In Praise of Exact-Functional-Secrecy in Circuit Locking

Kaveh Shamsi, *Member, IEEE,* Yier Jin, *Senior Member, IEEE,*

***Abstract*—Many logic locking schemes have been proposed and subsequently broken in recent years most notably by oracle-guided SAT-solver-based attacks. This has in part been due to a lack of formal definitions of security. Recent work has however taken the first steps towards this by defining some notions of security. One such notion, exact-functional-secrecy (EFS) is satisfied as soon as the attacker is not able to learn the *precise* functionality of the original circuit. This is less stringent than the approximate-functional-secrecy (AFS) notion of security which captures approximation-resiliency. This paper focuses on EFS. We present first a novel SAT-based attack that can automatically divide the deobfuscation of a locked circuit into two different processes: a) deobfuscating high-activity/entropy nets which contribute to AFS and are best handled by a few queries and heavy SAT-solving, and b) deobfuscating low-activity nets which require many useless queries in search of a few rare informative queries. The attack, called the rare-fast-querying (RFQ) SAT attack, guarantees key-correctness for logic outside of low-activity cones, and is not exclusive to a specific low-activity locking scheme. We show how the RFQ attack can under some conditions, avoid exponential querying altogether. Given the insight from this attack, we then present a deeper look into EFS and discuss simple techniques to achieve always-exponential EFS with bearable overhead. We show how one can take advantage of the abundance of comparator logic at the RT-level of control-oriented designs to achieve EFS with even less overhead via absorbing existing structures.**

***Index Terms*—Hardware Security, Logic Obfuscation, IC Camouflaging, Logic Locking.**

I. Introduction

The high costs of maintaining semiconductor nano-fabrication in the sub-100nm regime have over the years resulted in a separation of design and fabrication. With more and more companies from various sectors investing in application-specific ICs (ASIC), the fabless business model in which designers outsource fabrication to consolidated foundries has been on the rise. An untrusted foundry in such a setting raises several security and privacy concerns including 1) reverse engineering for IP theft, 2) overproduction, and 3) malicious modification of the design. In addition to untrusted foundries, end-user microscopy-based reverse-engineering of fabricated ICs can cause similar pains for IP holders and designers [1].

Three broad categories of techniques exist for hiding the design of a circuit from untrusted foundries or end-users. 1) IC Camouflaging: in which ambiguous-under-microscopy nano-structures are dispersed throughout the layout to hinder end-user reverse-engineering while providing no protection against foundries [2]. 2) Split-manufacturing: which splits the design (typically by metal layer) and fabricates the less costly part in a lower-end trusted foundry [3]. 3) Logic locking [4]: in which (one-time) programmable (ambiguous-under-microscopy) elements are added to the circuit, ensuring that correct operation requires a post-fabrication configuration with a secret bit-string (aka a secret *key*, hence the name locking).

Logic locking, due to its protection against both foundry and end-users without requiring a trusted foundry, has become increasingly appealing. However, it is the hardest to secure against attacks with low overhead. This is because programmable elements need programming structures that leak the location of ambiguity to the attacker and increase overhead. Furthermore, it is difficult to flood the empty spaces in the layout with programmable elements, an approach that produces very strong low-overhead IC camouflaging [5].

All three schemes, especially locking, can be modeled mathematically as transforming a Boolean (possibly sequential) circuit to an augmented/locked/obfuscated Boolean circuit with added hidden/key variables, with the goal of the attacker being to obtain the original circuit (find the correct key) from the keyed one. The question of security under this model heavily relies on a) the threat model, and b) the definition/notion of security itself. The main threat models are 1) oracle-guided (OG) [6]: where the attacker, in addition to the obfuscated circuit, has access to a black-box implementation of the original circuit, 2) sequential-oracle-guided (SOG) [7]: where the oracle has inaccessible state-elements 3) oracle-less (OL): where the attacker has access only to the ambiguous design[1]. Any kind of access to an oracle can allow for the attacker to use input-output patterns extracted from the oracle to disambiguate the obfuscated circuit.

Under the above threat models, some notions/definitions-of-security (what it means mathematically for a locking scheme to be secure) were defined in [8]. These notions focus on keeping the functionality of the original circuit secret, rather than the key per se. One is exact-functional-secrecy EFS where the attacker is barred *only* from *perfect* learning of the functionality of the original circuit in a given time. EFS hence allows for arbitrary approximation of the original circuit. A much

Yier Jin is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, Florida, USA (email: yier.jin@ece.ufl.edu).

Kaveh Shamsi is with the Department of Electrical and Computer Engineering, University of Texas at Dallas, Richardson, Texas, USA (email: kaveh.shamsi@utdallas.edu).

Corresponding Author: Yier Jin.

[1]One can define probing attacks by defining a new threat model t-probed-oracle-guided tpOG in which an attacker can probe t "arbitrary" locations in the circuit. It is easy to show that exponential-security under tpOG for $t \geq 1$ is impossible to achieve without exponentially many keys as the attacker can simply probe key bits or their immediate fanouts. A slightly more hopeful threat model is one where the attacker can probe t locations from a subset of nets W. If W is functionally distant enough from key-bits security may be achievable. Either way without physical package-level protection against probing (active/passive-shielding etc.), a probing-capable attacker is very hard if not impossible to defeat with algorithmic techniques and hence outside the scope of (un)locking algorithms research and this paper.

stronger notion per [9] is approximate-functional-secrecy AFS in which the attacker is barred from approximating the original circuit beyond a given accuracy limit.

The strongest oracle-guided attacks are SAT-based attacks [10], [11], which use iterative SAT-solver calls to trim the key hypothesis space and recover a guaranteed correct key. These attacks have been successful in defeating the majority of existing low-overhead gate-based locking/camouflaging. To thwart these attacks several "point-function" schemes were proposed [12], [13], [14] which use functions with sparse truth-tables to create exponential minimum-query-counts for oracle-guided attacks. These schemes were subsequently attacked with removal attacks [15], [9] that find these structures in the circuit and remove them, and approximation attacks [16], [17], [9] which disregard the point-function and recover the remainder of the circuit in cases where the point-function schemes were mixed with traditional schemes.

In this paper, we take a deeper look at EFS and attacks against it. The paper delivers the following:

- We show that in general, locked circuits can have either *query hardness* present for instance in point-function schemes with low-activity nets in which the attacker can recover the function if they simply make a large number of queries with little book-keeping; or alternatively, the locked circuit can have *algebraic hardness* in which very high-entropy/nonlinear/deep circuits do not need many queries, but the system of equations resulting from the few queries is difficult to solve. For the latter part, we use an argument from pseudo-random-functions (PRFs) in cryptography.
- We present an attack that unlike existing approximate attacks which either query blindly like AppSAT [16], or are tailored to specific point-function schemes such as DDIP[17], SigSAT[18], and kDIP[9], can in fact target any low-activity net (including ones inherent to the original circuit), and avoid adding exponentially many copies of useless query conditions to the SAT-solver for such nets. Instead, it fast-queries the fanin of these low activity nodes until it encounters rare and interesting queries which are then added to the solver (hence called the rare-fast-query (RFQ) attack). This not only speeds up attacks on low-activity (e.g. point-function) locked circuits, it creates a mechanism to detect such nodes in the circuit and separate the circuit nodes into those with query hardness (EFS-like) and those with algebraic hardness (AFS-like).
- Using the formal definition of EFS we show how in certain cases the RFQ attack can avoid exponential querying with early termination, and then propose simple defense tricks to avoid these conditions and achieve always-exponential security for EFS schemes. Using these tricks we propose a new EFS scheme that takes advantage of inherent comparator logic at the RT-level for always-exponential-query EFS locking. We showcase our EFS locking approach on a control-heavy RS232 Verilog design.

The paper is organized as follows. Section II covers some preliminaries. Section III presents the RFQ attack and its experimental results. Section IV presents the improved EFS schemes along with a case study. Section V concludes the paper.

II. Preliminaries

A. Definitions

Following [8], Circuit Locking $c\mathcal{L}$ can be defined as follows:

Definition 1 (Combinational Circuit Locking ($c\mathcal{L}$) scheme)**.** *A Combinational Circuit Locking ($c\mathcal{L}$) scheme for a family of combinational circuits C_o is a probabilistic polynomial-time (PPT) algorithm Lock^{C_o} that takes security parameter λ and an original circuit $c_o \in C_o$, 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 $O = \mathbb{F}_2^m$, then $c_e : K \times I \to O$ where $K = \mathbb{F}_2^l$.*
- *(Correct functionality under correct key) We have $\forall x \in I, \quad c_e(k_*, x) = c_o(x)$.*
- *(Polynomial overhead) We have $size(c_e) \leq poly^2(size(c_o))$ and $depth(c_e) \leq poly(depth(c_o))$.*

This definition is sufficient for directly modeling locking, and with some polynomial work camouflaging and split-manufacturing. The PPT assumption on Lock does not disqualify locking schemes that use SAT/BDD or other worst-case-exponential subroutines since such schemes will work only on circuits with subexponential SAT/BDD complexity which can be encoded in C_o. Additional a priori information on the original circuit can also be encoded in C_o.

Under Definition 1 two notions of security proposed in [8] focusing on hiding the functionality of c_o rather than key-recovery (functional-secrecy implies key-security). In the following definition a computational adversary $\mathcal{A}$ is playing an *attack-game* with a challenger that initially runs $(c_e, k_*) \leftarrow \mathsf{Lock}^{C_o}(c_o, \lambda)$ and hands c_e over to $\mathcal{A}$. OG/OL are used to denote access/no-access to an oracle of c_o:

Definition 2 (Approximate Functional Secrecy (AFS))**.** *The adversary $\mathcal{A}$ has c_e, can make up to q chosen input queries to c_o and has to return an ϵ-approximation[3] of c_o to win. 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. For OL attackers, (t, ϵ, σ)-AFS-OL $\equiv (t, 0, \epsilon, \sigma)$-AFS-OG.*

Definition 3 (Exact Functional Secrecy (EFS))**.** *equivalent to AFS but with an $\epsilon = 0$:*
$((t, q, \sigma)$-EFS $\equiv (t, q, 0, \sigma)$-AFS-OG).*

The defender would want σ to be exponentially small (negligible) and t to be exponentially large. In such a case, observing q chosen queries on the oracle does not help the attacker much with predicting the value of c_o on any of the non-queried points. In EFS the attacker only wins, if he/she

[2] *poly(x)* denotes a polynomial expression in x

[3] An ϵ-approximation of f differs from f on at most an ϵ fraction of the input space.

can predict precisely the truth-table of c_o. Recovering a circuit c_o^+ perfectly equivalent c_o satisfies this as well.

The AFS-OG definition of approximation-resiliency is quite strong as it essentially demands that the attacker not be able to learn the functionality of c_o at any rate significantly faster than entry-by-entry querying of the oracle for c_o. Achieving this with exponentially large t was shown to be impossible for many classes of Boolean functions especially small circuits since simply learning these classes of functions from their oracle queries achieves super-linear learning rates [8]. For instance, consider the simple case of a highly sparse function. i.e. the output is 1/0 for $1-\Delta$ proportion of inputs and 0/1 for the rest where Δ is small. The attacker can make a few queries and simply decide the rest of the truth-table entries to be all equal to the majority of the observations from these queries. This all-0 or all-1 Boolean circuit is an Δ-approximation of the original sparse function even though the attacker did not even look at the locked circuit c_e. This simple case also proves why a circuit with signal-probability $\frac{1}{2}-\Delta$ at its output, can achieve no better than $(\text{poly}(\frac{1}{\Delta},\frac{1}{\epsilon},\delta,q),q,\epsilon,\delta)$-AFS. As such, exponentially secure AFS attainability is constrained by the original circuit c_o's learnability itself. A more relaxed weaker notion of approximate security called best-possible approximate-functional-secrecy BPAFS-OG was defined in [8] which avoids this impossibility result and is achievable via universal circuits.

In the remainder of the paper, we focus mostly on EFS and we may simplify the notation by saying "exponential" EFS to mean that q and consequently t in (t,q,σ)-EFS are exponentially large in the number of key-bits and σ is exponentially small. We shall also drop the OG qualifier as we mean EFS-OG unless otherwise specified.

B. The SAT Attack

The SAT attack seen in Algorithm 1 [11], [10] is a practical and generic oracle-guided attack using modern SAT solvers that upon termination returns a guaranteed correct key. It starts by building a miter circuit $M \equiv c_e(k_1,x) \neq c_e(k_2,x)$. Satisfying the miter returns a discriminating input pattern (DIP) $\hat{x}$ and two different keys $\hat{k_1}$ and $\hat{k_2}$. $\hat{x}$ is queried on the oracle getting $\hat{y} = c_o(\hat{x})$ and the resulting input-output (IO) observation pair is added to the miter formula. The process repeats until the miter+IO-conditions is UNSAT at which point the IO-conditions identify a correct key if $c_o \in C_e$ where C_e is the possible-function-space of the locked circuit; $C_e = \{c_e(k,x) | k \in K\}$.

Algorithm 1: Given oracle access to c_o and the circuit c_e returns a guaranteed correct key $k_* \in K_*$ if $c_o \in C_e$

```
Function SATAttack(c_e, c_o as black-box):
    F ← true
    M ← c_e(k_1, x) ≠ c_e(k_2, x)
    while F ∧ M is solvable do
        x̂, k̂_1, k̂_2 ← SAT(F ∧ M)
        ŷ    c_o(x̂)
        F    F ∧ (c_e(k_1, x̂) = ŷ) ∧ (c_e(k_2, x̂) = ŷ)
    satisfy F with k̂_1 and k̂_2
    return k̂_1 as a correct key k_*
```

AppSAT [16] and DDIP [17] are approximate SAT attacks in that they can exit early if a sufficiently good approximation of the key/functionality is achieved throughout the attack loop. AppSAT uses random sampling every d DIP iterations to measure the current key-hypothesis' error and exits at a specific error threshold. DDIP modifies the miter to capture "DIPs that disqualify no less than 2 keys". This becomes unsatisfiable and terminates the attack as soon as the disqualifying power of the remaining DIPs falls below 2 precisely. This is relevant to breaking point-functions schemes that are discussed shortly.

C. Point-Function Schemes

A (single)-point-function P_{x_*} on n-bit vectors/inputs is simply a comparator function that outputs 1 when the input is equal to a specific pattern x_* and 0 everywhere else. A multi-point-function $P_{\{x_*\}}$ will output 1 if the input is equal to any member of a set of vectors $\{x_*\}$ and 0 otherwise.

Such a point-function will have a very low-activity output, i.e. the probability of the output activating is $m/2^n$ for an m-point-functions. Various schemes have been proposed that use these functions for locking. Fig. 1a shows AntiSAT [12] in which two complementary point-function are ANDed and cancel each other out when the two key vectors are equal. Fig. 1b shows a general stripped-functionality-logic-locking (SFLL) [19], [14], [20] scheme in which a low-activity function $F(x_*,x)$ is used first to flip the functionality of the circuit, and after resynthesis of the flipped logic, $F(k,x)$ is used to restore the output yielding the correct key x_*. The attacker that finds and removes $F(k,x)$ from c_e gets the functionally-stripped circuit instead of the original circuit c_o.

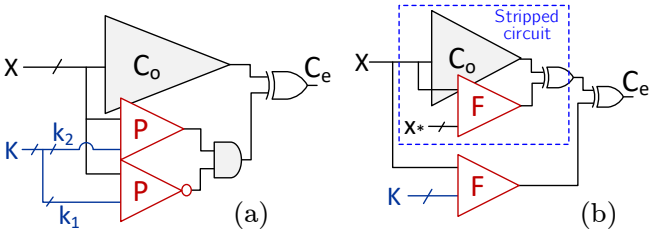


Fig. 1: Two point-function EFS schemes. (a) AntiSAT (b) SFLL.

III. The Rare-Fast-Query (RFQ) Attack

A. Deobfuscation Hardnesses

Consider the locked circuit $c_e(k,x)$. What key-recovery hardness means in this context, is that given arbitrary queries of the form $c_e(k_*,x_i) = y_i$, it is hard to find k_*. This, in fact, carries a certain resemblance to an array of cryptographic functions, most notably pseudo-random-functions (PRF).

A q-query secure PRF is a function of two input vectors $f(k,x) : K \times X \rightarrow Y$, for which given random secret key k_*, $f(k_*,x)$ is computationally indistinguishable from a randomly selected function with $|x|$ inputs given q chosen queries of x on f. Computational indistinguishably is defined as no efficient adversary having nonnegligible distinguishing success rate[4].

[4]The definition of "efficient" and "negligible" vary from application to application but one typically defines efficient as an algorithm with t operations (and queries in the case of PRF) where t is poly-bounded. Negligible is typically defined as at least super-polynomially small. For AES-128 which is a pseudo-random-permutation (PRP), key-recovery success rate is next to zero for adversaries running in time less than 2^{120}.

The main approach for building efficient PRFs and other such high entropy cryptographic functions is to mix linear (XOR in the Boolean finite field) with nonlinear (AND/OR) operations along with shuffles and permutations of bits to create a "round" function that has decent entropy but is not by itself secure. Then the round function is iterated (applied back-to-back) multiple times to build a deep/wide/nonlinear circuit. Note that this is a heuristic endeavor guided by decades of practical cryptanalysis experience rather than a provably-secure or provably-optimal approach. AES for instance, which is a modern standardized block cipher and can be used as a PRF, is composed of 10 such rounds.

PRF indistinguishability from a random function requires that both finding its key k_*, and approximating any output bit of the function be difficult given arbitrary chosen queries of the form $f(x_i, k_*)$. This implies that the system of equations formed from q queries of the form $f(x_i, k) = y_i$ for a PRF must be hard to solve computationally.

For the purpose of our discussion, we are interested in an important result that can be derived regarding the query-complexity of breaking a PRF. We can prove that learning k_* given the ability to query $f(k_*, x)$ does not and cannot need exponentially many queries if f is a PRF.

Lemma 1. *Given a PRF $f(x, k) : \{0,1\}^n \times \{0,1\}^l \rightarrow \{0,1\}$, and a secret key k_* a system of equations derived from $O(l)$ adaptive chosen queries on x of the form $\{f(x_i, k_*) = y_i\}$ will have a unique solution. Therefore, computationally solving this system for k_* must be hard for the PRF to be secure.*

Proof. For f to be a PRF, every output bit of f will have to be indistinguishable from a randomly selected function from all possible functions from $\{0,1\}^n$ to $\{0,1\}$. If we fix x in $f(k, x)$ to $\hat{x}$ and look at the distribution of some bit of $f(k, \hat{x})$ over $k \in \{0,1\}^l$, this distribution should be computationally indistinguishable from a random Boolean variable. Otherwise, an adversary can use $f(k_*, \hat{x})$ to distinguish f from a random function. Therefore, $\{f(k, \hat{x}) | k \in \{0,1\}^l\}$ must statistically have an equal number of 0s and 1s. This means that $f(k_*, \hat{x}) \neq f(k, \hat{x})$ for half of the possible k, all of which can be disqualified by the query. For an l bit key therefore, $O(l)$ queries must disqualify all bad keys and identify a k_*. □

The above lemma highlights a core point regarding PRFs, which is that their learning hardness cannot come from query complexity and instead should come from algebraic complexity, i.e. the small system of equations being hard to solve, rather than relying on the size of the system exploding.

While a PRF is great at hiding k_*, a fundamentally different way for hiding the secret input of a function from its output observations is to simply have the secret input not affect the output that much. Consider the point function $P_{k_*}(x)$ that activates only when x is equal to k_*. If an attacker wants to find the value of k_* from queries of $P_{k_*}(x_i)$, they will have to perform in the worst case $2^n - 1$ queries until a 1 is observed at the output at which point k_* is revealed. This is the opposite of the PRF case in several regards. The output of the point-function is highly skewed (low-activity) as opposed to the PRF which has a balanced output (high-activity). The query complexity of the point-function is exponential in the number of key bits while the PRF is linear in the same number of bits. The algebraic complexity of the point-function defined loosely as its nonlinearity/invertibility/circuit-complexity is very low since a SAT solver can easily come up with input guesses for even a large point-function (AND-tree comparator) in less than a fraction of a second, as opposed to the large/deep/nonlinear PRF which can overwhelm the solver.

Despite these differences, both functions are in fact hiding the secret input k_*. Whereas the PRF unlike the point-function is not approximable, in some sense the point-function is hiding the most important fact about itself, which is the single point on which it is activated. Hence, depending on the original circuit, both these cases can provide intuitively what we expect from locking, that is, hiding important features of the functionality of the original circuit.

B. Low-Activity Wires

Given the dichotomy between query hardness and algebraic hardness, we begin first by devising an attack that intends to separate these cases during deobfuscation. An important property of point-function-driven query complexity is the presence of nets with highly skewed signal probability in the circuit. In general, the presence of such skewed wires typically leads to high query complexities and causes problems for the baseline SAT attack. When facing a query-heavy locked circuit $c_e(k, x)$ with a small onset of size h, the baseline SAT attack will iteratively come up with new DIPs on x, query them on the oracle, and then add two copies of $c_e(k, x_i)$ to the miter circuit. This process continues until either all x_i patterns are queried and the output never activates, or the output activates on a few patterns and the attack may stop. This results in a super-linear increase in runtime and a linear increase in memory of the attack. This is while we know that fundamentally, the point-function deobfuscation problem is one of "scan"-querying the entire input range and waiting for the rare occasion where the output activates. This in theory should have linear runtime in the number of queries $O(2^n)$ and a memory footprint linear in the size of the rare inputs of the low-activity function $O(h)$. The RFQ attack tries to achieve these limits.

Our RFQ attack relies on skewed wires therefore as indicators of a query-heavy scenario signaling to the attack that the baseline SAT approach may be suspended for such wires. We can use this because we know that for query complexity q we need $\left|\mathsf{Pr}_{x \in I, k \in K}[c_e(k, x) \neq c_o(x)] - \frac{1}{2}\right| < L(1/q, 2^n, 2^l)$. While the precise behavior of L is somewhat of an open problem on hypergraphs [9], we know that asymptotically an exponential query complexity in the key size leads to an exponentially small error rate for at least one net in the circuit and we can expect the reverse to hold as well.

Therefore, the RFQ attack's first step is to find low-activity/skewed wires. Finding the precise signal probability of wires in a circuit is an NP-hard problem [21]. Nonetheless, there has been a significant amount of work on precise and approximate practical algorithms for the problem. One can retrieve precise probability values from the circuit nodes' BDDs if they have manageable sizes [22]. Another approach is to propagate the signal probability values from input wires

(unconstrained wires assumed 0.5) to internal and output wires via propagation rules along a topological order. For example, a probability propagation rule would compute the signal probability of an output of an AND-gate as $p_a \times p_b$ where p_a and p_b are the probability values of the gate's inputs. This approach however can produce large inaccuracies since reconvergent paths in the circuit violate the independence of the input probabilities of gates, which is an assumption that the multiplication propagation rule demands.

Another approximate approach is to simulate the circuit on N random patterns and record the 1/0-ness rate of each wire. One can estimate confidence intervals from these statistics as well [21]. It is possible to use the propagation rules for non-reconvergent nets and resort to random simulation otherwise, yielding a hybrid scheme as in [23].

For the RFQ attack, we experimented with several approaches: Random simulation with N patterns, precise-BDD, naive propagation, and the hybrid method in [23]. The BDD-based approach while being the most precise fails on BDD-exploding nodes. In our experiments, we used the hybrid method from [23] which was shown to perform better than naive propagation. Plus, with the hybrid method, the number of tested patterns can be adjusted to trade-off runtime with accuracy.

The skewed-net-detection via signal probability estimation is performed every j DIP-mining steps. The probability of net $w(x, k)$ is performed using unconstrained x but several different k extracted from the current IO-constraints. This makes it such that the signal probability values are based on the most recent hypothesis on the key which is bound to improve over the course of the attack.

In the RFQ attack, we are interested in skewed "cones"/functions rather than single nets. Consider the AND-tree cone in the circuit in Fig. 2a. In this typical point-function structure the internal nodes in the tree have skewed probabilities and this skewness increases as we get closer to the tip of the tree. In order to extract the tip of the skewed cone, after finding skewed nets, a containment analysis is performed which prunes the skewed nets that are contained within the transitive-fanin of another (possibly more) skewed net. The end result is a set of skewed cones identified by their tip. This procedure can be seen in `FindSkewedNets` as the major part the RFQ algorithm `RFQSATAttack` in Algorithm 2. Note that one can combine or replace this procedure with a tree-finding approach such as the one in [24] to focus on point-function locking trees rather than any skewed net in the circuit.

C. Outside-of-Skewed-Cone Querying

Once the skewed cones are detected the first important novelty in the RFQ attack is its ability to conditionally deobfuscate the remainder of the circuit. Given a set of skewed cones g_i, what the RFQ attack can do, is launch a SAT attack that aims to learn everything there is to learn about the key absent precise knowledge of the skewed cones g_i.

The way this works is simple. The miter circuit in the SAT attack has two copies of c_e. $c_e(k_1, x)$ and $c_e(k_2, x)$. During the attack, the SAT solver makes the outputs of these two circuits differ by keeping x common among them, and coming up with two different keys $\hat{k}_1$ and $\hat{k}_2$ for which the difference between the two keys propagates to a difference at the output. What the "outside-of-skewed-cone" querying (OSCQ) subroutine of the RFQ attack does for a single skewed cone g_w, is to tie $g_w(k_1, x)$ in $c_e(k_1, x)$ to its corresponding net $g_w(k_2, x)$ in $c_e(k_2, x)$. This forces the SAT solver to come up with $\hat{x}$, $\hat{k}_1$ and $\hat{k}_2$ for which $g_w(\hat{k}_1, \hat{x}) = g_w(\hat{k_2}, \hat{x})$, but $c_e(\hat{k}_1, \hat{x}) \neq c_e(\hat{k_2}, \hat{x})$. i.e. the skewed cone is *not* the source of difference in the miter. Essentially by keeping the skewed cones invariant, we ask the SAT solver: querying what input pattern helps disqualify keys that are producing ambiguity *outside* of g_w?

One may be tempted to think that tying the key bits that are in the transitive fanin of g_w in the two c_e copies in the miter achieves the same goal. However, the two approaches result in slightly different Boolean conditions. The divergence appears in cases where difference in the keys does not lead to a difference at g_w, but does lead to a difference at the output between $c_e(k_1, x)$ and $c_e(k_2, x)$. This happens when key bits inside a skewed-cone have paths to the output that do not necessarily go through g_w.

Once the above DIP-mining and IO-constraint addition concludes with the constrained miter becoming unsatisfiable, what we have is a guarantee that the only way to improve the attacker's hypothesis on the key is to learn the functionality of g_w. This task is discussed later in this section. The algorithm for this OSCQ is provided in the `OutSkwConeQuery` routine in Algorithm 2.

D. A Deeper Look into EFS-OG Security

Before describing the various algorithms for querying the inside of the skewed cone, we take a much-needed deeper look into EFS-OG security. Since the advent of EFS schemes such as AntiSAT [12], SARLock [13], SFLL, etc. there have been numerous proposed so-called "removal" attacks [15], [9]. The idea in these attacks is that since point-function schemes insert these tree-like structures in the circuit if the attacker finds them, they can be removed from the locked circuit to obtain the original circuit. The signal-probability-skew (SPS) attack [15], the wire-disagreement analysis of AppSAT [9], and the RFQ attack itself, all have the ability to detect low-activity nets which in the case of point-function schemes is typically the output of the inserted point-function structure. After detection, the attacker can proceed with a removal attack.

However, there is a very important caveat here regarding the correctness of removal attacks that to the best of our knowledge has not been discussed before. This point reveals itself once we embrace the formal definition of EFS security. All three notions of security defined in [8], and reiterated here, start with the locking being performed on a family of original circuits C_o. While this may seem like a minor formality in the security definition, it plays a critical role in deciding the security of a given scheme. For instance, AFS-OG's impossibility results apply only to certain circuit families. This restriction on circuit families can be encoded-in/represented-by C_o.

As for EFS, the success and validity of removal attacks relies heavily on the family C_o. To illustrate this point,

consider the AntiSAT locked circuit in Fig. 1a. Here the inserted structure, i.e. the AntiSAT block, is XORed with a wire in the circuit. We know from the description of AntiSAT that under the correct key this block outputs 0 on all input patterns x and therefore never affects the circuit. Hence an attacker that finds the tip of this block in the circuit, can proceed with a removal and recover the original circuit c_o uniquely. The caveat here is that, if C_o is not restricted in any way, then *there exists numerous other* $c'_o \in C_o$ *which 1) are not equivalent to* c_o*, but 2) could have been locked with a non-AntiSAT locking algorithm to produce the same* c_e.

To elaborate, consider that the AntiSAT block $ant(k,x)$ is the AND of two complementary blocks $g(k_1,x)$ and $\overline{g(k_2,x)}$ where k_1 and k_2 are two equal-length subkeys. If $k_1 = k_2$ then $ant(k,x)$ is always 0. If $k_1 \neq k_2$, if the g functions are AND-trees (maximum-query-complexity configuration), then $ant(k,x)$ will serve as a point-function that flips the output on $x = k_1$. This is because $\overline{g(k_2,x)}$ will be 1 allowing the $g(k_1,x)$ point-function to propagate, except for when $x = k_2$ where $\overline{g(k_2,x)}$ will turn 0 and block an already 0 $g(k_1,x)$. Now here is the critical question for the removal attacker: Who is to say that the original circuit did not *naturally* include a point-function $P(x_*,x)$ which was then subsequently obfuscated to $g(k_1,x)$ through some unknown locking scheme, with the correct key being $k_1 = x_*$?

What is the attacker's advantage if he goes ahead and performs a removal attack despite the above warning? If $ant(k,x)$ is the only low-activity cone in the circuit, besides the possible c_o that results from assuming that $ant(k,x)$ is constant 0, C_o includes 2^l (l being the length of k_1) other possible original circuits c'_o which could have been locked to produce the same c_e. Hence, the success rate (advantage) of the removal attacker in the EFS game is exponentially low in the width of the point-function $O(1/2^l)$.

What if we begin restricting C_o? What this restriction means in practice is that the attacker has some sort of "a priori" knowledge on c_o. This can be represented with a probability distribution on C_o or by simply limiting C_o to a subset of equiprobable original circuits. The attacker for instance knows that $size(c_o) \leq size(c_e)$ and $depth(c_o) \leq depth(c_e)$. This immediately disqualifies an enormous portion of the space of all possible n-to-1-bit Boolean functions. Does this help in improving the advantage of the removal attacker? The answer is no, because, the 2^l possible c'_o that can lock to the same c_e in the case of AntiSAT, all lie within this size and depth limit. Point-functions are log-depth poly-size circuits and hence could easily fit in the size/depth-limited C_o.

If we continue this restriction, we will realize that the advantage of the removal attacker does not rise significantly until we start specifically omitting from C_o circuits that include point-functions. This in practice is a strong/restrictive/unrealistic assumption. Practical circuits as we will show in our experimental case study, are replete with (multi)-point-functions in the form of comparator logic.

What about knowledge of the Lock^{C_o} algorithm? If the attacker knows precisely that AntiSAT is the sole locking scheme used to obfuscate the design, he can match the double-AND-tree structure to that of AntiSAT if no other naturally occurring such structures exist in the circuit. However, in practice, this is a brittle assumption. The attacker may have a distribution on a family of Lock algorithms but it is not plausible to assume that an attacker reverse engineering a design can vouch that a specific version of AntiSAT was solely used to lock the design. The defender can simply mix a couple of different point-function schemes to confuse such an attacker. We will discuss simple defense tricks like this further in Section IV.

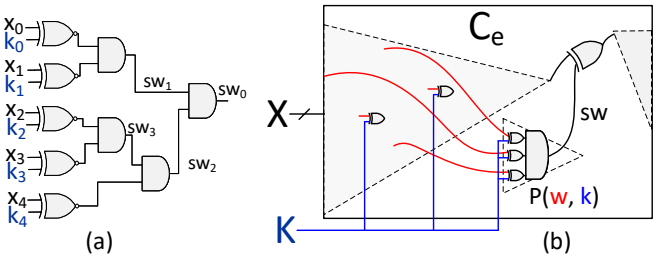


Fig. 2: (a) And-tree with progressively more skewed nets where we are interested in sw_0 as the tip of the skewed cone. (b) A skewed cone inserted into a larger circuit. The red wires are the nets that first come into contact with the key. An AllSAT on these wires will capture all the important queries required for precise learning of the skewed net sw.

We can summarize the above discussions in the following lemma.

Lemma 2. *(informal) The advantage of a removal attacker against SFLL, AntiSAT, and SARLock when 1) the attacker does not know the precise description of* Lock^{C_o}*, and 2)* C_o *is only size/depth-limited by* c_e*, is* $O(1/2^l)$ *in the* EFS-OG *game,* l *being the width of the point-functions used.*

An import result of Lemma 2 is that under the assumption that C_o is only slightly size/depth-limited and Lock^{C_o} is ambiguous to the attacker, then EFS security is quite easy to achieve. The mere XORing of a single keyed multi-point-function with a net in the circuit, without removing anything from the circuit, will satisfy the formal definition of EFS with exponential security. Given the abundance of point-functions in practical circuits, the first assumption seems quite realistic. As for the second assumption of unknown Lock^{C_o}, despite it being reasonable in practical settings, we will present a randomized Lock scheme in Section IV which allows for loosening this assumption.

E. Faster Inside-of-Skewed-Cone Querying

Better informed about the limitations of removal attacks, we now come back to the RFQ attack. After the first phase of the RFQ attack, low-activity nets are extracted and the circuit is correct for all patterns that do not disturb these rare nets. As we discussed in the previous subsection, it is rather easy to achieve EFS-OG under weakly restricted C_o and randomized Lock^{C_o}. i.e., given these low-activity nets if they result from keyed point-functions with (effective)-key-width l, then it is highly likely that at least 2^l input patterns will all have to be queried to learn this point-function precisely. This may disparage us from further investing in deobfuscating these skewed nets if $l > 100$ and exiting the attack after the outside-skewed-cone querying step. This is certainly an option for approximate

Algorithm 2: Given oracle access to c_o and the expression for c_e performs two-tier deobfuscation with possible speed-up for deobfuscating skewed nets.

```
1  Function RFQSATAttack(c_e, c_o as black-box):
2      M ← c_e(k_1,x) ≠ c_e(k_2,x), F ← true
3      while F ∧ M is solvable do
4          x̂, k̂_1, k̂_2 ← SAT(F ∧ M)
5          ŷ ← c_o(x̂)
6          F ← F ∧ (c_e(k_1,x̂) = ŷ) ∧ (c_e(k_2,x̂) = ŷ)
7          if d rounds passed then
8              Skw ← FindSkewedNets(F, c_e, c_o)
9              OutSkwConeQuery(F, c_e, c_o, Skw)
10             InSkwConeQuery(F, c_e, c_o, Skw)
11     satisfy F with k̂_1, k̂_2
12     return k̂_1 as correct key
13 Function FindSkewedNets(F, c_e, c_o):
14     ProbMap ← ComputeSigProb(c_e, F, NumKeyPatt,
         NumInputPatt)
15     for w ∈ Wires(c_e) do
16         if ProbMap[w] < ProbThreshold then
17             Skw ← Skw ∪ {w}
18     for w_i ∈ Skw do
19         for w_j ∈ Skw do
20             if w_j ∈TransFanin(w_i) then
21                 Skw ← Skw \ {w_j}
22     return Skw
23 Function OutSkwConeQuery(F, c_e, c_o, Skw):
24     for w ∈ Skw do
25         w_1 ≡ g_w(k_1,x) and w_2 ≡ g_w(k_2,x)
26         F ← F ∧ (w_1 = w_2)
27     SATAttack(F, c_e, c_o)
28 Function InSkwConeQuery(F, c_e, c_o, Skw):
29     Q ← F
30     for w ∈ Skw do
31         Ktl_w ← FirstKeyTouching(w)
32         Act_w = (w_1 = RareVal[w])
33     Act ← ⋁_{w∈Skw} Act_w
34     Ktl ← ⋃_{w∈Skw} Ktl_w
35     while M ∧ Q ∧ Act solvable do
36         for w ∈ Skw do
37             if M ∧ Q ∧ Act_w is UNSAT then
38                 if w never activated then
39                     ResolvedNodes[w] ← ~RareVal[w]
40                     Q ← Q ∧ (w ⊕ RarVal[w])
41                     F ← F ∧ (w ⊕ RarVal[w])
42                 Skw ← Skw \ w
43             else
44                 x̂_r, Vars, Model ← SAT(M ∧ F ∧ Act_w)
45                 ŷ_r ← c_o(x̂_r)
46                 bt ← BackTrack(F, c_e, c_o, w, x̂_r, ŷ_r)
47                 if bt = ProvenInactive then
48                     ban ← {u ⊕ Model[u] | u ∈ Ktl_w}
49                 else
50                     if bt = ProvenActive then
51                         F ← F ∧ (c_e(k_1,x̂_r) = ŷ_r)
52                         ∧(c_e(k_2,x̂_r) = ŷ_r)
53                     ban ← {u ⊕ Model[u] | u ∈ Ktl}
54                 Q.AddClause(ban)
```

```
1  Function BackTrack(F, c_e, c_o, w, x̂_r, ŷ_r):
2      B ← (c_e(k_1,x̂_r) = y_r) ∧ F
3      if B ∧ ~(w ⊕ RareVal[w]) is UNSAT then
4          return ProvenActive
5      else if B ∧ (w ⊕ RareVal[w]) is UNSAT then
6          return ProvenInactive
7      else
8          return ActivationUnclear
```

attackers.

However, there are two reasons why RFQ does not terminate here and performs a second phase focusing on these skewed cones. First, there are many naturally occurring low-activity nets practical circuits that require an exponential but manageable number of queries (e.g. $l < 2^{30}$ or 1B), which cause problems for the baseline SAT attack, AppSAT, and DDIP, due to the excessive clause build-up. This is while we know that if a particular net is query-complex with an onset of size h, scan-querying runs in $O(2^l)$ with memory $O(h)$ without any NP-complete SAT solver calls. This is a performance gain that can significantly boost deobfuscation speed for circuits with low-activity nets with manageable sizes in an automatic generic fashion. Consider a point-function with size 30. 2^{30} is around a thousand mega-queries. With a 1Mhz clock frequency for a combinational circuit, we can perform one mega-queries every second, and in a thousand seconds (16 minutes) we can learn a point-function of size 2^{30}. Compare this to the baseline SAT attack which will require storing and then solving a SAT problem with 2^{30} copies of the obfuscated circuit c_e corresponding to a terabyte of data if each circuit copy takes up only a kilobyte. KC2 [25] which is the main existing approach for generically dealing with such excessive clause buildup, will have to simplify incrementally a circuit that is growing to size $2^{30}.size(c_e)$ using complex circuit simplification techniques.

Second, via a smarter more complicated inside-skewed-cone strategy we can end up with an automatic generic *early termination* in certain cases, avoiding excessive querying altogether. Consider the (SFLL) family of schemes. These schemes are based on first corrupting the functionality of a net using point-functions, and then correcting the functionality using a restore-unit which is typically a look-up-table (LUT).

The corruption part of SFLL can leak to the attacker the location of the corruption. This in some cases can violate EFS security even under the two assumptions discussed earlier. Consider the case of an SFLL scheme where a point-function is used to corrupt a net w at x_* creating $w_{crpt} = w \oplus P(x_*, x)$. Then a single point-function $P(k, x)$ is used to correct the functionality of w_{crpt} getting $w_{crct} = w_{crpt} \oplus P(k, x)$. In this case, the circuit even after resynthesis is highly likely to contain two point-functions leading to low-activity nets. The RFQ attack can pick them up and analyze them. If we find the $P(x_*, x)$ net in the circuit, and then try to activate it, the DIP that we discover will be x_*. Querying this on the oracle will cause the corrective function to kick in and correct for the activation of $P(x_*, x)$. By observing the output we will be able to tell in certain cases that this correction has occurred on x_*. Since we know that there is only one point on which corrective $P(k, x)$ activates, then we must have $k = x_*$. The key will be

resolved instantly with no need for additional queries.

The FALL attack proposed in [26] uses various functional analysis techniques to find comparator logic and then extracts input patterns that might activate them to resolve cases like the above in SFLL. The RFQ attack uses a much more generic approach by trying to activate skewed nets. We discuss the various parts of this inside-of-skewed-cone querying (ISCQ) of the RFQ attack herein. The subroutine begins by receiving a set of detected skewed wires Skw.

Attempting to Activate/Learn the Rare Net. ISCQ-RFQ first tries to activate the skewed wires and produce an output ambiguity in the process. This is done by demanding that of the two miter copies of net w, $w_1 = g_w(k_1, x)$ and $w_2 = g_w(k_2, x)$, one take the rare value: $Act_w = (g_w(k_1, x) = RareVal[w])$. When Act_w is ANDed with the miter condition M, the result is that either both w_1 and w_2 take their rare values, and the miter is asserted via ambiguity in the path from w to the output; or that the miter condition is asserted via an ambiguity at w itself, i.e. w taking its rare value under some key, and its common value under another key hypothesis. The result is that w staying inactive is avoided.

The Act_w are activated individually in a round-robin fashion, and if asserting an Act_w term is UNSAT, this means that the functionality of the wire w may be fully resolved and it can be excluded from further activation attempts.

Backtracking to Verify Activation. If the fanin of the skewed cone includes no keys, asserting Act_w will produce input patterns that will definitively activate an internal skewed wire. If however, the cone includes yet-unresolved key-bits, then, forcing $g_w(k, x)$ to its rare output in the solver can yield a DIP $\hat{x}_r$ that in fact does not lead to a rare output in the oracle. This is because the rare query was extracted based on a yet incomplete key hypothesis, i.e. $g_w(k_*, \hat{x}_r) \neq g_w(k, \hat{x}_r)$. To deal with this, the RFQ attack performs a backtracking step asking: given an input pattern x_r that we suspected would rare-activate a skewed net, and observed output $y_r = c_o(x_r)$, can we *prove* that the skewed net w has been activated/not-activated?

This can be done by running a couple of SAT queries. First, given $y_r = c_e(k, x_r)$ under the current key condition, can w take its *common* value? If the result is UNSAT, then w must have been activated to its rare value, which means we have found a rare query and we can add it as a valuable IO-condition. If not, we ask the opposite, i.e. can w be assigned its *rare* value? If the call is UNSAT, then w has definitely not been activated. If the call is SAT, this means that under the current key condition it is not immediately possible to prove whether w was activated or not.

Expanded Input Exploration. ISCQ-RFQ performs the above query+backtrack in a loop to query every input pattern that might activate a rare wire. This exploration must avoid redundant queries as much as possible.

Consider a keyed tree-based point-function $T(k, x)$ inserted in the circuit. If this tree is attached to the primary inputs, when studying the skewed cone that results from this tree, $g(k, x) = T(k, x)$, i.e. the skewed cone precisely matches the tree. Hence in order to query-learn $g(k, x)$ we can simply go through all possible patterns of x which achieves the complexity $O(2^{|x|})$ precisely. However, assume the case where T is instead connected to a set of internal nodes in the circuit $w = \{w_0, ..., w_t\}$ as seen in Fig. 2b. Then the fanin cone of $T(k, w)$, includes all the fanins of each of the w_i nets as well. This results in an expansion of the domain of T from w to $x = \{x_j | x_j \text{ is PI} \in \text{transfanin}(w_i), w_i \in w\}$. The naive approach of querying the entire domain of x_w will now query a much larger space than the domain of the inserted tree T. For each extra bit x_j that is added, the query space size is doubled. Furthermore, it might become the case that even though $T(k, w)$ is a point-function with small onset, $g(k, x)$ will have an exponentially larger onset.

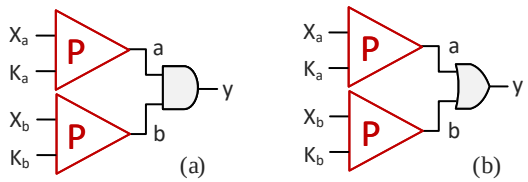


Fig. 3: Circuit examples for inside-of-skewed-cone querying.

In the case of such expansion, ISCQ-RFQ uses a specific SAT-based approach to query only necessary patterns in the domain of T. The approach is as follows. Take the skewed cone $g_w(k, x)$. We perform a breadth-first-search from the inputs x of g towards the skewed tip. In this search, the first wires to come in contact with key-logic are recorded as "first-key-touching" wires (shown in Fig. 2b with red). It is easy to prove that by assigning all possible values to these first-key-touching wires, we will exhaust all key-dependent ambiguity in $g_w(k, x)$ since the interaction of key-wires with non-key-wires is contained in these contact points. The exploration is done by banning the state of these wires by adding a ban-clause to the SAT solver.

Backtrack-Dependent Exploration. In the presence of multiple skewed cones, the above exploration in the RFQ attack is done with respect to the backtracking status. If the skewed net w is proven activated, this is a rare query that is immediately added to the solver as an IO-condition and exploration continues with the new information. If the skewed net w is proven inactive, the IO-condition is discarded, but the state of the key-touching-wires in the fanin of w is banned to avoid exploring this portion of the input again. If the activation status is uncertain, a weaker condition is banned. The state of the *union* of the key-touching-wires is banned. The following example shows how the above approach can avoid some unnecessary queries.

Example. Take the circuit in Fig. 3a. Assume that the X inputs and K keys are all of width 4. Also assume that the wires a and b have been selected as skewed nets for inside-of-cone querying. The Act condition hence captures that a or b take on their rare value and propagate their difference to the output. On the first call to the solver, a pattern like $\hat{X}_a = \hat{X}_b = 0000$ and $\hat{K}_{a1} = \hat{K}_{b1} = 0000 \neq \hat{K}_{a2} = \hat{K}_{b2} = 0010$ can be returned, which under the current key hypothesis should activate both of a and b and propagate their effect to the output. This is queried on the oracle. The oracle returns $y = c(K_a^*, K_b^*, \hat{X}_a, \hat{X}_b) = 0$. From this 0 at y, backtracking will not be able to say whether a or b were provably activated.

Hence, the union of the key-touching-logic state is banned. i.e. the condition $((X_a, X_b) \neq (0000, 0000))$ is added to the solver. This process will ban a single input on every iteration and conclude in 2^8 iterations if no activation is ever proved, which is the theoretical minimum number of queries required to resolve the circuit in Fig. 3a.

For the circuit in Fig. 3b, the same occurs, except that when a $y = 0$ is observed at the output, this means that a and b must have both been 0, which backtracking will prove. Hence on every iteration, the key-touching-logic state of the individual skewed cones are banned. i.e. After querying $(X_a, X_b) = (0000, 0000)$ the condition $(X_a \neq 0000) \wedge (X_b \neq 0000)$ is added to the solver. After 2^4 queries, a and b will be activated at some point and the result stored as a rare IO-condition. Hence, the overall procedure will query the oracle at most 2×2^4 times.

Limitations. Despite the backtracking, ISCQ-RFQ still uses a somewhat "blind" querying procedure to learn functions with an input space of size 2^n with 2^n queries. The SAT attack however, can terminate in less queries than this given the long list of input-output observations that ISCQ-RFQ simply discards. For instance take the case of diversified-tree-logic (DTL) from [9], [27]. In AntiSAT-DTL, t of the first-layer AND gates in the AND-trees in AntiSAT are replaced with OR gates. Against this, the baseline SAT attack can terminate at $2^n/3^t$ queries, using facts about the onset shape/size of the diversified trees. However, ISCQ-RFQ will blindly exhaustively query the 2^n input space. Also if dummy key logic is used to artificially expand the input space of the skewed net ISCQ-RFQ can explore more than the minim query count. In such cases, ISCQ-RFQ will only win against the SAT attack if the super-linear growth in the SAT attack space and runtime is more severe than the extra querying resulting from ISCQ-RFQ's blind approach.

The `InSkwConeQuery` routine in Algorithm 2 shows the procedure for the ISCQ part of RFQ.

F. Sequential RFQ Attack

The RFQ attack can be extended to sequential circuit deobfuscation. Sequential deobfuscation is based on replacing the SAT calls in the SAT attack with model-checking (MC) queries [25], [7]. Bounded-model-checking (BMC) produces the fastest sequential attacks for shorter depth state graphs. Essentially, a sequential miter is built similar to the combinational case, and then a BMC query up to round u is used to find a discriminating input sequence (DIS). The DIS is queried on the sequential oracle and then added as a condition to the model-checking model, or in the case of SAT-based BMC, directly to the SAT solver responsible for the BMC routine.

The RFQ routine can be adapted to the sequential case, especially that sequential circuits, due to counter-logic and control finite-state-machines (FSM), have a lot more comparator logic than arithmetic circuits. For a sequential RFQ attack, skewed signals can be detected similar to the combinational version, except the pattern simulation will have to be done with random sequences of randomly selected depth up to the current bound in the attack. Then since the BMC has an unrolled circuit, the skewed nets can be tied to each other in the unrolled miter circuit for outside-cone querying which will yield correct-under-skewed-assumption sequential keys. We were able to deobfuscate conditionally some query-heavy sequential benchmarks such as s400 with this approach in less than a few seconds.

Inside-skewed-cone querying however is more difficult for sequential circuits since the skewed cone g may include state-elements. Therefore, since we do not have direct control of s, bounded unrolling is necessary to enumerate all sequences of x. There likely exists much better sequential RFQ inside-cone-querying strategies that we leave for future work.

G. Experiments

We implemented the RFQ attack on combinational circuits in C++ using the Gluecose SAT solver. All tests were run on an AMD Threadripper 128 core machine with 256GB of memory running Linux. We used the combinational ISCAS benchmarks for testing as seen in Table I.

TABLE I: ISCAS combinational benchmarks used.

bench	#in	#out	#gate	bench	#in	#out	#gate
c432	36	7	160	c2670	157	64	1193
c499	41	32	202	c3540	50	22	1669
c880	60	26	383	c5315	178	123	2307
c1355	41	32	546	c6288	178	123	2307
c1908	33	25	880	c7552	206	107	3512

Skewed-net Detection. Table II shows the results of skewed net detection using probability propagation versus the hybrid approach from [23] with a 1000 input patterns under 5 different IO-conforming key patterns. The benchmark circuits were locked with various AntiSAT parameters such as width, number of inserted blocks, level of obfuscation of the tree itself, and added RLL keys. The probability of catching all AntiSAT tips after 20 iterations with RFQ analysis performed every 5 DIPs is listed. As can be seen from the data, the hybrid approach does somewhat better in finding AntiSAT tips. Fig. 4 shows the signal probability of the different classes of nets in the c432 benchmark for AntiSAT, SFLL-point (corrupt and correct on a single point), and AntiSAT-DTL for different parameters. It can be seen that the tree tips maintain a high probability skew throughout the attack.

Outside-of-Skewed-Cone Querying. We performed some compound locking schemes, by combining AntiSAT with RLL. The goal was to see the correctness of the keys outside of the skewed nets. The runtime and iterations count of OSCQ for the benchmark circuits locked with AntiSAT+RLL is shown in Table III comparing OSCQ of the RFQ attack, to the baseline SAT attack running on the RLL locked circuit before the insertion of the AntiSAT block. It can be seen that RFQ's OSCQ recovers the non-skewed part of the circuit precisely in time/query-count on par with the baseline SAT attack running against the non-skewed part of the circuit.

Inside-of-Skewed-Cone Querying. To demonstrate the speed-up of the ISCQ-RFQ, we ran the ban-clause-based exploration of the skewed-net's input with backtracking on the benchmark circuits versus the baseline SAT attack and KC2 with ABC simplification every 50 DIPs. The proportion of input space traversed is shown in Fig. 5, and runtime is

TABLE II: Success rate of finding all skewed tips in different locked benchmark circuits using random pattern simulation and probability propagation. (n, m, t, e) represents t AntiSAT trees of width n inserted in a circuit with e RLL keys, where each tree is itself obfuscated with m XOR gates.

params	(12,0,1,0)		(12,0,2,16)		(12,4,2,32)		(12,4,3,64)		(16,0,1,0)		(16,0,2,16)		(16,4,2,32)		(16,6,2,64)		(32,0,1,0)		(32,8,1,32)	
bench	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop	hybr	nprop
c432	100%	100%	100%	100%	97.5%	70%	90%	27.5%	100%	100%	100%	100%	100%	80%	100%	62.5%	100%	100%	100%	100%
c499	100%	100%	100%	100%	97.5%	55%	97.5%	37.5%	100%	100%	100%	100%	100%	85%	100%	77.5%	100%	100%	100%	97.5%
c880	100%	100%	100%	100%	97.5%	70%	97.5%	67.5%	100%	100%	100%	100%	100%	85%	100%	70%	100%	100%	100%	95%
c1355	100%	100%	100%	100%	97.5%	50%	95%	25%	100%	100%	100%	100%	100%	62.5%	97.5%	55%	100%	100%	100%	97.5%
c1908	92.5%	92.5%	100%	100%	97.5%	55%	95%	57.5%	100%	100%	100%	100%	100%	80%	100%	57.5%	100%	100%	100%	97.5%
c2670	100%	100%	100%	100%	97.5%	80%	92.5%	52.5%	100%	100%	100%	100%	100%	97.5%	95%	82.5%	100%	100%	100%	97.5%
c3540	100%	100%	100%	100%	100%	65%	97.5%	55%	97.5%	97.5%	100%	100%	100%	87.5%	100%	77.5%	97.5%	97.5%	100%	100%
c5315	100%	100%	100%	100%	95%	75%	97.5%	55%	100%	100%	100%	100%	100%	82.5%	100%	77.5%	100%	100%	100%	95%
c6288	100%	100%	100%	100%	100%	80%	97.5%	75%	100%	100%	100%	100%	100%	95%	100%	80%	100%	100%	100%	100%
c7552	100%	100%	100%	100%	95%	67.5%	92.5%	52.5%	100%	100%	100%	100%	100%	87.5%	100%	67.5%	100%	100%	100%	97.5%
average	99.2%	99.2%	100%	100%	97.5%	66.8%	95.2%	50.5%	99.8%	99.8%	100%	100%	100%	84.2%	99.2%	70.8%	99.8%	99.8%	100%	97.8%

TABLE III: Circuit locked with (n, m, t, e) AntiSAT+RLL. The baseline SAT attack is run on the RLL locked circuit alone. OSCQ-RFQ is run on the compound RLL+AntiSAT. m is irrelevant to these results.

params	(10,0,1,16)				(15,0,1,32)				(25,0,1,64)				(30,0,1,128)			
method	base		rfq		base		rfq		base		rfq		base		rfq	
bench	time	iter	time	iter	time	iter	time	iter	time	iter	time	iter	time	iter	time	iter
c432	0.02	10	0.03	11	0.02	11	0.04	15	0.06	27	0.11	30	2.3	142	28	431
c499	0.04	13	0.05	7	0.06	7	0.09	11	0.17	14	0.24	17	1.2	47	1.5	53
c880	0.07	7	0.09	7	0.08	15	0.1	8	0.13	17	0.13	14	0.16	22	0.32	25
c1355	0.08	10	0.12	18	0.14	13	0.36	31	0.32	30	0.63	40	6.1	80	4.8	73
c1908	0.12	9	0.15	10	0.19	15	0.25	17	0.28	19	0.51	27	8.9	50	6.3	53
c2670	0.16	9	0.19	9	0.21	13	0.32	26	24	384	20	320	67	539	390	1045
c3540	0.93	8	1.5	7	1.2	14	1.2	17	1.4	22	1.1	18	2.1	37	2.8	33
c5315	0.92	7	1.1	8	1	17	1	13	1.2	28	1.6	30	2.8	47	3.1	41
c7552	1.4	11	1.5	9	1.5	17	1.5	14	1.9	28	2.1	22	5.8	68	3.3	34

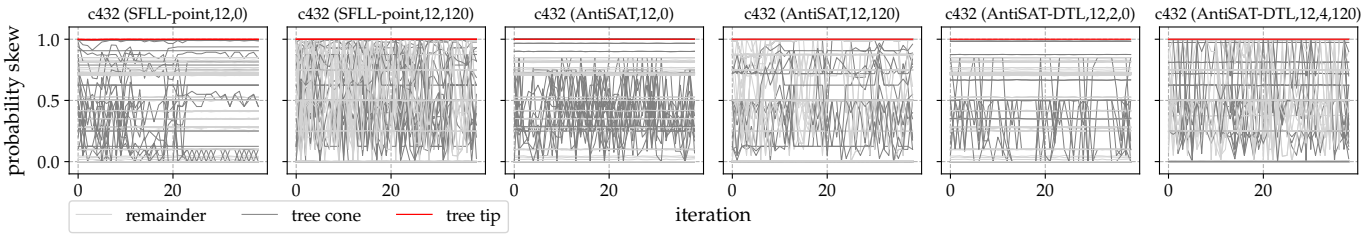


Fig. 4: Signal probability of different wires in the c432 benchmark locked with different schemes. The parameters are in the following format: (AntiSAT, width, num-RLL-keys), (SFLL-point, width, num-RLL-keys), and (AntiSAT-DTL, width, num-XOR-gates-first-layer, num-RLL-keys). Probability skew here is $2|0.5 - p_w|$. The tree cone wires leading to tree tips and the remainder of the nets are color-coded per the legend. The SFLL-point instances have two skewed tree tips.

shown in Fig. 6. Fig. 6 demonstrates that the runtime of the ban-clause-based routine is slightly super-linear. Using better AllSAT approaches [28] instead of ban-clause addition can improve this trend.

In addition, we ran ISCQ-RFQ on 5 different locked instances of the circuits from Table I with SFLL-point (single point corrupt+correct) with widths of 12, 20, and 30 resynthesized with ABC before being attacked. The attack was able to find the corruption comparator and early terminate in all cases in under 10 minutes each. The round-robin assertion of Act_w in line 36 of Algorithm 2 is crucial to this outcome.

IV. Boosting EFS

A. EFS Scheme Range/Capacity Expansion

We now begin to focus on improving upon EFS defenses. We first start with how the RFQ attack (and FALL attack for that matter [26]) succeed in breaking specific EFS schemes without exponential querying. Take the simplest SFLL scheme in which a point-function $P(x, x_*)$ is used to corrupt the original circuit, and a single point-function $P(x, k)$ is used to restore/correct the output. In this case, if the attacker queries x_*, he can observe with backtracking from the output that the restore comparator must have activated. Because the restore comparator can only correct a single pattern, x_* must have been the flipped pattern and the attack can conclude with a key-correctness guarantee. This can easily be extended to the N-pattern case. In general, if an N-point SFLL scheme is used, and N rare queries are encountered during the attack and successfully backtracked to definite restore-unit activations, the attack can exit without having to query the remainder of the input space. This is precisely what allows RFQ and FALL to break certain SFLL schemes. We call the number of patterns that the restore-unit can correct for, the *capacity* of the restore-unit which is N in this case.

Another important factor of an SFLL point-function scheme's security is the *width* of the point-function used. The width p is the (maximum) length of the patterns that are corrupted/corrected. For a point-function/restore-unit of width p there are 2^p input patterns in its domain that need to be

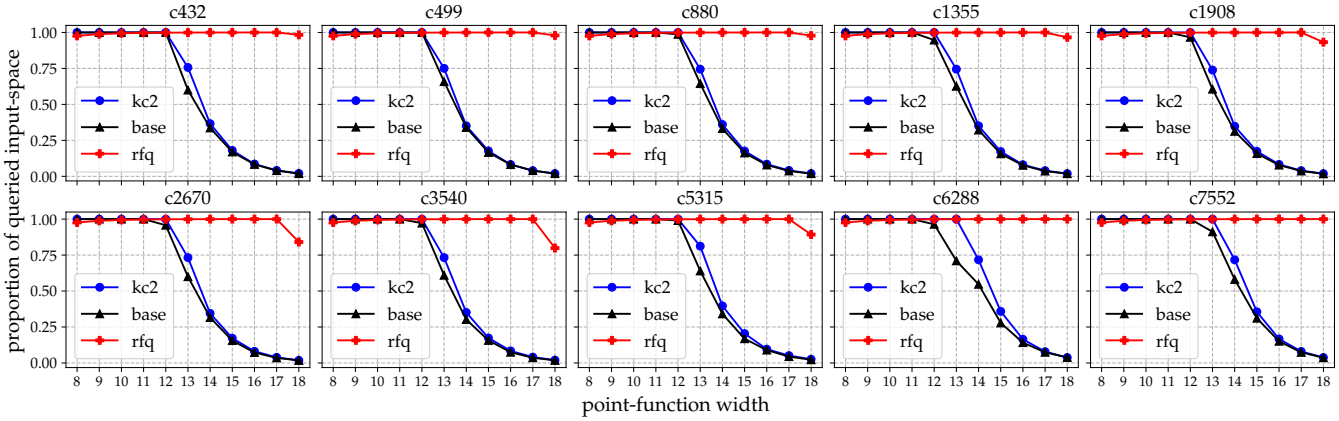


Fig. 5: Proportion of input range of 2^n (n on x-axis) queried within 2 hours with the RFQ, KC2, and baseline SAT attacks.

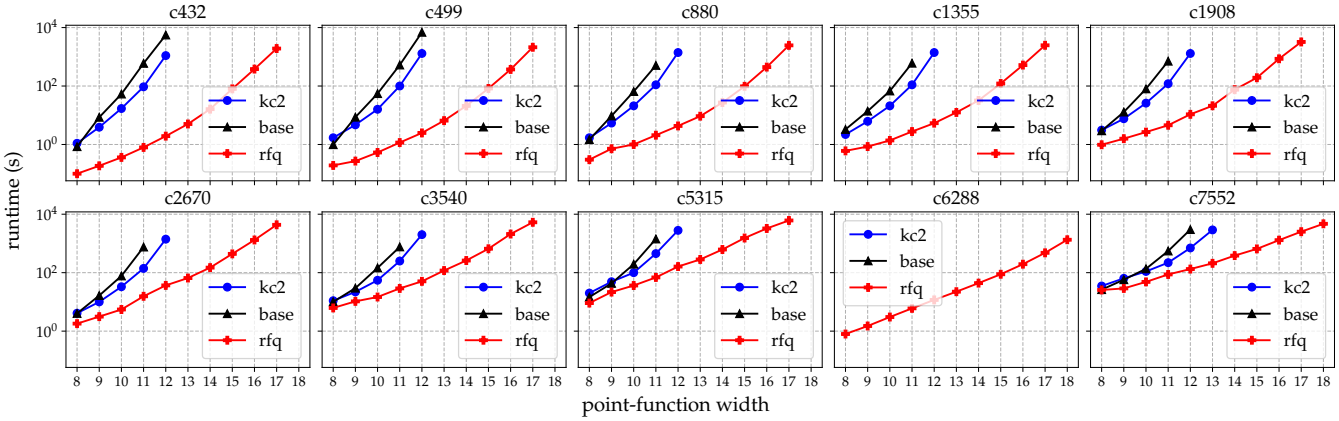


Fig. 6: Time in seconds for full recovery of circuit locked with AntiSAT of width n (x-axis) which requires 2^n queries for cases that were completed in less than 2 hours using RFQ, baseline SAT attack, and KC2. c6288 being a multiplier is not solved in 2 hours with KC2 and the baseline SAT attack.

queried. Consequently, the maximum query complexity of any SFLL scheme that uses a corrupt/restore-unit of width p is 2^p.

Before we go deeper into improving EFS schemes we describe a special restore-function that will prove useful later on. Recall that in AntiSAT/SARLock AND-trees fed with XORs of input and key bits are used, and in SFLL [29], [14] fixed size look-up-tables or hamming-distance (HD) units [19] are used for corruption and restoration. In [8] a new primitive was presented which we call a "row-activated" look-up-table (RA-LUT). With this LUT each entry has an additional key-bit that can be programmed to deactivate that entry. Hence, a row-activated LUT with p rows can be programmed to never activate (i.e. output a 0), or to activate on up-to-p distinct patterns.

Capacity Boosting. As we discussed earlier, given the definition of EFS under the assumption of weakly restricted C_o, removal attacks have exponentially low success rate. We can use this fact through Lemma 2 to prevent early attack termination and ensure always-exponential-query-count EFS.

Consider an RA-LUT of capacity 1 inserted in the circuit and XORed with a random wire. Unlike the fixed point-function, the RA-LUT can in fact be programmed to deactivate, i.e. output a constant-0 leaving no effect on the original circuit operation. Now from the attacker's perspective, the presence of the 1-capacity RA-LUT means that there might be a single pattern on which it activates/corrects. As per our discussions in Lemma 2 the original circuit may have naturally had a point-function that was absorbed intelligently by the defender into the RA-LUT, and the attacker has no reason to rule out this possibility.

This notion can be used to boost the query complexity of any SFLL scheme. If an SFLL scheme has corrupted N patterns and is correcting them with a restore-unit of capacity N, then by introducing p additional detachable-rows we can create the illusion for the attacker that more than N patterns are flipped. This will send him onto the task of finding these extra p patterns by scanning the entire input space, just to find out that they were fake to begin with.

Width Boosting. In addition to masquerading the capacity of the restore-unit to the attacker, it is possible to artificially boost the width of the restore-unit as well. A column-activated LUT (CA-LUT), has additional key-bits that will decide which bits in the incoming vector in a specific row are activated. When a certain column in the CA-LUT is deactivated the incident bits on that column will not affect the outcome, i.e. will be don't-care bits. By connecting dummy wires from randomly

selected (but nearby) nets, and connecting them to the column-activated LUT we can boost artificially the apparent width of the restore unit. The attacker does not know which input bits are active and is hence forced to query a doubly larger input space per each added dummy input.

Site Selection. We know now that inserting a row-column-activated-LUT (RCA-LUT) without any previous corruption will satisfy the theoretical EFS-OG notion with exponentially small advantage under weakly restricted C_o. However, if our Lock algorithm insists on inserting only dummy superfluous RCA-LUTs, 1) if the attacker knows that this is the Lock strategy, a removal attack will have perfect success rate 2) we can miss out on area savings that can come from absorbing naturally occurring existing point-logic in the original circuit.

A simple look at RT-level designs will immediately reveal that HDL is replete with comparator logic. Besides arithmetic units such as adders and multipliers, the remainder of RTL designs are dominated by decode/control logic, count-up/down-to-value logic and other syntax dominated by equality conditions. In fact, the microcode ROM used as a way to dispatch control signals in processors if implemented using switch-case/memory-like RTL that can be mapped to an RCA-LUT.

We advocate hence for performing RCA-LUT EFS locking with a footing in the RTL. Many RTL equality conditions will be mapped and/or removed during optimization or translated to combinational logic, from which extracting the word-level information becomes difficult. In addition to equality logic, constant values are present in the RTL which can also be mapped to keyed-logic. Constants hold critical information when it comes to control logic.

Hence, our proof-of-concept EFS locking flow is integrated with a Verilog parser (yosys), that finds RTL equality conditions which always map to comparator/AND-trees and replaces some of them with RCA-LUTs. With this approach there will exist RCA-LUTs that are in fact absorbing existing logic, making sure that the attacker is forced to query these blocks rather than assuming that they are all fake. *Note that randomizing this absorption and boosting in the above scheme allows one to weaken the second assumption of Lemma 2 (unknown* Lock *for the attacker), since even an attacker that knows* Lock*'s description does not know which LUTs are dummy, and which ones absorbed existing logic.*

B. Circuit Implementation

How can the RCA-LUT which is the heart of provably strong EFS-OG locking be implemented in silicon? The most straightforward way is to simply build a gate-level circuit that implements this. A gate-level implementation of this is rather straightforward. Per Fig. 7a, for a p-entry LUT of width n we need first p different AND-trees of width n. The incoming input bits are XORed with key-bits once for each row and these key-bits are the patterns stored in the LUT's rows. The output of these XORs can be killed with a key-controlled AND gate, the key to which will decide whether a particular bit is active on a particular row. An entire row can be deactivated by setting each one of these activation key bits in that row to 0.

The RCA-LUT may be implemented with much less overhead at the transistor-level. (Ternary)-Content-Addressable-Memory or (T)CAM [30] arrays are special transistor-level circuits that can be used to perform value-lookup. The structure of a CAM is similar to that of an SRAM memory array with vertical and horizontal lines and a regular layout. Each cell in the CAM array is a special cell that will pull down a line that is shared across all cells in a row called the match-line (ML). Existing CAMs are sequential elements, in that they need a clock to precharge the ML line and read out the result using sense-amplifiers. These sequential CAMs can be used to absorb sequential comparators, i.e. point-functions for which the result need not be ready until the next clock cycle. An RTL synthesis engine can easily recognize such patterns.

Per Fig. 7b, a combinational CAM may be designed by adding a pull-up resistor to the ML line so that when no pull-down is asserted (a match case) the ML line can rise. An inverter can be used to amplify this. The rise-time of the signal will be decided by the RC circuit formed by this pull-up resistor and the pull-down network. ML segmentation is a technique used to speed up CAMs by breaking down the ML into smaller pieces. CAMs have been the topic of circuit design research for decades. Better combinational CAMs will lead to better EFS locking per our discussions.

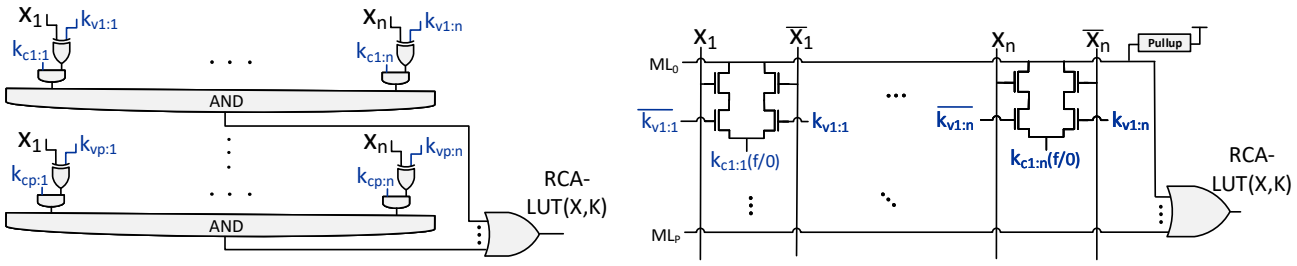
C. Experiments

RCA-LUT Overhead. We first implement standalone RCA-LUTs of different widths and capacities and synthesis them using the NanGate15nm standard-cell library with Design Compiler (DC). The power/delay/area of these LUTs is compared to that of a 16-bit comparator in Fig. 8. A linear growth in overhead can be observed as the width and capacity of the RCA-LUT increases.

RTL-level Absorption-based Locking. We used the pyverilog parser to detect equality conditions and constants in the RTL. We use the code-generator in pyverilog to then replace these conditions/constants with black-box modules and produce a locked-RTL. An example of this replacement can be seen in Fig. 9. The transformed design is then synthesized using DC. DC will treat these modules as undefined blocks and synthesis/map the rest of the design to gates. The flattened design is then written out.

We then implemented a C++ tool that reads the synthesized gate-level design, and maps the condition/constant modules based on user configuration parameters to RCA-LUTs or key-bits. The widest conditions are mapped to RCA-LUTs first as to maximize absorption. The user can choose to insert additional dummy RCA-LUTs into the design as well. While we perform this site-selection automatically herein for profiling purposes, it is advised that designers manually choose which conditions/constants to hide given that they have more intimate knowledge of their design.

The above flow was applied to an RS232 design from OpenCores. The flattened design seen in Fig. 11 contains 14 different comparator units 9 of which are bare "==" syntax, and the rest fall into switch-case statements. One of the bare comparators has a width of 16 which is the widest comparison in the design. We can artificially boost these values with RCA-LUTs. The comparators and constants from the yosys-



(a) (b)

Fig. 7: RCA-LUT implementations. The $k_{ci:j}$ keys "activate" the comparison at row i column j. The k_{vi} are the "compared-to" key values for row i. An entire row can be deactivated by setting each activation key in it to 0. (a) Combinational gate-level implementation of RCA-LUT. (b) Possible transistor-level CAM-based implementation of RCA-LUT. The key values can be implemented with one-time-programmable devices.

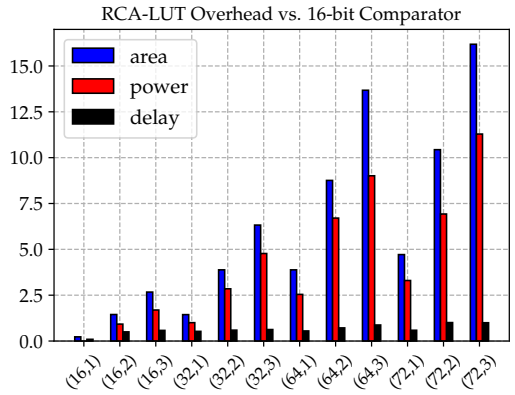
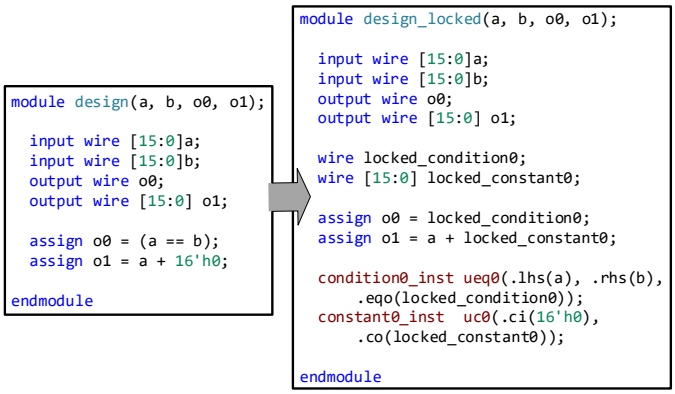


Fig. 8: Area/power/delay overhead ratios for standalone synthesized RCA-LUTs compared to a 16-bit comparator. (w, c) represents RCA-LUTs of width w and capacity c.



```
module design(a, b, o0, o1);

  input wire [15:0]a;
  input wire [15:0]b;
  output wire o0;
  output wire [15:0] o1;

  assign o0 = (a == b);
  assign o1 = a + 16'h0;

endmodule
```

```
module design_locked(a, b, o0, o1);

  input wire [15:0]a;
  input wire [15:0]b;
  output wire o0;
  output wire [15:0] o1;

  wire locked_condition0;
  wire [15:0] locked_constant0;

  assign o0 = locked_condition0;
  assign o1 = a + locked_constant0;

  condition0_inst ueq0(.lhs(a), .rhs(b),
      .eqo(locked_condition0));
  constant0_inst  uc0(.ci(16'h0),
      .co(locked_constant0));

endmodule
```

Fig. 9: RTL rewrite detecting constants and equality conditions and mapping them to modules. Later on the modules are mapped to RCA-LUTs or key-bits based on user parameters.

generated flow-graph of the design can be seen highlighted in Fig. 11.

Fig. 10 shows the area/power/delay overhead with different locking parameters. It can be seen that absorbing 2 comparators in the design, one of which absorbs the native 16-bit comparator, with widths boosted to 64 and capacities of 2 leads to an area overhead below 60%, power overhead of below 35%. The trees land off of the critical path and hence do not induce a delay overhead. This produces EFS security with a 2^{64} minimum query count.

Note that the locking security here is EFS meaning that approximation resiliency is not guaranteed at all. However,

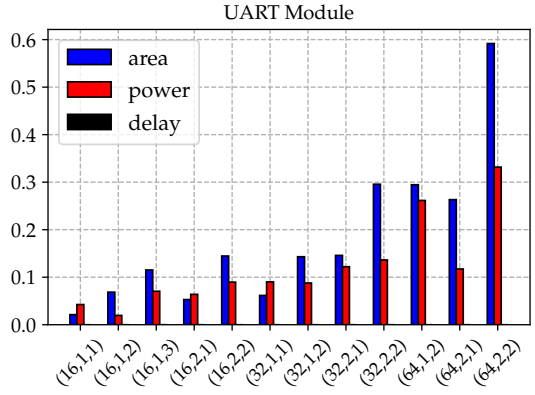


Fig. 10: Area/power/delay overhead ratios for RS232 module locked with boosted EFS. (w, n, c) represents n RCA-LUTs of width w and capacity c inserted in the design. In this design, the first RCA-LUT absorbed an existing 16-bit comparator, and the remaining $n-1$ are dummy inserted RCA-LUTs.

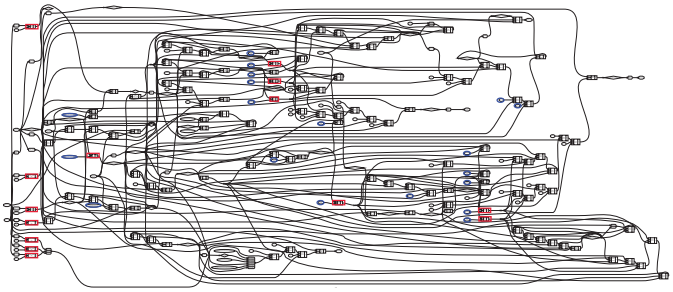


Fig. 11: RS232 RT-level flattened design. The red boxes are comparators that can be mapped to boosted RCA-LUTs, and the blue circles are constants. Constants can also be stored in a tamper-proof memory as secret keys for locking.

the equality conditions typically govern critical control signals that guide arithmetic units to process data. Often times these control signals are more unique to the design and valuable to the designer than arithmetic circuits which are instantiations of mostly publicly available and well-known adder/multiplier structures. Note that some arithmetic circuits already have look-up structures that can be mapped to keyed-RCA-LUTs providing secure locking. For instance, the S-Boxes in cryptographic modules or other translation tables can be mapped to (one-time) programmable logic as well.

V. Conclusion

In this paper we presented a novel SAT attack that can deal with query-complex deobfuscation tasks with near-constant

memory avoiding exponential blow-up of SAT solver times. We believe the two-pronged framework of the attack to be the best current approach to generic deobfuscation in the presence of rare nets. We further explored the notion of EFS security showing how with a few simple tricks, EFS security may be the first truly provably secure locking with bearable overhead and sound mathematical foundations. EFS can be a great option for hiding a critical part of modern designs, i.e. that of control/decode/FSM/counter-logic at least in the absence of low-overhead AFS solutions.

REFERENCES

[1] L. Azriel, R. Ginosar, and A. Mendelson, "Sok: An overview of algorithmic methods in ic reverse engineering," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*. ACM, 2019, pp. 65–74.

[2] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. IEEE/ACM Design Automation Conf.*, 2014.

[3] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation," in *USENIX Security Symposium*. Washington, D.C.: USENIX, 2013, pp. 495–510.

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

[5] S. Patnaik, M. Ashraf, J. Knechtel, and O. Sinanoglu, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," *CoRR*, vol. abs/1711.05284, 2017. [Online]. Available: http://arxiv.org/abs/1711.05284

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

[7] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential integrated circuits without scan access," *arXiv preprint arXiv:1710.10474*, 2017.

[8] K. Shamsi, D. Z. Pan, and Y. Jin, "On the impossibility of approximation-resilient circuit locking," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 161–170.

[9] 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.

[10] 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.

[11] 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*. IEEE, 2015, pp. 137–143.

[12] 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.

[13] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*, 2016, pp. 236–241.

[14] 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*, 2017, pp. 1–1.

[15] 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.

[16] 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*, 2017, pp. 46–51.

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

[18] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "Sigattack: New high-level sat-based attack on logic encryptions," in *Proc. Design, Automation and Test in Europe*. IEEE, 2019, pp. 940–943.

[19] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Ttlock: Tenacious and traceless logic locking," in *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 166–166.

[20] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Camoperturb: secure ic camouflaging for minterm protection," in *Proc. Int. Conf. on Computer Aided Design*. IEEE, 2016, pp. 1–8.

[21] F. N. Najm, "Statistical estimation of the signal probability in vlsi circuits," *Coordinated Science Laboratory Report no. UILU-ENG-93-2211, DAC-37*, 1993.

[22] A. Dutta and N. A. Touba, "Iterative opdd based signal probability calculation," in *24th IEEE VLSI Test Symposium*. IEEE, 2006, pp. 6–pp.

[23] S. Dupuis, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Identification of hardware trojans triggering signals," in *First Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, 2013.

[24] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (sfll-hd)–unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, 2019.

[25] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Kc2: Key-condition crunching for fast sequential circuit deobfuscation," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 534–539.

[26] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 936–939.

[27] H. Zhou, "A humble theory and application for logic encryption," Cryptology ePrint Archive, Report 2017/696.(2017). https://eprint. iacr. org/2017/696, Tech. Rep., 2017.

[28] T. Toda and T. Soh, "Implementing efficient all solutions sat solvers," *Journal of Experimental Algorithmics (JEA)*, vol. 21, pp. 1–12, 2016.

[29] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "Atpg-based cost-effective, secure logic locking," in *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[30] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712–727, 2006.

Kaveh Shamsi is an Assistant Professor in the Department of Electrical and Computer Engineering (ECE) at the University of Texas at Dallas (UTD). He received his PhD degree in Electrical and Computer Engineering from the University of Florida (UF) in 2020, his M.S. degree in Computer Engineering from the University of Central Florida in 2017, and his B.S. degree in Electrical Engineering from the Sharif University of Technology in Iran in 2014. His research focuses on formal methods and algorithms in hardware-oriented security, including circuit (de)obfuscation, reverse engineering, and side-channel analysis. He is the recipient of best paper awards at HOST'17 and GLSVLSI'18, and a best paper candidate at DATE'19.

Yier Jin i(M'12-SM'19) is an Associate Professor and IoT Term Professor in the Department of Electrical and Computer Engineering (ECE) in the University of Florida (UF). He received his PhD degree in Electrical Engineering in 2012 from Yale University after he got the B.S. and M.S. degrees in Electrical Engineering from Zhejiang University, China, in 2005 and 2007, respectively. His research focuses on the areas of hardware security, embedded systems design and security, trusted hardware intellectual property (IP) cores and hardware-software co-design for modern computing systems. He is also interested in the security analysis on Internet of Things (IoT) and wearable devices with particular emphasis on information integrity and privacy protection in the IoT era. Dr. Jin is a recipient of the DoE Early CAREER Award in 2016 and ONR Young Investigator Award in 2019. He received Best Paper Award at DAC'15, ASP-DAC'16, HOST'17, ACM TODAES'18, GLSVLSI'18, DATE'19, and AsianHOST'20. He is also the IEEE Council on Electronic Design Automation (CEDA) Distinguished Lecturer.