that $|V| \leq t \cdot k'$. Thus $(\hat{I}, \hat{C}, \hat{O})$ is a $(t, t \cdot k', 2^{t/2-k'-1})$-parity-to-probing circuit.

Realizing NAND *Gadgets.* Thus, to obtain a (t, k, ϵ)-parity-to-probing circuit compiler in the plain model, our goal is to remove the NAND gadget in the construction in [GIM+16].

We summarize our construction for NAND in Figure 3 and refer the readers to Section 2.4 for intuitions about our construction.

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

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

   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

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

Fig. 3: Circuit Implementation of NAND Gadget

Our Construction. We summarize our construction in Figure 4.

Theorem 4. *For all circuit $\tilde{C}$, the construction in Figure 4 is a (t, k, ϵ)-parity-to-probing circuit, where*

$$\epsilon = 2^{\frac{t}{2}+1} \cdot \left(\frac{7}{8}\right)^{\frac{k}{2t}}.$$

We prove Theorem 4 in Appendix G.

Towards t-Parity-Tolerant Circuits. By Theorem 3 and Theorem 4, we have the following corollary.

Corollary 3. *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 $poly(t, \kappa, s)$.*

(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)

Input Encoder I: For input $\boldsymbol{x} \in \{0,1\}^{n_i}$, $I(\boldsymbol{x}) = \{(u_i, v_i, u_i \cdot v_i \oplus x_i)\}_{i=1}^{n_i}$, where u_i, v_i are random bits for all $i \in \{1, \ldots, n_i\}$.

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 and samples the following random bits:
   - $\boldsymbol{u}, \boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)} \in \{0,1\}^{|\tilde{C}|}$ where the first n_i bits of $\boldsymbol{u}$ are set to be the same as those in $I(\boldsymbol{x})$.
   - $\boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)} \in \{0,1\}^{|\tilde{C}|}$ where the first n_i bits of $\boldsymbol{v}$ are set to be the same as those in $I(\boldsymbol{x})$.
   - $\boldsymbol{\rho}, \boldsymbol{\rho}^{(1)}, \boldsymbol{\rho}^{(2)} \in \{0,1\}^{|\tilde{C}|}$.
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 3 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 $\boldsymbol{y} = (u_{|\tilde{C}|-n_o+1} \cdot v_{|\tilde{C}|-n_o+1} \oplus z_{|\tilde{C}|-n_o+1}, \ldots, u_{|\tilde{C}|} \cdot v_{|\tilde{C}|} \oplus z_{|\tilde{C}|})$.

Fig. 4: (t, k, ϵ)-Parity-to-Probing Circuit

Acknowledgements. 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.

References

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.

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.

BCG+19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 291–308, New York, NY, USA, 2019. Association for Computing Machinery.

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.

BDF+22. Andrej Bogdanov, Krishnamoorthy Dinesh, Yuval Filmus, Yuval Ishai, Avi Kaplan, and Akshayaram Srinivasan. Bounded indistinguishability for simple sources. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

BFKL93. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

BGR18. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.

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

Bra11. Mark Braverman. Poly-logarithmic independence fools bounded-depth boolean circuits. *Commun. ACM*, 54(4):108–115, apr 2011.

CGLS21. Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021.

CGPZ16. Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 498–514. Springer, 2016.

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

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

DF12. Stefan Dziembowski and Sebastian Faust. Leakage-resilient circuits without computational assumptions. In Ronald Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 230–247. Springer, 2012.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 445–465, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

DLZ15. Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou. Leakage-resilient circuits revisited - optimal number of computing components without leak-free hardware. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 131–158. Springer, 2015.

DP08. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 293–302. IEEE Computer Society, 2008.

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+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.

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

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

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.

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.

MV13. Eric Miles and Emanuele Viola. Shielding circuits with groups. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 251–260. ACM, 2013.

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.

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

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.

A Leakage-Resilient Stateful Circuits

We generalize the definitions of memory cells, stateful circuits, and t-probing-resilient stateful circuits in [ISW03] to the general case.

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 $\boldsymbol{s}_0$. A stateful circuit C can also have external input and output wires.*

Remark 1. Stateful circuits can have external input and output wires. One example 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 7 (Leakage-Resilient Stateful Circuits). *Let $\mathcal{L}$ be a class of leakage functions and $C[\boldsymbol{s}_0]$ be a stateful circuit C with an initial state $\boldsymbol{s}_0$. A stateful circuit $C'[\boldsymbol{s}'_0]$ is an $\mathcal{L}$-leakage-resilient implementation of $C[\boldsymbol{s}_0]$ if it is secure against an $\mathcal{L}$-leakage-limited interactive adversary. Specifically, an $\mathcal{L}$-leakage-limited interactive adversary is an adversary which is given access to $C'[\boldsymbol{s}'_0]$. The adversary may invoke C' q times where q is an arbitrary polynomial in the security parameter. In each invocation, the adversary adaptively chooses an input $\boldsymbol{x}$ and a leakage function $L \in \mathcal{L}$ with input size $|C'|$ based on the observed output values and answers to the leakage queries in previous invocations, and the adversary learns the output values corresponding to the chosen input and the answer to the leakage query L. The security requires the existence of a simulator $\mathtt{Sim}$ that can simulate the adversary's view using only a black-box access to $C[\boldsymbol{s}_0]$, without the access to $\boldsymbol{s}_0$, with negligible statistical error.*

A.1 From Leakage-Tolerant Circuits to Leakage-Resilient Stateful Circuits

In this section, we show that for any leakage class $\mathcal{L}$, $\mathcal{L}$-leakage-resilient stateful circuits can be constructed from $\mathcal{L}$-leakage tolerate circuits and 2-adaptive $\mathcal{L}$-leakage-resilient encoding schemes. Concretely, for a pair of randomized functions $(\mathtt{Enc}, \mathtt{Dec})$,

- We say $(\mathtt{Enc}, \mathtt{Dec})$ is an encoding scheme if for all $m \in \{0,1\}$,

$$\Pr[\mathtt{Dec}(\mathtt{Enc}(m)) = m] = 1.$$

- We say $(\mathtt{Enc}, \mathtt{Dec})$ is 2-adaptive $\mathcal{L}$-leakage-resilient if 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:

$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$
$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$

Theorem 5. *Let $\mathcal{L}$ be a leakage class. Suppose there is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding, and an efficient compiler transforming any binary function $f : \{0,1\}^n \to \{0,1\}^m$ to an $\mathcal{L}$-leakage-tolerant circuit. Then, there is an efficient compiler transforming any stateful circuit $C[\boldsymbol{s}_0]$ to an equivalent $\mathcal{L}$-leakage-resilient stateful circuit $C'[\boldsymbol{s}'_0]$.*

Proof. We follow the high-level idea presented in Section 2.1. Let $(\mathtt{Enc}, \mathtt{Dec})$ denote a 2-adaptive $\mathcal{L}$-leakage-resilient encoding scheme. For a string $\boldsymbol{s}$, we use $\mathtt{Enc}(\boldsymbol{s})$ to denote the concatenation of the encoding of every bit in $\boldsymbol{s}$.

Initially, we set $\boldsymbol{s}'_0 = \mathtt{Enc}(\boldsymbol{s}_0)$. Let f be a binary function defined as follows.

- The function f takes $\boldsymbol{s}', \boldsymbol{x}$ as input.
- Then it decodes $\boldsymbol{s}'$ by using $\mathtt{Dec}$ and obtains $\boldsymbol{s}$.
- Next it computes $C[\boldsymbol{s}]$ with public input $\boldsymbol{x}$ and obtains the updated state $\tilde{\boldsymbol{s}}$ with public output $\boldsymbol{y}$.
- After that, it encodes $\tilde{\boldsymbol{s}}$ by using $\mathtt{Enc}$ and obtains $\tilde{\boldsymbol{s}}'$.
- Finally f outputs $(\tilde{\boldsymbol{s}}', \boldsymbol{y})$.

Let $\hat{C}$ be an $\mathcal{L}$-leakage-tolerant circuit for f. The construction of $C'[\boldsymbol{s}'_0]$ is as follows: In the i-th invocation with public input $\boldsymbol{x}_i$, $C'[\boldsymbol{s}'_{i-1}]$ computes $\hat{C}(\boldsymbol{s}'_{i-1}, \boldsymbol{x})$ and obtains $(\boldsymbol{s}'_i, \boldsymbol{y})$, which are regarded as the updated state and the public output respectively.

We first give the construction of the simulator $\mathtt{Sim}$. Let $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$ be the simulator guaranteed by the $\mathcal{L}$-leakage tolerance of $\hat{C}$. In the beginning, $\mathtt{Sim}$ sets $\tilde{\boldsymbol{s}}_0$ to be an all-0 string of the same length as $\boldsymbol{s}_0$. Then $\mathtt{Sim}$ computes $\tilde{\boldsymbol{s}}'_0 = \mathtt{Enc}(\tilde{\boldsymbol{s}}_0)$. $\mathtt{Sim}$ does the following for the i-th invocation.

- $\mathtt{Sim}$ invokes $C[\boldsymbol{s}_{i-1}]$ with public input $\boldsymbol{x}_i$ and obtains the public output $\boldsymbol{y}_i$.
- $\mathtt{Sim}$ sets $\tilde{\boldsymbol{s}}_i$ to be an all-0 string of the same length as $\boldsymbol{s}_i$. Then $\mathtt{Sim}$ computes $\tilde{\boldsymbol{s}}'_i = \mathtt{Enc}(\tilde{\boldsymbol{s}}_i)$.
- Upon receiving the leakage query L_i from $\mathcal{A}$, $\mathtt{Sim}$ runs $\mathtt{Sim}'_1(L_i)$ and obtains $(\mathtt{st}_i, L'_i)$. Then $\mathtt{Sim}$ runs $\mathtt{Sim}'_2(\mathtt{st}_i, L'_i(\tilde{\boldsymbol{s}}'_{i-1}, \boldsymbol{x}_i, \tilde{\boldsymbol{s}}'_i, \boldsymbol{y}_i))$ and obtains $\boldsymbol{b}_i$. Finally $\mathtt{Sim}$ returns $\boldsymbol{b}_i$ as the simulated leakage to $\mathcal{A}$.

We prove that $C'[\boldsymbol{s}'_0]$ is an $\mathcal{L}$-leakage-resilient stateful circuit for $C[\boldsymbol{s}_0]$ by a list of hybrid arguments.

$\mathbf{Hybrid}_0$: In the initial hybrid, we consider the real execution.

$\mathbf{Hybrid}_1$: In this hybrid, we consider a sequence of small hybrids that change the answer to the i-th leakage query by the one simulated by $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$. In the i-th small hybrid, for the leakage query L_i in the i-th invocation, we first run $\mathtt{Sim}'_1(L_i)$ and obtains $(\mathtt{st}_i, L'_i)$. Then we run $\mathtt{Sim}'_2(\mathtt{st}_i, L'_i(\boldsymbol{s}'_{i-1}, \boldsymbol{x}_i, \boldsymbol{s}'_i, \boldsymbol{y}_i))$ and obtains $\boldsymbol{b}_i$. Finally we return $\boldsymbol{b}_i$ as the simulated leakage to $\mathcal{A}$. Since $\hat{C}$ is $\mathcal{L}$-leakage-tolerant, the distribution of $\mathbf{Hybrid}_{1,i}$ is statistically close to $\mathbf{Hybrid}_{1,i-1}$. Note that in the last small hybrid, all leakage queries are simulated by $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$, which only depend on the encodings of states.

$\mathbf{Hybrid}_2$: In this hybrid, we consider a sequence of small hybrids that switch $\{\boldsymbol{s}'_i\}_{i=1}^q$ by $\{\tilde{\boldsymbol{s}}'_i\}_{i=1}^q$ one by one. In the j-th small hybrid, the only difference is that we use $\tilde{\boldsymbol{s}}'_j$ rather than $\boldsymbol{s}'_j$. Given $\{\tilde{\boldsymbol{s}}'_i\}_{i<j}$ and $\{\boldsymbol{s}'_i\}_{i>j}$, the only difference are the answers to the j-th leakage query and the $(j+1)$-th leakage query, which translates to two adaptive leakage queries in $\mathcal{L}$ to $\tilde{\boldsymbol{s}}'_j$. Since $(\mathtt{Enc}, \mathtt{Dec})$ is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding scheme, the distribution of $\mathbf{Hybrid}_{2,j}$ is statistically close to $\mathbf{Hybrid}_{2,j-1}$. Note that the last small hybrid corresponds to the case where all leakage queries are simulated by $\mathtt{Sim}$. Therefore, $C'[\boldsymbol{s}'_0]$ is an $\mathcal{L}$-leakage-resilient stateful circuit for $C[\boldsymbol{s}_0]$.

In the following, we give instantiations of 2-adaptive $\mathcal{L}$-leakage-resilient encodings for leakage classes we are interested in. We also show how to construct an $\mathcal{L}$-leakage-resilient stateful circuit with efficient simulation even if the underlying $\mathcal{L}$-leakage-tolerant circuits require inefficient simulation.

A.2 Construction of 2-Adaptive $\mathcal{L}$-Leakage-Resilient Encodings.

In this subsection, we first show that any $\mathcal{L}^{\otimes 2}$-leakage-resilient encoding (i.e., an encoding scheme that is secure against 2 *non-adaptive* leakage queries in $\mathcal{L}$) achieves 2-adaptive $\mathcal{L}$-leakage resilience.

Lemma 2. *Let* $(\mathtt{Enc}, \mathtt{Dec})$ *be an* $\mathcal{L}^{\otimes 2}$*-leakage-resilient encoding with error* ϵ*. Let* t *be the output length of functions in* $\mathcal{L}$*. Then* $(\mathtt{Enc}, \mathtt{Dec})$ *is a 2-adaptive* $\mathcal{L}$*-leakage-resilient encoding with error* $2^t\epsilon$*.*

Proof. For the sake of contradiction, suppose there exists a 2-adaptive $\mathcal{L}$-leakage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that the statistical distance between the following two distributions is $\eta > 2^t\epsilon$:

$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$
$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))$$

We prove the existence of an $\mathcal{L}^{\otimes 2}$-leakage adversary $\mathcal{A}'$ such that the following two distributions is at least $\eta/2^t > \epsilon$:

$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, L_2) \leftarrow \mathcal{A}'(|\mathtt{Enc}(0)|)$$
$$(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, L_2) \leftarrow \mathcal{A}'(|\mathtt{Enc}(0)|)$$

This is equivalent to the existence of $(\mathcal{A}'', D'')$ which satisfies that

$$\begin{aligned}&|\Pr[c \leftarrow \mathtt{Enc}(0), (L_1, L_2, \mathtt{st}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \mathtt{st}) = 1]\\&-\Pr[c \leftarrow \mathtt{Enc}(1), (L_1, L_2, \mathtt{st}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \mathtt{st}) = 1]| \geq \eta/2^t.\end{aligned}$$

This is because we may choose the random tape $\boldsymbol{r}$ of $\mathcal{A}''$ to maximize the above difference. Then $\mathcal{A}''$ becomes a deterministic algorithm, and we may hardcode $\mathtt{st}$ inside D''. Now we can construct $\mathcal{A}'$ which simply runs $\mathcal{A}''$ with random tape $\boldsymbol{r}$ and outputs L_1, L_2. Then the above difference implies a lower bound on the statistical distance between the following two distributions:

$$\begin{aligned}&(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, L_2) \leftarrow \mathcal{A}'(|\mathtt{Enc}(0)|)\\&(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, L_2) \leftarrow \mathcal{A}'(|\mathtt{Enc}(0)|)\end{aligned}$$

In the following we focus on the construction of $\mathcal{A}''$ and D''. We construct $\mathcal{A}''$ as follows.

1. $\mathcal{A}''$ invokes $\mathcal{A}_1$ with $|\mathtt{Enc}(0)|$ and receives $(L_1, \mathtt{st})$.
2. $\mathcal{A}''$ randomly samples $\boldsymbol{b} \in \{0,1\}^t$ as a guess of applying L_1 to the encoding.
3. $\mathcal{A}''$ invokes $\mathcal{A}_2$ with $\mathtt{st}, |\mathtt{Enc}(0)|, \boldsymbol{b}$ and receives L_2.
4. $\mathcal{A}''$ outputs $(L_1, L_2, \boldsymbol{b})$.

By the definition of statistical distance, there exists a computationally unbounded distinguisher D such that D can distinguish the following two distributions with advantage η:

$$\begin{aligned}&(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(0), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))\\&(L_1(c), L_2(c)) : c \leftarrow \mathtt{Enc}(1), (L_1, \mathtt{st}) \leftarrow \mathcal{A}_1(|\mathtt{Enc}(0)|), L_2 \leftarrow A_2(\mathtt{st}, |\mathtt{Enc}(0)|, L_1(c))\end{aligned}$$

We construct D'' as follows.

– D'' takes $(L_1(c), L_2(c), \boldsymbol{b})$ as input. Then D'' checks whether $\boldsymbol{b} = L_1(c)$. If true, D'' outputs $D(L_1(c), L_2(c))$. Otherwise D'' outputs a random bit.

Then we have

$$\begin{aligned}&|\Pr[c \leftarrow \mathtt{Enc}(0), (L_1, L_2, \boldsymbol{b}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \boldsymbol{b}) = 1]\\&-\Pr[c \leftarrow \mathtt{Enc}(1), (L_1, L_2, \boldsymbol{b}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \boldsymbol{b}) = 1]|\\=\ &\frac{1}{2^t}|\Pr[c \leftarrow \mathtt{Enc}(0), (L_1, L_2, \boldsymbol{b}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \boldsymbol{b}) = 1 \mid L_1(c) = \boldsymbol{b}]\\&-\Pr[c \leftarrow \mathtt{Enc}(1), (L_1, L_2, \boldsymbol{b}) \leftarrow \mathcal{A}'' : D''(L_1(c), L_2(c), \boldsymbol{b}) = 1 \mid L_1(c) = \boldsymbol{b}]|\\=\ &\frac{\eta}{2^t}.\end{aligned}$$

The last step is because given $L_1(c) = \boldsymbol{b}$, $\mathcal{A}''$ behaves identically to $\mathcal{A}$ and D'' behaves identically to D.

Instantiations of 2-Adaptive $\mathcal{L}$-Leakage-Resilient Encodings. By Lemma 2, it is sufficient to construct $\mathcal{L}^{\otimes 2}$-leakage-resilient encodings. We given instantiations for the leakage classes we are interested in this work.

For t-probing leakage, it is sufficient to use a random additive secret sharing with $2t+1$ shares as the encoding scheme. I.e., $\mathtt{Enc}(m)$ outputs $2t+1$ random bits with parity m. The obtained encoding scheme is 2-adaptive t-probing-resilient with no error.

For t-D1 leakage, it is sufficient to use a random additive secret sharing with $2t(\kappa+t)$ shares as the encoding scheme. I.e., $\mathtt{Enc}(m)$ outputs $2t(\kappa+t+\log(4t))$ random bits with parity m. The obtained encoding scheme is 2-adaptive t-D1-resilient with error $2^{-\kappa}$. This can be proved by following a similar argument to that in the proof of Theorem 7.

For t-parity leakage, we follow the idea in [GIM+16] by first splitting the input bit to an additive secret sharing with $2t+\kappa+1$ shares and then encoding each share by the small-bias encoding scheme in Figure 1. The obtained encoding scheme is 2-adaptive t-parity-resilient with error $2^{-\kappa}$. This can be proved by combining Claim 5 and the XOR Lemma 4

A.3 Leakage-Resilient Stateful Circuits with Efficient Simulation

We note that for the $\mathcal{L}$-leakage-resilient stateful circuit $C'[\boldsymbol{s}'_0]$ constructed in Theorem 5, the efficiency of the simulator depends on the simulator of the underlying $\mathcal{L}$-leakage-tolerant circuit $\hat{C}$. If $\hat{C}$ requires inefficient simulation (in the output length of each leakage query), then $C'[\boldsymbol{s}'_0]$ also requires inefficient simulation. This is the case when using our construction for t-parity-tolerant circuits to build t-parity-resilient stateful circuits. On the other hand, we provides evidence that the inefficient simulation of our t-parity-tolerant circuits may be inherent in Appendix H.

In this subsection, we show how to construct leakage-resilient stateful circuits with efficient simulation even if the underlying $\mathcal{L}$-leakage-tolerant circuit requires inefficient simulation. To this end, we need to use a one-to-one $\mathcal{L}$-leakage-resilient encoding. To be more concrete, we say $(\mathtt{Enc}, \mathtt{Dec})$ is a one-to-one encoding scheme, if $\mathtt{Enc} : \{0,1\} \to \{0,1\}^n$ satisfies that any $\boldsymbol{y} \in \{0,1\}^n$ is a valid encoding of either $m = 0$ or $m = 1$.

Theorem 6. *Let $\mathcal{L}$ be a leakage class. Suppose there is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding, a one-to-one $\mathcal{L}$-leakage-resilient encoding, and an efficient compiler transforming any binary function $f : \{0,1\}^n \to \{0,1\}^m$ to an $\mathcal{L}$-leakage-tolerant circuit. Then, there is an efficient compiler transforming any stateful circuit $C[\boldsymbol{s}_0]$ to an equivalent $\mathcal{L}$-leakage-resilient stateful circuit $C'[\boldsymbol{s}'_0]$ with efficient simulation.*

Proof. At a high level, our idea is to include a control bit b in the state. We modify the stateful circuit C by taking an auxiliary input in each invocation. If $b = 0$, then the circuit ignores the auxiliary input and computes the output by using the internal state and the public input as the original stateful circuit. Otherwise, the circuit keeps the state unchanged and outputs the decoding of the auxiliary input. To achieve the same functionality as the original stateful circuit, the control bit is set to be 0 by default and we just use random bits as the auxiliary input. Now the construction of the $\mathcal{L}$-leakage-resilient stateful circuit is applying the compiler in Theorem 5 to the modified stateful circuit.

To show efficient simulation, the simulator will set the initial state to be all 0 values and set the control bit to be 1 (so by the construction the control bit will remain to be 1 in all invocations). In each invocation, after learning the public input and output, the simulator will replace the auxiliary input by an encoding of the public output. In this way, the simulator simply evaluates the $\mathcal{L}$-leakage-resilient stateful circuit with fake initial state and computes the leakage queried by the adversary.

We describe the formal construction below. Let $(\mathtt{Enc}, \mathtt{Dec})$ be the 2-adaptive $\mathcal{L}$-leakage-resilient encoding, and $(\mathtt{Enc}', \mathtt{Dec}')$ be the one-to-one $\mathcal{L}$-leakage-resilient encoding. Without loss of generality, suppose the public output is of length ℓ.

Initially, we set $\boldsymbol{s}'_0 = \mathtt{Enc}(\boldsymbol{s}_0 \| 0)$, where the appended bit is viewed as the control bit. Let f be a binary function defined as follows.

- The function f takes $\boldsymbol{s}', \boldsymbol{x}, \boldsymbol{r}$ as input, where $|\boldsymbol{r}| = |\mathtt{Enc}'(\mathbf{0}_\ell)|$, and $\mathbf{0}_\ell$ is a string of 0s of length ℓ.
- Then it decodes $\boldsymbol{s}'$ by using $\mathtt{Dec}$ and obtains $\boldsymbol{s} \| b$. It also decodes $\boldsymbol{r}$ and obtains $\tilde{\boldsymbol{y}}$.
- Next it computes $C[\boldsymbol{s}]$ with public input $\boldsymbol{x}$ and obtains the updated state $\tilde{\boldsymbol{s}}$ with public output $\boldsymbol{y}$.
- After that, depending on the control bit b, f computes the following.
  - If $b = 0$, it encodes $\tilde{\boldsymbol{s}} \| b$ by using $\mathtt{Enc}$ and obtains $\tilde{\boldsymbol{s}}'$.
  - Otherwise, it encodes $\boldsymbol{s} \| b$ by using $\mathtt{Enc}$ and obtains $\tilde{\boldsymbol{s}}'$. Then it resets $\boldsymbol{y}$ to be $\tilde{\boldsymbol{y}}$.
- Finally, f outputs $(\tilde{\boldsymbol{s}}', \boldsymbol{y})$.

Let $\hat{C}$ be an $\mathcal{L}$-leakage-tolerant circuit for f. The construction of $C'[\boldsymbol{s}'_0]$ is as follows: In the i-th invocation with public input $\boldsymbol{x}_i$ and internal randomness $\boldsymbol{r}_i$, $C'[\boldsymbol{s}'_{i-1}]$ computes $\hat{C}(\boldsymbol{s}'_{i-1}, \boldsymbol{x}_i, \boldsymbol{r}_i)$ and obtains $(\boldsymbol{s}'_i, \boldsymbol{y}_i)$, which are regarded as the updated state and the public output respectively.

We first give the construction of the simulator $\mathtt{Sim}$. In the beginning, $\mathtt{Sim}$ sets $\tilde{\boldsymbol{s}}_0$ to be an all-0 string of the same length as the actual initial state. Then $\mathtt{Sim}$ computes $\tilde{\boldsymbol{s}}'_0 = \mathtt{Enc}(\tilde{\boldsymbol{s}}_0 \| 1)$. Note that the control bit is $b = 1$. $\mathtt{Sim}$ does the following for the i-th invocation.

- $\mathtt{Sim}$ invokes $C[\boldsymbol{s}_{i-1}]$ with public input $\boldsymbol{x}_i$ and obtains the public output $\boldsymbol{y}_i$.
- $\mathtt{Sim}$ computes $\tilde{\boldsymbol{r}}_i = \mathtt{Enc}'(\boldsymbol{y}_i)$ and computes $\hat{C}(\tilde{\boldsymbol{s}}'_{i-1}, \boldsymbol{x}_i, \tilde{\boldsymbol{r}}_i)$. The output $(\tilde{\boldsymbol{s}}'_i, \boldsymbol{y}_i)$ is regarded as the updated state and the public output respectively.

– Upon receiving the leakage query L_i from $\mathcal{A}$, $\mathtt{Sim}$ computes L_i on the wire values of $\hat{C}$ and returns the simulated leakage to $\mathcal{A}$.

We prove that $C'[\boldsymbol{s}'_0]$ is an $\mathcal{L}$-leakage-resilient stateful circuit for $C[\boldsymbol{s}_0]$ by a list of hybrid arguments. Let $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$ be the simulator guaranteed by the $\mathcal{L}$-leakage tolerance of $\hat{C}$.

$\mathbf{Hybrid}_0$: In the initial hybrid, we consider the real execution.

$\mathbf{Hybrid}_1$: In this hybrid, we consider a sequence of small hybrids that change the answer to the i-th leakage query by the one simulated by $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$. In the i-th small hybrid, for the leakage query L_i in the i-th invocation, we first run $\mathtt{Sim}'_1(L_i)$ and obtains $(\mathtt{st}_i, L'_i)$. Then we run $\mathtt{Sim}'_2(\mathtt{st}_i, L'_i(\boldsymbol{s}'_{i-1}, \boldsymbol{x}_i, \boldsymbol{s}'_i, \boldsymbol{y}_i))$ and obtains $\boldsymbol{b}_i$. Finally we return $\boldsymbol{b}_i$ as the simulated leakage to $\mathcal{A}$. Since $\hat{C}$ is $\mathcal{L}$-leakage-tolerant, the distribution of $\mathbf{Hybrid}_{1,i}$ is statistically close to $\mathbf{Hybrid}_{1,i-1}$. Note that in the last small hybrid, all leakage queries are simulated by $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$, which only depend on the encodings of states and the auxiliary inputs.

$\mathbf{Hybrid}_2$: In this hybrid, we consider a sequence of small hybrids that switch $\{\boldsymbol{s}'_i, \boldsymbol{r}_i\}_{i=0}^q$ by $\{\tilde{\boldsymbol{s}}'_i, \tilde{\boldsymbol{r}}_i\}_{i=0}^q$. (Here we assume $\boldsymbol{r}_0$ and $\tilde{\boldsymbol{r}}_0$ are all-0 strings. Note that they are not used in the construction.) In the $(j+1)$-th small hybrid, the only difference is that we use $\tilde{\boldsymbol{s}}'_j, \tilde{\boldsymbol{r}}_j$ rather than $\boldsymbol{s}'_j, \boldsymbol{r}_j$. Given $\{\tilde{\boldsymbol{s}}'_i, \tilde{\boldsymbol{r}}_i\}_{i<j}$ and $\{\boldsymbol{s}'_i, \boldsymbol{r}_i\}_{i>j}$, the only difference are the answers to the j-th leakage query and the $(j+1)$-th leakage query, which translates to two adaptive leakage queries in $\mathcal{L}$ to $\tilde{\boldsymbol{s}}'_j$ and one leakage query in $\mathcal{L}$ to $\tilde{\boldsymbol{r}}_j$. Since $(\mathtt{Enc}, \mathtt{Dec})$ is a 2-adaptive $\mathcal{L}$-leakage-resilient encoding scheme and $(\mathtt{Enc}', \mathtt{Dec}')$ is a one-to-one $\mathcal{L}$-leakage-resilient encoding scheme, the distribution of $\mathbf{Hybrid}_{2,j+1}$ is statistically close to $\mathbf{Hybrid}_{2,j}$. Note that in the last small hybrid, we use $\tilde{\boldsymbol{s}}'_i, \tilde{\boldsymbol{r}}_i$ in all invocations.

$\mathbf{Hybrid}_3$: In this hybrid, we consider a sequence of small hybrids that change the simulated answer to the i-th leakage query back to that on the wire values of $\hat{C}$. Specifically, in the i-th small hybrid, for the leakage query L_i in the i-th invocation, we output the result of applying L_i on the wire values of $\hat{C}$ with input $(\tilde{\boldsymbol{s}}'_{i-1}, \boldsymbol{x}_i, \tilde{\boldsymbol{r}}_i)$ and output $(\tilde{\boldsymbol{s}}'_i, \boldsymbol{y}_i)$. Since $\hat{C}$ is $\mathcal{L}$-leakage-tolerant and $(\mathtt{Sim}'_1, \mathtt{Sim}'_2)$ is the simulator guaranteed by the $\mathcal{L}$-leakage tolerance of $\hat{C}$, the distribution of $\mathbf{Hybrid}_{3,i}$ is statistically close to $\mathbf{Hybrid}_{3,i-1}$. Note that the last hybrid corresponds to the case where all leakage queries are simulated by $\mathtt{Sim}$. Therefore, $C'[\boldsymbol{s}'_0]$ is an $\mathcal{L}$-leakage-resilient stateful circuit for $C[\boldsymbol{s}_0]$.

Remark 2. We note that the leakage-resilient encoding schemes we constructed in Appendix A.2 for t-probing leakage, t-D1 leakage, and t-parity leakage are all one-to-one encoding schemes.

B Depth-1 $\mathcal{AC}0$ Leakage Tolerance

In this section, we introduce our construction for leakage tolerant circuits against depth-1 $\mathcal{AC}0$ leakage (D1 leakage for short). We follow the overview in Section 2.3 and first show that the ISW construction [ISW03] for linear functions achieves D1 tolerance. Then we show how to construct D1-tolerant circuits for general functions. In Appendix B.4, we give a counter example showing that the ISW construction is not $\mathtt{OR}$-tolerant even for linear functions.

B.1 Review of ISW Construction for Linear Functions

We first consider a linear function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ where each y_j is a linear combination of $\{x_i\}_{i=1}^n$. We will show that the ISW construction for f is D1-tolerant. Recall the ISW construction for the linear function f:

1. Input Encoding Phase: For all $i \in \{1, \ldots, n\}$, do the following.
   – Prepare k random bits $x_{i,0}, \ldots, x_{i,k-1}$.
   – Compute $x_{i,k} = x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,k-1}$ in order.
   – Set $[x_i] = (x_{i,0}, \ldots, x_{i,k})$. Note that the parity of bits in $[x_i]$ is equal to x_i.
2. Circuit Emulation Phase: The circuit computes $[y_j]$ via a proper linear combination of $[x_1], \ldots, [x_n]$.
3. Output Decoding Phase: For all $j \in \{1, \ldots, m\}$, set $(y_{j,0}, \ldots, y_{j,k}) = [y]$ and compute $y_j = y_{j,k} \oplus y_{j,k-1} \oplus \ldots \oplus y_{j,0}$ in order.

For all $j \in \{0, \ldots, k\}$, let $\mathcal{I}_j$ be the set of wires that are linear combinations of $\{x_{i,j}\}_{i=1}^n$. We first prove the following lemma.

Lemma 3. *For all $j \in \{0, \ldots, k\}$, let w_j be an arbitrary variable in $\mathcal{I}_j$. Then any $k-1$ variables of $\{w_j\}_{j=0}^k$ are uniformly random.*

Proof. Note that $[x_1], [x_2], \ldots, [x_n]$ are uniformly random additive sharings. Therefore, for all $j^\star \in \{0, \ldots, k\}$, we have that $\{x_{0,j}, \ldots, x_{n,j}\}_{j \neq j^\star}$ are uniformly random.

Since each variable w_j is a linear combination of $\{x_{i,j}\}_{i=1}^n$, we have that $\{w_j\}_{j \neq j^\star}$ are uniformly random.

B.2 D1 Tolerance of ISW Construction for Linear Functions

We prove the following main theorem about the ISW construction for linear functions.

Theorem 7. *Let κ denote the security parameter. For all positive integer t and for all linear function $f : \{0,1\}^n \to \{0,1\}^m$, when $k = t \cdot \kappa$, the ISW construction for f is $(t, t \cdot 2^{-\kappa})$-D1-tolerant.*

Proof. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ denote the input bits and $\boldsymbol{y} = (y_1, \ldots, y_m)$ denote the output bits. Let C denote the circuit after applying the ISW construction on the linear function f. We represent each D1 leakage on wire values in C as follows:

1. An adversary first chooses an arbitrary set W of wires in C.
2. Then the adversary may arbitrarily flip some wires in W. Denote the resulting set of literals (wire variables or their negations) by $\widetilde{W}$.
3. Finally, the leakage is defined to be $\mathtt{OR}(\widetilde{W})$.

For each D1 leakage, if the number of indices $j \in \{0, \ldots, k\}$ such that $W \cap \mathcal{I}_j \neq \emptyset$ is at least κ, then by Lemma 3, there exist at least κ variables in W that are uniformly random. This means that $\mathtt{OR}(\widetilde{W}) = 1$ with probability at least $1 - 2^{-\kappa}$.

Now suppose $(W_1, \widetilde{W_1}), \ldots, (W_t, \widetilde{W_t})$ are the sets decided by the adversary, and the t D1 leakage bits are $\alpha_1 = \mathtt{OR}(\widetilde{W_1}), \ldots, \alpha_t = \mathtt{OR}(\widetilde{W_t})$. Let T be the set of indices $\ell \in \{1, \ldots, t\}$ such that $W_\ell \cap \mathcal{I}_j \neq \emptyset$ for at most κ indices $j \in \{0, \ldots, k\}$. Then by the above analysis, for each $\ell \notin T$, $\alpha_\ell = 1$ with probability at least $1 - 2^{-\kappa}$. Consider the following two hybrids.

- $\mathbf{Hybrid}_0$: Output $(\alpha_1, \ldots, \alpha_t)$.
- $\mathbf{Hybrid}_1$: For all $\ell \in \{1, \ldots, t\}$, if $\ell \in T$, set $\beta_\ell = \alpha_\ell$; otherwise, set $\beta_\ell = 1$. Output $(\beta_1, \ldots, \beta_t)$.

By the union bound, the statistical distance between $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ is at most $t \cdot 2^{-\kappa}$. We are going to simulate $(\beta_1, \ldots, \beta_t)$ by at most t D1 queries on the input and output bits. Note that it is sufficient to focus on $\{\beta_\ell\}_{\ell \in T} = \{\alpha_\ell\}_{\ell \in T}$.

Recall that $|T| \leq t$ and for all $\ell \in T$, $W_\ell \cap \mathcal{I}_j \neq \emptyset$ for at most κ index $j \in \{0, \ldots, k\}$. Since $k = t \cdot \kappa$, there exists $j^\star \in \{0, \ldots, k\}$ such that $W_\ell \cap \mathcal{I}_{j^\star} = \emptyset$ for all $\ell \in T$. Consider the following simulation process.

1. For all $i \in \{1, \ldots, n\}$, sample k random bits as $\{x_{i,j}\}_{j \neq j^\star}$. By the property of the additive secret sharing scheme, the distribution of $\{x_{i,j}\}_{j \neq j^\star}$ remain unchanged. Then compute all variables in $\bigcup_{j \neq j^\star} \mathcal{I}_j$. Note that it includes all wire values in the circuit emulation phase except wires in $\mathcal{I}_{j^\star}$. Note that all generated variables are independent of the input and output bits.
2. For all $i \in \{1, \ldots, n\}$, compute the intermediate wires in the input encoding phase as follows. For $j < j^\star$, compute $x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,j}$ as it is described, which is an equation depending on x_i. For $j \geq j^\star$, compute $x_i \oplus x_{i,0} \oplus \ldots \oplus x_{i,j}$ by $x_{i,k} \oplus x_{i,k-1} \oplus \ldots \oplus x_{i,j+1}$, which is a known bit.
3. For all $i \in \{1, \ldots, m\}$, compute the intermediate wires in the output decoding phase as follows. For $j > j^\star$, compute $y_{i,k} \oplus \ldots y_{i,j}$ as it is described, which is a known bit. For $j \leq j^\star$, compute $y_{i,k} \oplus \ldots y_{i,j}$ by $y_i \oplus y_{i,0} \oplus \ldots \oplus y_{i,j-1}$, which is an equation depending on y_i.

4. Use the above equation of wires in C to compute $\alpha_\ell = \mathtt{OR}(\widetilde{W}_\ell)$ for all $\ell \in T$. Note that we have generate all wire values except those in $I_{j^\star}$. However, since $W_\ell \cap I_{j^\star} = \emptyset$, we can compute $\mathtt{OR}(\widetilde{W}_\ell)$ by the wires we have generated. Also note that the equation of each variable depends on at most one input bit or output bit. Thus $\mathtt{OR}(\widetilde{W}_\ell)$ is just a D1 leakage query on the input and output bits. Simulate $\{\alpha_\ell\}_{\ell \in T}$ by making $|T|$ D1 leakage queries on the input and output bits.

Note that we just generate the wire values in C except those in $\mathcal{I}_{j^\star}$ in a different way. However, from the first point, the distribution of those wire values remains unchanged. Thus, the output of the simulation has the same distribution as $\{\alpha_\ell\}_{\ell \in T}$.

B.3 D1 Leakage Tolerance for General Functions

In this subsection, we show how to use the ISW construction for linear functions to construct a D1-tolerant circuit for any function $f : \{0,1\}^n \to \{0,1\}^m$. Let $\boldsymbol{x} \in \{0,1\}^n$ denote the input of f, $\boldsymbol{y} = f(\boldsymbol{x})$, and C denote the circuit that computes the function f. At a high level, for each wire w in C, we will compute an additive sharing of w.

1. For each input bit x_i, we sample k random bits $(x_{i,0}, \ldots, x_{i,k-1})$. To obtain $[x_i]$, we want to compute $x_{i,k} = x_i \oplus (\bigoplus_{\ell=0}^{k-1} x_{i,\ell})$. We view the computation of $x_{i,k}$ as a linear function and apply the ISW construction to it.
2. For each addition gate with two input additive sharings $[a], [b]$, we sample random bits $c_0, \ldots, c_{k-1}$. To obtain $[c] = [a+b]$, we want to compute $c_k = (\bigoplus_{\ell=0}^{k} a_\ell) \oplus (\bigoplus_{\ell=0}^{k} b_\ell) \oplus (\bigoplus_{\ell=0}^{k-1} c_\ell)$. We view the computation of c_k as a linear function and apply the ISW construction to it.
3. For each multiplication gate with two input additive sharings $[a], [b]$, we sample random bits $c_0, \ldots, c_{k-1}$. Then we compute $a_{\ell_1} \cdot b_{\ell_2}$ for all $\ell_1, \ell_2 \in \{0, \ldots, k\}$. Our goal is to compute $c_k = (\bigoplus_{\ell_1,\ell_2=0}^{k} a_{\ell_1} \cdot b_{\ell_2}) \oplus (\bigoplus_{\ell=0}^{k-1})$. We view the computation of c_k as a linear function and apply the ISW construction to this linear function.
4. After obtaining an additive secret sharing $[y_j]$ for each output y_j, we want to reconstruct $y_j = \bigoplus_{\ell=0}^{k} y_{j,\ell}$. We view the computation of y_j as a linear function and apply the ISW construction to it.

Theorem 8. *Let κ denote the security parameter. For all positive integer t and for all $f : \{0,1\}^n \to \{0,1\}^m$, when $k = t \cdot \kappa$, the above construction for f is $(t, t(|C|+1) \cdot 2^{-\kappa})$-D1-tolerant, where C is the circuit implementation of f.*

Proof. Let $\widetilde{C}_g$ denote the ISW construction for the linear function that is used to compute the gate g in C, where g can be an input gate, an output gate, an addition gate, or a multiplication gate.

Consider a D1 leakage on wire values in $\widetilde{C}$ defined as follows:

1. An adversary first chooses an arbitrary set W of wires in $\widetilde{C}$.
2. Then the adversary may arbitrarily flip some wires in W. The new set of variables is denoted by $\widetilde{W}$.
3. Finally, the leakage is defined to be $\mathtt{OR}(\widetilde{W})$.

Now suppose $(W_1, \widetilde{W}_1), \ldots, (W_t, \widetilde{W}_t)$ are the sets decided by the adversary, and the t bits of D1 leakage are $\alpha_1 = \mathtt{OR}(\widetilde{W}_1), \ldots, \alpha_t = \mathtt{OR}(\widetilde{W}_t)$. For each gate g in C, let $\widetilde{W}_{\ell,g}$ be the wire values of $\widetilde{W}_\ell$ in $\widetilde{C}_g$. Then we have $\widetilde{W}_\ell = \bigcup_{g \in C} \widetilde{W}_{\ell,g}$. (Note that for a multiplication gate g with inputs a, b, $\widetilde{C}_g$ only includes the wires $a_{\ell_1} \cdot b_{\ell_2}$ for all $\ell_1, \ell_2 \in \{0, \ldots, k\}$ while $[a], [b]$ are covered by the circuits for previous gates where a and b are the output wires.) Consider the following two hybrids.

- **Hybrid**$_0$: Output $(\alpha_1, \ldots, \alpha_t)$.
- **Hybrid**$_1$: Let T_g be the set defined in the proof of Theorem 7 when applying the ISW construction to the linear function for the gate g. Then for all $\ell \notin T_g$, $\mathtt{OR}(\widetilde{W}_{\ell,g}) = 1$ with probability at least $1 - 2^{-\kappa}$. This also implies that $\mathtt{OR}(\widetilde{W}_g) = 1$ with probability at least $1 - 2^{-\kappa}$.
  Let $T = \bigcap_g T_g$. For all $\ell \in \{1, \ldots, t\}$, if $\ell \in T$, set $\beta_\ell = \alpha_\ell$; otherwise, set $\beta_\ell = 1$. Output $(\beta_1, \ldots, \beta_t)$.

By the union bound, the statistical distance between $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ is at most $t \cdot 2^{-\kappa}$. Thus, it is sufficient to simulate $\{\beta_\ell\}_{\ell \in T} = \{\alpha_\ell\}_{\ell \in T}$. Note that for each g, $\{\mathtt{OR}(\widetilde{W}_{\ell,g})\}_{\ell \in T}$ can be perfectly simulated by $|T|$ D1 queries on the input and output bits of g. In particular, the simulation would directly output $|T|$ D1 queries and the simulation results are just the query results. Thus, $\{\alpha_\ell\}_{\ell \in T}$ can be perfectly simulated by $|T|$ D1 queries on (1) the input and output bits $\{\boldsymbol{x}, \boldsymbol{y}\}$, (2) the additive secret sharings of all wires $\{[w]\}_{w \in C}$, (3) and

$$\{a_{\ell_1} \cdot b_{\ell_2} \mid a, b \text{ are inputs of a multiplication gate in } C,\ \ell_1, \ell_2 \in \{0, \ldots, k\}\}.$$

Consider the following hybrids. In $\mathbf{Hybrid}'_i$, we replace the additive secret sharings for the first i wires by random additive secret sharings of 0 and compute $a_{\ell_1} \cdot b_{\ell_2}$ for each multiplication gate with inputs (a, b) accordingly. Then we compute and output $\{\alpha_\ell\}_{\ell \in T}$. Note that in the last hybrid, all additive secret sharings are replaced by random additive secret sharings of 0, which can be generated explicitly, and all $a_{\ell_1} \cdot b_{\ell_2}$ can be computed explicitly as well. In this case, $\{\alpha_\ell\}_{\ell \in T}$ become $|T|$ D1 queries on the input and output bits, which can be simulated perfectly. Now we show that the statistical distance between $\mathbf{Hybrid}'_{i-1}$ and $\mathbf{Hybrid}'_i$ is bounded by $t \cdot 2^{-\kappa}$.

Note that the only difference between $\mathbf{Hybrid}'_{i-1}$ and $\mathbf{Hybrid}'_i$ is the additive secret sharing for the i-th wire. By fixing all other additive secret sharings, $\{\alpha_\ell\}_{\ell \in T}$ become $|T|$ D1 queries on the additive secret sharing for the i-th wire. Following a similar argument to that in Theorem 7, the distribution of $\{\alpha_\ell\}_{\ell \in T}$ in $\mathbf{Hybrid}'_{i-1}$ is statistically close to that in $\mathbf{Hybrid}'_i$ with distance $t \cdot 2^{-\kappa}$.

Therefore, the statistical distance between $\mathbf{Hybrid}'_0$ and $\mathbf{Hybrid}'_{|C|}$ is bounded by $t|C| \cdot 2^{-\kappa}$.

In summary, any t-D1 leakage on $\widetilde{C}$ can be simulated by t-D1 leakage on the input and output bits of f with statistical distance $t(|C| + 1) \cdot 2^{-\kappa}$.

B.4 ISW Construction is Not OR-Tolerant

Consider the ISW construction for the circuit $C(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. The wires of C can be summarized as follows:

1. Input Wires and Input Encodings: $\{x_i, [x_i] = (x_{i,0}, x_{i,1}, \ldots x_{i,k})\}_{i=1}^3, \{x_i \oplus (\bigoplus_{j=0}^{\ell} x_{i,j}) \mid i \in \{1, 2, 3\}, \ell \in \{0, 1, \ldots, k-1\}\}$.
2. Intermediate Results: $\{x_{1,j} \oplus x_{2,j}, x_{1,j} \oplus x_{2,j} \oplus x_{3,j}\}_{j=1}^k$. Let $y_j = x_{1,j} \oplus x_{2,j} \oplus x_{3,j}$.
3. Output Wire and Output Encodings: $\{\bigoplus_{j=0}^{\ell} y_{k-j}\}_{\ell=0}^k$. Note that $y = y_n \oplus \ldots \oplus y_1$.

Consider the following attack: $(x_1 \oplus x_{1,1}) \vee (x_2 \oplus x_{2,1}) \vee (x_{1,1} \oplus x_{2,1}) = (x_1 \oplus x_{1,1}) \vee (x_2 \oplus x_{2,1}) \vee (x_1 \oplus x_2)$. Let $r_1 = x_1 \oplus x_{1,1}$ and $r_2 = x_2 \oplus x_{2,1}$. Then $(x_1 \oplus x_{1,1}) \vee (x_2 \oplus x_{2,1}) \vee \oplus(x_{1,1} \oplus x_{2,1}) = r_1 \oplus r_2 \oplus (x_1 \oplus x_2)$.

We show that this cannot be simulated by one query of OR on x_1, x_2, x_3, y.

Proof. For the sake of contradiction, assume that there exists an simulator Sim that can simulate $(x_1 \oplus x_{1,1}) \vee (x_2 \oplus x_{2,1}) \vee \oplus(x_{1,1} \oplus x_{2,1})$ with one query of OR on x_1, x_2, x_3, y.

For $r_1 \oplus r_2 \oplus (x_1 \oplus x_2)$, when $x_1 \oplus x_2 = 0$, with probability $1/4$, the result is 0. For all subset $W \subset \{x_1, x_2, x_3, y\}$, there exist $p_W, q_{W,0}, q_{W,1} \in [0, 1]$ such that Sim will query the OR of bits in W with probability p_W and if the result is equal to 0, Sim would output 0 with probability $q_{W,0}$, and otherwise, Sim would output 0 with probability $q_{W,1}$. Without loss of generality, we assume that the OR of bits in an empty set is 0.

We first show that for any non-empty set W, $p_W \cdot q_{W,1}$ is negligible. If not, consider the two different inputs $(x_1, x_2, x_3, y) = (1, 0, 0, 1)$ and $(x_1, x_2, x_3, y) = (0, 1, 1, 0)$. In this case, $r'_{1,1} \oplus r'_{2,1} \oplus (x_1 \oplus x_2)$ is always 1. On the other hand, for any non-empty set W, the OR of bits in W is 1 for at least one of the two inputs. Then for at least one of these two cases, Sim would output 0 with non-negligible probability, which leads to a contradiction.

When $W = \emptyset$, we show that $p_W \cdot q_{W,0}$ is negligible. Otherwise, when $(x_1, x_2, x_3, y) = (1, 0, 0, 1)$, Sim would output 0 with non-negligible probability while $r_1 \oplus r_2 \oplus (x_1 \oplus x_2)$ is always 1.

Now we show that such $\mathtt{Sim}$ cannot exist. Consider the input $(x_1, x_2, x_3, y) = (1, 1, 1, 1)$. In this case, the $\mathtt{OR}$ of any non-empty set W is equal to 1. Consider the probability that $\mathtt{Sim}$ outputs 0:

$$\Pr[\mathtt{Sim} = 0] = p_\emptyset \cdot q_{\emptyset,0} + \sum_{W \subset \{x_1,x_2,x_3,y\}, W \neq \emptyset} p_W \cdot q_{W,1}.$$

By the above argument, $p_W \cdot q_{W,1}$ is negligible and when W is an empty set, $p_\emptyset \cdot q_{\emptyset,0}$ is also negligible. This means that $\Pr[\mathtt{Sim} = 0]$ is negligible. However, when $(x_1, x_2, x_3, y) = (1, 1, 1, 1)$, $r_1 \oplus r_2 \oplus (x_1 \oplus x_2) = 0$ with probability $1/4$. It contradicts with the assumption that $\mathtt{Sim}$ can simulate $(x_1 \oplus x_{1,1}) \vee (x_2 \oplus x_{2,1}) \vee \oplus (x_{1,1} \oplus x_{2,1})$ with one query of $\mathtt{OR}$ on x_1, x_2, x_3, y.

C Preliminaries: Fourier Analysis and XOR Lemma

We follow the standard Fourier analysis for binary functions. Let n be an positive integer. For all set $S \subset \{1, \ldots, n\}$, the characters $\chi_S : \{0,1\}^n \to \mathbb{R}$ is defined by $\chi_S(\boldsymbol{x}) = \prod_{i \in S} (-1)^{x_i}$. We have the following two properties:

- For all $\boldsymbol{x}$ and $\boldsymbol{x}'$, $\chi_S(\boldsymbol{x} \oplus \boldsymbol{x}') = \chi_S(\boldsymbol{x}) \cdot \chi_S(\boldsymbol{x}')$.
- For all $S, S' \subset \{1, \ldots, n\}$, let $S'' = (S \cup S') \backslash (S \cap S')$. We simply write $S'' := S \oplus S'$. Then $\chi_{S''}(\boldsymbol{x}) = \chi_S(\boldsymbol{x}) \cdot \chi_{S'}(\boldsymbol{x})$.

The Fourier coefficients of a function $f : \{0,1\}^n \to \mathbb{R}$ are denoted by

$$\hat{f}(S) = \frac{1}{N} \sum_{\boldsymbol{x} \in \{0,1\}^n} f(\boldsymbol{x}) \chi_S(\boldsymbol{x})$$

for all $S \subset \{1, \ldots, n\}$, where $N = 2^n$. Then for all $\boldsymbol{x}$, we have

$$f(\boldsymbol{x}) = \sum_{S \subset \{1,\ldots,n\}} \hat{f}(S) \chi_S(\boldsymbol{x}).$$

Our work uses the XOR lemma from [Gol11].

Lemma 4 (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 $2^{n/2} \cdot \epsilon$.*

D Construction in [GIM+16]

In this section, we show that the construction in [GIM+16] gives a $(1, k, 2^{-k-1})$-parity-to-probing circuit.

Let $\tilde{C}$ denote the input circuit. Without loss of generality, we assume that $\tilde{C}$ only contains $\mathtt{NAND}$ gates. The compiler in [GIM+16] can be divided into two steps. In the first step, the compiler transforms $\tilde{C}$ to $(\hat{I}, \hat{C}, \hat{O})$, where $\hat{I}, \hat{O}$ are the input encoder and output decoder, and $\hat{C}$ is a circuit assuming the existence of a secure $\mathtt{NAND}$ gadget. One may view the first step as a $(1, k, 2^{-k-1})$-parity-to-probing circuit compiler with the aid of a secure $\mathtt{NAND}$ gadget. In the second step, the compiler instantiates the $\mathtt{NAND}$ gadget and outputs (I, C, O).

Step 1: Transforming the Circuit Assuming $\mathtt{NAND}$ *Gadget.* The idea of the first step is to encode each wire value in $\tilde{C}$ by a small-bias encoding scheme.

Definition 8 (Small-bias Encoding [GIM+16]). *We say $(\mathtt{Enc}, \mathtt{Dec})$ is an ϵ-bias encoding if*

– $\mathtt{Enc} : \{0,1\} \to \{0,1\}^n$ *is a randomized function such that (1)* $\mathtt{Enc}$ *output its random tape, and (2) for all* $x \in \{0,1\}$ *and for all* $S \subset \{1,\ldots,n\}$,

$$|\Pr[\chi_S(\mathtt{Enc}(x)) = 1] - \Pr[\chi_S(\mathtt{Enc}(x)) = -1]| \leq \epsilon.$$

– $\mathtt{Dec} : \{0,1\}^n \to \{0,1\}$ *is a deterministic function such that for all* $x \in \{0,1\}$, $\Pr[\mathtt{Enc}(\mathtt{Dec}(x)) = x] = 1$.

A small-bias encoding scheme ensures that any parity of its output is statistically close to a uniform bit with error ϵ independent of its input. An example of a 1/2-bias encoding is presented in Figure 1. In the following, we assume $\epsilon = 1/2$. We refer the readers to [GIM+16] for constructions of other ϵ-bias encoding schemes.

The construction of $(\hat{I}, \hat{C}, \hat{O})$ is as follows:

– $\hat{I}$ takes $\boldsymbol{x} \in \{0,1\}^{n_i}$ as input and outputs $\{\mathtt{Enc}(x_i)\}_{i=1}^{n_i}$.
– $\hat{C}$ takes $\{\mathtt{Enc}(x_i)\}_{i=1}^{n_i}$ as input. For each random bit r used in $\tilde{C}$, $\hat{C}$ samples a random encoding $\mathtt{Enc}(r)$. For each $\mathtt{NAND}$ gate in $\tilde{C}$ with input x_0, x_1, $\hat{C}$ takes as input $\mathtt{Enc}(x_0), \mathtt{Enc}(x_1)$ and outputs $\mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1)$. The computation from $\mathtt{Enc}(x_0), \mathtt{Enc}(x_1)$ to $\mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1)$ is done by a secure $\mathtt{NAND}$ gadget.
– $\hat{O}$ takes $\{\mathtt{Enc}(y_i)\}_{i=1}^{n_o}$ as input and outputs $\{y_i = \mathtt{Dec}(\mathtt{Enc}(y_i))\}_{i=1}^{n_o}$.

Now consider a set W of wires in $\hat{C}$. Intuitively, if W touches more than k different small-bias encodings in $\hat{C}$, then $\mathtt{parity}(W)$ would have bias at most 2^{-k}, which translates to statistical error 2^{-k-1}. Thus, $\mathtt{parity}(W)$ can be simulated by a random bit. On the other hand, if W touches at most k different small-bias encodings in $\hat{C}$, then $\mathtt{parity}(W)$ can be perfectly simulated if we learn the underlying k wires in $\tilde{C}$.

Step 2: Instantiating the $\mathtt{NAND}$ *Gadget by Masking Lemma.* Recall $\hat{I}(\boldsymbol{x}) = \{\mathtt{Enc}(x_i)\}_{i=1}^{n_i}$. Let $\tau(\hat{C}, \hat{I}(\boldsymbol{x}))$ denote all wire values in $\hat{C}$. In the second step, $\tau(\hat{C}, \hat{I}(\boldsymbol{x}))$ are split to additive shares

$$(\boldsymbol{R}, \tau(\hat{C}, \hat{I}(\boldsymbol{x})) \oplus \boldsymbol{R})$$

where $\boldsymbol{R}$ is a random string of size $|\tau(\hat{C}, \hat{I}(\boldsymbol{x}))|$.

As we will show later, the benefit of doing this is that one may compute any function f on $\boldsymbol{R}$ and any function g on $\tau(\hat{C}, \hat{I}(\boldsymbol{x})) \oplus \boldsymbol{R}$ such that $f(\boldsymbol{R}) \oplus g(\tau(\hat{C}, \hat{I}(\boldsymbol{x})) \oplus \boldsymbol{R})$ can be perfectly simulated by one parity of $\tau(\hat{C}, \hat{I}(\boldsymbol{x}))$. Essentially, we may add new wires that are arbitrary functions on either the first share or the second share without breaking the parity-to-probing property obtained in Step 1.

Now the idea is to build a circuit that realizes the $\mathtt{NAND}$ gadget such that each inner wire of the circuit can be solely computed from either $\boldsymbol{R}$ or $\tau(\hat{C}, \hat{I}(\boldsymbol{x})) \oplus \boldsymbol{R}$. With more details, recall that the $\mathtt{NAND}$ gadget takes as input $\mathtt{Enc}(x_0), \mathtt{Enc}(x_1)$ and outputs $\mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1)$. After we split each bit in $\hat{C}$ to additive shares, what we want to compute becomes

$$(\boldsymbol{r}_0, \mathtt{Enc}(x_0) \oplus \boldsymbol{r}_0), (\boldsymbol{r}_1, \mathtt{Enc}(x_1) \oplus \boldsymbol{r}_1) \to (\boldsymbol{r}_2, \mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1) \oplus \boldsymbol{r}_2).$$

We may interpret it as computing some function h that maps $(\boldsymbol{r}_0, \mathtt{Enc}(x_0) \oplus \boldsymbol{r}_0), (\boldsymbol{r}_1, \mathtt{Enc}(x_1) \oplus \boldsymbol{r}_1), \boldsymbol{r}_2$ to $\mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1) \oplus \boldsymbol{r}_2$. The works [GIM+16,GIW17] show an instantiation of h such that every inner wire can be solely computed from either $(\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$ or $(\mathtt{Enc}(x_0) \oplus \boldsymbol{r}_0, \mathtt{Enc}(x_1) \oplus \boldsymbol{r}_1, \mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1) \oplus \boldsymbol{r}_2)$, which is a part of $\boldsymbol{R}$ or $\tau(\hat{C}, \hat{I}(\boldsymbol{x})) \oplus \boldsymbol{R}$.

We give a very high-level intuition about how this is achieved. Let $\boldsymbol{a} = (\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$, $\boldsymbol{b} = (\mathtt{Enc}(x_0) \oplus \boldsymbol{r}_0, \mathtt{Enc}(x_1) \oplus \boldsymbol{r}_1)$, and $\boldsymbol{c} = \mathtt{Enc}(x_0\ \mathtt{NAND}\ x_1) \oplus \boldsymbol{r}_2$. Then the function h maps $\boldsymbol{a}, \boldsymbol{b}$ to $\boldsymbol{c}$. We want to construct a circuit computing h such that each inner wire can be computed solely from $\boldsymbol{a}$ or $(\boldsymbol{b}, \boldsymbol{c})$. The main observation is that for all public constants $\boldsymbol{\beta}$ of size $|\boldsymbol{b}|$,

$$\prod_i (b_i \oplus \beta_i \oplus 1) \cdot h(\boldsymbol{a}, \boldsymbol{b}) = \prod_i (b_i \oplus \beta_i \oplus 1) \cdot h(\boldsymbol{a}, \boldsymbol{\beta}).$$

To see why this is the case, note that when $\boldsymbol{b} = \boldsymbol{\beta}$, the LHS is $h(\boldsymbol{a}, \boldsymbol{b}) = h(\boldsymbol{a}, \boldsymbol{\beta})$ which is the RHS. When $\boldsymbol{b} \neq \boldsymbol{\beta}$, the LHS is always 0, and so is the RHS. Now the circuit of computing h is constructed as follows:

1. For all $\boldsymbol{\beta}$, the circuit computes $\prod_i (b_i \oplus \beta_i \oplus 1)$ and $h(\boldsymbol{a}, \boldsymbol{\beta})$. Note that the computation of $\prod_i (b_i \oplus \beta_i \oplus 1)$ only depends on $\boldsymbol{b}$ while the computation of $h(\boldsymbol{a}, \boldsymbol{\beta})$ only depends on $\boldsymbol{a}$.
2. Then for all $\boldsymbol{\beta}$, the circuit computes $\prod_i (b_i \oplus \beta_i \oplus 1) \cdot h(\boldsymbol{a}, \boldsymbol{\beta}) = \prod_i (b_i \oplus \beta_i \oplus 1) \cdot h(\boldsymbol{a}, \boldsymbol{b}) = \prod_i (b_i \oplus \beta_i \oplus 1) \cdot \boldsymbol{c}$. Note that the result only depends on $\boldsymbol{b}, \boldsymbol{c}$.
3. Finally, the circuit computes $\boldsymbol{c} = \oplus_{\boldsymbol{\beta}} \prod_i (b_i \oplus \beta_i \oplus 1) \cdot \boldsymbol{c}$. The computation only depends on $\boldsymbol{b}, \boldsymbol{c}$.

In summary, the circuit C is obtained from $\hat{C}$ by splitting the wire values to additive shares and replacing the secure `NAND` gadget by an instantiation such that every inner wire can be solely computed from either the first share or the second share. The input encoder and output decoder are modified as follows so that they properly provide the shares of the small-bias encodings and decode the shares of the small-bias encodings:

- I takes as input $\boldsymbol{x} \in \{0,1\}^{n_i}$ and outputs $\{(\boldsymbol{r}_i, \mathtt{Enc}(x_i) \oplus \boldsymbol{r}_i)\}_{i=1}^{n_i}$.
- O takes as input $\{\boldsymbol{r}'_i, \mathtt{Enc}(y_i) \oplus \boldsymbol{r}'_i\}_{i=1}^{n_i}$ and outputs $\boldsymbol{y}$.

Theorem 9. *The construction in [GIM+16] described above with the small-bias encoding scheme instantiated by Figure 2 is a $(1, k, 2^{-k-1})$-parity-to-probing circuit compiler.*

Before proving Theorem 9, we first introduce the parity simulation lemma in Appendix D.1. We will give the proof of Theorem 9 in Appendix D.2.

Obtaining $(1, \epsilon)$-Parity-Tolerant Circuits from [GIM+16]. We note that:

- From [ISW03,GIS22], for all f, there exists a k-probing-tolerant implementation of f.
- The construction from [GIM+16] shows that for all circuit $\tilde{C}$, there is a $(1, k, 2^{-k-1})$-parity-to-probing circuit implementation of $\tilde{C}$.

Thus, we have the following corollary.

Corollary 4. *There exists a polynomial-time circuit compiler that takes as input $(C_f, 1^\kappa)$, where C_f is a circuit of size s that computes f, and outputs a $(1, 2^{-\kappa})$-parity-tolerant implementation of C_f with circuit size poly(κ, s).*

D.1 Parity Simulation Lemma

Recall that the compiler in [GIM+16] first transforms the input circuit $\tilde{C}$ to $(\hat{I}, \hat{C}, \hat{O})$ assuming a secure `NAND` gadget. Then in the second step, the secure `NAND` gadget is instantiated by standard `AND` and `XOR` gates. We note that the second step of the construction in [GIM+16] can be viewed as a randomized function which takes the wire values of $\hat{C}$ as input and outputs the wire values of C.

In [GIM+16], Goyal et al. have shown that any parity of wire values in $\hat{C}$ can be simulated by at most k wire values of the underlying circuit $\tilde{C}$. Thus, if we can prove that *any parity of wire values in C can be simulated by one parity of wire values in $\hat{C}$*, then we can reduce the parity of wire values in C to k probings in $\tilde{C}$, showing that the construction in [GIM+16] is a parity-to-probing circuit compiler.

To this end, we first extend the notion of parity tolerance to randomized functions.

Parity Tolerance for Randomized Functions. Intuitively, we say a randomized function F is t-parity tolerant if any t parity attacks towards the output of F can be simulated by t parities of its input.

Definition 9 (t-Parity-Tolerant Functions). *For a randomized function $F : \{0,1\}^n \to \{0,1\}^m$, we say F is (t, ϵ)-parity-tolerant (or (t, ϵ)-PT) if there exists a simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ with the following syntax:*

- $\mathtt{Sim}_1$ *takes as input t subsets $W_1, \ldots, W_t$ of the output of F, and outputs a state* `st` *and t subsets $V_1, \ldots, V_t$ of the input of F;*
- $\mathtt{Sim}_2$ *takes the state* `st` *and* $\mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t)$ *as input, and outputs t bits $(b_1, \ldots, b_t)$.*

The simulator $\mathtt{Sim}$ *satisfies that for all input* $\boldsymbol{x} \in \{0,1\}^n$ *and for all* t *subsets* $W_1, \ldots, W_t$ *of the output of* F*, the following two distributions are statistically close with error* ϵ*:*

$$\begin{aligned}&(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t))\\ \approx_\epsilon\ &(\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t))) : (\mathtt{st}, V_1, \ldots, V_t) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t).\end{aligned}$$

Sometimes, we may explicitly write the randomness of F by $F(\boldsymbol{x}; \boldsymbol{r})$, which means that F takes $\boldsymbol{x}$ as input and uses $\boldsymbol{r}$ as random tape.

Parity Simulation Lemma. While we may directly deal with the randomized function that maps the wire values in $\hat{C}$ to the wire values in C, we note that it is sufficient to only focus on the parity of a chosen subset W of wires in C.

Recall that the second step in [GIM+16] works as follows:

1. The compiler splits the wire values in $\hat{C}$ to additive shares $\boldsymbol{R}$ and $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$.
2. The secure $\mathtt{NAND}$ gadget is replaced by some circuit such that each wire value is either a function on $\boldsymbol{R}$ or a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$.

Thus, we may separate W to two parts W_L and W_R, where each wire in W_L is a function on $\boldsymbol{R}$ and each wire in W_R is a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Then $\mathtt{parity}(W) = \mathtt{parity}(W_L) \oplus \mathtt{parity}(W_R) = f(\boldsymbol{R}) \oplus g(\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R})$ for some functions f, g. We consider the randomized function F that takes as input $\tau(\hat{C}, \boldsymbol{x})$ and outputs $\mathtt{parity}(W)$. It is sufficient to show that for all f, g, the corresponding randomized function F is $(1, \epsilon)$-PT.

We have the following lemma.

Lemma 5 (Parity Simulation Lemma). *Let* n *be a positive integer. For all* $f, g : \{0,1\}^n \to \{0,1\}$*, the randomized function* $F : \{0,1\}^n \to \{0,1\}$ *defined by*

$$F(\boldsymbol{x}; \boldsymbol{r}) = f(\boldsymbol{r}) \oplus g(\boldsymbol{x} \oplus \boldsymbol{r}),$$

where $\boldsymbol{r} \in \{0,1\}^n$ *is uniformly distributed, is* $(1,0)$*-PT.*

Proof. We first give a sufficient condition of $(1,0)$-PT.

Claim 2 *Let* $F : \{0,1\}^n \to \{0,1\}$ *be a randomized function. If there exists* $\hat{v}(S)$ *in the range* $[-1,1]$ *for all* $S \subset \{1, \ldots, n\}$ *such that*

- *For all* $\boldsymbol{x} \in \{0,1\}^n$*,* $\Pr[F(\boldsymbol{x}) = 0] - \Pr[F(\boldsymbol{x}) = 1] = \sum_{S \subset \{1,\ldots,n\}} \hat{v}(S) \chi_S(\boldsymbol{x})$*,*
- $\sum_{S \subset \{1,\ldots,n\}} |\hat{v}(S)| \leq 1$*,*

then F *is* $(1,0)$*-PT.*

Proof (Proof of Claim 2). Note that the output of F is a single bit. Therefore, we only need to construct a simulator $\mathtt{Sim}$ that simulates the output of F. Consider the following construction:

- $\mathtt{Sim}_1$ generates the states $\mathtt{st}$ and the set V as follows:
  1. For all $S \subset \{1, \ldots, n\}$, with probability $|\hat{v}(S)|$, $\mathtt{Sim}_1$ generates
  $$(\mathtt{st} = (S, \frac{1 - \mathtt{sign}(\hat{v}(S))}{2}), V = \{x_i\}_{i \in S}).$$
  In other words, $\mathtt{st}_2 = 0$ if $\hat{v}(S) \geq 0$ and $\mathtt{st}_2 = 1$ if $\hat{v}(S) < 0$.
  2. With probability $1 - \sum_{S \subset \{1,\ldots,n\}} |\hat{v}(S)|$, $\mathtt{Sim}_1$ generates
  $$(\mathtt{st} = (\bot, \bot), V = \emptyset),$$
  where u is a uniform bit.

– $\mathtt{Sim}_2$ takes $\mathtt{st}$ and $\mathtt{parity}(V)$ as input. If $\mathtt{st} = (\bot, \bot)$, $\mathtt{Sim}_2$ outputs a random bit. Otherwise, $\mathtt{Sim}_2$ outputs $b = \mathtt{st}_2 \oplus \mathtt{parity}(V)$.

Note that the above simulator is well defined since $\sum_{S\subset\{1,\ldots,n\}} |\hat{v}(S)| \leq 1$.

Let $\mathtt{Sim}(\boldsymbol{x})$ denote the output of $\mathtt{Sim}$ when the input of F is $\boldsymbol{x}$. Since we always have $\Pr[\mathtt{Sim}(\boldsymbol{x}) = 0] + \Pr[\mathtt{Sim}(\boldsymbol{x}) = 1] = \Pr[F(\boldsymbol{x}) = 0] + \Pr[F(\boldsymbol{x}) = 1] = 1$, to show that $\Pr[\mathtt{Sim}(\boldsymbol{x}) = 0] = \Pr[F(\boldsymbol{x}) = 0]$, it is sufficient to prove that

$$\Pr[\mathtt{Sim}(\boldsymbol{x}) = 0] - \Pr[\mathtt{Sim}(\boldsymbol{x}) = 1] = \Pr[F(\boldsymbol{x}) = 0] - \Pr[F(\boldsymbol{x}) = 1].$$

Observe that

$$\begin{aligned}
&\Pr[\mathtt{Sim}(\boldsymbol{x}) = 0] - \Pr[\mathtt{Sim}(\boldsymbol{x}) = 1] \\
&= \sum_{(\mathtt{st},V)} (\Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 0 \mid (\mathtt{st}, V)] - \Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 1 \mid (\mathtt{st}, V)]) \\
&\quad \cdot \Pr[\mathtt{Sim}_1(F, W) = (\mathtt{st}, V)]
\end{aligned}$$

When $(\mathtt{st}, V = ((\bot, \bot), \emptyset)$, we have $\Pr[\mathtt{Sim}_1(F, W) = (\mathtt{st}, V)] = 1 - \sum_{S\subset\{1,\ldots,n\}} |\hat{v}(S)|$. In this case, since $\mathtt{Sim}_2$ outputs a uniform bit, we have

$$\Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 0 \mid (\mathtt{st}, V)] = \Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 1 \mid (\mathtt{st}, V)] = 1/2.$$

When $(\mathtt{st}, V) = ((S, \frac{1-\mathtt{sign}(\hat{v}(S))}{2}), \{x_i\}_{i\in S})$ for some $S \subset \{1, \ldots, n\}$, we have $\Pr[\mathtt{Sim}_1(F, W) = (\mathtt{st}, V)] = |\hat{v}(S)|$. In this case, $\mathtt{Sim}_2(\mathtt{st}, V) = \mathtt{st}_2 \oplus \mathtt{parity}(V)$ is fixed. Therefore,

$$\begin{aligned}
&\Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 0 \mid (\mathtt{st}, V)] - \Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 1 \mid (\mathtt{st}, V)] \\
&\Pr[\mathtt{st}_2 \oplus \mathtt{parity}(V) = 0 \mid (\mathtt{st}, V)] - \Pr[\mathtt{st}_2 \oplus \mathtt{parity}(V) = 1 \mid (\mathtt{st}, V)] \\
&= (-1)^{\mathtt{st}_2 \oplus \mathtt{parity}(V)} = \mathtt{sign}(\hat{v}(S)) \cdot \chi_S(\boldsymbol{x}).
\end{aligned}$$

Thus,

$$\begin{aligned}
&\Pr[\mathtt{Sim}(\boldsymbol{x}) = 0] - \Pr[\mathtt{Sim}(\boldsymbol{x}) = 1] \\
&= \sum_{(\mathtt{st},V)} (\Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 0 \mid (\mathtt{st}, V)] - \Pr[\mathtt{Sim}_2(\mathtt{st}, V) = 1 \mid (\mathtt{st}, V)]) \\
&\quad \cdot \Pr[\mathtt{Sim}_1(F, W) = (\mathtt{st}, V)] \\
&= \sum_{S\subset\{1,\ldots,n\}} |\hat{v}(S)| \cdot \mathtt{sign}(\hat{v}(S)) \cdot \chi_S(\boldsymbol{x}) \\
&= \sum_{S\subset\{1,\ldots,n\}} \hat{v}(S) \cdot \chi_S(\boldsymbol{x}) \\
&= \Pr[F(\boldsymbol{x}) = 0] - \Pr[F(\boldsymbol{x}) = 1].
\end{aligned}$$

□

With Claim 2, it is sufficient to show that for all $f, g : \{0,1\}^n \to \{0,1\}$, there exist $\{\hat{v}(S)\}_{S\subset\{1,\ldots,n\}}$ satisfying Claim 2 for the randomized function $F(\boldsymbol{x}; \boldsymbol{r}) = f(\boldsymbol{r}) \oplus g(\boldsymbol{x} \oplus \boldsymbol{r})$.

For all $\boldsymbol{x} \in \{0,1\}^n$, let $v(\boldsymbol{x}) = \Pr[F(\boldsymbol{x}) = 0] - \Pr[F(\boldsymbol{x}) = 1]$. We set $\hat{v}(S) = \frac{1}{N} \sum_{\boldsymbol{x}\in\{0,1\}^n} v(\boldsymbol{x})\chi_S(\boldsymbol{x})$ for all $S \subset \{1, \ldots, n\}$, where $N = 2^n$. Then $\hat{v}(S)$ is in the range $[-1, 1]$. We will show that $\sum_{S\subset\{1,\ldots,n\}} |\hat{v}(S)| \leq 1$.

Observe that

$$\begin{aligned}\hat{v}(S) &= \frac{1}{N}\sum_{\boldsymbol{x}\in\{0,1\}^n} v(\boldsymbol{x})\chi_S(\boldsymbol{x}) \\ &= \frac{1}{N}\sum_{\boldsymbol{x}\in\{0,1\}^n} (\Pr[F(\boldsymbol{x})=0]-\Pr[F(\boldsymbol{x})=1])\chi_S(\boldsymbol{x}) \\ &= \frac{1}{N}\sum_{\boldsymbol{x}\in\{0,1\}^n} (\Pr[F(\boldsymbol{x})=\oplus_{i\in S}x_i]-\Pr[F(\boldsymbol{x})=1\oplus(\oplus_{i\in S}x_i)])\end{aligned}$$

Now consider a random variable $\boldsymbol{X}$ uniformly distributed over $\{0,1\}^n$. We have $\Pr[\boldsymbol{X}=\boldsymbol{x}]=1/N$. Thus

$$\begin{aligned}&\frac{1}{N}\sum_{\boldsymbol{x}\in\{0,1\}^n} (\Pr[F(\boldsymbol{x})=\oplus_{i\in S}x_i]-\Pr[F(\boldsymbol{x})=1\oplus(\oplus_{i\in S}x_i)]) \\ =&\sum_{\boldsymbol{x}\in\{0,1\}^n} \Pr[\boldsymbol{X}=\boldsymbol{x}]\cdot(\Pr[F(\boldsymbol{X})=\oplus_{i\in S}X_i \mid \boldsymbol{X}=\boldsymbol{x}]-\Pr[F(\boldsymbol{X})=1\oplus(\oplus_{i\in S}X_i)\mid \boldsymbol{X}=\boldsymbol{x}]) \\ =&\Pr[F(\boldsymbol{X})=\oplus_{i\in S}X_i]-\Pr[F(\boldsymbol{X})=1\oplus(\oplus_{i\in S}X_i)].\end{aligned}$$

Since $F(\boldsymbol{x};\boldsymbol{r})=f(\boldsymbol{r})\oplus g(\boldsymbol{x}\oplus\boldsymbol{r})$, then

$$\begin{aligned}\Pr[F(\boldsymbol{X})=\oplus_{i\in S}X_i] &= \Pr[f(\boldsymbol{R})\oplus g(\boldsymbol{X}\oplus\boldsymbol{R})=\oplus_{i\in S}X_i] \\ &= \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=g(\boldsymbol{X}\oplus\boldsymbol{R})\oplus(\oplus_{i\in S}(X_i\oplus R_i))].\end{aligned}$$

Here, we separate $\oplus_{i\in S}X_i$ to be $(\oplus_{i\in S}R_i)\oplus(\oplus_{i\in S}(X_i\oplus R_i))$.

Since $\boldsymbol{X}$ and $\boldsymbol{R}$ are independent and uniformly distributed over $\{0,1\}^n$, we have $\boldsymbol{X}\oplus\boldsymbol{R}$ and $\boldsymbol{R}$ are independent and uniformly distributed over $\{0,1\}^n$. Let $\boldsymbol{R'}=\boldsymbol{X}\oplus\boldsymbol{R}$. Then,

$$\begin{aligned}\Pr[F(\boldsymbol{X})=\oplus_{i\in S}X_i] &= \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=g(\boldsymbol{X}\oplus\boldsymbol{R})\oplus(\oplus_{i\in S}(X_i\oplus R_i))] \\ &= \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)] \\ &= \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=0]\cdot\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=0] \\ &\quad + \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=1]\cdot\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=1].\end{aligned}$$

Similarly, we have

$$\begin{aligned}\Pr[F(\boldsymbol{X})=1\oplus(\oplus_{i\in S}X_i)] = &\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=0]\cdot\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=1] \\ &+ \Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=1]\cdot\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=0].\end{aligned}$$

Thus,

$$\begin{aligned}\hat{v}(S) &= \Pr[F(\boldsymbol{X})=\oplus_{i\in S}X_i]-\Pr[F(\boldsymbol{X})=1\oplus(\oplus_{i\in S}X_i)] \\ &= (\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=0]-\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=1]) \\ &\quad\cdot(\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=0]-\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=1]).\end{aligned}$$

Recall that we want to show $\sum_{S\subset\{1,\ldots,n\}}|\hat{v}(S)|\le 1$. The above equation implies that

$$\begin{aligned}|\hat{v}(S)| \le &\frac{1}{2}\cdot(\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=0]-\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=1])^2 \\ &+\frac{1}{2}\cdot(\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=0]-\Pr[g(\boldsymbol{R'})\oplus(\oplus_{i\in S}R'_i)=1])^2.\end{aligned}$$

Thus,

$$\sum_{S\subset\{1,\ldots,n\}} |\hat{v}(S)| \le \frac{1}{2} \sum_{S\subset\{1,\ldots,n\}} (\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1])^2 + \frac{1}{2} \sum_{S\subset\{1,\ldots,n\}} (\Pr[g(\boldsymbol{R}') \oplus (\oplus_{i\in S} R'_i) = 0] - \Pr[g(\boldsymbol{R}') \oplus (\oplus_{i\in S} R'_i) = 1])^2.$$

It is sufficient to prove the following claim.

Claim 3 *For all function $f : \{0,1\}^n \to \{0,1\}$,*

$$\sum_{S\subset\{1,\ldots,n\}} (\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1])^2 = 1,$$

where $\boldsymbol{R}$ is uniformly distributed over $\{0,1\}^n$.

Proof (Proof of Claim 3). Observe that

$$\begin{aligned} &\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1] \\ &= \frac{1}{N} \sum_{\boldsymbol{r}\in\{0,1\}^n} (\Pr[f(\boldsymbol{r}) \oplus (\oplus_{i\in S} r_i) = 0] - \Pr[f(\boldsymbol{r}) \oplus (\oplus_{i\in S} r_i) = 1]). \end{aligned}$$

Note that given $\boldsymbol{r}$, $f(\boldsymbol{r}) \oplus (\oplus_{i\in S} r_i)$ is fixed. Then,

$$\begin{aligned} &\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1] \\ &= \frac{1}{N} \sum_{\boldsymbol{r}\in\{0,1\}^n} (-1)^{f(\boldsymbol{r})} \cdot \chi_S(\boldsymbol{r}). \end{aligned}$$

We have

$$\begin{aligned} &\sum_{S\subset\{1,\ldots,n\}} (\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1])^2 \\ &= \frac{1}{N^2} \sum_{S\subset\{1,\ldots,n\}} \left(\sum_{\boldsymbol{r}\in\{0,1\}^n} (-1)^{f(\boldsymbol{r})} \cdot \chi_S(\boldsymbol{r}) \right)^2 \\ &= \frac{1}{N^2} \sum_{S\subset\{1,\ldots,n\}} \sum_{\boldsymbol{r}\in\{0,1\}^n} \sum_{\boldsymbol{r}'\in\{0,1\}^n} (-1)^{f(\boldsymbol{r})\oplus f(\boldsymbol{r}')} \cdot \chi_S(\boldsymbol{r} \oplus \boldsymbol{r}') \\ &= \frac{1}{N^2} \sum_{\boldsymbol{r}\in\{0,1\}^n} \sum_{\boldsymbol{r}'\in\{0,1\}^n} (-1)^{f(\boldsymbol{r})\oplus f(\boldsymbol{r}')} \cdot \left(\sum_{S\subset\{1,\ldots,n\}} \chi_S(\boldsymbol{r} \oplus \boldsymbol{r}') \right). \end{aligned}$$

Note that when $\boldsymbol{r} \neq \boldsymbol{r}'$, we always have $\sum_{S\subset\{1,\ldots,n\}} \chi_S(\boldsymbol{r}\oplus\boldsymbol{r}') = 0$. And when $\boldsymbol{r} = \boldsymbol{r}'$, $\sum_{S\subset\{1,\ldots,n\}} \chi_S(\boldsymbol{r}\oplus\boldsymbol{r}') = N$. Thus,

$$\begin{aligned} &\sum_{S\subset\{1,\ldots,n\}} (\Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 0] - \Pr[f(\boldsymbol{R}) \oplus (\oplus_{i\in S} R_i) = 1])^2 \\ &= \frac{1}{N^2} \sum_{\boldsymbol{r}\in\{0,1\}^n} \sum_{\boldsymbol{r}'\in\{0,1\}^n} (-1)^{f(\boldsymbol{r})\oplus f(\boldsymbol{r}')} \cdot \left(\sum_{S\subset\{1,\ldots,n\}} \chi_S(\boldsymbol{r} \oplus \boldsymbol{r}') \right) \\ &= \frac{1}{N^2} \sum_{\boldsymbol{r}=\boldsymbol{r}'} (-1)^{f(\boldsymbol{r})\oplus f(\boldsymbol{r}')} \cdot N \\ &= \frac{1}{N^2} \sum_{\boldsymbol{r}=\boldsymbol{r}'} N \\ &= 1. \end{aligned}$$

□

Finally, with Claim 3, we have

$$\begin{aligned}\sum_{S\subset\{1,\dots,n\}}|\hat{v}(S)| &\le \frac{1}{2}\sum_{S\subset\{1,\dots,n\}}(\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=0]-\Pr[f(\boldsymbol{R})\oplus(\oplus_{i\in S}R_i)=1])^2\\ &\quad+\frac{1}{2}\sum_{S\subset\{1,\dots,n\}}(\Pr[g(\boldsymbol{R}')\oplus(\oplus_{i\in S}R'_i)=0]-\Pr[g(\boldsymbol{R}')\oplus(\oplus_{i\in S}R'_i)=1])^2\\ &=\frac{1}{2}+\frac{1}{2}=1.\end{aligned}$$

By Claim 2, this implies that F is $(1,0)$-PT.

D.2 Proof of Theorem 9

In this section, we formally prove Theorem 9. Let $\tilde{C}$ denote the input circuit, $(\hat{I},\hat{C},\hat{O})$ denote the construction after the first step of [GIM+16], and (I,C,O) denote the final construction of [GIM+16].

The correctness follows from the construction. In the following, we show that the input encoder I and the output decoder O are trivially parity-tolerant, and C achieves parity-to-probing security.

Trivial Parity Tolerance. Recall the small-bias encoding scheme.

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

Notice that the input encoder I takes as input $\boldsymbol{x}\in\{0,1\}^{n_i}$ and outputs $\{\boldsymbol{\rho}_i,\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho_i}\}_{i=1}^{n_i}$. Consider an implementation of I which for all $i\in\{1,\dots,n_i\}$, the circuit computes $\boldsymbol{\rho}_i,\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i$ from x_i as follows:

1. The circuit uses random bits $\rho_{i,0},\rho_{i,1},\rho_{i,2},r_{i,0},r_{i,1}$. Let $\boldsymbol{\rho}_i=(\rho_{i,0},\rho_{i,1},\rho_{i,2})$ and $\boldsymbol{r}_i=(r_{i,0},r_{i,1})$.
2. The circuit computes $r_{i,0}\cdot r_{i,1}$ and then $r_{i,0}\cdot r_{i,1}\oplus x_i$. Let $\mathtt{Enc}(x_i;\boldsymbol{r}_i)=(r_{i,0},r_{i,1},r_{i,0}\cdot r_{i,1}\oplus x_i)$.
3. The circuit computes $\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i$ and outputs $(\boldsymbol{\rho}_i,\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i)$.

Now we examine that all wires can be expressed as linear combinations of $x_i,(\boldsymbol{\rho}_i,\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i)$.

- For $\boldsymbol{\rho}_i=(\rho_{i,0},\rho_{i,1},\rho_{i,2})$, these three bits are directly output. For $\boldsymbol{r}_i=(r_{i,0},r_{i,1})$, note that $\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i=(r_{i,0}\oplus\rho_{i,0},r_{i,1}\oplus\rho_{i,1},r_{i,0}\cdot r_{i,1}\oplus x_i\oplus\rho_{i,2})$. Therefore they are equal to the first two bits of $\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i$ XOR with $(\rho_{i,0},\rho_{i,1})$.
- For $r_{i,0}\cdot r_{i,1}$, it is equal to the last bit of $\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i$ XOR with $\rho_{i,2}$ and x_i. Similarly, for $r_{i,0}\cdot r_{i,1}\oplus x_i$, it is equal to the last bit of $\mathtt{Enc}(x_i)\oplus\boldsymbol{\rho}_i$ XOR with $\rho_{i,2}$.

As for the output decoder O, it takes as input $\{\boldsymbol{\rho}'_i,\mathtt{Enc}(y_i)\oplus\boldsymbol{r}'_i\}_{i=1}^{n_i}$ and outputs $\boldsymbol{y}$. Consider an implementation of O which for all $i\in\{1,\dots,n_o\}$, the circuit computes y_i from $\boldsymbol{\rho}'_i,\mathtt{Enc}(y_i)\oplus\boldsymbol{\rho}'_i$.

1. Assume that the random bits used in $\mathtt{Enc}(y_i)$ are $\boldsymbol{r}'_i=(r'_{i,0},r'_{i,1})$. Then $\mathtt{Enc}(y_i)\oplus\boldsymbol{\rho}'_i$ is in the form of

$$(r'_{i,0}\oplus\rho'_{i,0},r'_{i,1}\oplus\rho'_{i,1},r'_{i,0}\cdot r'_{i,1}\oplus y_i\oplus\rho'_{i,2}).$$

   The circuit first computes $\mathtt{Enc}(y_i;\boldsymbol{r}'_i)=(r'_{i,0},r'_{i,1},r'_{i,0}\cdot r'_{i,1}\oplus y_i)$ by XORing $\boldsymbol{\rho}'_i$ and $\mathtt{Enc}(y_i)\oplus\boldsymbol{\rho}'_i$.
2. The circuit computes $r'_{i,0}\cdot r'_{i,1}$ and then computes $y_i=(r'_{i,0}\cdot r'_{i,1})\oplus(r'_{i,0}\cdot r'_{i,1}\oplus y_i)$.

Following a similar analysis, all wires can be expressed as linear combinations of $y_i,(\boldsymbol{\rho}'_i,\mathtt{Enc}(y_i)\oplus\boldsymbol{\rho}'_i)$.

Parity-to-Probing Security. We will construct a simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ required in Definition 4. $\mathtt{Sim}_1$ works as follows.

1. $\mathtt{Sim}_1$ receives a set W of wires in C. Let $\tau(\hat{C}, \boldsymbol{x})$ denote the wire values in $\hat{C}$ and $\boldsymbol{R}$ be a random string of size $|\tau(\hat{C}, \boldsymbol{x})|$. Recall that each wire in C is either a function on $\boldsymbol{R}$ or a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Thus, we may separate W to two parts W_L and W_R, where each wire in W_L is a function on $\boldsymbol{R}$ and each wire in W_R is a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Then $\mathtt{parity}(W) = \mathtt{parity}(W_L) \oplus \mathtt{parity}(W_R) = f(\boldsymbol{R}) \oplus g(\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R})$ for some functions f, g. Consider the randomized function F that takes as input $\tau(\hat{C}, \boldsymbol{x})$ and outputs $\mathtt{parity}(W)$. By Lemma 5, F is $(1, 0)$-PT. Let $\mathtt{Sim}'$ be the simulator in Definition 9.
2. $\mathtt{Sim}_1$ invokes $\mathtt{Sim}'_1$ on the output wire of F and receives $(\mathtt{st}', \hat{V})$, where $\hat{V}$ is a set of wires in $\hat{C}$. Let $\boldsymbol{w} = (w_1, \ldots, w_{|\tilde{C}|})$ be all wire values in $\tilde{C}$. Then $\tau(\hat{C}, \boldsymbol{x})$ is in the form of $\{\mathtt{Enc}(w_i)\}_{i=1}^{|\tilde{C}|}$. For all $i \in \{1, \ldots, |\tilde{C}|\}$, Let $\hat{V}_i \subset \hat{V}$ be the set of wires from $\mathtt{Enc}(w_i)$.
   - If there exists k indices $i \in \{1, \ldots, |\tilde{C}|\}$ such that $\hat{V}_i \neq \emptyset$, then $\mathtt{Sim}_1$ samples a random bit u and sets
   $$(\mathtt{st}, V) = ((\mathtt{st}', u, \emptyset), \emptyset).$$
   - Otherwise, let $V = \{w_i \mid \hat{V}_i \neq \emptyset\}$. Note that $|V| \leq k$. $\mathtt{Sim}_1$ sets
   $$(\mathtt{st}, V) = ((\mathtt{st}', 0, \hat{V}), V).$$

   $\mathtt{Sim}_1$ outputs $(\mathtt{st}, V)$.

   As for $\mathtt{Sim}_2$,

- If $V = \emptyset$, $\mathtt{Sim}_2$ sets $\mathtt{parity}(\hat{V}) = u$. Then $\mathtt{Sim}_2$ outputs the result of $\mathtt{Sim}'_2(\mathtt{st}', \mathtt{parity}(\hat{V}))$.
- Otherwise, $\mathtt{Sim}_2$ receives $\mathsf{val}(V)$ and computes $\mathtt{Enc}(w_i)$ for all $w_i \in V$. Then $\mathtt{Sim}_2$ computes $\mathtt{parity}(\hat{V})$ from $\{\mathtt{Enc}(w_i) \mid w_i \in V\}$ and outputs the result of $\mathtt{Sim}'_2(\mathtt{st}', \mathtt{parity}(\hat{V}))$.

Claim 4 *For all input $\boldsymbol{x}$ and for all W, the following two distributions are statistically close with error $\epsilon = 2^{-k-1}$:*

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

Proof. 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 $\mathtt{parity}(W)$.
3. Finally, we output
$$(\mathtt{parity}(W), \boldsymbol{y})$$

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

Hybrid$_1$: In this hybrid, we change the way of computing the wire values in C. Concretely

1'. Given the input $\boldsymbol{x}$, we compute $\boldsymbol{y} = \tilde{C}(\boldsymbol{x})$. For all wire values $(w_1, \ldots, w_{|\tilde{C}|})$ in $\tilde{C}$, we compute $\{\mathtt{Enc}(w_i)\}_{i=1}^{|\tilde{C}|}$. Let $\tau(\hat{C}, \boldsymbol{x}) = \{\mathtt{Enc}(w_i)\}_{i=1}^{|\tilde{C}|}$. Then we sample a random string $\boldsymbol{R}$ of size $|\tau(\hat{C}, \boldsymbol{x})|$ and compute $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Finally, from $\boldsymbol{R}$ and $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$, we compute all wire values in C.

According to the construction, the wire values in C as well as the output $\boldsymbol{y}$ have the same distribution in both **Hybrid**$_0$ and **Hybrid**$_1$. Thus, the output distribution of **Hybrid**$_1$ is identical to that of **Hybrid**$_0$.

Hybrid$_2$: In this hybrid, we utilize $\tau(\hat{C}, \boldsymbol{x})$ and $\mathtt{Sim}'$ to compute $\mathtt{parity}(W)$. Concretely

2". We separate W to two parts W_L and W_R, where each wire in W_L is a function on $\boldsymbol{R}$ and each wire in W_R is a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Then $\mathtt{parity}(W) = \mathtt{parity}(W_L) \oplus \mathtt{parity}(W_R) = f(\boldsymbol{R}) \oplus g(\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R})$ for some functions f, g. Let F be the randomized function that takes as input $\tau(\hat{C}, \boldsymbol{x})$ and outputs $\mathtt{parity}(W)$. By Lemma 5, F is $(1, 0)$-PT. Let $\mathtt{Sim}'$ be the simulator in Definition 9.
We invoke $\mathtt{Sim}'_1$ on the output of F and receive $(\mathtt{st}', \hat{V})$ where $\hat{V}$ is a set of wires in $\hat{C}$. We compute $\mathtt{parity}(\hat{V})$ by using $\tau(\hat{C}, \boldsymbol{x})$ Then, we invoke $\mathtt{Sim}'_2$ on $(\mathtt{st}', \mathtt{parity}(\hat{V}))$ and view the output as $\mathtt{parity}(W)$.

According to Lemma 5, for all fixed $\tau(\hat{C}, \boldsymbol{x})$, the output of $\mathtt{Sim}'$ is identically distributed to the output of F, which is $\mathtt{parity}(W)$ in $\mathbf{Hybrid}_1$. Thus, the output distribution of $\mathbf{Hybrid}_2$ is identical to that of $\mathbf{Hybrid}_1$.

Note that, $\boldsymbol{R}$ and $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$ are not needed. We do not need to compute them in Step 1.

$\mathbf{Hybrid}_3$: In this hybrid, we utilize the wire values in $\tilde{C}$ to compute $\mathtt{parity}(\hat{V})$. Concretely,

2"'. We run Step 2 in $\mathbf{Hybrid}_2$ till receiving $(\mathtt{st}', \hat{V})$ from $\mathtt{Sim}'_1$. For all $i \in \{1, \ldots, |\tilde{C}|\}$, let $\hat{V}_i \subset \hat{V}$ be the wires from $\mathtt{Enc}(w_i)$. Let $V = \{w_i \mid \hat{V}_i \neq \emptyset\}$. If $|V| \geq k$, then we set $\mathtt{parity}(\hat{V})$ to be a random bit. Otherwise, we compute $\mathtt{parity}(\hat{V})$ from $\tau(\hat{C}, \boldsymbol{x})$. We continue Step 2 in $\mathbf{Hybrid}_2$.

The only difference between $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ is that we set $\mathtt{parity}(\hat{V})$ to be a random bit when $|V| \geq k$ in $\mathbf{Hybrid}_3$ rather than computing it from $\tau(\hat{C}, \boldsymbol{x})$ as in $\mathbf{Hybrid}_2$. Recall that $(\mathtt{Enc}, \mathtt{Dec})$ is a small-bias encoding scheme with $\epsilon = 1/2$. Then for all $\hat{V}_i \neq \emptyset$, the bias of $\mathtt{parity}(\hat{V}_i)$ is at most $1/2$. We borrow the following claim from [GIM+16] (Claim 2 in Appendix A.1 in [GIM+16]).

Claim 5 ([GIM+16]) *Let $X_1, \ldots, X_n$ be independent random variables such that for all $j \in \{1, \ldots, n\}$, the distribution of X_j is ϵ_j-biased. Then the random variable $\oplus_{j=1}^n X_j$ is $\prod_{j=1}^n \epsilon_j$-biased.*

Thus, when $|V| \geq k$, $\mathtt{parity}(\hat{V}) = \oplus_{i=1}^{|\tilde{C}|} \mathtt{parity}(\hat{V}_i)$ is at most 2^{-k}-biased, which implies that the statistical distance between $\mathtt{parity}(\hat{V})$ and a uniform bit is at most 2^{-k-1}. Therefore, the output distribution of $\mathbf{Hybrid}_3$ is statistically close to $\mathbf{Hybrid}_2$ with error 2^{-k-1}.

$\mathbf{Hybrid}_4$: In this hybrid, we change the order of the generation process.

1. Given the input $\boldsymbol{x}$, we compute $\boldsymbol{y} = \tilde{C}(\boldsymbol{x})$.
2. We separate W to two parts W_L and W_R, where each wire in W_L is a function on $\boldsymbol{R}$ and each wire in W_R is a function on $\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R}$. Then $\mathtt{parity}(W) = \mathtt{parity}(W_L) \oplus \mathtt{parity}(W_R) = f(\boldsymbol{R}) \oplus g(\tau(\hat{C}, \boldsymbol{x}) \oplus \boldsymbol{R})$ for some functions f, g. Let F be the randomized function that takes as input $\tau(\hat{C}, \boldsymbol{x})$ and outputs $\mathtt{parity}(W)$. By Lemma 5, F is $(1, 0)$-PT. Let $\mathtt{Sim}'$ be the simulator in Definition 9.
   We invoke $\mathtt{Sim}'_1$ on the output of F and receive $(\mathtt{st}', \hat{V})$ where $\hat{V}$ is a set of wires in $\hat{C}$. For all $i \in \{1, \ldots, |\tilde{C}|\}$, let $\hat{V}_i \subset \hat{V}$ be the wires from $\mathtt{Enc}(w_i)$.
   - If there exists k indices $i \in \{1, \ldots, |\tilde{C}|\}$ such that $\hat{V}_i \neq \emptyset$, then we sample a random bit u and sets
   $$(\mathtt{st}, V) = ((\mathtt{st}', u, \emptyset), \emptyset).$$
   - Otherwise, let $V = \{w_i \mid \hat{V}_i \neq \emptyset\}$. Note that $|V| \leq k$. We set
   $$(\mathtt{st}, V) = ((\mathtt{st}', 0, \hat{V}), V).$$
3. From the wire values in $\tilde{C}$, we obtain $\mathsf{val}(V)$.
4. If $V = \emptyset$, it means that there exists k indices $i \in \{1, \ldots, |\tilde{C}|\}$ such that $\hat{V}_i \neq \emptyset$. We set $\mathtt{parity}(\hat{V}) = u$ and invoke $\mathtt{Sim}'_2$ on $(\mathtt{st}', \mathtt{parity}(\hat{V}))$. We view the output as $\mathtt{parity}(W)$.
5. If $V \neq \emptyset$, then $|V| \leq k$. We compute $\mathtt{Enc}(w_i)$ for all $w_i \in V$ and compute $\mathtt{parity}(\hat{V})$. Then we invoke $\mathtt{Sim}'_2$ on $(\mathtt{st}', \mathtt{parity}(\hat{V}))$ and view the output as $\mathtt{parity}(W)$.
6. Finally we output
   $$(\mathtt{parity}(W), \boldsymbol{y}).$$

Compared with $\mathbf{Hybrid}_3$, we only change the order of generating $\mathtt{Enc}(w_i)$ and we only generate $\mathtt{Enc}(w_i)$ for $w_i \in V$. This does not change the output distribution. Thus, the output distribution of $\mathbf{Hybrid}_4$ is identical to that of $\mathbf{Hybrid}_3$.

Note that Step 2 corresponds to $\mathtt{Sim}_1$ and Step 4,5 correspond to $\mathtt{Sim}_2$. Thus, the output distribution of $\mathbf{Hybrid}_4$ is identical to

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

D.3 Separating 2-Parity Tolerance from 1-Parity Tolerance

It is a natural question to ask whether a 1-parity-tolerant circuit/function is automatically secure against more than 1 parity attack. While this is true for parity resilience, in this section, we show that it is not the case for parity tolerance for either circuits or randomized functions.

We first give a counter example of a randomized function which is $(1,0)$-PT but not $(2,\epsilon)$-PT for all $\epsilon < 1/64$. Then we construct a $(1,0)$-parity-tolerant circuit from this counter example and showing that this circuit is not $(2,\epsilon)$-parity-tolerant for all $\epsilon < 1/64$.

Counter Example for Randomized Function. Consider $n = 3$ and we define the randomized function $F : \{0,1\}^3 \to \{0,1\}^2$ to be

$$F(\boldsymbol{x};\boldsymbol{r}) = ((r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1), (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3)),$$

where $\boldsymbol{r} = (r_1, r_2, r_3)$ is uniformly distributed over $\{0,1\}^3$. Then F is $(1,0)$-PT following from the parity simulation lemma 5.

We show that F is *not* $(2,\epsilon)$-PT for all $\epsilon < 1/64$. Suppose there exists such a simulator $\mathtt{Sim}$ which can simulate the output of F within 2 parity queries to the input $\boldsymbol{x}$. Then there is also a simulator $\mathtt{Sim}'$ which can simulate the multiplication of the two output bits of F within 2 parity queries to the input $\boldsymbol{x}$. Note that the multiplication of the two output bits of F is $v = (r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1) \cdot (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3) = (r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1) \cdot x_1x_2x_3$.

- If $x_1x_2x_3 = 0$, v is always 0.
- Otherwise, with probability $1/8$, v is equal to 1.

Intuitively, to simulate v, the simulator $\mathtt{Sim}'$ has to learn $x_1x_2x_3$, which is impossible within two parity queries to the input $\boldsymbol{x}$.

To formally argue this observation, we first describe $\mathtt{Sim}'$ from $\mathtt{Sim}$:

- $\mathtt{Sim}'_1$ invokes $\mathtt{Sim}_1$ with input $W_1 = \{(r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1)\}, W_2 = \{(x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3)\}$. Then $\mathtt{Sim}'_1$ receives $(\mathtt{st}, V_1, V_2)$ from $\mathtt{Sim}_1$ and outputs them.
- $\mathtt{Sim}'_2$ receives $(\mathtt{st}, \mathtt{parity}(V_1), \mathtt{parity}(V_2))$ and feeds them to $\mathtt{Sim}_2$. Then $\mathtt{Sim}'_2$ receives b_1, b_2 from $\mathtt{Sim}_2$ and outputs $b_1 \cdot b_2$.

If (b_1, b_2) is statistically close to $((r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1), (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3))$ with error ϵ, then $b_1 \cdot b_2$ is statistically close to $v = (r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1) \cdot (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3)$ with error at most ϵ.

On one hand, when $\boldsymbol{x} = 111$, with probability $1/8$, $v = 1$. This means that $\Pr[\mathtt{Sim}'(111) = 1] \geq 1/8 - \epsilon$. On the other hand, for all $\boldsymbol{x} \neq 111$, v is always equal to 0. This means that $\Pr[\mathtt{Sim}'(\boldsymbol{x}) = 1] \leq \epsilon$ for all $\boldsymbol{x} \neq 111$.

Since the output of S'_1 is independent of $\boldsymbol{x}$, we have

$$\begin{aligned}
&\Pr[\mathtt{Sim}'(111) = 1] = \Pr[\mathtt{Sim}'(\boldsymbol{X}) = 1 \mid \boldsymbol{X} = 111] \\
&= \sum_{(\mathtt{st},V_1,V_2)} \Pr[\mathtt{Sim}'_2(\mathtt{st}, \mathtt{parity}(V_1), \mathtt{parity}(V_2)) = 1 \mid (\mathtt{st}, V_1, V_2) \ \&\ \boldsymbol{X} = 111] \\
&\quad \cdot \Pr[\mathtt{Sim}'_1(\cdot) = (\mathtt{st}, V_1, V_2)].
\end{aligned}$$

Note that, for all V_1, V_2, there exists $\boldsymbol{x}' \neq 111$ such that when $\boldsymbol{x} = \boldsymbol{x}'$, $\mathtt{parity}(V_1), \mathtt{parity}(V_2)$ are identical with those when $\boldsymbol{x} = 111$. This means that

$$\begin{aligned}
&\Pr[\mathtt{Sim}'_2(\mathtt{st}, \mathtt{parity}(V_1), \mathtt{parity}(V_2)) = 1 \mid (\mathtt{st}, V_1, V_2) \ \&\ \boldsymbol{X} = 111] \\
&\leq \sum_{\boldsymbol{x}' \neq 111} \Pr[\mathtt{Sim}'_2(\mathtt{st}, \mathtt{parity}(V_1), \mathtt{parity}(V_2)) = 1 \mid (\mathtt{st}, V_1, V_2) \ \&\ \boldsymbol{X} = \boldsymbol{x}'].
\end{aligned}$$

Thus,

$$\begin{aligned}
&\Pr[\mathtt{Sim}'(111)=1] = \Pr[\mathtt{Sim}'(\boldsymbol{X})=1 \mid \boldsymbol{X}=111] \\
=& \sum_{(\mathtt{st},V_1,V_2)} \Pr[\mathtt{Sim}'_2(\mathtt{st},\mathtt{parity}(V_1),\mathtt{parity}(V_2))=1 \mid (\mathtt{st},V_1,V_2)\ \&\ \boldsymbol{X}=111] \\
&\cdot \Pr[\mathtt{Sim}'_1(\cdot)=(\mathtt{st},V_1,V_2)] \\
\leq& \sum_{(\mathtt{st},V_1,V_2)} \sum_{\boldsymbol{x}'\neq 111} \Pr[\mathtt{Sim}'_2(\mathtt{st},\mathtt{parity}(V_1),\mathtt{parity}(V_2))=1 \mid (\mathtt{st},V_1,V_2)\ \&\ \boldsymbol{X}=\boldsymbol{x}'] \\
&\cdot \Pr[\mathtt{Sim}'_1(\cdot)=(\mathtt{st},V_1,V_2)] \\
=& \sum_{\boldsymbol{x}'\neq 111} \Pr[\mathtt{Sim}'(\boldsymbol{X})=1 \mid \boldsymbol{X}=\boldsymbol{x}'] \\
=& \sum_{\boldsymbol{x}'\neq 111} \Pr[\mathtt{Sim}'(\boldsymbol{x}')=1]
\end{aligned}$$

Since $\Pr[\mathtt{Sim}'(111)=1] \geq 1/8-\epsilon$ and $\Pr[\mathtt{Sim}'(\boldsymbol{x}')=1] \leq \epsilon$ for all $\boldsymbol{x}' \neq 111$, we have $1/8-\epsilon \leq 7\epsilon$. This implies that $\epsilon \geq 1/64$. Thus, for all $\epsilon < 1/64$, such a simulator $\mathtt{Sim}$ does not exists, which implies that F is not $(2,\epsilon)$-PT.

Counter Example for Parity-Tolerant Circuits. Consider the following circuit C:

1. C takes as input x_1, x_2, x_3 and uses random bits r_1, r_2, r_3.
2. C computes $x_1 \oplus r_1$, $x_2 \oplus r_2$, $x_3 \oplus r_3$ and then computes $(x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3)$.
3. C computes $r_1 \oplus 1$, $r_2 \oplus 1$, $r_3 \oplus 1$ and then computes $(r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1)$.
4. C computes and outputs $x_1 \oplus x_2 \oplus x_3$.

Note that the output of C is a linear combination of its input. Thus, any parity query to its input and output is equivalent to some parity query only to its input.

We first show that C is $(1,0)$-parity-tolerant. Let $\boldsymbol{x} = (x_1, x_2, x_3)$ and $\boldsymbol{r} = (r_1, r_2, r_3)$. Observe that every wire in C can be expressed as $f(\boldsymbol{r}) \oplus g(\boldsymbol{x} \oplus \boldsymbol{r})$ for some functions f, g. Thus, the parity of any subset of wires of C is also in this form. Following from Lemma 5, C is $(1,0)$-parity-tolerant.

Now we show that C is not $(2,\epsilon)$-parity-tolerant for all $\epsilon < 1/64$. This follows from the fact that $(r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1), (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3)$ are inner wires of C. Since we have shown that the randomized function

$$F(\boldsymbol{x};\boldsymbol{r}) = ((r_1 \oplus 1)(r_2 \oplus 1)(r_3 \oplus 1), (x_1 \oplus r_1)(x_2 \oplus r_2)(x_3 \oplus r_3))$$

is not $(2,\epsilon)$-PT for all $\epsilon < 1/64$, it implies that C is not $(2,\epsilon)$-parity-tolerant for all $\epsilon < 1/64$.

Remark 3. We note that the counter example F also implies that the parity simulation lemma 5 does not extend to 2 parities. Thus the above proving technique fails to show that the construction in [GIM+16] achieves $(2,k,\epsilon)$-parity-to-probing security. Unfortunately, we do not know how to disprove this argument either.

E Proof of Claim 1

Suppose $\mathtt{Enc}$ admits a trivially parity-tolerant implementation C and $\mathtt{Enc}$ is ϵ-parity-resilient.

Let $w_1, \ldots, w_{|C|}$ denote the wires in C. Suppose $x \in \{0,1\}$ is the input of $\mathtt{Enc}$ and $\boldsymbol{y} = \mathtt{Enc}(x)$ is the output. Since C is trivially parity-tolerant, for all $i \in \{1, \ldots, |C|\}$, there exist $a_i \in \{0,1\}, \boldsymbol{c}_i \in \{0,1\}^n$ such that $w_i = a_i \cdot x \oplus \boldsymbol{c}_i \cdot \boldsymbol{y}$. Then for each output wire w_i, $a_i = 0$. Otherwise, $x = w_i \oplus \boldsymbol{c}_j \cdot \boldsymbol{y}$ is a linear combination of the output $\boldsymbol{y}$, which again contradicts the ϵ-parity-resilience of $\mathtt{Enc}$.

For all $\mathtt{XOR}$ gate in C with input wires w_i, w_j and output wire w_k, we have

$$(a_i \oplus a_j \oplus a_k) \cdot x = (\boldsymbol{c}_i \oplus \boldsymbol{c}_j \oplus \boldsymbol{c}_k) \cdot \boldsymbol{y}.$$

We claim that $a_i \oplus a_j \oplus a_k = 0$. Otherwise, the input x can be expressed as a linear combination of the output $\boldsymbol{y}$, contradicting the ϵ-parity-resilience of $\texttt{Enc}$. Therefore, we always have $\boldsymbol{c}_i \cdot \boldsymbol{y} \oplus \boldsymbol{c}_j \cdot \boldsymbol{y} = \boldsymbol{c}_k \cdot \boldsymbol{y}$.

Now we show that for all $\boldsymbol{r}$, there exists an $\texttt{AND}$ gate in C with input wires w_i, w_j and output wire w_k such that $(\boldsymbol{c}_i \cdot \boldsymbol{y}) \cdot (\boldsymbol{c}_j \cdot \boldsymbol{y}) \neq (\boldsymbol{c}_k \cdot \boldsymbol{y})$. Otherwise, let $w'_i = \boldsymbol{c}_i \cdot \boldsymbol{y}$ for all $i \in \{1, \ldots, |C|\}$. Then

- For all $\texttt{XOR}$ gate in C with input wires w_i, w_j and output wire w_k, we have
$$w'_i \oplus w'_j = w'_k.$$
- For all $\texttt{AND}$ gate in C with input wires w_i, w_j and output wire w_k, by assumption, we have
$$w'_i \cdot w'_j = w'_k.$$

Then $(w'_1, \ldots, w'_{|C|})$ is a valid transcript for $\texttt{Enc}(0)$ for some $\boldsymbol{r}'$. Note that for each output wire w_i, we have $a_i = 0$, implying that $w_i = w'_i$. Therefore $\boldsymbol{y} = \texttt{Enc}(1; \boldsymbol{r}) = \texttt{Enc}(0; \boldsymbol{r}')$. This contradicts the assumption that $\texttt{Enc}$ is an encoding function.

Thus, when $\boldsymbol{r}$ is randomly sampled, there exists an $\texttt{AND}$ gate in C with input wires w_i, w_j and output wire w_k such that when $x = 1$, $(\boldsymbol{c}_i \cdot \boldsymbol{y}) \cdot (\boldsymbol{c}_j \cdot \boldsymbol{y}) \neq (\boldsymbol{c}_k \cdot \boldsymbol{y})$ with probability at least $1/|C|$. On the other hand, when $x = 0$, we always have $(\boldsymbol{c}_i \cdot \boldsymbol{y}) \cdot (\boldsymbol{c}_j \cdot \boldsymbol{y}) = (\boldsymbol{c}_k \cdot \boldsymbol{y})$. Therefore the statistical distance between $(\boldsymbol{c}_i \cdot \boldsymbol{y}, \boldsymbol{c}_j \cdot \boldsymbol{y}, \boldsymbol{c}_k \cdot \boldsymbol{y})$ when the input $x = 0$ and that when $x = 1$ is at least $1/|C|$.

However, since $\texttt{Enc}$ is ϵ-parity-resilient, by the XOR lemma [Gol11], $(\boldsymbol{c}_i \cdot \boldsymbol{y}, \boldsymbol{c}_j \cdot \boldsymbol{y}, \boldsymbol{c}_k \cdot \boldsymbol{y})$ when $x = 0$ is statistically close to that when $x = 1$ with distance $2^{3/2}\epsilon < 1/|C|$, which leads to a contradiction.

F Proof of Lemma 1

The correctness of $C^\star$ follows from the description of the construction. In the following, we show the existence of the simulator $\texttt{Sim} = (\texttt{Sim}_1, \texttt{Sim}_2)$. We first analyse the wires in $C^\star$:

- The first part of $C^\star$ encodes the input $\boldsymbol{x}$. The wires include $\boldsymbol{x}, \boldsymbol{r}, \hat{\boldsymbol{x}}$ and all inner wires in $\texttt{Enc}(\boldsymbol{x} \| \boldsymbol{0}; \boldsymbol{r})$. By the property of $\texttt{Enc}$, each inner wire of $\texttt{Enc}$ is a linear combination of its input and random tape $\boldsymbol{x}, \boldsymbol{r}$. Thus, if an inner wire of $\texttt{Enc}$ is in the set W, we may replace it by the corresponding wires in $\boldsymbol{x}, \boldsymbol{r}$ so that the parity of W does not change after the replacement. Also by the property of $\texttt{Enc}$, $\boldsymbol{r}$ is a part of its output $\hat{\boldsymbol{x}}$. Therefore, we only need to consider $\boldsymbol{x}, \hat{\boldsymbol{x}}$.
- The second part of $C^\star$ computes the function f'. The wires include $\hat{\boldsymbol{x}}, \boldsymbol{r}', \hat{\boldsymbol{y}}$ and all inner wires in I, C, O. Since I, O are trivially parity-tolerant, each inner wire of I, O is a linear combination of the input and output wires of I, O. Thus, if an inner wire of I, O is in the set W, we may replace it by the corresponding input wires and output wires of I, O so that the parity of W does not change after the replacement. In this way, we only need to consider $\hat{\boldsymbol{x}}, \boldsymbol{r}', \hat{\boldsymbol{y}}$ and all inner wires in C.
  Recall that $\boldsymbol{r}'$ is used as the random tape for $\texttt{Enc}$. By the property of $\texttt{Enc}$, $\boldsymbol{r}'$ is a part of its output $\hat{\boldsymbol{y}}$. Thus, we only need to consider $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$ and all inner wires in C.
- The third part of $C^\star$ decodes the output $\hat{\boldsymbol{y}}$. The wires include $\hat{\boldsymbol{y}}, \boldsymbol{y}$ and all inner wires in $\texttt{Dec}(\hat{\boldsymbol{y}})$. By the property of $\texttt{Dec}$, each inner wire of $\texttt{Dec}$ is a linear combination of its input $\hat{\boldsymbol{y}}$. Thus, if an inner wire of $\texttt{Dec}$ is in the set W, we may replace it by the corresponding wires in $\hat{\boldsymbol{y}}$ so that the parity of W does not change after the replacement. In this way, we only need to consider $\hat{\boldsymbol{y}}, \boldsymbol{y}$.

In summary, it is sufficient to only consider wires in $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$ and all wires in C.

Construction of $\texttt{Sim}$. Suppose $W_1, \ldots, W_t$ are the input sets of $\texttt{Sim}$. For each set W_i, we separate it into two disjoint parts $W_i = W_{i,1} \bigcup W_{i,2}$ where $W_{i,1}$ only contain wires in $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$, and $W_{i,2}$ only contain wires in C. Our strategy is to first simulate the joint distribution of $\{\texttt{parity}(W_{i,2})\}_{i=1}^t$. Then, we decide the sets $V_1, \ldots, V_t$, and simulate the joint distribution of $\{\texttt{parity}(W_{i,1})\}_{i=1}^t$ given $\{\texttt{parity}(W_{i,2})\}_{i=1}^t$ and $\{\texttt{parity}(V_i)\}_{i=1}^t$.

Since (I, C, O) is a (t, k, ϵ)-parity-to-probing circuit, let $\texttt{Sim}'$ denote the simulator in Definition 4. We construct $\texttt{Sim}_1$ as follows.

1. $\mathtt{Sim}_1$ invokes $\mathtt{Sim}'_1$ with input $W_{1,2}, \ldots, W_{t,2}$ and receives $(\mathtt{st}', V')$ from $\mathtt{Sim}'_1$. Here V' is a set of k wires in $\tilde{C}$.
2. Since $\tilde{C}$ is a k-probing-tolerant implementation of f, let $\widetilde{\mathtt{Sim}}$ denote the simulator in Definition 2. $\mathtt{Sim}_1$ invokes $\widetilde{\mathtt{Sim}}_1$ with input V' and receives $(\widetilde{\mathtt{st}}, \tilde{V})$ from $\widetilde{\mathtt{Sim}}_1$. Here $\tilde{V}$ is a set of k input and output wires of f', which are $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$. (Recall that $\boldsymbol{r}'$ is a part of $\hat{\boldsymbol{y}}$.)
3. $\mathtt{Sim}_1$ samples k random values as the wire values in $\tilde{V}$. Then $\mathtt{Sim}_1$ invokes $\widetilde{\mathtt{Sim}}_2$ with input $(\widetilde{\mathtt{st}}, \mathsf{val}(\tilde{V}))$ and receives $\mathsf{val}(V')$ from $\widetilde{\mathtt{Sim}}_2$.
4. $\mathtt{Sim}_1$ invokes $\mathtt{Sim}'_2$ with input $(\mathtt{st}', \mathsf{val}(V'))$ and receives t bits $(b'_1, \ldots, b'_t)$.
5. So far $\mathtt{Sim}_1$ has simulated the joint distribution of $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$ when fixing the wire values in $\tilde{V}$. Let $X = \{x_1, \ldots, x_{n_i}\}$ and $Y = \{y_1, \ldots, y_{n_o}\}$. Let $J = \tilde{V} \cup X \cup Y$. Note that the random variables in J are linearly independent. This follows from the property of $\mathtt{Enc}$ where any k bits from the output of $\mathtt{Enc}$ are uniformly distributed, and that $\tilde{V}$ contains at most k wires in $\hat{\boldsymbol{x}} = \mathtt{Enc}(\boldsymbol{x}\|\mathbf{0}; \boldsymbol{r})$ and at most k wires in $\hat{\boldsymbol{y}} = \mathtt{Enc}(\boldsymbol{y}\|\mathbf{0}; \boldsymbol{r}')$.
   Let $R = \{r_1, \ldots, r_\ell\}$ and $R' = \{r'_1, \ldots, r'_\ell\}$. Recall that $\hat{\boldsymbol{x}}$ is fully determined by $\boldsymbol{x}$ and $\boldsymbol{r}$, and $\hat{\boldsymbol{y}}$ is fully determined by $\boldsymbol{y}$ and $\boldsymbol{r}'$. Let J' be a set of random variables such that $J \subset J' \subset (J \cup R \cup R')$ and J' is a basis of the span of $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$. For each variable in $J' \backslash J$, $\mathtt{Sim}_1$ assigns a random bit.
6. For all $i \in \{1, \ldots, t\}$, $\mathtt{parity}(W_{i,1})$ is a linear combination of random variables in J'. Let $\mathtt{support}_{J'}(W_{i,1})$ denote the support of $\mathtt{parity}(W_{i,1})$ in J'. We have

$$\mathtt{parity}(W_{i,1}) = \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1})).$$

   Recall that $\mathtt{Sim}_1$ has assigned values for all variables in $J' \backslash (X \cup Y)$. Then, $\mathtt{Sim}_1$ computes $b''_i = \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1}) \backslash (X \cup Y))$ and sets $\mathtt{st} = (b'_1 \oplus b''_1, \ldots, b'_t \oplus b''_t)$. For all $i \in \{1, \ldots, t\}$, $\mathtt{Sim}_1$ sets $V_i = \mathtt{support}_{J'}(W_{i,1}) \cap (X \cup Y)$.

As for $\mathtt{Sim}_2$, it takes as input $\mathtt{st} = (b'_1 \oplus b''_1, \ldots, b'_t \oplus b''_t)$ and $\mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t)$, and outputs $(b'_1 \oplus b''_1 \oplus \mathtt{parity}(V_1), \ldots, b'_t \oplus b''_t \oplus \mathtt{parity}(V_t))$.

Claim 6 *For all input $\boldsymbol{x}$ and for all $W_1, \ldots, W_t$, the following two distributions are statistically close with error ϵ:*

$$\begin{aligned} &(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), C^\star(\boldsymbol{x})) \\ \approx_\epsilon\ &(\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t)), f(\boldsymbol{x})) : (\mathtt{st}, V_1, \ldots, V_t) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t). \end{aligned}$$

Proof. Consider the following hybrids.

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

1. Given the input $\boldsymbol{x}$, we first randomly sample $\boldsymbol{r}, \boldsymbol{r}'$. Then we compute $\hat{\boldsymbol{x}} := \mathtt{Enc}(\boldsymbol{x}\|\mathbf{0}; \boldsymbol{r})$.
2. Given $\hat{\boldsymbol{x}}, \boldsymbol{r}'$, we evaluate the circuit C and obtain output $\hat{\boldsymbol{y}}$.
3. We compute $\mathtt{Dec}(\hat{\boldsymbol{y}})$ and set $\boldsymbol{y}$ to be the first n_o bits of $\mathtt{Dec}(\hat{\boldsymbol{y}})$.
4. So far we have computed $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$ and all wire values in C. We use $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$ to compute $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$.
5. We use the wire values in C to compute $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$.
6. Finally, we output

$$(\mathtt{parity}(W_{1,1}) \oplus \mathtt{parity}(W_{1,2}), \ldots, \mathtt{parity}(W_{t,1}) \oplus \mathtt{parity}(W_{t,2}), \boldsymbol{y})$$

Note that this is exactly the procedure of computing $C^\star(\boldsymbol{x})$. The output distribution is identical to $(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t), C^\star(\boldsymbol{x}))$.

Hybrid$_1$: In this hybrid, we change the way of computing $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$ by using the simulator $\mathtt{Sim}'$ in Definition 4. Concretely,

2'. Given $\hat{\boldsymbol{x}}, \boldsymbol{r}'$, we evaluate the circuit $\tilde{C}(\hat{\boldsymbol{x}}, \boldsymbol{r}')$ and obtain output $\hat{\boldsymbol{y}}$.
5'. We invoke $\mathtt{Sim}'_1$ on $W_{1,2}, \ldots, W_{t,2}$ and receive $(\mathtt{st}', V')$. Here V' is a set of k wires in $\tilde{C}$. We then invoke $\mathtt{Sim}'_2(\mathtt{st}', \mathsf{val}(V'))$ and receive $b'_1, \ldots, b'_t$. These t bits are viewed as $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$.

Since (I, C, O) is a (t, k, ϵ)-parity-to-probing circuit implementation of $\tilde{C}$, the distribution of the output in $\mathbf{Hybrid}_1$ is statistically close to the distribution of the output in $\mathbf{Hybrid}_0$ with error ϵ.

$\mathbf{Hybrid}_2$: In this hybrid, we change the way of computing $\mathsf{val}(V')$ by using the simulator $\widetilde{\mathtt{Sim}}$ in Definition 2. Concretely,

2". Given $\hat{\boldsymbol{x}}, \boldsymbol{r}'$, we compute $\hat{\boldsymbol{y}} = f'(\hat{\boldsymbol{x}}, \boldsymbol{r}')$.
5". We invoke $\mathtt{Sim}'_1$ on $W_{1,2}, \ldots, W_{t,2}$ and receive $(\mathtt{st}', V')$. Here V' is a set of k wires in $\tilde{C}$. We then invoke $\widetilde{\mathtt{Sim}_1}$ on V' and receives $(\tilde{\mathtt{st}}, \tilde{V})$. Here $\tilde{V}$ is a set of k wires of $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$. (Recall that $\boldsymbol{r}'$ is a part of $\hat{\boldsymbol{y}}$.) Then we invoke $\widetilde{\mathtt{Sim}_2}$ on $(\tilde{\mathtt{st}}, \mathsf{val}(\tilde{V}))$ and receives $\mathsf{val}(V')$. Next we invoke $\mathtt{Sim}'_2(\mathtt{st}', \mathsf{val}(V'))$ and receive $b'_1, \ldots, b'_t$. These t bits are viewed as $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$.

Since $\tilde{C}$ is a k-probing-tolerant implementation of f', the output of $\mathbf{Hybrid}_2$ is identically distributed to the output of $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$: In this hybrid, we change the way of computing $\hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$. Concretely,

1"'. Given the input $\boldsymbol{x}$, we first compute $\boldsymbol{y} = f(\boldsymbol{x})$.
2"'. We randomly sample $\boldsymbol{r}, \boldsymbol{r}'$.
3"'. We compute $\hat{\boldsymbol{x}} = \mathtt{Enc}(\boldsymbol{x} \| \mathbf{0}; \boldsymbol{r})$ and $\hat{\boldsymbol{y}} = \mathtt{Enc}(\boldsymbol{y} \| \mathbf{0}; \boldsymbol{r}')$.

Recall that f' is defined to be

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

Thus, the distributions of $\hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$ in both hybrids are identical. Therefore, the output of $\mathbf{Hybrid}_3$ is identically distributed to the output of $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_4$: In this hybrid, we further change the way of computing $\hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$. Consider the following generation process.

1. Given the input $\boldsymbol{x}$, we compute $\boldsymbol{y} = f(\boldsymbol{x})$.
2. We run Step 5 in $\mathbf{Hybrid}_3$ with the modification that the wire values in $\tilde{V}$ are sampled uniformly: We invoke $\mathtt{Sim}'_1$ on $W_{1,2}, \ldots, W_{t,2}$ and receive $(\mathtt{st}', V')$. Here V' is a set of k wires in $\tilde{C}$. We then invoke $\widetilde{\mathtt{Sim}_1}$ on V' and receives $(\tilde{\mathtt{st}}, \tilde{V})$. Here $\tilde{V}$ is a set of k wires of $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$. (Recall that $\boldsymbol{r}'$ is a part of $\hat{\boldsymbol{y}}$.) Now we samples random bits as wire values in $\tilde{V}$. Then we invoke $\widetilde{\mathtt{Sim}_2}$ on $(\tilde{\mathtt{st}}, \mathsf{val}(\tilde{V}))$ and receives $\mathsf{val}(V')$. Next we invoke $\mathtt{Sim}'_2(\mathtt{st}', \mathsf{val}(V'))$ and receive $b'_1, \ldots, b'_t$. These t bits are viewed as $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$.
3. Recall that $X = \{x_1, \ldots, x_{n_i}\}$, $Y = \{y_1, \ldots, y_{n_o}\}$, $R = \{r_1, \ldots, r_\ell\}$, and $R' = \{r'_1, \ldots, r'_\ell\}$. Let $J = \tilde{V} \cup X \cup Y$ and let J' be a set of random variables such that $J \subset J' \subset (J \cup R \cup R')$ and J' is a basis of the span of $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$. For each variable in $J' \backslash (X \cup Y)$, we sample a random bit as its value. Then we compute $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ from $\mathsf{val}(J')$.
4. We use $\boldsymbol{x}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \hat{\boldsymbol{y}}$ to compute $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$.
5. Finally, we output

$$(\mathtt{parity}(W_{1,1}) \oplus \mathtt{parity}(W_{1,2}), \ldots, \mathtt{parity}(W_{t,1}) \oplus \mathtt{parity}(W_{t,2}), \boldsymbol{y})$$

The main difference between $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ is how we compute $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$.

- In $\mathbf{Hybrid}_3$, we first sample $\boldsymbol{r}$ and $\boldsymbol{r}'$ and then compute $\hat{\boldsymbol{x}} = \mathtt{Enc}(\boldsymbol{x} \| \mathbf{0}; \boldsymbol{r})$ and $\hat{\boldsymbol{y}} = \mathtt{Enc}(\boldsymbol{y} \| \mathbf{0}; \boldsymbol{r}')$. Next, we determine the set $\tilde{V}$, which contains k wires of $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$, and feed $\mathsf{val}(\tilde{V})$ to $\widetilde{\mathtt{Sim}_2}$. By the property of $\mathtt{Enc}$, when $\boldsymbol{r}$ and $\boldsymbol{r}'$ are uniformly sampled, any k wires in $\hat{\boldsymbol{x}}$ are uniformly random and any k wires in $\hat{\boldsymbol{y}}$ are also uniformly random.
- In $\mathbf{Hybrid}_4$, we first determine the set $\tilde{V}$, then we sample a random bit for each wire in $\tilde{V}$. After that, we uniformly sample $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ given $\mathsf{val}(\tilde{V})$.

They only differ in the order of the generation of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ while keeping the distribution unchanged. Thus, the output of $\mathbf{Hybrid}_4$ is identically distributed to the output of $\mathbf{Hybrid}_3$.

$\mathbf{Hybrid}_5$: In this hybrid, we first determine the set $V_1, \ldots, V_t$ and then modify the way of computing $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$. Concretely,

1. Given the input $\boldsymbol{x}$, we compute $\boldsymbol{y} = f(\boldsymbol{x})$.
2. We run Step 2 in $\mathbf{Hybrid}_4$. At the end of this step, we sampled $\mathsf{val}(\tilde{V})$ and obtained $\{\mathtt{parity}(W_{i,2})\}_{i=1}^t$.
3. We define X, Y, R, R', J, J' following Step 3 in $\mathbf{Hybrid}_4$. For each variable in $J' \backslash (X \cup Y)$, we sample a random bit as its value.
4. For all $i \in \{1, \ldots, t\}$, $\mathtt{parity}(W_{i,1})$ is a linear combination of variables in J'. Let $\mathtt{support}_{J'}(W_{i,1})$ be the support of $\mathtt{parity}(W_{i,1})$ in J'. Then $\mathtt{parity}(W_{i,1}) = \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1}))$.
   Since we have assigned values for variables in $J' \backslash (X \cup Y)$, we compute $b_i'' = \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1}) \backslash (X \cup Y))$. We set $V_i = \mathtt{support}_{J'}(W_{i,1}) \cap (X \cup Y)$.
5. Given $\boldsymbol{x}, \boldsymbol{y}$, we compute $\mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t)$.
6. For all $i \in \{1, \ldots, t\}$, we compute

$$\begin{aligned}\mathtt{parity}(W_{i,1}) &= \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1})) \\ &= \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1}) \backslash (X \cup Y)) \oplus \mathtt{parity}(\mathtt{support}_{J'}(W_{i,1}) \cap (X \cup Y)) \\ &= b_i'' \oplus \mathtt{parity}(V_i).\end{aligned}$$

7. Finally, we output

$$(\mathtt{parity}(W_{1,1}) \oplus \mathtt{parity}(W_{1,2}), \ldots, \mathtt{parity}(W_{t,1}) \oplus \mathtt{parity}(W_{t,2}), \boldsymbol{y})$$

The main difference between $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_5$ is how we compute $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$.

- In $\mathbf{Hybrid}_4$, we first compute $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$ from $\mathsf{val}(J')$ and then compute $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$ from $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}$.
- In $\mathbf{Hybrid}_5$, we simply write each $\mathtt{parity}(W_{i,1})$ as a linear combination of variables in J'. We separate the linear combination into two parts, where the first part only contains variables in $J' \backslash (X \cup Y)$ and the second part only contains variables in $J' \cap (X \cup Y)$. For the first part, it has been sampled without the knowledge of $\boldsymbol{x}, \boldsymbol{y}$. For the second part, it is a linear combination of the values in $\boldsymbol{x}, \boldsymbol{y}$. Then, we compute the second part from $\boldsymbol{x}, \boldsymbol{y}$.

Thus, the distribution of $\{\mathtt{parity}(W_{i,1})\}_{i=1}^t$ is the same in both hybrids. Therefore, the output of $\mathbf{Hybrid}_5$ is identically distributed to the output of $\mathbf{Hybrid}_4$.

Note that in $\mathbf{Hybrid}_5$, Step $2, 3, 4$ correspond to $\mathtt{Sim}_1$ and Step $6, 7$ correspond to $\mathtt{Sim}_2$. Thus, the distribution of the output of $\mathbf{Hybrid}_5$ is just

$$(\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_t)), f(\boldsymbol{x})) : (\mathtt{st}, V_1, \ldots, V_t) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t).$$

□

G Proof of Theorem 4

In this section, we formally prove Theorem 4. Let $\tilde{C}$ denote the input circuit, and (I, C, O) denote the construction in Figure 4. 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.

Trivial Parity Tolerance. For the input encoder, I takes as input $\boldsymbol{x} \in \{0,1\}^{n_i}$ and outputs $\{u_i, v_i, u_i \cdot v_i \oplus x_i\}_{i=1}^{n_i}$. Consider the following circuit implementation of I:

1. The circuit takes $\boldsymbol{x}$ as input.
2. The circuit randomly samples u_i, v_i for all $i \in \{1, \ldots, n_i\}$.
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 $\{u_i, v_i, z_i\}_{i=1}^{n_i}$.

Note that the sampled randomness $\{u_i, v_i\}_{i=1}^{n_i}$ is a part of the output, and 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.

Analysis of Wires of `NAND` in Figure 3. Consider the wires in Figure 3:

1. The input wires, auxiliary input wires, and the output wire.
2. The inner wires of Step 1.(a) are linear combinations of the input wires. If some of the wire is in the subset W, we may replace this wire by a proper subset of input wires so that the parity of W remains unchanged. Thus, without loss of generality, we do not need to consider these wires.
3. The inner wires of Step 1.(b) are computation on v_1, v_2, v_1', v_2'.
4. The inner wires of Step 1.(c) can be rewritten by $(z_1' \oplus \rho_1) \cdot (v_1' \oplus a_1 \oplus 1) \cdot (v_1 \oplus a_2 \oplus 1)$ and $(z_2' \oplus \rho_2) \cdot (v_2' \oplus a_1 \oplus 1) \cdot (v_2 \oplus a_2 \oplus 1)$ for all $\boldsymbol{a} \in \{0,1\}^2$.
5. The inner wires of Step 1.(d) are z_1' and z_2'.
6. The inner wires of Step 2.(a) are $z_1' \cdot z_2', z_1' \cdot u_2', u_1' \cdot z_2', u_1' \cdot u_2'$.
7. The inner wires of Step 2.(b) can be computed by linear combinations of $u_0, z_1' \cdot z_2', z_1' \cdot u_2', u_1' \cdot z_2', u_1' \cdot u_2'$. Without loss of generality, we do not need to consider these wires.
8. The inner wires of Step 2.(c) are computation on v_0, v_1', v_2'.
9. The inner wires of Step 2.(d) can be rewritten by $(z_0 \oplus \rho_3) \cdot (v_0 \oplus a_0 \oplus 1) \cdot (v_1' \oplus a_1 \oplus 1) \cdot (v_2' \oplus a_2 \oplus 1)$ for all $\boldsymbol{a} \in \{0,1\}^3$.

Let $\rho_1' = z_1' \oplus \rho_1$, $\rho_2' = z_2' \oplus \rho_2$, and $\rho_3' = z_0 \oplus \rho_3$. Then ρ_1, ρ_2, ρ_3 can be computed by linear combinations of $z_1', z_2', z_0, \rho_1', \rho_2', \rho_3'$. It is sufficient to only consider $\rho_1', \rho_2', \rho_3'$. Then the wires we need to consider can be summarized as follows:

- The input wires and output wire: $(u_0, v_0, z_0), (u_1, v_1, z_1), (u_2, v_2, z_2)$. (Part of Item 1)
- The refreshed input encodings: $(u_1', v_1', z_1'), (u_2', v_2', z_2')$. (Part of Item 1 and Item 5)
- The wires $z_1' \cdot z_2', z_1' \cdot u_2', u_1' \cdot z_2', u_1' \cdot u_2'$. (Item 6)
- The random wires $\rho_1', \rho_2', \rho_3'$. (Part of Item 1)
- The wires that are computation on $v_0, v_1, v_2, v_1', v_2', \rho_1', \rho_2', \rho_3'$. (Items 3,4,8,9)

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 $\boldsymbol{z}^{(2)} = (z_{\pi_2(\ell)})_{\ell=1}^{|\tilde{C}|}$. Following the analysis for the wires of `NAND` (Figure 3) above, we set $\hat{\boldsymbol{\rho}} = \boldsymbol{z} \oplus \boldsymbol{\rho}, \hat{\boldsymbol{\rho}}^{(1)} = \boldsymbol{z}^{(1)} \oplus \boldsymbol{\rho}^{(1)}, \hat{\boldsymbol{\rho}}^{(2)} = \boldsymbol{z}^{(2)} \oplus \boldsymbol{\rho}^{(2)}$. Then the wires we need to consider in C can be summarized as follows:

- Encodings of Wires in $\tilde{C}$: $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{z})$.
- Refreshed Encodings of Wires in $\tilde{C}$: $(\boldsymbol{u}^{(1)}, \boldsymbol{v}^{(1)}, \boldsymbol{z}^{(1)})$ and $(\boldsymbol{u}^{(2)}, \boldsymbol{v}^{(2)}, \boldsymbol{z}^{(2)})$.
- The wires $\boldsymbol{z}^{(1)} * \boldsymbol{z}^{(2)}, \boldsymbol{z}^{(1)} * \boldsymbol{u}^{(2)}, \boldsymbol{u}^{(1)} * \boldsymbol{z}^{(2)}, \boldsymbol{u}^{(1)} * \boldsymbol{u}^{(2)}$.
- Random Wires: $\hat{\boldsymbol{\rho}}, \hat{\boldsymbol{\rho}}^{(1)}, \hat{\boldsymbol{\rho}}^{(2)}$.
- Wires that are computation on $\boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{\rho}}^{(1)}, \hat{\boldsymbol{\rho}}^{(2)}$.

For a subset W of wires in C, we define the following subsets of wires in $\tilde{C}$:

$$\begin{aligned}\mathcal{I} &= \{w_i \mid z_i \text{ is in } W\}\\ \mathcal{I}^{(1)} &= \{w_{\pi_1(i)} \mid \text{At least one of } z_i^{(1)}, z_i^{(1)} \cdot u_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in } W\}\\ \mathcal{I}^{(2)} &= \{w_{\pi_2(i)} \mid \text{At least one of } z_i^{(2)}, u_i^{(1)} \cdot z_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in } W\}\end{aligned}$$

Let $k' = k/t$. We set

$$\mathcal{V}(W) = \begin{cases}\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}, & \text{If } |\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}| \leq k'\\ \emptyset, & \text{Otherwise}\end{cases}$$

We note that the function $\mathcal{V}(\cdot)$ satisfies that for all positive integer t and for all $W_1, \ldots, W_t$, if $\mathcal{V}(W_1), \ldots, \mathcal{V}(W_t)$ are not empty sets, then for $W_0 := \oplus_{i=1}^t W_i$, $\mathcal{V}(W_0) \subset \cup_{i=1}^t \mathcal{V}(W_i)$. To see why this is the case, let $(\mathcal{I}_i, \mathcal{I}_i^{(1)}, \mathcal{I}_i^{(2)})$ be the sets defined above for W_i for all $i \in \{0, 1, \ldots, t\}$. Then for each wire in W_0, this wire is also in W_i for some $i \in \{1, \ldots, t\}$. By definition, we have

$$\mathcal{I}_0 \subset \cup_{i=1}^t \mathcal{I}_i \quad , \quad \mathcal{I}_0^{(1)} \subset \cup_{i=1}^t \mathcal{I}_i^{(1)} \quad , \quad \mathcal{I}_0^{(2)} \subset \cup_{i=1}^t \mathcal{I}_i^{(2)}.$$

Therefore, $(\mathcal{I}_0 \cup \mathcal{I}_0^{(1)} \cup \mathcal{I}_0^{(2)}) \subset \cup_{i=1}^t (\mathcal{I}_i \cup \mathcal{I}_i^{(1)} \cup \mathcal{I}_i^{(2)})$. Since $\mathcal{V}(W_1), \ldots, \mathcal{V}(W_t)$ are not empty sets, we have $\mathcal{V}(W_i) = \mathcal{I}_i \cup \mathcal{I}_i^{(1)} \cup \mathcal{I}_i^{(2)}$ for all $i \in \{1, \ldots, t\}$.

- If $\mathcal{V}(W_0) \neq \emptyset$, then $\mathcal{V}(W_0) = \mathcal{I}_0 \cup \mathcal{I}_0^{(1)} \cup \mathcal{I}_0^{(2)} \subset \cup_{i=1}^t \mathcal{V}(W_i)$.
- Otherwise, $\mathcal{V}(W_0) = \emptyset \subset \cup_{i=1}^t \mathcal{V}(W_i)$.

Now for all t subsets $W_1, \ldots, W_t$, let $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S)$, where $W_S = \oplus_{i \in S} W_i$. We show that $|V| \leq t \cdot k' = k$. Let $\mathcal{W} = \{W_S \mid |\mathcal{V}(W_S)| \neq \emptyset\}$. Then $V = \cup_{W_S \in \mathcal{W}} \mathcal{V}(W_S)$. 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 property of $\mathcal{V}(\cdot)$, we have that for all $W_S \in \mathcal{W}$, $\mathcal{V}(W_S) \subset \cup_{i=1}^{t'} \mathcal{V}(W_i')$. Thus $V = \cup_{i=1}^{t'} \mathcal{V}(W_i')$. Then $|V| = |\cup_{i=1}^{t'} \mathcal{V}(W_i')| \leq \sum_{i=1}^{t'} |\mathcal{V}(W_i')| \leq t' \cdot k' \leq t \cdot k' = k$.

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. $\mathtt{Sim}_1$ sets $\mathtt{st} = (W_1, \ldots, W_t)$ and $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S)$.
- $\mathtt{Sim}_2$ 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 $\mathtt{Sim}_2$ randomly samples $\boldsymbol{u}, \boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{\rho}}^{(1)}, \hat{\boldsymbol{\rho}}^{(2)}$ 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 analysed above (see Appendix G) using $\boldsymbol{w}$. Concretely, we randomly sample $\boldsymbol{u}, \boldsymbol{v}, \hat{\boldsymbol{\rho}}, \boldsymbol{u}^{(1)}, \boldsymbol{v}^{(1)}, \hat{\boldsymbol{\rho}}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{v}^{(2)}, \hat{\boldsymbol{\rho}}^{(2)}$. Then, we compute $\boldsymbol{z} = \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}$ and $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$ accordingly. Finally, we faithfully compute the rest of wires.
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 $\mathbf{Hybrid}_1$ is identical to that of $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: 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 set $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S)$. 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 analysed above (see Appendix G) using $\boldsymbol{w}'$. Concretely, we randomly sample $\boldsymbol{u}, \boldsymbol{v}, \hat{\boldsymbol{\rho}}, \boldsymbol{u}^{(1)}, \boldsymbol{v}^{(1)}, \hat{\boldsymbol{\rho}}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{v}^{(2)}, \hat{\boldsymbol{\rho}}^{(2)}$. Then, we compute $\boldsymbol{z} = \boldsymbol{u} * \boldsymbol{v} \oplus \boldsymbol{w}'$ and $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$ accordingly. Finally, we faithfully compute the rest of wires.
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 wires while $\mathbf{Hybrid}'_2$ uses $\boldsymbol{w}'$. In particular $w'_i = w_i$ for all $w_i \in V$. 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 $\epsilon' = \epsilon / 2^{t/2} = 2 \cdot (7/8)^{k/2t} = 2 \cdot (7/8)^{k'/2}$, where $k' = k/t$. To this end, let $\mathcal{I}, \mathcal{I}^{(1)}, \mathcal{I}^{(2)}$ be the sets defined in Appendix G for W_S. we consider two cases:

Case 1: $|\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}| \leq k'$. In this case, we have $\mathcal{V}(W_S) = \mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}$. Then $\mathtt{parity}(W_S)$ only depends on wires in $\mathcal{V}(W_S)$ in $\tilde{C}$, and $w'_i = w_i$ for all $w_i \in \mathcal{V}(W_S)$. Thus, X_S and Y_S are identically distributed.

Case 2: $|\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}| > k'$. We define the following sets $\mathcal{U}, \mathcal{U}^{(1)}, \mathcal{U}^{(2)}$:

$$\begin{aligned}
\mathcal{U} &= \{u_i \mid \text{At least one of } u_i, z_i \text{ is in } W\} \\
\mathcal{U}^{(1)} &= \{u_i^{(1)} \mid \text{At least one of } u_i^{(1)}, z_i^{(1)}, u_i^{(1)} \cdot u_i^{(2)}, u_i^{(1)} \cdot z_i^{(2)}, z_i^{(1)} \cdot u_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in } W\} \\
\mathcal{U}^{(2)} &= \{u_i^{(2)} \mid \text{At least one of } u_i^{(2)}, z_i^{(2)}, u_i^{(1)} \cdot u_i^{(2)}, u_i^{(1)} \cdot z_i^{(2)}, z_i^{(1)} \cdot u_i^{(2)}, z_i^{(1)} \cdot z_i^{(2)} \text{ is in } W\}
\end{aligned}$$

By definition, $|\mathcal{U}| \geq |\mathcal{I}|, |\mathcal{U}^{(1)}| \geq |\mathcal{I}^{(1)}|, |\mathcal{U}^{(2)}| \geq |\mathcal{I}^{(2)}|$. Thus, $|\mathcal{U}| + |\mathcal{U}^{(1)}| + |\mathcal{U}^{(2)}| \geq |\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}| > k'$. Then at least one of $|\mathcal{U}| + |\mathcal{U}^{(1)}| \geq k'/2$, $|\mathcal{U}| + |\mathcal{U}^{(2)}| \geq k'/2$ holds. Without loss of generality, we assume that $|\mathcal{U}| + |\mathcal{U}^{(1)}| \geq k'/2$.

For fixed $\boldsymbol{w}, \boldsymbol{u}^{(2)}, \boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{\rho}}^{(1)}, \hat{\boldsymbol{\rho}}^{(2)}$, $\mathtt{parity}(W)$ is a linear combination of $\boldsymbol{u}, \boldsymbol{u}^{(1)}$. Since $\boldsymbol{u}, \boldsymbol{u}^{(1)}$ are uniform, $\mathtt{parity}(W)$ is uniformly random if and only if not all coefficients of $\boldsymbol{u}, \boldsymbol{u}^{(1)}$ are 0. Now we analyse the coefficient of each $u_i \in \mathcal{U}$ and $u_i^{(1)} \in \mathcal{U}^{(1)}$.

- For each $u_i \in \mathcal{U}$, the coefficient of u_i is a function on v_i, denoted by $f_i(v_i)$. In particular, $f_i(v_i) \not\equiv 0$. Thus, when v_i is randomly sampled, with probability at least $1/2$, $f_i(v_i) = 1$.
- For each $u_i^{(1)} \in \mathcal{U}^{(1)}$, the coefficient of $u_i^{(1)}$ is a function on $v_i^{(1)}, u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)}$, denoted by $f_i^{(1)}(v_i^{(1)}, u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)})$. In particular, $f_i^{(1)}(v_i^{(1)}, u_i^{(2)}, v_i^{(2)}, 0) \not\equiv 0$ and $f_i^{(1)}(v_i^{(1)}, u_i^{(2)}, v_i^{(2)}, 1) \not\equiv 0$. Thus when $v_i^{(1)}, u_i^{(2)}, v_i^{(2)}$ are randomly sampled, with probability at least $1/8$, $f_i^{(1)}(v_i^{(1)}, u_i^{(2)}, v_i^{(2)}, w_{\pi_2(i)}) = 1$.

Note that the coefficients of all $u_i \in \mathcal{U}$ and all $u_i^{(1)} \in \mathcal{U}^{(1)}$ are independent. Thus, for all $\boldsymbol{w}$, when $\boldsymbol{u}^{(2)}, \boldsymbol{v}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}$ are randomly sampled, the probability that all coefficients are 0 is bounded by $(1/2)^{|\mathcal{U}|} \cdot (7/8)^{|\mathcal{U}^{(1)}|} \leq (7/8)^{k'/2}$. Therefore, when $|\mathcal{I} \cup \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}| > k'$, $\mathtt{parity}(W)$ is statistically close to a uniform bit with error $(7/8)^{k'/2}$. 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 $2 \cdot (7/8)^{k'/2} = \epsilon'$.

In summary, the statistical distance between X_S and Y_S in both 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,

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

This finishes the proof of Theorem 4.

H Evidence Against Efficient Simulation

Recall that in our main feasibility result for parity tolerance, the simulator needs to run in super-polynomial time. Indeed, in our construction of parity-to-probing circuits, we need the simulator to compute a set $V = \cup_{S \subset \{1,\ldots,t\}} \mathcal{V}(W_S)$. A straightforward implementation of this step would be computing $\mathcal{V}(W_S)$ for all $S \subset \{1, \ldots, t\}$, which takes exponential time in t. In this section, we provide evidence that the above inefficient simulation may be inherent.

We consider a relaxed notion of parity tolerance for a randomized function. Informally, for a (t, t')-parity-tolerant randomized function, we allow the simulator to query at most $t' \geq t$ parities of the function input to simulate t parities of the function output.

Definition 10 ((t, t')-Parity-Tolerant Functions). *For a randomized function $F : \{0,1\}^n \to \{0,1\}^m$, we say F is (t, t', ϵ)-parity-tolerant if there exists a simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ with the following syntax:*

- *$\mathtt{Sim}_1$ takes as input t subsets $W_1, \ldots, W_t$ of the output of F, and outputs t a state $\mathtt{st}$ and t' subsets $V_1, \ldots, V_{t'}$ of the input of F;*
- *$\mathtt{Sim}_2$ takes the state $\mathtt{st}$ and $\mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'})$ as input, and outputs t bits $(b_1, \ldots, b_t)$.*

The simulator $\mathtt{Sim}$ satisfies that for all input $\boldsymbol{x} \in \{0,1\}^n$ and for all t subsets $W_1, \ldots, W_t$ of the output of F, the following two distributions are statistically close with error ϵ:

$$\begin{aligned} &(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)) \\ \approx_\epsilon\ &(\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'}))) : (\mathtt{st}, V_1, \ldots, V_{t'}) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t). \end{aligned}$$

In the following, we construct a (t, t', ϵ)-parity-tolerant randomized function such that if $\mathtt{Sim}$ is a probabilistic polynomial-time algorithm, then the following assumption is broken.

Assumption 1 (LPN Assumption with Fixed Weight Noise [BFKL93,BCG+19]) *For $k, N \in \mathbb{N}$, let $\mathcal{HW}_{k,N}$ be the distribution of uniform, weight-k vectors over $\{0,1\}^N$. Let κ be the security parameter. For $t = t(\kappa), N = N(\kappa), k = k(\kappa)$, the $\mathtt{LPN}(t, N, k)$ assumption states that*

$$\begin{aligned}&\{(\boldsymbol{A}, \boldsymbol{b}) : \boldsymbol{A} \xleftarrow{\$} \{0,1\}^{N\times t}, \boldsymbol{e} \xleftarrow{\$} \mathcal{HW}_{k,N}, \boldsymbol{s} \xleftarrow{\$} \{0,1\}^t, \boldsymbol{b} \leftarrow \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}\}\\ \overset{c}{\approx}\ &\{(\boldsymbol{A}, \boldsymbol{b}) : \boldsymbol{A} \xleftarrow{\$} \{0,1\}^{N\times t}, \boldsymbol{b} \xleftarrow{\$} \{0,1\}^N\}.\end{aligned}$$

Theorem 10 (Impossibility of Black-Box Simulation). *Let κ be the security parameter. Let $t = \kappa, N = 40\kappa^3, k = 5\kappa$. Then there exists a family of randomized functions $\{F : \{0,1\}^{2N} \to \{0,1\}^N\}_N$ such that:*

1. ***Inefficient Simulation**: F is $(t, 2k^2t, \epsilon)$-parity-tolerant, where $\epsilon = 2^{t/2} \cdot (3/4)^k \le 2^{-\kappa}$.*
2. ***Impossibility of Black-Box Simulation**: Assuming the $\mathtt{LPN}(t-1, N, k)$ assumption, there exists a distribution $\mathcal{D}(t, N, k)$ such that for all large enough security parameter κ and for all PPT algorithms $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ with the syntax in Definition 10, there exists an input $\boldsymbol{x}$ such that the statistical distance between*

$$(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)) : (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), \boldsymbol{y} = F(\boldsymbol{x})$$

and

$$\begin{aligned}&\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'}))\\ &: (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), (\mathtt{st}, V_1, \ldots, V_{t'}) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t)\end{aligned}$$

is at least $1/16$, where $t' = 2k^2t$.

Proof. We first give the construction of F as follows. Let $\boldsymbol{x} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) \in \{0,1\}^{2N}$ denote the input of F such that $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \in \{0,1\}^N$. The output of F is defined as follows:

- F samples N independent random bits $(a_1, \ldots, a_N)$ from the Bernoulli distribution with probability $q = (1 - (3/4)^{1/k})/2$ to be 1. Then it outputs $\boldsymbol{y} = \boldsymbol{a} \oplus (\boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)})$, where $*$ denotes coordinate-wise multiplication (i.e., bitwise AND).

Part I: Inefficient Simulation. We first show the existence of $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$. Let $\boldsymbol{x}$ denote the function input and $\boldsymbol{y} = F(\boldsymbol{x})$. We construct a matrix $\boldsymbol{M}$ of size $N \times t$:

- For all $i \in \{1, \ldots, N\}, j \in \{1, \ldots, t\}$, $M_{ij} = 1$ if $y_i \in W_j$ and $M_{ij} = 0$ otherwise.

Observe that for all $j \in \{1, \ldots, t\}$, $\mathtt{parity}(W_j) = \langle \boldsymbol{M}_{\star,j}, \boldsymbol{y} \rangle$. In the following, we use $\mathtt{span}(\boldsymbol{M})$ to denote the span of the column vectors of $\boldsymbol{M}$.

The simulator $\mathtt{Sim}_1$ first finds all indices i such that there is a vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most k^2 and $c_i = 1$. Suppose I is the set that contains all such indices. We have the following claim.

Claim 7 *Suppose I is the set of all indices i such that there is a vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most k^2 and $c_i = 1$. Then $|I| \le k^2t$.*

Proof (Proof of Claim 7). Consider all vectors of hamming weight no more than k^2 in $\mathtt{span}(\boldsymbol{M})$. Suppose the rank of these vectors is $\widetilde{t} \le t$. Let $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{\widetilde{t}}$ be a basis. Consider the set $I = \{i \mid \exists j \in \{1, \ldots, \widetilde{t}\},\ \text{s.t.},\ c_{ji} = 1\}$. Then $|I| \le k^2\widetilde{t} \le k^2t$. We claim that I is the set in the above statement.

To see why this is the case, note that for each vector $\boldsymbol{c}$ that has hamming weight no more than k^2 in $\mathtt{span}(\boldsymbol{M})$, $\boldsymbol{c}$ is a linear combination of $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\widetilde{t}}$. Therefore, for all $i \in \{1, \ldots, N\}$, $c_i = 1$ only if there exists j such that $c_{ji} = 1$, which implies that $i \in I$.

Now $\mathtt{Sim}_1$ computes the output as follows:

- Suppose $I = \{i_1, \ldots, i_{|I|}\}$. For all $j \in \{1, \ldots, |I|\}$, $\mathtt{Sim}_1$ sets $V_{2j-1} = \{x^{(1)}_{i_j}\}$ and $V_{2j} = \{x^{(2)}_{i_j}\}$.

– $\mathtt{Sim}_1$ sets $\mathtt{st} = (I, W_1, \ldots, W_t)$.

As for $\mathtt{Sim}_2$, it receives $\mathtt{st} = (I, W_1, \ldots, W_t)$ and $\{\mathtt{parity}(V_j)\}_{j=1}^{2|I|}$. Then for all $i \in \{1, \ldots, N\}$, $\mathtt{Sim}_2$ randomly samples a_i. For all $i_j \in I$, $\mathtt{Sim}_2$ computes $x_{i_j}^{(1)} \cdot x_{i_j}^{(2)} = \mathtt{parity}(V_{2j-1}) \cdot \mathtt{parity}(V_{2j})$. For all $i \notin I$, $\mathtt{Sim}_2$ sets $x_i^{(1)} \cdot x_i^{(2)} = 0$. Based on a_i and $x_i^{(1)} \cdot x_i^{(2)}$, $\mathtt{Sim}_2$ computes y_i and outputs $\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)$.

Claim 8 *For all input $\boldsymbol{x}$ and for all $W_1, \ldots, W_t$, the output distribution of* $\mathtt{Sim}$ *is statistically close to the joint distribution of* $(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t))$ *with error* $\epsilon = 2^{t/2} \cdot (3/4)^k$.

Proof (Proof of Claim 8). Let I be the set defined above. Then

– The set I contains all indices i such that there exists a vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most k^2 and $c_i = 1$. By Claim 7, $|I| \leq k^2 t$.

By definition, $\mathtt{Sim}_2$ learns $\{x_i^{(1)}, x_i^{(2)}\}_{i \in I}$. Let $\boldsymbol{y} = F(\boldsymbol{x})$ and $\widetilde{\boldsymbol{y}}$ be the vector computed by $\mathtt{Sim}_2$ (i.e., replacing $x_i^{(1)} \cdot x_i^{(2)} = 0$ for all $i \notin I$). We want to show that

$$(\mathtt{parity}(W_1(\boldsymbol{y})), \ldots, \mathtt{parity}(W_t(\boldsymbol{y}))) \approx_\epsilon (\mathtt{parity}(W_1(\widetilde{\boldsymbol{y}})), \ldots, \mathtt{parity}(W_t(\widetilde{\boldsymbol{y}}))).$$

By the XOR lemma [Gol11], it is sufficient to show that for all $S \subset \{1, \ldots, t\}$, the statistical distance between $\mathtt{parity}(W_S(\boldsymbol{y}))$ and $\mathtt{parity}(W_S(\widetilde{\boldsymbol{y}}))$ is $\epsilon' = \epsilon / 2^{t/2} = (3/4)^k$. Observe that

$$\begin{aligned}
\mathtt{parity}(W_S(\boldsymbol{y})) &= \oplus_{j \in S} \mathtt{parity}(W_j(\boldsymbol{y})) \\
&= \oplus_{j \in S} \langle \boldsymbol{M}_{\star,j}, \boldsymbol{y} \rangle \\
&= \langle \oplus_{j \in S} \boldsymbol{M}_{\star,j}, \boldsymbol{y} \rangle.
\end{aligned}$$

Let $\boldsymbol{c} = \sum_{j \in S} \boldsymbol{M}_{\star,j}$. Then $\boldsymbol{c}$ is in $\mathtt{span}(\boldsymbol{M})$, and $\mathtt{parity}(W_S(\boldsymbol{y})) = \langle \boldsymbol{c}, \boldsymbol{y} \rangle$. Similarly, $\mathtt{parity}(W_S(\widetilde{\boldsymbol{y}})) = \langle \boldsymbol{c}, \widetilde{\boldsymbol{y}} \rangle$. We consider two cases.

– If the hamming weight of $\boldsymbol{c}$ is larger than k^2, then $\langle \boldsymbol{c}, \boldsymbol{y} \rangle = \langle \boldsymbol{c}, \boldsymbol{a} \rangle \oplus \langle \boldsymbol{c}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)} \rangle$. Given $\boldsymbol{x} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$, $\langle \boldsymbol{c}, \boldsymbol{a} \rangle$ is statistically close to uniform with error $(1-2q)^{k^2}/2 = (3/4)^k/2$ (following from Claim 5). Thus, $\langle \boldsymbol{c}, \boldsymbol{y} \rangle$ is statistically close to uniform with error $(3/4)^k/2$.
  Similarly, $\langle \boldsymbol{c}, \widetilde{\boldsymbol{y}} \rangle$ is also statistically close to uniform with error $(3/4)^k/2$. Thus the statistical distance between $\mathtt{parity}(W_S(\boldsymbol{y}))$ and $\mathtt{parity}(W_S(\widetilde{\boldsymbol{y}}))$ is at most $(3/4)^k$.
– If the hamming weight of $\boldsymbol{c}$ is no more than k^2, then for all i such that $c_i = 1$, we have $i \in I$. Therefore, $\widetilde{y}_i$ and y_i have the same distribution. Thus, $\mathtt{parity}(W_S(\boldsymbol{y}))$ and $\mathtt{parity}(W_S(\widetilde{\boldsymbol{y}}))$ are identically distributed.

Part II: Impossibility of Efficient Simulation. We construct the distribution $\mathcal{D}(t, N, k)$ as follows: $(W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k)$ is sampled by

1. Sampling $\boldsymbol{A} \xleftarrow{\$} \{0,1\}^{N \times (t-1)}$, $\boldsymbol{e} \xleftarrow{\$} \mathcal{HW}_{k,N}$, and $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{t-1}$;
2. Computing $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$;
3. Setting $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{b})$;
4. For all $j \in \{1, \ldots, t\}$, $W_j = \{y_i \mid M_{ij} = 1\}$.

Now assume that there exists a PPT simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$ such that for all input $\boldsymbol{x}$,

$$\begin{aligned}
&(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)) : (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), \boldsymbol{y} = F(\boldsymbol{x}) \\
\approx_\eta\ &\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'})) : (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), \\
&(\mathtt{st}, V_1, \ldots, V_{t'}) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t),
\end{aligned}$$

where $\eta \leq 1/16$. We will construct an PPT algorithm $\mathtt{Sim}'$ that finds the vector $\boldsymbol{e}$ with probability $\eta' = 1/16 - 2^t \cdot e^{-N/16} = 1/16 - \mathrm{negl}(\kappa)$:

1. $\mathtt{Sim}'$ invokes $\mathtt{Sim}$ on $W_1, \ldots, W_t$ and receives $V_1, \ldots, V_{t'}$, where $t' = 2k^2t$.
2. $\mathtt{Sim}'$ initially sets $I' = \emptyset$. For all $i \in \{1, \ldots, N\}$, if $x_i^{(1)}$ is linearly dependent with $\mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'})$, add i in I'.
3. Let $J = \{1, \ldots, n\} \backslash I$. Solve $\boldsymbol{M}_J \cdot \boldsymbol{\alpha} = \mathbf{0}$, where $\boldsymbol{M}_J$ denotes the sub-matrix of $\boldsymbol{M}$ that only contains rows of indices in J. $\mathtt{Sim}'$ finds the first non-zero solution $\boldsymbol{\alpha}$ (if exists). If $\boldsymbol{M} \cdot \boldsymbol{\alpha}$ has hamming weight k, output $\boldsymbol{M} \cdot \boldsymbol{\alpha}$. Otherwise, output $\perp$.

Let I be the set of all indices i such that there exists a vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most $t' = 2k^2t$ and $c_i = 1$ (Note that I is different from that in Part I). We show that with overwhelming probability, there is a unique non-zero vector $\boldsymbol{c}$ of hamming weight at most t' and $\boldsymbol{c} = \boldsymbol{e}$ has hamming weight k.

Claim 9 *Let $\boldsymbol{M}$ be the random matrix defined above. With probability at least $1 - \delta$, where $\delta = 2^t \cdot e^{-N/16}$, $\boldsymbol{M}$ has full column rank and there is exactly one non-zero vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most $t' = 2k^2t$ and its hamming weight is k.*

Proof. Recall the sampling process of $\boldsymbol{M}$:

1. Sample $\boldsymbol{A} \xleftarrow{\$} \{0,1\}^{N \times (t-1)}$, $\boldsymbol{e} \xleftarrow{\$} \mathcal{HW}_{k,N}$, and $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{t-1}$;
2. Compute $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$;
3. Set $\boldsymbol{M} = (\boldsymbol{A}, \boldsymbol{b})$;

Therefore the vector $\boldsymbol{e}$ is in $\mathtt{span}(\boldsymbol{M})$ and its hamming weight is k. We show that with probability $1 - \delta$, all other non-zero linear combination of the column vectors of $\boldsymbol{M}$ has hamming weight larger than t'. Note that this implies that $\boldsymbol{M}$ has full column rank.

Note that for all non-zero linear combination of the column vectors of $\boldsymbol{M}$, denoted by $\boldsymbol{c}$, such that $\boldsymbol{c} \neq \boldsymbol{e}$, either $\boldsymbol{c}$ or $\boldsymbol{c} \oplus \boldsymbol{e}$ can be written as a non-zero linear combination of column vectors in $\boldsymbol{A}$. Since $\boldsymbol{A}$ is a uniform matrix, it implies that $\boldsymbol{c}$ is a random vector. By Chernoff Bound, with probability at least $1 - e^{-N/16}$, the hamming weight of $\boldsymbol{c}$ is larger than t'. By union bound, with probability at least $1 - 2^t \cdot e^{-N/16}$, all non-zero vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ such that $\boldsymbol{c} \neq \boldsymbol{e}$ has hamming weight larger than t'.

Now assume that $\boldsymbol{M}$ has full column rank and there is exactly one non-zero vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most t' and $\boldsymbol{c} = \boldsymbol{e}$ has hamming weight k. We show that, with constant probability, $I \subset I'$.

Since $\boldsymbol{e} \in \mathtt{span}(\boldsymbol{M})$, there exists $S \subset \{1, \ldots, t\}$ such that $\mathtt{parity}(W_S) = \langle \boldsymbol{e}, \boldsymbol{y} \rangle$, where $\boldsymbol{y} = F(\boldsymbol{x})$ is the function output. Therefore, from the output of $\mathtt{Sim}$, we can compute a bit b that is statistically close to $\langle \boldsymbol{e}, \boldsymbol{y} \rangle$ with error η. Since $\langle \boldsymbol{e}, \boldsymbol{y} \rangle = \langle \boldsymbol{e}, \boldsymbol{a} \rangle \oplus \langle \boldsymbol{e}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)} \rangle$, we have $\Pr[\langle \boldsymbol{e}, \boldsymbol{y} \rangle = \langle \boldsymbol{c}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)} \rangle] = \Pr[\langle \boldsymbol{e}, \boldsymbol{a} \rangle = 0]$. Recall that each a_i is sampled independently with probability q to be 1. Thus, $\Pr[\langle \boldsymbol{e}, \boldsymbol{a} \rangle = 0] = (1 + (1 - 2q)^k)/2 = 7/8$.

Let E denote the event that b computed from the output of $\mathtt{Sim}$ is equal to $\langle \boldsymbol{c}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)} \rangle$. When $\boldsymbol{x}$ is sampled uniformly, we have

$$\begin{aligned} \Pr[E] &= \Pr[E \mid I \not\subset I'] \cdot \Pr[I \not\subset I'] + \Pr[E \mid I \subset I'] \cdot \Pr[I \subset I'] \\ &\leq \Pr[E \mid I \not\subset I'] + \Pr[I \subset I']. \end{aligned}$$

We show that $\Pr[E \mid I \not\subset I'] \leq 3/4$.

Assume that $x_{i^\star}^{(1)} \in I$ but $x_{i^\star}^{(1)} \notin I'$. Let $z_0 = x_{i^\star}^{(1)}$ and $z_i = \mathtt{parity}(V_i)$ for all $i \in \{1, \ldots, t'\}$. Since $z_0 = x_{i^\star}^{(1)} \notin I'$, z_0 is linearly independent of $z_1, \ldots, z_{t'}$. We randomly pick $z_{t'+1}, \ldots, z_{2n-1}$ such that $z_0, \ldots, z_{2n-1}$ is a basis of the span of $\boldsymbol{x}$. Then each $x_i^{(1)}$ and $x_i^{(2)}$ can be written as a linear combination of $z_0, \ldots, z_{2n-1}$. Let $\boldsymbol{z} = (z_1, \ldots, z_{2n-1})$. Suppose $x_i^{(1)} = \alpha_{i,0} \cdot z_0 \oplus \boldsymbol{\alpha}_i \cdot \boldsymbol{z}$ and $x_i^{(2)} = \beta_{i,0} \cdot z_0 \oplus \boldsymbol{\beta}_i \cdot \boldsymbol{z}$. Then

$$\begin{aligned} &\langle \boldsymbol{e}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)} \rangle \\ &= \oplus_{i \in I} x_i^{(1)} \cdot x_i^{(2)} \\ &= \oplus_{i \in I} (\alpha_{i,0} \cdot z_0 \oplus \boldsymbol{\alpha}_i \cdot \boldsymbol{z}) \cdot (\beta_{i,0} \cdot z_0 \oplus \boldsymbol{\beta}_i \cdot \boldsymbol{z}) \\ &= z_0 \cdot ((\oplus_{i \in I} \alpha_{i,0} \cdot \beta_{i,0}) \oplus (\oplus_{i \in I} (\alpha_{i,0} \cdot \boldsymbol{\beta}_i \oplus \beta_{i,0} \cdot \boldsymbol{\alpha}_i)) \cdot \boldsymbol{z}) \oplus (\oplus_{i \in I} (\boldsymbol{\alpha}_i \cdot \boldsymbol{z}) \cdot (\boldsymbol{\beta}_i \cdot \boldsymbol{z})). \end{aligned}$$

We want to show that the coefficient of z_0 is not a constant. This is equivalent to show that $\oplus_{i\in I}(\alpha_{i,0}\cdot\boldsymbol{\beta}_i \oplus \beta_{i,0}\cdot\boldsymbol{\alpha}_i) \neq \mathbf{0}$. First note that $\{x_i^{(1)}, x_i^{(2)}\}_{i\in I}$ are linearly independent. Since $z_0 = x_{i^\star}^{(1)}$, we have

$$\{x_i^{(1)} \oplus \alpha_{i,0}\cdot z_0\}_{i\in I, i\neq i^\star} \cup \{x_i^{(2)} \oplus \beta_{i,0}\cdot z_0\}_{i\in I}$$

are linearly independent. Recall that $x_i^{(1)} = \alpha_{i,0}\cdot z_0 \oplus \boldsymbol{\alpha}_i\cdot\boldsymbol{z}$ and $x_i^{(2)} = \beta_{i,0}\cdot z_0 \oplus \boldsymbol{\beta}_i\cdot\boldsymbol{z}$. Therefore, $\{\boldsymbol{\alpha}_i\cdot\boldsymbol{z}\}_{i\in I, i\neq i^\star} \cup \{\boldsymbol{\beta}_i\cdot\boldsymbol{z}\}_{i\in I}$ are linearly independent, which implies that $\{\boldsymbol{\alpha}_i\}_{i\in I, i\neq i^\star} \cup \{\boldsymbol{\beta}_i\}_{i\in I}$ are linearly independent. Note that $\oplus_{i\in I}\alpha_{i,0}\cdot\boldsymbol{\beta}_i \oplus \beta_{i,0}\cdot\boldsymbol{\alpha}_i$ is a linear combination of $\{\boldsymbol{\alpha}_i\}_{i\in I, i\neq i^\star} \cup \{\boldsymbol{\beta}_i\}_{i\in I}$, and in particular, $\alpha_{i^\star,0} = 1$, which means that not all coefficients are 0s. Thus, $\oplus_{i\in I}\alpha_{i,0}\cdot\boldsymbol{\beta}_i \oplus \beta_{i,0}\cdot\boldsymbol{\alpha}_i \neq \mathbf{0}$. Now we have shown that the coefficient of z_0 is a non-zero linear combination of $\boldsymbol{z}$.

When $\boldsymbol{x}$ is sampled uniformly, $z_0, \boldsymbol{z}$ are also uniform. Therefore, the coefficient of z_0 is uniformly distributed. Since z_0 is linearly independent of $\boldsymbol{z}$, and the simulator only learns $\{\mathtt{parity}(V_i)\}_{i=1}^{t'} \subset \{z_1, \ldots, z_{2n-1}\}$, z_0 is uniformly distributed given $\{\mathtt{parity}(V_i)\}_{i=1}^{t'} \subset \{z_1, \ldots, z_{2n-1}\}$. We have the following two cases:

- If the coefficient of z_0 is 0, which happens with probability $1/2$, E happens with probability at most 1.
- If the coefficient of z_0 is 1, which happens with probability $1/2$, E happens with probability at most $1/2$.

Thus, when $\boldsymbol{x}$ is sampled uniformly, $\Pr[E \mid I \not\subset I'] \leq 3/4$.

On the other hand, since from the output of $\mathtt{Sim}$, we can compute a bit b that is statistically close to $\langle \boldsymbol{c}, \boldsymbol{y}\rangle$ with error $\eta = 1/16$, and $\Pr[\langle \boldsymbol{c}, \boldsymbol{y}\rangle = \langle \boldsymbol{c}, \boldsymbol{x}^{(1)} * \boldsymbol{x}^{(2)}\rangle] \geq 7/8$, we have

$$\Pr[I \subset I'] \geq \Pr[E] - \Pr[E \mid I \not\subset I'] \geq 7/8 - \eta - 3/4 = 1/8 - \eta \geq 1/16.$$

Thus, with constant probability, $I \subset I'$.

Recall that $J = \{1, \ldots, n\}\backslash I'$. When $I \subset I'$, $I \cap J = \emptyset$. On the other hand, $|J| = N - |I'| = N - 2k^2t$. For any non-zero vector $\boldsymbol{\gamma} \in \{0,1\}^t$ such that $\boldsymbol{M}_J\cdot\boldsymbol{\gamma} = \mathbf{0}$, $\boldsymbol{M}\cdot\boldsymbol{\gamma}$ is a non-zero vector and its hamming weight is at most $2k^2t$. This implies that $\boldsymbol{M}\cdot\boldsymbol{\gamma} = \boldsymbol{e}$, the vector with hamming weight k.

Thus, given that $\boldsymbol{M}$ has full column rank and there is exactly one non-zero vector $\boldsymbol{c} \in \mathtt{span}(\boldsymbol{M})$ with hamming weight at most t' and $\boldsymbol{c} = \boldsymbol{e}$ has hamming weight k, with probability $1/16$, the algorithm $\mathtt{Sim}'$ outputs $\boldsymbol{e}$. By Claim 9, $\mathtt{Sim}'$ outputs $\boldsymbol{e}$ with probability at least $\epsilon' = 1/16 - 2^t\cdot e^{-N/16}$.

Note that $\mathtt{Sim}'$ breaks the $\mathtt{LPN}(t-1, N, k)$ assumption. Therefore, for all PPT simulator $\mathtt{Sim} = (\mathtt{Sim}_1, \mathtt{Sim}_2)$, there exists $\boldsymbol{x}$ such that the statistical distance between

$$(\mathtt{parity}(W_1), \ldots, \mathtt{parity}(W_t)) : (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), \boldsymbol{y} = F(\boldsymbol{x})$$

and

$$\begin{aligned}&\mathtt{Sim}_2(\mathtt{st}, \mathtt{parity}(V_1), \ldots, \mathtt{parity}(V_{t'}))\\ &: (W_1, \ldots, W_t) \xleftarrow{\$} \mathcal{D}(t, N, k), (\mathtt{st}, V_1, \ldots, V_{t'}) \leftarrow \mathtt{Sim}_1(W_1, \ldots, W_t)\end{aligned}$$

is at least $1/16$.