IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–24. DOI:XXXXXXXX

Towards a Formal Treatment of Logic Locking

Peter Beerel[1], Marios Georgiou[2], Ben Hamlin[2], Alex J. Malozemoff[2] and Pierluigi Nuzzo[1]

[1] University of Southern California, Los Angeles, California
[2] Galois, Inc., Portland, Oregon

Abstract. Logic locking aims to protect the intellectual property of a circuit from a fabricator by modifying the original logic of the circuit into a new "locked" circuit such that an entity without the key should not be able to learn anything about the original circuit. While logic locking provides a promising solution to outsourcing the fabrication of chips, unfortunately, several of the proposed logic locking systems have been broken. The lack of established secure techniques stems in part from the absence of a rigorous treatment toward a notion of security for logic locking, and the disconnection between practice and formalisms. We seek to address this gap by introducing formal definitions to capture the desired security of logic locking schemes. In doing so, we investigate prior definitional efforts in this space, and show that these notions either incorrectly model the desired security goals or fail to capture a natural "compositional" property that would be desirable in a logic locking system. Finally we move to constructions. First, we show that universal circuits satisfy our security notions. Second, we show that, in order to do better than universal circuits, cryptographic assumptions are necessary.

Keywords: Logic Locking · Security Definitions

1 Introduction

Integrated circuits often represent the root of trust of modern computing systems. However, over the years, the design and manufacturing process has been decentralized to include multiple players in the supply chain, and this decentralization has raised the risk of threats such as intellectual property piracy and reverse engineering. For example, a malicious manufacturer may attempt to steal and reproduce a proprietary algorithm, extract secret keys or information hardwired in the design, or overproduce.

Logic locking has gained significant attention as a potential solution to mitigate the above vulnerabilities [RKM10, RZZ+13, YMRS16, SLM+17, YSN+17, XS18, KKN+19, KAHS20, MAS+21]. The goal of logic locking is to modify the logic of the circuit in such a way that the circuit becomes "useless" without the knowledge of an additional secret key that is only known to the designer. That is, a designer could "lock" the circuit and hand the locked circuit to the foundry. Upon receiving the resulting (locked) chip, the designer could then "unlock" the circuit to recover the original circuit's functionality.

1.1 A Journey Towards Formal Security

Unfortunately, formally defining the above property has been elusive. Until recently, the approaches had a cat-and-mouse flavor, where a locking scheme would be claimed secure only to be broken later, followed by fixes that attempt to protect against the breaks, and so on. This should not come as a surprise—after all, this is the existing process when developing practically efficient cryptographic primitives such as block ciphers

Table 1: Summary of several proposed schemes for combinational circuits developed after the introduction of the SAT attack [SRM15, EGT19]. We also list whether the scheme proposes a (or references an existing) security definition, and whether there exists a published attack against the given scheme.

Scheme	Security Definition?	Published Attack?
SFLL [YSN+17]	Flawed (§4.1)	FALL [SS20]
DLL [XS17]	✗	SMT [AKHS19]
Cyclic Obf. [SLM+17]	✗	CycSAT [ZJK17]
SRCLock [RKS18]	✗	SMT [AKHS19]
LUT-Lock [KAG+18]	✗	[KKN+19]
Customized LUTs [KKN+19]	✗	
[DSSY20]	Flawed (§4.3)	
InterLock [KAHS20]	✗	
$\text{OneChaff}_{\text{hd}}$ [CS21]	Yes (§4.4)	
Redaction [MAS+21]	✗	

and hash functions. However, the state of logic locking is different in a significant way: *there exists no precise formalism for what exactly a logic locking scheme should achieve*. This has several undesirable effects. For one, it is hard to reason about the security of a logic locking scheme without a formal notion of what a secure scheme *should* achieve. This has led to a multitude of claimed-secure constructions that have since been broken [BTZ10, DBN+14, RZZ+15, SRZ18, SS20, SRM15, APSS21]. In addition, even if it is not possible to directly use the formalism to prove security (in the sense that one cannot construct a security proof against that formalism), the formalism can still guide the designer in knowing what to focus on in the design. As an example, all commonly used block ciphers—such as AES—do *not* have security proofs. Despite this, understanding precisely *what* a secure block cipher should achieve guides designers in avoiding common pitfalls. This has led to practical block ciphers that have withstood decades of cryptanalysis.

1.2 Existing Schemes and Their Security

We are not the first to identify the need for a formal security definition for logic locking. Indeed, several notions have been proposed in the literature, and it is becoming more common for proposed constructions to claim security against a security model, usually one introduced alongside the proposed construction. We find two concerns with this approach. First, the fact that a security definition exists does *not* mean it is the most appropriate definition, in the sense that it rigorously captures the particular adversary setting logic locking aims to address. While there have been several *attempts* at formalizing logic locking [YSN+17, SPJ19, DSSY20], we show in §4 that these approaches are amenable to important flaws. Second, rather than introducing its own security definition, each construction should target a *well established and agreed upon* security definition. Consider encryption: while there are definitional variants, the standard notion of "secure encryption"—indistinguishability against chosen plaintext attack (IND-CPA)—is well established, and any new encryption scheme must at least achieve this baseline. One of the aims of this work is to move the community towards such a standard notion.

In Table 1, we list several proposed logic locking schemes targeting combinational circuits, whether the construction references a security definition, and whether the scheme has a published attack. Not all schemes in the table have known attacks. We caution the reader that this does *not* mean that a scheme is "secure"—where secure is relative to a specific security definition—in that, while a scheme may avoid *existing* attacks, it may fall victim to future attacks. We can see this in several instances; for example,

SRCLock [RKS18] was designed to prevent SAT attacks but was later found vulnerable to SMT attacks [AKHS19].

1.3 Contributions

In this work, we take a fresh look at formalizing the security of logic locking, focusing on combinational constructions. We begin by defining a concrete syntax for logic locking; namely, what exactly a logic locking construction *is*. We then recast prior definitional attempts using this syntax. In doing so, we identify several shortcomings that may emerge from prior formalization attempts. We propose two new security definitions for logic locking, and discuss the relation between these new notions and prior notions. Finally, we show that a "universal circuit" approach satisfies our security notions, possibly encompassing several programmable logic-based methods [KKN+19, KAHS20, CZHN21, SPJ19], including "redaction" [MAS+21], which can be modeled within a universal circuit framework. We discuss our contributions in more detail below.

A syntax for logic locking (§3). We begin by defining a logic locking scheme as a randomized algorithm Lock that takes as input a circuit $\mathbb{C}$ and outputs a locked circuit $\mathbb{L}$ and a key K. We require that $\mathbb{L}$, when given as input the right key K, has the same "behavior" as the original circuit $\mathbb{C}$. This captures the intuitive notion that for a logic locking scheme to be useful, the designer must be able to "unlock" the fabricated chip to produce the expected behavior of the original design. Requiring Lock to return K, rather than taking it as input as in the definitions of many cryptographic primitives, allows K to depend on $\mathbb{C}$, which is necessary to capture schemes based on universal circuits.

A survey of prior definitions (§4). By fixing the syntax of logic locking, we can recast all prior definitions in this framework to allow for ease of comparison between these notions. In doing so, we identify several flaws that lead to definitions that are either impossible or trivial to satisfy. We note that prior definitions mostly follow a similar flavor: the adversary $\mathcal{A}$, given the locked circuit $\mathbb{L}$ as input and, possibly, oracle access to an unlocked, working version of the original circuit $\mathbb{C}$, needs to output a circuit $\widetilde{\mathbb{C}}$ that is *functionally equivalent* to—that is, has the same input-output behavior as—$\mathbb{C}$. What differs among these definitions is what constitutes a "win" for the adversary: approaches such as attack resilience [YSN+17] and function recovery [CS21] require that the probability that $\mathcal{A}$ can output such a $\widetilde{\mathbb{C}}$ is sufficiently small, whereas approaches such as best possible approximate functional secrecy [SPJ19] and lock security [DSSY20] require that $\mathcal{A}$ do no better than a "simulator" $\mathcal{S}$ that has *only* oracle access to the input/output behavior of $\mathbb{C}$ and knowledge of its size (that is, $\mathcal{S}$ does *not* take the locked circuit $\mathbb{L}$ as input).

Unfortunately, all these prior definitions—except function recovery—exhibit important flaws: they become either impossible or trivial to satisfy. This is primarily due to the specification of the definition differing from the perceived "intent" of the definition. We thus recast these notions as a notion that we call *functional secrecy*, which aims to capture the intent of these prior notions while avoiding their pitfalls. However, as we show in §6, even this notion—alongside function recovery—still fails to guarantee a natural and intuitive composition requirement that one would want a logic locking scheme to satisfy.

New definitions for logic locking (§5). Given the state of prior definitions, we propose two new definitions for logic locking. The difference between these and prior efforts lies in the observation that it is not sufficient to guarantee that an adversary cannot recover the entire protected circuit. Any part of the protected circuit—or indeed, any predicate about it—that the adversary can learn is potentially damaging. Thus, to use our definitions, one first rigorously identifies the information a locking scheme inevitably "leaks," then

proves that it reveals nothing else. This opens the door to proving the security of efficient schemes that hide "enough" information about a circuit, while requiring designers to be explicit about exactly what is hidden and what is not.

Our first notion, which we call *indistinguishable logic locking* (IND-LL), requires that the adversary cannot "learn" which of two circuits was locked. In detail, we consider two arbitrary circuits $\mathbb{C}_0$ and $\mathbb{C}_1$ and lock one of them at random. The adversary "wins" if it can guess which circuit was locked. Leakage is handled implicitly by requiring that $\mathbb{C}_0$ and $\mathbb{C}_1$ have the leaked information in common. This approach is very similar to other "indistinguishability"-based notions in cryptography, adapted to the logic locking setting.

Our second notion, which we call *simulation-secure logic locking* (SIM-LL), requires that an adversary cannot "learn" any more about the original circuit from the locked circuit than a simulator can learn, given only access to some leakage information on the circuit. This is similar to our functional secrecy notion, except we again are only requiring the adversary to output some arbitrary predicate, versus requiring the adversary to output a functionally equivalent circuit as in the functional secrecy case.

Finally, we note that in practice the entire design does not necessarily need to be protected from an adversarial foundry. Rather, there is likely some small subset of the design that contains the sensitive IP, with the rest containing non-sensitive IP.[1] As we show in §6, all prior security definitions we investigated were insufficient to capture security in this setting. In contrast, although the definitions of IND-LL and SIM-LL do not explicitly allow part of the circuit to remain unlocked, they *do* guarantee security in this scenario. We show this by means of a notion we call *contextual functional secrecy* (CFS), which IND-LL and SIM-LL both imply, and prior notions do not. CFS captures the goal that a scheme should remain secure if it is used on a circuit $\mathbb{C}$ composed with any public context $(\mathbb{C}_I, \mathbb{C}_O)$, as $\mathbb{C}_O \circ \mathbb{C} \circ \mathbb{C}_I$.

Relations between definitions (§6). Given this set of definitions, we next investigate the *relationship* between these definitions. We show that IND-LL is strictly stronger than SIM-LL and that SIM-LL is strictly stronger than all prior notions we investigated.

Constructions (§7). Finally, we show that our new notions are *satisfiable* by showing that a "universal circuit" approach (akin to certain programmable logic-based methods [KKN+19, KAHS20, CZHN21, SPJ19, MAS+21]) satisfies IND-LL. While this approach is not new (for example, redaction [MAS+21] takes a similar approach using FPGAs and the work of Di Crescenzo et al. [DSSY20] provide a more theoretical analysis), we demonstrate that this approach—when applied to Boolean circuits—is IND-LL secure (and hence SIM-LL secure as well) when leaking the size of the circuit to the adversary. We caution that this does *not* imply that the redaction approach is necessarily secure: for example, our circuit model is different than the circuit model of FPGAs. However, it is an initial step towards reasoning about the security of a redaction-like scheme, and provides a basis for future work proving the security of more realistic circuit models.

Universal circuits come with a lower bound on their size that stems from a lower bound on the size of edge-universal graphs.[2] We show in §7.2 that cryptographic assumptions are necessary to do better than this bound when targeting the IND-LL notion. Interestingly, no scheme we are aware of (cf. Table 1) makes use of cryptographic assumptions, suggesting that these schemes are not IND-LL secure.

[1] This is the same observation that guides Mohan et al.'s redaction approach to logic locking [MAS+21].
[2] Roughly, edge-universal graphs can embed any directed acyclic graph of a particular size.

2 Preliminaries

We use upper-case letters to denote strings (e.g., X) and bold-face letters to denote circuits (e.g., $\mathbb{C}$). Our definitions are largely irrespective of the internal computational model of the circuit, so for now we leave it unspecified, besides the requirement that circuits be *stateless*. Only in §7, where we prove the security of a specific construction, do we specify the circuit computational model. For circuits $\mathbb{C}$ and $\mathbb{C}'$, we say $\mathbb{C} \equiv \mathbb{C}'$ if the two circuits have identical input-output behavior (that is, their *functionalities* are the same). For some fixed $\epsilon \in [0,1]$, we say $\mathbb{C} \equiv_\epsilon \mathbb{C}'$ if the two circuits have identical input-output behavior except on an ϵ-fraction of inputs.

At times, we want to make explicit additional information that can be hardwired in a circuit $\mathbb{C}$ but never used by it. For example, we would like to have a way of adding a secret key K to the *description* of $\mathbb{C}$ that can be extracted easily by just observing $\mathbb{C}$'s description. We denote such an *augmented* circuit as $\mathbb{C}' = [\mathbb{C}, K]$. In particular, this implies that $[\mathbb{C}, K] \equiv \mathbb{C}$. Likewise, at times we want to reference a bit in the description of a circuit $\mathbb{C}$. To denote the ith bit of the description of $\mathbb{C}$, we use the notation $\langle \mathbb{C} \rangle_i$.

We use $X : Y$ to denote that variable X has "type" Y; e.g., $\mathbb{C} : \{0,1\}^n \to \{0,1\}^m$ says that $\mathbb{C}$ is an n-input m-output circuit. We use $x \twoheadleftarrow S$ to denote that x is sampled uniformly at random from set S and $x \twoheadleftarrow \mathcal{D}$ to denote that x is sampled from a distribution $\mathcal{D}$.

When we use the term "adversary," we mean a probabilistic algorithm with access to zero or more oracles. We assume an oracle call takes unit time. We use the notation $\mathcal{A}^{\mathcal{O}}$ to denote an adversary $\mathcal{A}$ with oracle access to $\mathcal{O}$. We often consider oracles that implement the functionality of a circuit $\mathbb{C}$. In this case, we use $\mathcal{A}^{\mathbb{C}}$ to denote that adversary $\mathcal{A}$ has oracle access to circuit $\mathbb{C}$. We use $\Pr[\mathcal{A} \Rightarrow 1]$ to denote the probability that $\mathcal{A}$ outputs the value 1. Occasionally, we need to provide the adversary with explicit randomness. We use the notation $\mathcal{A}(X; R)$ to denote that adversary $\mathcal{A}$ is run on input X using randomness R.

Throughout the paper, we use ϵ as a measure of functional equivalence between two circuits—i.e., the fraction of inputs they differ on—and use σ, τ, and δ for (difference of) probabilities, t for time, and q for number of queries.

Concrete security. We utilize the *concrete security* framework [BKR94], where in lieu of a security parameter and an asymptotic adversary bound, we quantify everything in terms of the adversary's resources. Concrete security allows one to clearly quantify the adversary's advantage, whereas this information often gets lost when only considering asymptotic bounds. Thus, for security definitions aimed at practical deployments, we believe a concrete security treatment is the correct choice.

3 Logic Locking Syntax

In this section, we define what a logic locking scheme *is* and what it means for a logic locking scheme to be *correct*.

Definition 1 (Logic locking)**.** A logic locking scheme is a randomized algorithm Lock with the following syntax:

> $(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C})$ takes as input a circuit $\mathbb{C} : \{0,1\}^n \to \{0,1\}^m$ and outputs a circuit $\mathbb{L} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ and a key $K : \{0,1\}^k$. ♢

Definition 2 (Correctness)**.** Logic locking scheme Lock is *correct* if for any circuit $\mathbb{C} : \{0,1\}^n \to \{0,1\}^m$, any $(\mathbb{L}, K)$ in the support of $\mathsf{Lock}(\mathbb{C})$, and any input $X : \{0,1\}^n$, it holds that $\mathbb{L}(K, X) = \mathbb{C}(X)$. ♢

Leakage. When trying to define what it means for a logic locking scheme to be "secure" we often need to consider what the adversary knows ahead of time about the underlying circuit. We thus define a leakage function $\mathcal{L}$ that takes a circuit as input and outputs any information about that circuit that we allow to be leaked to the adversary. For example, one could consider a leakage function $\mathcal{L}_{\mathsf{size}}(\mathbb{C})$ that leaks the size of the input circuit $\mathbb{C}$. Another leakage function could be $\mathcal{L}_{\mathsf{topo}}(\mathbb{C})$ that leaks the circuit topology of $\mathbb{C}$. We only consider leakage functions that are *invertible* in the sense that given leakage ℓ one can efficiently derive a circuit $\mathbb{C}$ that achieves that leakage.

4 Previous Definitions for Logic Locking

In this section we discuss several previous (and one largely concurrent) attempts towards defining security for logic locking. For each definition, we present our best interpretation of the definition (not all definitions presented in prior works are fully specified, so we occasionally need to interpret the *intent* of the definition as best as we can), followed by a discussion of the implications of said definition.

4.1 Attack Resilience [YSN+17]

Yasin et al. [YSN+17] provided one of the first attempts at formalizing security for logic locking,[3] by introducing three notions: *SAT attack resilience*, *sensitization attack resilience*, and *removal attack resilience*. We focus on the first notion, as it most closely aligns with the adversary setting considered in this work.[4] We begin by presenting a generalization of SAT attack resilience we call simply "attack resilience."

Definition 3 (Attack resilience)**.** Logic locking scheme Lock is (t, λ)-$\mathsf{attack\ resilient}$ if, for any t-time adversary $\mathcal{A}$ and for any circuit $\mathbb{C}$,

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C} \mid \begin{array}{c}(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L})\end{array}\right] \leq q(\lambda)/2^{\lambda},$$

where $q(\cdot)$ is some polynomial denoting the number of queries made by $\mathcal{A}$ to its oracle. $\diamondsuit$

Let us break this definition down. The adversary's goal is to derive some circuit $\widetilde{\mathbb{C}}$ that is functionally equivalent to the original circuit $\mathbb{C}$, and must do so given only the locked circuit $\mathbb{L}$ and the ability to query the circuit oracle $q(\lambda)$ times. It wins if the circuit it produces is equivalent with probability greater than $q(\lambda)/2^{\lambda}$.

The actual definition of Yasin et al. considers a restricted form of attack resilience they call *SAT attack resilience*. The definition is the same as Definition 3 except with respect to a class of adversaries that *only* run SAT attacks. If we denote an adversary in this class as $\mathcal{A}_{\mathbb{SAT}}$, then SAT attack resilience can be viewed as attack resilience with respect to adversaries $\mathcal{A}_{\mathbb{SAT}}$.

Discussion. Unfortunately, SAT attack resilience is ill-defined in the sense that what constitutes a "SAT attack adversary" $\mathcal{A}_{\mathbb{SAT}}$ is not formally defined. For example, one could attempt to define a SAT attack adversary as a probabilistic polynomial time (PPT) algorithm with access to a SAT oracle. However, this adversary is actually *stronger* than the standard cryptographic adversary (who is only assumed to be PPT with no external

[3] The authors in addition proposed a novel construction, SFLL. We do not discuss this construction in this work as our focus is on the formalism.

[4] Sensitization attack resilience models an attacker that attempts to learn the underlying key while removal attack resilience models the removal of the locking mechanism without altering the functionality. Both of these intuitively try to capture a setting *weaker* than SAT attack resilience, although formally proving this is difficult due to the imprecision discussed in this section.

oracle). One could instead try to limit the adversary to exist in some *weaker* complexity class than PPT; however, it is unclear what the real-world implications of this would be, and it could unnecessarily rule out practical attacks. Indeed, a scheme claimed as SAT attack resilient [YSN+17] has since been broken [SS20].

However, even if we consider the more general attack resilience notion, the definition is unfortunately impossible to satisfy. Consider the adversary $\mathcal{A}$ that ignores its input and outputs some fixed circuit $\widetilde{\mathbb{C}}$. Note that there exists *some* circuit $\mathbb{C}$ for which $\widetilde{\mathbb{C}} \equiv \mathbb{C}$, hence $\mathcal{A}$ succeeds with probability one, independent of the implementation of Lock. Intuitively, the security definition is missing some form of "randomization" to make such trivial attacks impossible. This randomization should appear after an adversary is fixed, thus generating uncertainty for the adversary. As an example, consider security that is defined as a game between an adversary and a challenger, where the challenger samples a random value (for example a bit) and the goal of the adversary is to guess this bit. In this case, the adversary cannot hardcode the right bit, because the bit is chosen after the adversary is fixed.

4.2 Best Possible Approximate Functional Secrecy [SPJ19]

Subsequent to Yasin et al.'s work, Shamsi et al. [SPJ19] define several notions of security, culminating in *best possible approximate functional secrecy* (BPAFS), which we focus on here. BPAFS aims to improve upon attack resilience by comparing the success probability of the adversary $\mathcal{A}$ attacking the locked circuit with some other adversary $\mathcal{B}$ that has only query access and knowledge of the original circuit's size.

In particular, Shamsi et al. present a notion that allows the adversary to restore an *approximation* of the original circuit $\mathbb{C}$. This is captured by defining a "win" for the adversary as it producing a circuit $\widetilde{\mathbb{C}}$ that is equivalent to $\mathbb{C}$ *except on up to* ϵ input-output pairs, where ϵ is a parameter of the security notion. In addition, rather than requiring that this probability be "sufficiently low" (as in attack resilience), the authors instead require that the adversary should do no better than another adversary that only has (1) oracle access to the original circuit, and (2) some fixed "leakage" of the original circuit—in this case, the size of the circuit.

Definition 4 (Best-possible approximate functional secrecy)**.** Logic locking scheme Lock is (t, q, ϵ, δ)-BPAFS-secure if for any circuit $\mathbb{C}$, for any t-time q-query adversary $\mathcal{A}$, and for any $O(t)$-time q-query adversary $\mathcal{B}$, the following probability holds:

$$\left| \Pr\left[\tilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \begin{array}{c} (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L}) \end{array} \right] - \Pr\left[\tilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \tilde{\mathbb{C}} \leftarrow \mathcal{B}^{\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C})) \right] \right| \leq \delta. \qquad \diamondsuit$$

Discussion. Unfortunately, like attack resilience, BPAFS as presented is impossible to satisfy for any $\delta < 1$. Consider adversary $\mathcal{A}$ that outputs hardcoded circuit $\mathbb{C}'$ where $\mathbb{C}'$ differs in more that ϵ input-output pairs from the "zero" circuit (that is, a circuit that on any input outputs the zero string), and adversary $\mathcal{B}$ that outputs the "zero" circuit. For any $\mathbb{C} \equiv \mathbb{C}'$, it holds that $\delta = 1$, independently of the implementation of Lock. The issue largely comes down to the fact that we are comparing two adversaries and quantifying over all possible adversaries. Clearly, for any adversary $\mathcal{A}$, there will always be *some* adversary $\mathcal{B}$ that differs "sufficiently" from $\mathcal{A}$ to make δ sufficiently large.

4.3 Lock Security [DSSY20]

Di Crescenzo et al. [DSSY20] introduced a notion similar to BPAFS which they call *lock security*. Lock security is similar to BPAFS; however, instead of considering *any* adversary $\mathcal{B}$, lock security requires the *existence* of such an adversary.

Definition 5 (Lock security)**.** Logic locking scheme Lock is (t, q, δ)-$\mathsf{lock\ secure}$ if, for any circuit $\mathbb{C}$ and any t-time q-query adversary $\mathcal{A}$, there exists $\mathsf{poly}(t)$-time $\mathsf{poly}(q)$-query adversary $\mathcal{B}$ such that

$$\left|\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C} \mid \begin{array}{c}(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L})\end{array}\right] - \Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C} \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{B}^{\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C}))\right]\right| \leq \delta. \qquad \diamondsuit$$

Discussion. Unfortunately, this notion is trivially satisfiable. Let Lock be the scheme that sets $\mathbb{L}$ to its input circuit $\mathbb{C}$. Clearly, this trivial Lock scheme should *not* be secure (it reveals $\mathbb{C}$ in the clear). However, we show that Lock is "secure" according to the above definition. Consider an adversary $\mathcal{B}$ that has $\mathbb{C}$ and $\mathcal{A}$ hardcoded (which is possible due to the order of quantifiers in the definition). Adversary $\mathcal{B}$ simply runs $(\mathbb{L}', K') \leftarrow \mathsf{Lock}(\mathbb{C})$ followed by $\mathcal{A}$ on input $\mathbb{L}'$, answering any oracle calls from $\mathcal{A}$ by running $\mathbb{C}$. Note that $\mathbb{L} = \mathbb{L}' = \mathbb{C}$, and thus the input to $\mathcal{A}$ in both probabilities is identical. Thus, $\delta = 0$.

In this case, the flaw is with respect to the quantifiers in the definition: because $\mathcal{B}$ depends on the circuit $\mathbb{C}$, it *knows* $\mathbb{C}$. Requiring that $\mathcal{B}$ work *for all* $\mathbb{C}$ addresses the above attack; we discuss this modified notion in §4.5.

4.4 Function Recovery [CS21]

Most recently (and largely concurrently), Chhotaray and Shrimpton [CS21] introduced the notion of *design hiding* in an effort to formalize the security requirements of logic locking. They define two security notions: function recovery and key recovery; we focus on the function recovery setting in this work, since hiding the key does not necessarily hide the function. Their motivation for function recovery is very similar to ours in that they view prior definitions as either under-specified or insufficient, and provide a rigorous security notion alongside a construction satisfying that notion. While the authors consider both *malicious* and *honest-but-curious* chip fabricators, in this work we focus solely on the honest-but-curious case and thus only consider their honest-but-curious notion.

Definition 6 (Function recovery for design hiding)**.** Logic locking scheme Lock is (t, q, δ)-$\mathsf{FR\text{-}secure}$ for function class $\mathcal{F}$ if for any t-time q-query adversary $\mathcal{A}$,

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \begin{array}{c}F \twoheadleftarrow \mathcal{F} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_F) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}_F}(\mathbb{L})\end{array}\right] \leq \delta. \qquad \diamondsuit$$

Discussion. FR is similar to attack resilience: the main difference is that there is an additional "function class" parameter $\mathcal{F}$, which captures any prior knowledge the adversary may have about the circuit to be fabricated. This addresses the main shortcoming of attack resilience, as now the adversary cannot hard-code a circuit $\widetilde{\mathbb{C}}$ and win with probability one.

However, we note two potential downsides to the FR notion. The notion requires the adversary output a circuit functionally equivalent to the original circuit; this is a strong requirement, and one that may not matter in practice—for example, recovering part of the circuit may be sufficient for the adversary. While this partial-recovery setting can be captured by carefully specifying the function class, one would have to design (and prove) constructions secure for each function class of interest, which may not be scalable. Lastly, we show in §6 that FR is *insufficient* for capturing a natural "composition" notion of logic locking within a larger unprotected circuit.

4.5 Functional Secrecy

As we note above, both BPAFS and lock security have technical issues that cause them to be either unsatisfiable or trivially true for all locking schemes. Nonetheless, they have in

common an *intuitive* notion of security that could be summarized as follows:

> No efficient adversary $\mathcal{A}$, given a locking $\mathbb{L}$ of circuit $\mathbb{C}$ and an oracle for $\mathbb{C}$ (and perhaps some leakage $\mathcal{L}(\mathbb{C})$) should be able to reconstruct an approximation $\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C}$ with significantly greater probability than an adversary who has access to an oracle for $\mathbb{C}$ (and perhaps some leakage $\mathcal{L}(\mathbb{C})$) alone.

Although the intuition behind this notion is clear, the devil is in the details when it comes to security definitions. We propose the following definition, which we call *functional secrecy* (FS), which we believe captures the spirit of both BPAFS and lock security while avoiding their technical shortcomings.

Definition 7 (Functional secrecy)**.** Logic locking scheme Lock is $(t, q, \epsilon, \sigma, \tau, \mathcal{L})$-FS-secure if for any t-time q-query adversary $\mathcal{A}$, there exists a $\mathsf{poly}(t, q)$-time $\mathsf{poly}(q)$-query simulator $\mathcal{S}$ such that for any circuit $\mathbb{C}$ and auxiliary information aux,

$$\Pr\left[\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \begin{array}{c}(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}(\mathbb{C}), \mathsf{aux})\end{array}\right] \geq \sigma \implies \Pr\left[\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathbb{C}}(\mathcal{L}(\mathbb{C}), \mathsf{aux})\right] \geq \tau.$$

♢

Contrary to BPAFS and lock security, in our definition we make both the adversary's and the simulator's winning probabilities explicit. We do this because we believe that the definition remains sensible even if this difference is large. For example, consider the asymptotic setting where an adversary's advantage at guessing the circuit is bounded below by $1/p(\lambda)$ for some polynomial p whereas the simulator's advantage is bounded below by $1/q(\lambda)$ for some different polynomial q, where λ corresponds to the security parameter. The difference between p and q might be large—for example, $q(\lambda) = p^2(\lambda)$—nonetheless, the scheme would still be secure.

Relationship with BPAFS. This notion bears a strong similarity to BPAFS, but it avoids the pitfall that prevents BPAFS from being satisfiable. Since the simulator appears after the adversary in the quantifiers ("for any $\mathcal{A}$, there exists $\mathcal{S}$, ..."), it can be assumed to know the code of the adversary. This prevents an adversary from winning by simply hard-coding a random circuit.

In addition, we generalize over the leakage a locking scheme may have. While we share the intuition of Shamsi et al. [SPJ19] that no locking scheme can successfully hide an upper bound on the size and depth of the circuit $\mathbb{C}$, practical—and in particular, *efficient*—locking schemes may leak slightly more information and still be deemed acceptable, provided the leaked information can be precisely quantified. Or we may consider scenarios where a certain amount of leakage is unavoidable, such as when the adversary knows the circuit class to which $\mathbb{C}$ belongs. A generalized leakage parameter captures these scenarios.

Relationship with lock security. In addition to generalizing lock security by adding a leakage function and allowing adversaries that merely approximate the input circuit, the above definition avoids the problem of trivially admitting a simulator for any Lock. In particular, the simulator must hold for all circuits $\mathbb{C}$, so cannot simply hard-code $\mathbb{C}$.

Discussion. The above notion is appealing, since like attack resilience and function recovery, it directly captures the adversary's inability to reconstruct the locked circuit. However, we argue that *it is insufficient to capture the goal of security against an untrusted foundry.* In particular, it only captures one of many possible security goals. Other reasonable security goals may be preventing an adversary from learning a subcircuit of $\mathbb{C}$ or even learning a circuit that *behaves* like $\mathbb{C}$ in a given fixed context. We formalize the second of these goals and show that FS fails in this setting; see §5.3.

5 New Security Definitions for Logic Locking

We introduce two new security definitions for logic locking. These differ from all prior attempts in that they move beyond functional equivalence of circuits to allowing the adversary to win if it learns *anything* about the circuit that it should not. We call these two notions *indistinguishable logic locking* (§5.1) and *simulation-secure logic locking* (§5.2). We also introduce a notion that aims to capture the setting where only a *component* of a larger design needs to be protected—we call this notion *contextual function security* (§5.3). Finally, we discuss our particular definitional choices in more detail in §5.4.

5.1 Indistinguishable Logic Locking

Our first notion, called *indistinguishable logic locking* (IND-LL), differs from prior logic locking definitions in that it more closely matches classic indistinguishability notions such as indistinguishability against chosen-plaintext attack (IND-CPA).

Definition 8 (Indistinguishable logic locking)**.** Logic locking scheme Lock is $(t, \delta, \mathcal{L})$-IND-LL-secure if for any adversary $\mathcal{A}$ running in time t and circuits $\mathbb{C}_0$ and $\mathbb{C}_1$ such that $\mathcal{L}(\mathbb{C}_0) = \mathcal{L}(\mathbb{C}_1)$,

$$\Pr\left[\widetilde{b} = b \mid \begin{array}{c} b \twoheadleftarrow \{0,1\} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b) \\ \widetilde{b} \leftarrow \mathcal{A}(\mathbb{L}) \end{array}\right] \leq \frac{1}{2} + \delta. \qquad \diamondsuit$$

IND-LL security can be interpreted as follows. We define the following security game between an adversary and a challenger. The challenger picks a random bit b and locks $\mathbb{C}_b$ as the output of $(\mathbb{L}_b, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b)$ and gives $\mathbb{L}_b$ to the adversary. The goal of the adversary is to guess the bit b sampled by the challenger. We say that Lock is $(t, \delta, \mathcal{L})$-IND-LL-secure if, for any adversary running in time t, the probability that it guesses the bit b is δ-far from ½.

Note that, unlike all prior definitions, the IND-LL adversary is *not* given oracle access to a circuit. This is because the adversary *knows* the circuits; thus, it could trivially win when given oracle access by simply querying its oracle on some input X in which $\mathbb{C}_0(X) \neq \mathbb{C}_1(X)$. While this may suggest that IND-LL is *weaker* than notions providing oracle access to the circuit, we show in §6 the opposite: IND-LL is *stronger* than all other existing notions.

5.2 Simulation-Secure Logic Locking

Our second notion, called *simulation-secure logic locking* (SIM-LL), follows the simulation paradigm of functional secrecy (cf. Definition 7). However, there are two fundamental differences: (1) we require a black-box simulator with oracle access to the adversary (versus a white-box simulator as in Definition 7), and (2) rather than the requirement that a "close to functionally equivalent" circuit be produced, we instead allow the adversary to output *any predicate*. Namely, the adversary wins if it could learn *anything more* than a simulator given only oracle access to the circuit alongside any relevant circuit leakage.

Definition 9 (Simulation-secure logic locking)**.** Logic locking scheme Lock is $(t, q, \delta, \mathcal{L})$-SIM-LL-secure if there exists a $\mathsf{poly}(t)$-time $\mathsf{poly}(t, q)$-query simulator $\mathcal{S}$ such that for all t-time q-query adversaries $\mathcal{A}$ and for all circuits $\mathbb{C}$,

$$\left|\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}(\mathbb{C})) \Rightarrow 1 \mid (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C})\right] - \Pr\left[\mathcal{S}^{\mathcal{A}, \mathbb{C}}(\mathcal{L}(\mathbb{C})) \Rightarrow 1\right]\right| \leq \delta. \qquad \diamondsuit$$

Intuitively, SIM-LL states that whatever an adversary $\mathcal{A}$ can learn given the locked circuit $\mathbb{L}$ and oracle access to $\mathbb{C}$, can also be learned by a simulator $\mathcal{S}$ *only* using oracle

access to $\mathbb{C}$. In other words, $\mathbb{L}$ does not leak any more information than what can be leaked by simply querying $\mathbb{C}$.

Note that, contrary to FS and CFS, in SIM-LL we are only interested in the difference between the two probabilities. A definition where the difference is large is not interesting. For example, if we allowed the difference to be ½, then the simulator can just flip a coin.

5.3 Contextual Function Security

In practice, we seldom use a particular component in isolation. Rather, it is used in a context where its inputs are generated and its outputs processed by different components, either on or off chip. This "context-driven" protection is used, for example, by the redaction approach of Mohan et al. [MAS+21], where the chip designer may choose which parts of a chip are sensitive and should be locked, leaving the rest unlocked to lower the overhead of locking. We formalize this idea below.

Definition 10 (Contextual function secrecy)**.** Consider a circuit $\mathbb{C}^* : \{0,1\}^{n_I} \to \{0,1\}^{n_O}$ that can be decomposed into three sequentially composed subcircuits $\mathbb{C}_I : \{0,1\}^{n_I} \to \{0,1\}^n$, $\mathbb{C} : \{0,1\}^n \to \{0,1\}^m$, $\mathbb{C}_O : \{0,1\}^m \to \{0,1\}^{n_O}$ such that $\mathbb{C}^*(X) = \mathbb{C}_O(\mathbb{C}(\mathbb{C}_I(X)))$. Let Lock be a logic-locking scheme and $\mathcal{L}$ a leakage function. We say that Lock is $(t, q, \sigma, \tau, \mathcal{L})$-CFS-secure if there exists a simulator $\mathcal{S}$ such that, for any t-time q-query adversary and for any $\mathbb{C}^* = \mathbb{C}_O \circ \mathbb{C} \circ \mathbb{C}_I$,

$$\Pr\left[\mathbb{C}^* \equiv \mathbb{C}_O \circ \widetilde{\mathbb{C}} \circ \mathbb{C}_I \mid \begin{array}{c}(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)\end{array}\right] \geq \sigma$$
$$\implies \Pr\left[\mathbb{C}^* \equiv \mathbb{C}_O \circ \widetilde{\mathbb{C}} \circ \mathbb{C}_I \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathcal{A},\mathbb{C}}(\mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)\right] \geq \tau,$$

where $\mathcal{S}$ runs in time $\mathsf{poly}(t)$ and makes $\mathsf{poly}(t, q)$ queries to $\mathbb{C}$. ♢

This notion focuses on an adversary who does not need to recover the entire functionality of $\mathbb{C}^*$, but just enough of it to reproduce its behavior in some context $(\mathbb{C}_I, \mathbb{C}_O)$. We stress that this notion is not meant to be used on its own to show the security of a scheme. It reflects just one specific property that a secure scheme should have. Rather, to prove that a scheme is CFS-secure, one should instead prove that it is secure in the sense of IND-LL or SIM-LL, since as we show in Theorem 6, these imply CFS.

5.4 Discussion

We now provide justification for our definitional choices, particularly in light of criticism of logic locking as a whole which argues that it can "never be secure" [EHP19]. In particular, the cited work challenges two assumptions common in logic locking: (1) that the adversary is *passive*—that is, the adversary acts as an honest fab while still aiming to learn something about the input circuit—and (2) that the adversary only has input-output access to the fabricated chip. We address each in turn.

Our definitions—and all prior definitions that we are aware of, except the concurrent work of Chhotaray and Shrimpton [CS21]—considers passive adversaries. That is, we do not consider adversaries that may fabricate invalid circuits or otherwise tamper with the fabrication process. While building schemes secure against this stronger adversary is obviously preferred, we believe that targeting the weaker passive setting is an important first step. Indeed, this is a common approach taken in cryptography; designing schemes secure against weaker adversaries can both help inform designs secure against stronger adversaries and also isolate the components in the design vulnerable to such stronger adversaries.

Likewise, our work models the fabricated chip as a *black-box oracle* that provides access to only the input-output behavior of the original circuit. Engels et al. [EHP19] argue that

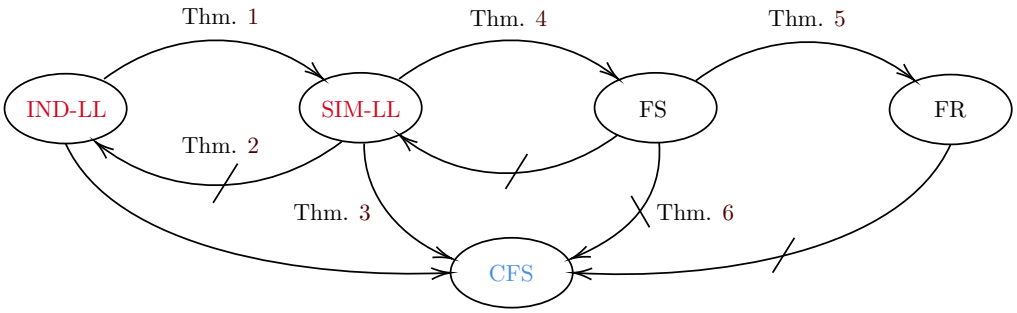

Figure 1: Relations between security definitions from §4 (black) and §5.1 and §5.2 (red). We show that the latter are strictly and meaningfully stronger than the former via the intermediate notion from §5.3 (blue). An arrow between A and B means that any construction that satisfies A also satisfies B. A strikethrough arrow between A and B means that there exists a scheme that satisfies A but not B. Arrows without references are simple corollaries of other theorems.

this is unrealistic, and in practice an adversary can launch arbitrary side channel attacks against the fabricated chip to learn more than just its black-box behavior. In particular, the authors point to an attack that allows the adversary to probe registers to directly learn the key in an unlocked chip.

We note that, when constructing security definitions, some scope needs to be placed on the adversary to allow for designs to focus their efforts on the core underlying problem. The goal of protecting a chip from side channel attacks seems to us orthogonal to the goal of logic locking—and indeed, these approaches could be layered. This approach also models the definitional approach in well-established areas of cryptography such as encryption. That is, standard encryption definitions do not model side channel attacks. Of course, side channel attacks certainly *do* exist against encryption schemes. However, modeling these attacks separately allows designers to focus their efforts by first focusing on designing secure encryption schemes, and *then* modifying those schemes to be secure against side channel attacks. We take the same approach with logic locking.

6 Relations Between Logic Locking Notions

In this section, we prove relations between four notions: IND-LL, SIM-LL, FR, and FS; see Figure 1 for a pictographic depiction of these relations. In §6.1 we show a separation between IND-LL and SIM-LL: IND-LL schemes are SIM-LL-secure but not the other way around. In §6.2 we show that SIM-LL implies CFS, and thus any scheme shown to be IND-LL or SIM-LL also satisfies CFS. In §6.3 we show that SIM-LL implies FS and FR but not the other way around.

6.1 IND-LL Is Stronger Than SIM-LL

In this section, we show that IND-LL implies SIM-LL (Theorem 1), but not the other way around (Theorem 2). Interestingly, this is in contrast to common cryptographic intuition that simulation-based definitions are stronger than indistinguishability-based ones.

Theorem 1 (IND-LL $\implies$ SIM-LL)**.** *For any leakage $\mathcal{L}$, if Lock is $(t, \delta, \mathcal{L})$-IND-LL-secure then Lock is $(t, q, \delta, \mathcal{L})$-SIM-LL-secure for any q.*

Proof. Fix leakage $\mathcal{L}$, and assume Lock is $(t, \delta, \mathcal{L})$-IND-LL-secure for some t and δ. Consider a t'-time q-query SIM-LL adversary $\mathcal{A}$. We define a simulator $\mathcal{S}^{\mathcal{A},\mathbb{C}}(\ell = \mathcal{L}(\mathbb{C}))$ as follows:

1. Pick circuit $\mathbb{C}'$ such that $\mathcal{L}(\mathbb{C}') = \ell$.
2. Compute $(\mathbb{L}', K') \leftarrow \mathsf{Lock}(\mathbb{C}')$.
3. Run $\mathcal{A}$ on input $\mathbb{L}'$, outputting whatever $\mathcal{A}$ outputs. For each oracle query X made by $\mathcal{A}$, query oracle $\mathbb{C}$ on X and return the result to $\mathcal{A}$.

By construction, it holds that for any circuit $\mathbb{C}$ and circuit $\mathbb{C}'$ as chosen by $\mathcal{S}$,

$$\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}', \ell) \Rightarrow 1 \mid (\mathbb{L}', \_) \leftarrow \mathsf{Lock}(\mathbb{C}')\right] = \Pr\left[\mathcal{S}^{\mathcal{A},\mathbb{C}}(\ell) \Rightarrow 1\right].$$

We show below that

$$\begin{aligned}\Big|\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \ell) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C})\right] + \\ - \Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}', \ell) \Rightarrow 1 \mid (\mathbb{L}', \_) \leftarrow \mathsf{Lock}(\mathbb{C}')\right]\Big| \leq \delta,\end{aligned} \tag{1}$$

and hence

$$\left|\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \ell) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C})\right] - \Pr\left[\mathcal{S}^{\mathcal{A},\mathbb{C}}(\ell) \Rightarrow 1\right]\right| \leq \delta.$$

To prove Equation 1, we show that any scheme that is IND-LL is also IND-LL with respect to an oracle to one of its circuits. Assume not; namely, that there exists a t-time adversary $\mathcal{B}$ and circuits $\mathbb{C}_0$, $\mathbb{C}_1$ such that

$$\Pr\left[\widetilde{b} = b \mid \begin{array}{c} b \twoheadleftarrow \{0,1\} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b) \\ \widetilde{b} \leftarrow \mathcal{B}^{\mathbb{C}_0}(\mathbb{L}) \end{array}\right] > \frac{1}{2} + \delta.$$

We can use $\mathcal{B}$ to create an adversary $\mathcal{B}'$ against IND-LL. $\mathcal{B}'(\mathbb{L})$ runs $\mathcal{B}$ on input $\mathbb{L}$ and for every query X from $\mathcal{B}$, $\mathcal{B}'$ returns $\mathbb{C}_0(X)$ to $\mathcal{B}$. Thus,

$$\Pr\left[\widetilde{b} = b \mid \begin{array}{c} b \twoheadleftarrow \{0,1\} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b) \\ \widetilde{b} \leftarrow \mathcal{B}'(\mathbb{L}) \end{array}\right] > \frac{1}{2} + \delta,$$

completing the contradiction. □

The following theorem shows that there exists a SIM-LL-secure logic locking scheme for leakage $\mathcal{L}_{\mathsf{size}}$ that is *not* IND-LL-secure. The proof can be easily generalized to other (non-contrived) leakages, such as $\mathcal{L}_{\mathsf{topo}}$.

Theorem 2 (SIM-LL $\not\Rightarrow$ IND-LL)**.** *Fix q. For any $\delta < 1$, if there exists $(t, q+1, \delta, \mathcal{L}_{\mathsf{size}})$-SIM-LL-secure scheme Lock, then there exists $(O(t), q, \delta, \mathcal{L}_{\mathsf{size}})$-SIM-LL-secure scheme Lock′ that is not $(O(t), \delta, \mathcal{L}_{\mathsf{size}})$-IND-LL-secure.*

Proof. Let Lock be a scheme that is $(t, q+1, \delta, \mathcal{L}_{\mathsf{size}})$-SIM-LL-secure. We define a new scheme Lock′ as follows:

1. On input $\mathbb{C}$, run $(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C})$.
2. Let $Y = \mathbb{C}(\vec{0})$. Output $(\mathbb{L}', (0, K))$ where

$$\mathbb{L}'((b, \hat{K}), X) = \begin{cases} Y & \text{if } b = 1 \\ \mathbb{L}(\hat{K}, X) & \text{otherwise.} \end{cases}$$

Without loss of generality, we assume that $\mathbb{L}$ can be recovered from the netlist of $\mathbb{L}'$ and that Y can be found by inspecting the netlist of $\mathbb{L}'$. To emphasize this, we sometimes write $\mathbb{L}'$ as $[\mathbb{L}, Y]$.

Claim. *Lock′ is not* $(O(t), \delta, \mathcal{L}_{size})$*-IND-LL-secure for any* $\delta < 1$. Indeed, if we let $\mathbb{C}_0$ and $\mathbb{C}_1$ be any two circuits subject to $\mathcal{L}_{\mathsf{size}}(\mathbb{C}_0) = \mathcal{L}_{\mathsf{size}}(\mathbb{C}_1)$ and $\mathbb{C}_0(\vec{0}) \neq \mathbb{C}_1(\vec{0})$, an adversary on input $\mathbb{L}' = [\mathbb{L}, Y]$ can simply output zero if $Y = \mathbb{C}_0(\vec{0})$ and one otherwise, and thus can distinguish with probability one.

Claim. *Lock′ is* $(O(t), q+1, \delta, \mathcal{L}_{size})$*-SIM-LL-secure.* To see this, consider a t-time q-query adversary $\mathcal{A}'$ against Lock'. We construct a simulator $\mathcal{S}'$ that satisfies the required bounds. We begin by constructing an adversary $\mathcal{A}$ for Lock as follows:

1. On input $(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C}))$, $\mathcal{A}$ constructs locked circuit $\mathbb{L}' = [\mathbb{L}, \mathbb{C}(\vec{0})]$, where $\mathbb{C}(\vec{0})$ is computed using $\mathcal{A}$'s oracle access to $\mathbb{C}$.
2. $\mathcal{A}$ runs $\mathcal{A}'$ on input $(\mathbb{L}', \mathcal{L}_{\mathsf{size}}(\mathbb{C}))$, outputting whatever $\mathcal{A}'$ outputs, and answering $\mathcal{A}'$'s queries with its own oracle.

By construction of $\mathcal{A}$, we have that

$$\begin{aligned}&\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C})\right] \\ &\quad = \Pr\left[\mathcal{A}'^{\mathbb{C}}(\mathbb{L}', \mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1 \mid (\mathbb{L}', \_) \leftarrow \mathsf{Lock}'(\mathbb{C})\right],\end{aligned}$$

where $\mathcal{A}$ makes $q+1$ queries to its oracle. Since Lock is $(t, q+1, \delta, \mathcal{L}_{\mathsf{size}})$-SIM-LL-secure, there exists simulator $\mathcal{S}$ such that

$$\left|\Pr\left[\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C})\right] - \Pr\left[\mathcal{S}^{\mathcal{A},\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1\right]\right| \leq \delta.$$

Let $\mathcal{S}'^{\mathcal{A}',\mathbb{C}}$ be a simulator that runs $\mathcal{S}$, forwarding queries to $\mathbb{C}$ directly and answering queries to $\mathcal{A}$ by using $\mathcal{A}'$ as in the construction described above. Thus,

$$\Pr\left[\mathcal{S}'^{\mathcal{A}',\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1\right] = \Pr\left[\mathcal{S}^{\mathcal{A},\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1\right].$$

By combining the equations above, we have that

$$\left|\Pr\left[\mathcal{A}'^{\mathbb{C}}(\mathbb{L}', \mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1 \mid (\mathbb{L}', \_) \leftarrow \mathsf{Lock}'(\mathbb{C})\right] - \Pr\left[\mathcal{S}'^{\mathcal{A}',\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C})) \Rightarrow 1\right]\right| \leq \delta. \quad \square$$

6.2 SIM-LL Implies CFS

The following theorem says that any scheme that is SIM-LL-secure is also CFS-secure. Combined with Theorem 1, this says that any IND-LL-secure scheme is also CFS-secure.

Theorem 3 (SIM-LL $\implies$ CFS)**.** *Let* $\mathcal{L}$ *be any leakage function, and suppose* Lock *is* $(t, q, \delta, \mathcal{L})$*-SIM-LL-secure. Then, for any positive* σ*,* Lock *is* $(t, q, \sigma, \sigma \cdot (1-\delta)^t, \mathcal{L})$*-CFS-secure.*

Proof. Informally, for any CFS adversary we want to define a class of adversaries that output a single bit. This is because for these adversaries, due to SIM-LL, there exist corresponding simulators. Finally, by composing all these simulators into one, we obtain a simulator for the CFS adversary. More precisely, by using the CFS adversary $\mathcal{A}_{\mathsf{CFS}}$, we obtain a circuit $\widetilde{\mathbb{C}}$, which reproduces the functionality of $\mathbb{C}$ in the circuit context $(\mathbb{C}_I, \mathbb{C}_O)$. By running $\mathcal{A}_{\mathsf{CFS}}$ and outputting $\langle\widetilde{\mathbb{C}}\rangle_i$, we can construct an adversary $\mathcal{A}_{\mathsf{SIM\text{-}LL},i}$ to achieve SIM-LL. Since Lock is SIM-LL-secure, there must be a simulator $\mathcal{S}_{\mathsf{SIM\text{-}LL},i}$ that can output $\langle\widetilde{\mathbb{C}}\rangle_i$ without access to the locking $\mathbb{L}$. Finally, by running this simulator $t \geq |\widetilde{\mathbb{C}}|$ times, we can recover $\widetilde{\mathbb{C}}$. This gives us a simulator $\mathcal{S}_{\mathsf{CFS}}$ for CFS.

There is one problem with the proof sketched above: $\mathcal{A}_{\mathsf{CFS}}$ may be a randomized algorithm. So, each time we run it to obtain $\langle\widetilde{\mathbb{C}}\rangle_i$, we may get a different $\widetilde{\mathbb{C}}$. The solution is to derandomize $\mathcal{A}_{\mathsf{CFS}}$. We do this by sampling some explicit randomness ahead of time and giving it to $\mathcal{A}_{\mathsf{CFS}}$ each time we run it.

In more detail, let $\mathbb{C}$ be any circuit and $\mathbb{C}_I$ and $\mathbb{C}_O$ be any context circuits with appropriate in- and out-degree. Suppose there is a t-time q-query CFS adversary $\mathcal{A}_{\mathsf{CFS}}$ with

$$\Pr\left[\mathbb{C}_O \circ \mathbb{C} \circ \mathbb{C}_I \equiv \mathbb{C}_O \circ \widetilde{\mathbb{C}} \circ \mathbb{C}_I \mid \begin{array}{c}(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}_{\mathsf{CFS}}(\mathbb{L}, \mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)\end{array}\right] = \sigma.$$

We construct an adversary for $\mathsf{SIM\text{-}LL}$ by first sampling randomness R, and then defining $\mathcal{A}_{\mathsf{SIM\text{-}LL},\mathbb{C}_I,\mathbb{C}_0,R,i}(\mathbb{L}, \mathcal{L}(\mathbb{C}))$ as: Run $\widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}_{\mathsf{CFS}}(\mathbb{L}, \mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O; R)$ and output $\langle\widetilde{\mathbb{C}}\rangle_i$. Going forward, we simplify notation and use $\mathcal{A}_i$ to mean $\mathcal{A}_{\mathsf{SIM\text{-}LL},\mathbb{C}_I,\mathbb{C}_0,R,i}$.

Now $\mathcal{A}_i$ is an adversary for $\mathsf{SIM\text{-}LL}$ that outputs a predicate about $\mathbb{C}$ with probability σ, and since Lock is $(t, q, \delta, \mathcal{L})$-$\mathsf{SIM\text{-}LL}$-secure, there is a simulator $\mathcal{S}_i$ with

$$\left|\Pr\left[b = \langle\widetilde{\mathbb{C}}\rangle_i \mid \begin{array}{c}(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ b \leftarrow \mathcal{A}^{\mathbb{C}}_i(\mathbb{L}, \mathcal{L}(\mathbb{C}), (\mathbb{C}_I, \mathbb{C}_O))\end{array}\right] - \Pr\left[b = \langle\widetilde{\mathbb{C}}\rangle_i \mid b \leftarrow \mathcal{S}^{\mathbb{C}}_i(\mathcal{L}(\mathbb{C}), (\mathbb{C}_I, \mathbb{C}_O))\right]\right| \leq \delta.$$

Note that $|\widetilde{\mathbb{C}}| \leq t$. By repeating $\mathcal{S}_i$, we can construct a CFS simulator. Define $\mathcal{S}^{\mathbb{C}}_{\mathsf{CFS}}(\mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)$ by the following procedure:

> Sample R uniformly at random.
>
> For $i \in \{1, \ldots, t\}$, run $\langle\widetilde{\mathbb{C}}\rangle_i \leftarrow \mathcal{S}^{\mathbb{C}}_i(\mathcal{L}(\mathbb{C}), (\mathbb{C}_I, \mathbb{C}_O))$.
>
> Output $\widetilde{\mathbb{C}}$.

$\mathcal{S}_{\mathsf{CFS}}$ runs in $t \cdot \mathsf{poly}(t) = \mathsf{poly}(t)$ time and makes $t \cdot q = \mathsf{poly}(t, q)$ oracle queries. Since each simulator is run independently, we get

$$\Pr\left[\mathbb{C}_O \circ \mathbb{C} \circ \mathbb{C}_I \equiv \mathbb{C}_O \circ \widetilde{\mathbb{C}} \circ \mathbb{C}_I \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathbb{C}}_{\mathsf{CFS}}(\mathbb{L}, \mathcal{L}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)\right] \geq \sigma \cdot (1-\delta)^t. \qquad \square$$

We briefly discuss the $\sigma \cdot (1-\delta)^t$ bound. Consider a $\mathsf{SIM\text{-}LL}$-secure scheme with sufficiently small δ. In this case, $(1-\delta)^t$ is close to one, and hence $\sigma \cdot (1-\delta)^t \approx \sigma$. Thus, while this theorem does present some security loss in the reduction, for "reasonable" $\mathsf{SIM\text{-}LL}$-secure schemes it should be small.

6.3 SIM-LL Is Stronger Than FS and FR

In this section we show that $\mathsf{SIM\text{-}LL}$ implies FS and FR but not the other way around. We start by showing the implication from $\mathsf{SIM\text{-}LL}$ to FS.

Theorem 4 ($\mathsf{SIM\text{-}LL} \implies \mathsf{FS}$)**.** *Let $\mathcal{L}$ be any leakage function, and suppose Lock is $(t, q, \delta, \mathcal{L})$-SIM-LL-secure. Then for any positive σ, Lock is $(t, q, 0, \sigma, \sigma(1-\delta)^t, \mathcal{L})$-FS-secure.*

Proof sketch. The proof is very similar to the proof of Theorem 3 with the difference being that now we do not need to consider any context circuits. $\square$

Next we show that FS implies FR. We begin by defining the notion of an "unlearnable" family of circuits [KV94, BGI+01]. This is one that cannot be exactly learned simply through oracle queries to the circuit, and captures the notion that certain circuits cannot be fully learned solely by querying an oracle to the circuit. For example, a pseudorandom function family or the family of point functions[5] are such unlearnable circuit families.

[5]We emphasize that approximately learning a point function is easy: one can just output the constant zero function. Point functions are unlearnable in the exact sense.

Definition 11. A function class $\mathcal{F}$ is $(t, q, \delta, \mathcal{L})$-unlearnable if for all t-time q-query adversaries $\mathcal{A}$, it holds that

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \begin{array}{c} F \leftarrow \mathcal{F} \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}_F}(\mathcal{L}(\mathbb{C}_F)) \end{array}\right] \leq \delta.$$

If $\mathcal{L} = \perp$ (the "empty" leakage), we omit it and say that $\mathcal{F}$ is (t, q, δ)-unlearnable. ♢

Observe that the above definition is only achievable as long as $\delta > \frac{1}{|\mathcal{F}|}$ since one can simply guess the correct circuit with probability $\frac{1}{|\mathcal{F}|}$.

Using this notion, we now proceed to our proof that FS implies FR. As the FR notion is with respect to a function class, our proof is with respect to a fixed class of function classes; in particular, those that share a common leakage and are not "sufficiently" learnable.

Theorem 5 (FS $\implies$ FR)**.** *Fix $\mathcal{L}$, ℓ, function class $\mathcal{F}_\ell = \{F \mid \mathcal{L}(\mathbb{C}_F) = \ell\}$, and $\delta \leq 1$. Suppose Lock is $(t, q, 0, \sigma', \sigma' - \delta, \mathcal{L})$-FS-secure for any $\sigma' \leq 1$. Then, for any $\sigma \leq 1$, if $\mathcal{F}_\ell$ is a $(\mathsf{poly}(t), \mathsf{poly}(q), \sigma - \delta)$-unlearnable function class, then Lock is (t, q, σ)-FR-secure for $\mathcal{F}_\ell$.*

Proof. At a high level, we would like to prove that if a scheme is FS-secure for an unlearnable function class $\mathcal{F}_\ell$, then it is also FR-secure for $\mathcal{F}_\ell$. We approach this as follows. For the sake of contradiction, suppose the scheme is FS-secure but not FR-secure and therefore there exists an adversary $\mathcal{A}_{\mathsf{FR}}$ that can output a circuit that is functionally equivalent to the original with "good" probability. Since the scheme is FS-secure, we know that for this $\mathcal{A}_{\mathsf{FR}}$, there is a corresponding simulator $\mathcal{S}$ that can also output a circuit that is functionally equivalent to the original with "good" probability. However, this simulator is also a successful learner, hence violating the unlearnability property of $\mathcal{F}_\ell$.

Formally, fix $\mathcal{F}_\ell$ as defined in the theorem statement and consider adversary $\mathcal{A}_{\mathsf{FR}}$ such that

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \begin{array}{c} F \twoheadleftarrow \mathcal{F}_\ell \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_F) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}_{\mathsf{FR}}^{\mathbb{C}_F}(\mathbb{L}) \end{array}\right] = \sum_{F \in \mathcal{F}_\ell} \frac{1}{|\mathcal{F}_\ell|} \sigma_F \geq \sigma,$$

where

$$\sigma_F = \Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \begin{array}{c} (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_F) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}_{\mathsf{FR}}^{\mathbb{C}_F}(\mathbb{L}) \end{array}\right].$$

Construct FS adversary $\mathcal{A}_{\mathsf{FS}}^{\mathbb{C}}$ as follows: on input $(\mathbb{L}, \ell)$, run $\mathcal{A}_{\mathsf{FR}}$ on input $\mathbb{L}$, answering any oracle queries with oracle queries to $\mathbb{C}$. Thus, for any $F \in \mathcal{F}_\ell$, the adversary $\mathcal{A}_{\mathsf{FS}}$ succeeds with probability σ_F. Now, because Lock is FS secure, we know that there exists a $\mathsf{poly}(t)$-time $\mathsf{poly}(q)$-query simulator $\mathcal{S}$ such that for any $F \in \mathcal{F}_\ell$,

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathbb{C}_F}(\mathcal{L}(\mathbb{C}_F))\right] \geq \sigma_F - \delta.$$

Therefore, we get that

$$\Pr\left[\widetilde{\mathbb{C}} \equiv \mathbb{C}_F \mid \begin{array}{c} F \twoheadleftarrow \mathcal{F}_\ell \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathbb{C}_F}(\mathcal{L}(\mathbb{C}_F)) \end{array}\right] \geq \sum_{F \in \mathcal{F}_\ell} \frac{1}{|\mathcal{F}_\ell|}(\sigma_F - \delta) \geq \sigma - \delta.$$

This implies that $\mathcal{F}_\ell$ is $(\mathsf{poly}(t), \mathsf{poly}(q), \sigma - \delta)$-learnable, reaching a contradiction. □

Finally, we show that FS does *not* imply the CFS notion of contextual security. Transitively, this means that FR also does not imply CFS.

Theorem 6 (FS $\not\implies$ CFS)**.** *Suppose a $(t, q, \sigma, \tau, \mathcal{L}_{\mathsf{size}})$-FS-secure scheme Lock exists, and for some fixed n, a $(t', q', \iota, \mathcal{L}_{\mathsf{size}})$-unlearnable class of $\frac{n}{2}$-bit to $\frac{n}{2}$-bit functions $\mathcal{F}$ exists. Then there is a $(t, q, \sigma, \tau, \mathcal{L}_{\mathsf{size}})$-FS-secure scheme that is not $(t', q', \sigma', \tau', \mathcal{L}_{\mathsf{size}})$-CFS-secure for any σ' and any $\tau' > \iota$.*

Proof. Intuitively, FS ensures that, given the locking $\mathbb{L}$ of a circuit $\mathbb{C}$, no efficient adversary can recover the *entire* circuit $\mathbb{C}$. In contrast, CFS reflects a scenario where the adversary does not need to recover the entire circuit because, for example, it is used in a context that ignores some of the inputs or outputs. Our proof proceeds by exhibiting a locking scheme that leaks a fraction of the circuit (in some cases) while protecting the rest. This construction is FS-secure, since the leaked portion of the circuit contains no information about the unleaked portion (by construction). However, it is not CFS-secure, since there is a context in which the leaked fraction is all that is needed.

As a preliminary, we call a circuit $\mathbb{C} : \{0,1\}^n \to \{0,1\}^m$ λ-separable for $\lambda \in [0,1]$ if it can be separated into two circuits that run "in parallel", the first of which takes a λ fraction of the input bits and returns a λ fraction of the output bits of $\mathbb{C}$. In other words, $\mathbb{C}(X_1, \ldots, X_n) = \mathbb{C}_1(X_1, \ldots, X_{\lfloor \lambda n \rfloor}) \,\|\, \mathbb{C}_2(X_{\lfloor \lambda n \rfloor + 1}, \ldots, X_n)$, for some $\mathbb{C}_1 : \{0,1\}^{\lfloor \lambda n \rfloor} \to \{0,1\}^{\lfloor \lambda m \rfloor}$ and $\mathbb{C}_2 : \{0,1\}^{n - \lfloor \lambda n \rfloor} \to \{0,1\}^{m - \lfloor \lambda m \rfloor}$, where $\|$ denotes bitwise concatenation. In this case, we write $\mathbb{C} = \mathbb{C}_1 \otimes_\lambda \mathbb{C}_2$, or if λ is unimportant or clear from context, simply $\mathbb{C}_1 \otimes \mathbb{C}_2$.

Assume there exists a $(t, q, \sigma, \tau, \mathcal{L}_{\mathsf{size}})$-FS-secure scheme Lock. We construct scheme Lock′ as follows:

- If $\mathbb{C}$ is ½-separable, proceed as follows. Let $\mathbb{C}_1$ and $\mathbb{C}_2$ be the circuits such that $\mathbb{C} \equiv \mathbb{C}_1 \otimes_{\frac{1}{2}} \mathbb{C}_2$.
  1. Run $(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_2)$.
  2. Let $\mathbb{L}' = \mathbb{C}_1 \otimes \mathbb{L}$.
  3. Output $(\mathbb{L}', K)$.
- Otherwise, run $\mathsf{Lock}(\mathbb{C})$ and output the result.

Claim. *Lock′ is not $(t', q', \sigma', \tau', \mathcal{L}_{\mathsf{size}})$-CFS secure for any efficient t' and q', any σ', and any $\tau' > \iota$.* Let $\mathbb{C} : \{0,1\}^n \to \{0,1\}^n$ be a ½-separable function, where $\mathbb{C} = \mathbb{C}_1 \otimes_{\frac{1}{2}} \mathbb{I}_{\frac{n}{2}}$, $\mathbb{C}_1 \in \mathcal{F}$, and $\mathbb{I}_{\frac{n}{2}}$ is the $n/2$-bit identity circuit. Let $(\mathbb{L}', K) \leftarrow \mathsf{Lock}'(\mathbb{C})$. In addition, let $\mathbb{C}_I = \mathbb{I}_n$ and $\mathbb{C}_O = \mathbb{I}_{\frac{n}{2}} \otimes \mathbb{Z}_{\frac{n}{2}}$, where $\mathbb{Z}_k(X) = 0^k$ for all $X \in \{0,1\}^k$.

We first show that there is a CFS adversary for Lock′ who succeeds with probability one. Consider the adversary $\mathcal{A}^{\mathbb{C}}_{\mathsf{CFS}}(\mathbb{L}', \mathcal{L}_{\mathsf{size}}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)$ who simply parses $\mathbb{L}'$ as $(\mathbb{C}_1, \mathbb{L})$ and returns $\widetilde{\mathbb{C}} = \mathbb{C}_1 \otimes \mathbb{Z}_{\frac{n}{2}}$. Then we have that for all X, $\mathbb{C}_O(\widetilde{\mathbb{C}}(\mathbb{C}_I(X))) = \mathbb{C}_O(\mathbb{C}(\mathbb{C}_I(X)))$. Thus, $\mathcal{A}_{\mathsf{CFS}}$ runs in constant time, makes no queries, and succeeds with probability $\sigma' = 1$.

Next, we show that any efficient simulator for the same CFS game succeeds with probability at most ι. Suppose there exists a t'-time q'-query CFS simulator $\mathcal{S}_{\mathsf{CFS}}$ such that

$$\Pr\left[\mathbb{C}_O \circ \widetilde{\mathbb{C}}' \circ \mathbb{C}_I \equiv \mathbb{C}_O \circ \mathbb{C} \circ \mathbb{C}_I \mid \widetilde{\mathbb{C}}' \leftarrow \mathcal{S}^{\mathbb{C}, \mathcal{A}_{\mathsf{CFS}}}_{\mathsf{CFS}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)\right] = \tau'.$$

We use this to construct an adversary $\mathcal{B}^{\mathbb{C}_1}(\mathcal{L}_{\mathsf{size}}(\mathbb{C}_1))$ for unlearnability on $\mathbb{C}_1$ as follows: $\mathcal{B}$ runs $\mathcal{S}^{\mathbb{C}, \mathcal{A}_{\mathsf{CFS}}}_{\mathsf{CFS}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C}), \mathbb{C}_I, \mathbb{C}_O)$, with $\mathbb{C} = \mathbb{C}_1 \otimes \mathbb{I}_{\frac{n}{2}}$ and $\mathcal{L}_{\mathsf{size}}(\mathbb{C}) = \mathcal{L}_{\mathsf{size}}(\mathbb{C}_1) + \mathcal{L}_{\mathsf{size}}(\mathbb{I}_{\frac{n}{2}})$ to get $\widetilde{\mathbb{C}}'$, then outputs the restriction $\widetilde{\mathbb{C}}_1$ of $\widetilde{\mathbb{C}}'$ to the first $\frac{n}{2}$ of its input and output bits. By construction, $\mathcal{B}$ takes $\mathsf{poly}(t')$ time and makes q' queries, and $\widetilde{\mathbb{C}}_1 \equiv \mathbb{C}_1$ if $\mathcal{S}_{\mathsf{CFS}}$ succeeds, which occurs with probability τ'. But since $\mathcal{F}$ is unlearnable, $\mathcal{B}$ succeeds with probability at most ι, so $\tau' \leq \iota$.

Claim. *Lock′ is $(t, q, \epsilon, \sigma, \tau)$-FS secure.* Let $\mathcal{A}'$ be a t-time q-query adversary for Lock′ such that

$$\Pr\left[\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \begin{array}{c}(\mathbb{L}, \_) \leftarrow \mathsf{Lock}'(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}'^{\mathbb{C}}(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C}))\end{array}\right] \geq \sigma.$$

We construct a $\mathsf{poly}(t)$-time q-query adversary $\mathcal{A}$ for Lock. Let $\mathbb{C}$ be any circuit with n inputs and m outputs. Define $\mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C}))$ as

1. Let $\mathbb{L}' = \mathbb{Z} \otimes \mathbb{L}$, where $\mathbb{Z}$ is the n-input, m-output circuit that ignores its inputs and outputs all zeroes.
2. Let $\ell = \mathcal{L}_{\mathsf{size}}(\mathbb{Z}) + \mathcal{L}_{\mathsf{size}}(\mathbb{C})$.
3. Run $\widetilde{\mathbb{C}}' \leftarrow \mathcal{A}'^{\mathbb{Z}\otimes\mathbb{C}}(\mathbb{L}', \ell)$, and output the restriction of $\widetilde{\mathbb{C}}'$ to the second half of its input and output bits.

In the event that $\mathcal{A}'$ succeeds, $\mathcal{A}$ also succeeds. So we have

$$\Pr\left[\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \begin{array}{c}(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}) \\ \widetilde{\mathbb{C}} \leftarrow \mathcal{A}^{\mathbb{C}}(\mathbb{L}, \mathcal{L}_{\mathsf{size}}(\mathbb{C}))\end{array}\right] \geq \sigma.$$

Now by assumption, there is a $\mathsf{poly}(t)$-time, $\mathsf{poly}(q)$-query simulator $\mathcal{S}$ for Lock with

$$\Pr\left[\widetilde{\mathbb{C}} \equiv_\epsilon \mathbb{C} \mid \widetilde{\mathbb{C}} \leftarrow \mathcal{S}^{\mathbb{C}}(\mathcal{L}_{\mathsf{size}}(\mathbb{C}))\right] \geq \tau.$$

But notice that $\mathcal{S}$ is also a simulator for Lock'. In particular, whether or not $\mathbb{C}$ is ½-separable, $\mathcal{S}$ recovers an ϵ-approximation with probability τ. Therefore a simulator exists, and Lock' is FS-secure. □

7 Secure Logic Locking

In this section, we show how to create IND-LL secure logic locking for Boolean circuits using universal circuits (§7.1). This approach provides unconditional security (no adversary, even one running in exponential time, can succeed in the IND-LL game with better than ½ probability), however it comes at the cost of a $O(n \log n)$ circuit size blowup, where n denotes the number of gates in the input circuit. We then show that, in order to do better than this, cryptographic assumptions are necessary (§7.2).

We frame our results in the context of Boolean circuits, so it is reasonable to ask whether they can be applied more broadly to the class of practical circuits. We believe that they can. We rely on the Boolean circuit formalism only to provide an evaluation semantics (i.e., input-output behavior) and a notion of circuit size. This could easily be translated into the setting of, say, combinational CMOS logic by using netlist size and an appropriate evaluation semantics in place of these. Of course, the notion of what information about a circuit is leaked is also specific to the formalism being used, but we primarily consider size (i.e., gate count) and depth (i.e., longest path length), which should be applicable to practical models as well.

We model the key to the locked circuit merely as extra input wires. In a real-world setting, of course, the end product of logic locking would be a fabricated circuit that would need to be securely combined with a key. This is orthogonal to the security of the logic locking itself, however. It is sufficient to think of key insertion as wiring the fabricated chip to a separate memory, and perhaps protecting the combination of these using tamper-proofing techniques.

7.1 Secure Logic Locking Using Universal Circuits

We now show that universal circuits can be used to create an IND-LL-secure logic locking scheme. We are not the first to make this observation: Di Crescenzo et al. [DSSY20], among others, proposed universal circuits as a secure instantiation targeting their "lock security" definition (cf. §4.3). Thus, this construction should be viewed more as a feasibility result—that is, it shows that IND-LL indeed can be satisfied—versus a particularly novel contribution of this work. That being said, modern schemes such as redaction [MAS+21] point to the potential practical feasibility of utilizing universal circuits for logic locking.

Theorem 7. *For any $\mathcal{L}$, an $(\infty, 0, \mathcal{L})$-IND-LL-secure logic-locking scheme Lock exists.*

Proof. It is sufficient to construct a scheme that is $(\infty, 0, \mathcal{L})$-IND-LL-secure where $\mathcal{L}(\mathbb{C}) = |\mathbb{C}|$ for all circuits $\mathbb{C}$. This allows the adversary to pick any two circuits as long as they are of the same size and, in particular, implies security for any other more restrictive leakage function.

Let UC_n be a universal circuit such that $\mathsf{UC}_n(\mathbb{C}, X) = \mathbb{C}(X)$ for any circuit $\mathbb{C}$ of size n and any input X. Define the algorithm $\mathsf{Lock}(\mathbb{C})$ as outputting $(\mathsf{UC}_n, \mathbb{C})$.

For any adversary $\mathcal{A}$ running in time ∞ and circuits $\mathbb{C}_0$ and $\mathbb{C}_1$ of the same size,

$$\begin{aligned}\Pr\left[\widetilde{b} = b \mid \begin{array}{c} b \twoheadleftarrow \{0,1\} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b) \\ \widetilde{b} \leftarrow \mathcal{A}(\mathbb{L}) \end{array}\right] &= \frac{1}{2}\Pr\left[\widetilde{b} = 0 \mid \begin{array}{c} (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_0) \\ \widetilde{b} \leftarrow \mathcal{A}(\mathbb{L}) \end{array}\right] \\ &\quad + \frac{1}{2}\Pr\left[\widetilde{b} = 1 \mid \begin{array}{c} (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_1) \\ \widetilde{b} \leftarrow \mathcal{A}(\mathbb{L}) \end{array}\right] \\ &= \frac{1}{2}\left(\Pr\left[\mathcal{A}(\mathsf{UC}_n) \Rightarrow 0\right] + \Pr\left[\mathcal{A}(\mathsf{UC}_n) \Rightarrow 1\right]\right) = \frac{1}{2}. \quad \square\end{aligned}$$

7.2 More Efficient Logic Locking Requires Cryptographic Assumptions

For a circuit of size n, the universal circuit approach requires a key of length $O(n \log n)$. We show that if the key size of the locking scheme is any smaller than this, then cryptographic assumptions, and in particular one-way functions, are necessary.

Definition 12 (One-way Functions)**.** A function $f : \mathcal{D} \to \{0,1\}^n$ is (t, δ)-one-way if it is efficiently computable, domain $\mathcal{D}$ is efficiently sampleable, and for any adversary $\mathcal{A}$ running in time t, we have that

$$\Pr\left[Y \in f^{-1}(f(X)) \mid \begin{array}{c} X \twoheadleftarrow \mathcal{D} \\ Y \leftarrow \mathcal{A}(f(X)) \end{array}\right] \leq \delta. \qquad \diamondsuit$$

Computational Assumptions. Di Crescenzo et al. [DSSY20] have proven that assuming one-way functions exist, it is possible to build a logic-locking scheme whose key size is small. In this paper, we extend this result by proving that one-way functions are not only sufficient but also necessary for such efficiency.

The following theorem states that any IND-LL-secure logic locking scheme that has keys smaller than $O(n \log n)$—even by a *constant* amount—can be used to construct a one-way function. Since one-way functions are believed to require cryptographic assumptions, this implies that any more efficient logic locking scheme will also require cryptographic assumptions. Note that all existing logic locking schemes that we are aware of do *not* require cryptographic assumptions, hence putting into doubt their security, at least with respect to IND-LL.

Theorem 8. *Let $\mathcal{L}$ be a leakage that returns the size of a circuit, i.e., $\mathcal{L}(\mathbb{C}) = |\mathbb{C}|$ and let Lock be a logic locking scheme that on input $\mathbb{C}$ outputs key K such that $|K| \leq |\mathbb{C}| \log |\mathbb{C}| - \gamma$ for some fixed γ. Then (t, δ)-one-way functions exist if Lock is $\left(O(t), \frac{(1-2^{-\gamma})\cdot\delta - 2^{-\gamma}}{2}, \mathcal{L}\right)$-IND-LL.*

Proof. In this proof we use the fact that any universal circuit that can evaluate circuits of size n has to have at least $n \log n$ circuit description bits, since there are at least n^n circuits of size n. If the key size is at most $n \log n - \gamma$ bits for some fixed γ, then some fixed fraction of circuits *cannot* be represented in the locked circuit, and in particular, the chance that a random circuit can be represented is at most $c = 2^{-\gamma}$.

We define a one-way function $f((\mathbb{C}_0, \mathbb{C}_1, b, r))$ as follows: Run $(\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b; r)$, where r is the explicit randomness used by Lock, and output $(\mathbb{C}_0, \mathbb{C}_1, \mathbb{L})$. We claim that f is a one-way function. For the sake of contradiction, assume that there exists an inverter $\mathcal{I}$ running in time t such that

$$\Pr_{\mathbb{C}_0, \mathbb{C}_1, b, r}[\mathcal{I}(f(\mathbb{C}_0, \mathbb{C}_1, b, r)) \in f^{-1}(f(\mathbb{C}_0, \mathbb{C}_1, b, r))] > \delta.$$

We create an $\mathsf{IND\text{-}LL}$ adversary $\mathcal{A}$ operating over fixed circuits $\mathbb{C}_0$ and $\mathbb{C}_1$. On input $\mathbb{L}$, adversary $\mathcal{A}$ computes the following:

1. $(\mathbb{C}_0', \mathbb{C}_1', b, r) \leftarrow \mathcal{I}(\mathbb{C}_0, \mathbb{C}_1, \mathbb{L})$
2. $(\mathbb{L}', K) \leftarrow \mathsf{Lock}(\mathbb{C}_b; r)$
3. If $\mathbb{C}_b = \mathbb{C}_b'$ and $\mathbb{L} = \mathbb{L}'$, return b, else, return a random bit.

In the case that $\mathcal{A}$ is given the locking $\mathbb{L}$ of $\mathbb{C}_1$, the chance that it outputs one is at least the probability that $\mathbb{L}$ cannot evaluate $\mathbb{C}_0$ (denote this event by G_0) multiplied by the probability that $\mathcal{I}$ inverts successfully (denote this event by H) plus the probability that its random guess is correct when $\mathbb{L}$ cannot evaluate $\mathbb{C}_0$ and $\mathcal{I}$ fails:

$$\begin{aligned}\Pr_{(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_1)}[\mathcal{A}(\mathbb{L}) \Rightarrow 1] &= \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_0] \cdot \Pr[G_0] + \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_0] \cdot \Pr[\neg G_0] \\ &\geq \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_0] \cdot (1 - c).\end{aligned}$$

Moreover, we have that

$$\begin{aligned}\Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_0] &= \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_0, H] \cdot \Pr[H] + \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_0, \neg H] \cdot \Pr[\neg H] \\ &= \Pr[H] + \Pr[\neg H]/2 = \delta + (1 - \delta)/2,\end{aligned}$$

where the probabilities are implicitly over $(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_1)$. Putting them together, we get that

$$\Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_1)] \geq (1 - c)\delta + (1 - c)(1 - \delta)/2. \tag{2}$$

Similarly, in the case that $\mathcal{A}$ is given the locking $\mathbb{L}$ of $\mathbb{C}_0$, the chance that it outputs one is at most the probability that $\mathbb{L}$ can evaluate $\mathbb{C}_1$ (denote this event by G_1) plus the probability that its random guess is correct when $\mathbb{L}$ cannot evaluate $\mathbb{C}_1$ and $\mathcal{I}$ fails:

$$\begin{aligned}\Pr_{(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_0)}[\mathcal{A}(\mathbb{L}) \Rightarrow 1] &= \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_1] \cdot \Pr[G_1] + \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1] \cdot \Pr[\neg G_1] \\ &= c \cdot \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | G_1] + (1 - c) \cdot \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1] \\ &\leq c + (1 - c) \cdot \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1].\end{aligned}$$

Moreover, we have that

$$\begin{aligned}\Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1] &= \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1, H] \cdot \Pr[H] + \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 | \neg G_1, \neg H] \cdot \Pr[\neg H] \\ &= 0 \cdot \delta + 1/2 \cdot (1 - \delta),\end{aligned}$$

where the probabilities are now implicitly over $(\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_0)$. Putting the two together, we get that

$$\Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_0)] \leq c + (1 - c)(1 - \delta)/2. \tag{3}$$

Combining equations 2 and 3, we get that

$$\Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_1)] - \Pr[\mathcal{A}(\mathbb{L}) \Rightarrow 1 \mid (\mathbb{L}, \_) \leftarrow \mathsf{Lock}(\mathbb{C}_0)] \geq (1 - c) \cdot \delta - c.$$

Finally, conditioning on $b = 0, 1$, we get that

$$\Pr\left[\widetilde{b} = b \mid \begin{array}{c} b \twoheadleftarrow \{0,1\} \\ (\mathbb{L}, K) \leftarrow \mathsf{Lock}(\mathbb{C}_b) \\ \widetilde{b} \leftarrow \mathcal{A}(\mathbb{L}) \end{array}\right] \geq \frac{1}{2} + \frac{(1-c)\cdot\delta - c}{2},$$

which concludes the proof. □

As an example, consider a 128-bit key K, and the class of circuits of size 2^{20}. Then $\gamma = 20 \cdot 2^{20} - 2^7 > 2^{24}$. Therefore, if the scheme is $(t, \delta, \mathcal{L}_{\mathsf{size}})$-IND-LL, then there exist $(O(t), \delta')$-one-way functions with $\delta' \approx \frac{\delta}{2}$.

Interpreting the lower bound. The above lower bound suggests that any secure locking scheme with sufficiently short keys has to somehow invoke or build a one-way function in it. One might find this result disheartening since we would like to avoid performing complex operations such as evaluating one-way functions on the chip. We stress that the result requires *the locking scheme* Lock to perform cryptographic operations and not necessarily the resulting circuit $\mathbb{L}$.

8 Conclusion and Future Work

The first step toward creating secure systems is to rigorously define what we mean by "secure." In this paper, we argue that our two proposed formal definitions for logic locking—IND-LL and SIM-LL—capture the security goals of logic locking in ways that prior definitions do not, while avoiding the technical pitfalls of those approaches.

The focus of this work is definitional in nature—as we have argued, without proper definitions, the claimed security of a construction cannot be grounded. Of course, definitions are only as useful as the constructions that follow, or alternatively, as a means to demonstrate that the security goals are simply unachievable (cf. the work on the impossibility of software obfuscation [BGI+01]). Fortunately, in the logic locking case, we have shown that security *can* be achieved, albeit at the overhead inherent in using universal circuits. The next step is then to improve on this, constructing novel secure schemes with better performance parameters than universal circuits.

We also note that the focus of this work has been in the *combinational* space. However, several logic locking schemes utilize properties of *sequential* circuits [CB09, DHW+13, MZZ+17, DY18, RSK+18, RKS18, KRV19, HYNN21], which our definitions as is cannot handle. Thus, a natural next step is to adapt our definitions to sequential circuits, and identify secure constructions in this more complex circuit model.

References

[AKHS19] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks. In *CHES*, 2019.

[APSS21] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. Omla: An oracle-less machine learning-based attack on logic locking. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 1–1, 2021.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 341–358. Springer, Heidelberg, August 1994.

[BTZ10] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Des. Test Comput.*, 27(1):66–75, 2010.

[CB09] Rajat Subhra Chakraborty and Swarup Bhunia. HARPOON: An obfuscation-based SoC design methodology for hardware protection. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[CS21] Animesh Chhotaray and Thomas Shrimpton. Hardening circuit-design ip against reverse-engineering attacks. In *S&P*. IEEE, 2021.

[CZHN21] Subhajit Dutta Chowdhury, Gengyu Zhang, Yinghua Hu, and Pierluigi Nuzzo. Enhancing sat-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation. In *ISCAS*. IEEE, 2021.

[DBN+14] Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *IOLTS*. IEEE, 2014.

[DHW+13] Avinash R Desai, Michael S Hsiao, Chao Wang, Leyla Nazhandali, and Simin Hall. Interlocking obfuscation for anti-tamper hardware. In *Proc. Cyber Security and Information Intelligence Research Workshop*, pages 1–4, 2013.

[DSSY20] Giovanni Di Crescenzo, Abhrajit Sengupta, Ozgur Sinanoglu, and Muhammad Yasin. Logic Locking of Boolean Circuits: Provable Hardware-based Obfuscation from a Tamper-proof Memory. In *SecITC*, 2020.

[DY18] Jaya Dofe and Qiaoyan Yu. Novel dynamic state-deflection method for gate-level design obfuscation. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 37(2):273–285, 2018.

[EGT19] Mohamed El Massad, Siddharth Garg, and Mahesh V. Tripunitara. The SAT attack on IC camouflaging: Impact and potential countermeasures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8), 2019.

[EHP19] Susanne Engels, Max Hoffmann, and Christof Paar. The end of logic locking? A critical view on the security of logic locking. Cryptology ePrint Archive, Report 2019/796, 2019. https://eprint.iacr.org/2019/796.

[HYNN21] Yinghua Hu, Kaixin Yang, Shahin Nazarian, and Pierluigi Nuzzo. SANSCrypt: Sporadic-authentication-based sequential logic encryption. In Andrea Calimera, Pierre-Emmanuel Gaillardon, Kunal Korgaonkar, Shahar Kvatinsky, and Ricardo Reis, editors, *VLSI-SoC: Design Trends*, pages 255–278, Cham, 2021. Springer International Publishing.

[KAG+18] Hadi Mardani Kamali, Kimia Zamiri Azar, Kris Gaj, Houman Homayoun, and Avesta Sasan. LUT-lock: A novel LUT-based logic obfuscation for FGPA-bitstream and ASIC-hardware protection. In *ISVLSI*. IEEE, 2018.

[KAHS20] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. InterLock: An intercorrelated logic and routing locking. In *ICCAD*. IEEE, 2020.

[KKN+19] Gaurav Kolhe, Hadi Mardani Kamali, Miklesh Naicker, Tyler David Sheaves, Hamid Mahmoodi, PD Sai Manoj, Houman Homayoun, Setareh Rafatirad, and Avesta Sasan. Security and complexity analysis of LUT-based obfuscation: From blueprint to reality. In *ICCAD*, 2019.

[KRV19] Yasaswy Kasarabada, Sudheer Ram Thulasi Raman, and Ranga Vemuri. Deep state encryption for sequential logic circuits. In *IEEE Computer Society Annual Symp. VLSI (ISVLSI)*, pages 338–343, 2019.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[MAS+21] Prashanth Mohan, Oguz Atli, Joseph Sweeney, Onur Kibar, Larry Pileggi, and Ken Mai. Hardware redaction via designer-directed fine-grained eFPGA insertion. In *DATE*. IEEE, 2021.

[MZZ+17] Travis Meade, Zheng Zhao, Shaojie Zhang, David Pan, and Yier Jin. Revisit sequential logic obfuscation: Attacks and defenses. In *IEEE Int. Symp. Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[RKM10] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. Ending piracy of integrated circuits. *Computer*, 43(10), 2010.

[RKS18] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. SRCLock: SAT-resistant cyclic logic locking for protecting the hardware. In *GLSVLSI*. ACM, 2018.

[RSK+18] Amin Rezaei, Yuanqi Shen, Shuyu Kong, Jie Gu, and Hai Zhou. Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks. In *DATE*. IEEE, 2018.

[RZZ+13] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Trans. Computers*, 64(2):410–424, 2013.

[RZZ+15] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok K. Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Trans. Computers*, 64(2):410–424, 2015.

[SLM+17] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. Cyclic obfuscation for creating SAT-unresolvable circuits. In *GLSVLSI*. ACM, 2017.

[SPJ19] Kaveh Shamsi, David Z. Pan, and Yier Jin. On the impossibility of approximation-resilient circuit locking. In *HOST*. IEEE, 2019.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *HOST*. IEEE, 2015.

[SRZ18] Yuanqi Shen, Amin Rezaei, and Hai Zhou. Sat-based bit-flipping attack on logic encryptions. In Jan Madsen and Ayse K. Coskun, editors, *DATE*. IEEE, 2018.

[SS20] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. *IEEE Transactions on Information Forensics and Security*, 15, 2020.

[XS17] Yang Xie and Ankur Srivastava. Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In *DAC*, 2017.

[XS18] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2018.

[YMRS16] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. In *HOST*. IEEE, 2016.

[YSN+17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *CCS*. ACM, 2017.

[ZJK17] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. CycSAT: SAT-based attack on cyclic logic encryptions. In *ICCAD*. IEEE, 2017.