Breaking CAS-Lock and Its Variants by Exploiting Structural Traces

Abhrajit Sengupta[1*], Nimisha Limaye[1*], and Ozgur Sinanoglu[2]

[1] New York University, NY, US, {as9397,nsl278}@nyu.edu
[2] New York University Abu Dhabi, AD, UAE, ozgursin@nyu.edu
[*]These authors contributed equally to this work.

Abstract. Logic locking is a prominent solution to protect against design intellectual property theft. However, there has been a decade-long cat-and-mouse game between defenses and attacks. A turning point in logic locking was the development of miter-based Boolean satisfiability (SAT) attack that steered the research in the direction of developing SAT-resilient schemes. These schemes, however achieved SAT resilience at the cost of low output corruption. Recently, cascaded locking (CAS-Lock) [SXTF20a] was proposed that provides non-trivial output corruption all-the-while maintaining resilience to the SAT attack. Regardless of the theoretical properties, we revisit some of the assumptions made about its implementation, especially about security-unaware synthesis tools, and subsequently expose a set of structural vulnerabilities that can be exploited to break these schemes. We propose our attacks on baseline CAS-Lock as well as mirrored CAS (M-CAS), an improved version of CAS-Lock. We furnish extensive simulation results of our attacks on ISCAS'85 and ITC'99 benchmarks, where we show that CAS-Lock/M-CAS can be broken with ∼94% success rate. Further, we open-source all implementation scripts, locked circuits, and attack scripts for the community. Finally, we discuss the pitfalls of point function-based locking techniques including Anti-SAT [XS18] and Stripped Functionality Logic Locking (SFLL-HD) [YSN+17], which suffer from similar implementation issues.

Keywords: Anti-SAT · CAS-Lock/M-CAS · IP piracy · logic locking · removal attack · SAT attack · structural analysis

1 Introduction

With deep sub-micron technology, the capital cost considerations dictate outsourcing of integrated circuit (IC) manufacturing. Today, most of the semiconductor industry operates in a fabless manner, where foundries are organizationally and geographically separate from the design houses. However, such outsourcing to potentially *untrusted* entities has introduced threats such as intellectual property (IP) piracy [re112, re212, Tec17], IC counterfeiting/overbuilding [pen13], and hardware Trojans [big18]; such threats have become a pressing concern for commercial and government organizations alike. Besides, counterfeiting also poses significant security risks; according to a 2013 report by the Semiconductor Industry Association, 15% of all the "spare and replacement semiconductors" bought by the Pentagon are counterfeit [pen13].

1.1 Logic locking: Protecting the IC supply chain

Thus, protecting against such hardware threats has become paramount, and several design-for-trust solutions such as logic locking, split manufacturing, IC camouflaging, and physically unclonable functions (PUF) have been proposed over the years [RKK14]. Among them, logic locking is perceived as a one-stop solution that can protect against different entities in the IC supply chain, viz., foundry, test facility, and end user. It embeds a lock, in the form of logic gates, into the design which can only be unlocked by loading a secret key onto the chip's memory. An example for logic locking is shown in Fig. 1. Note that the original circuit (see Fig. 1a) is locked by inserting additional XOR/XNOR gates,

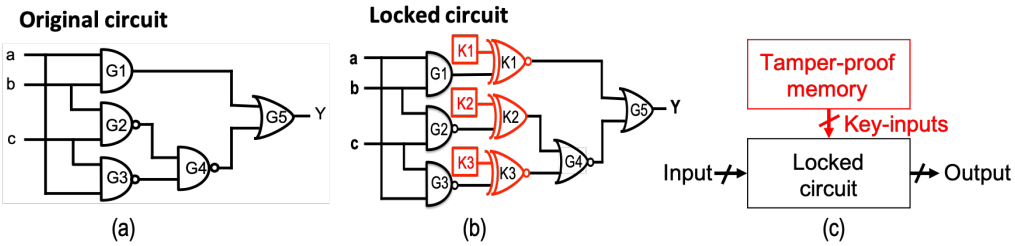


Figure 1: Illustration of logic locking. (a) Original circuit, (b) Circuit locked with secret key {K1, K2, K3} = 110, and (c) Tamper-proof memory driving the key-inputs. Source: [YSS+17].

Figure 2: IC supply chain in the presence of logic locking that protects against the untrusted foundry, test facility, and end user. Note that after chip fabrication and testing, the chip is unlocked by loading the correct key onto the memory of the chip.

called *key-gates*, into the design. One input of the key-gate is driven by a *key-input* that is driven from a *tamper-proof* on-chip memory, while the other input is the functional net. The design will work correctly only upon supplying the correct key, i.e., 110; otherwise, it produces incorrect outputs. A logic locking enabled IC design flow is shown in Fig. 2. The design is locked at the end of the design phase, and since only the IP owner possesses the secret key, the design is protected from any unauthorized access. After fabrication and testing, the chip is unlocked by loading the correct key onto the chip's memory.

1.2 Logic locking: Recent attacks

The concept was introduced in EPIC [RKM10], followed by a set of works such as FLL [RZZ+13] and SLL [YRSK15]. However, all these techniques were broken by a miter-based Boolean satisfiability (SAT) attack [SRM15], thereby exposing a serious vulnerability. It iteratively identifies *distinguishing input patterns* (DIPs) that eliminate incorrect keys from the search space. The computation of a DIP involves construction of a *miter circuit* with two copies of the locked circuit, where the two circuits share the primary inputs but have different key-inputs. When a SAT solver finds an assignment that satisfies the "miter" formula, this assignment is called a DIP. This DIP is applied to an oracle, i.e., a functional IC with the secret key loaded onto its memory, to prune out the incorrect keys in an iterative way. The SAT attack can break logic locking with a relatively small number of DIPs, and thus, the complexity of the attack, is low in terms of the number of DIPs required.

Naturally, several SAT-resilient techniques were developed by leveraging point functions that necessitated an exponential number of DIPs in the key-size such as SARLock [YMRS16], Anti-SAT[XS18], SFLL [YSN+17]. Nevertheless, the use of point functions introduced structural-analysis-based attacks [XSTF17, YMSR17, YTS19, SS20] as summarized in Table 1. Further, new schemes such as SFLL-fault [SNL+20] suffer from low output corruption, thereby facilitating approximate circuit recovery for an incorrectly unlocked design [SLM+17]. Other solutions include SAT-hard structures [KAHS19, SLPJ18] to thwart SAT attack by increasing the time taken by each iteration rather than increasing the number of iterations. These schemes, however, were recently broken by reduced-encoding approach and neural-network-based attacks [KAHS20, SHP20, AKHS20].

1.3 Contribution

Recently, cascaded locking (CAS-Lock) [SXTF20a] was proposed as a resilient defense against state-of-the-art attacks, including SAT and structural attacks, all-the-while

Table 1: Summary of attack efficacy against different post-SAT logic locking techniques. ✓ denotes attack success, whereas ✗ denotes attack failure.

	Bypass [XSTF17]	Removal [YMSR17]	FALL [SS20]	SFLL-HD-Unlock [YTS19]	GNN-Unlock [APK+21]	This work
SARLock [YMRS16]	✓	✓	✗	✗	✗	✗
Anti-SAT [XS18]	✓	✓	✗	✗	✓	✓
SFLL-HD [YSN+17]	✗	✗	✓	✓	✓	✗
CAS/M-CAS [SXTF20a]	✗	✗	✗	✗	✗	✓

Table 2: An overview of proposed attacks under different adversarial threat models.

	Oracle-less	Oracle-guided
CAS		IFS KBM-SAT
M-CAS	IFS-SAT	

maintaining non-trivial output corruption. CAS-Lock has two variants: (1) CAS and (2) Mirrored CAS (M-CAS).[1] The authors establish the security of the schemes with rigorous proofs. However, in this work, we challenge such claims by exposing a set of structural vulnerabilities that can successfully break both CAS and M-CAS defenses. We define two adversarial threat models, viz., oracle-guided and oracle-less on which we carry out our attacks. The main contributions of this work are shown in Table 2, and are summarized below.

- **Attacking CAS.** We propose two attacks against CAS in the oracle-guided model:
  - **Identify flip signal (IFS).** This attack exploits the structural traces to recover the original IP.
  - **Key-bit mapping & SAT (KBM-SAT).** This attack exploits the connectivity of key-inputs, thereby enabling the SAT attack to decipher the secret key with only a polynomial number of queries as opposed to the theoretical expectation of an exponential number of queries in the key-size.
- **Attacking M-CAS.** We demonstrate an attack on M-CAS in the oracle-less model:
  - **Identify flip signal & SAT (IFS-SAT).** We exploit structural flaws in M-CAS, and upon identification, successfully launch SAT attack to recover the secret key. Note that our SAT attack can be launched in an oracle-less setting, whereas the original SAT attack requires an oracle, i.e., a functional IC with the secret key loaded onto its memory. To this end, we substitute the oracle with a generic CAS-block that will be described later.
- We provide extensive simulation results for ISCAS'85 and ITC'99 benchmarks. The success rates for IFS and KBM-SAT attacks against CAS are ∼93% (14/15) and 100% (15/15), respectively. Further, the success rate of the IFS-SAT attack against M-CAS is also ∼93% (14/15).
- Finally, we establish the effectiveness of our attack under different technology libraries and synthesis tools, and as a caution urge the designers to refrain from merely relying on state-of-the-art synthesis tools for a secure implementation.

2 Background

Before delving into our attacks, we provide the details that would be necessary for the rest of the paper.

2.1 Adversarial threat model

Prior to analyzing the security of any scheme, it is critical to accurately identify, communicate, and assess the potential threats within the context of adversarial capabilities. Depending on these capabilities, we define two adversarial models:

[1]Hereafter, we abbreviate CAS-Lock as CAS, and mirrored CAS as M-CAS.

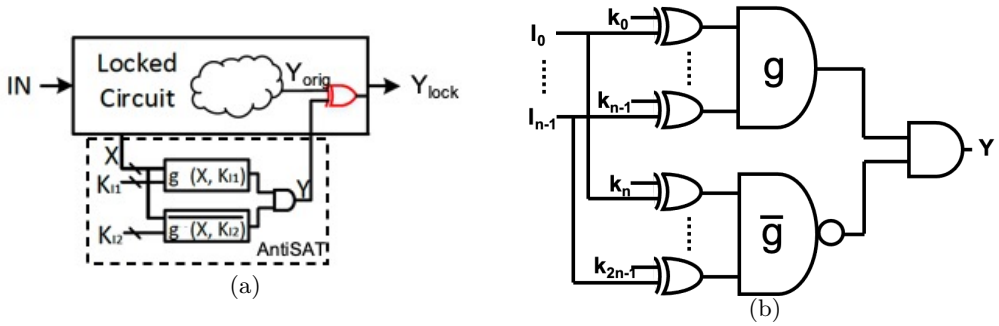


Figure 3: (a) Overview of Anti-SAT [XS18]. (b) Circuit to generate flip signal Y.

- **Oracle-less.** The adversary is assumed to reside at an untrusted foundry which is usually well-equipped with high-end tools, and thus can perform reverse-engineering. Note that she has access to the GDSII representation of the logic-locked design IP, thus enabling her to reverse engineer the GDSII file to obtain the locked netlist. However, access to an oracle, i.e., a working chip with the correct key, is not considered as it only becomes available during later stages of the supply chain. The adversary's goal is to identify, isolate, and infer the secret key-bit values from the structure and examples include De-synthesis [MZGT17], SPS [YMSR17], FALL [SS20, YTS19] attacks. We demonstrate our IFS-SAT attack against M-CAS in this setting.

- **Oracle-guided.** The adversary has access to *(1) an oracle, i.e., a working chip with the correct key loaded onto its memory, as well as (2) a reverse-engineered locked netlist.* A likely scenario would be a colluding adversary at an untrusted foundry and during field-use. Besides, we assume full knowledge of the technology mapping, synthesis tools, etc. used during locking. Note that this is a valid assumption as off-the-shelf synthesis tools are available from major EDA vendors, such as Synopsys Design Compiler (DC) and Cadence RTL Compiler (RC). Now, the adversary can simulate the netlist to produce meaningful input patterns rather than brute-forcing and can apply these input patterns to the working chip to decipher the secret key which is unknown in simulations. Indeed, many attacks assume this threat model; well-known examples include sensitization attack [YRSK15] and SAT attack [SRM15]. We demonstrate our IFS and KBM-SAT attacks against CAS in this setting.

2.2 Anti-SAT

2.2.1 Design

The design of CAS directly follows Anti-SAT [XS18], which is shown in Fig. 3a. Note that Anti-SAT is a point function-based defense, where the outputs of two complementary Boolean functions g and $\overline{g}$ are ANDed together to generate the point function Y. This Y then flips a high observability net such as a primary output (PO). The two blocks g and $\overline{g}$ takes two different n-bit keys K_{l1} and K_{l2}; it produces $Y = 1$ for incorrect keys, thereby, corrupting the PO. The internal architecture of the Anti-SAT block is shown in Fig. 3b. Note that setting K_{l1} equal to K_{l2} produces $Y = 0$ for all input patterns, thus, unlocking the chip and ensuring correct (corruption-free) functionality.

2.2.2 Security analysis

The security of Anti-SAT stems from the difficulty of setting the two n-bit keys as same, *since the bit-wise mapping between them is unknown to the adversary.* Note that if an adversary is able to figure out this bit-wise mapping, then setting $K_{l1} = K_{l2}$ becomes trivial, and thus, leads to a successful attack. This is given by the following equation:

$$\begin{aligned} Y &= g(I \oplus K_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l2})} \\ &= g(I \oplus K) \quad \wedge \quad \overline{g(I \oplus K)}, \quad K_{l1} = K_{l2} = K \\ &= g(J) \quad \wedge \quad \overline{g(J)}, \quad J = I \oplus K \\ &= 0, \quad \forall I \end{aligned} \tag{1}$$

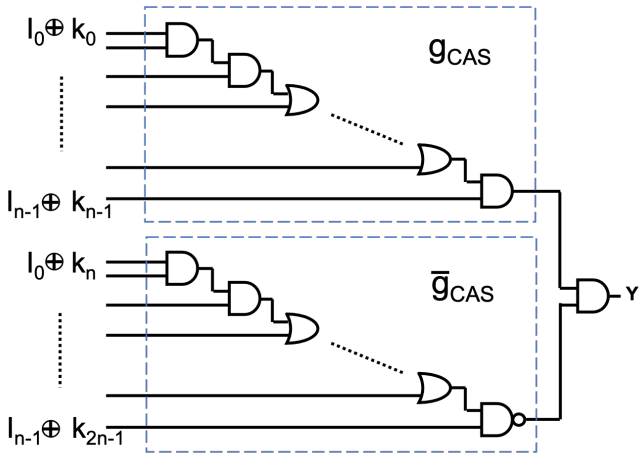


Figure 4: Circuit to generate the flip signal Y in CAS [SXTF20a].

2.3 CAS-Lock

Figure 4 shows the architecture of CAS, where the outputs of two complementary Boolean functions g_{cas} and $\overline{g}_{cas}$ are ANDed together to produce the flip signal Y. However, instead of the AND-tree structure in Anti-SAT, CAS incorporates a *daisy-chained structure of alternating AND-OR gates*. The flip signal Y of CAS, when XORed with the primary output of the circuit, manifests error similar to Fig. 3a. As in Anti-SAT, a user must set the key K_{l1} same as K_{l2} to unlock the design. CAS improves over Anti-SAT on multiple fronts: 1) it is resilient against bypass attack [XSTF17], 2) it prevents AppSAT by ensuring high output corruption [SLM+17], and 3) it prevents signal probability skew (SPS) attack [YMSR17].

Existing attack. Recently, a trivial yet highly effective attack has been demonstrated against CAS [SS19]. As powerful this attack may be, it can be thwarted by a relatively simple countermeasure *by including a random combination of XOR/XNOR of key-inputs with the primary inputs (PI) before feeding it to the CAS* block [SXTF20b]. This improved implementation ensures that the key for g_{CAS} is never equal to that of $\overline{g_{CAS}}$, i.e., $K_{l1} \neq K_{l2}$. The random combination of XOR/XNORs can be expressed as follows:

$$Y = g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l2} \oplus R_{l2})}$$

, where

$$I \oplus K_{l1} \oplus R_{l1} = \begin{cases} I[i] \oplus K_{l1}[i], \text{ if } R_{l1}[i] = 0 \\ I[i] \odot K_{l1}[i], \text{ if } R_{l1}[i] = 1 \end{cases} \qquad I \oplus K_{l2} \oplus R_{l2} = \begin{cases} I[i] \oplus K_{l2}[i], \text{ if } R_{l2}[i] = 0 \\ I[i] \odot K_{l2}[i], \text{ if } R_{l2}[i] = 1 \end{cases}$$

$$R_{lj} \xleftarrow{\$} \{0,1\}^n, j = 1, 2$$

, $R_{lj}[i]$ denotes the i-th bit of R_{lj}, $I[i]$ denotes the i-th bit of I, and $K_{lj}[i]$ denotes the i-th bit of K_{lj}. Note that R_{lj} *is an n-bit binary string selected uniformly randomly from the set of all n-bit binary strings* that is unknown to the adversary. K_{l1}, K_{l2} is a correct key for CAS *iff*

$$K_{l1} \oplus K_{l2} = R_{l1} \oplus R_{l2} = R \tag{2}$$

This is clear from the following equation,

$$\begin{aligned} Y \quad &= g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l2} \oplus R_{l2})} \\ &= g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus (K_{l1} \oplus R) \oplus R_{l2})} \quad , K_{l2} = K_{l1} \oplus R \text{ from Eqn. (2)} \\ &= g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l1} \oplus R_{l1} \oplus \cancel{R_{l2}} \oplus \cancel{R_{l2}})} \\ &= g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l1} \oplus R_{l1})} \\ &= 0, \quad \forall I \end{aligned}$$

For the remainder of this paper, *we consider this improved version of CAS [SXTF20b] and take a deeper look into it.* We also assume this improved version for Anti-SAT while carrying out attacks on it.

3 Attacking CAS

As described earlier, the security of CAS stems from the unintelligible mapping between the key-inputs following a series of compilation steps using a state-of-the-art synthesis tool. However, contrary to this assumption, all synthesis tools have been developed for co-optimizing area, power, and timing, and thus, security was never considered. Consequently, we expose a set of vulnerabilities in CAS, left behind by synthesis tools, to develop two new attacks: (1) *identify flip signal (IFS)* attack, and (2) *key-bit mapping & SAT (KBM-SAT) attack.* In the first attack, we show that the reliance on state-of-the-art synthesis tools leaves structural traces, enabling an adversary to recover the original IP through re-synthesis. In the second attack, we go one-step further by recovering the secret key. While both attacks can successfully recover the original design IP, the second attack is more potent than the first one since it can decipher the secret key; knowledge of the secret key enables an adversary to unlock overproduced chips.

3.1 Identify flip signal (IFS)

The security of CAS is critically linked to the synthesis of the locked circuit and *the dissolution of the traces* of the CAS block structure. In our attack, a simple re-synthesis of the reverse-engineered locked circuit reveals the structure of CAS, and thus, the flip signal Y.[2] Afterward, simply fixing Y to a constant logic zero eliminates the protection offered by CAS. This is shown in the equation below:

$$Y_{lock} = Y_{orig} \oplus Y = Y_{orig} \oplus 0 = Y_{orig}$$

The question that remains to be answered is how to identify the *flip signal* Y in the netlist. To this end, we utilize the following property of signal Y:

– Merge condition: All the key-inputs always merge at Y (see Fig. 4).

Thus, any net containing all the key-inputs in its fanin cone, would constitute a likely candidate for Y.[3] Now, consider the path from Y to the locked PO, Y_{lock} (see Fig. 3a). It is clear that all the nets in this path also satisfy the merge condition, and therefore, constitute multiple likely candidates. To uniquely identify the flip signal Y, we select the net that is *topologically* farthest from Y_{lock} and also satisfies the merge condition. Accordingly, we develop the methodology shown in Algorithm 1, and describe it below.

Methodology. First, all the key-inputs K are identified from the locked netlist Y_{lock}. Next, we obtain Y_{re} from the reverse-engineered locked netlist Y_{lock} by re-synthesizing with certain constraints. We define two variables, viz., *cur* and *prev*, initialized to the primary output (PO) and ϕ, respectively. Next, we check if the fanin cone of *cur* contains all key-inputs, i.e., if it satisfies the merge condition. If yes, then we iteratively trace backward, otherwise, we output *prev* as the flip signal. We recover the correct value of the flip signal *prev* using an automatic test pattern generation (ATPG) tool. We construct a miter-like circuit Y_{miter} with two instances of the locked netlist, where the two circuits share the primary inputs as shown in Fig. 5b. In one instance, denoted by Y_{lock0}, we assign logic 0 to the *prev* signal, whereas in the other instance, denoted by Y_{lock1}, it is assigned logic 1. Next, an ATPG tool is leveraged to provide a test pattern IN_{tp} that detects a stuck-at-0 fault (s-a-0) at the output of this miter-like circuit (i.e., return an input pattern for which the output becomes logic 1). Note that such IN_{tp} guarantees that the outputs of Y_{lock0} and Y_{lock1} are different. Next, we query the oracle with IN_{tp}, and obtain the corresponding output OUT_{tp}. Further, we obtain output values for IN_{tp} for each of the locked instances, viz., Y_{lock0} and Y_{lock1}. Finally, the correct value of the flip signal is identified by comparing the outputs of the oracle to that of these locked instances.

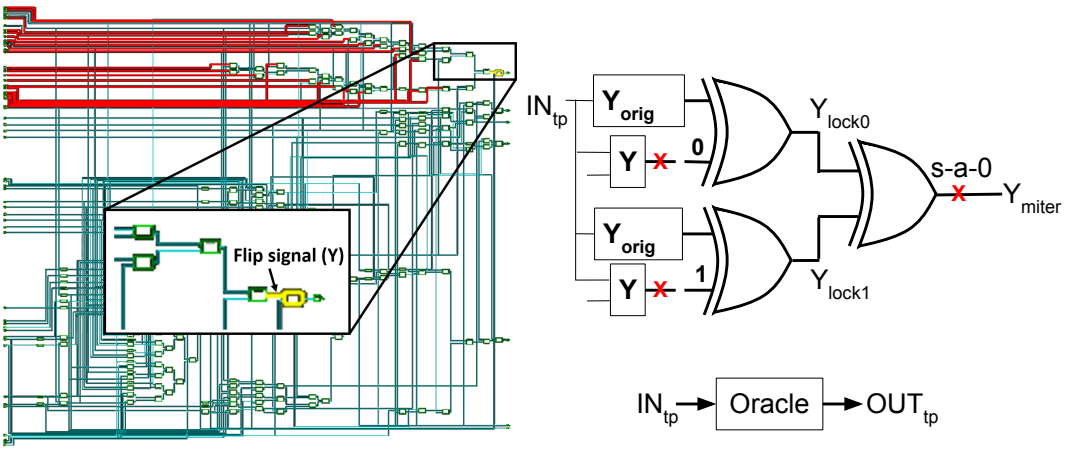
[2] Note that as commercial synthesis tools are readily available, an adversary can always re-synthesize a locked design with parameters of her choice.

[3] Throughout the paper, we use signal/net interchangeably.

Algorithm 1: Identify Flip Signal (IFS) attack

Input: Locked netlist Y_{lock}, Functional IC C
Output: Unlocked netlist Y_{unlock}

$K \leftarrow$list_keyinputs(Y_{lock});
$Y_{re} \leftarrow$re_synth(Y_{lock});
$cur \leftarrow$identify_PO(Y_{re});
$prev \leftarrow \phi$;
// Check fanin cone of cur
while $K \subseteq$ fanin(cur) **do**
  $prev \leftarrow cur$;
  $cur \leftarrow$ find_prev_net(cur);
end
// Verify if $prev = 0$ or $prev = 1$
$Y_{lock0} \leftarrow$ create_ckt($Y_{lock}, prev = 0$);
$Y_{lock1} \leftarrow$ create_ckt($Y_{lock}, prev = 1$);
$Y_{miter} \leftarrow$ create_miter(Y_{lock0}, Y_{lock1});
$IN_{tp} \leftarrow$ ATPG(Y_{miter}, s-a-0);
$OUT_{tp} \leftarrow C(IN_{tp})$;
if $Y_{lock1}(IN_{tp}) == OUT_{tp}$ **then**
  prev = 1;
else
  prev = 0;
end
$Y_{unlock} \leftarrow$ re_synth($Y_{lock}, prev$);



(a) (b)

Figure 5: (a) Execution of IFS attack on c432 locked with 20-bit key. The flip signal Y is clearly visible, marked in yellow. The red lines mark the paths from the key-inputs. (b) Automatic test pattern generation (ATPG)-aided identification of flip signal value. Conforming to Fig. 3a, Y_{orig} corresponds to the original netlist and Y is the flip signal. (top) Generation of test pattern IN_{tp} to detect stuck-at-0 fault at the output. (bottom) Correct response OUT_{tp} for input pattern IN_{tp} using oracle.

This accounts for any possible inversions and we successfully return the unlocked netlist Y_{unlock} by fixing the flip signal to the identified constant value.

Example. Consider the example in Fig. 5a, where the attack is launched on c432 circuit locked with a 20-bit key. Note that the CAS structure is easily discernible, as all the key-inputs (shown in red) merge into the flip signal Y (as shown in the zoomed figure). Fig. 5b showcases the approach using ATPG to decipher the flip value of Y by querying the oracle. The naming convention is consistent with Fig. 3a.

Limitation. The above IFS attack suffers from two drawbacks. First, such exploitable flip signal may not always exist in the circuit. Second, it fails to decipher the secret key from the netlist. The following attack successfully overcomes these issues, described next.

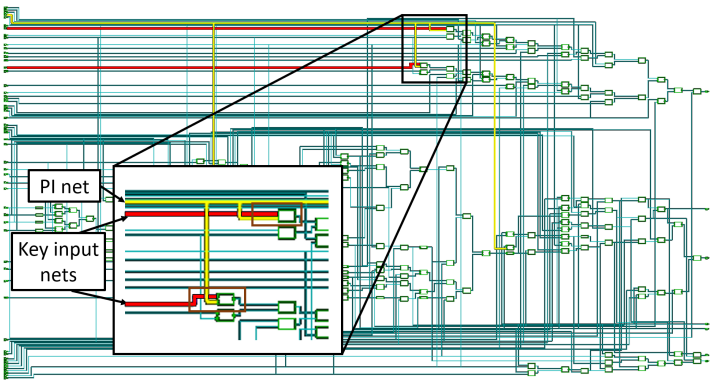


Figure 6: Execution of KBM attack on c432 locked with a 20-bit key. Orange boxes in the zoomed figure show that the PI G118GAT is connected to both k_0 and k_{10}, thereby establishing (k_0, k_{10}) as a bit-symmetric pair.

3.2 Key-bit mapping & SAT (KBM-SAT)

We now propose a key-recovery attack using a two-step approach involving key-bit mapping, followed by the SAT attack [SRM15].

3.2.1 Key-bit mapping (KBM)

It is evident from Fig. 4 that the i-th key-input from both secret keys K_{l1} and K_{l2} are always XORed with the *same primary input* (PI). We say that the key-input pair (k_a, k_b) is *bit-symmetric* if the following properties hold:

- k_a and k_b merge with a fanout from the same PI.
- k_a is the i-th bit of K_{l1}, and k_b is the i-th bit of K_{l2}.

In Fig. 4, k_0 and k_n, first key-inputs of the two secret keys K_{l1} and K_{l2}, respectively, are XOR-ed with I_0. This establishes (k_0, k_n) as a bit-symmetric pair. Intuitively, checking against the PI connection reveals the bit-symmetry for each pair of key-inputs. Note that due to different optimizations applied by the synthesis tool, at times the key-inputs may get connected to an internal net instead of a PI. However, we empirically verified that bit-symmetry still holds, i.e., i-th key-input from K_{l1} merges onto the same internal net as i-th key-input from K_{l2}. We exploit this bit-symmetry to successfully decipher the complete bit-wise mapping between the two secret keys.

Methodology. The methodology is shown in Algorithm 2. We initialize the search set S with all the key-inputs, and the map set M to NULL. A key-input k_i is picked at random from the search set S, and checked for bit-symmetry. To this end, we list all the N that connect to k_i, and all nets V that connect to the nets in N. If there exists a key-input k_j in V and $k_j \neq k_i$, then (k_i, k_j) is a bit-symmetric pair. Thus, we add this pair to map set M, and remove k_i and k_j from search set S. Otherwise, it fails to find bit-symmetric pair for k_i, and is removed from S. We repeat these steps for all the key-inputs in S. In the end, the mapping M between the secret keys K_{l1} and K_{l2} is returned.

Example. Consider the example in Fig. 6, where the attack is launched on c432 circuit locked with a 20-bit key. Note that key-input k_0 (shown in red) is connected to the PI $G118GAT$ (shown in yellow). If we trace PI $G118GAT$, we can see the connection to another key-input k_{10}. This immediately establishes (k_0, k_{10}) as a bit-symmetric pair.

3.2.2 Launching SAT

Note that despite identifying the complete bit-wise mapping, setting $K_{l1} \oplus K_{l2} = R$ is non-trivial as R is unknown to the adversary. This prompts us to derive the following mathematical condition that can successfully decipher the secret key by launching the SAT attack against CAS. From Eqn. (2) we get,

$K_{l2} = K_{l1} \oplus R$, R is a uniformly random n-bit binary string unknown to the adversary.

Algorithm 2: KBM-SAT attack

Input: Locked netlist Y_{lock}, Functional IC C
Output: Secret key K_{l1}, K_{l2}
// Launch KBM
$S = \{k_0, k_1 \cdots k_{2n-1}\}$;
$M = \phi$;
$Y \leftarrow$re_synth(Y_{lock});
while $S \neq \phi$ **do**
  $k_i \xleftarrow{\$} S$;
  $N \leftarrow$all_connect(k_i, Y);
  $V \leftarrow$all_connect(N, Y);
  // Check if $\exists$ key-input in set V
  if $\exists k_j \in V$ & $k_j \neq k_i$ **then**
    $M = M \cup \{\{k_i, k_j\}\}$;
    $S = S - \{k_i, k_j\}$;
  else
    $S = S - \{k_i\}$;
  end
end
// Launch SAT to decipher key [SRM15]
while $M \neq \phi$ **do**
  $\{k_i, k_j\} \leftarrow$ pick_pair(M);
  $k_i = 0$;
  $M = M - \{\{k_i, k_j\}\}$;
end
$K_{l1}, K_{l2} \leftarrow$ SAT(Y_{lock}, C);

Without loss of generality (WLOG), let us assume that we can set $K_{l1} = 0$ as the bit-wise mapping is already known. It is immediately clear that $K_{l1} = 0, K_{l2} = R$ constitutes a correct key for CAS according to Eqn. (2),

$$\begin{aligned} Y \quad &= g(I \oplus K_{l1} \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus K_{l2} \oplus R_{l2})} \\ &= g(I \oplus 0 \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus R \oplus R_{l2})} \\ &= g(I \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus R_{l1} \oplus \cancel{R_{l2}} \oplus \cancel{R_{l2}})} \\ &= g(I \oplus R_{l1}) \quad \wedge \quad \overline{g(I \oplus R_{l1})} \\ &= 0, \quad \forall I \end{aligned}$$

Note that the security of CAS/Anti-SAT relies on the fact that the *total key space is* 2^{2n}*, and a SAT solver can only eliminate* 2^n *incorrect keys at each iteration,* thus forcing it to iterate an exponential number of times in the key-size n. However, fixing $K_{l1} = 0$ immediately refutes this property, as with $K_{l2} = R$, R being an unknown n-bit binary string, *the total key space reduces to* 2^n *from* 2^{2n}. This directly violates the SAT-resiliency of CAS/Anti-SAT, thereby deciphering the secret key within only a polynomial number of iterations as opposed to the exponential number of iterations in the key-size mandated by Anti-SAT/CAS. We showcase the success of KBM-SAT in Section 5.2.2, where it can successfully decipher the secret key for all the locked circuits. *Note that setting* $K_{l1} = 0$ *may not always be possible for an adversary, however, as long as one of the i-th bits from* K_{l1} *or* K_{l2} *is fixed to a constant value, i.e., 0/1, the above property holds; i.e., the total key space reduces to* 2^n*,* and is thus vulnerable to the SAT attack [SRM15].

Methodology. The methodology is shown in Algorithm 2. We first trace the netlist to identify bit-wise symmetric key-bit pairs. Next, we pick a random pair of key-bits $\{k_i, k_j\}$ from the map set M. Afterward, we fix one of the key-inputs to a constant value, i.e., to 0/1, and leave the other key-input unknown. We repeat this step for all the mapped pairs in M, and finally launch the SAT attack with this partial key information. Finally, SAT attack returns the complete key K_{l1}, K_{l2}.

3.3 Improving on CAS

In all fairness, the authors did anticipate such structural attacks against CAS, and thus forwarded the following three strategies to thwart such attacks. Below, we discuss the strengths and weaknesses of each strategy.

- **RLL-integrated CAS.** The structure of CAS is obfuscated by incorporating random logic locking (RLL) in an asymmetric fashion [RKM10]. This prevents an adversary from performing structural analysis, as she has no knowledge of the RLL key. However, this poses little/no challenge, as RLL can be easily peeled off from any RLL-integrated compound locking technique using AppSAT [SLM+17]. This is acknowledged in the paper itself, where CAS+RLL gets broken ∼50% of the time, and is thus, disregarded from further consideration.
- **AND/OR-camouflaged CAS.** The structure of CAS is obfuscated with AND/OR gate camouflaging [Syp17]. This thwarts any structural analysis as well, restricting the adversary to an oracle-less model. However, for a secure implementation of camouflaging, *the foundry needs to be trusted*, which directly breaks one of the fundamental promises of logic locking. Further, a camouflaged netlist can easily be transformed into an equivalent logic locked netlist [YS15], which can then be broken using AppSAT [SLM+17] as in RLL-integrated CAS. Therefore, we discard AND/OR-camouflaged CAS from our attack, as it fails to provide any additional benefit to the overall security of CAS.
- **Mirrored CAS (M-CAS).** Finally, the authors also present an improved version of CAS, called Mirrored CAS (M-CAS), shown in Fig. 7. It is claimed to be secure against any structural attacks such as our proposed IFS and KBM-SAT attacks. *In this light, we consider M-CAS as the strongest defense amongst the three strategies, and accordingly, devise an attack that we describe in detail in the next section.*

4 Attacking M-CAS

An overview of Mirrored CAS (M-CAS) is shown in Fig. 7. M-CAS is implemented by locking the original circuit with two back-to-back CAS blocks with two keys K_{secret} and K_{CAS}. The key K_{secret} for the first CAS block (shown in red), is hardcoded into the netlist; it is not an input that is driven from the tamper-proof memory. After integrating the first CAS block into the design, a second CAS block (shown in blue) is invoked with identical structure as the first one. However, here the key K_{CAS} is applied from the tamper-proof memory through the key-input ports. Only upon applying $K_{CAS} = K_{secret}$, the correct functionality can be recovered as follows:

$$\begin{aligned} Y_{OUT} &= Y_{orig} \oplus Y_{secret} \oplus Y_{CAS} \\ &= Y_{orig} \oplus \cancel{Y_{secret}} \oplus \cancel{Y_{secret}}, \quad K_{CAS} = K_{secret} \\ &= Y_{orig} \end{aligned} \tag{3}$$

The authors claim that this implementation is secure against structural attacks, as removing the key-controlled second CAS block Y_{CAS} does not pose any threat. This is due to the fact that removing Y_{CAS} leaves the adversary with a non-functional design Y_{mod} (shown with the dashed box in Fig. 7), which is immediate from the following equation:

$$\begin{aligned} Y_{OUT} &= Y_{orig} \oplus Y_{secret} \oplus Y_{CAS} \\ &= Y_{orig} \oplus Y_{secret} \oplus 0 \\ &= Y_{mod} \end{aligned} \tag{4}$$

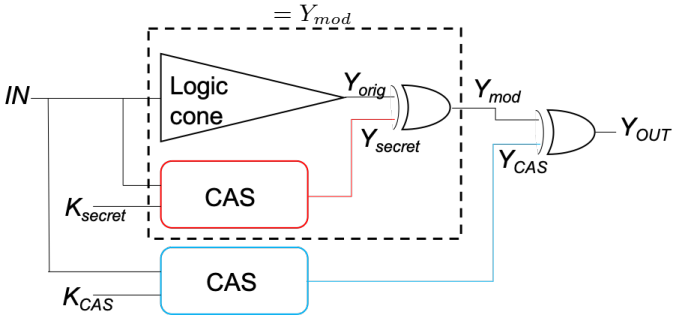


Figure 7: Architecture of M-CAS which uses two CAS blocks with keys K_{secret} and K_{CAS}.

Challenge. The authors assume that as the key K_{secret} is hardcoded into the design, proper synthesis steps such as *constant propagation, bubble push, and technology mapping* would dissolve any structural traces. Thus, any attempt to inspect the netlist for identifying the flip signal Y_{secret}, and consequently recovering the key K_{secret} is deemed futile. In this section, *we re-examine this assumption, and subsequently disprove it.* As the SAT resilience of M-CAS relies on this assumption, the defense can be compromised by developing an attack that targets this assumption. As such, we demonstrate an attack that can launch SAT on M-CAS to decipher the secret key K_{secret}.

Our attack, called IFS-SAT, aims to inspect the locked netlist for any structural traces to decipher the secret key K_{secret}. It consists of three steps: (1) peel off Y_{CAS}, (2) identify flip signal Y_{secret}, and (3) launch SAT attack to decipher K_{secret}.

4.1 Peel off Y_{CAS}

This can be achieved in a couple of ways as discussed in Section 3, viz., IFS or KBM-SAT attack. Note that the security of M-CAS is solely reliant on Y_{secret}, and not on Y_{CAS} as acknowledged by the authors. Indeed this assumption is followed in other prior works such as SFLL [YSN+17, SNL+20]. It is assumed that the *restore unit* can easily be identified by tracing the key-inputs (analogous to IFS), and thus be removed from the design as shown in [SS20, YTS19]. However, removing Y_{CAS} does not defeat the security of M-CAS, as the adversary only recovers Y_{mod} shown in Eqn. (4). Thus, in this paper, we concentrate on the later part, i.e., breaking Y_{mod}.

4.2 Identify flip signal Y_{secret}

Next, we concentrate on identifying the flip signal Y_{secret} (IFS part of IFS-SAT) from the Y_{mod} block. Note that if this is possible, then the adversary can recover the original IP by simply setting $Y_{secret} = 0$ as given below:

$$\begin{aligned} Y_{mod} &= Y_{orig} \oplus Y_{secret} \\ &= Y_{orig} \oplus 0 \\ &= Y_{orig} \end{aligned} \tag{5}$$

The question is how to identify the flip signal Y_{secret}. With careful observation, we identify the following three properties of the flip signal Y_{secret} that could help us shortlist the possible candidates.

1. Fan-in cone of Y_{secret} must contain *exactly* n PIs, where n is the number of PIs in CAS.

2. From Fig. 4, it is clear that the structure of Y_{secret} contains *at most* $2n$ two-input gates, excluding buffers/inverters. Note that due to internal optimizations applied by the synthesis tools, few gates may get merged. However, the upper bound for the number of gates remains unaffected.

3. From Fig. 4, we see that Y_{secret} block exhibits a *linear structure*. Note that due to alternate sequence of AND/OR gates, the scope of common optimization techniques such as *path balancing* is limited, and we can reasonably expect Y_{secret} to retain this property after synthesis with a constrained library, i.e., with two-input gates.

Example. Consider the example in Fig. 8, where the `c432` circuit locked with a 64-bit key using M-CAS is shown. Here we ignore Y_{CAS}, and only concentrate on Y_{secret} embedded with the original `c432` circuit. The flip signal Y_{secret} and its fanin cone are marked in red. We see that all the three preceding properties hold in this case; *1) the fan-in cone contains exactly 32 PIs, 2) the number of gates is 31, excluding buffers/inverters, and 3) it retains the linear structure.*

Thus, the preceding three criteria help in shortlisting possible candidates for the flip signal Y_{secret}. Note that in a few scenarios, a single one of the three criteria could be sufficient to uniquely identify the flip signal. If applying a criterion uniquely identifies the flip signal, we immediately stop and return the current candidate as the solution. However, if there exists ambiguity, i.e., there are multiple possible candidates, we move to the next

Algorithm 3: IFS-SAT attack

Input: Peeled netlist Y_{mod}, Functional IC C, # PIs in CAS n
Output: Secret key K_{SAT}

$T \leftarrow$ fanin(Y_{mod});
$N \leftarrow$ list_nets(T);
$S_1 = \phi, S_2 = \phi, S_3 = \phi$;
// Check # primary inputs in fanin of net
for $x \in N$ **do**
  $m \leftarrow$ num_inputs_fanin(x);
  if $m == n$ **then**
    $S_1 = S_1 \cup x$;
  end
end
// Check for unique solution
if $|S_1| == 1$ **then**
  $cur = S_1$;
  goto 29;
end
// Check # gates in fanin of net
for $x \in S_1$ **do**
  $m \leftarrow$ num_gates_fanin(x);
  if $m \leq 2n$ **then**
    $S_2 = S_2 \cup x$;
  end
end
// Check for unique solution
if $|S_2| == 1$ **then**
  $cur = S_2$;
  goto 29;
end
// Check linear structure
for $x \in S_2$ **do**
  if *is_fanin_linear*(x) **then**
    $S_3 = S_3 \cup x$;
  end
end
$cur = S_3$;
// Verify if $cur = 0$ or $cur = 1$
$Y_{lock0} \leftarrow$ create_ckt($Y_{mod}, cur = 0$);
$Y_{lock1} \leftarrow$ create_ckt($Y_{mod}, cur = 1$);
$Y_{miter} \leftarrow$ create_miter(Y_{lock0}, Y_{lock1});
$IN_{tp} \leftarrow$ ATPG(Y_{miter}, s-a-0);
$OUT_{tp} \leftarrow$ C(IN_{tp});
if $Y_{lock1}(IN_{tp}) == OUT_{tp}$ **then**
  $cur = 1$;
else
  $cur = 0$;
end
$Y_{secret} \leftarrow$ fanin(cur);
// Launch SAT to decipher key [SRM15]
$K_{SAT} \leftarrow$ SAT(Y_{secret}, Y_{CAS});

criterion to further prune the candidate set. Therefore, we advocate applying the above criteria in sequence to minimize run-time.

We iteratively search all the nets in the circuit to uniquely identify the flip signal Y_{secret}. Afterward, the correct value of the flip signal is identified by comparing the outputs of the oracle to that of the fixed flip-signal netlist similar to Algorithm 1. Finally, simply fixing the flip signal to the constant logic recovers the original IP.

Limitation. Although the previous steps do recover the original IP, it suffers from a couple of drawbacks. First, *it succeeds in the oracle-guided model,* i.e., deciphering the correct value of the flip signal requires a working oracle. Second, it fails to decipher the secret key K_{secret} from the netlist. Thus, in the next section, we present a different attack that successfully overcomes these issues.

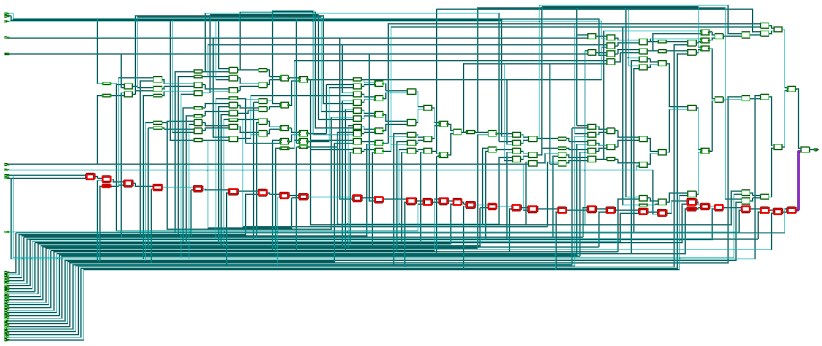


Figure 8: Identifying the hardcoded CAS block in the c432 M-CAS-locked design. The flip signal Y_{secret} is highlighted in purple, and its structure is marked in red.

4.3 Decipher the key K_{secret}

If K_{secret} is recovered, then setting $K_{CAS} = K_{secret}$ is trivial, and thus, the original IP can be recovered according to Eq. (3). To this end, we leverage the well-known SAT attack [SRM15], where we treat Y_{secret} *of the locked netlist which we can simulate, as the working oracle, and* Y_{CAS} *as the logic-locked circuit.* Thus, once the flip signal Y_{secret} is identified, the SAT attack is launched on Y_{CAS} [SRM15] to recover K_{SAT}. Next, setting K_{CAS} equal to K_{SAT} unlocks the design. Note that the SAT attack could return a key $K_{SAT} \neq K_{secret}$, however, as long as it satisfies the condition $Y_{CAS} = Y_{secret}$, any K_{SAT} suffices to unlock an M-CAS-locked IC. This is clear from Eqn (3). Contrary to all the illustrations of SAT attack in the literature that only succeed in oracle-guided model, our IFS-SAT attack is *launched in an oracle-less setting.* We accomplish this by considering the key-controlled CAS block as the locked circuit, and the extracted cone from flip signal Y (containing the hardcoded key) as the original circuit for the SAT attack, eliminating the need for an oracle. The complete methodology for IFS-SAT is shown in Algorithm 3.

Note that this is a direct counter-example to the theoretical SAT-resilience of M-CAS. At large, SAT is considered ineffective against M-CAS. However, state-of-the-art synthesis tools leave unintentional vulnerabilities in the circuit that allows us to apply various structural attacks such as IFS, followed by SAT attack to decipher the secret key K_{secret}.

5 Experimental results

5.1 Experimental setup

In this section, the results are presented for our attacks on ISCAS-85 and ITC-99 benchmarks; we conduct our experiments on the largest logic cone of each circuit.[4] All the experiments are carried out on a 24 core Intel Xeon processor running at 2.5GHz having 264 GB RAM. The circuits are synthesized using Synopsys Design Compiler (DC) with 65nm GlobalFoundries LPe technology. Moreover, the ISCAS-85 benchmarks are locked with 64-bit key as proposed in CAS-Lock [SXTF20a], whereas ITC-99 benchmarks are locked with 160-bit key. Further, in our experiments, we use Cadence Conformal LEC to formally verify the logical equivalence between the recovered design and the original design, and thus the success of our attacks.

5.2 Attacks on CAS

5.2.1 IFS attack

We present the results for this attack in the following scenarios:

- **Full library.** Table 3 summarizes the results for IFS attack on CAS-locked circuits *without constraining* the technology library. It is clear that the attack breaks 14/15 circuits, i.e., has a success rate of ∼93%. The second column shows the correct value for the flip signal, which can be easily deciphered by verifying against a working

[4]Note that the CAS/M-CAS locked circuits are not open-sourced by the authors, thus, we implement CAS/M-CAS by ourselves, and upload CAS/M-CAS scripts, all locked benchmarks, and all attacks for the community in [cas21].

Table 3: IFS attack on CAS-locked circuits synthesized with 1) full technology library, and 2) constrained technology library, i.e., with two input gates. "-" indicates the attack failed to find the flip signal.

Benchmark	Full technology library			Constrained technology library		
	Flip signal value	Level	Execution time (sec)	Flip signal value	Level	Execution time (sec)
c432	0	2	13	1	2	13
c499	1	2	13	1	3	13
c1355	1	2	12	0	4	13
c1908	1	2	13	1	3	13
c2670	0	2	13	0	2	13
c3540	1	5	13	0	4	13
c5315	0	2	12	0	2	12
c7552	0	2	13	0	2	12
b14_C	1	2	13	1	2	12
b15_C	0	2	13	1	2	13
b17_C	1	2	13	0	2	13
b18_C	-	-	-	-	-	-
b20_C	1	3	13	1	2	13
b21_C	1	2	13	1	2	13
b22_C	1	2	13	0	2	13

oracle as described in Algorithm 1. Note that due to the technology mapping and the optimizations applied by the synthesis tool, the flip value varies randomly among circuits. However, the signal trace still remains which is identified by the IFS attack. Further, the third column shows the circuit depth from the PO at which such flip signals exist. It is evident that such signals are present close to the locked PO, except for `c3540`. The execution time for our attack is reported in the fourth column, where we see even for large circuits such as `b17_C` with 30K+ gates, the attack terminates within only a few seconds.

- **Constrained library.** In this particular case, we re-synthesize the circuits by using only two-input gates from the technology library. The results are reported in Table 3. The attack is able to break 14/15 circuits, i.e., at a success rate of ∼93%. Further, the attack terminates within only a few seconds for all circuits.

Note that the attack success is independent of the type of library cells used. This is expected as IFS exploits connectivity of key-inputs to identify the flip signal instead of structural analysis such as SPS attack [YMSR17].

5.2.2 KBM-SAT attack

KBM attack. The results from launching KBM on the CAS-locked circuits are presented in Table 4. Column 2 highlights the number of key-bits successfully mapped. For ISCAS-85 benchmarks, there are a total of 64 key-bits, and all key-bits are successfully mapped for all circuits, whereas for ITC-99 benchmarks, there are a total of 160 key-bits, and all key-bits are successfully mapped for all circuits as well, except `b18_C` and `b22_C`.

SAT attack. Building on the key-bit mapping information from KBM attack, we launch the traditional SAT attack [SRM15] as described in Section 5.2.2. To this end, we fix one key-input from each identified pair to a constant value, and the resulting circuit is fed to the SAT solver to decipher the key. Columns 3, 4, and 5 report the corresponding results, where it can be seen that it breaks 15/15 circuits, i.e., a success rate of 100%. Note that the number of iterations required have reduced to 208 from the theoretically expected 2^{32} for ISCAS-85 locked benchmarks, and to 1379 from the theoretically expected 2^{80} for ITC-99 locked benchmarks. Finally, column 5 shows the execution time for the SAT attack, which is less than 4 minutes even for large circuits such as `b18_C` having 100K+ gates. Note that for certain instances such as `b18_C` and `b22_C`, KBM failed to identify all key-bit mappings; in such cases the unidentified key-inputs are left untouched, i.e., we did not fix them to any value, instead let the SAT solver handle them. However, it can be seen from the results that only a few unidentified key-inputs do not hinder the SAT solver in any significant way.

Effectiveness of KBM-SAT. To further validate the effectiveness of the KBM-SAT attack, we lock the c432 circuit with four different key sizes, viz., 14, 16, 18, and 20 with CAS, and perform SAT without and with the KBM information. The results are plotted

Table 4: KBM-SAT attack results on ISCAS-85 benchmarks locked with a 64-bit key and ITC-99 benchmarks locked with a 160-bit key.

Benchmarks	Key-bit mappings	Key recovered?	# SAT iterations	Execution time (sec)
c432	64	Yes	156	0.95
c499	64	Yes	208	1.64
c1355	64	Yes	152	0.92
c1908	64	Yes	98	0.66
c2670	64	Yes	162	1.18
c3540	64	Yes	85	3.36
c5315	64	Yes	84	0.78
c7552	64	Yes	147	1.11
b14_C	160	Yes	420	15.5
b15_C	160	Yes	325	16.94
b17_C	160	Yes	249	11.51
b18_C	156	Yes	457	210.39
b20_C	160	Yes	511	27.89
b21_C	160	Yes	915	136.43
b22_C	154	Yes	1379	188.4

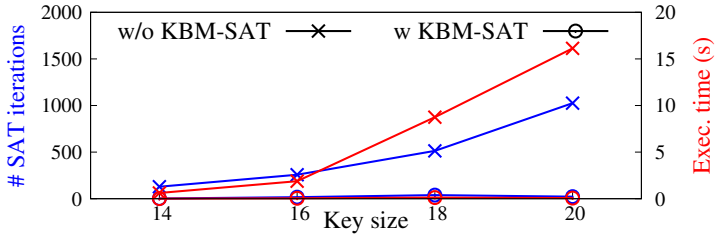


Figure 9: Performance of the SAT attack [SRM15] on CAS with and without KBM information on the `c432` circuit locked with key sizes 14, 16, 18, and 20.

in Fig 9, where it can be seen that the number of SAT iterations and the execution time conform to the theoretically expected exponential growth in the key size without the KBM information, whereas with the KBM information, they grow only linearly in the key size, thereby breaking CAS.

5.3 Attacks on M-CAS

In this section, we highlight the results of IFS-SAT carried out on M-CAS in oracle-less model. As IFS-SAT consists of two major steps, namely, IFS and SAT, we present the results separately.

5.3.1 IFS attack

Similar to IFS against CAS, we conduct our experiments on M-CAS-locked circuits in two different settings: full technology library and constrained technology library.

- **Full library.** Table 5 summarizes the results of launching IFS attack on M-CAS-locked circuits synthesized with a full technology library. It successfully identifies the flip signal in 14/15 circuits, i.e., has a success rate of ∼93%. In the second column, we report the flip signal value which can be deciphered by verifying the circuit output against a working oracle. Further, such signal always exists at a close proximity from the PO. The execution time of the attack remains within few seconds even for large circuits such as `b17_C`. Note that the attack fails to identify the flip signal for `b18_C`. As we are using full technology library, the flip signal might have truly merged within the circuit, though the chances of such an event remains low.

- **Constrained library.** We resort to constraining the technology library to only two-input gates, and launch IFS after re-synthesizing with this constrained library. The results are summarized in Table 5, where it can be seen that the success rate is 14/15, i.e., ∼93%.

Table 5: IFS-SAT attack on M-CAS-locked circuits synthesized with 1) full technology library, and 2) constrained technology library, i.e., with two-input gates. "-" indicates the attack failed to find the flip signal.

Benchmark	IFS						SAT			Total execution time (sec)
	Full technology library			Constrained technology library			Key recovered?	# SAT iterations	Execution time (sec)	
	Flip signal value	Level	Execution time (sec)	Flip signal value	Level	Execution time (sec)				
c432	1	2	13	1	2	13	Yes	358	4	17
c499	1	2	13	1	3	13	Yes	84	1	14
c1355	0	2	14	0	2	13	Yes	6646	1398	1411
c1908	1	2	13	0	2	13	Yes	89	1	14
c2670	0	2	12	0	2	13	Yes	70	1	14
c3540	0	4	13	1	4	13	Yes	68	1	14
c5315	0	2	13	0	2	12	Yes	160	2	14
c7552	1	2	13	1	2	13	Yes	4335	672	685
b14_C	0	2	14	1	2	13	Yes	327	8	21
b15_C	1	2	13	0	2	13	Yes	441	15	28
b17_C	0	2	15	0	2	14	Yes	1161	104	118
b18_C	-	-	-	-	-	-	-	-	-	-
b20_C	0	2	14	1	2	15	Yes	169	4	19
b21_C	0	2	14	0	2	14	Yes	1196	137	151
b22_C	1	2	14	1	2	14	Yes	937	99	113

5.3.2 SAT attack

Post-IFS, we move to the SAT step to recover the secret key from the netlist. Table 5 also summarizes the results for IFS-SAT attack on M-CAS-locked circuits. The attack successfully deciphers the secret key for circuits whose flip signal has been identified during IFS. In the ninth column, we report the number of SAT iterations required by the SAT attack to break the circuits. The number of SAT iterations can be as small as 84 (`c499`) and as large as 6646 (`c1355`); we don't necessarily observe a trend in the number of SAT iterations against the circuit size.

5.4 Effectiveness of our attack

In this section, we discuss different aspects of our attack, and demonstrate its effectiveness across these settings.

5.4.1 Effect of key-size

Here, we investigate the effect of key size on the effectiveness of IFS-SAT on M-CAS-locked circuits. The results are presented in Table 6, where the attack succeeds across key-sizes, viz., 64, 128, 160, and 256. Note that in all cases, the flip signal is present immediately before the PO. Further, the number of SAT iterations increases only linearly in key size, thereby, refuting the theoretical SAT-resilience that dictates an exponential increase.

5.4.2 Scalability

The execution time for IFS-SAT on M-CAS-locked circuits is reported in the last column of Table 5. It is clear that the SAT step is the heaviest and slowest component of the entire attack that takes up to few minutes to complete, whereas the IFS step takes only a few seconds. Yet, the overall attack time remains within few minutes even for large benchmarks such as `b17_C` having 30K+ gates, establishing the scalability of our attack.

5.4.3 Effectiveness of selection criteria

Recall that we discussed three properties of the flip signal that can be exploited to shortlist possible candidates, viz., 1) input size, 2) gate count, and 3) linearity. Here, we

Table 6: Effect of key-size on IFS-SAT against M-CAS-locked `b14_C` circuit.

Key-size	Flip signal value	Level	Key recovered?	# SAT iteration	Execution time (sec)
64	0	2	Yes	124	14
128	1	2	Yes	1488	101
160	1	2	Yes	327	21
256	0	2	Yes	5013	2552

Table 7: Effectiveness of the selection criteria in identifying the flip signal in M-CAS. Each column shows the # candidates after applying the corresponding criterion. "-" indicates the criterion need not be applied as the flip signal has been uniquely identified in the previous step.

Benchmark	Criterion			Success?
	#1	#2	#3	
c432	2	1	-	Yes
c499	2	2	1	Yes
c1355	1	-	-	Yes
c1908	1	-	-	Yes
c2670	1	-	-	Yes
c3540	10	1	-	Yes
c5315	2	1	-	Yes
c7552	1	-	-	Yes
b14_C	1	-	-	Yes
b15_C	1	-	-	Yes
b17_C	1	-	-	Yes
b18_C	1	-	-	No
b20_C	5	1	-	Yes
b21_C	5	1	-	Yes
b22_C	2	1	-	Yes

Table 8: Effect of technology nodes on the effectiveness of IFS on M-CAS-locked circuits. "-" indicates the attack failed to find the flip signal.

Benchmark	32nm			65nm		
	Flip signal value	Level	Execution time (sec)	Flip signal value	Level	Execution time (sec)
c432	1	2	2	1	2	13
c499	1	3	2	0	3	13
c1355	1	3	2	0	2	13
c1908	1	2	2	0	2	12
c2670	1	2	2	0	2	13
c3540	0	5	2	0	6	13
c5315	1	2	2	1	2	13
c7552	1	2	2	0	2	12
b14_C	1	2	3	0	2	13
b15_C	1	2	2	1	2	13
b17_C	0	2	4	1	2	14
b18_C	-	-	-	-	-	-
b20_C	1	2	3	1	2	14
b21_C	1	2	3	0	2	13
b22_C	1	19	4	1	2	14

investigate the effectiveness of these criteria in identifying the flip signal in M-CAS. The results are presented in Table 7, where we report the number of candidates after applying each criterion. It is seen that the first criterion alone uniquely identifies the flip signal ∼53% of the time. For the remainder, we apply the second criterion, which successfully identifies the flip signal in all but one case, i.e., `c499`. The third criterion, i.e., the linear structure is used only once for `c499` to uniquely identify the flip signal. However, even if criterion #1 uniquely identifies a candidate for the flip signal for `b18_C`, the attack fails as the identified signal is not the actual flip signal. This could be attributed to the fact that the flip signal might indeed have merged with the original circuit, though the probability of such an event remains low as seen from our experiments.

5.4.4 Effect of technology nodes

Here, we investigate the effect of different technology nodes, viz., 32nm and 65nm on the effectiveness of our attack. *Note that different technology nodes have different sets of cell types that can lead to varying optimizations by the synthesis tools.* To this end, we launch the IFS attack on M-CAS-locked circuits synthesized with 32nm and 65nm technology libraries, and the corresponding results are reported in Table 8. It is clear that the attack is successful across technology nodes, except one, i.e., `b18_C`. However, the degree of difficulty of the attack varies among benchmarks. For example, the existence of the flip signal is unclear for `b22_C` in 32nm, as it lies deep inside the circuit structure. Nonetheless, the attack succeeds in 14/15 cases for both technologies, as well as the execution time remains within few seconds. Note that once IFS is successful, launching SAT attack becomes trivial, and is omitted for the sake of brevity.

Table 9: Effect of Synopsys Design Compiler and Cadence RTL Compiler synthesis tools on the effectiveness of IFS on M-CAS-locked circuits for 65nm technology node. "-" indicates the attack failed to find the flip signal.

Benchmark	Synopsys Design Compiler			Cadence RTL Compiler		
	Flip signal value	Level	Execution time (sec)	Flip signal value	Level	Execution time (sec)
c432	1	2	12	0	2	13
c499	0	3	12	1	3	13
c1355	0	3	12	1	3	13
c1908	1	2	12	0	2	12
c2670	1	2	12	0	4	12
c3540	0	8	13	-	-	-
c5315	0	2	13	1	3	13
c7552	1	2	13	0	2	13
b14_C	0	2	13	0	2	14
b15_C	0	2	13	0	2	14
b17_C	1	2	14	1	2	15
b18_C	-	-	-	0	2	28
b20_C	0	2	14	1	3	14
b21_C	1	2	14	1	2	15
b22_C	1	2	14	0	3	14

Table 10: Extending IFS attack to Anti-SAT. The circuits are synthesized with 1) full technology library, and 2) constrained technology library, i.e., with two input gates. "-" indicates the attack failed to find the flip signal.

Benchmark	Full technology library			Constrained technology library		
	Flip signal value	Level	Execution time (sec)	Flip signal value	Level	Execution time (sec)
c432	1	2	13	1	2	13
c499	1	2	13	0	3	13
c1355	1	2	12	0	3	13
c1908	1	2	12	0	2	12
c2670	1	2	13	0	2	13
c3540	1	5	13	0	5	13
c5315	0	2	13	0	2	13
c7552	1	2	13	0	2	13
b14_C	1	2	13	1	2	13
b15_C	0	2	13	0	2	13
b17_C	1	2	12	1	2	13
b18_C	-	-	-	-	-	-
b20_C	1	2	13	1	2	13
b21_C	0	2	13	1	2	13
b22_C	0	2	13	1	2	13

5.4.5 Effect of synthesis tools

It is clear from the above results that state-of-the-art synthesis tools leave behind traces that enable different attacks. *Thus, it is interesting to study the effects of different synthesis tools on IFS-SAT as they can lead to different optimizations, resulting in differing locked netlists.* To this end, we investigate the effect of two industry-leading synthesis tools on our attack, viz., Synopsys Design Compiler (DC) and Cadence RTL Compiler (RC), and the results are presented in Table 9. It can be observed that the effect of synthesis tools is minimal on the attack; the success rate for DC and RC is ∼93% and 100%. This is a further empirical evidence that synthesis tools fail to merge the structure of M-CAS, independent of tool/library/technology node used.

6 Discussion

6.1 Applicability to other locking techniques

The attacks developed in this paper are not limited to CAS/M-CAS, and they can be easily extended to other variants of point function-based locking techniques such as Anti-SAT [XS18] or SFLL [YSN+17]. The structural analysis carried out in IFS attack (see Section 3) against CAS, can also be applied to break the closely-related Anti-SAT technique. To this end, we launch our IFS attack on Anti-SAT, and the results are presented in Table 10. It can be seen that IFS breaks 14/15 circuits, i.e., a success rate of ∼93%.

Building on this result, we posit that the structural analysis of the IFS-SAT attack against M-CAS, can be leveraged against other techniques such as SFLL [YSN+17] and

SFLL-fault [SNL+20]. As these techniques also rely on point functions for SAT resilience, the additional structure could be identified by tweaking the three criteria that are described in IFS (see Section 4.2), however, we leave it as an open problem for future work.

Nevertheless, the last step of the IFS-SAT attack, i.e., deciphering the secret key using SAT, is generic that can be applied to any *structurally vulnerable* point function-based locking technique. Note that this is an improvement over the traditional SAT attack [SRM15] that *only succeeds in the oracle-guided model*, while, our IFS-SAT can be launched in an *oracle-less model*, where the flip signal is simulated as the oracle.

6.2 Limitation of state-of-the-art synthesis tools

To thwart the SAT attack, researchers have proposed several point function-based locking techniques such as Anti-SAT [XS18], SARLock [YMRS16], SFLL [YSN+17], SFLL-fault [SNL+20], CAS-Lock [SXTF20a] etc. However, most/all of these techniques have been broken by different structural attacks [YMSR17, YTS19, SS20, XSTF17]. This is attributed to the fact that the implementation of point function requires insertion of additional circuit into the netlist such as an AND-tree [SNL+20]. However, commercially available state-of-the-art synthesis tools fail to blend this structure into the circuit, leaving behind traces that subsequently leads to its identification as has been demonstrated in our attack, as well as in a plethora of other works [YMSR17, XSTF17, YTS19, SS20].

This naturally raises the following question, *"is it possible to securely implement a point function-based locking technique that thwarts any structural attack?"* In Meerkat, the authors argue that it is indeed possible [MZGT17]. The crux of the technique is to leverage canonical representations of boolean functionality via reduced ordered binary decision diagrams (ROBDDs) to achieve indistinguishability obfuscation (i$\mathcal{O}$). Note that the application of an i$\mathcal{O}$ obfuscator allows a designer to prove that a locked netlist do not reveal any information about the secret key. However, ROBDD is inefficient, does not scale, and thus, can not be applied for practical purposes. Although efficient i$\mathcal{O}$ obfuscators do exist such as [GGH+16], the power, performance, and area (PPA) overheads incurred are so large that they are of little practical interest at this time. When specifically talking about application specific integrated circuit (ASIC) design that can not tolerate the slightest of PPA degradation, application of such i$\mathcal{O}$ obfuscators is deemed totally impractical.[5] This leaves the designers at the mercy of commercially available state-of-the-art synthesis tools that unfortunately fail to provide adequate security against structural attacks.

6.3 Preventing structural attacks

Given the limitation of state-of-the-art synthesis tools, there are roughly three ways to achieve SAT-resilience without relying on such tools.

The first approach is to insert structures that create hard SAT instances, thereby, throttling the effectiveness of the SAT solver. Representative techniques following this approach include Full-Lock [KAHS19], where SAT-hard logic and routing blocks are inserted and InterLock [KAHS20], where inter-correlated logic and routing locking is used.

A second approach involves locking the scan chains in a circuit to thwart oracle access. Note that scan chains are inserted to facilitate thorough testing of the circuit. However, since SAT attack works only on combinational logic, it leverages these scan chains to access the internal combinational logic. Hence, obfuscating the stimuli and response of scan flip-flops can thwart the SAT attack [KCK18, KCK19, WZH+17, KKC19]. But, caution must be taken as some schemes have been shown to be vulnerable to modeling-based attacks due to linear obfuscation of scan chains [AYL+19, LS20].

Finally, a new direction to thwart SAT attack was proposed in DisORC, where the secret key is withdrawn from the key-register whenever access to scan chains is detected [LKK+20]. Instead of obfuscating the scan stimuli and responses, DisORC detaches the secret key coming from tamper-proof memory and instead feeds user-defined key to the key-registers. SAT attack launched on this setup returns the key fed by the user, clearing any traces of the secret key in the outcome.

[5] Note that logic locking was introduced to protect ASIC design.

7 Related work

Several attacks have been proposed against existing point function-based techniques such as SARLock [YMRS16], Anti-SAT [XS18], and SFLL [YSN+17]. One such attack, called Bypass attack, can break SARLock and Anti-SAT by stitching additional logic onto the SAT-resistant circuit. Further, compound locking schemes where traditional locking schemes are compounded with point function based techniques to thwart aforementioned attacks were also short-lived by variants of SAT-based attack [SLM+17, LPS21]. However, the current CAS/M-CAS technique is secure against such an attack by construction that has been theoretically established in [SXTF20a].

Examples of structural attacks include signal probability skew (SPS) and AppSAT-guided removal attack (AGR) [YMSR17] that can identify and remove the SAT-resistant logic from the circuit, thereby recovering the original design IP. Though these attacks have successfully been mounted against Anti-SAT and SARLock, M-CAS remains secure against such attacks by virtue of its construction.

Recently, a functional analysis-based logic locking attack (FALL) [SS20] has been proposed against a similar locking technique SFLL [YSN+17]. The attack exploits mathematical properties of Hamming distance (HD)-based SFLL techniques, viz., unateness, non-overlapping errors, and sliding window. These properties can successfully decipher a hardcoded key from a SFLL-locked netlist. However, these functional properties do not hold for CAS/M-CAS, and as such FALL cannot be applied. *However, the unateness property applies individually to g_{cas} and $\overline{g}_{cas}$, and thus, we believe that it may be applied to find possible candidates for the flip signal. Nevertheless, this would require further complex mathematical formulations, and the attack fails to break CAS/M-CAS in its current form.*

A new structural analysis to identify the flip signal is presented in [YTS19], where the authors establish that the flip signal exhibits a tree-like structure. However, this attack relies on manual inspection of the netlist, whereas our attack is completely automated. Further, the tree-like structure characteristic is inapplicable to M-CAS, making M-CAS secure against [YTS19].

Concurrent to this work, a graph neural network-based work is introduced in [APK+21]. This work leverages neural network to learn structural features of the SAT-resilient logic, and subsequently predict its existence. However, the prediction accuracy depends on the training data set, and unlike our attack, it fails to recover the secret key that can enable overbuilding.

A comparative overview of our attack against the state-of-the-art attacks is presented in Table 1, from which it is immediately clear that our attack complements the other attacks, as it is the only one which successfully breaks CAS/M-CAS.

8 Conclusion

In this paper, we present a series of attacks that break the newly proposed locking technique CAS/M-CAS. First, we present two new attacks against CAS, viz., IFS and KBM-SAT in the oracle-guided model that successfully defeat advanced versions of CAS by exploiting its implementation flaws. Next, we break the improved version of CAS, viz., mirrored CAS (M-CAS) by developing IFS-SAT that exploits certain structural characteristics of M-CAS; we launch this attack in an oracle-less setting.

Through extensive experiments, we establish the efficacy of our attacks against different technology libraries, for different library cells, against different synthesis tools, for varying key-sizes, and for multiple point function-based schemes. Further, these experiments support our claim that *state-of-the-art synthesis tools can introduce unknown/unintentional pitfalls in a design* and we urge researchers to consider these shortcomings while developing logic locking techniques.

References

[AKHS20] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[APK+21] Lilas Alrahis, Satwik Patnaik, Faiq Khalid, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur Sinanoglu. GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking. In *IEEE Design, Automation and Test in Europe Conference (DATE)*, pages 1–6, 2021.

[AYL+19] Lilas Alrahis, Muhammad Yasin, Nimisha Limaye, Hani Saleh, Baker Mohammad, Mahmoud Alqutayri, and Ozgur Sinanoglu. ScanSAT: Unlocking Static and Dynamic Scan Obfuscation. *IEEE Transactions on Emerging Topics in Computing*, 2019.

[big18] The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies. https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies, 2018.

[cas21] Breaking CAS-Lock. https://github.com/DfX-NYUAD/Breaking_CAS-Lock, 2021.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[KAHS19] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks. In *ACM Annual Design Automation Conference (DAC)*, pages 1–6, 2019.

[KAHS20] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Interlock: An intercorrelated logic and routing locking. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[KCK18] Rajit Karmakar, Santanu Chatopadhyay, and Rohit Kapur. Encrypt flip-flop: A novel logic encryption technique for sequential circuits. *arXiv preprint arXiv:1801.04961*, 2018.

[KCK19] Rajit Karmakar, Santanu Chattopadhyay, and Rohit Kapur. A scan obfuscation guided design-for-security approach for sequential circuits. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(3):546–550, 2019.

[KKC19] Rajit Karmakar, Harshit Kumar, and Santanu Chattopadhyay. Efficient Key-gate Placement And Dynamic Scan Obfuscation Towards Robust Logic Encryption. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2019.

[LKK+20] Nimisha Limaye, Emmanouil Kalligeros, Nikolaos Karousos, Irene G. Karybali, and Ozgur Sinanoglu. Thwarting All Logic Locking Attacks: Dishonest Oracle with Truly Random Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2020.

[LPS21] Nimisha Limaye, Satwik Patnaik, and Ozgur Sinanoglu. FaSAT- Fault-aided SAT-based attack on Compound Logic Locking Techniques. In *IEEE Design, Automation and Test in Europe Conference (DATE)*, pages 1–6, 2021.

[LS20] Nimisha Limaye and Ozgur Sinanoglu. DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys. In *Design, Automation Test in Europe Conference Exhibition*, pages 270–273, 2020.

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

[pen13] Detecting and Removing Counterfeit Semiconductors in the U.S. Supply Chain. https://www.semiconductors.org/wp-content/uploads/2018/06/ACTF-Whitepaper-Counterfeit-One-Pager-Final.pdf, 2013.

[re112] Intel's 22-nm Trigate Transistors Exposed. http://electroiq.com/chipworks-real-chips-blog/2012/04/24/intels-22-nm-trigate-transistors-exposed, 2012.

[re212] iPhone 5 A6 SoC reverse engineered, reveals rare hand-made custom CPU, and tri-core GPU. http://www.extremetech.com/computing/136749-iphone-5-a6-soc-reverse-engineered-reveals-rare-hand-made-custom/-cpu-and-a-tri-core-gpu, 2012.

[RKK14] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[RKM10] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Ending piracy of integrated circuits. *Computer*, 43(10):30–38, 2010.

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

[SHP20] Joseph Sweeney, Marijn JH Heule, and Lawrence Pileggi. Modeling techniques for logic locking. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[SLM+17] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. AppSAT: Approximately deobfuscating integrated circuits. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100, 2017.

[SLPJ18] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin. Cross-lock: Dense layout-level interconnect locking using cross-bar architectures. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI)*, pages 147–152, 2018.

[SNL+20] Abhrajit Sengupta, Mohammed Nabeel, Nimisha Limaye, Mohammed Ashraf, and Ozgur Sinanoglu. Truly Stripping Functionality for Logic Locking: A Fault-based Perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4439–4452, 2020.

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

[SS19] Abhrajit Sengupta and Ozgur Sinanoglu. CAS-Unlock: Unlocking CAS-Lock without Access to a Reverse-Engineered Netlist. *IACR Cryptology ePrint Archive*, 2019:1443, 2019.

[SS20] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. *IEEE Transactions on Information Forensics and Security*, 15:2514–2527, 2020.

[SXTF20a] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):175–202, 2020.

[SXTF20b] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. Defeating CAS-Unlock. *IACR Cryptology ePrint Archive*, 2020:324, 2020.

[Syp17] SypherMedia. SypherMedia Library Circuit Camouflage Technology. http://www.smi.tv/syphermedia_library_circuit_camouflage_technology.html, 2017.

[Tec17] TechInsights. Samsung Galaxy S8 (SM-G950W) Teardown. http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s8-teardown, 2017.

[WZH+17] Xiaoxiao Wang, Dongrong Zhang, Miao He, Donglin Su, and Mark Tehranipoor. Secure scan and test using obfuscation throughout supply chain. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 37(9):1867–1880, 2017.

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

[XSTF17] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks. In *International conference on cryptographic hardware and embedded systems (CHES)*, pages 189–210. Springer, 2017.

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

[YMSR17] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal attacks on logic locking and camouflaging techniques. *IEEE Transactions on Emerging Topics in Computing*, 8(2):517–532, 2017.

[YRSK15] Muhammad Yasin, Jeyavijayan JV Rajendran, Ozgur Sinanoglu, and Ramesh Karri. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1411–1424, 2015.

[YS15] Muhammad Yasin and Ozgur Sinanoglu. Transforming between logic locking and IC camouflaging. In *IEEE International Design & Test Symposium (IDT)*, pages 1–4, 2015.

[YSN+17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1601–1618, 2017.

[YSS+17] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. What to Lock? Functional and Parametric Locking. In *ACM Great Lakes Symposium on VLSI 2017*, page 351–356, 2017.

[YTS19] Fangfei Yang, Ming Tang, and Ozgur Sinanoglu. Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd)–Unlocked. *IEEE Transactions on Information Forensics and Security*, 14(10):2778–2786, 2019.