# Leakage-Tolerant Circuits Against $\mathsf{AC}^0$ Leakage

Yaohua Ma

Tsinghua University
ma-yh21@mails.tsinghua.edu.cn

Yifan Song

Tsinghua University and Shanghai Qi Zhi Institute
yfsong@mail.tsinghua.edu.cn

**Abstract**

A leakage-tolerant circuit (LTC) can be viewed as a black-box implementation of a given functionality $f$ with respect to a leakage class $\mathcal{L}$ in the sense that any leakage function $L \in \mathcal{L}$ applied to the circuit's internal wires can be simulated by a similar leakage function $L' \in \mathcal{L}$ applied only to the circuit's inputs and outputs. Previous works have demonstrated extensive applications of LTCs in constructing variants of leakage-resilient circuits (LRC): black-box construction of both stateless and stateful LRCs, and construction of deterministic stateful LRCs which only require external fresh randomness in the first invocation. However, feasibility results for LTCs are still limited to simple leakage classes, including only probing leakage, depth-1 $\mathsf{AC}^0$ leakage, and parity leakage.

In this work, we instantiate the study of constructing LTCs and deterministic stateful LRCs against $\mathsf{AC}^0$ leakage, obtaining the following results:

- We present the first construction of LTCs against generic $\mathsf{AC}^0$ leakage. As a corollary, we also construct LTCs against parity leakage with efficient simulation, refuting a conjecture proposed by Ishai and Song (Eurocrypt 2024).
- We provide a generic framework to convert LTCs into computationally secure deterministic LRCs assuming one-way functions, and instantiate the paradigm for $k$-CNF leakage (with a sufficiently small $k$). This is the first instance of deterministic stateful LRCs against *non-decomposable* leakage.

## 1 Introduction

Modern cryptographic systems are often deployed in complicated environments where adversaries may infer the intermediate computation results of the systems from their physical reactions to the environments, whereas theoretical security is typically only preserved against idealized input-output attacks. This type of attack on computational leakage is known as side-channel attacks, including timing [BB03], energy consumption [KJJ99], electromagnetic radiation [QS01, GMO01], memory access pattern [Koc96, BM06], and so on. Side-channel attacks pose severe challenges to cryptographic primitives, thus studying the defense becomes a fundamental problem of cryptography.

Leakage-resilient cryptography aims to bridge the gap between theory and practice by converting cryptographic primitives into variants that remain secure even when adversaries can obtain bounded or structured leakage information. To capture different types of realistic attacks, researchers have studied diverse theoretical defense frameworks. In this work, we are interested in the notion of *leakage-resilient circuits* (LRC for short), a clean and powerful model proposed by Ishai et.al. [ISW03], which captures computation leakage as a boolean function $L$ chosen from some function family that takes the values of all wires as input.

**Leakage-Resilient Circuits.** Let $\mathcal{L}$ be a class of leakage functions. A *stateless* LRC implementation of a function $f : \{0,1\}^n \to \{0,1\}^m$ is a randomized boolean circuit $C : \{0,1\}^{\widehat{n}} \to \{0,1\}^{\widehat{m}}$ mapping an encoded input $\widehat{\boldsymbol{x}}$ to an encoded output $\boldsymbol{y}$, together with an input encoder $I : \{0,1\}^n \to \{0,1\}^{\widehat{n}}$ and an output decoder $O : \{0,1\}^{\widehat{m}} \to \{0,1\}^m$. We require that for any input $\boldsymbol{x}$, $O(C(I(\boldsymbol{x})))$ computes the function output $f(\boldsymbol{x})$; and applying any leakage function $L \in \mathcal{L}$ to the wire values of $C$ essentially reveals no information about the function input $\boldsymbol{x}$. Here we assume that $I$ and $O$ are implemented by secure hardware whose inner wires cannot be accessed by $L$.

However, stateless LRC falls short to model realistic needs in many scenarios due to two inherent limitations: (1) its security relies on leak-free implementations of $I$ and $O$, and (2) it only supports a *one-shot* evaluation. To address these two limitations, the notion of *stateful* LRC [ISW03] captures leakage resilience in a reactive functionality. Concretely, for an ideal stateful circuit $C$ with a secret state $\boldsymbol{s}$, a stateful LRC of $C$ is a stateful circuit $C'$ with an encoded secret state $\boldsymbol{s}'$ such that $C'[\boldsymbol{s}']$ achieves the same reactive functionality as $C[\boldsymbol{s}]$, and the secret state $\boldsymbol{s}$ remains secure even if an adversary obtains leakage on the computation in each invocation from the prescribed leakage class $\mathcal{L}$. Compared with stateless LRC, a stateful LRC only requires leak-free input encoding of the initial secret state and can tolerate continuous leakage that may be unbounded over time.

Although stateful LRC is already a powerful primitive that protects continuous computation from leakage, its security guarantee highly relies on external fresh randomness in each invocation. As a result, an adversary who can tamper the external randomness required by the stateful LRC may break its security entirely. A recent work by Ishai and Song [IS25] aims to address this concern by proposing the notion of *deterministic* stateful LRC, which is a stateful LRC but only requires external fresh randomness in the initial invocation. Such external fresh randomness can be provided along with the initial encoded secret state.

To be able to capture realistic side-channel attacks as much as possible, it is imperative to construct stateful LRCs against broader leakage classes. However, this task appears to be very difficult even for simple leakage classes. As an evidence, up to date, feasibility solutions for stateful LRCs are only known for a few leakage classes, including probing leakage [ISW03], parity leakage [IS24], $\mathsf{AC}^0$ leakage [BIS21], the "only computation leak" leakage [MR04, GR12], and bounded-communication leakage [IS25]. The line of works [MV13, Mil14, Wan25] also constructs stateful LRCs against $\mathsf{NC}^1$ leakage but requires leak-free hardware. Even fewer results are known for deterministic stateful LRCs, including only probing leakage [ISW03], parity leakage [IS25], and depth-1 $\mathsf{AC}^0$ leakage [IS25] (i.e., the conjunction or disjunction function).

In this work, we are interested in constructing deterministic stateful LRCs against richer leakage classes.

**Leakage-Tolerant Circuits.** Another relevant notion is *leakage-tolerant* circuit (or LTC for short). Informally speaking, an LTC $C$ is a randomized Boolean circuit that maps an unprotected input $\boldsymbol{x}$ to an unprotected output $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$, while the security requirement is the best we can expect in the sense that any leakage query $L \in \mathcal{L}$ applied to the inner wires of $C$ can be *simulated* by a similar leakage query $L' \in \mathcal{L}$ that is only applied to $\boldsymbol{x}$ and $\boldsymbol{y}$. Effectively, an LTC $C$ is a black-box implementation of the target functionality $f$ with respect to the leakage class $\mathcal{L}$. Thus, this notion itself is already very interesting in theory as it studies what kind of security can be provided from a circuit itself. In addition, recent works have identified applications of LTCs in constructing stateless and stateful LRCs:

- In [IS24], the authors show that an LTC against the leakage class $\mathcal{L}$ can be used to construct both stateless and stateful LRCs against the same leakage class in a black-box way.

- Moving forward, in [IS25], the authors construct deterministic stateful LRCs against parity leakage and depth-1 $\mathsf{AC}^0$ leakage from LTCs against the same leakage classes, with an additional requirement on randomness complexity of the LTCs.

However, building LTCs appears to be much more difficult than building (stateful or stateless) LRCs. As an evidence, solutions for both stateful and stateless LRCs against $\mathsf{AC}^0$ leakage are known in [FRR+10, Rot12, BIS21] whereas feasibility results for LTCs are only known against the simple depth-1 $\mathsf{AC}^0$ leakage [IS24].

Motivated by the importance of LRCs and LTCs in both theory and practice, we are interested in the following research question.

*"Can we construct leakage-tolerant circuits and deterministic stateful leakage-resilient circuits against a broader leakage class?"*

## 2 Our Contribution

In this work, we provide a positive answer to the above question by constructing the first LTC against generic $\mathsf{AC}^0$ leakage. In addition, we also construct a deterministic stateful LRC against the $k$-CNF leakage class (with a sufficiently small $k$), an important sub-class of depth-2 $\mathsf{AC}^0$ leakage. Our contribution can be summarized in the following three points.

**Leakage Tolerant Circuits Against $\mathsf{AC}^0$ Leakage.** Our work gives the first LTC against $\mathsf{AC}^0$ leakage. The core of our construction is a new notion of LTC which we refer to as *strong LTC*. Very informally, a strong $\mathcal{L}$-LTC $C$ for a function $f$ maps an input $\boldsymbol{x}$ to an output $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$, while all wire values of $C$ can be simulated by $\boldsymbol{x}$ and $\boldsymbol{y}$ via a very simple (say, $\mathsf{NC}^0[1]$) function $L_W(\cdot,\cdot)$, such that any leakage query in $\mathcal{L}$ cannot distinguish the real wire values from simulated wire values. This leads to an interesting phenomenon where any leakage function $L \in \mathcal{L}$ on wire values of $C$ can be naturally simulated by a function $L \circ L_W \in \mathcal{L}$ on inputs $\boldsymbol{x}$ and $\boldsymbol{y}$, implying the standard leakage tolerance against $\mathcal{L}$. Surprisingly, while strong $\mathsf{AC}^0$-LTC possesses stronger security guarantee, we will show that it is also easier to realize. We refer readers to Section 3.2 for a more concrete discussion.

An interesting application of our strong $\mathsf{AC}^0$-LTC is to construct LTCs against parity leakage with efficient simulation. In [IS24], the authors construct an LTC against parity leakage leveraging a probing-LTC as a building block. However, the simulator in their construction suffers from exponential running time in the leakage output size. The authors also conjecture that the inefficiency of the simulator might be inherent. Surprisingly, we show that our strong $\mathsf{AC}^0$-LTC, which is also a strong probing-LTC, can be used to construct a strong parity-LTC. This naturally implies a parity-LTC with efficient simulation, proving negative of their conjecture.

**New Framework for Deterministic Stateful LRCs Against Generic Leakage Class.** In [IS25], the authors provide a generic framework of constructing deterministic stateful LRCs against decomposable leakage class, assuming a randomness-efficient LTC against the same leakage class. Very informally, a leakage class $\mathcal{L}$ is decomposable if for any leakage query $L(\cdot) \in \mathcal{L}$ applied on a string $\boldsymbol{x} \in \{0,1\}^n$ and for any decomposition of $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ where each $\boldsymbol{x}_i$ is a sub-vector of $\boldsymbol{x}$, there exist $k$ functions $L_1, \ldots, L_k \in \mathcal{L}$ such that $L(\boldsymbol{x})$ can be simulated by $L_1(\boldsymbol{x}_1), \ldots, L_k(\boldsymbol{x}_k)$. Following this framework, the authors give feasibility results for parity leakage and depth-1 $\mathsf{AC}^0$ leakage.

In this work, we generalize their approach by removing the need of the decomposable property of the leakage class, with the help of a new primitive that we refer to as *global randomness extractor*.

**Deterministic Stateful LRCs Against $k$-CNF Leakage.** Finally, we instantiate our framework for the $k$-CNF leakage class (with $k = O(\log \kappa)$ where $\kappa$ is the security parameter), an important sub-class of depth-2 $\mathsf{AC}^0$ leakage. This is the first feasibility result for deterministic stateful LRC against a non-decomposable leakage class. To achieve this, we construct (a) a global randomness extractor that works for generic $\mathsf{AC}^0$ leakage, and (2) a randomness-efficient LTC against $k$-CNF leakage that only uses polylogarithm many random bits.

# 3 Technical Overview

In this section, we give an overview of our techniques.

## 3.1 Why Previous Techniques Fail

We start by reviewing previous techniques that are relevant to constructing LTCs against $\mathsf{AC}^0$ leakage and discuss why their techniques are not sufficient.

**Upgrading from Stateless LRC to LTC.** In [BIS21], the authors show that the classic ISW-style stateless leakage-resilient circuit [ISW03] (which was only shown to be secure against probing leakage in [ISW03]) achieves stronger resilience against $\mathsf{AC}^0$ leakage. Recall that compared with an LTC, a stateless LRC is equipped with a trusted input encoder and output decoder. Thus, a straightforward idea for constructing $\mathsf{AC}^0$-LTCs is to implement the input encoder and output decoder via $\mathsf{AC}^0$-LTCs and use the ISW-style LRC to carry out the intermediate computation.

To be more concrete, let $\mathcal{L}$ be a leakage class (which can be $\mathsf{AC}^0$) and $(I, C, O)$ be an $\mathcal{L}$-LRC. The goal is to construct an $\mathcal{L}$-LTC $C_I$ of the input encoder $I$ and an $\mathcal{L}$-LTC $C_O$ of the output decoder $O$. Then we wish to prove that $C_O \circ C \circ C_I$ is an $\mathcal{L}$-LTC of the target functionality $f$. Intuitively, the leakage-tolerance property of $C_I$ and $C_O$ can reduce a leakage query applied to the inner wires of $C_I$ and $C_O$ to another leakage query that is only applied to the input and output of $C_I$ and $C_O$, which are just the unprotected input and output $\boldsymbol{x}, \boldsymbol{y}$ and the input and output of $C$. Effectively, we may reduce the original leakage query to one that only accesses $\boldsymbol{x}, \boldsymbol{y}$ and the wires in $C$. Due to the leakage-resilience property of $(I, C, O)$, the wires in $C$ are resilient to any leakage query in $\mathcal{L}$. As a result, we can further reduce the leakage query to one that only accesses $\boldsymbol{x}, \boldsymbol{y}$ as desired.

Unfortunately, the above idea faces two issues:

- First, constructing $\mathcal{L}$-LTCs for the input encoder $I$ and the output decoder $O$ is not easy in general. And this is the case for $\mathcal{L} = \mathsf{AC}^0$ leakage class even if $I$ and $O$ of the ISW-style LRC only involve $\mathsf{XOR}$ gates.
- A more severe issue is that the above intuition in fact does not work. This is because the leakage query $L$ is applied to wires in the composed circuit $C_O \circ C \circ C_I$. However, the leakage-tolerance property of $C_I$ (or $C_O$) can be applied only on a leakage function that only involves wires in $C_I$ (or $C_O$). It is not clear how to decompose a leakage function $L \in \mathsf{AC}^0$ to individual leakage functions applied on $C_I$, $C$, and $C_O$ separately.

**Construction in [IS24] Against Depth-1 $\mathsf{AC}^0$ Leakage.** To overcome the above mentioned issues, the authors in [IS24] observe that the probing leakage-tolerant variant of the ISW-style construction for *parity circuits* also achieves leakage-tolerance against depth-1 $\mathsf{AC}^0$ leakage. Let $\kappa$ be the security parameter. The main insight is that for a depth-1 $\mathsf{AC}^0$ leakage query $L$, say the $\mathsf{OR}$ function of a subset of selected wires, if it involves at least $\kappa$ uniformly random bits, then the leakage result is 1 with overwhelming probability $1-2^{-\kappa}$. Thus, it is sufficient to focus on the case where the selected wires contain at most $\kappa-1$ random bits. The ISW-style construction ensures that in this case all selected wires can be perfectly simulated by a simple function (in $\mathsf{NC}^0[1]$ to be concrete) of the inputs and outputs, thus naturally reducing $L$ to a depth-1 $\mathsf{AC}^0$ leakage query on the inputs and outputs of the circuit. Finally, the authors in [IS24] show how to upgrade such an LTC for parity circuits to one for general circuits.

However, the same idea does not extend to even depth-2 $\mathsf{AC}^0$ leakage: even if a depth-2 $\mathsf{AC}^0$ leakage query $L$ is applied to $\kappa$ random bits, the leakage result may be neither a constant 0 nor a constant 1 with overwhelming probability. As a result, the leakage result cannot be easily simulated even if the selected wires contain $\kappa$ uniform random bits. It is unclear how to prove leakage-tolerance in this case.

### 3.2 Our Solution: Strong LTCs Against $\mathsf{AC}^0$ Leakage

Our idea is inspired by a recent work [Wan25] which constructs LTCs for an even broader leakage class $\mathsf{NC}^1$ but requires auxiliary (non-uniform) randomness. We note that the LTC in their work achieves a stronger leakage-tolerance property: all wire values in their construction can be simulated by simple functions (e.g., just negations) independent of the leakage query $L$ such that the simulated wires can fool any leakage function in $\mathcal{L}$. To be more concrete, there exists a simulator that samples a function $L_W(\cdot,\cdot) \in \mathsf{NC}^0[1]$, which on inputs $\boldsymbol{x}$ and $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$ generates all wire values in $C$, such that the output of $L_W(\boldsymbol{x},\boldsymbol{y})$ is indistinguishable from the real wire values by any leakage function in $\mathcal{L}$. We refer to the function $L_W(\cdot,\cdot)$ as a wire-value function, and refer to such a leakage-tolerant circuit as a *strong LTC*. Note that any leakage function $L \in \mathcal{L}$ applied to all wire values in $C$ naturally reduces to the leakage function $L \circ L_W \in \mathcal{L}$ applied only to the inputs and outputs of $C$, thus implying the standard leakage-tolerance security.

The major benefit of strong LTCs is that the wire-value function $L_W(\cdot,\cdot)$ is sampled independent of the leakage query. Our solution follows the first attempt in Section 3.1 by constructing strong $\mathsf{AC}^0$-LTCs for the input encoder $I$ and the output decoder $O$ and relies on the strong leakage-tolerant property to bypass the second issue.

**Our Framework.** For a bit $b \in \{0,1\}$, we use $\mathsf{PAR}(\kappa, b)$ to denote the set of all length-$\kappa$ additive secret sharings with secret $b$, i.e., the parity of all shares is equal to $b$. In [BIS21], the input encoder $I$ encodes each input bit $\boldsymbol{x}_i$ to a random element in $\mathsf{PAR}(\kappa, \boldsymbol{x}_i)$, and the output decoder $O$ takes as input an element in $\mathsf{PAR}(\kappa, \boldsymbol{y}_i)$ and computes the parity of all shares to obtain $\boldsymbol{y}_i$.

Our construction of strong $\mathsf{AC}^0$-LTCs $C_I$ and $C_O$ for $I$ and $O$ follows from the $\mathsf{AC}^0$-LRCs with leak-free hardware in [FRR+10, MV13]. We will show how to eliminate the need of leak-free hardware later.

- **Strong $\mathsf{AC}^0$-LTC $C_I$ for $I$**: For each input bit $\boldsymbol{x}_i$, sample $\boldsymbol{o}_{\boldsymbol{x}_i} \xleftarrow{\$} \mathsf{PAR}(\kappa, 0)$ by leak-free hardware, compute and output $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i} := (\boldsymbol{o}_{\boldsymbol{x}_i,1} \oplus \boldsymbol{x}_i, \boldsymbol{o}_{\boldsymbol{x}_i,2}, \ldots, \boldsymbol{o}_{\boldsymbol{x}_i,\kappa})$.
- **Strong $\mathsf{AC}^0$-LTC $C_O$ for $O$**: For each output encoding $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \in \mathsf{PAR}(\kappa, \boldsymbol{y}_i)$, sample $\boldsymbol{o}_{\boldsymbol{y}_i} \xleftarrow{\$} \mathsf{PAR}(\kappa, 0)$ by leak-free hardware, compute $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i} = \widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \oplus \boldsymbol{o}_{\boldsymbol{y}_i}$, and then compute the parity of $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$ to obtain the output bit $\boldsymbol{y}_i$.

We prove the strong leakage-tolerance property by constructing wire-value functions $L^I_W(\cdot,\cdot), L^O_W(\cdot,\cdot) \in \mathsf{NC}^0[2]$ that simulate the wires in $C_I$ and $C_O$. [1]For each input bit $\boldsymbol{x}_i$ and its encoding $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}$, $L^I_W(\boldsymbol{x}_i, \widehat{\boldsymbol{w}}_{\boldsymbol{x}_i})$ computes the inner wires $\boldsymbol{o}_{\boldsymbol{x}_i} := (\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i,1} \oplus \boldsymbol{x}_i, \widehat{\boldsymbol{w}}_{\boldsymbol{x}_i,2}, \ldots, \widehat{\boldsymbol{w}}_{\boldsymbol{x}_i,\kappa})$. For each output bit $\boldsymbol{y}_i$ and its encoding $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}$, $L^O_W(\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}, \boldsymbol{y}_i)$ samples a random element in $\mathsf{PAR}(\kappa, \boldsymbol{y}_i)$ as $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$ and then computes the inner wires $\boldsymbol{o}_{\boldsymbol{y}_i} := \widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \oplus \widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$ and the inner wires when reconstructing $\boldsymbol{y}_i$ from $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$.

Now we show that the composition of $C_O \circ C \circ C_I$ is a strong $\mathsf{AC}^0$-LTC by constructing the wire-value function $L_W(\cdot,\cdot) \in \mathsf{NC}^0[1]$ that simulates the wires in $C_O \circ C \circ C_I$. Consider the following hybrids.

- In $\mathsf{Hyb}_0$, we generate all wires following $C_O \circ C \circ C_I$ as in the real world.
- In $\mathsf{Hyb}_1$, we first randomly sample $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}$ from $\mathsf{PAR}(\kappa, \boldsymbol{x}_i)$ for each input bit $\boldsymbol{x}_i$. Then we compute all wires in $C$ taking $\{\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}\}$ as input, and obtain the output encoding $\{\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}\}$. Next, we compute $\boldsymbol{y}_i$ from $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}$. Finally, the inner wires of $C_I$ and $C_O$ are generated by $L^I_W(\boldsymbol{x}_i, \widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}), L^O_W(\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}, \boldsymbol{y}_i)$ for each input bit $\boldsymbol{x}_i$ and output bit $\boldsymbol{y}_i$. The distribution is unchanged.
- In $\mathsf{Hyb}_2$, we replace each $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}$ with a random element in $\mathsf{PAR}(\kappa, 0)$ and compute the wires in $C$ accordingly. Then we use $\boldsymbol{x}$ and $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$ to generate the inner wires of $C_I$ and $C_O$. In this step, we rely on the leakage-resilience property of $(I, C, O)$: for a fixed $\boldsymbol{x}$, given $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$, any leakage function in $\mathsf{AC}^0$ cannot distinguish the real wire values in $C(I(\boldsymbol{x}))$ from the fake wire values in $C(I(\mathbf{0}))$.

Note that the last hybrid implies a $\mathsf{NC}^0[1]$ function $L_W(\cdot,\cdot)$ as required: although $L_W(\cdot,\cdot)$ makes use of $L^I_W(\cdot,\cdot), L^O_W(\cdot,\cdot) \in \mathsf{NC}^0[2]$, note that we explicitly generate $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}, \widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}$. As a result, each output of $L_W(\cdot,\cdot)$ only depends on a single input bit.

**Eliminating Leak-Free Hardware.** It remains to eliminate the use of leak-free hardware. To this end, we construct a strong $\mathsf{AC}^0$-LTC $\mathcal{G}$ (without any input) for generating random elements in $\mathsf{PAR}(\kappa, 0)$. We argue that the strong leakage-tolerance property of our framework preserves after replacing the leak-free hardware by $\mathcal{G}$. Intuitively, this is because all wires in $\mathcal{G}$ can be simulated by a wire-value function $L^{\mathcal{G}}_W(\bot,\cdot) \in \mathsf{NC}^0[1]$. Thus, we can construct a wire-value function for the whole circuit by first using $L_W(\cdot,\cdot)$ to generate all wires in $C_O \circ C \circ C_I$ and then using $L^{\mathcal{G}}_W(\bot,\cdot)$ to generate all wires in $\mathcal{G}$. In Section 5, we directly prove the security of our construction assuming a strong $\mathsf{AC}^0$-LTC for generating random elements in $\mathsf{PAR}(\kappa, 0)$.

In the following, we focus on the instantiation of $\mathcal{G}$. Our construction leverages the stateless $\mathsf{AC}^0$-LRC [ISW03, BIS21] as a building block again:

1. Sample random vectors $\boldsymbol{w}_{\boldsymbol{r}_1}, \ldots, \boldsymbol{w}_{\boldsymbol{r}_\kappa} \xleftarrow{\$} \{0,1\}^\kappa$ and view each vector $\boldsymbol{w}_{\boldsymbol{r}_i}$ as a random encoding of a random bit $\boldsymbol{r}_i$. Then use the stateless $\mathsf{AC}^0$-LRC [ISW03, BIS21] to compute the product of these $\kappa$ encodings. Let $\boldsymbol{w}_p$ denote the output encoding. We have $p = \bigwedge_{i=1}^{\kappa} \boldsymbol{r}_i$.
2. To generate a random element in $\mathsf{PAR}(\kappa, 0)$, sample a random vector $\boldsymbol{w}_u \xleftarrow{\$} \{0,1\}^\kappa$ and view it as a random encoding of a random bit $u$. Then use the stateless $\mathsf{AC}^0$-LRC [ISW03, BIS21] to multiply the two encodings $\boldsymbol{w}_p$ and $\boldsymbol{w}_u$. The output is denoted by $\boldsymbol{o}$.

The correctness comes from a simple observation: if at least one of $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\kappa$ is 0, so does their product $p$, which happens with overwhelming probability. In this case, the output $\boldsymbol{o}$ is also an encoding of 0. We argue for the strong leakage-tolerance property of $\mathcal{G}$ by the following hybrids.

[1]Here we slightly relax the requirement of the strong leakage-tolerance property and allow each output bit of $L^I_W(\cdot,\cdot), L^O_W(\cdot,\cdot)$ to depend on two input bits.

- In $\mathsf{Hyb}_0$, we generate all wires following $\mathcal{G}$. This corresponds to the real world.
- In $\mathsf{Hyb}_1$, we randomly sample each $\boldsymbol{w}_{\boldsymbol{r}_i} \xleftarrow{\$} \mathsf{PAR}(\kappa, 1)$ and each $\boldsymbol{w}_u \xleftarrow{\$} \mathsf{PAR}(\kappa, 0)$. Note that each output encoding $\boldsymbol{o}$ still has parity 0. From the leakage-resilience property of the underlying $\mathsf{AC}^0$-LRC, the distribution of $\mathsf{Hyb}_1$ is indistinguishable from that in $\mathsf{Hyb}_0$ by $\mathsf{AC}^0$. Note that we always have $p = 1$.
- We observe that the $\mathsf{AC}^0$-LRC in [ISW03, BIS21] satisfies the following property: when computing the multiplication of two input encodings $\boldsymbol{w}_a, \boldsymbol{w}_b$ to an output encoding $\boldsymbol{w}_c$, for all fixed $\boldsymbol{w}_a \in \mathsf{PAR}(\kappa, 1)$ and the internal randomness used in the $\mathsf{AC}^0$-LRC, there is a $\mathsf{NC}^0[1]$ function that maps $\boldsymbol{w}_c$ to the rest of the wires in $\mathsf{AC}^0$-LRC (including another input encoding $\boldsymbol{w}_b$). We formalize this property in Lemma 6.2.

  From this property, we may construct the wire-value function $L_W^{\mathcal{G}}(\perp, \boldsymbol{o})$ as follows. We first randomly sample each $\boldsymbol{w}_{\boldsymbol{r}_i} \xleftarrow{\$} \mathsf{PAR}(\kappa, 1)$ explicitly and compute $\boldsymbol{w}_p$ and all internal wires. Then, we randomly sample the internal randomness used in the $\mathsf{AC}^0$-LRC. Finally, we use the $\mathsf{NC}^0[1]$ function obtained from the above property to compute the rest of the wires when multiplying the encodings $\boldsymbol{w}_p$ and $\boldsymbol{w}_u$.

Note that our construction does not achieve perfect correctness when all $\boldsymbol{r}_i = 1$, which happens with negligible probability. In Appendix A , we prove that the lack of perfect correctness is inherent in constructing strong LTCs against any non-trivial leakage class by refining a theorem presented in [Wan25].

**Application: Parity-LTC with Efficient Simulation.** One application of our strong $\mathsf{AC}^0$-LTC is to construct a parity-LTC with efficient simulation. In [IS24], the authors construct a leakage-tolerant circuit against $t$-parity leakage, i.e., the leakage query may output $t$ bits and each output bit is a parity function of a subset wire values chosen by the adversary. However, the simulator in their construction requires running time exponential in the leakage size $t$. This is due to the need of finding the first $t$ shortest parity functions in the span of the given $t$ parity functions.

Very informally, the authors in [IS24] construct a $(t, k)$-parity-to-probing compiler which converts a target circuit $\widetilde{C}$ to another circuit $C$ such that any $t$-parity query applied to the wires of $C$ can be simulated by probing $k$ wires in $\widetilde{C}$. When $\widetilde{C}$ is implemented by an LTC against $k$-probing leakage, they obtain an LTC against $t$-parity leakage. However, the security of a standard probing-LTC $\widetilde{C}$ requires providing in clear the $k$ wires chosen by the leakage function. Finding these $k$ wires requires solving the above mentioned shortest parity function problem.

We note that our strong $\mathsf{AC}^0$-LTC is also a strong probing-LTC. When instantiating $\widetilde{C}$ from our construction, we can simulate all wire values in $\widetilde{C}$ via a $\mathsf{NC}^0[1]$ wire-value function *without knowing the $k$ wires chosen by the leakage function.* In addition, with our instantiation, we show that all wire values in $C$ (i.e., after applying the $(t, k)$-parity-to-probing compiler on $\widetilde{C}$) can also be simulated by a $\mathsf{NC}^0[1]$ wire-value function $L_W$. As a result, we obtain a strong $t$-parity-LTC which naturally implies efficient simulation, avoiding the need of finding the first $t$ shortest parity functions.

### 3.3 A Generic Framework for Deterministic Stateful LRCs Against Non-Decomposable Leakage

In [IS25], the authors propose the notion of deterministic stateful LRCs, which only require fresh randomness in the first invocation, and provide a generic framework of constructing deterministic

stateful LRCs against *decomposable* leakage class. Another contribution of our work is to extend their framework to support non-decomposable leakage class, such as $\mathsf{AC}^0$.

**Prior Approach.** The idea in [IS25] is to use a cryptographically secure PRG to expand a short seed in each invocation and use the output as the internal randomness required in the next invocation. To be more concrete,

- In the trusted setup process, encode the initial state $\boldsymbol{s}_0$ and a short random seed $\mathsf{Seed}_0$ to $\widehat{\boldsymbol{s}_0}$ using an encoding scheme that is resilient to leakage functions in $\mathcal{L}$.
- Then, in the $i$-th invocation, use an $\mathcal{L}$-LTC circuit to compute the following functionality:
  - Decode the encoded state $\widehat{\boldsymbol{s}_{i-1}}$, and parse it to obtain the real state $\boldsymbol{s}_{i-1}$ and the random seed $\mathsf{Seed}_{i-1}$.
  - Compute the stateful function $f[\boldsymbol{s}_{i-1}]$ on public input $\boldsymbol{x}_i$ to obtain public output $\boldsymbol{y}_i$ and updated state $\boldsymbol{s}_i$.
  - Let $\mathsf{PRG}$ be a cryptographic pseudorandom generator. Use $\mathsf{PRG}$ (on input $\mathsf{Seed}_{i-1}$) to compute the random seed $\mathsf{Seed}_i$ and internal randomness $\mathsf{Rtape}_i$ for the $\mathcal{L}$-LTC used in the next invocation.
  - Finally, output $\boldsymbol{y}_i$, and set the secret state $\widehat{\boldsymbol{s}_i}$ to be an encoding of $(\boldsymbol{s}_i \parallel \mathsf{Seed}_i)$.

However, the concise idea described above faces two challenges: (1) The $\mathsf{PRG}$ is computed within each invocation via an $\mathcal{L}$-LTC whose randomness comes from the output in the last invocation. This requires the use of a *randomness-efficient* $\mathcal{L}$-LTC (say the amount of required randomness is polylogarithmic in the circuit size); otherwise, we cannot produce the same amount of internal randomness for the next invocation. (2) A more severe issue is that the internal randomness $\mathsf{Rtape}_i$ used in the $(i+1)$-th invocation is the output of the $i$-th invocation. Given the leakage result in the $i$-th invocation, $\mathsf{Rtape}_i$ is no longer uniformly random. This would break the security of the $\mathcal{L}$-LTC in the $(i+1)$-th invocation, as analyzed in [IS25].

Ishai and Song [IS25] resolve the first issue by listing it as a requirement and provide concrete constructions of randomness-efficient LTCs against depth-1 $\mathsf{AC}^0$ leakage and parity leakage. As for the second issue, the key observation is that $\mathsf{Rtape}_i$ can be viewed as imperfect randomness under leakage in the $i$-th invocation. Their idea is to apply a randomness extractor on $\mathsf{Rtape}_i$ to obtain a fresh $\mathsf{Rtape}'_i$ so that it is independent of the leakage in the last invocation. However, an additional issue comes with this idea: the inner wires of the randomness extractor are also under attack in the $(i+1)$-th invocation. To resolve this issue,

- They use a deterministic randomness extractor which only extracts a single random bit each time. Then $\mathsf{Rtape}_i$ is separated into small pieces $\{\mathsf{Rtape}_{i,j}\}_{j=1}^{n}$ (say $n := |\mathsf{Rtape}'_i|$ is the number of random bits required in the next invocation) and each time a single random bit is extracted from each piece.

  In this way, the inner wires of the randomness extractor can be expressed by a $\mathsf{NC}^0[1]$ function on the output bit. Thus attacking the inner wires of the randomness extractor does not give any advantage than only attacking the output wire of the randomness extractor.
- To implement the above idea, the simulator needs to reversely sample a pair of strings $\boldsymbol{r}_0, \boldsymbol{r}_1$ for each piece of $\mathsf{Rtape}_{i,j}$ such that
  1. $\mathsf{Ext}(\boldsymbol{r}_0) = 0$ and $\mathsf{Ext}(\boldsymbol{r}_1) = 1$. In this way, the inner wires of the randomness extractor can be computed as a $\mathsf{NC}^0[1]$ function from its output $\boldsymbol{b}_{i,j}$ by $\mathsf{Ext}(\boldsymbol{r}_0 \cdot (1 - \boldsymbol{b}_{i,j}) + \boldsymbol{r}_1 \cdot \boldsymbol{b}_{i,j})$.

2. Both the cases $\mathsf{Rtape}_{i,j} = \boldsymbol{r}_0$ and $\mathsf{Rtape}_{i,j} = \boldsymbol{r}_1$ are consistent with the leakage result in the $i$-th invocation.

However, the leakage query in the $i$-th invocation is applied to all wires, not restricting to wires in $\mathsf{Rtape}_{i,j}$. This is the place where decomposability is required: the leakage query in the $i$-th invocation is first decomposed to several leakage queries, including one for each piece $\mathsf{Rtape}_{i,j}$. Then it is sufficient to focus on the leakage result of each piece individually.

In [IS25], the authors show how to instantiate such randomness extractors against depth-1 $\mathsf{AC}^0$ leakage and parity leakage.

**Global Randomness Extractor to the Rescue.** To support the non-decomposable leakage class, a natural idea is to require a stronger property of the randomness extractor $\mathsf{Ext}$. Informally speaking, given a leakage function $L \in \mathcal{L}$ that takes $n$ strings as input, there is a simulator that outputs $n$ pairs of strings $(\boldsymbol{r}_{j,0}, \boldsymbol{r}_{j,1})$, such that:

- $\mathsf{Ext}(\boldsymbol{r}_{j,0}) = 0$ and $\mathsf{Ext}(\boldsymbol{r}_{j,1}) = 1$;
- For any sequence $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \{0,1\}^n$, applying $L$ on $(\boldsymbol{r}_{j,\boldsymbol{b}_j})_{j\in[n]}$ gives the same leakage result.

With such a randomness extractor, we may reversely sample $\{(\boldsymbol{r}_{j,0}, \boldsymbol{r}_{j,1})\}_{j=1}^n$ only given the global leakage result applied to $\mathsf{Rtape}_i$. However, this is again not sufficient due to the same reason: the leakage query in the $i$-th invocation is applied to all wires, not restricted to wires in $\mathsf{Rtape}_i$.

To be more concrete, relying on the leakage tolerance of $\mathcal{L}$-LTCs, the leakage query $L$ in the $i$-th invocation can be reduced to a leakage query $L'$ that is only applied to the input and output of the $i$-th invocation, i.e., $(\widehat{\boldsymbol{s}_{i-1}}, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_i, \widehat{\boldsymbol{s}}_i, \mathsf{Rtape}_i)$. In the ideal world, the simulator will receive $\boldsymbol{x}_i, \boldsymbol{y}_i$ and replace $\widehat{\boldsymbol{s}_{i-1}}, \widehat{\boldsymbol{s}}_i$ by explicitly generated fake encodings of 0. In this way, the simulator can obtain the leakage function $L''$ that is applied only to $\mathsf{Rtape}_i$. Then we may sample the leakage result relying on the property of the global randomness extractor introduced above.

The problem occurs when proving the security of this construction. Following [IS25], we wish to use hybrid arguments and replace one encoded state with a fake state each time. Consider $\mathsf{Hyb}_{i-1}$ where $\widehat{\boldsymbol{s}_{i-1}}$ has been replaced by a fake encoded state but $\widehat{\boldsymbol{s}}_i$ remains to be the real encoded state. To argue the indistinguishability between $\mathsf{Hyb}_{i-1}$ and $\mathsf{Hyb}_i$, we want to use the same strategy to simulate the leakage result in both cases: derive a leakage function $L''$ applied on $\mathsf{Rtape}_i$ and then simulate the leakage result. However, in $\mathsf{Hyb}_{i-1}$, the derived leakage function $L''$ would depend on the real encoded state $\widehat{\boldsymbol{s}}_i$. It is not clear how to argue the indistinguishability in this case.

To resolve this issue, we modify the functionality of the $\mathcal{L}$-LTC in the $i$-th invocation to output a random preimage of $\widetilde{\boldsymbol{s}}_i \xleftarrow{\$} \mathsf{Ext}^{-1}(\widehat{\boldsymbol{s}}_i)$. Then in the next invocation, we will decode $\widehat{\boldsymbol{s}}_i$ by applying $\mathsf{Ext}$ on $\widetilde{\boldsymbol{s}}_i$, just as extracting the randomness $\mathsf{Rtape}'_i$ from $\mathsf{Rtape}_i$. With this fix, it is sufficient to obtain a leakage query $L''$ applied on $(\widetilde{\boldsymbol{s}}_i, \mathsf{Rtape}_i)$, which is possible in both $\mathsf{Hyb}_{i-1}$ and $\mathsf{Hyb}_i$.

We refer the readers to Section 7 for more details.

### 3.4 Deterministic Stateful LRCs Against $k$-CNF Leakage

Finally, we briefly introduce the instantiation of our generic framework for deterministic stateful LRCs against $k$-CNF leakage for $k = O(\log \kappa)$, a sub-class of depth-2 $\mathsf{AC}^0$ leakage. Concretely, a $k$-CNF leakage function is an $\mathsf{AND}$ function of several clauses where each clause is an $\mathsf{OR}$ function of at most $k$ input variables or their negations.

For the global randomness extractor, our instantiation actually works for $\mathsf{AC}^0$ leakage. Our construction requires careful use of the multi-switching Hastad lemma [IMP12, Has14, ST22]. We refer the readers to Section 8.1 for an overview of our approach.

For the randomness-efficient LTCs, we only manage to provide solutions for $k$-CNF leakage. We give an overview of our construction below.

**Construction Sketch.** At a high level, our randomness efficient $k$-CNF-LTC $C'$ of $f$ can be divided into two parts: (1) an internal circuit $\mathsf{Gen}$ that generates pseudorandom randomness $R$ resilient to $k$-CNF functions; (2) an external circuit $C$ that is a secure $k$-CNF-LTC implementation of $f$ using $R$ as internal randomness.

This simple construction can be derived from a straightforward approach of building randomness-efficient $\mathcal{L}$-LTCs: use an $\mathcal{L}$-LTC to compute functionality $f$, then use an $\mathcal{L}$-resilient PRG to shrink the randomness of the $\mathcal{L}$-LTC scheme. However, we must solve two intractable problems:

1. For the external circuit $C$, the leakage-tolerance against $k$-CNF may not hold when the internal randomness is replaced by a pseudorandom source $R$ resilient to $k$-CNF functions. This is because within the computation of $C$, it may involve an arbitrary computation on the random tape $R$. Therefore the wire values of $C$ may not be a $k$-CNF function of $R$. As a consequence, even the simple probing leakage of $C$ may distinguish $R$ from uniformly random source.
2. The $k$-CNF leakage function can not only touch the wires of $C$, but also touch the wires of the pseudorandom generator $\mathsf{Gen}$. Therefore, we must protect the inner wire values of $\mathsf{Gen}$, say via a leakage-tolerant implementation. Making it even worse, since $k$-CNF is not decomposable, even a leakage-tolerant implementation of $\mathsf{Gen}$ appears to be insufficient due to the incapability of separating the leakage query to one that is only applied to the inner wires of $\mathsf{Gen}$.

To resolve the first problem, we adapt the approach of randomness-efficient LTC in [IS25] against depth-1 $\mathsf{AC}^0$ leakage, which uses multiple independent PRG instances so that each wire in the external circuit $C$ only depends on a single output bit of the output of each PRG. We refer the readers to Appendix G.2 for more details.

As for the second problem, a straightforward solution is to implement a strong LTC for $\mathsf{Gen}$ as what we did when removing the leak-free hardware in our strong $\mathsf{AC}^0$-LTC. Unfortunately, we do not know how to construct such a strong LTC for $\mathsf{Gen}$. Our idea is to first construct a standard LTC for $\mathsf{Gen}$ against $k$-CNF leakage. We will then discuss how to overcome the second problem.

**LTC for $\mathsf{Gen}$ Against $k$-CNF Leakage.** Our idea is inspired by the following observation: let $F : \{0,1\}^m \to \{0,1\}^n$ be a linear function, $\boldsymbol{u}$ be a random vector in $\{0,1\}^m$, and $\boldsymbol{x} = F(\boldsymbol{u})$. For any $k$-CNF function $L$, there exists a simulator which may simulate $L(\boldsymbol{x})$ by adaptively probing $O(2^k \cdot k\kappa)$ values in $\boldsymbol{x}$ such that the output is indistinguishable from $L(\boldsymbol{x})$ even given $\boldsymbol{x}$.

To explain this observation, we take $k = 3$ as an example. In this case, each clause in $L$ is an $\mathsf{OR}$ function of 3 input variables or their negations. To simulate $L(\boldsymbol{x})$,

1. The simulator starts from the first clause whose result is not a fixed constant 1 (if there is no such clause, then the simulator can already output $L(\boldsymbol{x}) = 1$ as the result), which can be locally tested by solving linear equations. Say this clause is in the form of $(\boldsymbol{x}_i \vee \boldsymbol{x}_j \vee \boldsymbol{x}_\ell)$. The simulator simply probes these three wires and computes the result.
2. If the result of this clause is a constant 0, then this means that the output of $L(\boldsymbol{x})$ is 0 since $L$ computes an $\mathsf{AND}$ function of all clauses. If the result of this clause is a constant 1, then go back to Step 1.

We argue that with overwhelming probability, the number of probings is bounded by $O(2^k \cdot k\kappa) = O(\kappa)$. Intuitively, if a clause is not a constant 1 given all previous probed values, then the probability that $\boldsymbol{x}_i = \boldsymbol{x}_j = \boldsymbol{x}_\ell = 0$ is not 0. Since $\boldsymbol{x}$ is a linear function $F$ applied to a random vector $\boldsymbol{u}$, the probability that all three variables are equal to 0 given all probed values is at least $1/2^k = 1/8$. Thus, either the simulator can compute $L(\boldsymbol{x}) = 1$ after determining fewer than $O(2^k \cdot \kappa) = O(\kappa)$ clauses, or with overwhelming probability, at least one clause would output 0 and the simulator terminates with $L(\boldsymbol{x}) = 0$. In both cases, the simulator only needs to query $O(2^k \cdot k\kappa) = O(\kappa)$ wires with overwhelming probability.

Utilizing the above observation, our construction works as follows.

- We first choose a linear PRG $G$ whose output is indistinguishable from uniform random source by $k$-CNF functions. In our construction, we use the Nisan-Wigderson [NW94] PRG. Note that all wires in the computation of $G$ is a linear function $F$ applied on the input random seed $\boldsymbol{u}$. From our above observation, any $k$-CNF leakage query applied to the wires in $G$ can already be simulated by adaptively probing $\eta - 1 := O(2^k \cdot k\kappa)$ queries to the wires in $G$.
- Then, we apply the trick in [ISW03] to construct a probing-LTC for $G$: the circuit $\mathsf{Gen}$ computes $\eta$ copies of $G$ on fresh input randomness $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_\eta$ and then computes the output as $R := G(\boldsymbol{u}_1) \oplus G(\boldsymbol{u}_2) \oplus \ldots \oplus G(\boldsymbol{u}_\eta)$.

  Notice that $\mathsf{Gen}$ is still a linear circuit and effectively, it computes $G(\bigoplus_{i=1}^{\eta} \boldsymbol{u}_i)$ as desired. In the same time, any $\eta - 1$ wires in $\mathsf{Gen}$ can be simulated by $\eta - 1$ output wires of $\mathsf{Gen}$. To see this, first notice that there exists at least one copy of $G(\boldsymbol{u}_t)$ whose inner wire is not probed. We may first randomly generate the rest of $\eta - 1$ copies of $G(\boldsymbol{u}_i)$ $(i \neq t)$, which allows us to simulate probings that are inner wires of $G(\boldsymbol{u}_i)$. Then every inner wire during the computation of $R := G(\boldsymbol{u}_1) \oplus G(\boldsymbol{u}_2) \oplus \ldots \oplus G(\boldsymbol{u}_\eta)$ can be simulated by probing the corresponding output bit in $R$.

As a result, we obtain a weaker LTC $\mathsf{Gen}$ where any $k$-CNF leakage query can be reduced to $O(2^k \cdot k\kappa)$ adaptive probings of the output of $\mathsf{Gen}$.

**Handling Global $k$-CNF Query.** As we discussed in the construction sketch, an LTC for $\mathsf{Gen}$ is still not sufficient since the $k$-CNF query $L$ is applied to the wires in the whole circuit $C' = (\mathsf{Gen}, C)$. It is not clear how to separate $L$ into two leakage queries that are applied to $\mathsf{Gen}$ and $C$ independently.

To address this issue, we attempt to use the same strategy to simulate the $k$-CNF leakage query $L$. The difficulty occurs in the first step where the simulator wants to find the first clause that is not a constant 1. This time, a clause may contain wires in $C$ that are relevant to the input of $C$, which is not known to the simulator, making it infeasible to decide whether the result of a clause is a constant 1 locally.

For this, we first fix all wires in $C$. Note that in this case the output $R$ of $\mathsf{Gen}$ is also fixed. Our solution is to find the first clause that is not a constant 1 by making $k$-CNF query to the wires in $C$. We again take $k = 3$ as an example and consider a clause in the form of $(\boldsymbol{x}_i \vee \boldsymbol{x}_j \vee \boldsymbol{x}_\ell)$. Suppose that $\boldsymbol{x}_i$ is an inner wire of some copy of $G(\boldsymbol{u}_i)$, $\boldsymbol{x}_j$ is an inner wire when computing the $j$-th bit of $R := G(\boldsymbol{u}_1) \oplus G(\boldsymbol{u}_2) \oplus \ldots \oplus G(\boldsymbol{u}_\eta)$, and $\boldsymbol{x}_\ell$ is an inner wire in $C$. Our goal becomes to test whether it is possible that $\boldsymbol{x}_i = \boldsymbol{x}_j = \boldsymbol{x}_\ell = 0$.

We note that the distribution of $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ only depends on $R_j$. Let $H_{R_j}(b) = 1$ indicates that it is possible that both $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are 0 when $R_j = b$, and $H_{R_j}(b) = 0$ otherwise. Note that $H_{R_j}(0)$ and $H_{R_j}(1)$ can be locally computed by the simulator. Then we may create a 3-CNF $q = ((H_{R_j}(0) \oplus 1) \vee (R_j \oplus 1) \vee \boldsymbol{x}_\ell) \bigwedge ((H_{R_j}(1) \oplus 1) \vee R_j \vee \boldsymbol{x}_\ell)$ that involves only wires in $C$. If the result is 0,

this means that $H_{R_j}(b) = 1, R_j = b, \boldsymbol{x}_\ell = 0$ for some $b \in \{0,1\}$, in which case it is possible that $\boldsymbol{x}_i = \boldsymbol{x}_j = \boldsymbol{x}_\ell = 0$. Thus the simulator may query $q$ to learn whether the result of the current clause is a constant 1.

However, to find the first clause whose result is not a constant 1, we may have to make a 3-CNF query for each clause in the worst case, which would be too much. We address this by using a binary search:

- We first create a 3-CNF query for each clause and query the $\mathsf{AND}$ of all queries, which is still a 3-CNF query. If the result is 1, it means that all clauses are constant 1. In this case, the leakage result would be 1. Otherwise, there exists $i^\star$ such that the result of the $i^\star$-th clause is not a constant 1.
- To locate $i^\star$, we perform a binary search. Each time, we query the $\mathsf{AND}$ of the queries associated with the first half of the remaining clauses. If the result is 1, then $i^\star$ is in the second half of the remaining clauses. Otherwise, $i^\star$ is in the first half of the remaining clauses. This step is repeated until $i^\star$ is located.

In this way, finding the first clause whose result is not a constant 1 can be done via $O(\log \kappa)$ $k$-CNF queries to $C$. After determining such a clause, we simulate all wires in this clause by probing wires in $C$ (and wires in $R$) and check whether the result is 0. Following the same analysis, with overwhelming probability, we may simulate the leakage result by making at most $O(2^k \cdot \kappa(k + \log \kappa))$ adaptive $k$-CNF queries to wires in $C$. When $C$ is instantiated by an LTC against (adaptive) $k$-CNF leakage, we obtain a randomness-efficient LTC against $k$-CNF leakage.

# 4 Preliminaries

**Notation.** In this work, we consider the standard arithmetic circuit model where a circuit consists of only $\mathsf{XOR}$ gates and $\mathsf{AND}$ gates. For a circuit $C$, the evaluation of $C$ starts with input wires $\boldsymbol{x}$ and random wires $R$ that are sampled from a specific distribution (which may not necessarily be the uniform distribution), and then computes each gate following the topological order of the circuit. We denote such an evaluation as $C(\boldsymbol{x}; R)$, and $C(\boldsymbol{x})$ when $R$ is uniformly sampled.

We will use the notation $W[C(\boldsymbol{x}; R)]$ to denote the set of wire values inside $C$ with input wires $\boldsymbol{x}$ and random wires $R$, and $W[C(\boldsymbol{x})]$ to denote the set of random variables of all wire values inside $C$ with input wires $\boldsymbol{x}$ when $R$ is uniformly sampled. If the input $\boldsymbol{x}$ and the random tape $R$ are clear in the context, we may abbreviate the notation to $W[C]$.

we use $\kappa$ to denote the security parameter. For a set $S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ of bits, we use $\mathsf{parity}(S)$ to denote the parity of its elements, that being to say $\mathsf{parity}(S) = \bigoplus_{i \in S} \boldsymbol{x}_i$. Then we define $\mathsf{PAR}(n, b)$ as the collection of all $n$-bit additive encodings of $b$:

$$\mathsf{PAR}(n, b) := \{(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}, \mathsf{parity}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})) : (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) \leftarrow \{0,1\}^{n-1}\}.$$

For a given vector $\boldsymbol{x}$ and an index set $I = (i_1, i_2, \ldots, i_k)$, we use $\boldsymbol{x}_I$ to abbreviate the vector $(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_k})$.

## 4.1 Circuit Complexity

We use $\mathsf{NC}^0[c]$ for the family of all multi-input, multi-output Boolean functions in which every output depends on at most $c$ inputs, $\mathsf{AC}^0(d, s, m)$ for the function family that uses unbounded fan-in $\mathsf{AND}$-$\mathsf{OR}$-$\mathsf{NOT}$ gates with a circuit depth at most $d$, a size at most $s$, and an output size at most $m$. If the output length is $m = 1$ the variable $m$ is left out of the notation.

We also consider the following important circuit family:

- **$k$-probing**: all $k$-projection functions that output at most $k$ input bits.
- **$k$-CNF and $k$-DNF**: all conjunctive forms and disjunctive forms of the input bits and their negations, such that each clause has a width (i.e. number of variables) at most $k$.
- **$t$-parity**: parity queries of $t$ subsets of input bits and their negations. Formally speaking, $t$-parity consists of the circuits that output $(b_i \oplus \mathsf{parity}(\boldsymbol{x}_{W_i}))_{i\in[t]}$ over subsets $W_1, \ldots, W_t$ and constants $b_1, \ldots, b_t \in \{0,1\}$.

We say that a circuit family $\mathcal{L}$ is *closed under restriction and negation* if $\mathcal{L} \circ \mathsf{NC}^0[1] \subseteq \mathcal{L}$. All the aforementioned circuit families are closed under restriction and negation. In general, we have the following fact about the composition between $\mathsf{AC}^0$ and $\mathsf{NC}^0$:

**Fact 4.1.**

$$\mathsf{AC}^0(d,s,m) \circ \mathsf{NC}^0[c] \subseteq \begin{cases} \mathsf{AC}^0(d,s,m), & \text{if } c = 1 \\ \mathsf{AC}^0(d+1, 2^c s, m), & \text{if } c > 1. \end{cases}$$

We cite the literature on the circuit lower bound of (non-adaptive) $\mathsf{AC}^0$ circuits:

**Lemma 4.2** ( [BIS21, Corollary 3]). *For any constant $d \in \mathbb{N}$, there exists some constant $c_d > 0$ such that $(\mathsf{PAR}(n,0), \mathsf{PAR}(n,1))$ are $\epsilon = 2^{-c_d n/(\log s)^{d-1}}$-indistinguishable by $\mathsf{AC}^0\left(d, s, c_d n/(\log s)^{d-1}\right)$.*

In this work, we also generalize the non-adaptive indistinguishability result to the adaptive regime. We use *$t$-adaptive circuits over function family $\mathcal{L}$* ($\mathcal{L}^t$ for short) to denote (possibly randomized) circuits that adaptively perform at most $t$ numbers of $\mathcal{L}$ queries during the computation.

## 4.2 Leakage-Resilience and Leakage-Tolerance

In this section, we recall some basic definitions about leakage-resilience and leakage-tolerance.

**Definition 4.3** ($\mathcal{L}$-Resilient Encoding)**.** We say that a bit encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ (where $\mathsf{Enc}$ is randomized and $\mathsf{Dec}$ is deterministic) is an $(\mathcal{L}, \epsilon)$-resilient encoding scheme, if it satisfies:

- *Correctness*: for all $b \in \{0,1\}$,
$$\Pr[\mathsf{Dec}(\mathsf{Enc}(b)) = b] = 1.$$
- *Leakage-resilience*: for all leakage function $L \in \mathcal{L}$, the following two distributions are $\epsilon$-close:
$$\{L(\mathsf{Enc}(0))\} \approx_\epsilon \{L(\mathsf{Enc}(1))\}.$$

We simply say that $(\mathsf{Enc}, \mathsf{Dec})$ is an $\mathcal{L}$-resilient encoding scheme if $\epsilon = \mathsf{negl}(\kappa)$.

**Definition 4.4** (Stateless Leakage-Resilient Circuit)**.** Given a (possibly randomized) function $f$, a stateless leakage-resilient circuit for $f$ consists of randomized circuits $(I, C, O)$ such that:

- **Input Encoder $I$.** On an input string $\boldsymbol{x}$, outputs an encoded input $\widehat{\boldsymbol{x}}$.
- **Transformer $C$.** On an encoded input $\widehat{\boldsymbol{x}}$, outputs an encoded output $\widehat{\boldsymbol{y}}$.
- **Output Decoder $O$.** On an encoded output $\widehat{\boldsymbol{y}}$, decodes output $\boldsymbol{y}$.

We say that $C$ is an $(\mathcal{L}, \epsilon)$-leakage-resilient circuit implementation of $f$ if it satisfies the following two requirements:

*Correctness* is satisfied if for all input $\boldsymbol{x}$, we have

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

*Non-adaptive (resp. $m$-adaptive) leakage-resilience* is satisfied if for all inputs $\boldsymbol{x}_0, \boldsymbol{x}_1$ and all leakage functions $L \in \mathcal{L}$ (resp. any $m$-adaptive circuit over $\mathcal{L}$), the following two distributions are $\epsilon$-indistinguishable:

$$L(W[C(I(\boldsymbol{x}_0))] \approx_\epsilon L(W[C(I(\boldsymbol{x}_1))]).$$

If the error parameter $\epsilon$ is a negligible function of $\kappa$, we abbreviate the notation to stateless $\mathcal{L}$-LRC.

**Definition 4.5** (Stateful Function)**.** A stateful function $f$ is a circuit with the following extensions:

- $f$ may contain *memory cells*, which are stateful gates with fan-in 1. On each invocation, a memory cell just outputs the previous input to the gate and stores the current input.
- $f$ may contain cycles where each cycle at least traverses one memory cell.

We use $(\boldsymbol{s}', \boldsymbol{y}) \leftarrow f[\boldsymbol{s}](\boldsymbol{x})$ to refer to a stateful function $f$ with state $\boldsymbol{s}$ of memory cells, takes the public input $\boldsymbol{x}$, computes the public output $\boldsymbol{y}$ and updates the memory cells to be $\boldsymbol{s}'$ at each invocation.

**Definition 4.6** (Stateful Leakage-Resilient Circuit)**.** Given a (possibly randomized) stateful function $f$, a stateful $\mathcal{L}$-leakage-resilient circuit implementation of $f$ consists of a randomized circuit $\mathsf{Setup}$ and a randomized stateful circuit $C$ such that:

- **Setup Algorithm** $\mathsf{Setup}$**.** On an initial state $\boldsymbol{s}_0$, outputs an encoded state $\widehat{\boldsymbol{s}}_0$.
- **Transformer** $C$**.** On an encoded state $\widehat{\boldsymbol{s}}$ and a public input $\boldsymbol{x}$, outputs an updated encoded state $\widehat{\boldsymbol{s}'}$ and a public output $\boldsymbol{y}$.

We say that the stateful $\mathcal{L}$-leakage-resilient circuit implementation of polynomial-sized $f$ satisfies *non-adaptive (resp. $m$-adaptive) computational leakage-resilience* if, for all polynomial query bounds $q = q(\kappa)$, there exists some stateful simulator $\mathsf{Sim}$, such that for all initial states $\boldsymbol{s}_0$ and all P.P.T. adversary $\mathcal{A}$, the output distributions of the following two experiments are computationally indistinguishable:

$$\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L},q,\boldsymbol{s}_0}(1^\kappa) \approx_C \mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L},q,\boldsymbol{s}_0}(1^\kappa).$$

**Definition 4.7** (Leakage-Tolerant Circuit)**.** Given a (possibly randomized) function $f$, an $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-leakage-tolerant circuit implementation of $f$ over $(\mathcal{L}_1, \mathcal{L}_2)$ is a randomized circuit $C$ satisfying leakage-tolerance property:

*Non-adaptive (resp. $m$-adaptive) leakage-tolerance* is satisfied if there exists some stateful simulator $\mathsf{Sim}$, such that for any adversary $\mathcal{A}$ and input $\boldsymbol{x}$, the output distributions of the following two experiments are $\epsilon$-indistinguishable:

$$\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa) \approx_\epsilon \mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa).$$

Note the $\epsilon$-leakage-tolerance property has already implied correctness of $1 - \epsilon$. If $\epsilon$ is a negligible function of $\kappa$, we may abbreviate the notation to $(\mathcal{L}_1, \mathcal{L}_2)$-LTC or $\mathcal{L}_1$-LTC if specifying $\mathcal{L}_2$ is not important.

**Experiment** $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L},q,\boldsymbol{s}_0}(1^\kappa)$

- $\widehat{\boldsymbol{s}_0} \leftarrow \mathsf{Setup}(\boldsymbol{s}_0)$
- For $i$ in $1, 2, \ldots, q$:
  - $\boldsymbol{x}_i \leftarrow \mathcal{A}$
  - $(W_i, \widehat{\boldsymbol{s}_i}, \boldsymbol{y}_i) \leftarrow C[\widehat{\boldsymbol{s}_{i-1}}](\boldsymbol{x}_i)$
  - **non-adaptive:**
    - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}$
    - $L_i(W_i) \rightarrow \mathcal{A}$
  - $m$**-adaptive:** for $j$ in $1, 2, \ldots, m$:
    - $L_{i,j} \leftarrow \mathcal{A}$ where $L_{i,j} \in \mathcal{L}$
    - $L_{i,j}(W_i) \rightarrow \mathcal{A}$
  - $\boldsymbol{y}_i \rightarrow \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in[q]}$

**Experiment** $\mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L},q,\boldsymbol{s}_0}(1^\kappa)$

- For $i$ in $1, 2, \ldots, q$:
  - $\boldsymbol{x}_i \leftarrow \mathcal{A}$
  - $(\boldsymbol{s}_i, \boldsymbol{y}_i) \leftarrow f[\boldsymbol{s}_{i-1}](\boldsymbol{x}_i)$
  - $(\boldsymbol{x}_i, \boldsymbol{y}_i) \rightarrow \mathsf{Sim}$
  - **non-adaptive:**
    - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}$
    - $\mathsf{Sim}(L_i) \rightarrow \mathcal{A}$
  - $m$**-adaptive:** for $j$ in $1, 2, \ldots, m$:
    - $L_{i,j} \leftarrow \mathcal{A}$ where $L_{i,j} \in \mathcal{L}$
    - $\mathsf{Sim}(L_{i,j}) \rightarrow \mathcal{A}$
  - $\boldsymbol{y}_i \rightarrow \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in[q]}$

Figure 1: Experiments of Stateful LRC

We say that a leakage-tolerant circuit implementation of $f$ additionally achieves *perfect correctness*, if

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

The leakage-tolerant circuit implementation of $f$ is said to be randomness-efficient, if its randomness complexity $\mathsf{RC}(|f|, \kappa)$ is $\mathsf{poly}(\log |f|, \kappa)$ plus the randomness complexity of $f$.

**Remark 4.8.** Readers may notice that our security notion of LTC is stronger than [IS24, IS25] in the sense of output-dependency: the adversary in our work can adaptively choose the leakage function $L$ according to the function output $\boldsymbol{y}$ whereas in [IS24, IS25] the leakage function $L$ is independent of $\boldsymbol{y}$.

On the one hand, for deterministic functionalities the two security notions are equivalent, since the output $\boldsymbol{y}$ is uniquely determined by the public input $\boldsymbol{x}$. On the other hand, we provide a counterexample in Appendix D , showing that output-independent security does not necessarily imply output-dependent security for randomized functionality.

We also consider a stronger notion of leakage-tolerance, which is already used in [Wan25] to define AI-LTC (auxiliary input leakage-tolerant circuits). Under this definition, the simulator directly computes each wire value as a simple function $L_W(\cdot, \cdot)$ of the circuit input $\boldsymbol{x}$ and output $\boldsymbol{y}$ which does not depend on any specific leakage query $L$. In this way, any leakage function $L$ of the wire values can be rewritten as another leakage function $L \circ L_W(\boldsymbol{x}, \boldsymbol{y})$.

**Definition 4.9** (Strong Leakage-Tolerant Circuit)**.** For (possibly randomized) function $f$, a strong $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-leakage-tolerant circuit implementation $C$ of $f$ is a leakage-tolerant circuit implementation of $f$ replacing the leakage-tolerance property with the following:

*Strong leakage-tolerance* is satisfied if there exists some simulator $\mathsf{Sim}$ that outputs $L_W \in \mathcal{L}_2$ of $W[C]$ on inputs $(\boldsymbol{x}, \boldsymbol{y} \leftarrow f(\boldsymbol{x}))$, such that for all input $\boldsymbol{x}$ and (non-adaptive) leakage function $L \in \mathcal{L}_1$ (chosen by adversary $\mathcal{A}$ after seeing output $\boldsymbol{y}$), the distributions $(L(W[C(\boldsymbol{x})]), C(\boldsymbol{x}))$ and

**Experiment** $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa)$

- $(W = W[C(\boldsymbol{x})], \boldsymbol{y}) \leftarrow C(\boldsymbol{x})$
- $\boldsymbol{y} \to \mathcal{A}$
- **non-adaptive:**
  - $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
  - $L(W) \to \mathcal{A}$
- $m$**-adaptive:** for $i$ in $1, 2, \ldots, m$:
  - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}_1$
  - $L_i(W) \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

**Experiment** $\mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa)$

- $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
- $\boldsymbol{y} \to \mathcal{A}$
- **non-adaptive:**
  - $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
  - $L' \leftarrow \mathsf{Sim}(L)$ where $L' \in \mathcal{L}_2$
  - $L'(\boldsymbol{x}, \boldsymbol{y}) \to \mathcal{A}$
- $m$**-adaptive:** for $i$ in $1, 2, \ldots, m$:
  - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}_1$
  - $L'_i \leftarrow \mathsf{Sim}(L)$ where $L'_i \in \mathcal{L}_2$
  - $L'_i(\boldsymbol{x}, \boldsymbol{y}) \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

Figure 2: Experiments of LTC

$(L \circ L_W(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{y})$ are $\epsilon$-indistinguishable. Formally, we have

$$\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa) \approx_\epsilon \mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa).$$

**Experiment** $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa)$

- $(W = W[C(\boldsymbol{x})], \boldsymbol{y}) \leftarrow C(\boldsymbol{x})$
- $\boldsymbol{y} \to \mathcal{A}$
- $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
- $L(W) \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

**Experiment** $\mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa)$

- $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
- $\boldsymbol{y} \to \mathcal{A}$
- $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
- $L_W \leftarrow \mathsf{Sim}$ where $L_W \in \mathcal{L}_2$
- $L \circ L_W(\boldsymbol{x}, \boldsymbol{y}) \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

Figure 3: Experiments of Strong LTC

Again, if $\epsilon$ is some negligible function of $\kappa$, we abbreviate the notation to strong $(\mathcal{L}_1, \mathcal{L}_2)$-LTC or just strong $\mathcal{L}_1$-LTC.

We claim that the strong leakage-tolerance property implies the standard (non-adaptive) leakage-tolerance:

**Lemma 4.10.** *A strong $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-LTC $C$ is also (non-adaptive) $(\mathcal{L}_1, \mathcal{L}_1 \circ \mathcal{L}_2, \epsilon)$-leakage-tolerant.*

*Specifically, for leakage class $\mathcal{L}$ that is closed under restriction and negation, a strong $(\mathcal{L}, \mathsf{NC}^0[1], \epsilon)$-LTC $C$ is $(\mathcal{L}, \mathcal{L}, \epsilon)$-leakage-tolerant.*

*Proof.* Given a simulator $\mathsf{Sim}_1$ for strong leakage-tolerance, we construct a simulator $\mathsf{Sim}_2$ for leakage-tolerance as follows: $\mathsf{Sim}_2$ first calls $\mathsf{Sim}_1$ to obtain a wire-value function $L_W \in \mathcal{L}_2$ where the domain is $(\boldsymbol{x}, \boldsymbol{y})$; then on any leakage function input $L \in \mathcal{L}_1$, $\mathsf{Sim}_2$ simply queries $L \circ L_W \in \mathcal{L}_1 \circ \mathcal{L}_2$ on $(\boldsymbol{x}, \boldsymbol{y})$. □

# 5 A Unified Framework: Constructing Strong $\mathsf{AC}^0$-LTCs

In this section, we formally describe our strong $\mathsf{AC}^0$-LTC construction introduced in Section 3.2. For a possibly randomized functionality $f$, we construct a strong $\mathsf{AC}^0$-LTC implementation of functionality $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$:

**Ingredient.**

- A stateless $(\mathsf{AC}^0(d, s, m), \epsilon_{\mathsf{LRC}})$-LRC $C$ with error parameter $\epsilon_{\mathsf{LRC}}$ that uses additive sharing of length-$n$ ("wire bundles") as the input encodings and output encodings, computing the *deterministic* function $\widehat{f}(\boldsymbol{x}, \boldsymbol{r})$ obtained from $f$ that additionally takes random wires $\boldsymbol{r}$ of $f$ as input (namely, $\widehat{f}(\boldsymbol{x}, \boldsymbol{r}) := f(\boldsymbol{x}; \boldsymbol{r})$). Note that the ISW stateless LRC [ISW03, BIS21] perfectly satisfies our requirement.
- A strong $(\mathsf{AC}^0(d, s, m), \mathsf{NC}^0[1], \epsilon_{\mathsf{rand}})$-LTC $\mathcal{G}$ with error parameter $\epsilon_{\mathsf{rand}}$ that computes the following randomized functionality $\mathsf{RandZero}$: $\mathsf{RandZero}$ does not take input and outputs $|f|$ i.i.d. random elements of $\mathsf{PAR}(n, 0)$. Let $\mathsf{SimG}$ be the associated simulator of $\mathcal{G}$. In Section 6, we construct $\mathcal{G}$ from a non-black-box use of ISW stateless LRC [ISW03, BIS21].

**Construction.**

**Strong $\mathsf{AC}^0$-LTC**

- **Input Encoding Phase.** For each input bit $\boldsymbol{x}_i$:
  - Receive $\boldsymbol{o}_{\boldsymbol{x}_i}$ from $\mathcal{G}$.
  - Compute wire bundle $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i} := (\boldsymbol{o}_{\boldsymbol{x}_i,1} \oplus \boldsymbol{x}_i, \boldsymbol{o}_{\boldsymbol{x}_i,2}, \ldots, \boldsymbol{o}_{\boldsymbol{x}_i,n}) \in \{0,1\}^n$.

  For each random wire $\boldsymbol{r}_i$ of $f$, assign wire bundle $\widehat{\boldsymbol{w}}_{\boldsymbol{r}_i} \xleftarrow{\$} \{0,1\}^n$.
- **Circuit Emulation Phase.** Emulate $C$ on input wire bundles $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}$ and $\widehat{\boldsymbol{w}}_{\boldsymbol{r}_i}$, obtain output wire bundles $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$.
- **Output Decoding Phase.** For each output bit $\boldsymbol{y}_i$:
  - Receive $\boldsymbol{o}_{\boldsymbol{y}_i}$ from $\mathcal{G}$.
  - Decode $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} := \widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i} \oplus \boldsymbol{o}_{\boldsymbol{y}_i}$ to obtain $\boldsymbol{y}_i$.

**Security.** We construct a strong LTC simulator $\mathsf{SimLTC}$ as follows:

- Randomly sample $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}, \widehat{\boldsymbol{w}}_{\boldsymbol{r}_i} \xleftarrow{\$} \mathsf{PAR}(n, 0)$ and the internal randomness of $C$, compute the internal wire values of $\widetilde{C}$ and the output wire bundles $\widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$.
- For each output bit $\boldsymbol{y}_i$, sample $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i,0} \xleftarrow{\$} \mathsf{PAR}(n, 0)$ and $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i,1} \xleftarrow{\$} \mathsf{PAR}(n, 1)$. Then $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i}$ can be written as $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i,0} \cdot (1 - \boldsymbol{y}_i) + \widehat{\boldsymbol{w}}_{\boldsymbol{y}_i,1} \cdot \boldsymbol{y}_i$, and the internal wire values of $\mathsf{Dec}(\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i})$ can be expressed as $\mathsf{NC}^0[1]$ functions of $\boldsymbol{y}_i$.

- Invoke $\mathsf{SimG}(\{\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i} \oplus (\boldsymbol{x}_i, 0, \ldots, 0)\}, \{\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \oplus \widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}\})$ to obtain the simulated wire values of $\mathcal{G}$. Note that because $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i} \oplus (\boldsymbol{x}_i, 0, \ldots, 0)$ is a $\mathsf{NC}^0[1]$ function of $\boldsymbol{x}_i$ and $\widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \oplus \widetilde{\boldsymbol{w}}_{\boldsymbol{y}_i}$ arises a $\mathsf{NC}^0[1]$ function of $\boldsymbol{y}_i$, the output of $\mathsf{SimG}$ can also be viewed as $\mathsf{NC}^0[1]$ functions of $\boldsymbol{x}, \boldsymbol{y}$.
- Output the wire values as $\mathsf{NC}^0[1]$ functions $L_W$ of $\boldsymbol{x}, \boldsymbol{y}$.

We prove the security of $\mathsf{SimLTC}$ using a simple hybrid argument.

**Hybrid $\mathsf{Hyb}_0$.** This hybrid is the real world execution.

**Hybrid $\mathsf{Hyb}_1$.** In this hybrid, we randomly sample $\boldsymbol{o}_{\boldsymbol{x}_i}$s and $\boldsymbol{o}_{\boldsymbol{y}_i}$s from $\mathsf{PAR}(n, 0)$, and invoke $\mathsf{SimG}$ to simulate the internal wire values of $\mathcal{G}$.

Note that the leakage query $L$ may take as input all wire values of the $\mathsf{AC}^0$-LTC. After fixing the input $\boldsymbol{x}$ and the internal randomness of $\widetilde{C}$ (including $\widehat{\boldsymbol{w}}_{\boldsymbol{r}_i}$), all the wire values of the subcircuit outside $\mathcal{G}$ can be fully determined by $\boldsymbol{o}$. Therefore, we can reduce $L$ to a query $L'$ of $W[\mathcal{G}]$ depending on $\boldsymbol{o}$. By the security of $\mathcal{G}$, $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_0$ are $\epsilon_{\mathsf{rand}}$-indistinguishable.

**Hybrid $\mathsf{Hyb}_2$.** In this hybrid, we first generate $\boldsymbol{r}_i \xleftarrow{\$} \{0,1\}$ and compute $\boldsymbol{y} \leftarrow \widehat{f}(\boldsymbol{x}, \boldsymbol{r})$, then sample $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i} \xleftarrow{\$} \mathsf{PAR}(n, \boldsymbol{x}_i), \widehat{\boldsymbol{w}}_{\boldsymbol{r}_i} \xleftarrow{\$} \mathsf{PAR}(n, \boldsymbol{r}_i), \widehat{\boldsymbol{w}}_{\boldsymbol{y}_i} \xleftarrow{\$} \mathsf{PAR}(n, \boldsymbol{y}_i)$ and compute $\boldsymbol{o}_{\boldsymbol{x}_i}, \boldsymbol{o}_{\boldsymbol{y}_i}$ accordingly.

By the perfect completeness of the underlying stateless LRC scheme, we know that $\mathsf{Hyb}_2$ and $\mathsf{Hyb}_1$ are identically distributed.

**Hybrid $\mathsf{Hyb}_3$.** In this hybrid, we switch each $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}, \widehat{\boldsymbol{w}}_{\boldsymbol{r}_i}$ to a random element of $\mathsf{PAR}(n, 0)$ and no longer generate $\boldsymbol{r}_i$. The output $\boldsymbol{y}$ is computed by the ideal functionality $f$ on the input $\boldsymbol{x}$.

Note that $\mathsf{Hyb}_3$ is the ideal world execution. Since in both $\mathsf{Hyb}_2$ and $\mathsf{Hyb}_3$ the output $\boldsymbol{y}$ is pre-determined before generating $\widehat{\boldsymbol{w}}_{\boldsymbol{x}_i}$ and $\widehat{\boldsymbol{w}}_{\boldsymbol{r}_i}$, by the security of the underlying stateless LRC scheme, $\mathsf{Hyb}_3$ and $\mathsf{Hyb}_2$ are $\epsilon_{\mathsf{LRC}}$-indistinguishable.

We conclude with the following result:

**Lemma 5.1.** *Assuming the building blocks described above, for any functionality $f$, constant $d$ and adversary bound $s, m$, our construction is a strong $(\mathsf{AC}^0(d, s, m), \mathsf{NC}^0[1], \epsilon_{\mathsf{LRC}} + \epsilon_{\mathsf{rand}})$-LTC implementation of $f$ with size $O((|f| + n)n^2)$.*

# 6 Instantiating the $\mathsf{AC}^0$-LTC Framework

In this section, we instantiate the framework introduced in Section 5 by designing the randomness generation LTC $\mathcal{G}$ that computes $\mathsf{RandZero}$, which generates a series of random elements in $\mathsf{PAR}(n, 0)$. Concretely, we adopt an ISW-style stateless $\mathsf{AC}^0$-LRC on random input encodings and view its output encodings as the LTC outputs. The security analysis of strong output-dependent leakage-tolerance involves non-trivial observations of the ISW construction.

## 6.1 Preliminaries: ISW Stateless LRCs

We first review the stateless LRC introduced by Ishai, Sahai, and Wagner [ISW03], which is perfectly secure against $(n/2 - 1)$-probing functions and statistically secure against $\mathsf{AC}^0(d, s, m)$ leakage [BIS21]. Given functionality $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$, we restate the ISW stateless LRC for $f$ as follows:

**ISW Stateless LRC**

The value of each wire $\alpha$ is replaced by a wire bundle $\boldsymbol{w}_\alpha \in \{0,1\}^n$ which is a $n$-bit additive sharing of $\alpha$.

- **Input Encoder $I$.** For each input bit $\boldsymbol{x}_i$, output wire bundle $\boldsymbol{w}_{\boldsymbol{x}_i}$ randomly sampled from $\mathsf{PAR}(n, \boldsymbol{x}_i)$.
- **Transformer $C$.**
  - Every random wire $\alpha$ is implemented by random wire bundle $\boldsymbol{w}_\alpha \xleftarrow{\$} \{0,1\}^n$.
  - Every $\mathsf{XOR}$ gate with input wires $\alpha, \beta$ and output wire $\gamma$ is implemented by $\boldsymbol{w}_\gamma = \boldsymbol{w}_\alpha \oplus \boldsymbol{w}_\beta$.
  - Every multiplication gate with input wires $\alpha, \beta$ and output wire $\gamma$ is implemented by a matrix-vector product as follows: sample random bits $\boldsymbol{z}'_{i,j} \xleftarrow{\$} \{0,1\}$ for all $1 \le i < j \le n$, then consider matrix $\boldsymbol{Z} \in \{0,1\}^{n \times n}$ given by

$$\boldsymbol{Z}_{i,j} = \begin{cases} \boldsymbol{z}'_{i,j}, & \text{if } i < j \\ \boldsymbol{w}_{\alpha,i}\boldsymbol{w}_{\beta,j}, & \text{if } i = j \\ \boldsymbol{Z}_{j,i} \oplus \boldsymbol{w}_{\alpha,i}\boldsymbol{w}_{\beta,j} \oplus \boldsymbol{w}_{\alpha,j}\boldsymbol{w}_{\beta,i}, & \text{if } i > j \end{cases}.$$

    The output wire bundle is $\boldsymbol{w}_\gamma = \boldsymbol{Z} \cdot \mathbf{1}$ where $\mathbf{1}$ denotes the all-1 vector of dimension $n$. Here, when computing the matrix-vector product, we accumulate the result from left to right for each coordinate.
- **Output Decoder $O$.** For each output bit $\boldsymbol{y}_i$, take $\boldsymbol{w}_{\boldsymbol{y}_i}$ as input and output the parity of $\boldsymbol{w}_{\boldsymbol{y}_i}$.

We claim that the ISW construction actually achieves a stronger notion of leakage-resilience: the leakage function can be adaptively chosen depending on the output wire bundle $\widehat{\boldsymbol{y}}$.

**Lemma 6.1** (Output-Dependent Leakage-Resilience)**.** *Assume that the function $f$ is deterministic and $(\mathsf{PAR}(n,0), \mathsf{PAR}(n,1))$ are $\epsilon$-indistinguishable by all $\mathsf{AC}^0(d,s,m) \circ \mathsf{NC}^0[2]$ circuits. For any input pair $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ such that $\boldsymbol{y} = f(\boldsymbol{x}_0) = f(\boldsymbol{x}_1)$, the ISW stateless LRC $(I, C, O)$ for $f$ is output-dependent $\epsilon$-indistinguishable between $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ against $\mathsf{AC}^0$ leakage: for any adversary $\mathcal{A}$ that on input $\widehat{\boldsymbol{y}}$ outputs $L \in \mathsf{AC}^0(d,s,m)$, we have*

$$\begin{aligned} &\{(L(W[C(I(\boldsymbol{x}_0))]), \widehat{\boldsymbol{y}}) : \widehat{\boldsymbol{y}} \leftarrow C(I(\boldsymbol{x}_0)), L \leftarrow \mathcal{A}(\widehat{\boldsymbol{y}})\} \\ \approx_\epsilon &\{(L(W[C(I(\boldsymbol{x}_1))]), \widehat{\boldsymbol{y}}) : \widehat{\boldsymbol{y}} \leftarrow C(I(\boldsymbol{x}_1)), L \leftarrow \mathcal{A}(\widehat{\boldsymbol{y}})\}. \end{aligned}$$

Finally, we show that the internal wire values of the ISW multiplication gate can be reverse-sampled from its output wire bundle when one of the input wire bundles is a fixed encoding of 1:

**Lemma 6.2** (Multiplication Gate Reconstruction)**.** *Consider ISW stateless LRC $(I, C, O)$ that computes a single multiplication gate $c = a \cdot b$ with input wire bundles $\boldsymbol{w}_a, \boldsymbol{w}_b$ and output wire bundle $\boldsymbol{w}_c$ of $C$. Assuming that $\boldsymbol{w}_a$ is a fixed encoding of 1 and $\boldsymbol{w}_b$ is randomly sampled from $\mathsf{PAR}(n, v)$ for $v \in \{0,1\}$, then there exists an efficient simulator $\mathsf{SimMul}$ that outputs a $\mathsf{NC}^0[1]$*

*function* $L_W$ *of* $W[C]$ *on input* $\boldsymbol{w}_c$*, such that*

$$\begin{aligned}&\{(W[C], \boldsymbol{w}_c) : \boldsymbol{w}_b \xleftarrow{\$} \mathsf{PAR}(n, v), \boldsymbol{w}_c \leftarrow C(\boldsymbol{w}_a, \boldsymbol{w}_b)\}\\ \equiv&\{(L_W(\boldsymbol{w}_c), \boldsymbol{w}_c) : \boldsymbol{w}_c \xleftarrow{\$} \mathsf{PAR}(n, v)\}\end{aligned}$$

The proofs of Lemma 6.1 and Lemma 6.2 are deferred to Appendix B .

### 6.2 Formal Construction

Our randomness generation LTC $\mathcal{G}$ computing $\mathsf{RandZero}$ is designed as follows:

- Sample random vectors $\boldsymbol{w}_{\boldsymbol{r}_1}, \ldots, \boldsymbol{w}_{\boldsymbol{r}_t} \xleftarrow{\$} \{0,1\}^n$ as random encodings of random bits $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t$, then use ISW stateless LRC to compute the product $\boldsymbol{w}_p$ of the $t$ encodings. We have $p = \bigwedge_{i=1}^{t} r_i$.
- For each output index $i$, randomly sample vector $\boldsymbol{w}_{\boldsymbol{u}_i} \xleftarrow{\$} \{0,1\}^n$ and view it as an encoding of random bit $\boldsymbol{u}_i$. Then compute the product of $\boldsymbol{w}_p$ and $\boldsymbol{w}_{\boldsymbol{u}_i}$ using an ISW stateless LRC $C_i$ that computes a single multiplication gate, output the product encoding as $\boldsymbol{o}_i$.

**Security.** We then construct a strong LTC simulator $\mathsf{SimG}$ of $\mathcal{G}$, the description is concise:

- Sample random encodings $\boldsymbol{w}_{\boldsymbol{r}_1}, \ldots, \boldsymbol{w}_{\boldsymbol{r}_t} \xleftarrow{\$} \mathsf{PAR}(n, 1)$, then use ISW stateless LRC to compute the encoding $\boldsymbol{w}_p$ of their product as real world execution. In this case, $p$ always equals 1.
- For each index $i$, invoke the multiplication gate simulator $\mathsf{SimMul}$ (Lemma 6.2) to obtain a $\mathsf{NC}^0[1]$ function $L_W^{C_i}(\boldsymbol{o}_i)$ on behalf of $W[C_i]$, since one of the input wire bundles $\boldsymbol{w}_p$ is an encoding of 1.
- Obtain the wire-value function $L_W$ according to $\boldsymbol{p}$ and each $L_W^{C_i}$.

We use a hybrid argument to prove the *strong leakage-tolerance* of $\mathcal{G}$: for any adversary $\mathcal{A}$ that on inputs $\{\boldsymbol{o}_i\}$ outputs a leakage function $L \in \mathsf{AC}^0(d, s, m)$,

$$\begin{aligned}&\{(L(W[\mathcal{G}]), \{\boldsymbol{o}_i\}) : \{\boldsymbol{o}_i\} \leftarrow \mathcal{G}, L \leftarrow \mathcal{A}(\{\boldsymbol{o}_i\})\}\\ \approx_\epsilon&\{(L \circ L_W(\{\boldsymbol{o}_i\}), \{\boldsymbol{o}_i\}) : L_W \leftarrow \mathsf{SimG}, \boldsymbol{o}_i \xleftarrow{\$} \mathsf{PAR}(n, 0), L \leftarrow \mathcal{A}(\{\boldsymbol{o}_i\})\}.\end{aligned}$$

**Hybrid** $\mathsf{Hyb}_0$**.** This hybrid is the real world execution.

**Hybrid** $\mathsf{Hyb}_1$**.** In this hybrid, we exclude the unexpected case that all random encodings $\boldsymbol{w}_{\boldsymbol{r}_i}$ ($i \in [t]$) are encodings of 1 (i.e. $\boldsymbol{r}_i = 1$). Thus, in $\mathsf{Hyb}_1$ the product encoding $\boldsymbol{w}_p$ is always an encoding of 0, as does all $\boldsymbol{o}_i \leftarrow C_i(\boldsymbol{w}_p, \boldsymbol{w}_{\boldsymbol{u}_i})$.

Since the unexpected event only occurs with probability $2^{-t}$ in $\mathsf{Hyb}_0$, the statistical distance of $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_0$ can be bounded by $2^{-t}$.

**Hybrid** $\mathsf{Hyb}_2$**.** In this hybrid, we sample each $\boldsymbol{w}_{\boldsymbol{r}_i}$ from $\mathsf{PAR}(n, 1)$ but sample each $\boldsymbol{w}_{\boldsymbol{u}_i}$ from $\mathsf{PAR}(n, 0)$.

Note that in both $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_2$ the outputs $\{\boldsymbol{o}_i\}$ are always encodings of 0. Assuming that $(\mathsf{PAR}(n, 0), \mathsf{PAR}(n, 1))$ are $\epsilon$-indistinguishable by $\mathsf{AC}^0(d, s, m) \circ \mathsf{NC}^0[2]$, by Lemma 6.1 we can show that $\mathsf{Hyb}_2$ and $\mathsf{Hyb}_1$ are also $\epsilon$-indistinguishable for any adversary $\mathcal{A}$.

**Hybrid $\mathsf{Hyb}_3$.** This hybrid is the ideal world execution: we compute $\boldsymbol{w}_p$ as before, but sample $\boldsymbol{o}_i \xleftarrow{\$} \mathsf{PAR}(n,0)$ for each $i$ just as the functionality $\mathsf{RandZero}$ and invoke the simulator $\mathsf{SimMul}$ to reconstruct each $W[C_i]$ as a $\mathsf{NC}^0[1]$ function $L_W^{C_i}$ of $\boldsymbol{o}_i$. By Lemma 6.2, $\mathsf{Hyb}_3$ and $\mathsf{Hyb}_2$ are identically distributed.

We conclude with the following result:

**Lemma 6.3.** *For any constant $d$ and adversary bounds $s, m$, assume that $(\mathsf{PAR}(n,0), \mathsf{PAR}(n,1))$ are $\epsilon$-indistinguishable by $\mathsf{AC}^0(d,s,m)\circ\mathsf{NC}^0[2]$ circuits. Then our construction $\mathcal{G}$ is strong $(\mathsf{AC}^0(d,s,m), \mathsf{NC}^0[1], \epsilon + 2^{-t})$-leakage-tolerant, computing the functionality $\mathsf{RandZero}$ which outputs i.i.d. samples of $\mathsf{PAR}(n,0)$.*

Set $t = n$, then $2^{-t}$ is bounded by $\epsilon$. Substituting into Lemma 5.1 and using the ISW circuit as the underlying stateless LRC in the circuit emulation phase, we obtain the following theorem:

**Theorem 6.4.** *For any constant $d$ and adversary bounds $s, m$, assume that $(\mathsf{PAR}(n,0), \mathsf{PAR}(n,1))$ are $\epsilon$-indistinguishable by $\mathsf{AC}^0(d,s,m) \circ \mathsf{NC}^0[2]$. Then for any functionality $f$, the construction in Section 5 is a strong $(\mathsf{AC}^0(d,s,m), \mathsf{NC}^0[1], O(\epsilon))$-LTC implementation of $f$ with an efficient simulator running in time $\mathsf{poly}(|f|, s, m, \kappa)$.*

A non-trivial application of our result is to construct a $t$-parity LTC with an efficient simulator.

**Theorem 6.5.** *For any adversary bound $t$ and functionality $f$, there exists a strong $(t\text{-}\mathsf{parity}, \mathsf{NC}^0[1])$-LTC implementation of $f$ with an efficient simulator running in time $\mathsf{poly}(|f|, t, \kappa)$.*

We defer the proof to Appendix C .

# 7 Framework for Deterministic Stateful LRCs Against Non-Decomposable Leakage

In this section, we introduce a generic approach for converting randomness-efficient $\mathcal{L}$-LTCs to (computationally secure) *deterministic* stateful $\mathcal{L}$-LRCs, which do not require any fresh randomness beyond the first invocation for arbitrary leakage class $\mathcal{L}$. Compared to existing work [IS25], our approach demonstrates a significant advantage in its wide-applicability: it is no longer limited to the simple decomposable leakage classes.

## 7.1 Global Randomness Extractor

Recall that in the approach of Section 3.3, we apply a global randomness extractor to reverse-sample the random tape $\mathsf{Rtape}_i$ given the global leakage result. Here we formally define the notion of global randomness extractor:

**Definition 7.1** (Global Randomness Extractor with Reverse-Sampling)**.** For polynomials $n, I = \mathsf{poly}(\kappa)$, a sequence of functions $\mathsf{Ext} = (\mathsf{Ext}_i)_{i\in[n]}$ such that $\mathsf{Ext}_i : \{0,1\}^I \to \{0,1\}$ is called an $(\mathcal{L}, \epsilon)$-global randomness extractor with reverse-sampling if it satisfies:

- *Regular*: for all $i \in [n]$,

$$\left| \Pr_{\boldsymbol{r}_i \xleftarrow{\$} \{0,1\}^I} [\mathsf{Ext}_i(\boldsymbol{r}_i) = 0] - \Pr_{\boldsymbol{r}_i \xleftarrow{\$} \{0,1\}^I} [\mathsf{Ext}_i(\boldsymbol{r}_i) = 1] \right| < \mathsf{negl}(\kappa).$$

As a result, define $\mathsf{Preimage}(\boldsymbol{b}_i)$ ($\boldsymbol{b}_i \in \{0,1\}$) as the uniform distribution on

$$\{(\boldsymbol{r}_i \in \{0,1\}^I)_{i\in[n]} : \mathsf{Ext}_i(\boldsymbol{r}_i) = \boldsymbol{b}_i\},$$

and define $\mathsf{Preimage}(\boldsymbol{b})$ as $\{(\boldsymbol{r}_i)_{i\in[n]} : \boldsymbol{r}_i \xleftarrow{\$} \mathsf{Preimage}(\boldsymbol{b}_i)\}$ for $\boldsymbol{b} \in \{0,1\}^n$, there exists an efficient sampler $\mathsf{Sext}(\boldsymbol{b})$ of $\mathsf{Preimage}(\boldsymbol{b})$ that uses $nJ$ uniform random bits (with negligible statistical distance) for $J = \mathsf{poly}(I,\kappa)$, using the rejection sampling method.

- *$m$-adaptive extractable*: there exists a simulator $\mathsf{SimExt}$ that takes $m$-adaptive query $L = (L_1,\ldots,L_m)$, outputs simulated leakage results $\{\ell_i\}_{i\in[m]}$ and $n$ pairs of strings $\boldsymbol{r}_{i,0},\boldsymbol{r}_{i,1} \in \{0,1\}^I$, such that for all $L \in \mathcal{L}$ and all $(\boldsymbol{b}_i)_{i\in[n]} \in \{0,1\}^n$, the following two distributions are $\epsilon$-indistinguishable:

$$\begin{aligned}&\{((L_i(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_n))_{i\in[m]},(\boldsymbol{r}_i)_{i\in[n]}) : \boldsymbol{r}_i \xleftarrow{\$} \mathsf{Preimage}(\boldsymbol{b}_i)\}\\ \approx_\epsilon &\{((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,\boldsymbol{b}_i})_{i\in[n]}) : ((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,0},\boldsymbol{r}_{i,1})_{i\in[n]}) \leftarrow \mathsf{SimExt}(L)\}.\end{aligned} \tag{1}$$

If $\epsilon$ is a negligible function of $\kappa$, we simply say that it is an $\mathcal{L}$-global randomness extractor with reverse-sampling.

**Remark 7.2.** Note that our definition is stronger than [IS25] in the following sense:

- We require the extractable property to hold for *all* sequences $\boldsymbol{b} \in \{0,1\}^n$, rather than a random sequence.
- We allow the query $L$ to be adaptively chosen. In this case, $\mathsf{SimExt}$ will respond a simulated query result $\ell_i$ upon receiving any sub-query $L_i \in L$ ($i \in [m]$), and determine the strings $(\boldsymbol{r}_{i,0},\boldsymbol{r}_{i,1})_{i\in[n]}$ afterwards.

## 7.2 Formal Construction

Given a leakage class $\mathcal{L}$ which is closed under restriction and negation, we now state our deterministic stateful $\mathcal{L}$-LRC implementation of (possibly randomized) functionality $f$.

**Ingredient.**

- A randomness-efficient $(\mathcal{L},\mathcal{L}')$-LTC implementation of deterministic functionality $f'$ (introduced later) with an efficient simulator $\mathsf{SimRELTC}$ for some intermediate leakage class $\mathcal{L}'$. Recall that the randomness complexity of the randomness-efficient LTC scheme is $\mathsf{RC}(|f'|,\kappa) = \mathsf{poly}(\log|f'|,\kappa)$.
- A pseudorandom generator $\mathsf{PRG}$ with seed length $\kappa$ that is secure against all polynomial-sized adversaries, such that the circuit size is $\mathsf{poly}(\eta,\kappa)$ where $\eta$ is the output length.
- An $\mathcal{L}'$-resilient encoding scheme $(\mathsf{Enc},\mathsf{Dec})$ such that $\mathsf{Enc}$ utilizes at most $w$-bit randomness to encode $b \in \{0,1\}$.
- An $\mathcal{L}'$-global randomness extractor $\mathsf{Ext} = (\mathsf{Ext}_i)_{i\in[n]}$, with a simulator $\mathsf{SimExt}$ and a sampler $\mathsf{Sext}$ that uses $J = \mathsf{poly}(I,\kappa)$ random bits to sample each $\boldsymbol{r}_i$.

**Construction.**

**Deterministic Stateful $\mathcal{L}$-LRC**

- In the setup process, output state $\widetilde{\boldsymbol{s}}_0 \xleftarrow{\$} \mathsf{Preimage}(\mathsf{Enc}(\boldsymbol{s}_0 \parallel \mathsf{Seed}_0))$, where $\mathsf{Seed}_0 \xleftarrow{\$} \{0,1\}^\kappa$ and the $\mathsf{Preimage}$ distribution is as defined in Definition 7.1.
- Define the following deterministic functionality $f'$:
  - $f'$ takes $\widehat{\boldsymbol{s}} = \mathsf{Enc}(\boldsymbol{s} \parallel \mathsf{Seed})$ and $\boldsymbol{x}$ as input, and decodes the first input to obtain $\boldsymbol{s}$ and $\mathsf{Seed}$.
  - $f'$ invokes $\mathsf{PRG}$ where the output length is $\eta = |f| + \kappa + w(|\boldsymbol{s}| + \kappa) + |\widehat{\boldsymbol{s}}|J + |r|I$ such that $|r| \geq \mathsf{RC}(|f'|, \kappa)$, and parses the output as the following parts:

    $$(\mathsf{Rint} \in \{0,1\}^{|f|}, \mathsf{Seed}' \in \{0,1\}^\kappa, \mathsf{Rseed} \in \{0,1\}^{w(|\boldsymbol{s}|+\kappa)},$$
    $$\mathsf{Rext} \in \{0,1\}^{|\widehat{\boldsymbol{s}}|J}, \mathsf{Rtape} \in \{0,1\}^{|r|I}).$$

    As $\mathsf{RC}(|f'|, \kappa) = \mathsf{poly}(\log |f'|, \kappa)$ and $|f'| = \mathsf{poly}(|f|, \eta, \kappa)$, there exists a sufficiently large $|r| = \mathsf{poly}(\log |f|, \kappa)$ satisfies the condition $|r| \geq \mathsf{RC}(|f'|, \kappa)$.
  - $f'$ computes $f[\boldsymbol{s}](\boldsymbol{x})$ to obtain the updated state $\boldsymbol{s}'$ and output $\boldsymbol{y}$, using $\mathsf{Rint}$ as internal randomness of $f$.
  - $f'$ computes $\widehat{\boldsymbol{s}'} \leftarrow \mathsf{Enc}(\boldsymbol{s}' \parallel \mathsf{Seed}'; \mathsf{Rseed})$.
  - $f'$ outputs $\boldsymbol{y}$, $\mathsf{Rtape}$ and a preimage of $\widehat{\boldsymbol{s}'}$: $\widetilde{\boldsymbol{s}'} \leftarrow \mathsf{Sext}(\widehat{\boldsymbol{s}'}; \mathsf{Rext})$.
- In $i$-th invocation, do the following:
  - If $i = 1$, parse the state as $\widetilde{\boldsymbol{s}}_0$ and additionally sample fresh random tape $\mathsf{Rtape}_0 \xleftarrow{\$} \{0,1\}^{|r|I}$; otherwise, parse the state as $\widetilde{\widehat{\boldsymbol{s}}_{i-1}}$ and $\mathsf{Rtape}_{i-1}$.
  - Compute $\widehat{\widehat{\boldsymbol{s}}_{i-1}} \leftarrow \mathsf{Ext}(\widetilde{\widehat{\boldsymbol{s}}_{i-1}})$ and $\mathsf{Rtape}'_{i-1} \leftarrow \mathsf{Ext}(\mathsf{Rtape}_{i-1})$.
  - Invoke the randomness-efficient LTC scheme to compute the functionality $f'(\widehat{\widehat{\boldsymbol{s}}_{i-1}}, \boldsymbol{x})$ using $\mathsf{Rtape}'_{i-1}$ as internal randomness and obtain $(\widetilde{\boldsymbol{s}}_i, \mathsf{Rtape}_i, \boldsymbol{y}_i)$.
  - Update the memory cell state to $(\widetilde{\boldsymbol{s}}_i, \mathsf{Rtape}_i)$, and output $\boldsymbol{y}_i$.

**Security.** We claim that our construction is indeed a deterministic stateful $\mathcal{L}$-LRC computing functionality $f$:

**Theorem 7.3.** *Assuming the building blocks described above, for any (possibly randomized) polynomial-sized stateful functionality $f$ and any leakage class $\mathcal{L}$ that is closed under restriction and negation, our construction is a computationally secure stateful $\mathcal{L}$-LRC implementation of $f$, which does not require fresh random tape beyond the first invocation.*

The proof is deferred to Appendix E .

# 8 Deterministic Stateful LRCs Against $k$-CNF Leakage

In this section, we give a concrete instantiation of the blueprint of deterministic stateful LRCs described in Section 7 against $k$-CNF leakage (for sufficiently small $k = O(\log \kappa)$), a sub-class of depth-2 $\mathsf{AC}^0$ leakage. We first introduce a global randomness extractor that works for generic $\mathsf{AC}^0$ leakage in Section 8.1, then provide a brief construction of a randomness-efficient $k$-CNF LTC in

Section 8.2. From these, we can realize a deterministic stateful LRC against $k$-CNF leakage that does not require fresh randomness beyond the first invocation.

## 8.1 Global Randomness Extractor for $\mathsf{AC}^0$ Leakage

We use a simple parity function as a global randomness extractor for all $m$-adaptive circuits over $\mathsf{AC}^0(d, s)$. Note that the same construction has already been proven to be secure against depth-1 $\mathsf{AC}^0$ functions in [IS25].

**Construction.** We consider the following XOR extractor Ext:

$$\left\{ \mathsf{Ext}_i(\boldsymbol{r}_i) := \bigoplus_{j=1}^{I} \boldsymbol{r}_{i,j} \right\}_{i\in[n]}$$

**Lemma 8.1.** *Given any constant $d$ and adversary bounds $m, s = \mathsf{poly}(\kappa)$, choose $\epsilon = \mathsf{negl}(\kappa)$ and set $I = \Theta(md(m+\log(s/\epsilon))\log^d s)$. For any polynomial $n = \mathsf{poly}(\kappa)$, Ext is an $\mathcal{L}$-global randomness extractor where $\mathcal{L}$ is the collection of $m$-adaptive leakage queries over $\mathsf{AC}^0(d, s)$; the sampler Sext uses exactly $J = I - 1$ random bits to sample each $\{\boldsymbol{r}_i\}_{i\in[n]}$; and the simulator runs in polynomial time.*

**Proof Sketch.** The regular property of Lemma 8.1 is trivial (the sampler uses exactly $J = I - 1$ bits to sample $\boldsymbol{r}_i$ from $\mathsf{Preimage}(\boldsymbol{b}_i) = \mathsf{PAR}(I, \boldsymbol{b}_i)$), thus we focus on the extractable property. For simplicity, here we only consider the case where the leakage function $L$ is an $\mathsf{AC}^0(d, s)$ query with only one output bit.

The Hastad switching lemma [Has86] indicates that a depth-2 $\mathsf{AC}^0$ leakage function of size $s$ and bottom fan-in $k$ collapses to a low-depth decision tree with high probability, after drawing a *random restriction* $\mathcal{R}_p$ ($p = \Theta(1/k)$) on input variables: each variable is independently assigned a random value with probability $1 - p$, and remains blank with probability $p$. Using $d - 1$ iterative applications of Hastad switching lemma, we can show that $L$ eventually collapses to a low-depth (say, depth $t$) decision tree $T$ with high probability after drawing a random restriction $\mathcal{R}_{p^{d-1}}$: in each invocation we draw a random restriction $\mathcal{R}_p$ to each of the bottom layer depth-2 subcircuits of $L$ and represent them as depth-$t$ decision trees, then we express each decision tree as a $t$-CNF or $t$-DNF, reducing the depth of $L$ by 1.

To construct a global randomness extractor against $L$, our first insight is to use a simulator SimExt that emulates the proof of Hastad switching lemma: SimExt first draws a random restriction $\mathcal{R}_{p^{d-1}}$. Then, the concentration inequality implies that for a sufficiently large parameter $I$, with overwhelming probability, every string $\boldsymbol{r}_i$ strictly leaves more than $t$ unassigned blanks. Therefore, SimExt can just sample a random path $\eta$ of the corresponding depth-$t$ decision tree $T$ to obtain the leakage result $\ell$ and assign the variables belonging to $\eta$ accordingly. Based on our hypothesis, each string $\boldsymbol{r}_i$ still contains at least one blank that does not affect the leakage result $\ell$ at all. Hence, if SimExt randomly samples the unassigned variables, $\boldsymbol{r}_i$ will become elements of $\mathsf{Preimage}(0)$ and $\mathsf{Preimage}(1)$ with identical probability.

However, the above algorithm suffers from an efficiency issue. Via the original approach of proving Hastad switching lemma [Has86], the decision tree depth $t$ is shown to be superlogarithmic, implying that we cannot expect the simulator to build $T$ in polynomial time.

To address this, one plausible optimization is to note that in the last invocation, the simulator no longer needs to explicitly build the decision tree $T$, but it suffices to efficiently sample a random

path $\eta$. Using this idea, we can fully resolve the case of depth-2 $\mathsf{AC}^0$ circuits, but even the depth-3 case remains intractable.

Our final solution makes use of the multi-switching Hastad lemma [IMP12, Has14, ST22]: given a set of depth-2 $\mathsf{AC}^0$ circuits (which refers to the set of every bottom layer depth-2 subcircuits of $L$ in our problem), after drawing $\mathcal{R}_p$, $L$ collapses to a "common partial" decision tree $T$ of depth $t = \Theta(\log(s/\epsilon))$ with high probability, such that if we fix the variables of any path $\eta$ of $T$, every depth-2 circuit collapses to a $O(\log s)$-depth decision tree which can be recovered in polynomial time! As a result, we can extend the above optimization strategy to arbitrary $\mathsf{AC}^0$ circuits: in each invocation the simulator samples a random path $\eta$ of the common partial decision tree $T$, then represents each subcircuit by an $O(\log s)$-depth decision tree with respect to $\eta$, and expresses it as an $O(\log s)$-CNF or an $O(\log s)$-DNF where the number of terms is polynomial in $s$.

We refer readers to Appendix F for the formal proof.

### 8.2 Overview of Randomness-Efficient LTCs Against $k$-CNF Leakage

Given a leakage class $\mathcal{L}$, a seemingly straightforward idea of constructing a randomness-efficient $\mathcal{L}$-LTC implementation of functionality $f$ works as follows:

1. Use an $\mathcal{L}$-resilient PRG $\mathsf{Gen}$ to expand random tape $R$ from a short seed.
2. Use an $\mathcal{L}$-LTC $C$ to compute $f$ with the (non-uniform) random tape $R$.

However, as we note in Section 3.4, the leakage-tolerance property of $C$ may not be preserved when the internal randomness is replaced by non-uniform random tape $R$, even if $R$ is $\mathcal{L}$-indistinguishable from uniform randomness. To resolve this issue, we strengthen the security requirement of $\mathcal{L}$-LTC to support non-uniform randomness:

**Definition 8.2** (Leakage-Tolerant circuit with Non-Uniform Randomness)**.** We say that a circuit $C$ is an $(\mathcal{L}_1, \mathcal{L}_2, \epsilon, \mathsf{Gen})$-leakage-tolerant circuit implementation of (possibly randomized) functionality $f$, if it is an $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-LTC implementation of $f$ such that its random tape can be divided into two parts $R$ and $R'$, and the security is preserved when the random tape $R$ is sampled by randomized circuit $\mathsf{Gen}$. The randomness complexity of $C$ is defined as the length of additional randomness $R'$.

Suppose that $\epsilon = \mathsf{negl}(\kappa)$, we may abbreviate the notation to $(\mathcal{L}_1, \mathcal{L}_2, \mathsf{Gen})$-LTC or $(\mathcal{L}_1, \mathsf{Gen})$-LTC.

A more severe issue is that the adversary can attack the internal wire values of the pseudorandom generator $\mathsf{Gen}$, making the direct computation of $\mathsf{Gen}$ insecure. Denoting the whole circuit $(\mathsf{Gen}, C)$ as $C'$, our idea to address this issue is to carefully design $\mathsf{Gen}$ so that any $\mathcal{L}$-leakage query of $C'$ can be converted to a $\mathcal{L}'$-leakage query of the external circuit $C$:

**Definition 8.3** (Internal-to-External Generator)**.** For a randomized circuit $C(\boldsymbol{x}; R \parallel R')$ (not necessarily $(\mathcal{L}_2, \mathsf{Gen})$-LTC), an $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-internal-to-external generator $\mathsf{Gen}$ of $C$ is a randomized circuit that outputs $R$, such that:

*Non-adaptive (m-adaptive) leakage-tolerance* is satisfied if there exists a simulator $\mathsf{Sim}$ such that for any adversary $\mathcal{A}$ and input $\boldsymbol{x}$, the following two distributions are $\epsilon$-indistinguishable:

$$\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa) \approx_\epsilon \mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa).$$

We may abbreviate the notation to $(\mathcal{L}_1, \mathcal{L}_2)$-internal-to-external generator or $\mathcal{L}_1$-internal-to-external generator if $\epsilon = \mathsf{negl}(\kappa)$.

**Experiment** $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A},\mathcal{L}_1,\boldsymbol{x}}(1^\kappa)$

- $(W' = W[C'(\boldsymbol{x})], \boldsymbol{y}) \leftarrow C'(\boldsymbol{x})$ where $C' = (\mathsf{Gen}, C)$
- $\boldsymbol{y} \to \mathcal{A}$
- **non-adaptive:**
  - $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
  - $L(W') \to \mathcal{A}$
- $m$**-adaptive:** for $i$ in $1, 2, \ldots, m$:
  - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}_1$
  - $L_i(W') \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

**Experiment** $\mathsf{Expt}^{\mathsf{Ideal}}_{\mathcal{A},\mathsf{Sim},\mathcal{L}_1,\mathcal{L}_2,\boldsymbol{x}}(1^\kappa)$

- $R \leftarrow \mathsf{Gen}$ and $R' \xleftarrow{\$} \{0,1\}^{|R'|}$
- $(W = W[C(\boldsymbol{x}; R \parallel R')], \boldsymbol{y}) \leftarrow C(\boldsymbol{x}; R \parallel R')$
- $\boldsymbol{y} \to \mathcal{A}$
- **non-adaptive:**
  - $L \leftarrow \mathcal{A}$ where $L \in \mathcal{L}_1$
  - $L' \leftarrow \mathsf{Sim}(L)$ where $L' \in \mathcal{L}_2$
  - $L'(W) \to \mathcal{A}$
- $m$**-adaptive:** for $i$ in $1, 2, \ldots, m$:
  - $L_i \leftarrow \mathcal{A}$ where $L_i \in \mathcal{L}_1$
  - $L'_i \leftarrow \mathsf{Sim}(L)$ where $L'_i \in \mathcal{L}_2$
  - $L'_i(W) \to \mathcal{A}$
- Output the view of $\mathcal{A}$ and $\boldsymbol{y}$

Figure 4: Experiments of Internal-to-External Generator

Now we use the above ideas to instantiate a randomness-efficient LTC for $k$-CNF leakage: given a functionality $f$, we build a $k$-CNF internal-to-external generator $\mathsf{Gen}$ and the corresponding non-uniform randomness LTC implementation of $f$ separately, where both schemes only leverage $\mathsf{poly}(\log|f|, \kappa)$ random bits except for the randomness complexity of $f$. By simply composing the two building blocks, we obtain a randomness-efficient $k$-CNF LTC computing $f$:

**Theorem 8.4.** *Given any functionality $f$, for adversary bound $m, k, s$, an error parameter $\epsilon$, and a sufficiently large parameter $\eta = \mathsf{poly}(m, 2^k, \log s, \log(1/\epsilon))$, there exists a $m$-adaptive $(\mathcal{L}_1, \mathcal{L}_2, O(\epsilon))$-LTC implementation of $f$, where $\mathcal{L}_1$ is the collection of $k$-CNF queries of size $s$, and $\mathcal{L}_2$ is the collection of $O(\eta/m)$-adaptive $k$-CNF queries of size $2^k s$. The randomness complexity $\mathsf{RC}(|f|, \kappa)$ of the LTC scheme is $\mathsf{poly}(m, 2^k, \log|f|, \log s, \log(1/\epsilon))$ plus the randomness complexity of $f$.*

*Specifically, if we fix $\epsilon = 2^{-\kappa}$, then for any adversary bound $k = O(\log\kappa)$, $m, s = \mathsf{poly}(\kappa)$, our LTC scheme is randomness-efficient with an exponentially small security loss, in which the simulator runs in polynomial time.*

We refer readers to Appendix G for more details.

**Summary.** Putting the global randomness extractor and the randomness-efficient LTC together, we immediately obtain a deterministic stateful LRC construction against adaptive $k$-CNF leakage following Theorem 7.3:

**Theorem 8.5.** *Assuming that one-way functions exist, for all (possibly randomized) polynomial-sized stateful functionalities $f$ and any adversary bounds $k = O(\log\kappa), m, s = \mathsf{poly}(\kappa)$, there exists a computationally secure stateful LRC implementation of $f$ against $m$-adaptive $k$-CNF leakage queries of size $s$ with an efficient simulator, which does not require fresh random tape beyond the first invocation.*

## References


[BB03] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, page 1, USA, 2003. USENIX Association.

[BIS21] Andrej Bogdanov, Yuval Ishai, and Akshayaram Srinivasan. Unconditionally secure computation against Low-Complexity leakage. *Journal of Cryptology*, 34(4):38, August 2021.

[BM06] Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against aes. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 201–215, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

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

[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.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 251–261, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, page 31–40, USA, 2012. IEEE Computer Society.

[Has86] J Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 6–20, New York, NY, USA, 1986. Association for Computing Machinery.

[Has14] Johan Hastad. On the correlation of parity and small-depth circuits. *SIAM J. Comput.*, 43(5):1699–1708, January 2014.

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for ac0. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, page 961–972, USA, 2012. Society for Industrial and Applied Mathematics.

[IS24] Yuval Ishai and Yifan Song. Leakage-tolerant circuits. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 196–225, Cham, 2024. Springer Nature Switzerland.

[IS25] Yuval Ishai and Yifan Song. Protecting computations against continuous bounded-communication leakage. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, page 1887–1897, New York, NY, USA, 2025. Association for Computing Machinery.

[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.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Mil14] Eric Miles. Iterated group products and leakage resilience against nc1. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, page 261–268, New York, NY, USA, 2014. Association for Computing Machinery.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In Moni Naor, editor, *Theory of Cryptography*, pages 278–296, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[MV13] Eric Miles and Emanuele Viola. Shielding circuits with groups. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 251–260, New York, NY, USA, 2013. Association for Computing Machinery.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Rot12] Guy N. Rothblum. How to compute under ac0 leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 552–569, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ST22] Rocco A. Servedio and Li-Yang Tan. Improved pseudorandom generators from pseudorandom multi-switching lemmas. *Theory of Computing*, 18(4):1–46, 2022.

[Wan25] Yuyu Wang. Leakage-resilient circuits against nc1, revisited. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, pages 38–69, Cham, 2025. Springer Nature Switzerland.

# A Negative Result of Strong LTC with Perfect Correctness

Although our strong LTC construction offers a much broader leakage class compared to the classical construction [IS24], it suffers from issues of imperfect completeness. We may wonder whether we can obtain a strong LTC construction with perfect correctness. Unfortunately, we can prove the following "no free lunch" theorem, showing that the imperfect correctness of strong LTC construction is inherent. The prototype of this theorem is first presented in [IS24], and a weaker version (leakage class containing depth-3 $\mathsf{AC}^0$ leakages) is presented in [Wan25].

**Theorem A.1.** *For any non-trivial leakage class $\mathcal{L}_1 \supseteq$ 3-probing and any leakage class $\mathcal{L}_2$, assume that there exists a $((3\text{-probing}) \circ \mathcal{L}_2)$-resilient encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$, then there also exists a polynomial-sized function $f$ such that no strong $(\mathcal{L}_1, \mathcal{L}_2)$-LTC $C$ can compute $f$ with perfect correctness.*

*Proof.* Let $f$ be the (deterministic) function $y \leftarrow \mathsf{Dec}(\boldsymbol{x})$, suppose that there exists a strong $(\mathcal{L}_1, \mathcal{L}_2)$-LTC implementation $C$ of $f$ which enjoys perfect correctness. Let $\mathsf{Sim}$ be the corresponding strong LTC simulator. Consider the following function family $\mathcal{L} \subseteq 3\text{-probing}$:

$$\begin{aligned}\mathcal{L} =&\{(\boldsymbol{w}_{\boldsymbol{x}_i})\} \cup \{(\boldsymbol{w}_\alpha, \boldsymbol{w}_\beta, \boldsymbol{w}_\gamma) : \\ &(\alpha, \beta, \gamma) \text{ are the input wires and output wire of some gate } g \in \mathcal{C}\} \cup \{(\boldsymbol{w}_y)\}.\end{aligned}$$

We say that a wire value assignment $\widetilde{W}$ of $C$ is a valid computation of $\boldsymbol{x}$, if the input wires of $\widetilde{W}$ are $\boldsymbol{x}$, the output wire is 0 and each gate assignment is consistent. Given any wire value assignment $\widetilde{W} = (\boldsymbol{w}_\alpha)_{\alpha \in \mathcal{C}}$, we can perform the following randomized verifier $V(\boldsymbol{x}, \widetilde{W})$ to check whether $\widetilde{W}$ is a valid computation of $\boldsymbol{x}$:

- $V$ randomly samples $L \xleftarrow{\$} \mathcal{L}$.
- If $L = (\boldsymbol{w}_{\boldsymbol{x}_i})$, verifies $\boldsymbol{w}_{\boldsymbol{x}_i} = \boldsymbol{x}_i$;
- Otherwise, if $L = (\boldsymbol{w}_\alpha, \boldsymbol{w}_\beta, \boldsymbol{w}_\gamma)$ corresponds to some gate $g$, verifies $g(\boldsymbol{w}_\alpha, \boldsymbol{w}_\beta) = \boldsymbol{w}_\gamma$.
- Otherwise $L = (\boldsymbol{w}_y)$, verifies $\boldsymbol{w}_y = 0$.

Note that if $(\boldsymbol{x}, \widetilde{W})$ is a valid computation of $\boldsymbol{x}$, it always passes the test; otherwise, it fails with non-negligible probability at least $1/|\mathcal{L}|$.

Now, let $\boldsymbol{x}$ be randomly sampled from $\mathsf{Enc}(0)$, we will have $V(\boldsymbol{x}, W[C(\boldsymbol{x})]) \equiv 1$. Since $V$ only invokes a random 3-probing leakage function $L \xleftarrow{\$} \mathcal{L}$, by the strong leakage-tolerance property $V(\boldsymbol{x}, L_W(\boldsymbol{x}, 0)) = 1$ holds with probability at least $1 - \epsilon_1$ for some $\epsilon_1 = \mathsf{negl}(\kappa)$, over the random choice of $\boldsymbol{x}$ and $L_W \leftarrow \mathsf{Sim}$:

$$\Pr[V(\boldsymbol{x}, L_W(\boldsymbol{x}, 0)) = 1 : \boldsymbol{x} \leftarrow \mathsf{Enc}(0), L_W \leftarrow \mathsf{Sim}] \geq 1 - \epsilon_1.$$

Observe that $V$ itself is a randomized 3-probing function on $(\boldsymbol{x}, \widetilde{W})$. Then if we sample $\boldsymbol{x}' \leftarrow \mathsf{Enc}(1)$, following the security of the encoding scheme we have:

$$\Pr[V(\boldsymbol{x}, L_W(\boldsymbol{x}', 0)) = 1 : \boldsymbol{x}' \leftarrow \mathsf{Enc}(1), L_W \leftarrow \mathsf{Sim}] \geq 1 - \epsilon_1 - \epsilon_2.$$

for some $\epsilon_2 = \mathsf{negl}(\kappa)$. Therefore, $L_W(\boldsymbol{x}', 0)$ is a valid computation of $\boldsymbol{x}'$ with non-zero probability $1 - |L|(\epsilon_1 + \epsilon_2)$, which contradicts the perfect correctness of $C$! □

**Remark A.2.** [Wan25] shows that under a weaker definition such that the LTC scheme may receive an auxiliary input $\mathsf{aux} \leftarrow \mathsf{Setup}(1^\kappa)$ during a trusted setup process (where the authors call this definition as "AI-LTC"), there exists a strong $\mathsf{NC}^1$-LTC with perfect completeness under mild assumption. We find that it is quite straightforward to transform our strong $\mathsf{AC}^0$-LTC construction (Theorem 6.4) into an AI-LTC with perfect correctness: receive a fresh $\boldsymbol{w}_p \xleftarrow{\$} \mathsf{PAR}(n, 0)$ used in the randomness generation LTC $\mathcal{G}$ as an auxiliary input.

# B Deferred Proofs of Section 6.1

## B.1 Proof of Lemma 6.1

In our analysis, we rely on the following characterization of the ISW LRC:

**Lemma B.1** ( [BIS21, Claim 4]). *Given a ISW stateless LRC $(I, C, O)$ for function $f$ with $k = |\boldsymbol{x}|$ input bits, there exists an efficient (randomized) simulator* $\mathsf{SimISW}$ *that takes any $k$ wire bundles $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k \in \{0,1\}^n$ and any $k$ bits $c_1, \ldots, c_k$ as input, outputs a $\mathsf{NC}^0[2]$ function $L_W$ on input $\boldsymbol{v} \in \{0,1\}^n$ such that:*

- *The distribution of $L_W(\boldsymbol{v})$ and $W[C(\boldsymbol{u}_1 \oplus c_1 \cdot \boldsymbol{v}, \ldots, \boldsymbol{u}_k \oplus c_k \cdot \boldsymbol{v})]$ are identically distributed.*
- *For each wire $\alpha$ of $\widehat{C}$, $L_W$ represents the wire bundle $\boldsymbol{w}_\alpha$ as a linear ($\mathsf{NC}^0[1]$) function $\boldsymbol{a}_\alpha \oplus b_\alpha \cdot \boldsymbol{v}$ for some $\boldsymbol{a}_\alpha \in \{0,1\}^n$ and $b_\alpha \in \{0,1\}$.*

We use Lemma B.1 to amplify the leakage-resilience of the ISW construction: Denote the distributions of $I(\boldsymbol{x}_0)$ and $I(\boldsymbol{x}_1)$ as $D_0$ and $D_1$, respectively. Let $c_i$ indicate the difference between $\boldsymbol{x}_{0,i}$ and $\boldsymbol{x}_{1,i}$ where $\boldsymbol{x}_{0,i}$ and $\boldsymbol{x}_{1,i}$ are the $i$-th bits of $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ respectively (namely, $c_i = 1$ if and only if $\boldsymbol{x}_{0,i} \neq \boldsymbol{x}_{1,i}$), then we have

$$D_i = \{(\boldsymbol{u}_j \oplus c_j \cdot \boldsymbol{v})_{j \in [k]} : \boldsymbol{v} \xleftarrow{\$} \mathsf{PAR}(n, i), \boldsymbol{u}_j \xleftarrow{\$} \mathsf{PAR}(n, \boldsymbol{x}_{0,j})\}$$

We claim that for any fixed $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ and $c_1, \ldots, c_k$ (in this case, only $\boldsymbol{v}$ is randomly sampled), $W[C(I(\boldsymbol{x}_0))]$ and $W[C(I(\boldsymbol{x}_1))]$ are $\epsilon$-indistinguishable by any adversary $\mathcal{A}(\widehat{\boldsymbol{y}})$. Applying Lemma B.1 to the input $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k, c_1, \ldots, c_k)$, we know that the wire value assignment is perfectly simulated by $L_W \leftarrow \mathsf{SimISW}$. Therefore, we only need to prove the indistinguishability result of $L_W(\boldsymbol{v})$ between $\boldsymbol{v} \xleftarrow{\$} \mathsf{PAR}(n, 0)$ and $\boldsymbol{v} \xleftarrow{\$} \mathsf{PAR}(n, 1)$ (corresponding to $D_0$ and $D_1$, respectively).

Notice that for each output bit $\boldsymbol{y}_i$, the output wire bundle $\widehat{\boldsymbol{y}}_i$ can be written as linear function $\boldsymbol{a}_{\boldsymbol{y}_i} \oplus b_{\boldsymbol{y}_i} \cdot \boldsymbol{v}$ in $L_W$. Since the value of $\boldsymbol{y}_i = \mathsf{parity}(\widehat{\boldsymbol{y}}_i)$ remains invariant no matter $\boldsymbol{x} = \boldsymbol{x}_0$ or $\boldsymbol{x} = \boldsymbol{x}_1$, moving $\boldsymbol{v}$ from $\mathsf{PAR}(n, 0)$ to $\mathsf{PAR}(n, 1)$ the parity of $\widehat{\boldsymbol{y}}$ does not change. The only possibility of this phenomenon is $b_{\boldsymbol{y}_i} = 0$ for all $\boldsymbol{y}_i$! Hence, $\widehat{\boldsymbol{y}} = (\boldsymbol{a}_{\boldsymbol{y}_1}, \boldsymbol{a}_{\boldsymbol{y}_2}, \ldots)$ is independent of $\boldsymbol{v}$ and revealing it to the adversary $\mathcal{A}$ is not beneficial to distinguish $\boldsymbol{v}$. The lemma then follows from the assumption that $(\mathsf{PAR}(n, 0), \mathsf{PAR}(n, 1))$ are $\epsilon$-indistinguishable.

**Remark B.2.** An analogous argument can be used to prove a tighter bound for the leakage-resilience of ISW construction against $\mathsf{AC}^0$ leakage, improving the security analysis in [BIS21] by a $k$ factor.

## B.2 Proof of Lemma 6.2

Consider the following equivalent process to generate $W[C]$ (for a correctness proof, see Section 2.1 of [BIS21]): $\mathsf{SimMul}$ still samples random bits $\boldsymbol{z}'_{i,j} \xleftarrow{\$} \{0,1\}$ for $1 \leq i < j \leq n$, but constructs matrix $\boldsymbol{Z}$ by:

$$\boldsymbol{Z}_{i,j} = \begin{cases} \boldsymbol{z}'_{i,j} \oplus \boldsymbol{w}_{a,j}\boldsymbol{w}_{b,i}, & \text{if } i < j \\ \boldsymbol{w}_{a,j}\boldsymbol{w}_{b,i}, & \text{if } i = j \\ \boldsymbol{Z}_{j,i} \oplus \boldsymbol{w}_{a,j}\boldsymbol{w}_{b,i} \oplus \boldsymbol{w}_{a,i}\boldsymbol{w}_{b,j}, & \text{if } i > j \end{cases},$$

and finally outputs $\boldsymbol{w}_c = \boldsymbol{Z} \cdot \boldsymbol{1}$ as before.

Fixing $\boldsymbol{z}'_{i,j}$, we find that there exists a one-to-one correspondence from $\boldsymbol{w}_c$ to $\boldsymbol{w}_b$: for each $i \in [n]$, by direct calculation we can show that

$$\boldsymbol{w}_{c,i} = \boldsymbol{w}_{b,i} \oplus \left( \sum_{j<i} \boldsymbol{z}'_{j,i} \oplus \sum_{j>i} \boldsymbol{z}'_{i,j} \right).$$

Hence, $\boldsymbol{w}_b$ can be uniquely represented as a $\mathsf{NC}^0[1]$ function on input $\boldsymbol{w}_c$ and all internal wire values can be expressed as $\mathsf{NC}^0[1]$ functions accordingly.

# C Deferred Proof of Theorem 6.5

## C.1 Parity-To-Probing Circuit

We first review the notion of parity-to-probing circuit [IS24]:

**Definition C.1** (Parity-to-probing Circuit)**.** Given a (possibly randomized) circuit $\widetilde{C}$, a $(t, k, \epsilon)$-parity-to-probing circuit implementation of $\widetilde{C}$ consists of randomized-sized circuits $(I, C, O)$ with the same syntax as leakage-resilient circuit defined in Definition 4.4 such that:

*Correctness* is satisfied if for all input $\boldsymbol{x}$, we have

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

*Trivial Parity Tolerance* is satisfied if the input encoder $I$ and the output decoder $O$ perform bitwise operations on the circuit input $\boldsymbol{x}$ and output $\boldsymbol{y}$, and:

- For each input bit $\boldsymbol{x}_i$, say $I(\boldsymbol{x}_i)$ computes encoding $\widehat{\boldsymbol{x}}_i$, then the internal wires of $I(\boldsymbol{x}_i)$ are linear combinations of $\boldsymbol{x}_i$ and $\widehat{\boldsymbol{x}}_i$.
- Similarly, for each output bit $\boldsymbol{y}_i$, say $O(\widehat{\boldsymbol{y}}_i)$ decodes to $\boldsymbol{y}_i$, the internal wires of $O(\widehat{\boldsymbol{y}}_i)$ are linear combinations of $\boldsymbol{y}_i$ and $\widehat{\boldsymbol{y}}_i$.

*Non-adaptive leakage-tolerance* is satisfied if there exists some (possibly inefficient) stateful simulator $\mathsf{SimP2P} = (\mathsf{SimP2P}_1, \mathsf{SimP2P}_2)$ such that, for any input $\boldsymbol{x}$ and $t$-parity leakage function $L$ (adaptively chosen by adversary according to output) on wire values of $C$, the following two distributions are $\epsilon$-indistinguishable:

$$(L(W[C(\boldsymbol{x})]), O(C(I(\boldsymbol{x})))) \approx_\epsilon \{(\mathsf{SimP2P}_2(L'(W[\widetilde{C}(\boldsymbol{x})])), \widetilde{C}(\boldsymbol{x})) : L' \leftarrow \mathsf{SimP2P}_1(L)\},$$

where $L'$ is $k$-probing leakage query on wire values of $\widetilde{C}$.

**Remark C.2.** Compared to the original definition of [IS24], we additionally require that the input encoder $I$ and the output decoder $O$ can be expressed as bitwise operations of $\boldsymbol{x}$ and $\boldsymbol{y}$, while this property is guaranteed by their construction.

Assuming that the gates of $\widetilde{C}$ are represented by only $\mathsf{NAND}$ gates, we restate the construction of $(t, k)$-parity-to-probing circuit implementation as follows:

**Parity-To-Probing Circuit**

- For every wire $\alpha$ of $\widetilde{C}$, randomly sample $\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha \xleftarrow{\$} \{0, 1\}$.
- **Input Encoder $I$.** For each input bit $\boldsymbol{x}_i$, compute the encoding of $\boldsymbol{x}_i$ as $\boldsymbol{w}_{\boldsymbol{x}_i} = (\boldsymbol{u}_{\boldsymbol{x}_i}, \boldsymbol{v}_{\boldsymbol{x}_i}, \boldsymbol{z}_{\boldsymbol{x}_i} = \boldsymbol{u}_{\boldsymbol{x}_i} \cdot \boldsymbol{v}_{\boldsymbol{x}_i} \oplus \boldsymbol{x}_i)$.
- **Transformer $C$.**

- For every random wire $\alpha$ of $\widetilde{C}$, sample $\boldsymbol{z}_\alpha \xleftarrow{\$} \{0,1\}$ and assign $(\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha, \boldsymbol{z}_\alpha)$ as wire label $\boldsymbol{w}_\alpha$.
- For every $\mathsf{NAND}$ gate $g$ of $\widetilde{C}$ with input wires $\alpha, \beta$ and output wire $\gamma$, use some randomized parity-tolerant implementation $F_{\mathsf{NAND}}$ to compute

$$\boldsymbol{z}_\gamma = \boldsymbol{u}_\gamma \cdot \boldsymbol{v}_\gamma \oplus (\boldsymbol{u}_\alpha \cdot \boldsymbol{v}_\alpha \oplus \boldsymbol{z}_\alpha) \cdot (\boldsymbol{u}_\beta \cdot \boldsymbol{v}_\beta \oplus \boldsymbol{z}_\beta) \oplus 1.$$

- **Output Decoder** $O$**.** For each output bit $\boldsymbol{y}_i$, the output decoder takes the encoding $\boldsymbol{w}_{\boldsymbol{y}_i} = (\boldsymbol{u}_{\boldsymbol{y}_i}, \boldsymbol{v}_{\boldsymbol{y}_i}, \boldsymbol{z}_{\boldsymbol{y}_i})$ and outputs $\boldsymbol{y}_i = \boldsymbol{u}_{\boldsymbol{y}_i} \cdot \boldsymbol{v}_{\boldsymbol{y}_i} \oplus \boldsymbol{z}_{\boldsymbol{y}_i}$.

We note that in the above construction, the output distribution of $W[C(\boldsymbol{x})]$ is identical to the following one:

- Compute a wire value assignment $\widetilde{W} = (\widetilde{\boldsymbol{w}}_\alpha)$ of $\widetilde{C}(\boldsymbol{x})$ using uniform random tape.
- Assign $(\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha, \boldsymbol{z}_\alpha = \boldsymbol{u}_\alpha \cdot \boldsymbol{v}_\alpha \oplus \widetilde{\boldsymbol{w}}_\alpha)$ to $\boldsymbol{w}_\alpha$ for each wire $\alpha$ of $\widetilde{C}$.
- For each gate $g$ with input wires $\alpha, \beta$ and output wire $\gamma$, compute the wire values of the corresponding $F_{\mathsf{NAND}}$ gadget on inputs $(\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha, \boldsymbol{z}_\alpha)$, $(\boldsymbol{u}_\beta, \boldsymbol{v}_\beta, \boldsymbol{z}_\beta)$ and $(\boldsymbol{u}_\gamma, \boldsymbol{v}_\gamma)$ with uniform internal randomness.

Denote the wire value distribution $W[C]$ gained above as $C(\widetilde{W})$ with respect to some assignment $\widetilde{W}$, the security in Definition C.1 indeed implies a stronger security notion:

**Lemma C.3.** *We say that a wire value assignment $\widetilde{W}$ of $\widetilde{C}$ is consistent if it is consistent for every gate.*

*Then for any consistent assignment $\widetilde{W}$, the above construction enjoys the following property:*

$$(L(W[C(\widetilde{W})]), O(C(\widetilde{W}))) \approx_\epsilon \{(\mathsf{SimP2P}_2(L'(\widetilde{W})), \boldsymbol{y}) : L' \leftarrow \mathsf{SimP2P}_1(L), \boldsymbol{y} \leftarrow \widetilde{W}\}.$$

*Proof.* By fixing the random tape of $\widetilde{C}$, we reduce the case of randomized circuit to deterministic one, then notice that any consistent assignment $\widetilde{W}$ of deterministic circuit $\widetilde{C}$ uniquely maps to some appropriate input $\boldsymbol{x}$. □

## C.2 Formal Construction

**Construction.** Given a function $f$, our strong parity LTC construction works as follows:

- Implement function $f$ using strong $(k\text{-}\mathsf{probing}, \mathsf{NC}^0[1], \epsilon_{\mathsf{probe}})$-LTC $\widetilde{C}$ $(k \geq 3)$ for some $\epsilon_{\mathsf{probe}} = \mathsf{negl}(\kappa)$, with an efficient simulator $\mathsf{Sim}'$. Recall that $k$-probing leakage is just a special case of $\mathsf{AC}^0$ leakage, thus we can adopt the construction in Theorem 6.4.
- Apply a $(t, k, \epsilon_{\mathsf{parity}})$-parity-to-probing circuit $C' := (I, C, O)$ to compute $\widetilde{C}$ on input $\boldsymbol{x}$ for some $\epsilon_{\mathsf{parity}} = \mathsf{negl}(\kappa)$, with an inefficient simulator $\mathsf{SimP2P} = (\mathsf{SimP2P}_1, \mathsf{SimP2P}_2)$.

**Security.** We construct an *efficient* strong parity LTC simulator $\mathsf{Sim}$ computing a $\mathsf{NC}^0[1]$ wire value function $L_W$ of $W[C'(\boldsymbol{x})]$ as follows:

- Randomly sample $\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha$ for each wire $\alpha$ in $\widetilde{C}$.
- Leverage the trivial parity tolerance property to express the internal wire values of $I$ and $O$ as $\mathsf{NC}^0[1]$ functions of $\boldsymbol{x}, \boldsymbol{y}$, since $\{\boldsymbol{u}_{\boldsymbol{x}_i}, \boldsymbol{v}_{\boldsymbol{x}_i}\}, \{\boldsymbol{u}_{\boldsymbol{y}_i}, \boldsymbol{v}_{\boldsymbol{y}_i}\}$ are all fixed and $\{\boldsymbol{z}_{\boldsymbol{x}_i}\}, \{\boldsymbol{z}_{\boldsymbol{y}_i}\}$ can be viewed as $\mathsf{NC}^0[1]$ functions of $\boldsymbol{x}, \boldsymbol{y}$.

- Invoke the $k$-probing LTC simulator $\mathsf{Sim}'$ to express the wire value assignment $\widetilde{W}$ of $\widetilde{C}$ as a $\mathsf{NC}^0[1]$ function $L_W^{\widetilde{C}}$ of $\boldsymbol{x}, \boldsymbol{y}$.
- Randomly sample the internal randomness of each $F_{\mathsf{NAND}}$ gadget and obtain a $\mathsf{NC}^0[2]$ wire value function $L_W^C(\widetilde{W})$ on behalf of the internal wire values. Then express the wire values as $\mathsf{NC}^0[1]$ functions $L_W^C \circ L_W^{\widetilde{C}}$ of $\boldsymbol{x}, \boldsymbol{y}$.

  Although $L_W^C$ is a $\mathsf{NC}^0[2]$ function of $\widetilde{W}$, the output $L_W^C \circ L_W^{\widetilde{C}}$ remains a $\mathsf{NC}^0[1]$ function of $\boldsymbol{x}, \boldsymbol{y}$. To argue this, we only need to verify that the construction in Theorem 6.4 satisfies the following property: for each gate $g$, either at least one of $\boldsymbol{w}_\alpha$ and $\boldsymbol{w}_\beta$ given by $L_W^{\widetilde{C}}$ is constant, or they are both linear functions of the same bit $b$ in $(\boldsymbol{x}, \boldsymbol{y})$.
- Finally, output $L_W$ as the concatenation of the above $\mathsf{NC}^0[1]$ functions.

We then argue for the security using a hybrid argument.

**Hybrid $\mathsf{Hyb}_0$.** This hybrid is the real world execution.

**Hybrid $\mathsf{Hyb}_1$.** In this hybrid, we modify the generation process of $W[C'(\boldsymbol{x})]$ to:

- Randomly sample $\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha$ for each wire $\alpha$ in $\widetilde{C}$.
- Compute the internal wire values of the input encoder $I$ on the input $\boldsymbol{x}$.
- Compute the wire values of $\widetilde{C}(\boldsymbol{x})$ (denoted as $\widetilde{W}$), and let the output of $\widetilde{C}$ be the function output $\boldsymbol{y}$.
- Generate the internal wire values of $C$ as $C(\widetilde{W})$.
- Compute the internal wire values of the output encoder $O$ on the input $\boldsymbol{w}_{\boldsymbol{y}} = (\boldsymbol{u}_{\boldsymbol{y}}, \boldsymbol{v}_{\boldsymbol{y}}, \boldsymbol{u}_{\boldsymbol{y}} \cdot \boldsymbol{v}_{\boldsymbol{y}} \oplus \boldsymbol{y})$.

  As we discussed earlier, this does not change the wire value distribution and $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_0$.

**Hybrid $\mathsf{Hyb}_2$.** In this hybrid, we leverage the trivial leakage tolerance property to translate a $t$-parity query $L$ on $W[C'(\boldsymbol{x})]$ to $t$-parity query $L'$ on $W[C(\boldsymbol{x})]$, it is clear that $\mathsf{Hyb}_2$ and $\mathsf{Hyb}_1$ are identically distributed.

**Hybrid $\mathsf{Hyb}_3$.** In this hybrid, we emulate $L'$ using the simulator $\mathsf{SimP2P}$. Namely, output

$$\mathsf{SimP2P}_2(L''(\widetilde{W})), L'' \leftarrow \mathsf{SimP2P}_1(L')$$

as the leakage result, where $L''$ is a $k$-probing leakage query.

Applying Lemma C.3, we have $\mathsf{Hyb}_3$ and $\mathsf{Hyb}_2$ are $\epsilon_{\mathsf{parity}}$-indistinguishable.

**Hybrid $\mathsf{Hyb}_4$.** In this hybrid, we emulate the wire value assignment $\widetilde{W}$ with the help of $\mathsf{Sim}'$: compute $\widetilde{W} \leftarrow L_{\widetilde{C}}(\boldsymbol{x}, \boldsymbol{y})$.

As the simulator $\mathsf{SimP2P}_2$ only performs $k$-probing leakage query $L''$ on $\widetilde{W}$, the security of the underlying probing LTC implies that $\mathsf{Hyb}_4$ and $\mathsf{Hyb}_3$ are $\epsilon_{\mathsf{probe}}$-indistinguishable.

**Hybrid $\mathsf{Hyb}_5$.** In this hybrid, we switch back to generate $W[C']$ as in $\mathsf{Hyb}_2$ except that the wire value assignment $\widetilde{W}$ is simulated by $\mathsf{Sim}'$.

To argue the security, we want to leverage Lemma C.3 again, but we need to prove that the simulated $\widetilde{W}$ is consistent:

**Lemma C.4.** *$\widetilde{W} = L_{\widetilde{C}}(\boldsymbol{x}, \boldsymbol{y})$ is an consistent assignment with probability at least $1 - |\widetilde{C}|\epsilon_{\mathsf{probe}}$.*

The argument is very similar to Theorem A.1.

*Proof.* Suppose that $\widetilde{W}$ is not consistent with probability larger than $|\widetilde{C}|\epsilon_{\mathsf{probe}}$. Then consider the following collection of 3-probing leakage queries:

$$\mathcal{L} = \{(\boldsymbol{w}_\alpha, \boldsymbol{w}_\beta, \boldsymbol{w}_\gamma) : (\alpha, \beta, \gamma) \text{ are the input wires and output wire of some gate } g \in \mathcal{C}\}$$

Thus, by the hypothesis with probability larger than $|\widetilde{C}|\epsilon_{\mathsf{probe}}/|\mathcal{L}| \geq \epsilon_{\mathsf{probe}}$ a random 3-probing leakage query $L \xleftarrow{\$} \mathcal{L}$ outputs 0 on $\widetilde{W}$. However, in the real world execution $L$ constantly outputs 1, contradicts with the probing leakage-tolerance property as $k \geq 3$. □

Consequently, $\mathsf{Hyb}_5$ and $\mathsf{Hyb}_4$ are $(|\widetilde{C}|\epsilon_{\mathsf{probe}} + \epsilon_{\mathsf{parity}})$-indistinguishable. Note that $\mathsf{Hyb}_5$ is the ideal world execution.

# D Separation Between Output-Independent Security and Output-Dependent Security

In this section, we will separate the output-dependent leakage-tolerance property from the (standard) output-independent leakage-tolerance property. Specifically, we construct a strong output-independent $(1\text{-}\mathsf{parity}, \mathsf{NC}^0[1])$-LTC $C$ that is not strong output-dependent leakage-tolerant over the same leakage class.

**Ingredients.** We require the following building blocks:

- A strong output-dependent $(1\text{-}\mathsf{parity}, \mathsf{NC}^0[1])$-LTC implementation $\widetilde{C}$ of the functionality $f$ (described later) with an efficient simulator $\mathsf{Sim}'$ (we can just adopt the approach of Theorem 6.5).
- A hard-core function $h : \{0,1\}^\kappa \to \{0,1\}$ against 1-parity leakage (e.g. $h(\boldsymbol{x}) = \langle \boldsymbol{u}, \boldsymbol{v} \rangle$ where $\boldsymbol{x}$ is partitioned into $\boldsymbol{u}$ and $\boldsymbol{v}$ each of length $\kappa/2$).
- A (1-parity)-resilient encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ as defined in Definition 4.3 such that

$$\mathsf{Enc}(c) = \{(\boldsymbol{u}_i, \boldsymbol{v}_i, \boldsymbol{w}_i = \boldsymbol{u}_i \cdot \boldsymbol{v}_i \oplus \widetilde{\boldsymbol{w}}_i)_{i\in[\kappa]} : \boldsymbol{u}, \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa, \widetilde{\boldsymbol{w}} \xleftarrow{\$} \mathsf{PAR}(\kappa, c)\}.$$

[IS24] makes the following useful observation of the encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$:

**Lemma D.1.** *Even if the adversary knows $\boldsymbol{v}$ after choosing the leakage function $L$, the security of $(\mathsf{Enc}, \mathsf{Dec})$ still holds. Specifically, for any 1-parity leakage function $L$, we have:*

$$(L(\mathsf{Enc}(0; \boldsymbol{v})), \boldsymbol{v}) \approx_S (L(\mathsf{Enc}(1; \boldsymbol{v}), \boldsymbol{v}).$$

*Proof.* Note that the leakage function $L$ can be written in the following form:

$$L(\mathsf{Enc}(c;\boldsymbol{v})) = b \bigoplus_{i\in I_1}(\boldsymbol{u}_i \cdot (\boldsymbol{v}_i \oplus b_i) \oplus \widetilde{\boldsymbol{w}}_i) \bigoplus_{i\in I_2} \boldsymbol{u}_i$$

such that $b, b_i \in \{0,1\}$ are some constants and $I_1 \cap I_2 = \emptyset$.

Leaving $\widetilde{\boldsymbol{w}}_0 \xleftarrow{\$} \mathsf{PAR}(\kappa, 0)$ and $\widetilde{\boldsymbol{w}}_1 \xleftarrow{\$} \mathsf{PAR}(\kappa, 1)$, we prove the indistinguishablity by considering the size of the subset $I_1$:

- If $|I_1| < \kappa$, then $\widetilde{\boldsymbol{w}}_{0,I_1}$ and $\widetilde{\boldsymbol{w}}_{1,I_1}$ are identically distributed.
- Otherwise $I_1 = [\kappa]$, with probability $1 - 2^{-\kappa}$ over the randomness of $\boldsymbol{v}$, at least one of the $\boldsymbol{v}_i \oplus b_i$ is nonzero and the output of $L$ is masked by $\boldsymbol{u}_i$.

□

**Construction.** We construct a circuit $C$ that emulates a strong LTC $\widetilde{C}$ computing the following *randomized* functionality $f$:

$$f(\boldsymbol{x}) := (\boldsymbol{v}, \boldsymbol{e} = \mathsf{Enc}(h(\boldsymbol{x}); \boldsymbol{v})), \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa,$$

and set only $\boldsymbol{v}$ to be the output of $C$.

**Strong Output-Independent Leakage-Tolerance.** We prove that the circuit $C$ is also a strong output-independent $(1\text{-}\mathsf{parity}, \mathsf{NC}^0[1])$-LTC by building a simulator $\mathsf{Sim}$:

- Invoke the simulator $\mathsf{Sim}'$ of $\widetilde{C}$ to express the wire values as $\mathsf{NC}^0[1]$ functions $L_W$ of $(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{e})$.
- Fix the internal randomness of $\mathsf{Enc}$ (excluding $\boldsymbol{v}$), then output the wire value function $L'_W = L_W(\boldsymbol{x}, \boldsymbol{v}, \mathsf{Enc}(0; \boldsymbol{v}))$ that only takes $(\boldsymbol{x}, \boldsymbol{v})$ as input.

  The security of $\mathsf{Sim}$ follows from the security of $\mathsf{Sim}'$ and Lemma D.1.

**Strong Output-Dependent Leakage-Tolerance.** We claim that it is impossible to simulate an adversary $\mathcal{A}$ which outputs an output-dependent leakage function $L \in 1\text{-}\mathsf{parity}$, where $L$ takes input $\boldsymbol{e} = (\boldsymbol{u}_i, \boldsymbol{v}_i, \boldsymbol{w}_i)_{i\in[\kappa]}$ and computes the parity of $S_{\boldsymbol{v}}$. Here, $S_{\boldsymbol{v}}$ is defined as follows: for each index $i \in [\kappa]$,

- If $\boldsymbol{v}_i = 0$, add $\boldsymbol{w}_i$ to $S_{\boldsymbol{v}}$.
- Otherwise, add both $\boldsymbol{u}_i$ and $\boldsymbol{w}_i$ to $S_{\boldsymbol{v}}$.

We observe that the imperfect correctness of the underlying strong output-independent LTC $\widetilde{C}$ guarantees for every $\boldsymbol{x} \in \{0,1\}^\kappa$:

$$\{(\boldsymbol{v}, L(\boldsymbol{e})) : \boldsymbol{v}, \boldsymbol{e} \leftarrow W[C], L \leftarrow \mathcal{A}(\boldsymbol{v})\} \approx_S \{(\boldsymbol{v}, h(\boldsymbol{x})) : \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa\}.$$

Conversely, assume that there exists a strong output-dependent LTC simulator $\mathsf{Sim}'$ outputting a wire value function $L_{\boldsymbol{e}} \in \mathsf{NC}^0[1]$ on inputs $\boldsymbol{x}, \boldsymbol{v}$ such that for every $\boldsymbol{x} \in \{0,1\}^\kappa$:

$$\{(\boldsymbol{v}, h(\boldsymbol{x})) : \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa\} \approx_S \{(\boldsymbol{v}, L \circ L_{\boldsymbol{e}}(\boldsymbol{x}, \boldsymbol{v})) : \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa, L \leftarrow \mathcal{A}(\boldsymbol{v})\}.$$

Then we randomly sample input $\boldsymbol{x}, \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa$ and determine $L$ according to $\boldsymbol{v}$. As a result, $L \circ L_{\boldsymbol{e}}$ can be realized as a 1-parity function $L'$ of $\boldsymbol{x}$ such that

$$\Pr[L'(\boldsymbol{x}) = h(\boldsymbol{x}) : \boldsymbol{x}, \boldsymbol{v} \xleftarrow{\$} \{0,1\}^\kappa, L \leftarrow \mathcal{A}(\boldsymbol{v}), L' \leftarrow L \circ L_{\boldsymbol{e}}] \geq 1 - \mathsf{negl}(\kappa).$$

Fixing $\boldsymbol{v}$ to be a string that maximizes the adversary's advantage, we break the security of the hard-core function $h$, which causes contradiction.

# E Deferred Proof of Theorem 7.3

## E.1 Simulator Construction

Recall that in our scheme, the global randomness extractor is applied on $R_i = (\widetilde{\boldsymbol{s}_i} \parallel \mathsf{Rtape}_i)$ of length $(|\widehat{\boldsymbol{s}_i}| + |r|)I$ in the $i$-th invocation: the first $|\widehat{\boldsymbol{s}_i}|I$ bits are used to extract $\widehat{\boldsymbol{s}_i}$ and the last $|r|I$ bits are used to obtain $\mathsf{Rtape}'_i$. In the simulated mode $((\ell_j^{(i)})_{j\in[mm']}, (\boldsymbol{r}_{j,0}^{(i)}, \boldsymbol{r}_{j,1}^{(i)})_{j\in[k]}) \leftarrow \mathsf{SimExt}$, the wire values of the randomness extractor $W[\mathsf{Ext}(R_i)]$ can be written as $\mathsf{NC}^0[1]$ functions of $R'_i = (\widehat{\boldsymbol{s}_i} \parallel \mathsf{Rtape}'_i)$:

$$L_W^{(i)}(R'_i) = \left((1 - R'_{i,j}) \cdot W[\mathsf{Ext}_j(\boldsymbol{r}_{j,0}^{(i)})] \bigoplus R'_{i,j} \cdot W[\mathsf{Ext}_j(\boldsymbol{r}_{j,1}^{(i)})]\right)_{j\in[k]}.$$

Then for any stateful P.P.T. adversary $\mathcal{A}$, our simulator $\mathsf{SimDLRC}$ does the following:

- Set $\widehat{\boldsymbol{s}_0} = \mathsf{Enc}(\boldsymbol{s}_0 \parallel \mathsf{Seed}_0)$ to be a random encoding of an all-0 string.
- Find a constant function $L_0 \in \mathcal{L}$ of input size $|\widehat{\boldsymbol{s}_0}| + |\mathsf{Rtape}_0|$, and invoke the randomness extractor simulator $\mathsf{SimExt}(L_0)$ to obtain $(\boldsymbol{r}_{j,0}^{(0)}, \boldsymbol{r}_{j,1}^{(0)})_{j\in[k]}$.
- In the $i$-th invocation, given the leakage query $L_i \in \mathcal{L}$ asked by $\mathcal{A}$:
  - Invoke $f[\boldsymbol{s}_{i-1}](\boldsymbol{x}_i)$ on public input $\boldsymbol{x}_i$ to obtain public output $\boldsymbol{y}_i$.
  - Adaptively run the LTC simulator $\mathsf{SimRELTC}$ on $L_i \circ L_R^{(i-1)} \in \mathcal{L}$ to obtain a leakage query $L'_i \in \mathcal{L}'$ that takes inputs $\widehat{\boldsymbol{s}_{i-1}}, \boldsymbol{x}_i, \boldsymbol{y}_i, \widetilde{\boldsymbol{s}_i}$ and $\mathsf{Rtape}_i$. Since the first three inputs are all known by $\mathsf{SimDLRC}$, such a leakage query can be realized as an equivalent leakage query $L''_i \in \mathcal{L}'$ that only takes $R_i = (\widetilde{\boldsymbol{s}_i} \parallel \mathsf{Rtape}_i)$ as input. From this, $\mathsf{SimDLRC}$ runs the extractor simulator $\mathsf{SimExt}$ on $L''_i$ simultaneously to obtain $((\ell_j^{(i)})_{j\in[mm']}, (\boldsymbol{r}_{j,0}^{(i)}, \boldsymbol{r}_{j,1}^{(i)})_{j\in[k]})$.
  - Set $\widehat{\boldsymbol{s}_i} = \mathsf{Enc}(\boldsymbol{s}_i \parallel \mathsf{Seed}_i)$ to be a random encoding of an all-0 string.

## E.2 Security Proof

We argue for the security using straightforward hybrid arguments. Let the polynomial $q = q(\kappa)$ be the number of queries made by the adversary $\mathcal{A}$.

**Hybrid $\mathsf{Hyb}_0$.** This hybrid is the real world execution.

**Hybrid $\mathsf{Hyb}_1$.** In this hybrid, we generate $\widetilde{\boldsymbol{s}_0}$ and $\mathsf{Rtape}_0$ as follows: invoke $\mathsf{SimExt}$ on a constant function $L_0 \in \mathcal{L}$ to obtain $(\boldsymbol{r}_{j,0}^{(0)}, \boldsymbol{r}_{j,1}^{(0)})_{j\in[k]}$. Then sample $\mathsf{Rtape}'_0 \xleftarrow{\$} \{0,1\}^{|r|}$, set $\widetilde{\boldsymbol{s}_0} = \left(\boldsymbol{r}_{j,(\widehat{\boldsymbol{s}_0})_j}^{(0)}\right)_{j\in[|\widehat{\boldsymbol{s}_0}|]}$ and $\mathsf{Rtape}_0 = \left(\boldsymbol{r}_{j+|\widehat{\boldsymbol{s}_0}|,(\mathsf{Rtape}'_0)_j}^{(0)}\right)_{j\in[|r|]}$.

To argue for security, we intend to apply the extractable property of the global randomness extractor $\mathsf{Ext}$. However, the context in $\mathsf{Hyb}_0$ does not align with Definition 7.1: $\mathsf{Rtape}_0$ is a random string whereas Definition 7.1 requires the input to be chosen from preimage of a fixed sequence $\boldsymbol{b}$. We may address this issue by showing a relaxed version of Definition 7.1:

**Lemma E.1.** *Let $\mathsf{Ext}$ be an $(\mathcal{L}, \epsilon)$-global randomness extractor with reverse-sampling. For all $S \subseteq [n]$, randomly sample variables $\boldsymbol{b}_S$ from $\{0,1\}^{|S|}$ and fix the rest of the variables $\boldsymbol{b}_{[n]/S}$, then*

*we have the extractable property over $S$:*

$$\begin{aligned}
&\{((L_i(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_n))_{i\in[m]},(\boldsymbol{r}_i)_{i\in[n]}):\\
&\forall i\in S,\boldsymbol{r}_i\xleftarrow{\$}\{0,1\}^I;\forall i\notin S,\boldsymbol{r}_i\xleftarrow{\$}\mathsf{Preimage}(\boldsymbol{b}_i)\}\\
\approx_{\epsilon'}&\{((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,\boldsymbol{b}_i})_{i\in[n]}):\\
&\boldsymbol{b}_S\xleftarrow{\$}\{0,1\}^{|S|},((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,0},\boldsymbol{r}_{i,1})_{i\in[n]})\leftarrow\mathsf{SimExt}(L)\}
\end{aligned}$$

*for some $\epsilon'=\epsilon+\mathsf{negl}(\kappa)$. Specifically, if $\epsilon=\mathsf{negl}(\kappa)$, then $\epsilon'$ is also negligible in $\kappa$.*

*Proof.* The regular property ensures that

$$\begin{aligned}
&\{((L_i(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_n))_{i\in[m]},(\boldsymbol{r}_i)_{i\in[n]}):\\
&\forall i\in S,\boldsymbol{r}_i\xleftarrow{\$}\{0,1\}^I;\forall i\notin S,\boldsymbol{r}_i\xleftarrow{\$}\mathsf{Preimage}(\boldsymbol{b}_i)\}\\
\approx_{\mathsf{negl}(\kappa)}&\{((L_i(\boldsymbol{r}_1,\ldots,\boldsymbol{r}_n))_{i\in[m]},(\boldsymbol{r}_i)_{i\in[n]}):\\
&\boldsymbol{b}_S\xleftarrow{\$}\{0,1\}^S,\boldsymbol{r}_i\xleftarrow{\$}\mathsf{Preimage}(\boldsymbol{b}_i)\},
\end{aligned}$$

and the RHS is $\epsilon$-close to

$$\{((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,\boldsymbol{b}_i})_{i\in[n]}):\boldsymbol{b}_S\xleftarrow{\$}\{0,1\}^{|S|},((\ell_i)_{i\in[m]},(\boldsymbol{r}_{i,0},\boldsymbol{r}_{i,1})_{i\in[n]})\leftarrow\mathsf{SimExt}(L)\}$$

following the extractable property Equation (1) of a fixed sequence by taking random $\boldsymbol{b}_S\xleftarrow{\$}\{0,1\}^S$. □

By the extractable property Lemma E.1 over $S=\{|\widehat{\boldsymbol{s}_0}|+1,\ldots,|r|\}$, $\mathsf{Hyb}_1$ is statistically close to $\mathsf{Hyb}_0$.

**Hybrid $\mathsf{Hyb}_{i+1}$ ($i\in\{1,2,\ldots,q\}$).** In the following small hybrids, we switch to simulate the leakage query $L_i$ and the function output of the $i$-th invocation:

**Hybrid $\mathsf{Hyb}_{i+1,0}$.** This hybrid is identical to $\mathsf{Hyb}_i$.

**Hybrid $\mathsf{Hyb}_{i+1,1}$.** In this hybrid, we randomly sample $\mathsf{Rtape}'_i\xleftarrow{\$}\{0,1\}^{|r|}$ and change $W[\mathsf{Ext}(R_{i-1})]$ (including $R_{i-1}$) to $L_W^{(i-1)}(\widehat{\boldsymbol{s}_{i-1}}\parallel\mathsf{Rtape}'_{i-1})$ accordingly. Clearly, $\mathsf{Hyb}_{i+1,1}$ and $\mathsf{Hyb}_{i+1,0}$ are identically distributed.

**Hybrid $\mathsf{Hyb}_{i+1,2}$.** In this hybrid, we invoke $f'[\widehat{\boldsymbol{s}_{i-1}}](\boldsymbol{x}_i)$ to obtain $(\widetilde{\boldsymbol{s}}_i,\boldsymbol{y}_i,\mathsf{Rtape}_i)$, and run the LTC simulator $\mathsf{SimRELTC}(L_i\circ L_W^{(i-1)})$ to obtain a simulated leakage query $L'_i\in\mathcal{L}'$. Finally, output the simulated query result given by
$L'_i(\widehat{\boldsymbol{s}_{i-1}},\boldsymbol{x}_i,\boldsymbol{y}_i,\widetilde{\boldsymbol{s}}_i,\mathsf{Rtape}_i)$.

By the security of the underlying randomness-efficient LTC scheme, $\mathsf{Hyb}_{i+1,2}$ is statistically close to $\mathsf{Hyb}_{i+1,1}$.

**Hybrid $\mathsf{Hyb}_{i+1,3}$.** In this hybrid, the encoded state $\widehat{\boldsymbol{s}_{i-1}}=\mathsf{Enc}(\boldsymbol{s}_{i-1}\parallel\mathsf{Seed}_{i-1})$ is replaced by a random encoding of an all-0 string.

Note that $\widehat{\boldsymbol{s}_{i-1}}$ is only used to compute the result of the $\mathcal{L}'$-leakage query $L'_i$. By the security of the $\mathcal{L}'$-resilient encoding scheme, $\mathsf{Hyb}_{i+1,3}$ and $\mathsf{Hyb}_{i+1,2}$ are statistically close.

**Hybrid** $\mathsf{Hyb}_{i+1,4}$**.** In this hybrid, we replace the output of $\mathsf{PRG}$ used in $f'[\widehat{\boldsymbol{s}_{i-1}}](\boldsymbol{x}_i)$ with a random string. Note that $\widehat{\boldsymbol{s}_i}$ becomes the random encoding of $(\boldsymbol{s}_i \parallel \mathsf{Seed}_i)$ and the preimage $\widetilde{\boldsymbol{s}_i}$ is now sampled by $\mathsf{Sext}$ with uniform internal randomness.

As now the seed $\mathsf{Seed}_{i-1}$ is never used anywhere, the security follows from the pseudorandomness of $\mathsf{PRG}$: we have that $\mathsf{Hyb}_{i+1,4}$ is computationally indistinguishable from $\mathsf{Hyb}_{i+1,3}$.

**Hybrid** $\mathsf{Hyb}_{i+1,5}$**.** In this hybrid, $\widetilde{\boldsymbol{s}_i}$ is uniformly sampled from $\mathsf{Preimage}(\widehat{\boldsymbol{s}_i})$. By the security of the sampler $\mathsf{Sext}$, $\mathsf{Hyb}_{i+1,5}$ is statistically close to $\mathsf{Hyb}_{i+1,4}$.

**Hybrid** $\mathsf{Hyb}_{i+1,6}$**.** In this hybrid, the leakage query $L'_i$ is converted to query $L''_i$ on the input $R_i = (\widetilde{\boldsymbol{s}_i} \parallel \mathsf{Rtape}_i)$ by substituting simulated values. As this does not change the output distribution, $\mathsf{Hyb}_{i+1,6}$ and $\mathsf{Hyb}_{i+1,5}$ are identically distributed.

**Hybrid** $\mathsf{Hyb}_{i+1,7}$**.** In this hybrid, we invoke $\mathsf{SimExt}(L''_i)$ to obtain $((\ell_j^{(i)})_{j\in[mm']}, (\boldsymbol{r}_{j,0}^{(i)}, \boldsymbol{r}_{j,1}^{(i)})_{j\in[k]})$. Then sample $\mathsf{Rtape}'_i \xleftarrow{\$} \{0,1\}^{|r|}$, set $\mathsf{Rtape}_i = \left(\boldsymbol{r}^{(i)}_{j+|\widehat{\boldsymbol{s}_i}|,(\mathsf{Rtape}'_i)_j}\right)_{j\in[|r|]}$ and $\widetilde{\boldsymbol{s}_i} = \left(\boldsymbol{r}^{(i)}_{j,(\widehat{\boldsymbol{s}_i})_j}\right)_{j\in[|\widehat{\boldsymbol{s}_i}|]}$.

Again following the extractable property Lemma E.1 over $S = \{|\widehat{\boldsymbol{s}_0}|+1, \ldots, |r|\}$, we find that $\mathsf{Hyb}_{i+1,7}$ is statistically close to $\mathsf{Hyb}_{i+1,6}$.

**Hybrid** $\mathsf{Hyb}_{q+2}$**.** In this hybrid, we change $\widehat{\boldsymbol{s}_q}$ to a random encoding of all-0 string and modify $\widetilde{\boldsymbol{s}_q}$ accordingly. Because we compute the leakage query result $\ell_q$ without relying on $\widetilde{\boldsymbol{s}_q}$, $\mathsf{Hyb}_{q+2}$ is identical to $\mathsf{Hyb}_{q+1}$.

We finish the proof by noticing that $\mathsf{Hyb}_{q+2}$ is the ideal world distribution.

# F Deferred Proof of Lemma 8.1

## F.1 Additional Preliminaries

A *restriction* $\rho$ of variables $x_1, \ldots, x_n$ is defined as an element of $\{0,1,*\}^n$. The size $|\rho|$ is defined as the number of fixed variables ($\boldsymbol{x}_i \in \{0,1\}$). For a given function $f(x_1, \ldots, x_n)$ and restriction $\rho$, we use $f \restriction \rho$ to denote the function obtained by restricting $x_i$ to be $\rho_i$ if $\rho_i \in \{0,1\}$ and free if $\rho_i = *$. The composition of two restrictions $\rho, \rho'$, denoted by $\rho\rho'$, can be defined intuitively:

$$(\rho\rho')_i = \begin{cases} \rho_i, & \text{if } \rho_i \in \{0,1\} \\ \rho'_i, & \text{if } \rho_i = *. \end{cases}$$

For a function family $\mathcal{F} = (F_1, \ldots, F_s)$ and a restriction $\rho$, we write $\mathcal{F} \restriction \rho$ to denote $(F_1 \restriction \rho, \ldots, F_s \restriction \rho)$.

A *decision tree* is a tree in which each internal node is labeled by a boolean variable, and each leaf is assigned a boolean value. A decision tree defines the computation of a boolean function by picking one path and traversing down the tree with respect to the internal variables and their assigned values, outputting the value of the reaching leaf. We use $\mathsf{depth}(T)$ to denote the depth of a decision tree $T$.

We then define the notion of canonical decision trees for CNFs. Note that the dual form of the following results in this section holds for DNFs.

**Definition F.1** (Canonical Decision Tree)**.** Given a CNF $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, we say that a decision tree is the canonical decision tree of $F$ ($\mathsf{CDT}(F)$ for short) if it is obtained in the following way:

- If any clause $C_i$ is identical to 0, the tree is all-0.
- If every clause $C_i$ is identical to 1, the tree is all-1.
- Otherwise, let $C_{i_1}$ be the first clause that is not identically 1 and $S$ be all the free variables of $C_{i_1}$. We create a depth-$|S|$ decision tree that exhaustively queries all the variables of $S$. For each leaf corresponding to a restriction $\sigma$ of variables in $S$, we attach a canonical decision tree $\mathsf{CDT}(F \restriction \sigma)$ to it.

The canonical decision tree is widely used to analyze the circuit lower bound of $k$-CNFs [Has86]. Let $\mathcal{R}_p$ denote the distribution of random restrictions such that $\mathcal{R}_p$ independently sets each variable to 0 or 1 with identical probability $(1-p)/2$, and $*$ with probability $p$. We have:

**Lemma F.2** (Hastad Switching Lemma)**.** *Given any $k$-CNF $F$, for any $t \geq 1, p > 0$:*

$$\Pr_{\rho \leftarrow \mathcal{R}_p}[\mathsf{depth}(\mathsf{CDT}(F \restriction \rho)) > t] \leq (5pk)^t.$$

As a direct corollary, given a function family $\mathcal{F} = (F_1, \ldots, F_s)$, by taking $p \leq 1/(10k)$, we can show that for $\epsilon = \mathsf{negl}(\kappa)$, with a probability of at least $1-\epsilon$, the depth of each canonical decision tree $\mathsf{CDT}(F \restriction \rho)$ is bounded by $O(\log(s/\epsilon))$ for a common restriction $\rho \leftarrow \mathcal{R}_p$. Unfortunately, this falls far short of our purpose, since $\log(s/\epsilon)$ is superlogarithm in $\kappa$ and we cannot hope to build such a decision tree in polynomial time. Therefore, we utilize a modern result – multi-switching Hastad lemma [IMP12, Has14, ST22]. We first explain some terminologies and notations:

**Definition F.3** (Common Partial Decision Tree)**.** Let $\mathcal{F} = (F_1, \ldots, F_s)$ be a collection of $k$-CNFs. We say that a decision tree $T$ is a common $\ell$-partial decision tree for $\mathcal{F}$, if each $F_i$ can be calculated by attaching $T$ a depth-$\ell$ decision tree to each leaf.

**Definition F.4** (Full Paths in the CDT)**.** Given any $k$-CNF $F = C_1 \wedge C_2 \wedge \ldots C_m$. Each path $\eta$ of the canonical decision tree $\mathsf{CDT}(F)$ can be written as the disjoint union of segments $\eta = \eta^{(1)} \circ \eta^{(2)} \circ \ldots \circ \eta^{(u)}$, where $\eta^{(j)}$ is the assignment of some free variables of $C_{i_j}$ such that $C_{i_j}$ is the first clause in $F \restriction (\eta^{(1)} \circ \ldots \circ \eta^{(j-1)})$ that is not identically 1.

Furthermore, for each $j \in [u-1]$, it is clear that the assignment $\eta^{(j)}$ contains all free variables in $C_{i_j} \restriction (\eta^{(1)} \circ \ldots \circ \eta^{(j-1)})$, and we say that $\eta$ is *full* if $\eta^{(u)}$ also contains all free variables in $C_{i_u} \restriction (\eta^{(1)} \circ \ldots \circ \eta^{(u-1)})$.

**Fact F.5.** *Let $F$ be a $k$-CNF. Suppose $\mathsf{depth}(\mathsf{CDT}(F)) > \ell$, then there exists a full path $\eta$ such that $|\eta| \in [\ell+1, \ell+k]$.*

*Proof.* Let $\eta'$ be a root-to-leaf path of $\mathsf{CDT}(F)$ where $|\eta'| > \ell$. By definition $\eta'$ itself is a full path that can be written in the form of $\eta^{(1)} \circ \eta^{(2)} \circ \ldots \circ \eta^{(r)}$ where each $\eta^{(i)}$ has a length at most $k$. Thus, there exists some $u \in [r]$ such that $|\eta^{(1)}| + \ldots + |\eta^{(u)}| \in [\ell+1, \ell+k]$. It is clear that $\eta = \eta^{(1)} \circ \ldots \circ \eta^{(u)}$ is a full path of length between $\ell+1$ and $\ell+k$. □

**Definition F.6** (Canonical Common $\ell$-Partial Decision Tree)**.** Let $\mathcal{F} = (F_1, \ldots, F_s)$ be an ordered collection of $k$-CNFs. The set of *all* canonical common $\ell$-partial decision trees (denoting as $\mathsf{CCDT}_\ell(\mathcal{F})$) is defined as follows:

- If $s = 0$, $\mathsf{CCDT}_\ell(\mathcal{F})$ only consists of an empty tree with no node.
- Otherwise, $s \geq 1$. If $\mathsf{depth}(\mathsf{CDT}(F_1)) \leq \ell$, $\mathsf{CCDT}_\ell(\mathcal{F})$ is simply $\mathsf{CCDT}_\ell(\mathcal{F}')$ where $\mathcal{F}' = (F_2, \ldots, F_s)$.

- Otherwise, $\mathsf{depth}(\mathsf{CDT}(F_1)) > \ell$. Let $P$ be the set of all full paths $\eta$ of length between $\ell+1$ and $\ell+k$ in $\mathsf{CDT}(F_1)$ (by Fact F.5, $P \neq \emptyset$). Then for each path $\eta \in P$, let $T_\eta$ be the depth-$|\eta|$ decision tree obtained by exhaustively querying all the variables in $\eta$. For each leaf of $T_\eta$ corresponding to restriction $\sigma$, attach tree $T' \in \mathsf{CCDT}_\ell(\mathcal{F} \restriction \sigma)$ to it. $\mathsf{CCDT}_\ell(\mathcal{F})$ is then defined as all the possible decision trees that can be obtained in this way.

We define $\mathsf{depth}(\mathsf{CCDT}_\ell(\mathcal{F}))$ as the maximum depth of any tree in $\mathsf{CCDT}_\ell(\mathcal{F})$.

The following multi-switching Hastad lemma establishes the relation between canonical common decision tree and circuit lower bound of $k$-CNFs family:

**Lemma F.7** ( [ST22, Theorem 3.8]). *Let $\mathcal{F} = (F_1, \ldots, F_s)$ be an ordered collection of $k$-CNFs. Then for all $\ell, t \geq 1, p > 0$, we have:*

$$\Pr_{\rho \leftarrow \mathcal{R}_p}[\mathsf{depth}(\mathsf{CCDT}_\ell(\mathcal{F})) > t] \leq s^{\lceil t/\ell \rceil}(32pk)^t.$$

## F.2 Simulator for Single $\mathsf{AC}^0(d, s)$ Leakage Query

Let $\epsilon_0 = \mathsf{negl}(\kappa)$ be a tolerance error, we consider the case of a single leakage query $L \in \mathsf{AC}^0(d, s)$, and show that we can construct a P.P.T. simulator $\mathsf{SimQ}$ in which with high probability it samples a random restriction $\rho \leftarrow \mathcal{R}_{\Theta(1/\log^d s)}$ together with a query-specific random restriction $\pi_L$ ($|\pi_L| = O(d \log(s/\epsilon_0))$), such that the result of $L$ is fixed under $\rho\pi_L$.

For simplicity, we replace each input wire of $L$ by an identity gate of fan-in 1 (which increases the circuit depth by 1), and make $d$ iterative applications of Lemma F.7 to the bottom-layer depth-2 subcircuits of $L$. We inductively prove that each of them has a polynomial size and a width at most $\ell = \log(2s)$.

- The above claim trivially holds for $i = 0$ since we have already fixed each bottom gate to have a fan-in 1.
- Suppose that the claim holds for $i = i^*$. Let $\mathcal{F}$ be the collection of polynomial-sized bottom layer depth-2 subcircuits; obviously $|\mathcal{F}| \leq s$. Choose $t = \log(s/\epsilon_0)$ and $p = 1/(128\ell) = \Theta(1/\log s)$. The simulator $\mathsf{SimQ}$ draws a random restriction $\rho_i \leftarrow \mathcal{R}_p$.
- Let $\mathcal{F}' = \mathcal{F} \restriction \rho_i$. $\mathsf{SimQ}$ samples a random root-to-leaf path on $\mathsf{CCDT}_\ell(\mathcal{F}')$ following the definition of $\mathsf{CCDT}_\ell$, and maintains a random restriction $\pi_{L,i}$:
  - If $|\mathcal{F}'| = 0$, finish.
  - Otherwise, $\mathsf{SimQ}$ can determine whether $\mathsf{depth}(\mathsf{CDT}(F_1)) \leq \ell$ in polynomial time $O(2^\ell) = O(s)$, in which $F_1$ is the first element of $\mathcal{F}'$. If so, it recursively samples $\mathsf{CCDT}_\ell(\mathcal{F}'/F_1)$.
  - Otherwise, $\mathsf{SimQ}$ can still compute the first $2\ell$ layers of $\mathsf{CDT}(F_1)$ in polynomial time $O(2^{2\ell}) = O(s^2)$, and find the lexicographically minimum full path $\eta$ of length between $\ell+1$ and $2\ell$. $\mathsf{SimQ}$ then samples all the variables in $\eta$, adds the query result $\sigma$ to $\pi_{L,i}$, and recursively samples $\mathsf{CCDT}_\ell(\mathcal{F}' \restriction \sigma)$.

  By Lemma F.7, with probability $1 - s^{\lceil t/l \rceil}(32p\ell)^t \geq 1 - \epsilon_0$, the tree $\mathsf{CCDT}_\ell(\mathcal{F}')$ has depth at most $t$, in this case $\mathsf{SimQ}$ obtains a random restriction $\pi_{L,i}$ such that $|\pi_{L,i}| \leq t$.
- For each $F_i \in \mathcal{F}$, we have that $\mathsf{CDT}(F_i \restriction \rho_i\pi_{L,i}) \leq \ell$ following the definition of a common $\ell$-partial decision tree, and such canonical decision tree can be built by $\mathsf{SimQ}$ in polynomial time. With the fact that a depth-$\ell$ decision tree can be expressed as both an $\ell$-CNF and an $\ell$-DNF of size $\ell 2^\ell = O(s \log s)$, the circuit depth decreases by 1 and each of the bottom-layer depth-2 subcircuits fan-in $\ell = \log(2s)$.

In summary, after $d$ iterative executions, with high probability $1 - d\epsilon_0$ the value of $L$ on restriction $\rho_1\pi_{L,1}\rho_2\pi_{L,2}\dots\rho_d\pi_{L,d}$ is fixed. We finish the proof by denoting $\rho_1\dots\rho_d$ as $\rho$ and $\pi_{L,1}\dots\pi_{L,d}$ as $\pi_L$.

### F.3 Simulator for Adaptive $\mathsf{AC}^0(d,s)^m$ Query

To handle a $m$-output adaptive leakage query $L \in \mathsf{AC}^0(d,s)^m$, we construct a simulator $\mathsf{SimExt}$ that emulates the random source and interacts with $\mathsf{SimQ}$ $m$ rounds. There are two main technical issues:

1. The randomness used in $m$ consecutive runs of $\mathsf{SimQ}$ may not be consistent.
2. An adversary $\mathcal{A}$ may adaptively choose the query depending on the previous query results. Therefore, the collection $\mathcal{F}$ can be correlated with the random restriction $\rho$, which breaks the security guarantee of Lemma F.7.

We address the first issue, assuming that the adversary is non-adaptive. Denote the non-adaptive query sequence as $L = (L_1, L_2, \dots, L_m)$. The simulator $\mathsf{SimExt}$ behaves as a random oracle: it pre-samples restriction $\rho_1, \dots, \rho_d \leftarrow \mathcal{R}_{\Theta(1/\log s)}$ before interacting with $\mathsf{SimQ}$; whenever $\mathsf{SimQ}$ asks for the value of a variable $x$ in the full path $\eta$ of query $L_j$, it responds with the recorded value if $x$ is fixed by $\rho_1\dots\rho_d\pi_{L_1}\dots\pi_{L_{j-1}}$, and a random value otherwise. It is clear that this modification does not change the output distribution.

The second issue can be fixed by the following lemma which amplifies the (sub-exponential) non-adaptive security to adaptive security:

**Lemma F.8.** *Given any set $S$, consider the following $m$-round two party communication protocol:*

- *In the $i$-th round, $P_A$ sends an element $\boldsymbol{u}_i \in S$ to $P_B$ and obtains a $t$-bit response $\boldsymbol{v}_i$.*
- *After $m$ rounds, $P_B$ may output some arbitrary string $\boldsymbol{y}$ depending on its view.*

*Assume that there exist two parties $P_B$ and $P'_B$, such that the transcripts $\tau$ of $(P_A, P_B)$ and $(P_A, P'_B)$ are $\epsilon$-indistinguishable for all non-adaptive $P_A$ (that is, $P_A$ chooses $\boldsymbol{u}_i$ regardless of messages $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_{i-1}$), then the transcripts $\tau$ are also $2^{tm}\epsilon$-indistinguishable for all adaptive $P'_A$ ($P'_A$ may adaptively choose $\boldsymbol{u}_i$ depending on $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_{i-1}$).*

*Proof.* Suppose that there exists some $(P'_A, P_B, P'_B)$ that breaks the adaptive indistinguishability property (without loss of generality, we may assume that $P'_A$ is deterministic and $(\boldsymbol{u}_i)_{i\in[m]}$ can be inferred from $(\boldsymbol{v}_i)_{i\in[m]}$). For any response sequence $(\boldsymbol{v}_1, \dots, \boldsymbol{v}_m) \in \{0,1\}^{tm}$, we denote the statistical distance conditioned on $(\boldsymbol{v}_1, \dots, \boldsymbol{v}_m)$ as:

$$\begin{aligned}&\Delta(\boldsymbol{v}_1, \dots, \boldsymbol{v}_t)\\ :=&\frac{1}{2}\sum_{\boldsymbol{y}}\left|\Pr_{(P_A,P_B)}[\tau = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_m, \boldsymbol{y})] - \Pr_{(P_A,P'_B)}[\tau = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_m, \boldsymbol{y})]\right|,\end{aligned}$$

then we have

$$\sum_{(\boldsymbol{v}_1,\dots,\boldsymbol{v}_t)} \Delta(\boldsymbol{v}_1, \dots, \boldsymbol{v}_t) \geq 2^{tm}\epsilon.$$

Using the averaging argument, there must exist $(\boldsymbol{v}^*_1, \dots, \boldsymbol{v}^*_m) \in \{0,1\}^{tm}$ such that $\Delta(\boldsymbol{v}^*_1, \dots, \boldsymbol{v}^*_m) \geq \epsilon$. Now, let $P_A$ be a party that outputs non-adaptive queries $(L^*_i \leftarrow P'_A(\boldsymbol{v}^*_1, \dots, \boldsymbol{v}^*_{i-1}))_{i\in[m]}$; it is clear that the transcripts of $(P_A, P_B)$ and $(P_A, P'_B)$ can be $\epsilon$-distinguished, which contradicts our hypothesis. □

*Proof of Lemma 8.1.* Choose $\epsilon_0 = 2^{-m}\epsilon$. By an union bound we can show that $\mathsf{SimExt}$ simulates any non-adaptive leakage queries $(L_1, \ldots, L_m)$ with statistical distance $md\epsilon_0$. Now, let $(\boldsymbol{b}_i)_{i\in[n]}$ be any fixed sequence and $\mathcal{O}$ serve as an oracle that samples $\boldsymbol{r}_i \xleftarrow{\$} \mathsf{PAR}(I, \boldsymbol{b}_i)$ and answers the leakage query faithfully. Now apply Lemma F.8 on $(\mathsf{SimExt}, \mathcal{O})$, it is obvious that with probability $1 - md2^m\epsilon_0 = 1 - \mathsf{negl}(\kappa)$, $\mathsf{SimExt}$ succeeds to simulate any adaptive query $L \in \mathsf{AC}^0(d, s)^m$. Define $\rho = \rho_1 \ldots \rho_d$ and $\pi_L = \pi_{L_1} \ldots \pi_{L_m}$, $L(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$ is fixed by $\rho\pi_L$ such that $\rho \leftarrow \mathcal{R}_{p^*} = \mathcal{R}_{\Theta(1/\log^d s)}$ and $|\pi_L| = O(md\log(s/\epsilon_0))$.

Choose $I = O(md\log(s/\epsilon_0)\log^d s)$ such that $p^*I \geq 2|\pi_L|$. Applying Chernoff bound and union bound, with probability at least $1 - n\exp(-\Theta(p^*I)) = 1 - \mathsf{negl}(\kappa)$, there is no index $i \in [n]$ such that the variables $\{\boldsymbol{r}_{i,1}, \ldots, \boldsymbol{r}_{i,I}\}$ are totally fixed by $\rho\pi_L$. Therefore, $\mathsf{SimExt}$ can randomly sample $\boldsymbol{r}_{i,0}, \boldsymbol{r}_{i,1}$ conditioned on $\rho\pi_L$ for every $i \in [n]$, such that $\mathsf{Ext}_i(\boldsymbol{r}_{i,0}) = 0$ and $\mathsf{Ext}_i(\boldsymbol{r}_{i,1}) = 1$. Since all the variables are randomly sampled, the output distribution is statistically close to uniformly random distribution. □

# G Randomness-Efficient LTCs Against $k$-CNF Leakage

In this section, we instantiate the idea of Section 8.2 to obtain a randomness-efficient LTC against $k$-CNF leakage. Recall that our construction can be divided into two parts: (1) a $k$-CNF internal-to-external generator $\mathsf{Gen}$ as an internal construction; (2) a $(k\text{-CNF}, \mathsf{Gen})$-LTC $C$ as an external construction. We construct them in Appendix G.1 and Appendix G.2 respectively, then put them together to prove Theorem 8.4.

In our construction, we leverage a pseudorandom generator that fools all $\mathsf{AC}^0$ adversaries. Analogously to the security requirements of $t$-wise independent PRG used in [GIS22, IS25], we require that the $\mathsf{AC}^0$-resilient PRG be linear and (more importantly) strong:

**Definition G.1.** A function $G : \{0,1\}^\lambda \to \{0,1\}^n$ is an $(\mathcal{L}, \epsilon)$-resilient PRG that $\epsilon$-fools all $\mathcal{L}$ circuits if, for all $\mathcal{A} \in \mathcal{L}$:

$$\left| \Pr_{\boldsymbol{x}\xleftarrow{\$}\{0,1\}^\lambda}[\mathcal{A}(G(\boldsymbol{x})) = 1] - \Pr_{\boldsymbol{y}\xleftarrow{\$}\{0,1\}^n}[\mathcal{A}(\boldsymbol{y}) = 1] \right| < \epsilon.$$

If the above inequality holds even when $\mathcal{A}$ additionally takes $\boldsymbol{x}$ as input, we say that $G$ is *strong*:

$$\left| \Pr_{\boldsymbol{x}\xleftarrow{\$}\{0,1\}^\lambda}[\mathcal{A}(\boldsymbol{x}, G(\boldsymbol{x})) = 1] - \Pr_{\boldsymbol{x}\xleftarrow{\$}\{0,1\}^\lambda, \boldsymbol{y}\xleftarrow{\$}\{0,1\}^n}[\mathcal{A}(\boldsymbol{x}, \boldsymbol{y}) = 1] \right| < \epsilon.$$

If each output bit of $G(\boldsymbol{x})$ is $\mathsf{XOR}$ of a subset of $\boldsymbol{x}$, we say that $G$ is *linear*.

We claim that a standard Nisan-Wigderson style PRG [NW94] suffices for our purpose (the proof is deferred to Appendix G.3):

**Lemma G.2.** *For any constant $d$ and adversary bound $s, m$, there exists a linear and strong PRG $G$ that $\epsilon$-fools all $m$-adaptive circuits over $\mathsf{AC}^0(d, s)$ (i.e., $\mathsf{AC}^0(d, s)^m$ adversaries) with seed length $\lambda = \log^{2d+O(1)}(msn)(m + \log(n/\epsilon))^2$ and output length $n$.*

## G.1 Internal Construction

In this section, we construct an internal-to-external generator $\mathsf{Gen}$ that is secure against adaptive $k$-CNF leakages. Inspired by [IS25], we implement $\mathsf{Gen}$ as $\mathsf{XOR}$ of multiple copies of an (unprotected) linear PRG.

**Construction.** Assume that the leakage class is a collection of $m$-adaptive circuits over $k$-CNFs of size $s$. Given an error parameter $\epsilon$ and a linear PRG $G$ with an input length $\lambda$ and an output length $n$, we then construct $\mathsf{Gen}$ computing $G$ as follows:

- Choose parameter $\eta = \mathsf{poly}(m, 2^k, \log s, \log(1/\epsilon))$ (the detailed choice of $\eta$ will be clarified in Appendix H), randomly sample $\eta$ random seeds $\boldsymbol{u}_j \xleftarrow{\$} \{0,1\}^\lambda$.
- For each $j \in [\eta]$, emulate the PRG $G$ to compute an output string $(\mathsf{out}_{i,j})_{i\in[n]}$ of length $n$ on the input $\boldsymbol{u}_j$.
- Sequentially compute $\mathsf{XOR}$ of the output strings:

$$\boldsymbol{y}_{i,j} = \begin{cases} \mathsf{out}_{i,1}, & \text{if } j = 1 \\ \boldsymbol{y}_{i,j-1} \oplus \mathsf{out}_{i,j}, & \text{if } j \in [2,\eta] \end{cases}.$$

- Denote $\boldsymbol{y}_{i,\eta}$ as $\boldsymbol{r}_i$, the output is $R = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$.

**Correctness.** The correctness of our construction directly follows from the linearity of $G$: the random tape $R$ can be viewed as the output of $G$ on random input $\boldsymbol{u} := \boldsymbol{u}_1 \oplus \ldots \oplus \boldsymbol{u}_\eta$.

**Security.** We claim that our construction is an $\mathcal{L}_1$-internal-to-external generator. Formally, we have

**Lemma G.3.** *For any adversary bounds $m, k, s$, let $\mathcal{L}$ be the family of $k$-CNF queries of size $s$. Given an error parameter $\epsilon$ and a linear PRG $G$ with seed length $\lambda$, for a sufficiently large parameter $\eta = \mathsf{poly}(m, 2^k, \log s, \log(1/\epsilon))$, the above construction is a $m$-adaptive $(\mathcal{L}, \mathcal{L}', \epsilon)$-internal-to-external generator that implements the functionality of $G$, where $\mathcal{L}'$ is the family of $O(\eta/m) = \mathsf{poly}(m, 2^k, \log s, \log(1/\epsilon))$-adaptive circuits over $k$-CNFs of size $2^k s$. The randomness complexity of our scheme is $\eta \cdot \lambda$.*

We defer the proof of Lemma G.3 to Appendix H.

## G.2 External Construction

In this section, we describe the construction of an $(\mathcal{L}, \mathsf{Gen})$-LTC with non-uniform randomness of functionality $f$. As we discussed in Section 8, we divide the large functionality $f$ into individual gates, and use small strong LTC $C_g$ to compute each of them, in which $C_g$ uses at most $M$ (independent of $|f|$) random bits. Here, we require that the output $R$ of the pseudorandom generator $\mathsf{Gen}$ is not only pseudorandom against $\mathcal{L}$-leakages, but also piecewise random: it can be partitioned into $|f|$ pieces (each of length $M$) such that the distribution of each piece is uniformly random.

**Definition G.4.** A randomized circuit $\mathsf{Gen}$ is $(\mathcal{L}, M, \epsilon)$-piecewise random if there exist $M$ independent $(\mathcal{L}, \epsilon)$-resilient PRGs $h_1, \ldots, h_M$ (each of output length $n$) such that:

- The output distribution of $\mathsf{Gen}$ is identical to the product of the output distributions of $h_1, \ldots, h_M$.
- For each pair of indexes $(i, j)$, the $j$-th output bit $h_{i,j}$ of $h_i$ is uniformly distributed in $\{0,1\}$ (as a corollary, the distribution of $(h_{1,j}, \ldots, h_{M,j})$ is identical to the uniform distribution over $\{0,1\}^M$).

**Ingredient.**

- Strong $(\mathcal{L}, \epsilon_{\mathsf{mask}})$-resilient PRG $G_{\mathsf{mask}}$ with seed length $\lambda_{\mathsf{mask}}$ and output length $|f|$, such that for each index $u$, the computation of the $u$-th output bit can be compressed to a circuit $G_{\mathsf{mask},u}$ of size $\mathsf{poly}(r_{\mathsf{mask}}, \log|f|, \kappa)$. We note that any linear PRG satisfies this requirement.
- A sequence of strong $(\mathcal{L}, \mathcal{L}', \epsilon_{\mathsf{LTC}})$-LTCs $(C_u)_{u\in[|f|]}$ for some closed under restriction and negation leakage class $\mathcal{L}'$ such that $C_u$ implements small functionality $f_u$ (described later) and has an efficient simulator $\mathsf{SimLTC}_u$. Denote the randomness complexity of $C_u$ as $\mathsf{RC}(|f_u|, \kappa)$ and the maximum number of random bits used in each $C_u$ as $M$.
- An $(\mathcal{L}, M, \epsilon_{\mathsf{Gen}})$-piecewise random generator $\mathsf{Gen}$ with output length $M|f|$.

**Construction.**

**$\mathcal{L}$-LTC with Non-Uniform Randomness**

- **Randomness Generation Phase.**
  - Label the output of $\mathsf{Gen}$ as the random tape $R = (\boldsymbol{r}_i^j)_{i\in[M],j\in[|f|]}$ where $\boldsymbol{r}_i$ is viewed as the output of the $i$-th $(\mathcal{L}, \epsilon_{\mathsf{Gen}})$-resilient PRG $h_i$.
  - Introduce fresh randomness $\boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^{\lambda_{\mathsf{mask}}}$ as the input of $G_{\mathsf{mask}}$.
- **Input Encoding Phase.** For each input bit $\boldsymbol{x}_i$:
  - Construct a strong LTC $C_{\boldsymbol{x}_i}$ computing the following functionality $f_{\boldsymbol{x}_i}$ with random tape $R_{\boldsymbol{x}_i} = \left(\boldsymbol{r}_1^{\boldsymbol{x}_i}, \ldots, \boldsymbol{r}_M^{\boldsymbol{x}_i}\right)$:

$$f_{\boldsymbol{x}_i}(\boldsymbol{u}, v) = G_{\mathsf{mask},\boldsymbol{x}_i}(\boldsymbol{u}) \oplus v.$$

  - Compute $\boldsymbol{w}_{\boldsymbol{x}_i} := C_{\boldsymbol{x}_i}(\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{x}_i; R_{\boldsymbol{x}_i})$ and we refer to $\boldsymbol{w}_{\boldsymbol{x}_i}$ as the wire label of $\boldsymbol{x}_i$.
- **Circuit Emulation Phase.** For each random wire $\alpha$ with value $\boldsymbol{c}_\alpha$, generate a fresh random bit as $\boldsymbol{w}_\alpha$ which can be viewed as $G_{\mathsf{mask},\alpha}(\boldsymbol{r}_{\mathsf{mask}}) \oplus \boldsymbol{c}_\alpha$.

  Then for each gate $g$ with input wires $\alpha, \beta$, output wire $\gamma$ and wire labels $\boldsymbol{w}_\alpha, \boldsymbol{w}_\beta, \boldsymbol{w}_\gamma$ respectively:
  - Construct a strong LTC $C_g$ computing the following functionality $f_g$ with random tape $R_g = \left(\boldsymbol{r}_1^g, \ldots, \boldsymbol{r}_M^g\right)$:

$$f_g(\boldsymbol{u}, v, w) = G_{\mathsf{mask},g}(\boldsymbol{u}) \oplus g\left(G_{\mathsf{mask},\alpha}(\boldsymbol{u}) \oplus v, G_{\mathsf{mask},\beta}(\boldsymbol{u}) \oplus w\right).$$

  - Denote the output wire label as $\boldsymbol{w}_\gamma := C_g(\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{w}_\alpha, \boldsymbol{w}_\beta; R_g)$.
- **Output Decoding Phase.** For each output bit $\boldsymbol{y}_i$ with wire label $\boldsymbol{w}_{\boldsymbol{y}_i}$:
  - Construct a strong LTC $C_{\boldsymbol{y}_i}$ computing the following functionality $f_{\boldsymbol{y}_i}$ with random tape $R_{\boldsymbol{y}_i} = \left(\boldsymbol{r}_1^{\boldsymbol{y}_i}, \ldots, \boldsymbol{r}_M^{\boldsymbol{y}_i}\right)$:

$$f_{\boldsymbol{y}_i}(\boldsymbol{u}, v) = G_{\mathsf{mask},\boldsymbol{y}_i}(\boldsymbol{u}) \oplus v.$$

  - Output $\boldsymbol{y}_i = C_{\boldsymbol{y}_i}(\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{w}_{\boldsymbol{y}_i}; R_{\boldsymbol{y}_i})$.

**Randomness Complexity.** We first analyze the size of $M$. Because each functionality $f_u$ has a size at most $|G_{\mathsf{mask},u}| + O(1) = \mathsf{poly}(\boldsymbol{r}_{\mathsf{mask}}, \log|f|, \kappa)$, $M$ is bounded by $\mathsf{RC}(|f_u|, \kappa) = \mathsf{RC}(\mathsf{poly}(\boldsymbol{r}_{\mathsf{mask}}, \log|f|, \kappa), \kappa)$.

**Security.** To prove security, we first build a simulator $\mathsf{SimOut}$ that converts $\mathcal{L}$-leakage query of $W[C]$ to leakage query of input $\boldsymbol{x}$ and output $\boldsymbol{y}$. Let $L'' = L \circ L'$, the outer simulator $\mathsf{SimOut}$ works as follows:

- $\mathsf{SimOut}$ randomly samples $\boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^{\lambda_{\mathsf{mask}}}$ as in real world execution.
- For each index $u \in [|f|]$, $\mathsf{SimOut}$ computes a fake wire label $\boldsymbol{w}_u = G_{\mathsf{mask},u}(\boldsymbol{r}_{\mathsf{mask}})$.
- $\mathsf{SimOut}$ simulates the LTC components:
  - For each input bit $\boldsymbol{x}_i$, $\mathsf{SimOut}$ emulates $C_{\boldsymbol{x}_i}$ by invoking the LTC simulator $\mathsf{SimLTC}_{\boldsymbol{x}_i}$, then writes the internal wire values of $C_{\boldsymbol{x}_i}$ and the random tape $R_{\boldsymbol{x}_i}$ as $\mathcal{L}'$ functions of $\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{x}_i$ and $\boldsymbol{w}_{\boldsymbol{x}_i}$.
  - For each gate $g$ with input wires $\alpha, \beta$ and output wire $\gamma$, $\mathsf{SimOut}$ emulates $C_g$ by invoking $\mathsf{SimLTC}_g$, then writes the internal wire values of $C_g$ and the random tape $R_g$ as $\mathcal{L}'$ functions of $\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{w}_\alpha, \boldsymbol{w}_\beta$ and $\boldsymbol{w}_\gamma$.
  - For each output bit $\boldsymbol{y}_i$, $\mathsf{SimOut}$ emulates $C_{\boldsymbol{y}_i}$ by invoking the LTC simulator $\mathsf{SimLTC}_{\boldsymbol{y}_i}$, then writes the internal wire values of $C_{\boldsymbol{y}_i}$ and the random tape $R_{\boldsymbol{y}_i}$ as $\mathcal{L}'$ functions of $\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{w}_{\boldsymbol{y}_i}$ and $\boldsymbol{y}_i$.

  Observe that $\mathsf{SimOut}$ has already generated $\boldsymbol{r}_{\mathsf{mask}}$ and all wire labels $\boldsymbol{w}_u$, $W[C]$ can be realized as an $\mathcal{L}'$ function $L_W$ on inputs $\boldsymbol{x}, \boldsymbol{y}$.
- Given any $\mathcal{L}$-leakage query $L$ of $W[C]$, $\mathsf{SimOut}$ converts it into a $L''$-leakage query $L \circ L_W$ on inputs $\boldsymbol{x}, \boldsymbol{y}$.

  We prove the security of our simulator $\mathsf{SimOut}$ via a hybrid argument:

**Hybrid $\mathsf{Hyb}_0$.** This hybrid is the real world execution, where the random tape $R$ is sampled from the output distribution of $\mathsf{Gen}$, and each LTC component $C_u$ is faithfully computed.

**Hybrid $\mathsf{Hyb}_1$.** In this hybrid, we first assign a random value to each random wire, then compute all wire values of $f$ (denoting as $\{\boldsymbol{c}_u\}_{u\in[|f|]}$). We then isolate each individual LTC: fix each wire label $\boldsymbol{w}_u$ to $G_{\mathsf{mask},u}(\boldsymbol{r}_{\mathsf{mask}}) \oplus \boldsymbol{c}_u$ and each output bit $\boldsymbol{y}_i$ to $\boldsymbol{c}_{\boldsymbol{y}_i}$ regardless of the output of each LTC component. After that, the output $\boldsymbol{y}$ is computed using the ideal functionality $f(\boldsymbol{x})$.

By the definition of a piecewise random generator, the random bits $(\boldsymbol{r}_i^j)_{i\in[M]}$ are uniformly random for each $j \in [|f|]$. By the union bound, the statistical distance of $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_0$ can be bounded by $|f|$ multiplying the security of each underlying LTC. Therefore, $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_0$ are $|f|\epsilon_{\mathsf{LTC}}$-indistinguishable.

**Hybrid $\mathsf{Hyb}_2$.** This hybrid is almost the same as $\mathsf{Hyb}_1$, except that the random tape $R$ becomes uniformly sampled.

Since the output $\boldsymbol{y}$ is independent of $R$, we only need to consider the $\mathcal{L}$-leakage query $L$. By a standard hybrid argument, we may iteratively switch the output of each $h_i$ to a random string. We then have that $\mathsf{Hyb}_2$ and $\mathsf{Hyb}_1$ are $M\epsilon_{\mathsf{Gen}}$-indistinguishable.

**Hybrid $\mathsf{Hyb}_3$.** In this hybrid, we use the strong leakage-tolerance property to simulate each LTC component $C_u$, namely, invoke the simulator $\mathsf{SimLTC}_u$ to compute the wire values as $\mathcal{L}'$ functions of inputs and outputs of $C_u$. The security of this step follows from two facts: (1) the random tapes of all LTCs are independently and uniformly random; (2) and the wire label $\boldsymbol{w}_u$ is fixed to $G_{\mathsf{mask},u} \oplus \boldsymbol{c}_u$, which means that the output of $C_u$ will never be used.

Again, following a standard hybrid argument, $\mathsf{Hyb}_3$ and $\mathsf{Hyb}_2$ are $|f|\epsilon_{\mathsf{LTC}}$-indistinguishable based on the security of the underlying LTCs.

**Hybrid $\mathsf{Hyb}_4$.** This hybrid is almost the same as $\mathsf{Hyb}_3$, except that we replace each $\boldsymbol{w}_u$ by the fake wire label $G_{\mathsf{mask},u}(\boldsymbol{r}_{\mathsf{mask}})$ instead of the real label $G_{\mathsf{mask},u}(\boldsymbol{r}_{\mathsf{mask}}) \oplus \boldsymbol{c}_u$.

Let $\boldsymbol{w} = (\boldsymbol{w}_u)_{u\in[|f|]}$. Observe that in $\mathsf{Hyb}_3$ the wire values $W[C]$ are written as $\mathcal{L}'$ functions $L_W$ of inputs $\boldsymbol{r}_{\mathsf{mask}}$ and $G_{\mathsf{mask}}(\boldsymbol{r}_{\mathsf{mask}}) \oplus \boldsymbol{w}$. Fix $\boldsymbol{w}$, since the output $\boldsymbol{y}$ is independent of the internal randomness $\boldsymbol{r}_{\mathsf{mask}}$, we only need to consider the indistinguishability of the leakage result. Specifically, we have

$$
\begin{aligned}
&\{L \circ L_W(\boldsymbol{r}_{\mathsf{mask}}, G_{\mathsf{mask}}(\boldsymbol{r}_{\mathsf{mask}}) \oplus \boldsymbol{w}) : \boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^\lambda\} \\
\approx_{\epsilon_{\mathsf{mask}}} &\{L \circ L_W(\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{r}^* \oplus \boldsymbol{w}) : \boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^\lambda, \boldsymbol{r}^* \xleftarrow{\$} \{0,1\}^{|f|}\} \\
\equiv &\{L \circ L_W(\boldsymbol{r}_{\mathsf{mask}}, \boldsymbol{r}^*) : \boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^\lambda, \boldsymbol{r}^* \xleftarrow{\$} \{0,1\}^{|f|}\} \\
\approx_{\epsilon_{\mathsf{mask}}} &\{L \circ L_W(\boldsymbol{r}_{\mathsf{mask}}, G_{\mathsf{mask}}(\boldsymbol{r}_{\mathsf{mask}})) : \boldsymbol{r}_{\mathsf{mask}} \xleftarrow{\$} \{0,1\}^\lambda\},
\end{aligned}
$$

where the two "$\approx_{\epsilon_{\mathsf{mask}}}$" lines follow the pseudorandomness of $G_{\mathsf{mask}}$. Hence, $\mathsf{Hyb}_4$ and $\mathsf{Hyb}_3$ are $O(\epsilon_{\mathsf{mask}})$-indistinguishable.

Note that $\mathsf{Hyb}_4$ is the ideal world execution, we then conclude that $\mathsf{Hyb}_0$ and $\mathsf{Hyb}_4$ are $O(\epsilon_{\mathsf{mask}} + |f|\epsilon_{\mathsf{LTC}} + M\epsilon_{\mathsf{Gen}})$ indistinguishable. Therefore, the above construction is an $(\mathcal{L}, \mathcal{L}'', \mathsf{Gen})$-LTC scheme:

**Lemma G.5.** *Assuming the building blocks described above, for any functionality $f$, our construction is an $(\mathcal{L}, \mathcal{L}'', O(\epsilon_{\mathsf{mask}} + |f|\epsilon_{\mathsf{LTC}} + M\epsilon_{\mathsf{Gen}}), \mathsf{Gen})$-LTC where $M = \mathsf{RC}(\mathsf{poly}(\boldsymbol{r}_{\mathsf{mask}}, \log|f|, \kappa), \kappa)$ and $L'' = L \circ L'$. The randomness complexity of the LTC scheme is $\lambda_{\mathsf{mask}}$ plus the randomness complexity of $f$.*

We are interested in the case where the leakage class $\mathcal{L}$ is a collection of adaptive $k$-CNF queries. In this case, we can prove the following result by tuning the parameters:

**Lemma G.6.** *For any functionality $f$ and adversary bound $m, k, s$, let $\epsilon$ be some error parameter, $\mathcal{L}$ be the family of $k$-CNF queries of size $s$, and $\mathsf{Gen}$ be an $(\mathcal{L}, M, \epsilon_{\mathsf{Gen}})$-piecewise random generator where $M = \mathsf{poly}(m, \log|f|, \log s, \log(1/\epsilon))$. Then there exists a $m$-adaptive $(\mathcal{L}, \mathcal{L}, O(M\epsilon_{\mathsf{Gen}} + \epsilon), \mathsf{Gen})$-LTC implementation of $f$, in which the randomness complexity is $\mathsf{poly}(m, 2^k, \log|f|, \log s, \log(1/\epsilon))$ plus the randomness complexity of $f$.*

*Proof.* We separately estimate the error terms in Lemma G.5 (note that $\mathcal{L}$ is a subset of all $\mathsf{AC}^0(2, s)$ circuits):

- We use a strong $(\mathsf{AC}^0(2,s)^m, \epsilon_{\mathsf{mask}})$-resilient PRG to implement $G_{\mathsf{mask}}$. Choosing $\lambda_{\mathsf{mask}} = \log^{O(1)}(ms|f|)(m + \log(|f|/\epsilon))^2 = \mathsf{poly}(m, \log|f|, \log s, \log(1/\epsilon))$, the first error term $\epsilon_{\mathsf{mask}}$ is bounded by $\epsilon$.

- Choosing $n = O(\log^2(ms)(m + \log(|f|/\epsilon)))$, we then know that $(\mathsf{PAR}(n,0), \mathsf{PAR}(n,1))$ are $2^{-m}\epsilon/|f|$-indistinguishable by $\mathsf{AC}^0(2, ms, m) \circ \mathsf{NC}^0[2]$ circuits by Lemma 4.2. Indeed, Theorem 6.4 shows that we can build strong $(\mathsf{AC}^0(2, ms, m), \mathsf{NC}^0[1], O(2^{-m}\epsilon/|f|))$-LTCs $(C_u)_{u\in[|f|]}$ each of size $O(((|G_{\mathsf{mask},u}| + O(1)) + n)n^2) = \mathsf{poly}(m, \log|f|, \log s, \log(1/\epsilon))$ (which also gives an upper bound of $M$). Now applying Lemma F.8, it is clear that the same construction is $m$-adaptive strong $(\mathsf{AC}^0(2, s), \mathsf{NC}^0[1], O(\epsilon/|f|))$-leakage-tolerant. Therefore, the second error term is upper bounded by $O(\epsilon)$.
- We note that Lemma G.5 constructs an $(\mathcal{L}, \mathcal{L}'', \mathsf{Gen})$-LTC scheme, while in our case $\mathcal{L}'' = \mathcal{L} \circ \mathsf{NC}^0[1] = \mathcal{L}$.

□

**Summary.** Right now we already have the internal construction $\mathsf{Gen}$ (Lemma G.3) and the external construction $C$ (Lemma G.6) against adaptive $k$-CNF leakages. Putting them together, we can construct an adaptive $k$-CNF LTC implementation $C'$ of any functionality $f$:

*Proof of Theorem 8.4.* Given any parameter $M$, we derive a linear PRG $G$ that is $(\mathcal{L}_2, M, \epsilon/M)$-piecewise random from concatenating $M$ independent $(\mathcal{L}_2, \epsilon_{\mathsf{Gen}} = \epsilon/M)$-resilient linear PRGs (Lemma G.2), where the seed length of $G$ is $M \cdot \mathsf{poly}(m, \log s, \log|f|, \log(1/\epsilon), \log M)$ and the output length is $M|f|$. Therefore, we can build a $m$-adaptive $(\mathcal{L}_1, \mathcal{L}_2, \epsilon)$-internal-to-external generator $\mathsf{Gen}$ implementing $G$, with randomness complexity $r_1 = \eta M \cdot \mathsf{poly}(m, \log s, \log|f|, \log(1/\epsilon), \log M) = \mathsf{poly}(m, 2^k, \log s, \log|f|, \log(1/\epsilon))$.

Then, for some parameter $M = \mathsf{poly}(\eta, \log|f|, \log(2^k s), \log(1/\epsilon))$ as defined in Lemma G.6, we can construct an $(\mathcal{L}_2, \mathcal{L}_2, O(M\epsilon_{\mathsf{Gen}} + \epsilon) = O(\epsilon), \mathsf{Gen})$-LTC implementation of $f$, with randomness complexity $r_2 = \mathsf{poly}(\eta, \log|f|, \log(2^k s), \log(1/\epsilon)) = \mathsf{poly}(m, 2^k, \log|f|, \log s, \log(1/\epsilon))$ plus the randomness complexity of $f$. Composing them we prove the result. □

### G.3 Deferred Proof of Lemma G.2

Recall that Nisan and Wigderson [NW94] construct a linear PRG against $\mathsf{AC}^0$ adversary, here we follow their approach to show that this construction indeed satisfies strong pseudorandomness.

**Definition G.7.** For a parameter $\lambda$, a collection of sets $\{S_1, \ldots, S_n\}$ such that $S_i \subseteq [\lambda]$ is called a $(k, \ell)$-design if it satisfies:

- For all $i \in [n]$, $|S_i| = \ell$.
- For all $i \neq j$, $|S_i \cap S_j| \leq k$.

**Lemma G.8** ( [NW94])**.** *There exists explicitly and efficiently computable $(k, \ell) = (\log n, O(\sqrt{\lambda}))$-design for all $\ell \geq \log n$.*

**Construction.** Given any $(k, \ell) = (\log n, \Theta(\sqrt{\lambda}))$-design $(S_1, \ldots, S_n)$, the Nisan-Wigderson $\mathsf{AC}^0$-resilient PRG simply outputs $G(\boldsymbol{x}) = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_n)$ such that $\boldsymbol{g}_i = \mathsf{parity}(\boldsymbol{x}_{S_i})$.

**Security.** We first prove that the Nisan-Wigderson PRG is strong pseudorandom against $\mathsf{AC}^0(d, ms, m)$ adversary by emulating the classical proof idea:

**Lemma G.9.** *Assume that $\ell \geq \log n$ and $(\mathsf{PAR}(\ell+1, 0), \mathsf{PAR}(\ell+1, 1))$ are $(\epsilon/n)$-indistinguishable by $\mathsf{AC}^0(d+1, msn^{O(1)}, m)$ circuits. Then the above construction is a strong PRG against $\mathsf{AC}^0(d, ms, m)$ adversary with security loss $\epsilon$.*

*Proof.* We use a hybrid argument over $(\boldsymbol{x}, \boldsymbol{y})$. In $\mathsf{Hyb}_i$ $(0 \le i \le n)$, we set the first $i$ bits of $\boldsymbol{y}$ to $(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_i)$ and the rest of $\boldsymbol{y}$ to uniformly random string. It is clear that $\mathsf{Hyb}_0 \equiv (\boldsymbol{x}, G(\boldsymbol{x}))$ and $\mathsf{Hyb}_n \equiv (\boldsymbol{x}, \boldsymbol{y} \xleftarrow{\$} \{0,1\}^n)$.

We claim that $\mathsf{Hyb}_{i+1}$ and $\mathsf{Hyb}_i$ are $(\epsilon/n)$-indistinguishable. Assume there exists some $\mathsf{AC}^0(d, ms, m)$ adversary $\mathcal{A}$ that breaks the indistinguishability of $\mathsf{Hyb}_{i+1}$ and $\mathsf{Hyb}_i$. Fix $\boldsymbol{x}_{[n]/S_{i+1}}$ and $y_{i+2}, \ldots, y_n$ to maximize the advantage of $\mathcal{A}$, then each $\boldsymbol{g}_j$ $(j \in [i])$ can be written as $\mathsf{parity}(\boldsymbol{x}_{S_j \cap S_{i+1}})$ plus a constant value where $|S_j \cap S_{i+1}| \le k$.

Because every $k$-ary function can be expressed as $k$-DNF or $k$-CNF of size $O(k2^k)$, we can rewrite $\mathcal{A}$ as $\mathcal{A}' \in \mathsf{AC}^0(d+1, ms + O(nk2^k), m) \subseteq \mathsf{AC}^0(d+1, msn^{O(1)}, m)$ on input $(\boldsymbol{x}_{S_{i+1}}, \boldsymbol{g}_{i+1})$. That is, $\mathcal{A}'$ can $(\epsilon/n)$-distinguish between $(\boldsymbol{x}_{S_{i+1}}, \mathsf{parity}(\boldsymbol{x}_{S_{i+1}}))$ and $(\boldsymbol{x}_{S_{i+1}}, \mathsf{parity}(\boldsymbol{x}_{S_{i+1}}) \oplus 1)$. This implies that it can also distinguish between $\mathsf{PAR}(\ell+1, 0)$ and $\mathsf{PAR}(\ell+1, 1)$ since $\boldsymbol{x}_{S_{i+1}}$ is randomly sampled from $\{0,1\}^\ell$, which contradicts our hypothesis. □

Then Lemma G.2 follows from a direct application of Lemma F.8.

# H Deferred Proof of Lemma G.3

**Notation.** We use $X, Y$ to denote the set of wires used to calculate the copies of PRG $G$ and the set of wires in computing the XOR (i.e. $\{\boldsymbol{y}_{i,j}\}$s), respectively. Then for a subset $E$ of wires belonging to $\mathsf{Gen}$, we may use $R_{E \cap Y}$ to denote the set of output random bits related to $E \cap Y$. Formally, $R_{E \cap Y} := \{\boldsymbol{r}_i : \exists j \text{ s.t. } \boldsymbol{y}_{i,j} \in E\}$.

## H.1 Overview of Proof

In this section, we prove the security of the internal-to-external generator $\mathsf{Gen}$ described in Appendix G.1, by implementing a simulator $\mathsf{SimIn}$ that converts the adaptive $k$-CNF leakage queries of the entire circuit $W[C']$ to adaptive $k$-CNF leakage queries of the external circuit $W[C]$.

Note that each leakage query $L$ of $W[C']$ can involve wires in both $W[C]$ and $W[\mathsf{Gen}]$. To simulate the leakage result, our idea is to "reverse-sample" the involved wire values of $W[\mathsf{Gen}]$, conditioned on random tape $R$. However, note that the simulator $\mathsf{SimIn}$ can only access $R$ by making adaptive $k$-CNF leakage queries to the external circuit.

Because the underlying PRG $G$ is linear, the circuit $\mathsf{Gen}$ is also linear in the sense that every internal wire value can be uniquely expressed as a linear combination of $\boldsymbol{u}$. According to this, it is possible to write each internal wire value as a linear combination of the output random tape $R$ (W.L.O.G. assuming that the output is longer than the input). The technical difficulty of this approach is that we cannot actually perform an arbitrary linear combination of $R$ using only $k$-CNF leakage queries of the external circuit $C$. Here we make two important observations:

- First, our construction allows us to efficiently reverse-sample a small subset of $W[\mathsf{Gen}]$ by probing the values of a small subset of $R$ (Lemma H.1).
- Second, when $k$ is sufficiently small (for our purpose $k = O(\log \kappa)$), for any fixed $W[C]$, a $k$-CNF leakage query applied to $W[C'] = W[(\mathsf{Gen}, C)]$ can be adaptively determined by a few input variables in $W[\mathsf{Gen}]$.

## H.2 Efficient Reverse-Sampling

We first focus on the following problem: given the probing oracle access to the output random tape $R$, reverse-samples a small subset $E$ of internal wire values $W[\mathsf{Gen}]$. Although the joint distribution

of $W[\mathsf{Gen}]$ depends on the output random tape $R$, We claim that for a small subset $E \subseteq \mathsf{Gen}$ such that $|E \cap X| < \eta$ (recall that $\eta$ is the number of independent copies of the PRG $G$), we can reverse-sample partial wire values $W[E] \subseteq W[\mathsf{Gen}]$ by just probing a small portion of $R$!

**Lemma H.1.** *Assume that $|E \cap X| < \eta$ for a subset $E \subseteq \mathsf{Gen}$. Then the distribution of $W[E]$ (conditioned on a fixed random tape $R$) is identical to the distribution only conditioned on $R_{E \cap Y}$.*

*Proof.* Let $t \in [\eta]$ be an index such that $E$ does not include any internal wires of the $t$-th copy of PRG $G$, and let $X_{[\eta]/t}$ denote the set of rest wires inside $X$. We may consider the distribution of $W[E'] = W[E \cup X_{[\eta]/t}]$.

Note that the function $\mathsf{Gen}$ can be viewed as applying PRG on the XORs of the seeds of all copies: $\mathsf{Gen}(\boldsymbol{u}) = G(\boldsymbol{u}_1 \oplus \ldots \oplus \boldsymbol{u}_\eta)$ where $\boldsymbol{u}_i$ is the random seed of the $i$-th copy. Therefore, the distribution of $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{t-1}, \boldsymbol{u}_{t+1}, \ldots, \boldsymbol{u}_\eta)$ (which decides $W[X_{[\eta]/t}]$) is independent of the random tape $R$. We finish the proof by observing that after fixing $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{t-1}, \boldsymbol{u}_{t+1}, \boldsymbol{u}_\eta)$, the wire values $W[E \cap Y]$ are fully determined by $R_{E \cap Y}$. □

Lemma H.1 indicates that we can design a sampler $\mathsf{Sgen}$ that supports reverse-sampling $W[E_i]$ conditioned on random tape $R$ on *adaptive* chosen subsets $\{E_i\}$ of $\mathsf{Gen}$ such that $|(\cup_i E_i) \cap X| < \eta$, in which $\mathsf{Sgen}$ only probes $R_{E_i \cap Y}$. A naive implementation of $\mathsf{Sgen}$ may run in exponential time. Note that our circuit $\mathsf{Gen}$ is linear and any constraints of the wire values can be written as linear equations, we can accelerate $\mathsf{Sgen}$ by solving the linear equations. In this way, reverse-sampling can be efficiently accomplished.

### H.3 Simulator Construction.

Now we are ready to present our simulator $\mathsf{SimIn}$. Given a $k$-CNF leakage query $L = \mathsf{CL}_1 \wedge \mathsf{CL}_2 \wedge \ldots \wedge \mathsf{CL}_\ell$ of $W[C']$, where the $i$-th $\mathsf{OR}$-clause $\mathsf{CL}_i$ is of the form:

$$\mathsf{CL}_i = ((\mathsf{vx}_{i,1} \oplus \mathsf{bx}_{i,1}) \vee \ldots \vee (\mathsf{vx}_{i,\mathsf{kx}_i} \oplus \mathsf{bx}_{i,\mathsf{kx}_i})) \bigvee ((\mathsf{vy}_{i,1} \oplus \mathsf{by}_{i,1}) \vee \ldots \vee (\mathsf{vy}_{i,\mathsf{ky}_i} \oplus \mathsf{by}_{i,\mathsf{ky}_i})) \bigvee \mathsf{OCL}_i,$$

such that $\mathsf{vx}_{i,j} \in X$, $\mathsf{vy}_{i,j} \in Y$, $\mathsf{bx}_{i,j}, \mathsf{by}_{i,j} \in \{0,1\}$, and $\mathsf{OCL}_i$ is an $\mathsf{OR}$-clause of variables in $W[C]$ and their negations; our simulator $\mathsf{SimIn}$ maintains a subset of wires $E \subseteq \mathsf{Gen}$ and $R_{E \cap Y}$ known so far, and scans each clause $\mathsf{CL}_i$ to answer $L$ in a simple strategy:

1. Use a binary search to find the first $\mathsf{OR}$-clause $\mathsf{CL}_i$ that is not constantly 1 according to $E$ and $W[C]$. This step incurs $O(\log \ell)$ number of adaptive $k$-CNF leakage queries to $W[C]$. If there exists no such $\mathsf{OR}$-clause, simply output 1 and abort.
2. Invoke $\mathsf{Sgen}$ to adaptively reverse-sample wire values $\{\mathsf{vx}_{i,j}\}_{j \in [\mathsf{kx}_i]}$ and $\{\mathsf{vy}_{i,j}\}_{j \in [\mathsf{ky}_i]}$ according to $E$ and $R$. If $\mathsf{vx}_{i,j} = \mathsf{bx}_{i,j}$ for all $j \in [\mathsf{kx}_i]$ and $\mathsf{vy}_{i,j} = \mathsf{by}_{i,j}$ for all $j \in [\mathsf{ky}_i]$, $\mathsf{CL}_i$ and the leakage query result should be both 0 under this assignment. Therefore, respond 0 and abort. Otherwise, update $E$ and rewind to Step 1.

Let $\epsilon$ be the tolerance error as described in Lemma G.3, we give a formal description of $\mathsf{SimIn}$:

**Simulator** $\mathsf{SimIn}$

- Set $E = \emptyset$ and $R_{E \cap Y}$.
- Receive $L = \mathsf{CL}_1 \wedge \ldots \wedge \mathsf{CL}_\ell$, one of the $m$-adaptive $k$-CNF leakage queries asked by adversary $\mathcal{A}$:
  - Set current index $i = 1$.

– Repeat the following for at most $\min\{2^k \log(m/\epsilon), \ell\}$ rounds:

1. Set $S = \emptyset$.
2. For each $j \in [i, \ell]$, get a $k$-CNF query $q_j \leftarrow \mathsf{Process}(\mathsf{CL}_j, E, R_{E\cap Y})$ of $W[C]$. If $q_j \neq \emptyset$, add $j$ to $S$.
3. Use binary search to find the first index $i^* \in S$ such that $q_{i^*} = 0$. To verify whether there exists some $i^* \leq t$ such that $i^* \in S$ and $q_{i^*} = 1$, it suffices to ask the following $k$-CNF leakage query $L_{S,t}$ of $W[C]$ (since each $q_j$ is just a $k$-CNF, the $\mathsf{AND}$ of them is also a $k$-CNF query):
$$L_{S,t} := \bigwedge_{j\leq t, j\in S} q_j.$$
If there exists no such $i^*$, output 1 and break.
4. Set $i = i^*$.
5. Invoke $\mathsf{Sgen}$ to reverse-sample wire values of $\{\mathsf{vx}_{i,j}\}_{j\in[\mathsf{kx}_i]}, \{\mathsf{vy}_{i,j}\}_{j\in[\mathsf{ky}_i]}$ of the clause $\mathsf{CL}_i$ according to $E$ and $R$ (note that $\mathsf{Sgen}$ may adaptively probe the output random bits related to $\boldsymbol{v}_{i,j}$s). If all the variables satisfy $\mathsf{vx}_{i,j} = \mathsf{bx}_{i,j}, \mathsf{vy}_{i,j} = \mathsf{by}_{i,j}$, output 0 and break.
6. Add the wires $\{\mathsf{vx}_{i,j}\}_{j\in[\mathsf{kx}_i]}, \{\mathsf{vy}_{i,j}\}_{j\in[\mathsf{ky}_i]}$ to $E$, and update $R_{E\cap Y}$ accordingly by probing $R$.

– If the loop never terminates in $2^k \log(m/\epsilon)$ rounds, the simulator fails and outputs the failure symbol $\perp$.

**Subroutine** $\mathsf{Process}(\mathsf{CL}_i, E, R_{E\cap Y})$

- Set $q_i = \emptyset$.
- Denote all the corresponding output random bits of $\mathsf{CL}_i$
$$\{\boldsymbol{r}_t : \boldsymbol{r}_t \notin R_{E\cap Y} \text{ and } \exists j, s \text{ s.t. } \mathsf{vy}_{i,j} = \boldsymbol{y}_{t,s}\}$$
as $R_i$.
- For each possible assignment $\boldsymbol{b}_{R_i} := \{(\boldsymbol{r}_t = \boldsymbol{b}_t) : \boldsymbol{r}_t \in R_i\}$ of $R_i$, verify whether the value of $\mathsf{OR}$-clause $\mathsf{CL}_i$ (except $\mathsf{OCL}_i$) can be 0 according to $\boldsymbol{b}_{R_i}$ and $(E, R_{E\cap Y})$ (denoting as $H_{R_i}(\boldsymbol{b}_{R_i})$):
$$\begin{cases} \mathsf{vx}_{i,j} = \mathsf{bx}_{i,j}, & \forall j \in [\mathsf{kx}_i] \\ \mathsf{vy}_{i,j} = \mathsf{by}_{i,j}, & \forall j \in [\mathsf{ky}_i] \end{cases}.$$
If $H_{R_i}(\boldsymbol{b}_{R_i}) = 1$, add an $\mathsf{OR}$-clause of width $\leq k$ to $q_i$:
$$q_i \leftarrow q_i \bigwedge \left( \mathsf{OCL}_i \bigvee_{\boldsymbol{r}_t \in R_i} (\boldsymbol{r}_t \oplus \boldsymbol{b}_t) \right),$$
means that the value of $\mathsf{OR}$-clause $\mathsf{CL}_i$ might be 0, supposing that $\mathsf{OCL}_i = 0$ and $\boldsymbol{r}_t = \boldsymbol{b}_t$ for all $\boldsymbol{r}_t \in R_i$.
- Return $q_i$.

**Query Complexity.** We first notice that the size of each $q_j \leftarrow \mathsf{Process}(\mathsf{CL}_j, E)$ is at most $2^k|\mathsf{CL}_j|$. Thus, the size of each leakage query $L_{S,t}$ of $W[C]$ is bounded by $2^k s$. Moreover, for each $k$-CNF leakage query $L$, the outer loop executes at most $2^k \log(m/\epsilon)$ rounds, and in each round the simulator $\mathsf{SimIn}$ performs at most $O(k+\log \ell) = O(k+\log s)$ queries to $W[C]$ (up to $O(\log l)$ $k$-CNF queries in Step 3 and up to $O(k)$ probing queries in Step 5,6).

Hence, the simulator $\mathsf{SimIn}^{\mathcal{O}}$ reduces each $k$-CNF leakage query $L$ of $W[C']$ to at most $T = O(2^k \log(m/\epsilon)(k+\log s))$ adaptive leakage queries of $W[C]$, and in total at most $mT = O(m2^k \log(m/\epsilon)(k+\log s)) = \mathsf{poly}(m, 2^k, \log s, \log(1/\epsilon))$ queries for $m$-adaptive $k$-CNF queries of each size $s$. We set $\eta = mT + 1$ to satisfy the requirement of Lemma H.1.

**Security.** Finally, we show that our simulator $\mathsf{SimIn}$ satisfies $m$-adaptive leakage-tolerance. Specifically, we will prove a stronger result: conditioned on any input $\boldsymbol{x}$, fixed random tape $R \parallel R'$ of the external circuit $C$, the distribution of wire value assignment sampled by $\mathsf{SimIn}$ is $\epsilon$-indistinguishable from the real world execution for any $m$-adaptive $k$-CNF query sequence $(L_1, \ldots, L_m)$. To see this, we only need to argue for the following two conclusions:

1. Except for the probability of failure, the output distribution of $\mathsf{SimIn}$ exactly matches $W[C']$ conditioned on $W[C(\boldsymbol{x}; R \parallel R')]$.
2. The probability of failure in answering each query $L_i$ is bounded by $\epsilon/m$. Thus, by a union bound the total failure probability is at most $\epsilon$.

The first conclusion is naturally implied by the security of $\mathsf{Sgen}$. And for the second conclusion, note that we always have $q_{i^*} = 1$, the linearity of $\mathsf{Gen}$ guarantees that in Step 5 $\mathsf{Sgen}$ has at least $2^{-\mathsf{kx}_i - \mathsf{ky}_i} \geq 2^{-k}$ probability to sample wire values such that $\mathsf{vx}_{i,j} = \mathsf{bx}_{i,j}$ and $\mathsf{vy}_{i,j} = \mathsf{by}_{i,j}$ for all variables. Hence, the probability that the loop never terminates until $2^k \log(m/\epsilon)$ rounds is bounded by $(1 - 2^{-k} \log(m/\epsilon))^{2^k} \leq \epsilon/m$.