Hardening Circuit-Design IP Against Reverse-Engineering Attacks

Animesh Chhotaray and Thomas Shrimpton

University of Florida

Abstract. Design-hiding techniques are a central piece of academic and industrial efforts to protect electronic circuits from being reverse-engineered. However, these techniques have lacked a principled foundation to guide their design and security evaluation, leading to a long line of broken schemes. In this paper, we begin to lay this missing foundation.
We establish formal syntax for *design-hiding (DH) schemes*, a cryptographic primitive that encompasses all known design-stage methods to hide the circuit that is handed to a (potentially adversarial) foundry for fabrication. We give two security notions for this primitive: function recovery (FR) and key recovery (KR). The former is the ostensible goal of design-hiding methods to prevent reverse-engineering the functionality of the circuit, but most prior work has focused on the latter. We then present the first provably (FR,KR)-secure DH scheme, $\mathit{OneChaff}_{\mathrm{hd}}$. A side-benefit of our security proof is a framework for analyzing a broad class of new DH schemes. We finish by unpacking our main security result, to provide parameter-setting guidance.

Keywords: · cryptography · provable security · design hiding · hardware obfuscation · logic locking · logic encryption · IC camouflaging

1 Introduction

Modern integrated-circuits (ICs, or "chips") are the product of a globally distributed supply-chain [TI19]. Much of this is driven by economics: the cost of building and operating a chip fabrication facility is exorbitant, so circuit designers are forced to send their digital intellectual property (IP) to external foundries for fabrication. In 2018, just ten foundries accounted for more than 95% of the chip-fabrication market [Tre18]. Given the lack of choice they have concerning who will fabricate and package their IP into chips, IP authors are motivated to protect their high-value circuit designs from being reverse-engineered by untrusted foundries, as this can lead to IP theft, counterfeiting, or other misuse.

These threats are enabled when the foundry obtains (effectively) the gate-and-wire layout for the circuit to be fabricated. Thus, the last decade has seen a surge in research on methods to "hide" the circuit IP [SPJ19]. We refer to such methods as *design-hiding* (DH) schemes. An important constraint on DH schemes is that the IP author still needs the foundry to fabricate something useful; simply applying traditional encryption to the gate-and-wire layout and handing the resulting ciphertext to the foundry does not suffice. (We discuss relation to other cryptographic primitives like program obfuscation, multi-party computation and function secret-sharing in Section 3.1.)

Framing the problem. Figure 1 gives a simplified picture of the setting in which DH techniques are typically deployed. An IP author attempts to hide (Figure 1(a)) the underlying functionality F that its plaintext circuit computes by presenting the foundry with an *opaque* version of the circuit. The threat assumption here is that the foundry will attempt to uncover F that enables profitable end goals (e.g., counterfeiting). Upon receiving the opaque circuit, the foundry is meant to fabricate it (Figure 1(b)) into physical chips that compute whatever the opaque circuit does. These chips will likely not compute F, but this functionality can be restored (Figure 1(c)) by the IP author or its

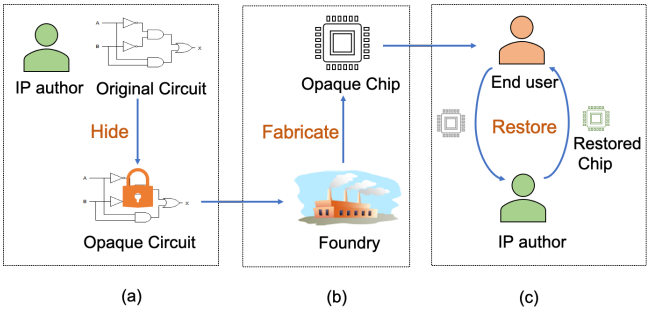

Figure 1: Simplified view of the setting for Design-Hiding schemes. The IP author is involved in hiding (a) and restoring (c) chips. The (potentially malicious) foundry controls fabrication (b) of chips.

designated proxies. Intuitively, the IP author should be the only party capable of producing a chip that computes F, via its restoration process.

Chips that have been restored may enter the market and are available for purchase by the end-user.

Characterizing prior approaches. Broadly speaking, prior work on DH schemes falls into three categories: logic locking [RKM08, ZAMKHS19, SLP+19], IC camouflaging [RSSK13, VPH+16, SLP+19], and split manufacturing [RSK13, SAFT16]. Logic-locking[1] techniques are employed when an IP author must outsource the entire fabrication process to an untrusted, third-party foundry. Here, the foundry is provided a complete circuit description (modulo, perhaps, a small block of uninitialized write-once, tamper-resistant memory) to fabricate. This is the setting most commonly considered in the literature on circuit-design hiding. Split manufacturing primarily aims to prevent an untrusted foundry from inserting hardware-trojans into the IP author's circuit [DFS16]; as the name suggests, the fabrication of chips is spread across multiple parties, one of which is (typically) the IP author or a trusted proxy. When the trust model is altered, so that the IP author trusts the foundry but not the end user, IC camouflaging methods [CBWC12, VPH+16] can be employed. Here, the foundry fabricates an opaque circuit that contains several "camouflaged" logic cells. An end user that purchases fabricated chips from the market does not know the functionality of these camouflaged cells and hence will fail to recover the hidden design.

Our focus will be on adversarial foundries. As such, IC camouflaging is out of scope. Split manufacturing targets adversarial foundries, but it requires (effectively) that the IP author has fabrication capabilities. This is most often not the case, in practice. Thus, we drill down on the logic-locking approach to hiding IP from untrusted foundries, and building DH schemes that are provably secure in the attack model assumed by logic-locking schemes. Here, the foundry has *unrestricted access*[2] to the opaque circuit and full control over the entire fabrication process, making this a challenging setting. We note that the foundry may also play the role of an end-user by purchasing packaged (restored) chips from the market. By running the chips on inputs of its choosing, it may learn the value of the original circuit design on a subset of inputs. Finally, we note that although our focus is on the logic-locking setting, our syntax also covers IC camouflaging and split manufacturing techniques.

Advancing the state of the art. Almost all logic-locking schemes have been shown to be vulnerable to efficient attacks, e.g., [SRM15, XSTF17, SZ17, SRZ18, YMSR17a, SLM+17a, SLR+19, ZJK17, Che18, SS19, YTS19] that allows the foundry to define the functionality F *in*

[1] This is one of several monikers used in the hardware community, others include hardware obfuscation, logic encryption, design withholding and encryption.

[2] We use the racially neutral term *unrestricted access* instead of whitebox access to a circuit/function. Similarly, we use *oracle access* instead of blackbox access.

full, given the opaque circuit and the ability to obtain input-output pairs $(X, F(X))$. We believe the root cause to be the lack of a principled, provable-security foundation for the area, i.e., a formal, syntactic definition of what a logic-locking (more generally, a DH) scheme is, and one or more formal security notions that capture the intended attack model and adversarial goals. We are not alone in this belief: in their 2019 paper that broke SFLL-HD [YSN+17], Sirone and Subramanyan state

> "Our results reinforce the observation that all logic locking schemes appear to be vulnerable to attack. We assert this is because the logic locking community has not adopted notions of provable security from cryptography." [SS19]

Crucially, the CCS'17 paper that presented SFLL-HD *claimed* provable security, because it resisted three prominent attacks. Sirone and Subramanyan's subsequent FALL attack [SS19] highlights the danger of such claims. A scheme should claim security, explicitly scoped by a principled and well defined security notion, only if it provably thwarts *all* (suitably efficient) attacks that are admitted by that notion.

In light of all of this, we make three primary contributions to the state of the art:

1. We formalize DH schemes as an abstract, syntactic object. A DH scheme will consist of two component algorithms, *Hide* and *Restore*, that correspond to the hiding and restoration phases just discussed. In addition, we elevate the fabrication step, of turning circuit descriptions (necessarily visible to the foundry) into fully packaged chips (which require considerable resources and expertise to "open" and analyze), to an explicit algorithm *Fab*. This has important implications for security that have previously not been surfaced.

2. We establish security notions that capture the capabilities, and goals, of an adversarial foundry. These notions make formal the attack model considered in prior works, and attends to details that have sometimes been quietly elided, e.g., a priori knowledge about the hidden function F that the foundry may have.

3. We give the first DH scheme, *OneChaff*$_{\mathrm{hd}}$, that provably protects a broad class of functions (or their circuit representation) against reverse-engineering attacks. Along the way, we observe that certain types of "simple" functions cannot be protected by any DH scheme.

2 Overview of Contributions

We now provide a more detailed overview of our contributions, before engaging with the technical core of the paper.

Formal foundations for DH schemes: Syntax. Notably absent from the area is a provable-security foundation for the design and analysis of DH schemes. Very few papers in the area offer anything along these lines. The works that do [YSN+17, MZGT17, LSM+16, SPJ19, SLP+19] fall short of what is needed, e.g., by giving syntactic descriptions that are imprecise or clearly mismatch existing schemes.

Such a foundation begins with a precise definition of a DH scheme as a syntactic object, i.e., what are the component algorithms that must be realized in a concrete scheme.

So, we begin by providing a formal syntax (in Section 5) for DH schemes, and our formalization captures all currently known methods of design-stage circuit hiding. Specifically, a DH scheme is a pair of algorithms $(\mathit{Hide}, \mathit{Restore})$ that abstract the portions (a) and (c) of Figure 1, respectively. Loosely, the *design-hiding* algorithm *Hide* takes as input a circuit C_F (and some design parameters θ), and it returns an opaque circuit C_L, along with the associated hiding key K_O. The *design-restoring* algorithm *Restore* takes an opaque *chip* $\mathbb{C}_L$, a hiding key K_O and design parameters θ as inputs; it returns either a restored chip $\mathbb{C}_F$ or an error symbol $\perp$, i.e, an indication that restoring has failed.

The opaque circuit is transformed into a chip by a separate *chip-fabrication* algorithm *Fab* that takes a circuit C_L and design parameters as inputs and returns a chip $\mathbb{C}_L$. Notice that we use the heavy typeface in $\mathbb{C}_L$ to distinguish between unrestricted access to circuits (e.g. C_L) and oracle access to chips. This is necessary as otherwise the foundry can purchase a restored chip and use invasive attacks [EHP19] to read the hiding key from (tamperproof) memory. In this work, we consider such attacks to be out of scope as protecting against invasive attacks will likely require design of special hardware like active shields [CDG+14] and are hence, orthogonal to the development of DH schemes. We note that no prior work considered the fabrication process, which turns circuits into *chips*, as a first-class syntactic primitive. The effect is that fabrication-specific security issues could not be cleanly surfaced. We will see that making the fabrication process explicit uncovers an important connection between the security of DH schemes, and detecting stealthy hardware trojans (in packaged chips). More on this in a moment.

Formal foundations: Security notions. Given a precise description of what a DH scheme *is*, we next define formal notions of what it means for a DH scheme (however it is realized) to be *secure*. An intuitive definition of reverse-engineering the opaque circuit is to recover from it, the hidden IP F by any means. But literature has tended to focus on attacks (and countermeasures) that attempt to recover the secret hiding key K_O. Thus, we give two formal notions of security (in Section 6): *function recovery* (FR) and *key recovery* (KR). In both notions, the adversary is provided unrestricted access to the opaque circuit, and various oracles that abstractly capture the powers of a foundry.

In the KR notion, the adversary's goal is to return a key K that is equivalent to K_O, in the sense that when one restores the functionality of an honestly fabricated chip using either K or K_O, we get restored chips with identical functionalities.

The FR notion captures a stronger attack model. In it, the adversary's goal is to find *any* chip that is functionally equivalent to F. As one expects, KR-insecurity implies FR-insecurity: if you can recover a key K equivalent to K_O, then you can win the FR game by returning an honestly fabricated chip restored with K. The converse is not necessarily true, i.e., reverse-engineering the hidden functionality of the opaque circuit does not necessarily require recovering something equivalent to the hiding key.

We note that certain kinds of functions cannot be protected by *any* DH scheme in the logic-locking setting, where the foundry may purchase honest chips and thereby learn input-output pairs of its choosing. For example, if the domain of the chip is small, the functionality of the chip can be recovered by querying the chip on its entire domain. In the case of Boolean functions, those whose decision-tree representations have small depth/size cannot be hidden [KM93], nor can those whose Fourier spectra contain relatively few significant components [GOS+11]. So, while our security notions are agnostic to structural characteristics of the function(s) one wishes to hide, our security results will surface this concern.

We also note that our security notions allow for fully malicious foundries that may fabricate arbitrary, "dishonest" chips, and submit these to be restored with the secret hiding-key K_O. The chip may have been fabricated from the opaque circuit, but (say) with an embedded hardware trojan that outputs K_O when triggered on a particular input. Unless knowledge of the secret K_O suffices to allow the *Restore* algorithm to detect such a trojan (and alert the IP author not to proceed), the restored chip can be run by the foundry (acting as user), leaking K_O and allowing it to win the FR game. Given the state of the art in trojan detection, we know of *no remotely practical* DH scheme that can be FR-secure against fully malicious foundries. Thus, we restrict our attention to designing DH schemes that are secure against honest-but-curious foundries, i.e., ones that will try to reverse-engineer the functionality of the IP, but will only fabricate chips that adhere to the IP author's opaque circuit. This is in keeping with all prior work on DH schemes.

A new family of DH schemes: *OneChaff*. We introduce a family of DH schemes that we call *OneChaff* (see Section 7), and analyze a particular scheme *OneChaff*$_{\mathrm{hd}}$ in this family. In *OneChaff*$_{\mathrm{hd}}$, the *Hide* algorithm takes inspiration from SFLL-flex [YSN+17] as it encodes a single

n-input-bit Boolean function H (one "chaff" function) and an uninitialized lookup table $\widetilde{\mathrm{Tab}}$ in the opaque circuit. While SFLL-flex allows arbitrary H, in our *OneChaff*$_{\mathrm{hd}}$ scheme, the chaff H matches the hidden function F, except on $\Delta \in \mathbb{N}$ uniformly chosen inputs. These are the so-called *distinguishing inputs* (DIs) for the pair (H, F). The hiding key K_O encodes the correct input-output behaviors on the DIs. (Practical *Hide* algorithms will have $\Delta \ll 2^n$.) On input of a key K, chip $\mathbb{C}_L$, and design parameters θ, the *Restore* algorithm in *OneChaff*$_{\mathrm{hd}}$ loads the key K into the (write-once, tamper-resistant) uninitialized lookup table of the chip. Under honest operation, key K is equal to K_O.

Proving security of *OneChaff*$_{\mathrm{hd}}$ for Boolean functions. After giving our foundations for DH schemes and introducing the *OneChaff* family of schemes, the remainder of this work is spent showing that *OneChaff*$_{\mathrm{hd}}$ provably prevents full recovery of *Boolean* functions in the presence of honest-but-curious adversarial foundries. While most real-world circuits do not compute functions returning a single bit, several prominent logic-locking schemes [XS16, YMRS16, YSS+17, YSN+17] only aim to hide Boolean functions. Moreover, no provably secure scheme exists for circuits implementing functions from this "base" class. We note that for circuits with multiple output bits, one can attempt to hide Boolean sub-functions that are determined by the transitive fan-in cone[3] (TFC) of individual output bits.

Our main security result (Theorem 3) gives an upperbound on the probability that a computationally bounded, honest-but-curious foundry manages to win the FR game against *OneChaff*$_{\mathrm{hd}}$. To the best of our knowledge, this is *the first positive provable-security result* on DH schemes.

Security holds for Boolean functions that are not "simple" in the sense we mentioned earlier (no DH scheme can hide those) under some assumptions about the *a priori* knowledge that the adversary has about F. All prior schemes assume that the adversary has *no* a priori knowledge of F. Such an assumption is unrealistic and also makes the adversary weak as the initial "guess" space of the adversary is the set of all Boolean functions. In our FR analysis of *OneChaff*$_{\mathrm{hd}}$, we assume that the adversary knows a priori the hamming weight h, i.e., the number of inputs that cause F to output one. This narrows the initial "guess space" to Boolean functions that have hamming weight of h. Also, the hamming-weight parameter allows us to capture the fact that the number of functions in the guess space of the adversary increases exponentially in h. Hence, functions with hamming weight close to 2^{n-1} will be more secure compared to functions with hamming weights close to zero or 2^n. This is also intuitive and in agreement to a result from learning theory that states that a random Boolean function (with sufficiently large domain) will not be simple, as it will lack the highly concentrated Fourier spectral structure that is typically needed for a function to be learnable. Note that a random Boolean function will have expected hamming weight close to 2^{n-1}.

Our analysis essentially bounds the number of functions that remain in the adversary's guess space after some number of true input-output observations $(X_1, F(X_1)), \ldots, (X_q, F(X_q))$ are (adaptively) obtained. Intuitively, if a large number of functions remain in the guess space, then the probability of winning the FR-game will be small, and conversely if the adversary is able to eliminate all but a few functions, the adversary's winning probability will be close to one.

From Theorem 3, and the analysis leading to it, we can glean some useful observations. In particular, the IP author should use *OneChaff*$_{\mathrm{hd}}$ to protect Boolean functions (or Boolean sub-functions) that have large domains, and hamming weights not too close to 0 or 2^n. Functions with small domains cannot be hidden by any DH scheme, at least not without severely restrictive assumptions on the adversary. When the hamming weight tends towards 0 or 2^n, the function tends towards a constant function. Intuitively, as the (known) hamming weight of the hidden function moves away from 2^{n-1} towards either 0 or 2^n, the number of possible functions decreases. This makes it more likely, although not necessarily "likely", that the hidden function can be guessed after seeing the opaque circuit and some true $(X, F(X))$ pairs. Finally, the IP author should choose to make Δ as large as is feasible. Intuitively, if Δ is small, the number of functions that remain in the adversary's guess space after it gets access to the opaque circuit (that encodes chaff H and

[3] The transitive fan-in cone of an output bit in a function is the smallest subgraph in the DAG (circuit) representation that connects the primary inputs to the output bit.

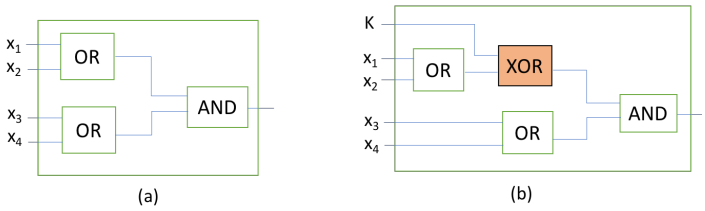

Figure 2: Random-logic locking (RLL) of a simple circuit. a) IP author's original circuit C_F b) Opaque circuit C_L obtained using RLL. When $K = 0$, the opaque circuit is correctly restored, i.e., $C_L(0, X) = C_F(X)$ for all inputs $X = x_1x_2x_3x_4$. The randomness in RLL is in the selection of the wire where the XOR gate will be inserted; in this example, RLL had seven choice of wires in C_F.

a lookup table $\widetilde{\text{Tab}}$) will also be small compared to large Δ. Note that the new guess space will contain only functions that have a hamming weight of h (due to its a priori knowledge of the hamming weight of F) and that also differ from H on Δ DIs (by construction).

3 Related Work

After more than a decade of research on practical logic-locking schemes (see Figure 2 for an example), almost all constructions are broken by attacks that recover the secret hiding key. A majority of these attacks exploit algorithmic weaknesses in the constructions. For example, in 2015, Subramanyan et al. [SRM15] developed a powerful attack that used a SAT solver to iteratively narrow the space of possible keys; for several logic-locking schemes [RKM08, BTZ10, RPSK12a, RPSK12b, DBDN+14] the SAT-attack algorithm [SRM15](Algorithm 1) can eliminate *all* incorrect keys using a small number of $(X, F(X))$ pairs.

The logic-locking schemes that were broken by the SAT attack were, in fact, designed to prevent the adversary from successfully mounting a key-recovery (KR) attack. The authors of these schemes *claimed* security based on the following: (a) these schemes admit large keys and the correct key is sampled uniformly at random; and (b) the schemes are resilient against *specific* attacks, the SAT attack not among them. From a provable-security viewpoint, these are necessary but not sufficient to claim security in any realistic adversarial model. Moreover, the security goal of preventing key recovery misses the original intent of DH schemes; namely, to prevent the adversary from learning the *functionality* of the IP author's circuit. Thus, preventing *function recovery* (FR) is arguably a more pertinent security goal.

We note that the SAT attack, and other key-recovery attacks that preceded it, are admitted by our formal KR-security notion. That notion and our formal FR-security notion share the same abstract attack model, and Theorem 1 shows that KR *insecurity* immediately implies FR *insecurity* Thus, the existence of these attacks implies that prior logic-locking schemes cannot achieve our notion of function recovery security.

Post 2015, the security goal shifted from preventing key recovery, to thwarting the SAT attack. Several SAT-attack-resistant DH schemes [XS16, YMRS16, YSS+17, SLM+17b, YSN+17, RMKS18, KAHS19] were designed. Some of these schemes [XS16, YMRS16, YSS+17] thwarted the SAT attack by increasing the number of true input-output pairs that were needed to recover the key, thereby making the SAT attack infeasible in practice. However, in this pursuit, they effectively leaked the entire functionality of the IP author's circuit. In AntiSAT [XS16], SARLock [YMRS16], TTLock [YSS+17] and SFLL-HD-0 [YSN+17], the circuit that results by restoring the opaque circuit with an incorrect key differs with C_F on only one or two inputs. The remaining schemes [SLM+17b, RMKS18] tried throttling the SAT solver (e.g. CryptominiSAT [SNC09])— the workhorse in SAT attack. As far as we know, all SAT-attack-resistant

schemes are claimed to be broken [XSTF17, SZ17, SRZ18, YMSR17a, SLM+17a, SLR+19, ZJK17, Che18, CCB19, CCB18], except the recent constructions: SRCLock (2018) [RMKS18] and FullLock (2019) [KAHS19]. It is worth noting that both SRCLock and FullLock are designed to thwart SAT-style attacks by introducing key-controllable "switching" networks [NASR04, SL91] which are known to present difficult instances for SAT-solvers; the positions in the original circuit where the networks are inserted are arbitrarily selected (similar to RLL and its variants [RKM08, BTZ10, RPSK12a, RPSK12b, DBDN+14]). The switching networks typically have a unique topology and hence will be easily identifiable in a circuit containing other components. Since an adversary gets full view of the opaque circuit via its unrestricted access, it can potentially remove these networks to get a "simplified" opaque circuit. We suspect that such a strategy can reduce the security of FullLock and SRCLock to the security of RLL and its variants. We leave verification of this hypothesis for a future work.

3.1 Relation to existing cryptographic primitives.

Our security notions capture an attack on the privacy of the IP author's circuit design. Thus, it is natural to think of related cryptographic primitives: program obfuscation, multi-party computation, encryption and function secret sharing.

PROGRAM OBFUSCATION. Design hiding is *orthogonal* to recent work in cryptography on program/circuit obfuscation (e.g. [BR14, BGI+01]). Loosely, obfuscation in the latter setting makes the syntactic requirement that the obfuscated circuit have the same input-output functionality as the original circuit. In the design-hiding setting, at an abstract level, the opaque circuit implements a *set* of functions. It is only when the chip is restored, via the secret key, that the chip must faithfully compute the intended function.

MULTIPARTY COMPUTATION. Secure multi-party computation [Yao82, EKR+18, BHR12, Lin05] allows two or more parties to "securely" compute some arbitrary function on inputs that are secrets of the respective parties.

Though MPC has become increasingly feasible to deploy in real-world systems [BHKR13, KMR14, ZRE15, NNOB11, CT16, KMR12, JNO14, MNP+04, KSMB13, Mal11, SHS+15], we *cannot* use it to design secure DH schemes in the logic-locking setting because of the fundamental differences in the threat models.

Logic locking involves two parties: the IP author (whose IC design needs to be protected), and the adversarial foundry (who is responsible for fabricating multiple opaque chips from the opaque circuit). The two popular settings of two-party computation that have some similarity to the logic-locking setting are: secure-function evaluation (SFE) [BHR12, KMR12, EKR+18] and private-function evaluation (PFE) [BHR12, EKR+18, CT16, Lin05], and the threat models of both are *different* compared to logic locking.

In SFE, the function that is evaluated is public, and the input of the other party is private. Clearly, SFE is not suitable in the logic-locking setting as it requires the IC design of the IP author to be public. In PFE, the function that is evaluated is a private input of one party, and the input to the function is private to the other party. The logic-locking setting satisfies this aspect of PFE as the IC design is the private input of the IP author. However, the adversarial foundry gets a lot of side information about the concealed IC design due to its unrestricted access to the opaque circuit and oracle access to an honestly-restored chip. This violates the PFE security goal as the party that does not have access to the function — secret input of the other party — should learn *only* the value of the function on the secret input. Hence, PFE is also not suitable in the logic-locking setting.

ENCRYPTION. In order to fabricate chips, the foundry parses the layout (described using the GDSII format) of a circuit design to replace the API calls to its hardware library with physical circuits. Thus, one cannot simply encrypt (say) the layout file before handing it to the foundry: secure encryption schemes produce ciphertexts that are indistinguishable from random bit-string, which almost certainly will not encode a valid circuit. Even if it did, that circuit would have no

relationship to one realizing the intended functionality, thereby forfeiting the economic benefit of outsourced fabrication. And clearly the foundry cannot be allowed to decrypt the ciphertext, if it is not trusted in the first place.

FUNCTION SECRET SHARING. In 2015, Boyle et al. [BGI15] introduced a cryptographic primitive called function secret sharing (FSS) that partitions a function F into multiple shares — the shares are distributed among multiple parties — such that an adversary that does not have access to all the shares learns *nothing* about F. The threat model of FSS prevents the adversary to learn the value of F on any input. In fact, if the adversary gets oracle access to F akin to the DH setting, the security of the FSS construction [BGI15][Section 3] for point functions in the two-party setting falls apart as the adversary can use a single oracle query to identify the point function that was protected by FSS in the security experiment.

4 Preliminaries

Basic notation. When X, Y are strings, we establish the following notations. We write $X \| Y$ to denote the concatenation of X and Y; $X[i]$ for the i-th element of X; and $|X|$ to denote the length of X. We extend the last two notations to ordered objects (e.g., a sequence, list, table).

When T is any ordered object, we write $T[i]$ for the i-th element, and $|T|$ to denote the number of elements in T. In pseudocode, our convention will be: all such T are initialized to $T[i] = \perp$ for all values of i, where $\perp$ is a distinguished symbol. Likewise, all sets will be initially empty. We use the notation $\langle V \rangle$ to denote the encoding of object V as a bit string. The method of encoding is left implicit, and it is silently overloaded to accommodate whatever is the type of V.

When m is an integer, we use the standard notation $[m]$ to denote the set $\{1, 2, \ldots, m\}$. We write $v_1, v_2, \ldots, v_r \xleftarrow{\$} V$ to denote sampling (with replacement) $r > 0$ uniform elements of V, where V is some non-empty set. An unembellished $\leftarrow$ denotes deterministic assignment. This notation is also used for randomized algorithms, i.e., $x \xleftarrow{\$} A(\cdots)$ means that algorithm A runs on its indicated inputs, and halts with an output that is assigned to x. (In this case, the distribution on output values is determined by A, and is not necessarily uniform.)

We write $A^{\mathcal{O}_1, \mathcal{O}_2, \cdots}$ to denote that algorithm A has oracle access to the superscripted oracles. For randomized algorithms X that may return a distinguished symbol $\perp$, the support of the algorithm $\mathrm{Sup}(X)$ is defined only over non-$\perp$ outputs. An *adversary* is a randomized algorithm.

Functions and their representations. When $\mathcal{D}, \mathcal{R}$ are non-empty sets, we write $\mathsf{Func}(\mathcal{D}, \mathcal{R})$ for the set of all functions $F \colon \mathcal{D} \to \mathcal{R}$. We write $\mathsf{Func}(n, m)$ as shorthand for functions with $\mathcal{D} = \{0,1\}^n$ and $\mathcal{R} = \{0,1\}^m$. When $m = 1$, $\mathsf{Func}(n, 1)$ is the set of all Boolean functions. We will use three representations of functions in this work: *circuits*, *chips* and *lookup tables*. While circuits and chips will be used to describe the design phase of the IC supply chain and are important to denote the type of access (oracle or unrestricted) to a circuit that an adversary gets as part of the threat model, lookup tables will be used primarily for functional analyses.

Formally, a *circuit* C_F is a directed acyclic graph that implements some mapping $F \in \mathsf{Func}(\mathcal{D}, \mathcal{R})$. Access to circuits will always be unrestricted. This captures a reality of our setting, in which the foundry is handed a description of a circuit to be fabricated. Once a circuit C_F is fabricated, we will refer to it as a *chip* and use the heavy typeface $\mathbb{C}_F$ to make this distinction clear. Crucially, access to $\mathbb{C}_F$ is *not* unrestricted; rather an adversary can only use $\mathbb{C}_F$ to make *oracle* queries. This syntactic choice is to make invasive attacks [EHP19] on chips (to leak secrets) out of scope as they are orthogonal to the (algorithmic) development of design-hiding schemes — the central primitive in our work.

The lookup-table representation of a function F with domain $\mathcal{D}$ and range $\mathcal{R}$ is a table $T_F = \langle (X_1, Y_1), (X_2, Y_2), \ldots \rangle$, where $X_i \in \mathcal{D}$ and $Y_i = F(X_i)$.

In all representations of functions, when the underlying mapping is implicit/understood, we will omit the subscript.

When F, G are two n-bit to m-bit functions, the *hamming distance* between F and G is the number of inputs X_i on which the value of the functions differ. We use $\mathcal{I}^{\neq}(F, G)$ to denote the set of such *distinguishing inputs*, i.e., $\mathcal{I}^{\neq}(F, G) = \mathcal{I}^{\neq}(G, F) = \{x \in \{0,1\}^n \mid F(x) \neq G(x)\}$. Formally, $\mathsf{hd}(F, G) = |\mathcal{I}^{\neq}(F, G)|$. We write $F \equiv G$ whenever $\mathsf{hd}(F, G) = 0$.

We will also find it useful to define the *hamming weight* of Boolean functions. When $F \in \mathsf{Func}(n, 1)$, the *hamming weight* of F is defined as the number of inputs that map to one, i.e., $\mathsf{hw}(F) = |\{X \in \{0,1\}^n \mid F(X) = 1\}|$. Also, for Boolean functions, we will use sets $\mathcal{X}_i(F) = \{X \mid F(X) = i\}$, where $i = 0$ or 1, to denote the set of inputs for which F map to 1 or 0. Notice that these two sets fully define F as we can construct the truth table of F using $\mathcal{X}_0(F)$ and $\mathcal{X}_1(F)$.

For any function $F \in \mathsf{Func}(n, m)$, we define the Δ-*neighborhood* of F as $N_\Delta(F) = \{H \in \mathsf{Func}(n, m) \mid \mathsf{hd}(F, H) = \Delta\}$.

5 DH Schemes

We begin by defining design-hiding scheme as a syntactic object. Loosely speaking, the syntax describes the inputs and outputs of the core algorithms that any DH scheme must realize, as well as what it means for a DH scheme to operate correctly.

Definition 1. Fix integers $n, m > 0$. A *design-hiding (DH) scheme* $\Pi = (\mathit{Hide}, \mathit{Restore})$ for $\mathsf{Func}(n, m)$, with key length $k_o\colon \{0,1\}^* \to \mathbb{N}$ is a tuple of algorithms with the following syntax.

- The randomized *design-hiding* algorithm *Hide*:
  Inputs: a circuit C_F implementing $F \in \mathsf{Func}(n, m)$, and a string of design parameters θ.
  Outputs: the distinguished symbol $\perp$ ("error"), or a tuple consisting of (1) a secret key $K_O \in \{0,1\}^{k_o(\theta)}$, (2) and an opaque circuit C_L, where $L \in \mathsf{Func}(n + k_o(\theta), m)$.
  Requirements: For all (F, θ), either $\Pr[\mathit{Hide}(C_F, \theta) = \perp] = 1$ or $\Pr[\mathit{Hide}(C_F, \theta) = \perp] = 0$.
- The deterministic *design-restoring* algorithm *Restore*:
  Inputs: a key $K \in \{0,1\}^{k_o(\theta)}$, a string of design parameters θ, and a chip $\mathbb{C}_L$.
  Outputs: a restored chip $\mathbb{C}_F$ or $\perp$. When K_O — the key that is used by *Hide* to produce C_L — is the key input to *Restore*, we refer to the resulting chip $\mathbb{C}_F$ as an *honestly-restored* chip.
  Requirements: for every input $(K, \theta, \mathbb{C}_L)$, it must be that $\mathbb{C}_F \leftarrow \mathit{Restore}(K, \theta, \mathbb{C}_L) \neq \perp$ implies that $F \in \mathsf{Func}(n, m)$.

We assume that if an algorithm is called on a point outside of its domain, in particular if any of its inputs are $\perp$, then the algorithm returns $\perp$. $\diamond$

The syntax that we have just established is fashioned to capture the techniques — logic locking, IC camouflaging, and split manufacturing — that an IP author uses to (a) protect the "privacy" of its high-value circuit-design C_F from adversarial entities in the post-design phase of the IC supply chain, and (b) make functionally-correct chips available to the end user. Let us elaborate using the logic-locking setting.

The IP author often outsources the fabrication of C_F into physical chips $\mathbb{C}_F$ to third-party foundries. (We will formalize this transformation in the next section.) The foundries are potentially malicious entities. Hence, the IP author cannot give the foundries the circuit design C_F (in plaintext). The design-hiding algorithm *Hide* abstracts the mechanism by which the IP author turns C_F into an opaque circuit C_L that "hides" the functionality F using a secret key K_O. The circuit C_L takes $k_o(\theta) + n$ bits of input, where n is the length of the input to F. Here, the additional $k_o(\theta)$ bits encode the key K_O, which will be used to restore the functionality of the chips (that the foundry produces) to the original, intended functionality F. Specific instantiations of *Hide* include the locking process in *logic-locking* schemes like random logic-locking (RLL) [RKM08] and its variants [RPSK12a, RPSK12b, DBDN+14, PM14], SAT-attack-resistant schemes [YSN+17, XS16, YMRS16, YSS+17, KAHS19], etc. We insist that for a given pair (F, θ), either *Hide* works always (returns valid K_O and L) or it always fails (returns $\perp$).

The meaning of the parameter string θ depends heavily on the particular instantiation of the design-hiding scheme. For example, it may encode the number of "key gates" to be inserted in an RLL scheme [RKM08], or the target hamming distance in SFLL-HD [YSN+17]. It may also encode other design constraints, such as the maximum size and depth of a locked circuit, restrictions on gate types, and so on.

We envision that the IP author will securely store the key K_O, and send the opaque circuit C_L to the foundry, instead of the (plaintext) circuit C_F. The foundry will fabricate and package one or more opaque chips; if the foundry is *honest*, then each chip will implement L. We formalize this in a moment.

Intuitively, the design-restoring algorithm *Restore* abstracts the mechanism by which a fabricated and packaged chip $\mathbb{C}_L$ is restored to its original, intended functionality F. Loosely, this entails fixing the $k_o(\theta)$ "key bits" in the input to $\mathbb{C}_L$ to K_O. In logic locking, the opaque chips are restored by (at least) having a key installed in some tamper-proof, one-time writable memory unit within the chip.

Fabrication and DH-scheme correctness. In order to define the correctness for a DH scheme, we need some mechanism for turning an opaque circuit into a chip. This is exactly the role of the fabrication process that the foundry is meant to provide. Thus, let *Fab* be a randomized *chip-fabrication algorithm*: it takes as inputs a circuit C_L and a string of design parameters θ, and it outputs either a chip $\mathbb{C}_L$, or the error symbol $\perp$.

Definition 2. A DH scheme $\Pi = (\textit{Hide}, \textit{Restore})$ is *correct* with respect to chip fabrication *Fab* if, for any (F, θ) and any $(K_O, C_L) \in \mathrm{Supp}(\textit{Hide}(C_F, \theta))$, we have

$$\Pr[\, \mathbb{C}_G \leftarrow\!\!\$\ \textit{Restore}(K_O, \theta, \textit{Fab}(C_L, \theta)) : (\mathbb{C}_G \neq \perp) \Rightarrow (G \equiv F)\,] = 1,$$

where, the probability is over the coins of *Fab*. ◇

In words, this requirement asserts that whenever (F, θ) is a pair that results in an opaque circuit C_L with associated key K_O, it must be the case that an honestly-restored chip computes F exactly.

We will find it useful to define an *honest* chip-fabrication algorithm. Loosely, we say that a chip-fabrication algorithm *Fab* is honest if the chip that it produces computes exactly what it is supposed to compute. Formally, this requires that for any circuit C_L and parameters θ, $\Pr\left[\, \mathbb{C}_{\hat{L}} \leftarrow\!\!\$\ \textit{Fab}(C_L, \theta) : \left(\mathbb{C}_{\hat{L}} \neq \perp\right) \Rightarrow (\hat{L} \equiv L)\, \right] = 1$, where the probability is over the coins of *Fab*. We note that the correctness of a DH scheme does not require honest *Fab*; indeed, $\textit{Fab}(C_L, \theta)$ may produce a chip $\mathbb{C}_{\hat{L}}$ such that for some $Y \in \mathrm{Dom}(F)$, $\hat{L}(K, Y) \neq L(K, Y)$ for (say) a small set of keys $K \neq K_O$. For example, a hardware trojan [CB11] can be embedded in the description of *Fab* such that a fabricated opaque chip $\mathbb{C}_{\hat{L}}$ implements $L(K_O, \cdot)$ honestly; on keys $\hat{K} \neq K_O$, $\hat{L}$ may leak information about K_O.

6 Security Notions

We consider two notions of security for DH schemes. Both notions deal with an adversarial foundry that attempts to recover the original functionality that the opaque circuit hides. The adversary gets unrestricted access to the opaque circuit, as this is something that a real foundry would receive in order to carry out fabrication. It is also given oracles that capture various capabilities that a foundry is likely to have.

Function recovery. We begin with a notion of function recovery for a design-hiding scheme $\Pi = (\textit{Hide}, \textit{Restore})$. The pseudocode for the FR experiment is given in Figure 3. The experiment is parameterized by the DH scheme Π, some design parameters θ that Π uses, and an honest chip-fabrication algorithm *Fab*. It takes as inputs: an adversary's attack algorithm A, and a set $\mathcal{F} \subseteq \mathsf{Func}(n, m)$. Intuitively, the set $\mathcal{F}$ captures the *a priori* uncertainty/knowledge that the

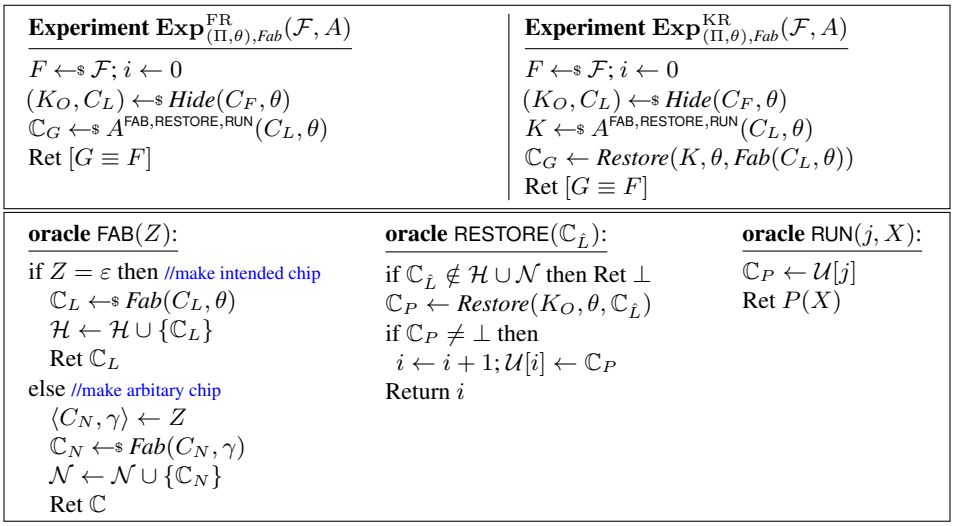

Figure 3: Experiments for function-recovery (FR) and key-recovery (KR) notions for DH scheme $\Pi = (Hide, Restore)$, given $\mathcal{F} \subseteq \mathsf{Func}(n, m)$ and design parameters θ, when chips are fabricated according to *Fab*.

adversary has about the function F that the DH scheme Π is used to protect. Clearly, there must be some uncertainty, since otherwise, the adversary already knows the functionality of the IP author's circuit design. Note that the design of Π does not depend on $\mathcal{F}$.

The experiment begins by sampling F uniformly from $\mathcal{F}$. A circuit C_F (that implements F) and the design parameters θ are inputs to the hiding algorithm *Hide*, which returns the key K_O, and the opaque circuit C_L. The adversary is given C_L and θ as inputs, and it is provided oracles named FAB, RESTORE, and RUN. These oracles model processes that an adversarial foundry can employ while trying to recover F.

When the foundry is *honest-but-curious*, it will *only* produce chips that are fabricated from the opaque circuit that the IP author generates using the DH scheme Π, and the design parameters θ. In the *fully-malicious* setting, the adversary can produce any *arbitrary* chip using arbitrary circuits and arbitrary design parameters.

For our security experiments, we define an honest-but-curious adversary as one that *always* queries FAB with zero arguments, i.e., $Z = \varepsilon$. On the other hand, a fully-malicious adversary can call FAB with any circuit C_N, and any design parameters γ of its choice, i.e., $Z = (C_N, \gamma)$.

When $Z = \varepsilon$, the FAB oracle runs *Fab* with C_L (generated in the FR experiment) and θ as inputs to get an *honest* chip $\mathbb{C}_L$. Otherwise, FAB parses Z to get (C_N, γ) and runs the *Fab* algorithm on (C_N, γ) to get an *arbitrary* chip $\mathbb{C}_N$ of its choice. We use the sets $\mathcal{H}$ and $\mathcal{N}$ to keep track of the honest and arbitrary chips, respectively.

The RESTORE oracle models the adversary's ability to obtain honestly-restored chips, i.e., chips that are restored with the secret key K_O. We allow the adversary to query RESTORE on any chip that it obtained from FAB, i.e., chips in set $\mathcal{H} \cup \mathcal{N}$. We use $\mathcal{U}$ to keep track of honestly-restored chips. The RESTORE oracle does not return the restored chips in order to prevent the adversary from reading the secret key from the description of the chip. Instead, we return the index of the restored chip in $\mathcal{U}$. Notice that we do not restrict the adversary to run the deterministic *Restore* algorithm *locally* on any triple $(K, \gamma, \mathbb{C}_N)$ of its choice.

The RUN oracle captures the foundry's ability to see the output of any honestly-restored chip $\mathbb{C}_P$ (that is stored in $\mathcal{U}$) on any input of its choice. The oracle takes as input the index j of $\mathbb{C}_P$ in $\mathcal{U}$ and $X \in \{0,1\}^n$, and returns the value $P(X)$. (By notation, $\mathbb{C}_P$ implements function P.)

The goal of the adversary in the FR experiment is to output a chip $\mathbb{C}_G$ as its guess for F. The

adversary is said to win the FR experiment if $F \equiv G$. Notice that we prevent trivial wins — by returning honestly-restored (honestly-fabricated) chips — by not allowing the adversary direct access to chips that are restored by the RESTORE oracle.

We define the *FR advantage* of the pair $(\mathcal{F}, A)$ against DH scheme Π, design parameters θ, and honest-fabrication algorithm *Fab* to be

$$\mathbf{Adv}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) = \Pr\Big[\,\mathbf{Exp}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) = 1\,\Big],$$

where the probability is over the indicated experiment. We say A is (t, q_f, q_s, q_r)-resource when its time complexity is t, and it makes q_f queries to the FAB oracle, q_s queries to the RESTORE oracle, and q_r queries to the RUN oracle. By convention, an FR adversary does not make pointless queries to any oracle, i.e. queries that cannot increase its advantage.

Key recovery. The notion of key recovery (KR) is similar to the FR notion, except the adversary's goal is to recover a key K, and the key is then used to get an honestly restored chip $\mathbb{C}_G$ that is obtained by running the honest *Fab* algorithm on (C_L, θ). In the final step of the KR notion, we check whether the function G that the chip $\mathbb{C}_G$ implements is functionally equivalent to F. We define the *KR advantage* of A as

$$\mathbf{Adv}^{\mathrm{KR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) = \Pr\Big[\,\mathbf{Exp}^{\mathrm{KR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) = 1\,\Big],$$

where the probability is over the indicated experiment. The resources are the same as those for the FR advantage.

Note that one might think it more natural to define key recovery as determining *the* secret key K_O. We define it as we do because some of the existing DH schemes, including RLL [RKM08] and strong logic obfuscation (SLO) [RPSK12a], admit multiple keys $K \in \{0,1\}^{k_o(\theta)}$ that map to the hidden function F, i.e., $F \equiv \mathit{Restore}(K_O, \theta, \mathbb{C}_L) \equiv \mathit{Restore}(K, \theta, \mathbb{C}_L)$. (These keys are said to constitute an equivalence class of the hidden function.) Thus, our KR notion captures (for example) the SAT attack of Subramanyan et. al [SRM15], which recovers some key in the equivalence class of the hidden function. It also captures other key-recovery attacks [XSTF17, YMSR17a, SS19] that exploit structural and functional characteristics of the opaque circuit.

It is intuitively clear that FR security implies KR security for any design-hiding scheme. We formalize this relation in Theorem 1.

Theorem 1. *Fix integers $n, m > 0$, $\mathcal{F} \subseteq \mathsf{Func}(n, m)$, and design parameters θ for a DH scheme $\Pi = (\mathit{Hide}, \mathit{Restore})$. Let Fab be honest. For any KR-adversary A, there is an FR-adversary B such that*

$$\mathbf{Adv}^{\mathrm{KR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) \le \mathbf{Adv}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, B).$$

Here, A is (t, q_f, q_s, q_r)-resource and B is $(O(t), O(q_f), O(q_s), O(q_r))$ resource; $t = O(q_f + q_s + q_r)$. ♦

The proof of this theorem is straightforward. Given the KR-adversary A guesses K, the FR adversary B can generate $\mathbb{C}_G$ by fixing the $k_o(\theta)$ input bits in the opaque circuit C_L with K (using the deterministic algorithm *Restore*), and then using *Fab* to fabricate the restored circuit. With $\mathbb{C}_G$, B wins the FR experiment with probability no less than the probability with which A wins the KR experiment.

The converse, i.e., KR security implies FR security, is not true. Consider that the opaque circuit that the hiding algorithm outputs is an encoding of the original circuit, and the secret key is sampled uniformly at random from $\{0,1\}^{128}$. While admittedly pathological, this example suffices to make the point.

Fully-malicious adversaries break KR security. Hardware trojans are malicious modifications to a target circuit that are hard to detect [CB11]. For example, a key-leaking hardware trojan in a cryptographic IP core (say, AES) leaks the secret key, when the IP core is run on a small and

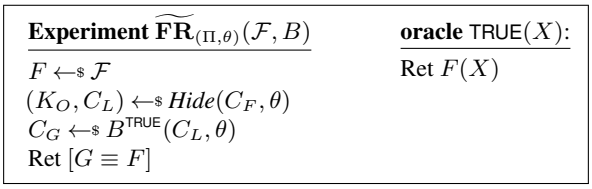

Figure 4: FR-game in the honest-but-curious setting. All three oracles in the original FR-security experiment are replaced with TRUE.

specific sequence of triggering inputs. A standard assumption in trojan-insertion attacks requires the adversary to have unrestricted access to the target circuit in order to make trojans stealthy.

In our security experiments, an adversarial foundry gets unrestricted access to the opaque circuit C_L that the experiments generate using *Hide*. When the foundry is fully malicious, it can insert a key-leaking hardware trojan (T) in C_L to leak the key K_O.

Consider the following (slightly informal) KR attack in the fully-malicious setting against *any* DH scheme Π where: *Restore* fixes the $k_o(\theta)$ key-input bits to restore the original, intended functionality. (Logic-locking schemes like RLL [RKM08] and its variants [RPSK12a, RPSK12b, DBDN+14] and SAT-attack-resistant schemes [YSN+17, XS16, YMRS16, YSS+17, KAHS19] fit this description perfectly.)

- Query FAB with C_L^T to get a chip $\mathbb{C}_L^T$, where C_L^T denotes the opaque circuit C_L with trojan T.
- Query RESTORE with $\mathbb{C}_L^T$ to get the index p of an honestly-restored chip $\mathbb{C}_P$. Let $\mathcal{I}$ be the sequence of triggering inputs for the trojan T, and $v = |\mathcal{I}|$. By design, $\mathbb{C}_P$ contains the trojan T and leaks the key K_o when $\mathbb{C}_P$ is run on inputs in $\mathcal{I}$.
- Query RUN v times with inputs $(p, \mathcal{I}[1]), (p, \mathcal{I}[2]), \ldots, (p, \mathcal{I}[v])$ to recover K_O.

This attack breaks the KR security of Π. In order to thwart this attack, *Restore* has to detect arbitrary, key-leaking hardware trojans. We do not know how this can be achieved in the setting of design-hiding schemes such as logic locking — the main focus of our work — where the IP author outsources the entire fabrication process to an external foundry. *Therefore, our upcoming security theorems will assume that the adversary is honest-but-curious.* Recall that none of the existing DH schemes considered *Fab* as part of the syntax. Hence, by default, these schemes are designed in the honest-but-curious setting.

Note that Dziembowski et al. [DFS16] showed that it is possible to design "trojan-resilient" circuits using split manufacturing as a DH scheme; as mentioned earlier, split manufacturing requires the IP author to fabricate a portion of the circuit and hence is not a common setting in IC supply chain; instead we focus on the stronger logic-locking setting in this work.

Simplified FR-notion in the honest-but-curious setting. Recall that in the security experiments, the goal of the adversary is to recover the full-functionality of F. In the honest-but-curious setting, the foundry effectively will have oracle access to F and unrestricted access to L. Let's see this in the context of the FR-security definition.

Let C_L be the opaque circuit that the IP author generates using *Hide*. In the honest-but-curious setting, the foundry uses *Fab* honestly. By definition of honest *Fab*, any opaque chip that *Fab* produces will implement L. Thus, running (the deterministic) *Restore* algorithm on the honestly-fabricated opaque chips with K_O will result in chips with identical functionalities. Since the adversary gains no additional information about F from multiple queries to the oracles FAB and RESTORE, we can fix $q_f = 1$ and $q_s = 1$, where the only (useful) query that A makes to FAB is $\hat{Z} = \langle C_L, \theta \rangle$, and the only (useful) query that A makes to RESTORE is $\widehat{\mathbb{C}_L} = \mathit{Fab}(\hat{Z})$. Notice that this results in $\mathcal{U}$ to store a single restored chip $\mathbb{C}_F = \mathit{Restore}(K_O, \theta, \widehat{\mathbb{C}_L})$ at index one. Using its q_r queries $(1, X_i)$ to RUN oracle, the adversary will learn q_r pairs $(X_i, F(X_i))$. Thus, in the

honest-but-curious setting, we can replace all the oracles with a single TRUE oracle, that takes X_i as input and returns $F(X_i)$. We show these changes in the security experiment $\widetilde{\mathbf{FR}}_{(\Pi,\theta)}(\cdot,\cdot)$; See Figure 4.

Notice that the changes are all either syntactic (i.e., no change to the advantage) or conservative (i.e., cannot decrease advantage) with respect to the original FR experiment. This observation gives rise to the following simple, but useful lemma.

Lemma 1. *Fix DH scheme* Π *and parameters* θ*. For every honest-but-curious FR-adversary* A *that is* (t, q_f, q_s, q)*-resource, there exists a closely related adversary* B *such that*

$$\Pr\Big[\mathbf{Exp}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}, A) = 1\Big] \leq \Pr\Big[\widetilde{\mathbf{FR}}_{\Pi,\theta}(\mathcal{F}, B) = 1\Big]$$

where B *makes at most* q *queries to* TRUE *(see Figure 4), and runs in time* $O(t)$*.* ♦

As discussed in Section 3, almost all existing DH schemes [RKM08, BTZ10, RPSK12a, RPSK12b, DBDN+14, XS16, YMRS16, YSS+17, SLM+17b, YSN+17] have been shown to be KR-insecure, i.e., these schemes have large (close to one) KR advantage due to various key-recovery attacks [SRM15, XSTF17, SZ17, SRZ18, YMSR17a, SLM+17a, SLR+19, ZJK17, Che18, SS19, YTS19]. Per our Theorem 1, if a DH scheme is KR-insecure, then it is also FR-insecure, i.e., large FR advantage. The few (recent) schemes [RMKS18, KAHS19] that have not been shown to be broken were designed specifically to thwart SAT attack. In order to prove FR-security, we need to show that a DH scheme thwarts *all* FR-attacks — not specific KR-attacks. In the next section, we will construct an FR-secure DH scheme.

7 A Framework for Designing Secure DH Schemes

In the FR-security experiment as well as in $\widetilde{\mathbf{FR}}$, the adversary's guess space is the set $\mathcal{F} \subseteq \mathsf{Func}(n, m)$ before it gets full access to the opaque circuit. As the adversary learns new information via unrestricted access to the opaque circuit C_L and oracle access to F, it can prune its guess space to a smaller set $\mathcal{M}_1 \subseteq \mathcal{F}$ of functions. Let $\mathcal{D}_1$ be the probability distribution of $\mathcal{M}_1$. Then, the probability that B wins $\widetilde{\mathbf{FR}}$ will be upperbounded by $2^{-H_\infty(\mathcal{D}_1)}$, where $H_\infty(\mathcal{D}_1)$ is the min-entropy of $\mathcal{D}_1$. In this section, we will describe a framework that an IP author can use to build a DH scheme (with formal descriptions of *Hide* and *Restore*) such that it can concretely define the distribution $\mathcal{D}_1$, and find concrete FR-security guarantees of its construction.

Abstractly, the opaque chip $\mathbb{C}_L$ can be viewed as a circuit that encodes a set $\mathcal{R}_L \subseteq \mathsf{Func}(n, m)$ of n-bit to m-bit functions (not necessarily distinct) that are selected by *Hide* (either implicitly or explicitly), and each key K in set $\mathcal{K} = \{0,1\}^{k_o(\theta)}$ is associated with some function in $\mathcal{R}_L$. Minimally, correctness will require that the hiding key K_O is associated with the true function F. We define a chaff function $H \in \mathsf{Func}(n, m)$ as one that is not functionally equivalent to F; we refer to the inputs on which H differs (in output) from F as "distinguishing" inputs.

In our framework, we insist that the set $\mathcal{R}_L$ be "extractable" from the opaque circuit, i.e., given C_L and the description of *Hide*, the IP author knows the full-functionality of each function in $\mathcal{R}_L$. This is a reasonable assumption as during the design of circuit, the functionality of a circuit is fixed first as part of the system specification and architectural design, and then the topology of the circuit is decided. In RLL [RKM08] and its variants [RPSK12a, RPSK12b, DBDN+14, KAHS19, RMKS18], the functionality of the chaff functions is *not* decided during the design of *Hide*. Rather, the chaff functions are an artifact of random structural modifications — the positions of key gates in the original circuit. So, for a non-pathological circuit, i.e., circuits with sufficiently complex functionality and reasonably large domain, it is almost impossible to extract the functionality of the chaff functions. Consequently, it is difficult to evaluate the FR security of such schemes in our framework. As mentioned earlier, RLL and its variants (designed before SAT attack) as well as SAT-attack-resistant schemes [XS16, YMRS16, YSS+17, SLM+17b, YSN+17] (designed post

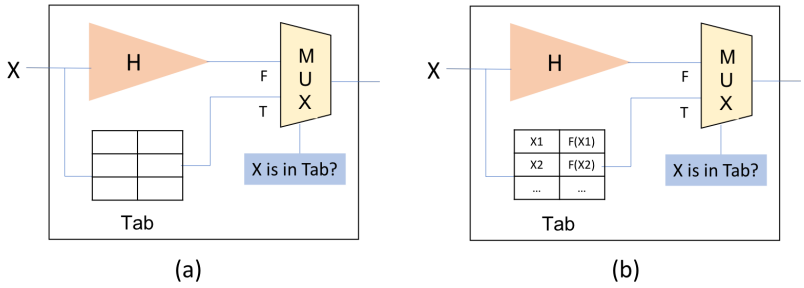

Figure 5: a) An opaque circuit generated by any OneChaff scheme. b) An honestly-fabricated-and-restored chip. Adversary gets unrestricted access to (a) and oracle access to (b).

SAT attack) are FR-insecure due to various key-recovery attacks [SRM15, XSTF17, SZ17, SRZ18, YMSR17a, SLM+17a, SLR+19, ZJK17, Che18, SS19, YTS19].

7.1 OneChaff: a family of DH schemes

We present a family of DH schemes called *OneChaff* with the following common features:

- the design-hiding algorithm returns an opaque circuit that contains a single *chaff* H with $\Delta \in \mathbb{N}$ distinguishing inputs and an uninitialized lookup table (see Figure 5(a));
- the hiding key encodes the set of distinguishing inputs and the correct value of the hidden function on the distinguishing inputs;
- the design-restoring algorithm, in an honest run, initializes the lookup table with the hiding key (see Figure 5(b)).

Notice that the description of OneChaff leaves H unspecified; specific OneChaff schemes will define H explicitly.

From a high level, OneChaff captures almost all SAT-attack-resistant DH schemes [XS16, YMRS16, YSS+17, YSN+17] — all of these schemes encode a single function (that is derived from the hidden function) and a "restore" mechanism. While SFLL-flex [YSN+17] uses a lookup table in the opaque circuit to restore the intended functionality of the hidden function, the remaining schemes use the following hamming-distance-based predicate: for some positive integer $h < 2^n$, is the hamming distance between the key input and primary input h? Except SFLL-HD [YSN+17], $h = 0$ in remaining schemes.

Several attacks [SS19, YMSR17a, YTS19, YMSR17b] have shown that K_O can be efficiently recovered by exploiting algorithmic weaknesses in the hamming-distance-based restore mechanisms. In the following sections, we will describe a OneChaff scheme called *OneChaff*$_{\text{hd}}$ and *prove* that it is FR-secure in the honest-but-curious setting for a broad class of functions. Let $\theta = (n, m, \Delta)$, where $n, m > 0$, and $\Delta \in [2^n - 1]$.

Description of *Hide*. Given F, the *Hide* algorithm in *OneChaff*$_{\text{hd}}$ selects F as the anchor function and samples a single chaff uniformly from its Δ-neighborhood, i.e., $H \leftarrow\!\!\$\ N_\Delta(F)$. Equivalently, the chaff H is initialized to F and then, on a random subset $\mathcal{I}^{\neq}(F, H) = \{X_1, X_2, \ldots, X_\Delta\}$ of the domain of F, the value of $H(X_i) \leftarrow\!\!\$\ \{0,1\}^m \setminus \{F(X_i)\}$; when $m = 1$, $H(X_i) \leftarrow \neg F(X_i)$.

Formally, the opaque circuit C_L returned by *Hide* computes the function $L\colon (\{0,1\}^n \cup \{\lightning\})^\Delta \times (\{0,1\}^m \cup \{\lightning\})^\Delta \times \{0,1\}^n \to \{0,1\}^m$ defined by

$$L\left(((U_1, V_1), (U_2, V_2), \ldots, (U_\Delta, V_\Delta)), X\right) = \begin{cases} V_i & \text{if } \exists U_i = X \\ H(X) & \text{otherwise} \end{cases}$$

where the distinguished (non-string) symbol $\lightning$ is understood to mean "uninitialized". We implicitly assume that all string-valued U_i are distinct so that L is well defined, and that if any $U_i = \lightning$, then all $U_i = \lightning$. Thus, L computes H and a family of functions $\mathcal{R}_L = N_\Delta(H)$, one of which is F.

In practice, we envision C_L to be a circuit that encodes H and an uninitialized table $\widetilde{\mathrm{Tab}}$, whose gate-and-wire representation allows for a subcircuit C_H (computing H) to be easily parsed out; overloading the $\langle\cdot\rangle$ notation, we will sometimes write $C_L = \langle H, \widetilde{\mathrm{Tab}}\rangle$ to reflect this. This way, one can evaluate C_H given C_L and, by loading $\widetilde{\mathrm{Tab}}$ in C_L with a list of Δ pairs in $\{0,1\}^n \times \{0,1\}^m$, any of the functions in $N_\Delta(H)$.

Note that *Hide* is abstracting the process by which an IP author goes from the "idea" (i.e., informal specification) of a functionality F to an opaque circuit described in some format that is suitable for fabrication (e.g., a GDSII file). This process typically involves writing a program for the desired functionality in some high-level language like Verilog, and then converting the program to a circuit description by an EDA tool. This process can be augmented to sample Δ random domain points, and incorporate these to yield a program (hence a circuit) for the chaff function H; the size of $\widetilde{\mathrm{Tab}}$ will be $n\Delta$. Thus, the efficiency of *Hide* in *OneChaff*$_{\mathrm{hd}}$ should be roughly same as that of the original process, at least when $\Delta \ll 2^n$.

Description of *Restore*. The *Restore* algorithm parses the input (design) parameters θ to determine n, m, Δ. It then expects a key $K = \langle (U_1, V_1), \ldots, (U_\Delta, V_\Delta)\rangle$ where each $(U_i, V_i) \in \{0,1\}^n \times \{0,1\}^m$, and a chip $\mathbb{C}_L$ that was fabricated from a circuit $C_L = \langle H, \widetilde{\mathrm{Tab}}\rangle$. When these expectations are met, *Restore* loads the pairs (U_i, V_i) into the uninitialized lookup table. (When not, it returns $\perp$.) We write $\mathbb{C}_L \uparrow K$ for this, the notation suggesting that K is "uploaded" into $\mathbb{C}_L$. Thus, when provided with an honestly fabricated chip from the opaque circuit C_L returned by $Hide(F, \theta)$, and the corresponding key K_O, the restored chip $\mathbb{C}_L \uparrow K_O$ properly computes the hidden function F. Note that, we abstract away the physical mechanism by which uploading the (U_i, V_i) is implemented, and assume that if this mechanism is able to detect upload failure, *Restore* returns $\perp$.

7.2 Efficiency of OneChaff$_{\mathrm{hd}}$

***Hide* is efficient.** The opaque circuit C_L that *Hide* generates consists of a chaff function H — that differs from F on Δ random inputs, and an uninitialized lookup table $\widetilde{\mathrm{Tab}}$. Since the IP author knows the full functionality of F, it can follow a two-step approach in order to create H from F. In the first step, it creates a set of Δ random n-bit inputs and finds the value of F on those inputs. Next, it sets the value of H on the Δ inputs X_i to $\neg F(X_i)$; on the remaining inputs H and F have identical values. Hence, runtime of *Hide* to create H will be $O(n\Delta) + T(H)$, where $T(H)$ is the time needed to create the circuit for H. Note that the IP author would have anyways spent $T(F)$ time to build the circuit for F in the honest setting. So, effectively, the additional runtime of *Hide* in *OneChaff*$_{\mathrm{hd}}$ is $O(n\Delta) + T(H) - T(F)$. (Time to create K_O will also be $O(n\Delta)$ as $|K_O| = n\Delta$.) When $\Delta \ll 2^n$, $T(H)$ will be roughly same as $T(F)$. Moreover, such an implementation does not leak any additional information about F apart from H when the adversary gets full view of the opaque circuit.

***Restore* is efficient.** Since *Restore* involves uploading the $n\Delta$-bits hiding key K_O to $\widetilde{\mathrm{Tab}}$, it's runtime will be $O(n\Delta)$.

7.3 FR security of OneChaff$_{\mathrm{hd}}$

In $\widetilde{\mathbf{FR}}$, given unrestricted access to $C_L = \langle H, \widetilde{\mathrm{Tab}}\rangle$, the IP author knows that the foundry can (at best) extract the set $\mathcal{R}_L = N_\Delta(H)$ and compute $\mathcal{M}_0 = \mathcal{F} \cap N_\Delta(H)$ to reduce the size of its original guess space $\mathcal{F}$. Using queries to TRUE, the foundry can further reduce the guess space to $\mathcal{M}_1 \subseteq \mathcal{M}_0$ by removing functions in $\mathcal{M}_0$ whose distinguishing-input sets contain query inputs.

In practice, the IP author will not know the set $\mathcal{F}$ because that captures the foundry's initial guess at the set of possible functions F that may be what the IP author intends to hide. (Recall that the FR-notion samples $F \leftarrow\!\!\$\ \mathcal{F}$, suggesting that the foundry's initial set is the correct set.) More plausibly, the author may assume that the foundry's initial guess is based upon knowledge of likely "properties" of F. These properties may be gleaned from discussions with the IP author, statements in fabrication contracts, historical and market information, etc. The IP author may use these assumed-known properties in its description of *Hide*, *Restore*, or into the parameters θ. In particular, it can use these assume-known properties to compute its estimates of the sets $\mathcal{M}_0$ and $\mathcal{M}_1$ that can help the foundry reduce the guess space. We will use these observations in our upcoming analysis.

Scoping the set $\mathcal{F}$. In what follows, we will focus on the case that $m = 1$, i.e., $\mathcal{F} \subset \mathsf{Func}(n, 1)$. We focus on Boolean functions for a few reasons. First, even if the circuit representation of the hidden function has $n' \geq n$ bits of input and $m > 1$ bits of output, we can take as a first consideration whether or not one can securely hide the transitive fan-in cone (TFC) of any particular output bit. For a collection of output bits that have disjoint transitive fan-in cones, one can consider hiding these in parallel.

Second, focusing on Boolean functions makes the analysis less complicated. In particular, for all distinguishing inputs $U_i \in \mathcal{I}^{\neq}(F, H)$, $V_i = \neg F(U_i)$; otherwise $V_i \leftarrow\!\!\$\ U_m \setminus F(U_i)$. Note that when $m > 1$, the TFCs of different output bits may not be disjoint. In such a case, the adversary can potentially use information it learns about one TFC to learn about the functionality of a different TFC.

Third, barring SFLL-flex [YSN+17], in the remaining SAT-attack-resistant DH schemes (Anti-SAT [XS16], SARLock [YMRS16], TTLock [YSS+17] and SFLL-HD [YSN+17]), F is Boolean.

Simple functions cannot be hidden. We borrow the definition of "simple" functions from learning theory; simple functions are those that can be efficiently learned via a reasonably small number of input-output values of F. Now, if $\mathcal{F}$ consists predominantly of simple functions, then *no* DH scheme will be secure[4] in hiding functions sampled from $\mathcal{F}$. For example, functions that have a small domain can be learned by brute force. In addition, results from computational learning theory tell us that functions whose decision-tree representation have small depth/size can be learned via the Kushilevitz-Mansour algorithm [KM93]. More generally, methods exist to (approximately) learn Boolean functions whose Fourier spectra are sparse (e.g., dominated by relatively few Fourier coefficients) [GOS+11].

To loosely capture a measure of the density of "simple" Boolean functions within a given set $\mathcal{F}$, we give the following definition.

Definition 3. ["Simple" functions in $\mathcal{F}$.] Let $\mathcal{F} \subseteq \mathsf{Func}(n, 1)$ be a set of Boolean functions. Let $t, q \geq 0$ be integers, and let $\delta \in [0, 1]$ be a real number. Let $\mathcal{F}_{t,q,\delta} \subseteq \mathcal{F}$ be a subset such that the following holds: $\exists$ a q-query, t-time adversary A such that, $\forall f \in \mathcal{F}_{t,q,\delta}$, when $g \leftarrow\!\!\$\ A^{f(\cdot)}$ we have $\Pr[\, x \leftarrow\!\!\$\ \{0,1\}^n : g(x) = f(x)\,] \geq \delta$. Furthermore, define $\epsilon_{t,q,\delta} = |\mathcal{F}_{t,q,\delta}|/|\mathcal{F}|$. In particular, $\epsilon_{t,q,1}$ is the fraction of $\mathcal{F}$ that can be exactly learned (by some A) with q input-output values and time-complexity t. $\diamond$

Our security bounds for *OneChaff*$_{\mathrm{hd}}$ will reflect the term $\epsilon_{t,q,1}$, although we stress that corresponding bounds for any DH scheme would also have to reflect this term (perhaps not explicitly) because any "simple" function will not be hideable.

In general, specifying a set $\mathcal{F}$ for which $\epsilon_{t,q,1}$ is small enough for practically meaningful security statements (for reasonable t, q) is challenging, as this would require results of the following kind: There exist no adversary that can learn any function in $\mathcal{F} \setminus \mathcal{F}_{t,q,1}$ with q queries in time t. We are not aware of any such results. Also, note that it is not sufficient for $\mathcal{F}$ to be large (although it

[4] Shamsi et al. [SPJ19] made a similar observation and gave impossibility results on logic locking when $\mathcal{F}$ consists of entirely simple functions. Our security experiments are more generic as they allow $\mathcal{F}$ to contain functions of varying degree of "simplicity".

is necessary to avoid simply guessing F), as one can specify large sets of functions with sparse Fourier spectra.

We conjecture that if $\mathcal{F}$ is sufficiently "unstructured", then $\epsilon_{t,q,1}$ will be small enough to not dominate the FR-security bounds we will prove, for practically reasonable t, q. For example, a random Boolean function lacks the highly concentrated spectral structure — the number of non-zero Fourier coefficients is $(1 - o(1))2^n$ [DW08]— that leads to efficient learnability from input-output pairs.

Under the above conjecture, we will focus on $\mathcal{F} = \mathcal{F}_{[h_0,h_1]} \subseteq \mathsf{Func}(n, 1)$ that consists of all Boolean functions with hamming weight at least h_0 and at most h_1. Recall that the set $\mathcal{F}$ in our FR experiment is meant to reflect the adversary's a priori "knowledge" about the function hidden in the opaque circuit, and that our FR-notion samples F uniformly from $\mathcal{F}$. At the extremes, setting $h_0 = 0, h_1 = 2^n$ considers the case that the adversary has no a priori knowledge; setting $h_0 = h_1$ considers an adversary that knows exactly the hamming weight of the hidden function F. In our analysis of *OneChaff*$_{\mathrm{hd}}$, we will conservatively assume the latter. Note that a random Boolean function will have $h = 2^{n-1}$ on average. Hence, if the IP-author's circuit has hamming weight close to 2^{n-1}, then it knows that $\epsilon_{t,q,1}$ will be small and it will be very unlikely for the foundry to learn F by just using queries to TRUE.

A practical warm-up: FR security with $\mathbf{q = 0}$. Chips that are used in critical infrastructure (e.g., military devices) will most likely require considerable effort to obtain. In this case, we can assume that the foundry cannot get access to honestly restored chips, and thereby learn the true input-output behavior of F on inputs of its choice. Thus, we begin our analysis of *OneChaff*$_{\mathrm{hd}}$ for $\mathcal{F}_{[h,h]}$ in the $q = 0$ case.

Given Lemma 1, recall that we were left to find an upper bound on $\Pr\left[\widetilde{\mathbf{FR}}(B) = 1\right]$, where B has some time-complexity t and asks q queries to its TRUE oracle. Let us fix $q = 0$, and consider an arbitrary adversary B_0 with these resource bounds.

Recalling the notation that $\mathcal{R}_L$ is the set of functions realizable by the opaque circuit $L = \langle H, \widetilde{\mathrm{Tab}}\rangle$, we have $\mathcal{R}_L = N_\Delta(H)$; $F \in \mathcal{R}_L$. The adversary B_0 knows a priori that $F \in \mathcal{F}_{[h,h]}$. Hence, for all $\hat{F} \in \mathcal{R}_L$, $\mathrm{hw}(\hat{F}) \in [\mathrm{hw}(H) - \Delta, \mathrm{hw}(H) + \Delta]$. To get an intuition of the claim on $\mathrm{hw}(\hat{F})$, consider two cases: $\mathrm{hw}(H) = 0$ and $\mathrm{hw}(H) = 2^n$. When $\mathrm{hw}(H) = 0$, $\mathrm{hw}(\hat{F}) = \mathrm{hw}(H) + \Delta = \Delta$; when $\mathrm{hw}(H) = 2^n$, $\mathrm{hw}(\hat{F}) = \mathrm{hw}(H) - \Delta = 2^n - \Delta$.

While attacking *OneChaff*$_{\mathrm{hd}}$, the optimal strategy for B_0 is to return the circuit implementation of the most-likely function in $\mathcal{F}_{[h,h]} \cap \mathcal{R}_L$. (We will use $\mathcal{F}^j_{[h,h]}$, $\mathcal{R}^j_L$ to denote the subset of functions in $\mathcal{F}_{[h,h]}$ and $\mathcal{R}_L$, respectively, that have correct values on some $0 \le j \le q$ inputs.) But as sampling is done in a uniform and independent fashion in $\widetilde{\mathbf{FR}}$, all functions in $\mathcal{M}_0 = \left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)$ are equally likely. Then, $\Pr\left[\widetilde{\mathbf{FR}}(B_0) = 1\right] = |\mathcal{M}_0|^{-1}$. We will proceed to give a lowerbound on $|\mathcal{M}_0|$ in the $q = 0$ case.

Claim 1. *Let $\Delta < h$ and $\mathrm{hw}(H) = h + \delta$, where $\delta < (2^{n-1} - h)$. Then,*

$$|\mathcal{M}_0| = \left|\left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)\right| = \binom{h+\delta}{(\Delta+\delta)/2}\binom{2^n - h - \delta}{(\Delta-\delta)/2}.$$

♦

Proof. Recall that $\mathcal{X}_i(H)$ denotes the set of all inputs for which H's value is i; $\mathrm{hw}(H) = \mathcal{X}_1(H)$. In Figure 6, we show the representation of F and H using their respective $\mathcal{X}_i$. Any Boolean function H can be described using sets $\mathcal{X}_0(H)$ and $\mathcal{X}_1(H)$.

For $i \in \{0, 1\}$ and for any $\hat{F} \in \mathcal{R}_L$, let $S_i(H, \hat{F}) = S_i = \mathcal{I}^{\neq}(H, \hat{F}) \cap \mathcal{X}_i(H)$. By construction: S_1, S_0 are disjoint, and $S_i \subset \mathcal{X}_{1-i}(\hat{F})$ as $\hat{F}(X) = \neg H(X)$ for all $X \in \mathcal{I}^{\neq}(H, \hat{F})$.

We can build $\mathcal{X}_1(\hat{F})$ using the sets $\mathcal{X}_1(H)$ and S_i in three steps. First, initialize $\mathcal{X}_1(\hat{F}) = \mathcal{X}_1(H)$. Then, remove all elements in set S_1 from $\mathcal{X}_1(\hat{F})$, and in the final step, add all elements of set S_0, i.e., $\mathcal{X}_1(\hat{F}) = (\mathcal{X}_1(H) \setminus S_1) \cup S_0$. ($\mathcal{X}_0(\hat{F}) = (\mathcal{X}_1(H) \setminus S_0) \cup S_1$.)

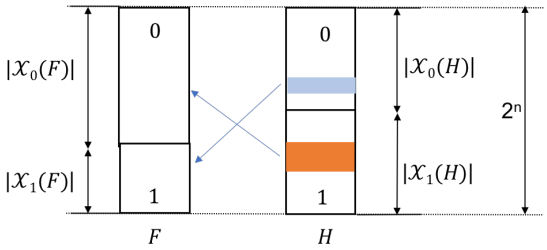

Figure 6: Representation of F and H using $\mathcal{X}_i(\cdot)$. The orange (resp. blue) box denotes the distinguishing inputs that fall in set $\mathcal{X}_1(H)$ (resp. $\mathcal{X}_0(H)$). By construction of *OneChaff*$_{\text{hd}}$, the orange (resp. blue) box belongs to $\mathcal{X}_0(F)$ (resp. $\mathcal{X}_1(F)$).

By construction, the hamming weight of all functions in $\mathcal{R}_L$ will fall in the window $[\text{hw}(H) - \Delta, \text{hw}(H) + \Delta]$, i.e., $\text{hw}(\hat{F}) \in [\text{hw}(H) - \Delta, \text{hw}(H) + \Delta]$ for all $\hat{F} \in \mathcal{R}_L$. Since $\mathcal{F}^0_{[h,h]}$ comprises of functions with hamming weight h only, we can construct $\left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)$ by only keeping functions $\hat{F} \in \mathcal{R}_L$ whose $\text{hw}(\hat{F}) = h$. That is, for each $\hat{F} \in \left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)$, the hamming weight of $\hat{F}$ will satisfy: $|\mathcal{X}_1(H)| - |S_1(H, \hat{F})| + |S_0(H, \hat{F})| = h$. Also, by construction of *OneChaff*$_{\text{hd}}$, the total number of distinguishing inputs will be Δ. Hence, $|S_1(H, \hat{F})| + |S_0(H, \hat{F})| = \Delta$. Solving these two equations, and using $\text{hw}(H) = h + \delta$ gives us:

$$|S_1(H, \hat{F})| = (\Delta + \delta)/2; |S_0(H, \hat{F})| = (\Delta - \delta)/2.$$

These equations tell us that given H, each function $\hat{F}$ in the set $\mathcal{F}_{[h,h]} \cap \mathcal{R}_L$ has $(\Delta + \delta)/2$ distinguishing inputs in $\mathcal{X}_1(H)$ and $(\Delta - \delta)/2$ distinguishing inputs in set $\mathcal{X}_0(H)$. Total number of such functions will be $\binom{|\mathcal{X}_1(H)|}{(\Delta+\delta)/2} \times \binom{|\mathcal{X}_0(H)|}{(\Delta-\delta)/2}$, where $\delta = \text{hw}(H) - h$. Substituting $|\mathcal{X}_1(H)| = \text{hw}(H)$ and $|\mathcal{X}_0(H)| = 2^n - \text{hw}(H)$ in the previous equation gives us the final bound in the claim. ■

When $\Delta < h$ and $\text{hw}(H) < 2^{n-1}$, we found (in the proof of Claim 1) that *more than half* $(\Delta/2 + \delta/2)$ of the distinguishing inputs will belong to $\mathcal{X}_1(H)$. Hence, we can assume, without any loss in the FR advantage of the adversary, that the adversary will make all of its queries from $\mathcal{X}_1(H)$ until it finds all $(\Delta + \delta)/2$ distinguishing inputs in $\mathcal{X}_1(H)$. (Later, we will show that when $q < \min(\text{hw}(H)/4, \Delta^2/64 \ln n)$, the adversary can find *at most* half of the distinguishing inputs in $\mathcal{X}_1(H)$, without loss.)

The lower bound on $|\mathcal{M}_0|$ in Claim 1, a function of $\text{hw}(H)$ as $\delta = (h - \text{hw}(H))$, gives us an upper bound on the FR advantage of adversary that attacks *OneChaff*$_{\text{hd}}$ without making any queries to any of the three oracles. But, $\text{hw}(H)$ is not a parameter of the FR-security experiment. In the next claim, we give a lower bound on the hamming weight of H using the Hoeffding lemma; the bound is a function of h, Δ and n and all three are parameters of the FR experiment.

Claim 2. *Let* $\Delta < h$ *and* $0 < (\Delta + h) < 2^{n-1}$. *Then, with probability at least* $(1 - 1/\Delta^2)$, $hw(H) \geq h + \Delta(1 - h/2^{n-1})$. ♦

Proof. We will use the sets $\mathcal{X}_i(F)$ and $S_i(F, H) = S_i = \mathcal{I}^{\neq}(F, H) \cap \mathcal{X}_i(F)$, where $i \in \{0, 1\}$, to define $\mathcal{X}_i(H)$ similar to how we used sets $\mathcal{X}_i(H)$ and $\mathcal{I}^{\neq}(F, H) \cap \mathcal{X}_i(H)$ to define $\mathcal{X}_i(F)$ in the proof of Claim 1. Note that the set $\mathcal{X}_i(F)$ is used as the base function in this proof; in Claim 1, we used $\mathcal{X}_i(H)$ as the base function. Following the same three steps as in the proof of Claim 1, we will get $\mathcal{X}_1(H) = (\mathcal{X}_1(F) \setminus S_1) \cup S_0$ and $|\mathcal{X}_1(H)| = |\mathcal{X}_1(F)| - |S_1| + |S_0|$.

Next, we need bounds on S_i's. Before that, observe that the set $\mathcal{I}^{\neq}(F, H) = \{X_1, X_2, \ldots, X_\Delta\}$ is a random variable that follows a uniform distribution, i.e., $\mathcal{I}^{\neq}(F, H) \xleftarrow{\$} (U_n)^\Delta$. Since $\text{hw}(F) = h$, $\Pr[\, X \in \mathcal{X}_1(F)\,] = h/2^n$, and $\Pr[\, X \in \mathcal{X}_0(F)\,] = 1 - h/2^n$, where $X \xleftarrow{\$} \mathcal{I}^{\neq}(F, H)$.

Now, the size of S_i, i.e., $|S_i|$ is a random variable that is binomially distributed, with $\mathbb{E}[|S_1|] = \mu_1 = \Delta(h/2^n)$ and $\mathbb{E}[|S_0|] = \mu_0 = \Delta(1 - h/2^n)$. By a standard Hoeffding bound, we have, for all $\epsilon > 0$, $\Pr[\,|S_i| \geq \mu_i + \Delta\epsilon\,] \leq \exp(-2\Delta\epsilon^2)$. Setting $\epsilon = \sqrt{(\ln\Delta)/\Delta}$, we get with probability at most $1/\Delta^2$, $\Pr\left[\,|S_i| \geq \mu_i + \sqrt{\Delta \ln \Delta}\,\right]$. We can write this otherwise as: $\Pr\left[\,|S_i| < \mu_i + \sqrt{\Delta \ln \Delta}\,\right]$ with probability at least $1 - 1/\Delta^2$. We will assume that the bounds on S_i are tight; we will reflect the uncertainty in the claim/theorem statements. Using the bounds on S_i in $|\mathcal{X}_1(H)| = |\mathcal{X}_1(F)| - |S_1| + |S_0|$, we get

$$\begin{aligned} |\mathcal{X}_1(H)| &= |\mathcal{X}_1(F)| - (\mu_1 + \sqrt{\Delta \ln \Delta}) + (\mu_0 + \sqrt{\Delta \ln \Delta}) \\ &= h - \Delta(h/2^n) + \Delta(1 - h/2^n) = h + \Delta - (\Delta h)/2^{n-1}, \end{aligned}$$

with probability at least $(1 - 1/\Delta^2)$. ■

We will assume that the bound on $\mathrm{hw}(H)$ is tight and we will reflect the uncertainty in the claim/theorem statements.

Now, we are ready to give the upper bound on the FR advantage of adversary A in the zero-query setting. We use the standard relation $\binom{p}{r} \geq (p/r)^r$ in the bound in Claim 1 to make the final bound in Theorem 2 easier to interpret. We also use the weaker bound $\mathrm{hw}(H) > h$ (instead of the tighter bound in Claim 2) to make the final bound even more interpretable. Note that practical *OneChaff*$_{\mathrm{hd}}$ schemes will have $\Delta \ll 2^{n-1}$ as the run time of *Hide* as well as *Restore* will be linear in Δ. Thus, the loss in using $\mathrm{hw}(H) > h$ instead of $\mathrm{hw}(H) \geq h + \Delta(1 - h/2^{(n-1)})$ will not be significantly large.

Theorem 2. *Fix $m = 1$ and integers $n, h, \Delta > 0$ such that $\Delta < h$, and $\Delta + h < 2^{(n-1)}$. When an honest-but-curious adversary A attacks the FR security of $\Pi =$ OneChaff$_{\mathrm{hd}}$ without making any queries to any oracle, then it achieves*

$$\mathbf{Adv}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}_{[h,h]}, A) \leq \left(\frac{\Delta^2}{2^n h}\right)^{\Delta/2} + \epsilon_{t,0,1},$$

with probability at least $(1 - 1/\Delta^2)$. ♦

Proof. From Lemma 1, Claims 1 and 2, we get

$$\mathbf{Adv}^{\mathrm{FR}}_{(\Pi,\theta),\mathit{Fab}}(\mathcal{F}_{[h,h]}, A) \leq \left(\binom{h+\delta}{(\Delta+\delta)/2}\binom{2^n - h - \delta}{(\Delta-\delta)/2}\right)^{-1} + \epsilon_{t,q,1}.$$

Since $\delta = \Delta(1 - h/2^{n-1})$, we have: $\Delta - \delta < \Delta$, $\Delta + \delta < 2\Delta$, and $h + \delta \leq h + \Delta < 2^{n-1}$. Using these observations, and $\binom{p}{r} \geq (p/r)^r$, we get,

$$\begin{aligned} &\left(\binom{h+\delta}{(\Delta+\delta)/2}\binom{2^n - h - \delta}{(\Delta-\delta)/2}\right)^{-1} \\ &\leq \left(\frac{h+\delta}{(\Delta+\delta)/2}\right)^{-((\Delta+\delta)/2)} \left(\frac{2^n - h - \delta}{(\Delta-\delta)/2}\right)^{-((\Delta-\delta)/2)} \\ &\leq \left(\frac{(\Delta+\delta)/2}{h+\delta}\right)^{(\Delta+\delta)/2} \left(\frac{(\Delta-\delta)/2}{2^n - h - \delta}\right)^{(\Delta-\delta)/2} \\ &\leq \left(\frac{\Delta}{h}\right)^{\Delta/2} \left(\frac{\Delta/2}{2^{n-1}}\right)^{\Delta/2} \leq \left(\frac{\Delta^2}{2^n \cdot h}\right)^{\Delta/2} \end{aligned}$$

■

In the next section, we will see that when we account for the adversary's access to the oracles, more specifically, the TRUE oracle, the upper bound on the FR advantage will be $\left(\frac{\Delta^2}{2^n \cdot (h-q)}\right)^{\Delta/4} + \epsilon_{t,q,1}$. Observe that the queries to TRUE increase the FR advantage of an adversary significantly (compared to the zero-query setting) as the exponent $\Delta/2$ decreases to $\Delta/4$; the denominator also is smaller compared to the zero-query setting. Since the first term is very similar to the bound in Theorem 3, we defer the unpacking of Theorem 2 to the next section.

FR analysis of OneChaff$_{\mathrm{hd}}$ with $\mathbf{q > 0}$. The ability to learn true input-output pairs, via queries to TRUE, provide a way for the adversary to verify guesses at portions of the key K_O. Recall that the key encodes $(X_1, F(X_1)), \ldots, (X_\Delta, F(X_\Delta))$ for $X_i \in \mathcal{I}^{\neq}(F, H)$, and the opaque circuit C_L allows for local computation of $H(X)$. Thus, as a first step in analyzing the FR security of *OneChaff*$_{\mathrm{hd}}$ in the $q > 0$ case, we derive a bound on the number of points in $\mathcal{I}^{\neq}(F, H)$ that an adversary uncovers in its q queries to the TRUE oracle.

Let $\mathcal{Q}_j = \{x_1, x_2, \ldots, x_j\}$ be the the first j queries to TRUE, and let random variable $\mathcal{Q}_j^{\mathrm{key}} = \mathcal{Q}_j \cap \mathcal{I}^{\neq}(F, H)$ denotes the queries in $\mathcal{Q}_j$ that uncover a portion of the distinguishing inputs. Observe that $|\mathcal{Q}_0^{\mathrm{key}}| = 0$, and for $j > 0$ the value of $|\mathcal{Q}_j^{\mathrm{key}}|$ depends only upon $|\mathcal{Q}_{j-1}^{\mathrm{key}}|$ and the query x_j; in particular that $|\mathcal{Q}_j^{\mathrm{key}}| = |\mathcal{Q}_{j-1}^{\mathrm{key}}| + 1$ if $x_j \in \mathcal{I}^{\neq}(F, H)$, and $|\mathcal{Q}_j^{\mathrm{key}}| = |\mathcal{Q}_{j-1}^{\mathrm{key}}|$ if not.

Let I_j be the indicator random variable indicating that the event $x_j \in \mathcal{Q}_j^{\mathrm{key}}$ occurs. We claim that $\Pr[\, I_j = 1\,] = (\Delta - |\mathcal{Q}_{j-1}^{\mathrm{key}}|)/\ (\mathrm{hw}(H) - (j-1))$. To see this, observe that the number of uncovered points in $\mathcal{I}^{\neq}(F, H)$ is precisely $(\Delta - |\mathcal{Q}_{j-1}^{\mathrm{key}}|)$ and, given how those points were sampled, any of the remaining, unqueried points in $\mathcal{X}_1(H)$ are equally likely to be in $\mathcal{I}^{\neq}(F, H)$. Given this, we can prove the following lemma.

Lemma 2. *Let $\Delta < h$ and $(\Delta + h) < 2^{n-1}$. Then, we have $\mathbb{E}\left[|\mathcal{Q}_q^{\mathrm{key}}|\right] = \frac{q\Delta}{hw(H)}$, and, $|\mathcal{Q}_q^{\mathrm{key}}| < \mathbb{E}\left[|\mathcal{Q}_q^{\mathrm{key}}|\right] + \sqrt{4q \ln \Delta}$ with probability at least $1 - (2/\Delta^2)$.* ♦

Proof. Let $j \leq q$ and $Z_j = \mathcal{Q}_j^{\mathrm{key}}$. We prove this claim by induction. We have already identified the base case of $\mathbb{E}[Z_0] = Z_0 = 0$ and $\mathbb{E}[Z_1] = \Delta/\mathrm{hw}(H)$. In the general case, we will compute $\mathbb{E}[Z_j] = \mathbb{E}[\mathbb{E}[Z_j \mid Z_{j-1}]]$. First,

$$\begin{aligned}\mathbb{E}\left[Z_j \mid Z_{j-1}\right] = &= (Z_{j-1}+1)\frac{\Delta - Z_{j-1}}{\mathrm{hw}(H)-(j-1)} + (Z_{j-1})\left(1 - \frac{\Delta - Z_{j-1}}{\mathrm{hw}(H)-(j-1)}\right)\\ &= Z_{j-1} + \left(\frac{\Delta - Z_{j-1}}{\mathrm{hw}(H)-(j-1)}\right)\end{aligned}$$

Now taking the expected value of both sides, we get the expression

$$\begin{aligned}\mathbb{E}\left[Z_j\right] &= \mathbb{E}\left[Z_{j-1}\right] + \frac{1}{\mathrm{hw}(H)-(j-1)}\left(\Delta - \mathbb{E}\left[Z_{j-1}\right]\right)\\ &= \left(1 - \frac{1}{\mathrm{hw}(H)-(j-1)}\right)\mathbb{E}\left[Z_{j-1}\right] + \frac{\Delta}{\mathrm{hw}(H)-(j-1)}\\ &= \left(1 - \frac{1}{\mathrm{hw}(H)-(j-1)}\right)\frac{(j-1)\Delta}{\mathrm{hw}(H)} + \frac{\Delta}{\mathrm{hw}(H)-(j-1)}\\ &= \left(\frac{\mathrm{hw}(H)-j}{\mathrm{hw}(H)-(j-1)}\right)\frac{(j-1)\Delta}{\mathrm{hw}(H)} + \frac{\Delta}{\mathrm{hw}(H)-(j-1)}\\ &= \left(\frac{(j-1)\Delta}{\mathrm{hw}(H)-(j-1)}\right)\left(\frac{\mathrm{hw}(H)-j}{\mathrm{hw}(H)}\right) + \frac{\Delta}{\mathrm{hw}(H)-(j-1)}\\ &= \left(\frac{\Delta}{\mathrm{hw}(H)-(j-1)}\right)\left((j-1)\frac{\mathrm{hw}(H)-j}{\mathrm{hw}(H)} + 1\right)\\ &= \left(\frac{\Delta}{\mathrm{hw}(H)-(j-1)}\right)\left(\frac{(j-1)(\mathrm{hw}(H)-j) + \mathrm{hw}(H)}{\mathrm{hw}(H)}\right)\end{aligned}$$

$$= \left(\frac{\Delta}{\text{hw}(H)-(j-1)}\right)\left(\frac{j(\text{hw}(H))-(j-1)(j)}{\text{hw}(H)}\right)$$
$$= \left(\frac{j\Delta}{\text{hw}(H)-(j-1)}\right)\left(\frac{\text{hw}(H)-(j-1)}{\text{hw}(H)}\right)$$
$$= \frac{j\Delta}{\text{hw}(H)}$$

Thus, for $j=q$, $Z_q = \frac{q\Delta}{\text{hw}(H)}$.
We now show that Z_q is tightly concentrated around its expectation. Observe that $Z_j - Z_{j-1} \leq 1$ for all $1 \leq j \leq q$, so by the Azuma-Hoeffding inequality $\Pr[\, Z_q \geq (1+\epsilon)\mathbb{E}\,[Z_q]\,] \leq 2\exp\left(-\frac{(\epsilon\mathbb{E}[Z_q])^2}{2q}\right)$. If we set the upperbound to equal a parameter α and solve for ϵ, we obtain $\epsilon = \frac{\sqrt{2q\ln\left(\frac{2}{\alpha}\right)}}{\mathbb{E}[Z_q]}$. Pushing a bit further, we set $\alpha = 2/\Delta^2$ and arrive at the following result: $Z_q < \mathbb{E}\,[Z_q] + \sqrt{4q\ln\Delta}$ with probability at least $1-(2/\Delta^2)$. ■

Notice that when $q < \Delta^2/(256\ln\Delta)$, $\sqrt{4q\ln\Delta} < \Delta/8$; when $q < \text{hw}(H)/8$, $q\Delta/\text{hw}(H) < \Delta/8$. Thus, with probability $1-(2/\Delta^2)$, the adversary will uncover *no more than* quarter of the distinguishing inputs in $\mathcal{I}^{\neq}(F,H)$ when $q < \min(\text{hw}(H)/8, \Delta^2/(256ln\Delta))$.

The adversary can increase its FR advantage by removing functions from $\left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)$ that are not correct on any input in $\mathcal{Q}^{\text{key}}_q$. In fact, it can remove all functions that do not agree with F on any point in the query set $\mathcal{Q}_q$.

Extending Claim 1 to account for $\mathcal{Q}^{\text{key}}_q$, we give a lower bound on $|\mathcal{M}_1| = \left|\left(\mathcal{F}^q_{[h,h]} \cap \mathcal{R}^q_L\right)\right|$ in Claim 3. Since one wins against *OneChaff*$_{\text{hd}}$ if and only if the hidden function F is completely recovered, a lowerbound on $|\mathcal{M}_1|$ will give us an upperbound on $\Pr\left[\widetilde{\mathbf{FR}}(B)=1\right]$, and in turn (by Lemma 1) an upper bound on the FR advantage of an adversary A that attacks *OneChaff*$_{\text{hd}}$.

Claim 3. *Let $\Delta < h$ and hw$(H) = h+\delta$, where $\delta = \Delta(1-h/2^{n-1})$. Then,*

$$|\mathcal{M}_1| = \left|\left(\mathcal{F}^q_{[h,h]} \cap \mathcal{R}^q_L\right)\right| \geq \binom{(h+\delta-q)}{(\Delta+\delta)/2-|\mathcal{Q}^{\text{key}}_q|}\binom{2^n-h-\delta}{(\Delta-\delta)/2},$$

with probability at least $(1-2/\Delta^2)$. ♦

Proof. After q queries, the adversary learns $|\mathcal{Q}^{\text{key}}_q|$ distinguishing inputs. Let $\delta = \text{hw}(H) - h$. From the proof of Claim 1, we know that given H, each function $\hat{F}$ in the set $\mathcal{F}_{[h,h]} \cap \mathcal{R}_L$ has $(\Delta+\delta)/2$ distinguishing inputs in $\mathcal{X}_1(H)$ and $(\Delta-\delta)/2$ distinguishing inputs in set $\mathcal{X}_0(H)$. Since $\mathcal{Q}_q \subset \mathcal{X}_1(H)$ with high probability (from Lemma 2) and the adversary knows $|\mathcal{Q}^{\text{key}}_q|$ distinguishing inputs in set $\mathcal{X}_1(H)$, the total number of functions in $\mathcal{F}_{[h,h]} \cap \mathcal{R}_L$ that have values $F(X_i)$ on all inputs $X_i \in \mathcal{Q}_q$ will be $\binom{|\mathcal{X}_1(H)|-q}{(\Delta+\delta)/2-|\mathcal{Q}^{\text{key}}_q|}\binom{|\mathcal{X}_0(H)|}{(\Delta-\delta)/2}$. ■

Comparing the bounds in Claims 1 and 3, we can see that the adversary B can remove a large number — at least $\binom{2^n-h-\delta}{(\Delta-\delta)/2}$ — of functions from $\left(\mathcal{F}^0_{[h,h]} \cap \mathcal{R}^0_L\right)$ using its queries to TRUE.

Upperbound on the FR security of OneChaff$_{\text{hd}}$. Recall that the lowerbound on $|\mathcal{M}_1|$ gives us an upperbound on the FR advantage of an adversary that attacks *OneChaff*$_{\text{hd}}$, where $\mathcal{R}^q_L$ denotes the set of functions in the Δ neighborhood of H that are correct on all the queries that the adversary makes to TRUE. The bound in Claim 3 is a little complex and hard to interpret. In the following Corollary to Claim 3, we derive a weaker, but easier to interpret bound on $|\mathcal{M}_1|$. In the derivation, we use the standard relation $\binom{p}{r} \geq (p/r)^r$, and the assumptions: $q < \min(h/8, \Delta^2/256n)$ and $(h < h+\Delta < 2^n)$ to get the bound.

Corollary 1. *Let* $0 \leq q < \min(h/8, \Delta^2/256n)$. *Let* $\Delta < h$ *and* $0 < (\Delta + h) < 2^{n-1}$. *With probability at least* $(1 - 2/\Delta^2)$,

$$|\mathcal{M}_1| = \left|\left(\mathcal{F}^q_{[h,h]} \cap \mathcal{R}^q_L\right)\right| \geq \left(\frac{2^n(h-q)}{\Delta^2}\right)^{\Delta/4}.$$

Proof. Let $Z_q = \mathcal{Q}^{\text{key}}_q$. From Claim 2 and Lemma 2, we have, $Z_q \leq \sqrt{4q \ln \Delta} + \frac{q\Delta}{h+\Delta(1-h/2^{(n-1)})}$, with probability at least $(1 - 2/\Delta^2)$. Since we are finding an upperbound on Z_q, we can substitute $h + \Delta(1 - h/2^{(n-1)})$ with h to get a weaker-but-easier-to-interpret bound. As discussed earlier, for practical *OneChaff*$_{\text{hd}}$ schemes, the loss in accuracy will not be much since $\Delta \ll 2^n$. In the first term, $\ln \Delta < \ln 2^{n-1} < (n-1)\ln 2 < n$ since we assume that $\Delta < \Delta + h < 2^{n-1}$. Thus, $\sqrt{4q \ln \Delta} < \sqrt{4qn}$.
In order to prove the bound, which is the inverse of the product of the two binomial terms in Claim 3, we will show that the left binomial term satisfies

$$\binom{(h+\delta-q)}{((\Delta+\delta)/2 - (q\Delta/h + \sqrt{4qn}))} \geq \left(\frac{h-q}{\Delta}\right)^{\Delta/4},$$

and the right binomial term satisfies

$$\binom{2^n - h - \delta}{(\Delta - \delta)/2} \geq \left(\frac{2^n}{\Delta}\right)^{\Delta/4}.$$

Under our assumption $q < h/8$, $q\Delta/h < \Delta/8$. When $q < \Delta^2/256n$, $\sqrt{4qn} < \Delta/8$. Thus, $(q\Delta/h + \sqrt{4qn}) < \Delta/4$, and

$$\binom{(h+\delta-q)}{((\Delta+\delta)/2 - (q\Delta/h + \sqrt{4qn}))} \geq \binom{(h+\delta-q)}{(\Delta/4 + \delta/2)} \geq \binom{(h+\delta-q)}{(\Delta/4 + \delta/4)}.$$

Using the standard lower bound on binomial coefficient $\binom{p}{r} = \binom{p}{r} \geq (p/r)^r$, we get:

$$\binom{(h+\delta-q)}{(\Delta+\delta)/4} \geq \left(\frac{h+\delta-q}{(\Delta+\delta)/4}\right)^{(\Delta+\delta)/4} \geq \left(\frac{h-q}{\Delta}\right)^{\Delta/4}.$$

Since $\delta = \Delta(1 - h/2^{n-1})$, and $h + \Delta < 2^{n-1}$, we have: $\Delta - \delta < \Delta$, and $h + \delta \leq h + \Delta < 2^{n-1}$. Using these observations and the standard lower bound on binomial coefficient, we get

$$\binom{2^n - h - \delta}{(\Delta - \delta)/2} \geq \binom{2^{n-1}}{(\Delta - \delta)/2} \geq \left(\frac{2^n}{(\Delta-\delta)/4}\right)^{\frac{\Delta-\delta}{4}} \geq \left(\frac{2^n}{(\Delta-\delta)/4}\right)^{\Delta/4} \geq \left(\frac{2^n}{\Delta}\right)^{\Delta/4}$$

■

We are now prepared to state the upper bound on the FR-advantage of an adversary attacking *OneChaff*$_{\text{hd}}$. Since $\mathbf{Adv}^{\text{FR}}_{(\Pi,\theta),\textit{Fab}}(\mathcal{F}_{[h,h]}, A) \leq 1/|\mathcal{M}_1| + \epsilon_{t,q,1}$, the final bound follows directly from Corollary 1. Recalling that $\epsilon_{t,q,1}$ captures the probability that the sampled hidden function F is a "simple" function (see Definition 3), we have the following theorem.

Theorem 3. *Fix* $m = 1$ *and integers* $n, h, \Delta, q > 0$ *such that* $\Delta < h$, $0 < \Delta + h < 2^{n-1}$, *and* $0 \leq q < \min(h/8, \Delta^2/256n)$. *Let an honest-but-curious adversary* A *attack the FR security of OneChaff*$_{\text{hd}}$ *using resources* $(t, q_f = 1, q_s = 1, q_r = q)$. *Let* $\epsilon_{t,q,1}$ *be as defined in Definition 3. Then,*

$$\mathbf{Adv}^{\text{FR}}_{(\textit{OneChaff}_{\text{hd}},\theta),\textit{Fab}}(\mathcal{F}_{[h,h]}, A) \leq \left(\frac{\Delta^2}{2^n(h-q)}\right)^{\Delta/4} + \epsilon_{t,q,1},$$

with probability at least $(1 - 2/\Delta^2)$. ♦

Let us unpack the Theorem a bit. In *OneChaff*$_{\mathrm{hd}}$, an IP author can configure Δ while running *Hide*. We will carve out three criteria from Theorem 3 in terms of: $|\mathrm{Dom}(F)|$, $\mathrm{hw}(F)$ and Δ, that are needed for FR security of *OneChaff*$_{\mathrm{hd}}$.

The FR security of *OneChaff*$_{\mathrm{hd}}$ increases polynomially in the size of the domain of the hidden function, as well as the hamming weight of the function (as the hamming weight approaches 2^{n-1}). The rate of increase depends on the value of Δ — large values of Δ result in faster rate of increase. All three criteria are quite intuitive. Functions with small domains will be simple because they can be learnt by brute force; functions with very small hamming weights will be "close" to being constant functions. On the contrary, when F is a uniformly sampled balanced Boolean function with large domain, the min-entropy of the (uniform) distribution on $\mathcal{F}$ will be very large (compared to functions with very small hamming weights) as $|\mathcal{F}| = \binom{2^n}{2^{n-1}} > \binom{2^n}{\ell}$ for any $\ell \in [0, 2^n] \setminus \{2^{n-1}\}$.

While efficient *OneChaff*$_{\mathrm{hd}}$ schemes will require Δ to be small (as the size of K_O is $n\Delta$), the IP author should choose as *large* Δ as possible to maximize FR-security guarantee. This criteria is also quite intuitive as large Δ implies large number of secret distinguishing inputs (that are encoded in K_O) on which the adversary does not know the value of F.

None of the previous works consider these factors while discussing the security of DH schemes. For example, in the evaluation of the SAT attack [SRM15], four base-benchmark circuits (c17, ex5, apex4, ex1010) have very small domains: in c17, $n = 5$; in ex5, $n = 8$; in apex4 and ex1010, $n = 10$. Note that the authors used each base-benchmark circuit to create 21 opaque circuits. That means roughly 19% of the benchmark circuits[5] in the test corpora of SAT attack cannot be protected by any DH scheme. It is noteworthy to also point that the open-source test corpora of SAT attack is used by other attack algorithms as well [SLM+17a, SLR+19].

8 Next Steps

This work initiates a provable-security exploration of design-hiding schemes. We stress *initiates*: the foundations that we have developed allow us to design the first DH scheme that is supported by a concrete security analysis, with respect to formal security notions that capture the attack model and adversarial goals intended by prior work. While we believe this work is an important step forward, we recognize that there is still much to be done.

Some obvious directions are to generalize the main security result for *OneChaff*$_{\mathrm{hd}}$ to non-Boolean functions. One idea would be to consider the transitive fan-in cone of each output bit of a multi-bit function, which defines a Boolean subfunction.

Also, it may be fruitful to consider more sophisticated modeling of the foundry's a priori knowledge.

Our FR-security notion demands that the foundry recover the *entire* hidden function. This is the demand of prior works, too, but it is likely too strong. It rules out attacks that, for all intents and purposes, effectively reverse-engineers the hidden function to any *efficient* test. One might modify the FR-security experiment to demand input-output correctness on a set $\mathcal{T} \subset \mathrm{Dom}(F)$. Sharper still would be to make a *Test* algorithm be a syntactic component of a DH scheme, and modify the FR-security experiment to declare an attack successful only if it fools the *Test* algorithm into saying that the foundry's dishonestly produced chip (not circuit) is functionally correct.

Going a different direction, while *OneChaff*$_{\mathrm{hd}}$ is the first provably secure DH scheme, we expect there are many others. We encourage efforts to discover them, with an eye towards practically desirable features, e.g. small area/power/delay overheads.

Finally, we consider DH schemes for stateless circuits. Modern, real-world circuits are often stateful, and comprised of multiple stateless subcircuits. Extending our formalisms to such circuits is an important next step. Note that one may treat each stateless sub-circuit as an independent circuit, and try to use *OneChaff*$_{\mathrm{hd}}$ to prevent full-function recovery attacks on each of these.

[5] Out of 441 benchmark circuit, 84 circuits have very small domains in the test corpora.

However, such a construction may add a lot of overhead, in terms of the size of the final chip. Finding a good trade-off between efficiency and security for practical circuits is also an interesting direction.

References

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE, 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[BR14] Zvika Brakerski and Guy N Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *Theory of Cryptography Conference*, pages 1–25. Springer, 2014.

[BTZ10] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing ic piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 27(1), 2010.

[CB11] Rajat Subhra Chakraborty and Swarup Bhunia. Security against hardware trojan attacks using key-based design obfuscation. *Journal of Electronic Testing*, 27(6):767–785, 2011.

[CBWC12] Lap Wai Chow, James P Baukus, Bryan J Wang, and Ronald P Cocchi. Camouflaging a standard cell based integrated circuit, April 3 2012. US Patent 8,151,235.

[CCB18] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. Sail: Machine learning guided structural analysis attack on hardware obfuscation. In *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 56–61. IEEE, 2018.

[CCB19] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. Surf: Joint structural functional attack on logic locking. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 181–190. IEEE, 2019.

[CDG+14] Jean-Michel Cioranesco, Jean-Luc Danger, Tarik Graba, Sylvain Guilley, Yves Mathieu, David Naccache, and Xuan Thuy Ngo. Cryptographically secure shields. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 25–31. IEEE, 2014.

[Che18] Yung-Chih Chen. Enhancements to sat attack: Speedup and breaking cyclic logic encryption. *ACM Trans. Des. Autom. Electron. Syst.*, 23(4):52:1–52:25, May 2018.

[CT16] Henry Carter and Patrick Traynor. Opfe: Outsourcing computation for private function evaluation. *IACR Cryptology ePrint Archive*, 2016:67, 2016.

[DBDN+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 *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 49–54. IEEE, 2014.

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits iii: Hardware trojan-resilience via testing amplification. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 142–153, New York, NY, USA, 2016. ACM.

[DW08] Ronald De Wolf. A brief introduction to fourier analysis on the boolean cube. *Theory of Computing*, pages 1–20, 2008.

[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.

[EKR+18] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[GOS+11] Parikshit Gopalan, Ryan O'Donnell, Rocco A Servedio, Amir Shpilka, and Karl Wimmer. Testing fourier dimensionality and sparsity. *SIAM Journal on Computing*, 40(4):1075–1100, 2011.

[JNO14] Thomas P Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 81–92, 2014.

[KAHS19] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 89:1–89:6, New York, NY, USA, 2019. ACM.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 797–808, 2012.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In *Annual Cryptology Conference*, pages 440–457. Springer, 2014.

[KSMB13] Ben Kreuter, Abhi Shelat, Benjamin Mood, and Kevin Butler. {PCF}: A portable circuit format for scalable two-party secure computation. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 321–336, 2013.

[Lin05] Yehuda Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI Global, 2005.

[LSM+16] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin, and David Z. Pan. Provably secure camouflaging strategy for ic protection. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 28:1–28:8, New York, NY, USA, 2016. ACM.

[Mal11] Lior Malka. Vmcrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 715–724, 2011.

[MNP+04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4, page 9. San Diego, CA, USA, 2004.

[MZGT17] Mohamed El Massad, Jun Zhang, Siddharth Garg, and Mahesh V. Tripunitara. Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism. *CoRR*, abs/1703.10187, 2017.

[NASR04] G-J Nam, Fadi Aloul, Karem A Sakallah, and Rob A Rutenbar. A comparative study of two boolean formulations of fpga detailed routing constraints. *IEEE Transactions on Computers*, 53(6):688–696, 2004.

[NNOB11] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. Cryptology ePrint Archive, Report 2011/091, 2011. https://eprint.iacr.org/2011/091.

[PM14] S. M. Plaza and I. L. Markov. Protecting integrated circuits from piracy with test-aware logic locking. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 262–269, Nov 2014.

[RKM08] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074. ACM, 2008.

[RMKS18] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. Srclock: Sat-resistant cyclic logic locking for protecting the hardware. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 153–158. ACM, 2018.

[RPSK12a] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *DAC Design Automation Conference 2012*, pages 83–89, June 2012.

[RPSK12b] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Logic encryption: A fault analysis perspective. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 953–958, San Jose, CA, USA, 2012. EDA Consortium.

[RSK13] J. Rajendran, O. Sinanoglu, and R. Karri. Is split manufacturing secure? In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1259–1264, March 2013.

[RSSK13] Jeyavijayan Rajendran, Michael Sam, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of integrated circuit camouflaging. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 709–720, New York, NY, USA, 2013. ACM.

[SAFT16] Bicky Shakya, Navid Asadizanjani, Domenic Forte, and Mark Tehranipoor. Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In *Proceedings of the 35th International Conference on Computer-Aided Design*, pages 1–8, 2016.

[SHS+15] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, 2015.

[SL91] D-J Shyy and C-T Lea. Log/sub 2/(n, m, p) strictly nonblocking networks. *IEEE Transactions on Communications*, 39(10):1502–1510, 1991.

[SLM+17a] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin. Appsat: Approximately deobfuscating integrated circuits. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100, May 2017.

[SLM+17b] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. Cyclic obfuscation for creating sat-unresolvable circuits. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 173–178, New York, NY, USA, 2017. ACM.

[SLP+19] Kaveh Shamsi, Meng Li, Kenneth Plaks, Saverio Fazzari, David Z Pan, and Yier Jin. Ip protection and supply chain security through logic obfuscation: A systematic overview. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(6):1–36, 2019.

[SLR+19] Yuanqi Shen, You Li, Amin Rezaei, Shuyu Kong, David Dlott, and Hai Zhou. Besat: behavioral sat-based attack on cyclic logic encryption. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 657–662. ACM, 2019.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 244–257, 2009.

[SPJ19] Kaveh Shamsi, David Z Pan, and Yier Jin. On the impossibility of approximation-resilient circuit locking. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 161–170. IEEE, 2019.

[SRM15] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, May 2015.

[SRZ18] Yuanqi Shen, Amin Rezaei, and Hai Zhou. Sat-based bit-flipping attack on logic encryptions. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 629–632. IEEE, 2018.

[SS19] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 936–939. IEEE, 2019.

[SZ17] Yuanqi Shen and Hai Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 179–184, New York, NY, USA, 2017. ACM.

[TI19] The-Intercept. Everybody does it: the messy truth of computer suppy chains. Technical report, 2019.

[Tre18] TrendForce. Trendforce reports top 10 ranking of global semiconductor foundries of 2018. Technical report, 2018.

[VPH+16] Arunkumar Vijayakumar, Vinay C Patil, Daniel E Holcomb, Christof Paar, and Sandip Kundu. Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques. *IEEE Transactions on Information Forensics and Security*, 12(1):64–77, 2016.

[XS16] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 127–146. Springer, 2016.

[XSTF17] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 189–210. Springer, 2017.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982.

[YMRS16] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. Sarlock: Sat attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241. IEEE, 2016.

[YMSR17a] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[YMSR17b] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of anti-sat. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 342–347. IEEE, 2017.

[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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1601–1618, New York, NY, USA, 2017. ACM.

[YSS+17] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. What to lock?: Functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 351–356, New York, NY, USA, 2017. ACM.

[YTS19] Fangfei Yang, Ming Tang, and Ozgur Sinanoglu. Stripped functionality logic locking with hamming distance based restore unit (sfll-hd)–unlocked. *IEEE Transactions on Information Forensics and Security*, 2019.

[ZAMKHS19] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. Threats on logic locking: A decade later. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, pages 471–476, New York, NY, USA, 2019. ACM.

[ZJK17] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 49–56. IEEE Press, 2017.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.

cryptography, provable security, design hiding, hardware obfuscation, logic locking, logic encryption, IC camouflaging,,