On the Impossibility of Approximation-Resilient Circuit Locking

Kaveh Shamsi*, David Z. Pan†, and Yier Jin*
*Department of Electrical and Computer Engineering, University of Florida
†Department of Electrical and Computer Engineering, University of Texas at Austin
kshamsi@ufl.edu, dpan@ece.utexas.edu, yier.jin@ece.ufl.edu

***Abstract*—Logic locking, and Integrated Circuit (IC) Camouflaging, are techniques that try to hide the design of an IC from a malicious foundry or end-user by introducing ambiguity into the netlist of the circuit. While over the past decade an array of such techniques have been proposed, their security has been constantly challenged by algorithmic attacks. This may in part be due to a lack of formally defined notions of security in the first place, and hence a lack of security guarantees based on long-standing hardness assumptions.**

In this paper we take a formal approach. We define the problem of *circuit locking* ($c\mathcal{L}$) as transforming an original circuit to a locked one which is "unintelligable" without a secret key (this can model camouflaging and split-manufacturing in addition to logic locking). We define several notions of security for $c\mathcal{L}$ under different adversary models. Using long standing results from computational learning theory we show the impossibility of exponentially approximation-resilient locking in the presence of an oracle for large classes of Boolean circuits. We then show how exact-recovery-resiliency and a more relaxed notion of security that we coin "best-possible" approximation-resiliency can be provably guaranteed with polynomial overhead. Our theoretical analysis directly results in stronger attacks and defenses which we demonstrate through experimental results on benchmark circuits.

I. Introduction

In today's globalized semiconductor industry, fabless design houses hand over the design to the foundry which then has complete knowledge of the physical layout of the design. This business model results in several major security concerns. These broadly include, 1) the theft of the design and the intellectual property inherent to it, 2) overproduction, and 3) the possibility of malicious modification of the design by the foundry. In addition to these foundry threats, fabricated ICs are under the threat of microscopy-based reverse engineering and netlist recovery by end-users.

There are three broad categories of techniques that allow the designer to hide the design of the circuit from the foundry, the end-user, or both. These are : 1) IC camouflaging [1], 2) logic locking [2], [3], and 3) split-manufacturing [4]. IC camouflaging is based on using special layout structures that are difficult to disambiguate under conventional microscopy-based reverse engineering techniques thwarting end-user attackers while providing no protection against the foundry. Logic locking is based on adding programmability to the design, such that a post-fabrication configuration step is necessary for the circuit to function correctly. Logic locking can protect against both the foundry and the end-user if the programmable technology is inspection-resilient. Split-manufacturing is based on fabricating only the lower layers of the design in the untrusted foundry hiding away the upper interconnect layers from the foundry while providing no protection against end-user microscopy-based reverse engineering. The primary metric of evaluation for these *design hiding* schemes is the security they provide versus/divided-by the overhead and cost that they incur to the IC and the manufacturing process.

Unlike split-manufacturing, logic locking and IC camouflaging do not need a second foundry. Various locking and camouflaging techniques have been proposed in the past decade and several intriguing algorithmic attacks have been demonstrated against them. A very successful category of such attacks is oracle-guided attacks in which the attacker is assumed to have access to a functional circuit that can produce correct outputs for chosen input patterns [1], [5], [6]. These patterns can be used to disambiguate the netlist. Among these attacks, SAT attacks [5], [6] which are based on iterative SAT queries, are the strongest and have been shown to deobfuscate a variety of low-overhead locking and camouflaging techniques.

Attempts have been made to thwart SAT attacks and oracle-guided attacks in general by exponentially increasing the minimum number of queries needed to resolve the circuit [3]. However, exponentially increasing the number of minimum queries often results in a degraded error rate, i.e. the probability of an incorrect key producing incorrect outputs. As a result, these schemes have been attacked with approximation attacks [7], [8] raising the question of whether approximation-resiliency is possible with strong guarantees.

Modern cryptography research is dominated by what is known as the *reductionist* approach to proving security. In this approach to cryptographic problems, security itself is defined formally, and then proved using assumptions on the security of sub-modules of the protocol, plus hardness of particular computational problems such as factoring. While such formalities can provide little help in the design of fast heuristic primitives such as block and stream ciphers (e.g. we lack a formal proof of security for AES), they dominate almost all other areas of modern research in cryptography such as public-key encryption, post-quantum cryptography, obfuscation, homomorphic encryption, and protocols. In this paper we try to take a similar approach to the security analysis of design hiding schemes in particular locking and camouflaging. We show how such a formalism can have several intriguing results. Specifically we deliver the following contributions:

- We formally define the problem of *circuit locking* that can model all three design hiding techniques. We then define

978-1-5386-8064-3/19/$31.00

several notions of security that model different practical attacker-defender scenarios under different threat models.
- We show via long-standing results in computational learning theory why it may be impossible to satisfy the approximation-resiliency notion of security for low depth combinational circuits under oracle-guided attacks.
- We define a more relaxed notions of security called best-possible approximation-resiliency. For this definition of security we present a construction that information-theoretically satisfies it with polynomial overhead.
- We complement our theoretical results with experiments on benchmark circuits. The theoretical formalization directly implies black-box-only exact/approximate deobfuscation attacks which we demonstrate in practice.

The paper is organized as follows. Section II defines circuit locking and security properties. Section III presents constructions that satisfy security definitions. Section IV presents the experimentation results and Section V concludes the paper.

II. Defining Circuit Locking and its Security

A. A Common Model for Design Hiding Schemes

Consider the following transformation that adds additional inputs to a Boolean function (circuit). This transformation takes an *original circuit* $c_o(i) : I \rightarrow O$ and converts it to a *keyed/augmented/locked* circuit $c_e(i,k) : I \times K \rightarrow O$ by adding l *key-inputs/variables* to the circuit, hence $K = \mathbb{F}_2^l$ is the key space and there is a *correct key* $k_* \in K_* \subset K$ for which $\forall i \in I$ we have $c_e(i,k_*) = c_o(i)$. i.e. loading the correct key into c_e makes it behave exactly as c_o on all inputs. c_o and c_e can be sequential/stateful circuits that operate on state vectors s_o and s_e respectively. Different values of k induce a function space $\mathcal{C} = \{c_e(i,k) | k \in K\}$ which the attacker has to sort through in order to recover c_o.

Logic locking or "key-based obfuscation" is implemented in silicon by inserting programmable elements in the IC design and configuring them post fabrication. Hence, it is easy to see how logic locking can be modeled directly by the above transformation; by taking the programmable elements as key-inputs. IC camouflaging on the other hand, is implemented using special nano-device structures that are ambiguous to the physical reverse engineer. For IC camouflaging, the ambiguity in the reverse engineered netlist can be encoded by a polynomially bounded number of key-inputs in the number of ambiguous camouflaged elements. There are numerous ways to encode the function space $\mathcal{C}$ as long as $\mathcal{C}$ still includes the original function c_o. Split-manufacturing can also be modeled this way by encoding all the missing interconnect information using multiplexers (MUXs) controlled by key-inputs and subsequently defining the function space $\mathcal{C}$. Figure 1 shows how various netlist ambiguities can be encoded with key-inputs with polynomial inflation.

Since this transformation directly models logic locking, and indirectly IC camouflaging and split-manufacturing, in our paper we refer to the mathematical problem itself as *circuit locking (c$\mathcal{L}$)*. We refrain from using the term *obfuscation*, to avoid confusion with the famous problem of *program/circuit obfuscation* in cryptography [9], in which the functionality of the obfuscated circuit/program is not altered. This is while in c$\mathcal{L}$ the functionality is exploded into a new space $\mathcal{C}$ and additional key-inputs are added[1]. In addition, we have to always keep in mind the distinction between the mathematical problem c$\mathcal{L}$ which we will define and discuss in detail in this paper, and the practical design hiding flow (logic locking/IC camouflaging/split-manufacturing) that it can model.

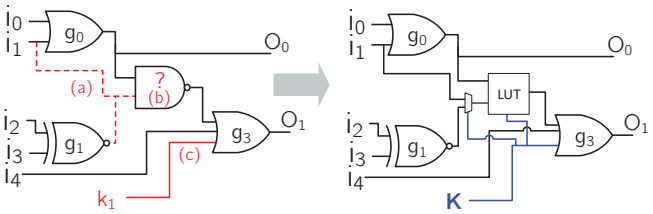

Fig. 1: (a) missing interconnect information, (b) ambiguous gate, and (c) key-wire, can all be modeled using key-inputs with polynomial overhead: (a) with key-controlled MUX logic, (b) with key-controlled look-up-tables (LUT), and (c) key-wires directly with key-inputs.

B. Threat Models and Physical Security

Oracle access to c_o is a critical issue in defining security for c$\mathcal{L}$. We take this into account in our mathematical framework using the oracle-guided (OG) and oracle-less (OL) tags. Before defining c$\mathcal{L}$, it is important to understand which attack model to use in practical scenarios. For instance, for camouflaged ICs, if an attacker can buy two camouflaged ICs, use one to recover the ambiguous netlist c_e, and use the other as an oracle c_o, then the oracle-guided attack model should be used. In the case of logic locking, an oracle-less attack model is viable only if the attacker is the foundry that is fabricating the first ever batch of a design, for which an activated version of the locked circuit is not available through any other means. For split-manufacturing an oracle-guided attack model is only reasonable for a foundry attacker that has the bottom layout, can buy a functional chip from the market, and does not have the capability or want to invest to physically reverse engineer the top layers of the oracle chip obtained from the market.

If the attacker has access to all internal registers, c_e and c_o are combinational circuits. If there are state-elements that are part of feedback loops in between observable and controllable points in the oracle, the c$\mathcal{L}$ problem becomes sequential with stateful c_e and c_o. The more granular the access of the attacker to the oracle wires the stronger the attacker becomes and the harder it is to secure c$\mathcal{L}$. For instance, if an attacker has access to $t > 0$ number of the active values of the oracle's internal wires through photon-emission microscopy, laser-probing, or physical probing [10], it will become almost impossible to secure c$\mathcal{L}$ since the attacker can simply probe the value of key-variables or their immediate cones and learn the circuit. Furthermore, for locking and camouflaging, the ambiguous nano-device itself should be secure against microscopy in order for the c$\mathcal{L}$ problem defined in this paper to model it.

[1]We can use the term *deobfuscation* to refer to algorithmic attacks against locking as it does not collide with a heavily used term in another area.

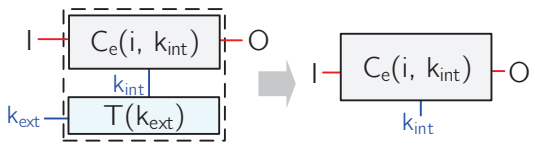

Fig. 2: Locking with a key-mapping. The attacker can find all of T's logic because all the wires in T are exclusively controlled by key-inputs. The attacker can remove T if the functionality of the original circuit is the target.

C. Circuit Locking Goals

Before defining $c\mathcal{L}$ and notions of security precisely, it is important to ask the question: what do we intuitively expect from circuit locking? The following two goals are used typically in related literature:

- Inoperability of fabricated ICs without the correct key. In this case, the defender only wants to make it difficult to activate a fabricated IC without the correct key.
- Hiding the functionality of the design. Functionality in this context is the truth-table or any other (concise) representations of the Boolean function implemented by the circuit such as a netlist, Binary-Decision-Diagram (BDD), etc.

The first goal has been used in several published work. A recent example is the "delay locking" presented in [11]. In this work a circuit equivalent to c_o can be recovered form c_e. However, added variable-delay elements that are controlled by key-inputs make it difficult to unlock another fabricated chip since a key that will result in the correct timing behavior for the fabricated chip is not known to the attacker. A second example is the performance locking in [12]. In this scheme again, a circuit equivalent to c_o can be recovered, however, the recovered circuit may be exponentially slower.

The third example is the scheme presented in [13] in which a key-transformation is used. Key transformations between an external and an internal key can seem quite useful in logic locking. Formally one transforms $c_e(i,k)$ to $c'_e(i,k_{ext}) \equiv c_e(i,T(k_{ext}))$ as seen in Figure 2. In [13], an XOR array is used to XOR key-bits with one another to cause a single key-bit flip on the external key to result in numerous internal key-bit flips which can improve error rate and entropy. It is tempting to use a physical-uncloanable-function (PUF) to perform the key-mapping giving each chip a unique external key with a shared internal key. While these techniques hamper activating an already fabricated IC, as for the functionality, the functional attacker is interested in the Boolean function of c_o and hence can replace c'_e with any other keyed circuit that induces a function space $\mathcal{C}$ as long as $c_o \in \mathcal{C}$. The attacker can find T by searching for wires controlled exclusively by keys and remove T from c'_e. Any key-exclusive logic that does not mix with primary inputs although can contribute to the first goal, it does not contribute to functional secrecy.

We present two arguments for focusing on the second goal of functional secrecy: 1) An attacker who obtains the functionality of the design can use re-synthesis and optimization techniques to improve the design. Hence, for a performance-based locking scheme, the security relies on ensuring that the subsequent performance optimization problem is difficult. This becomes particularly challenging to ensure for smaller circuits. 2) The first goal of inoperability of fabricated ICs can be easily guaranteed with much simpler solutions! For instance, comparing a sufficiently long input string to a unique key stored on a tamper-proof memory in the IC, and activating the power or enabling a critical signal in the design only upon receiving this correct key achieves this goal. The only overhead for this scheme comes from one comparator logic for the entire chip. As such, the second goal of functional secrecy is the more intuitive and elusive target for circuit locking. We build our formal framework around functional secrecy.

D. Combinational Circuit Locking

The complexity theoretic model of circuits is based on Directed-Acyclic-Graphs (DAGs) with each node representing a gate from the *basis*, i.e. the set of possible gates that the circuit can use. Circuits of depth $O(\log^i(n))$ on n inputs that use the AND/OR/NOT basis with bounded/unbounded fanin can solve problems in the complexity classes NC^i/AC^i respectively. Adding *threshold* gates which can compute the majority of input bits to the basis, with unbounded fanin, solves the complexity class TC^i. We define the combinational circuit locking scheme on DAG circuit families:

DEFINITION 1 (Combinational Circuit Locking ($c\mathcal{L}$) scheme). *A Combinational Circuit Locking ($c\mathcal{L}$) scheme for a family of stateless DAG circuits $\{C_\lambda\}$ is a probabilistic polynomial time (PPT) algorithm* Lock *that takes security parameter λ and an original circuit $c_o \in \{C_\lambda\}$ and returns the locked combinational circuit c_e and a correct key k_* where we have the following:*

- *(l Added Key-inputs) When $c_o : I \to O$ where $I = \mathbb{F}_2^n$ and $O = \mathbb{F}_2^m$,* $\mathsf{Lock}(\lambda, c_o)$ *returns $c_e : I \times K \to O$ where $K = \mathbb{F}_2^l$.*
- *(Correct Functionality under Correct Key) We have $\forall i \in I,\ \ c_e(i,k_*) = c_o(i)$.*
- *(Polynomial Overhead) We have $size(c_e) \leq poly(size(c_o))$ and $depth(c_e) \leq poly(depth(c_o))$.*

E. Defining Security

Before defining security we expand on a couple of important concepts herein.

Perfect security. Consider a circuit that can be programmed to implement all possible n to m bit Boolean functions by configuring the key. This circuit can be implemented by m look-up-tables (LUT) with 2^n entries, each entry controlled by a key-bit. We refer to this as the $\mathsf{ExpLUT}(n,m)$ circuit.

A $c\mathcal{L}$ scheme that simply produces $\mathsf{ExpLUT}(n,m)$ for any n-input m-output circuit is intuitively the best that a $c\mathcal{L}$ scheme can do. This is analogous to the inefficient one-time-pad in cryptography that achieves information-theoretic (perfect, or with zero leakage) security. The locked circuit is not leaking any information regarding c_o and each query to an oracle of c_o reveals a single truth-table entry of c_o. However, this scheme is not only inefficient but results in a security dichotomy: If the number of inputs n is small

enough for ExpLUT(n, m) to fit on an IC, it must be small enough for the attacker to query and learn completely. Hence, we are interested in achieving super-polynomial security with polynomially sized circuits which is why the third constraint in Definition 1 is necessary.

Oracle Access. In terms of oracle-access we have the oracle-less (OL) and oracle-guided (OG) models. We assume oracle-guided attacks to be adaptive in nature since in the context of circuit locking the chosen-plaintext versus adaptive-chosen-plaintext attacker distinction made in cryptography is not realistic. This is because an oracle circuit can be queried indefinitely by the attacker. We assume that queries on the oracle take constant time. Now we are ready to define security.

DEFINITION 2 (Exact Functional Secrecy (EFS)). EFS-OG *corresponds to the following adversary game. The adversary $\mathcal{A}$ has c_e and can make up to q chosen input queries to c_o and wins by returning a representation perfectly equivalent to c_o. We say a $c\mathcal{L}$ scheme is (t, q, σ)-*EFS-OG *secure, if the advantage (probability of winning the game) of any $\mathcal{A}$ bounded by t operations[2] is no more than σ better than adversary $\mathcal{A}'$ that makes q queries and randomly guesses the remaining $2^n - q$ entries of c_o's truth table. (t, σ)-*EFS-OL *corresponds to a similar game except the adversary has no access to an oracle that implements c_o ((t, σ)-*EFS-OL $\equiv$ $(t, 0, \sigma)$-*EFS-OG*).*

The above definition requires the attacker to extract a perfect reconstruction of c_o. A stronger security criteria is approximation-resiliency which we define as follows:

DEFINITION 3 (Approximate Functional Secrecy (AFS)). *The adversary $\mathcal{A}$ has c_e and can make up to q chosen input queries to c_o and has to return an ϵ-approximation[3] of c_o. We say that a $c\mathcal{L}$ scheme is (t, q, ϵ, σ)-*AFS-OG *secure if the advantage of any $\mathcal{A}$ bounded by t operations is no more than σ better than the advantage of the adversary $\mathcal{A}'$ that makes q queries to c_o and randomly guesses the remaining $2^n - q$ truth-table entries. As for* OL *attackers, (t, ϵ, σ)-*AFS-OL $\equiv$ $(t, 0, \epsilon, \sigma)$-AFS-OG

The following can immediately be derived:

PROPOSITION 1. AFS-OG *implies* AFS-OL *and* EFS-OG *implies* EFS-OL,*respectively. In addition,* AFS *implies* EFS.

PROPOSITION 2. t-EFS/AFS-OG *imply t-hardness of recovering an exact/approximate key (an approximate key is a k_{app} s.t. $c_e(i, k_{app})$ ϵ-approximates c_o) respectively, but the reverse is not true.*

F. Oracle Learnability

Intuitively, AFS security demands that the rate of learning the locked circuit should not exceed that of learning the truth-table entry by entry. Authors in [14] first showed that combinational benchmark circuits can be learned with deep neural networks. Here we prove that this is in fact a fundamental issue inhibiting AFS security. *Many classes of Boolean functions are learnable from random or chosen input-output pairs in polynomial time* at a rate significantly better than learning their truth-table. Hence, instead of bothering with attacking the locked circuit c_e, the OG attacker can directly try to learn c_o with significant statistical advantage.

We draw upon results from computational learning theory [15] for this argument. In learning theory a learner L intends to learn the functionality of a target function f_t. The target function f_t is a member of a hypothesis space $\mathcal{H}$. $\mathcal{H}$ can be used to also encode any a priori knowledge about f_t. Two kinds of queries are discussed in learning literature: *membership queries* in which the learner can query x and receive $f_t(x)$, and *equivalence queries* where the learner can query the function h_t and receive whether h_t and f_t are equivalent or not. Equivalence queries are not possible under the oracle-guided attack model in our case of $c\mathcal{L}$.

A Probably-Approximately-Correct (PAC)-learner for a class of representations C is an algorithm L that can produce an ϵ-approximation of any $f \in C$ given access to *random* $(x, f(x))$ pairs drawn from a distribution D with success rate δ. An *efficient* PAC-learner finishes this task in time poly$(n, 1/\epsilon, \delta)$. The δ parameter is necessary since any PAC-learner will have probabilistic success in reaching a particular ϵ. OG attacks in $c\mathcal{L}$ differ from PAC-learning in that there is no reason to restrict the attacker to samples drawn only from D. Instead the attacker is assumed to have the more powerful tool of membership queries (chosen inputs).

Learning theory studies some important function classes: a) Disjunctive-Normal-Form (DNF) / Conjunctive-Normal-Form (CNF) formulas which are OR/ANDs of AND/ORs of input variables respectively. The size of a CNF/DNF formula is the number of terms in it, and its width is the maximum number of literals in each term. b) Decision-Trees (DTs) are rooted trees with a variable x_i on each vertex and 1 and -1 outgoing edges that eventually lead to constant values. The size of a DT is the number of leafs in the tree and its depth is the length of the maximum path from the root to leafs, and c) DAG circuits in different bases which we discussed earlier.

Learning theory on Boolean functions heavily relies on the *Fourier spectrum* of such functions. If we encode binary values as $\{0, 1\} \rightarrow \{1, -1\}$, any Boolean function $f(x) : \{-1, 1\}^n \rightarrow \{-1, 1\}$ can be written as $f(x) = \sum_{S \in [n]} \hat{f}(S)\chi_S(x)$ where $\chi_S(x)$ is the S-parity function that computes the parity of bits of x whose indices are present in S by multiplying them: $\chi_S(x) = \prod_{i \in S} x_i$. (Multiplication on $\{-1, 1\}$ corresponds to XOR on $\{0, 1\}$). $\hat{f}(S)$ are called the Fourier coefficients of f. Based on the Fourier transform we know the following results for n-input functions.

LEMMA 1 (Low-Degree-Algorithm (LDA)). *For a collection $\mathcal{S}$, if f is $\varepsilon/2$-concentrated on $\mathcal{S}$, meaning that $\sum_{S \notin \mathcal{S}} \hat{f}^2(S) \leq \varepsilon/2$, where $\mathcal{S} = \{S : |S| \leq k\}$ then f can be learned in time poly$(n^k, 1/\varepsilon)$ using random queries only.*

LEMMA 2 (Kushilevitz-Mansour (KM)). *If every $f \in C$ is $\varepsilon/4$-concentrated on a collection of M sets, then C can be*

[2]Any algorithm with circuit size complexity bounded by t.

[3]An ϵ-approximation of f disagrees with f on at most an ϵ fraction of the input space. For multi-output f disagreement in this context is defined as the average of disagreement of individual output bits. A word-level ϵ can be defined in cases where world-level information is important which is outside the scope of this paper.

learned in time poly$(M, 1/\varepsilon, n)$ from membership queries.

Under the ε-concentration assumption since only a small number of coefficients are significant, we can estimate them one by one from random samples and reconstruct the function. These coefficients can be estimated from the fact that for a given index set S we have $\hat{f}(S) = \mathrm{E}_x[f(x)\chi_S(x)]$.

Several representation classes have been shown to have concentrated Fourier spectrums including low depth circuits shown by Linial, Mansour, and Nisan (LMN) [15]. In summary we have the following lemmas regarding ϵ-approximation (ϵ-learning) of Boolean functions.

LEMMA 3. *Poly-size DNF formulae and poly-size DTs are ϵ-learnable from random samples in $n^{O(\log(n/\epsilon))}$. With membership queries these are ϵ-learnable in poly(n, ϵ).*

LEMMA 4. *The class of functions computable by circuits of size poly(n) and of depth d are ϵ-learnable from random queries in time $n^{O(\log(n))^d}$ with $\epsilon = 1/poly(n)$.*

Given the above results it is easy to see that AFS-OG with exponential security ($t \in \Omega(\mathrm{Exp}(n))$) may not be possible for large classes of functions.

PROPOSITION 3. *If $c_o \in \{C_\lambda\}$ where $\{C_\lambda\}$ is a family of circuits that are learnable with ϵ accuracy with δ success rate from membership queries, the advantage of an attacker in the* AFS-OG *game is at least δ.*

THEOREM 1. *The class of poly-sized DTs, poly-sized CNF/DNF formulae, and poly-sized constant-depth circuits (AC^0/NC^0), cannot be locked with* AFS-OG *security with $\sigma \in \Omega(Exp(n))$, i.e. exponentially small adversary advantage.*

What practical circuits are in fact in this group? Unfortunately many. Integer addition is in AC^0. Parity functions are not in AC^0, however, there is an efficient algorithm for learning them using membership queries [15]. The Espresso tool can mine for minimal DNF representations of Boolean circuits. Adders, multipliers, and comparators with 8-bit operands have DNF sizes of just thousands of terms. Polynomial learning of such DNF formulae can make locking them with AFS-OG impossible. As we will see in the experimentation section most combinational benchmarks have short depth and can be approximated with significant accuracy from random samples.

In fact AFS-OG security is even more restrictive. We can add to the above any circuit which has significantly unbalanced output signal probabilities:

PROPOSITION 4. *The advantage of an* AFS-OG *attacker is lower bounded by a function of the unbalance of output signal probabilities of c_o's outputs themselves regardless of c_e. If $\exists m,\ \left|\Pr_{i\in I}[c_o^m(i) = 1] - \frac{1}{2}\right| \geq \Delta^m$ we have $\sigma \geq poly(\Delta^m, 1/\epsilon)$.*

Proof. Assume that c_o is significantly unbalanced for output m. The adversary makes q queries to output m. If for the majority of the q queries $c_o^m(x_i) = 1$, the adversary returns 1 for the remaining truth-table entries, and 0 otherwise. The advantage of this attacker on output m is poly$(\Delta^m, 1/\epsilon)$ better than randomly guessing truth-table entries of c_o. □

G. Best-Possible Approximation-Resilient Locking

We can relax the notion of AFS to avoid the negative results. We define the notion of best-possible approximation-resiliency by making two important modifications: 1) We define the advantage relative to the "strongest learner" of the black-box c_o. 2) It immediately follows that the strongest learner of c_o must be smart enough to know that c_o cannot be larger and deeper than c_e. Knowledge of this bound on the depth and size of the original circuit is quite significant. It reduces the attacker's hypothesis space from the set of all n-input Boolean functions to the infinitely smaller set of functions implemented by bounded-depth/size combinational circuits.

Consequently, best-possible approximation-resiliency means that the best the attacker can do is to try to black-box-learn c_o through input-output queries with the knowledge that c_o's size and depth are bounded to a known value. In other words, the locked circuit c_e, beyond what the attacker can extract from the oracle through querying, reveals nothing new except a bound on the depth and size of c_o.

DEFINITION 4 (Best-Possible Approximate Functional Secrecy (BPAFS)). *Consider the following adversary game. The adversary $\mathcal{A}$ has c_e and can make up to q chosen input queries to c_o and has to return an ϵ-approximation of c_o. We say that a $c\mathcal{L}$ scheme is (t, q, ϵ, σ)-*BPAFS-OG *secure if the advantage of any $\mathcal{A}$ bounded by t operations is no more than σ better than the advantage of any adversary $\mathcal{A}'$ that tries to learn the black-box c_o with a priori knowledge that $depth(c_o) \leq depth(c_e)$ and $size(c_o) \leq size(c_e)$ through q queries. For oracle-less attackers (t, ϵ, σ)-*BPAFS-OL $\equiv (t, 0, \epsilon, \sigma)$-BPAFS-OL.

Note that leaking of the depth and size of c_o through c_e directly ties the overhead of locking to its security. The smaller the overhead of c_e in terms of depth and size, the tighter the bound on possible c_o solutions and a smaller hypothesis space for the attacker. We get the following from Definition 4:

PROPOSITION 5. *Without an oracle:* BPAFS-OL $\equiv$ AFS-OL.

PROPOSITION 6. *If the advantage of an attacker in learning c_o from queries is negligible,* BPAFS-OG $\equiv$ AFS-OG.

Proposition 6 is a rather important result. It shows that if in fact c_o happens to be a circuit that is difficult to learn using black-box learning, a BPAFS-OG-secure $c\mathcal{L}$ scheme will achieve the much stronger AFS-OG notion of security. Therefore, BPAFS-OG is really the best that we can expect from locking in the presence of an oracle for a given c_o.

PROPOSITION 7. *The* BPAFS-OG *attacker advantage σ is lower bounded by a function of the disagreement probability between c_e and c_o. $\sigma \geq poly(\left|\Pr_{i\in I, k\in K}[c_e^m(i,k) \neq c_o^m(i)] - \frac{1}{2}\right|)$*

H. Oracle-less (OL) Security

Oracle-less attacks or removal attacks try to discern the value of keys directly from the structure of c_e without any access to the oracle c_o [16], [17]. This is possible since certain locking schemes leak key information through the structure of the added key logic. e.g. if key-controlled XOR/XNOR gates are inserted in the circuit, key-bits can be recovered based on

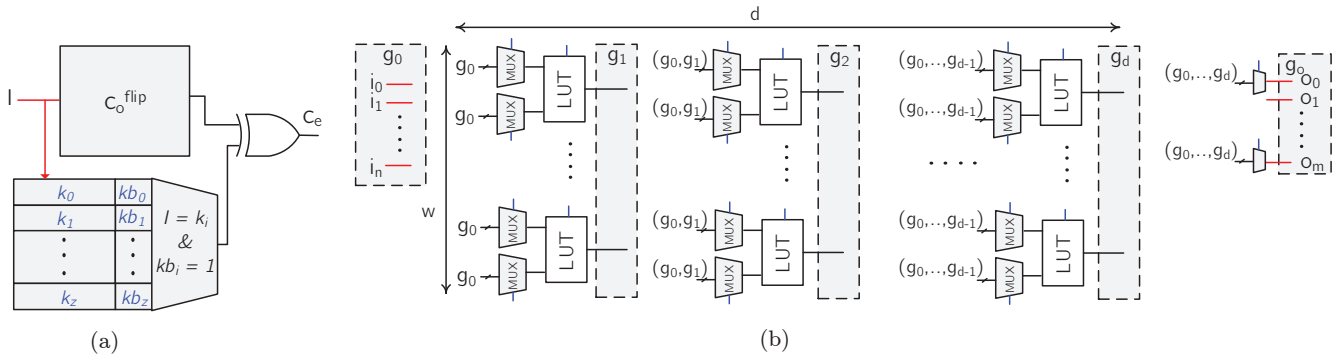

Fig. 3: (a) EFS-security can be guaranteed by flipping the output of c_o on p input patterns and correcting it using a conditional-look-up function implemented with TCAM logic with $z \geq p$ entries. (b) Universal circuit with depth d and width w. The blue wires are key-variables, and the red wires are input-outputs.

whether the key-gate is an XOR or an XNOR. We briefly discuss a couple of notes regarding BPAFS-OL herein.

PROPOSITION 8 (Structural Disassociation). *BPAFS-OL-security implies security regardless of the particular structure of c_e. i.e. if a scheme satisfies* BPAFS-OL*, the advantage of the adversary $\mathcal{A}$ which has access to c_e is negligibly different than the advantage of any other adversary $\mathcal{A}'$ that has access to a randomly chosen equivalent circuit implementing c_e.*

This condition relates somewhat to the notion of indistinguishability obfuscation ($i\mathcal{O}$) which is an operator that makes $i\mathcal{O}(c_1)$ indistinguishable from $i\mathcal{O}(c_2)$ when c_1 and c_2 implement the same function [9]. One may be tempted to use $i\mathcal{O}(c_e)$ to achieve OL-security. However, a stronger condition that locking needs to satisfy is that it should not leak the key through the functionality of c_e either. i.e. if an $i\mathcal{O}$ obfuscator is used to mangle c_e, it is still possible that $i\mathcal{O}(c_e)$ leaks information about c_o to the attacker through its functionality[4].

PROPOSITION 9 (Functional Disassociation). *If for a $c\mathcal{L}$ scheme the distributions $\{c_e\}$ and $\{c_{er}\}$ are indistinguishable in less than t operations with σ success rate, where c_{er} is the locking of a random circuit c_{or} which has the same size and depth as c_o, then the $c\mathcal{L}$ scheme is* (t, ϵ, σ)-BPAFS-OL.

The above functional disassociation condition appears to be stronger than BPAFS-OL meaning that it implies BPAFS-OL but the reverse is not obvious.

III. Candidate Constructions

In this section we first show how EFS-OG can be achieved. We then present a candidate construction that satisfies BPAFS-OG with polynomial overhead.

A. EFS *Security*

Subsequent to the SAT attack an array of methods were proposed to thwart the attack using "point-functions" [18]. Point-function $P_{x_*}(x)$ produces 1 on $x = x_*$ and 0 on all other inputs. These blocks can be added to c_o to flip its output for a particular set of input patterns. This can help create schemes where any oracle-guided attack has to query the entire input-space to resolve the key. However, it was shown in [19] how these blocks can be found and removed from the circuit allowing exact-recovery of c_o. Yasin et al. [20] were the first to propose removal-resilient locking/camouflaging schemes by first flipping c_o's output itself, and then using point-functions to "correct" the output on those inputs. We show how this technique can satisfy EFS-OG security if care is taken in implementing it.

THEOREM 2. *The following scheme is* EFS-OG *secure. The original circuit c_o is flipped on a set of input patterns $\mathcal{P}$ where $|\mathcal{P}| = p$. The conditional-lookup function $P(x, k = \langle\{k_1, k_2, ..., k_z\}, \{kb_1, kb_2, ..., kb_z\}\rangle)$ where $z \geq p$ is implemented which outputs 1 if $\exists k_i$ s.t $x = k_i$ and $kb_i = 1$. the scheme then produces $c_e(i,k) \leftarrow c_o^{flip}(i) \oplus P(i,k)$.*

In this scheme k_i represents a set of z secret patterns, and kb_i decide which ones are active in the comparison. The kb_is allow the scheme to leak no more than a bound on the number of flipped input patterns $z \geq p$. This is shown in Figure 3a. The lookup function can be implemented using TCAM logic with the kb_i signals masking each row.

Care must be taken in order to make sure the attacker cannot discern the location of the flipped patterns from the structure or functionality of c_o^{flip}. If comparators are used to perform the inversions and their structures is discernible from the rest of the circuit the flipped patterns can be leaked. Note that if the attacker suspects a location x to be in the set $\mathcal{P}$, he can test x's membership by comparing $c_o^{flip}(x)$ and $c_o(x)$. The set of input patterns $\mathcal{P}$ can be moved around to find a location that allows the flip-comparators to merge with other logic.

Yasin et al. in [20] and Shamsi et al. in [18] tried to cover a super-linear-size set $\mathcal{P}$ using functions $h(i,k)$ to increase the error rate. Shamsi et al. used a diversified-tree, and Yasin et al. used a hamming-distance block. Any such attempt however to produce a super-linear-sized set $\mathcal{P}$ with a function that is not one-way and has an statistically correlated onset risks leaking

[4] $i\mathcal{O}$ is a much hyped topic in theoretical cryptography. Candidate provably secure constructions of $i\mathcal{O}$ for large programs/circuits have impractical overhead. For smaller circuits the Reduced-Ordered-BDD of the circuit can be used as its perfect $i\mathcal{O}$. Careful multiple-rounds of random re-synthesis/restructuring/retiming may achieve a heuristic $i\mathcal{O}$ as well. One can think of $i\mathcal{O}$ as structural randomization/canonicalization. Our locking constructions in this paper do not rely on $i\mathcal{O}$ at all.

the patterns in $\mathcal{P}$ to the attacker.

B. BPAFS from Hypothesis Matching

Our construction for a BPAFS-OG secure $c\mathcal{L}$ scheme directly follows from the definition of BPAFS-OG. In the BPAFS game, we leaked to the adversary an upper bound on the size and depth of c_o. Given this information the hypothesis space $\mathcal{H}$ for the adversary that does not know anything else about c_o, is the set of all circuits of size $\leq size(c_e)$ and depth $\leq depth(c_e)$. If we were able to construct a circuit that can be programmed to implement this entire hypothesis space or a superset of it, the advantage of the attacker in attacking such a hypothesis is 0 in the BPAFS-OG game. We refer to this technique as *Hypothesis Matching*. This is because we are matching the strongest hypothesis that the attacker has on the original circuit c_o, with a circuit that implements at least a superset of all functions in that hypothesis space.

LEMMA 5. *(Hypothesis Matching) Let the strongest hypothesis of an attacker on c_o before querying it be $\mathcal{H}$ and $\mathcal{C} = \{c_e(i,k)|k \in K\}$. If $\mathcal{C}$ is not perversely leaking c_o[5], and if $\mathcal{H} \subseteq \mathcal{C}$, the advantage of this attacker in the* BPAFS-OG *game is zero.*

Does such a c_e exist? There are in fact many polynomially sized circuits that can be programmed to implement $\mathcal{D}$ where $\mathcal{D} = \{\text{set of circuits of size} \leq \text{s and depth} \leq \text{d}\}$.

LEMMA 6 ((d,s)-Universal Circuit). *There exists a circuit $u_{(d,s)}(i,k)$ over n input wires, and $l \leq s\log(s+n)$ key-inputs for which $\mathcal{C} \supseteq \mathcal{D}$, where $\mathcal{C} = \{u_{(d,s)}(i,k)|k \in K\}$ and $\mathcal{D} = \{$set of circuits of size $\leq s$ and depth $\leq d\}$.*

Proof. We prove this by constructing a $u_{(d,s)}(i,k)$. Without loss of generality we assume the DAG circuit basis to be the set of all 2-input Boolean functions. Per Figure 3b We start by placing the n input wires down on the circuit. We then add the layer g_1 of 4-key-bit LUTs. We then connect the $2|g_1|$ inputs of this layer to all the n inputs through n-to-1 key-controlled multiplexers with $\log(n)$ key-bits. We continue adding layers in this fashion except that for any layer after g_1 we not only connect the gates in g_i to g_{i-1} using MUXs, we also connect them to all previous layers all the way down to the input layer g_0. The number of key-bits in this circuit is:

$$l \in O\Big(|g_1|\log(n) + |g_2|\log(n+|g_1|)$$
$$+|g_3|\log(n+|g_1|+|g_2|)+...+|g_d|\log(n+|g_1|+...+|g_{d-1}|)\Big)$$
$$\leq O(d|g_d|\log(d|g_d|+n)) \leq O(d\frac{s}{d}\log(d\frac{s}{d}+n))$$
$$\leq O(s\log(s+n))$$

□

THEOREM 3. *The $c\mathcal{L}$ scheme that given n-input c_o outputs the n-input universal circuit $u_{(d,s)}$ where $d \geq depth(c_o)$ and $s \geq size(c_o)$ is $(\infty, 2^n, \epsilon, 0)$-*BPAFS-OG *secure.*

[5]This bars the case where the excess size of $\mathcal{C}$ with respect to $\mathcal{H}$, or the very encoding of $\mathcal{C}$ is used perversely to leak information about c_o.

Note that the above scheme is BPAFS-OL secure as well. OL-security follows from BPAFS-OG, however, an intuitive explanation for this is that we did not use any information from c_o to produce the locked circuit $u_{(d,s)}$ other than its depth and size, which are already known to the attacker.

C. Efficiency

While the above scheme is provably secure with polynomial overhead, it does not mean that it is practical. It results in a quadratic blow-up in the size of the DAG circuit. There are however some positive notes here. First, the proposed scheme perfectly matches the attacker hypothesis. Any deviation from the fully connected network of the universal circuit will shrink its hypothesis size with a rate that is not at all obvious from a theoretical stand-point. Hence in our experimentation we reduce the connectivity and profile changes in the attack success rate and runtime.

Second, the universal circuit proposed here is dominated by key-controlled MUX logic. Even though MUX gates require several other gates to implement in the complexity theoretic DAG model, in reality, the MUX logic simply means that the interconnect information is missing. In IC camouflaging this can be implemented with small metal-to-metal camouflaged vias and extra wiring. For split-manufacturing the very removal of top layers creates the interconnect ambiguity present in $u_{(d,s)}$. For logic locking, similar to IC camouflaging programmable vias can be used for improved overhead as they do not consume transistor area. However, the vias need to be programmed which requires additional programming circuitry. Shamsi et al. proposed a light-weight cross-bar programming architecture for this task [21]. Furthermore, the universal circuit resembles FPGA logic which can be embedded in an ASIC for locking.

D. Sequential Circuit Locking

It is possible to extend the framework described earlier to sequential circuits. We can allow the $c\mathcal{L}$ scheme to add state-elements to the sequential circuit c_o. Security is defined similar to the combinational locking problem, except ϵ-approximation is defined over the space of all possible input "sequences" rather than input patterns.

As for learning theory results, some FSMs can be modeled as Diterministic-Finite-Automata (DFA)s. We know that learning DFAs with membership queries is possible in polynomial time [22], however, the learning complexity grows with the number of possible input patterns (i.e. the size of the input alphabet in the DFA) which is exponential for FSM circuits [23]. It is possible to unroll the sequential circuit up to some number of rounds and apply the combinational model to it. The universal circuit for sequential circuits can be constructed as well by replacing the next-state and output logic with universal circuits. We leave an extended analysis of this to future work.

IV. EXPERIMENTS

We now present our experimentation results. In our experiments we used the SAT attack algorithm [7] implemented in an in-house developed C++ framework optimized with O3.

TABLE I: ISCAS and MCNC benchmark set

bench	#ins	#outs	#gates	bench	#ins	#outs	#gates
c432	36	7	160	i7	199	67	1315
c499	41	32	202	c3540	50	22	1669
i4	192	6	338	k2	46	45	1815
c880	60	26	383	dalu	75	16	2298
c1355	41	32	546	c5315	178	123	2307
apex2	39	3	610	i8	133	81	2464
c1908	33	25	880	c7552	206	107	3512
i9	88	63	1035	seq	41	35	3519
ex5	8	63	1055	ex1010	10	10	5066
c2670	157	64	1193	apex4	10	19	5360
i7	199	67	1315	des	256	245	6473

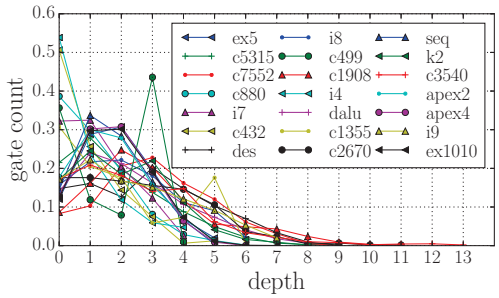

Fig. 4: Depth profile of the benchmark circuit set. Most circuits have a short depth and a cone-shaped topology, meaning that the percentage of gates in each layer drops as we get closer to the output.

We used the Glucose SAT-solver without its simplification routines. Our SAT attack implementation keeps track of a most recent valid key, so that in case the time deadline arrives, the attack will return an intermediate key before terminating. The key satisfies all input-output pairs queried so far. This is somewhat similar to AppSAT's intermediate key extraction [7]. In addition to the SAT attack we used the hill climbing attack from [24]. The hill climbing attack is based on using a simulated-annealing algorithm to flip key-bits to optimize for an error metric calculated from randomly querying c_e and comparing it to c_o. The hill climbing attack is surprisingly successful in learning universal circuits when the circuit is too large for SAT attacks. In addition to the C++ framework, we used the `scikit` framework in python for some learning subroutines. All tests were run on a server with 96 AMD EPYC cores at 1.9Ghz with 256 GB of RAM running Linux.

We used a set of ISCAS and MCNC benchmark circuits from [6]. The benchmark circuits were resynthesized using ABC to a toy library with 12 gates. The statistics of these benchmark circuits can be seen in Table I.

A. Oracle Learnability

We first perform black-box learning of c_o to get an idea of the learnability rate of different benchmark circuits. We used two decision-tree learning algorithms ID3 and CART along with a neural network (NN) using default parameters from `scikit` (e.g. hidden layer size: 100). We configured the learning algorithms as a binary classification problem and tried to learn the functionality of each output bit separate of the other bits. We used 1000 random input patterns with 80% dedicated to training and the other 20% to testing. The recovered accuracy for different benchmark circuits is shown in Figure 5 where each point is the accuracy of a different output of the circuit. It is quite common for some output bits to be learned with much better accuracy than others. As can be seen, the majority of outputs of the benchmark circuits can be learned with accuracy significantly above 50%.

While the main appraoch for proving learnability of Boolean functions is Fourier-based methods, in practice, more modern learning algorithms such as decision-trees and neural networks will outperform a direct implementation of the LDA or KM routines [25] which is what we observed in our experimentation as well. We defer more in-depth practical utilization of Fourier schemes to future work.

In addition, note that decision-tree and neural network learning methods do not return a small and fast Boolean circuit. Oliveira [26] proposed various algorithms for learning Boolean circuits directly which he showed to outperform ID3 and CART and return compact circuit representations. As we will discuss shortly, a deobfuscation attack (hill climbing or SAT attack) on the universal circuit with respect to an oracle, is in itself a learning algorithm that returns a Boolean circuit. If the SAT attack on a universal circuit c_e that includes c_o succeeds, c_o is recovered from oracle queries alone.

B. The Universal Circuit

Before describing our universal circuit model it is good to look at the depth profile of the benchmark circuit set that we used represented in Figure 4 as gate-percentage per layer. As can be seen these circuits all have a somewhat shallow and cone-like structure meaning that the number of gates in each layer drop as we get closer to the output. This fact can be used in the construction of the universal circuit to reduce its complexity by dropping the layer width as we get closer to the output creating a cone-like structure.

Given the above, our universal circuits are characterized by the following parameters: 1) number of inputs n and outputs m; 2) width of layers w; 3) depth d; 4) back-step b, which is an integer that decides how many of the layers before g_i are to be part of the candidate set for connecting to g_i through MUXs. For back-step 1, g_i will only connect to wires in its previous layer and hence the candidate set is $S_i = \{g_{i-1}\}$. A full back-step means that each g_i can be connected to all previous layers, $S_i = \{g_0, ..., g_{i-1}\}$. 5) Connectivity cn. Once back-step decides the set of candidates for the MUX gates, connectivity which is a value between 0 and 1 decides the proportion of S_i that will be selected for routing through MUXs. It is possible to pick from S_i randomly, however, to avoid disconnected networks we used a window selection technique that connects each wire $w_j \in g_i$ to $\{w'_j, ..., w'_{(j+cn*|S_i|) \mod |S_i|}\} \subseteq S_i$. We specify this universal circuit with $univ(n, m, w, d, b, cn)$.

C. SAT Attacks

We performed 1728 SAT attacks on $u_{(d,s)}(i,k)$-$c_o(i)$ pairs by sweeping the above parameters according to the x-axes of the plots in Figure 6. The original circuit c_o in these tests was generated by randomly assigning a key to the universal circuit and resynthesizing the circuit and picking only c_os that

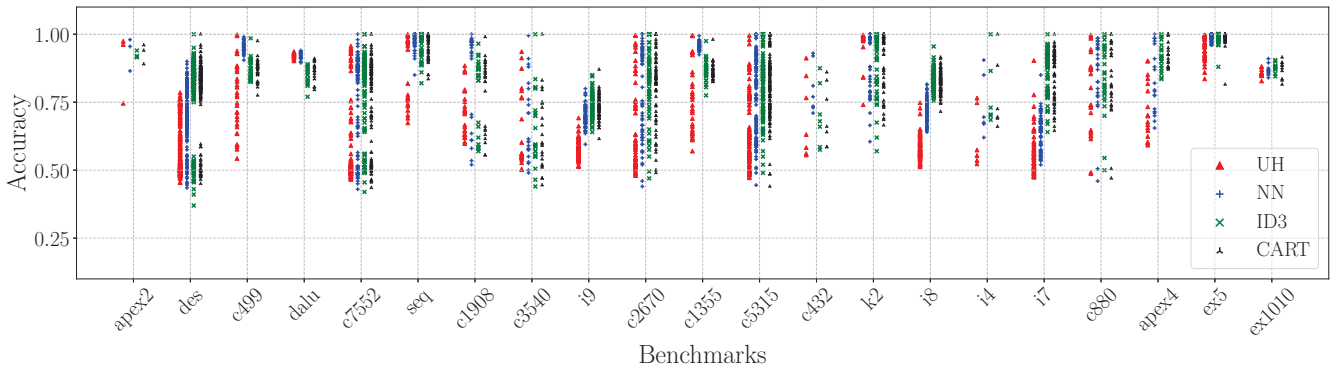

Fig. 5: Black-box learning of the oracle circuit itself c_o with two decision-tree learning schemes (ID3 and CART) and a neural network (NN). UH (red triangles) points are the best hill climbing attack on a universal circuit which demonstrates how learning the universal circuit in practice is no more successful than direct black-box learning of c_o with advanced machine learning schemes.

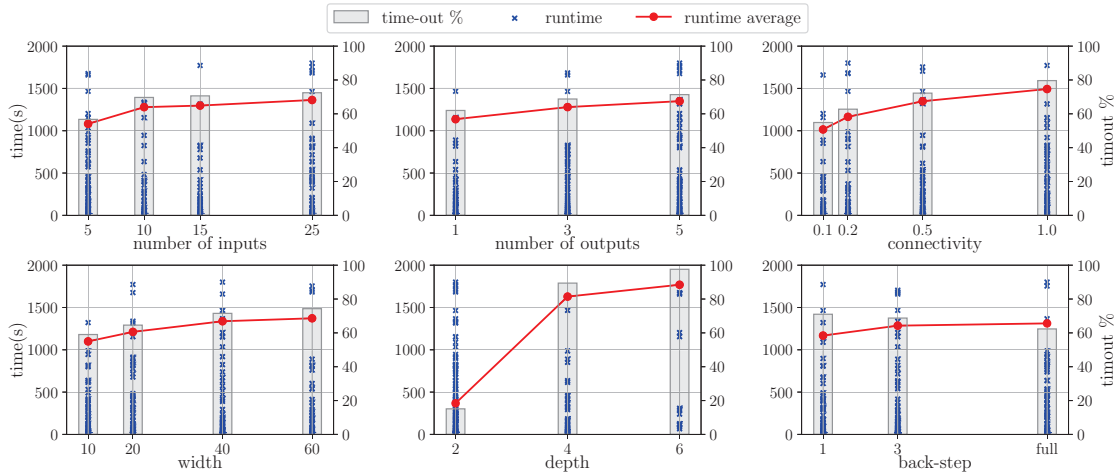

Fig. 6: SAT attack on universal circuits as c_e of various metrics, with respect to randomly chosen c_os. The c_os are generated by randomly assigning a key to $u_{(d,s)}(i,k)$ and resynthesizing using ABC to recover a non-constant c_o. The data points in all these plots are the same tests, viewed according to different variable sweeps.

are non-constant. The time-out for the attack was set to 30 minutes to allow for all the tests to complete. In Figure 6 the 1728 data points are viewed along different dimensions to analyze the effect of different parameters. The bars represent the percentage of tests in each category that reached time-out.

As can be seen from the data, connectivity and back-step generally increase the complexity of the SAT attack, however, not by a great deal. This signals that we may be able to move away from the fully connected universal circuit without much loss in terms of practical security. Depth seems to have a particularly significant effect on attack complexity. An interesting note is that each blue cross in Figure 6 shows a circuit c_o which can be learned from oracle queries with only knowledge of a bound on its size and depth eliminating the need for knowing c_e all together.

D. Hill Climbing

The size of the universal circuit can get very large even with moderate parameters. This quickly causes the SAT solver in the SAT attack to slow down significantly. On the other hand, the hill climbing attack has a constant memory footprint and is much more scalable for larger circuits. Figure 7 shows a comparison of the hill climbing attack versus the SAT attack on two different universal circuits while randomly varying only c_o. Each test was run 3 times for 30 minutes and the best results were selected.

As can be seen from Figure 7 for the larger universal circuit where the SAT attack fails, the success rate of the attack heavily varies with c_o. This again reiterates the message of this paper that OG-security is heavily dependant on c_o itself. The universal circuit simply ensures that the learning rate of the given c_e is not better than directly learning c_o. It can be seen that c_o circuits that are difficult to learn under the hill climbing attack were also consistently difficult to resolve in different runs of the SAT attack indicating a genuine relation between the original circuit c_o and the attack complexity regardless of the locked circuit c_e.

We also used the hill climbing attack to learn the benchmark circuits in Figure 5 from universal circuits. The red triangles represent the best accuracy results obtained by hill climbing on universal circuits for each output. It can be seen that almost no hill climbing attack outperforms machine learning schemes on the black-box. This is consistent with the BPAFS-OG criteria

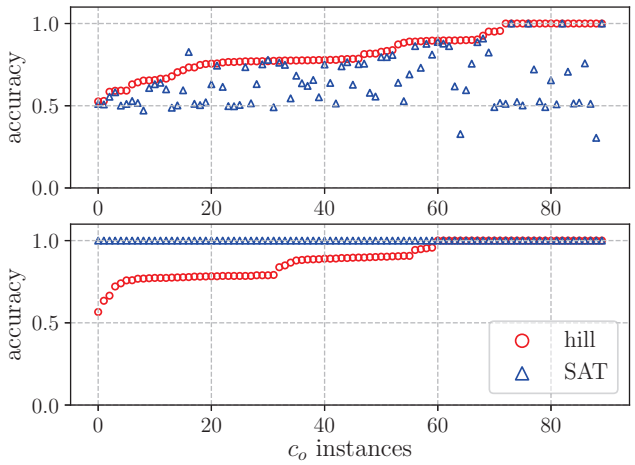

Fig. 7: The x-axis represents the c_o instance index. All tests in the upper plot use $c_e = \text{univ}(15, 1, 20, 4, 1, 0.2)$ and the lower plot use the $c_e = \text{univ}(10, 1, 15, 3, 1, 0.2)$ universal circuits. Hence the variation is primarily due to c_o. For the smaller universal circuit the SAT attack concludes on all runs. For the larger universal circuit the hill climbing attack outperforms the SAT attack in the same time.

that states that learning the universal circuit c_e should not be significantly better than black-box learning of c_o.

V. Conclusion

In this paper we presented a formal approach to the problem of netlist disambiguation. We defined the problem of circuit locking and different notions of security. We showed that approximation-resiliency under oracle-guided attacks depends partly on the original circuit c_o itself. We presented secure constructions for exact-recovery resiliency, and a relaxed notion of best-possible approximation-resiliency using universal circuits. Our experimental analysis showed the relation between various parameters of the secure constructions and practical security. Some open problems which remain are: 1) learnability of sequential circuits under oracle-guided attacks, 2) best-possible approximation-resiliency with linear overhead rather than poly-log, 3) utilization of Fourier spectrum techniques, and 4) automatically detecting which c_o circuits can be locked with smaller overhead and more security.

Acknowledgement

This work was partially supported by DARPA.

References

[1] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM Conf. on Computer & Communications Security*, 2013.

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proc. Design, Automation and Test in Europe*, DATE '08, pp. 1069–1074, 2008.

[3] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[4] B. Hill, R. Karmazin, C. Otero, J. Tse, and R. Manohar, "A split-foundry asynchronous fpga," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 1–4, 2013.

[5] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes.," in *Network and Distributed System Security Symposium (NDSS)*, 2015.

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pp. 137–143, IEEE, 2015.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, pp. 46–51, 2017.

[8] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Proc. IEEE Great Lakes Symp. on VLSI*, pp. 179–184, ACM, 2017.

[9] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: deniable encryption, and more," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pp. 475–484, ACM, 2014.

[10] M. T. Rahman, Q. Shi, S. Tajik, H. Shen, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "Physical inspection & attacks: New frontier in hardware security," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pp. 93–102, IEEE, 2018.

[11] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 9, ACM, 2017.

[12] M. Yasin, A. Sengupta, B. C. Schafer, Y. Makris, O. Sinanoglu, and J. J. Rajendran, "What to lock?: Functional and parametric locking," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 351–356, ACM, 2017.

[13] R. Karmakar, N. Prasad, S. Chattopadhyay, R. Kapur, and I. Sengupta, "A new logic encryption strategy ensuring key interdependency," in *VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID), 2017 30th International Conference on*, pp. 429–434, IEEE, 2017.

[14] F. Tehranipoor, N. Karimian, M. M. Kermani, and H. Mahmoodi, "Deep rnn-oriented paradigm shift through bocanet: Broken obfuscated circuit attack," *arXiv preprint arXiv:1803.03332*, 2018.

[15] R. O'Donnell, *Analysis of boolean functions*. Cambridge University Press, 2014.

[16] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism," *arXiv preprint arXiv:1703.10187*, 2017.

[17] P. Chakraborty, J. Cruz, and S. Bhunia, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," *arXiv preprint arXiv:1809.10743*, 2018.

[18] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2019.

[19] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.

[20] M. Yasin, A. Sengupta, M. Ashraf, M. Nabeel, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM Conf. on Computer & Communications Security*, pp. 1–1, 2017.

[21] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pp. 147–152, ACM, 2018.

[22] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.

[23] S. Eisenstat and D. Angluin, "Learning random dfas with membership queries: the goodsplit algorithm," in *ZULU workshop organised during ICGI*, p. 12, 2010.

[24] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in ic piracy with test-aware logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[25] A. Drake and D. Ventura, "Search techniques for fourier-based learning.," in *IJCAI*, pp. 1040–1045, 2009.

[26] A. M. L. de Oliveira, *Inductive learning by selection of minimal complexity representations*. Electronics Research Laboratory, College of Engineering, University of California, 1994.