Threats on Logic Locking: A Decade Later

Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, Avesta Sasan
George Mason University, Fairfax, VA, USA
{kzamiria,hmardani,hhomayou,asasan}@gmu.edu

ABSTRACT

To reduce the cost of ICs and to meet the market's demand, a considerable portion of manufacturing supply chain, including silicon fabrication, packaging and testing may be pushed offshore. Utilizing a global IC manufacturing supply chain, and inclusion of non-trusted parties in the supply chain has raised concerns over security and trust related challenges including those of overproduction, counterfeiting, IP piracy, and Hardware Trojans to name a few. To reduce the risk of IC manufacturing in an untrusted and globally distributed supply chain, the researchers have proposed various locking and obfuscation mechanisms for hiding the functionality of the ICs during the manufacturing, that requires the activation of the IP after fabrication using the key value(s) that is only known to the IP/IC owner. At the same time, many such proposed obfuscation and locking mechanisms are broken with attacks that exploit the inherent vulnerabilities in such solutions. The past decade of research in this area, has resulted in many such defense and attack solutions. In this paper, we review a decade of research on hardware obfuscation from an attacker perspective, elaborate on attack and defense lessons learned, and discuss future directions that could be exploited for building stronger defenses.

KEYWORDS

Reverse Engineering, Logic Locking, SAT Attack, SMT Attack

ACM Reference Format:
Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, Avesta Sasan. 2019. Threats on Logic Locking: A Decade Later. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* https://doi.org/10.1145/3299874.3319495

1 INTRODUCTION

The increasing cost of IC manufacturing has pushed several stages of the semiconductor device's manufacturing supply chain offshore [33]. However, many of these offshore facilities are identified as untrusted entities. Processing and fabrication of ICs in an untrusted supply chain poses a number of challenging security threats such as IP piracy and IC overproduction [14]. To counter these threats, various hardware design-for-trust techniques have been proposed. The term *logic locking, a.k.a.* hardware obfuscation, surfaced in 2008 by EPIC [6], in which a limited programmability was introduced into a netlist by means of inserting additional key programmable gates (KG)s at design time. After fabrication, the functionality of the IC is programmed by loading the correct key values. The key inputs could be stored in, and loaded from, an on-chip tamper-proof memory [21]. The purpose of inserting KGs is to hide the IC's functionality from untrusted foundries. Since the functionality of a design is locked with a secret key, the attacker cannot learn the functionality of the locked netlist after reverse engineering. Insertion of n KGs hides the ICs functionality between 2^n different circuit possibilities, each generated by a different key. The correct functionality will be recovered when the loaded n-bit key is correct.

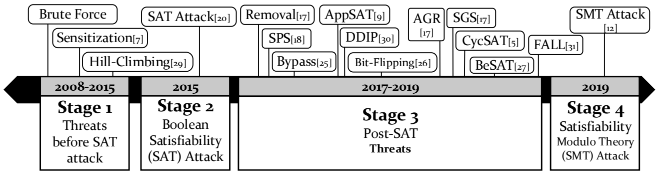


Figure 1: Categorization of attacks against logic locking schemes.

EPIC, however did not end the threat against IP piracy (or other related concerns), as this solution and many other obfuscation solutions that were proposed over the last decade were broken using various carefully crafted attacks. A decade of research in this area, has resulted in a wide range of defense [1, 3, 4, 7, 8, 10, 13, 15, 16, 19, 23, 24, 28, 32] and attack solutions [2, 5, 7, 9, 12, 17, 18, 20, 22, 25–27, 29–31]. In this paper, we review many of these obfuscation solutions, explain and reviewing most notable attack mechanisms, summarize and compare the effectiveness of obfuscation solutions against these attacks, and describe the strength and weaknesses of various obfuscation and attack solutions. As illustrated in Fig. 1, the defense and attack solutions related to hardware obfuscation, based on functionality, capability, effectiveness and time-line are categorized into four categories: (1) *Test-Inspired Attacks* that were mostly inspired from test concepts and attempted to discover the obfuscation key value based on the location of KGs, described in Section 2. (2) SAT Attack, formulation and revelation of which significantly affected the direction and presumed assumptions of the hardware obfuscation research community, explained in Section 3. (3) Post-SAT Attacks where the focus of hardware security researchers changed to the design of an attack against obfuscation solutions that resist the SAT attack, explained in 4. And (4) SMT Attack as a universal attack platform capable of instantiating many theory solvers to act as pre- post- or co- processors to the SAT solver, described in Section 5. We conclude the paper in Section 6 by summarizing the effectiveness of attacks discussed in this paper and provide a short discussion on new opportunities related to designing secure logic locking solutions.

2 STAGE 1: TEST-BASED ATTACKS

2.1 Brute Force Attack

The *brute force* attack is the most intuitive attack against obfuscated circuits. This attack exhaustively search for the correct key by testing all key and input values. For instance, assuming that adversary has access to the reverse-engineered netlist, and considering that the circuit has four PIs ($i_{0..4}$) and two KIs ($k_{0..1}$), an exhaustive search may result in applying of $2^{2+4} = 64$ test patterns (in the worst case) and checking the output against an activated (functionally correct) chip to verify correctness. Based on the number of primary inputs ($|PI|$) and the number of key bits ($|KI|$), the number of possible test patterns is ($2^{|PI|+|KI|}$). Hence, the search space for a brute force attack is extremely large, making the attack even for small circuits and small number of keys unfeasible in a reasonable amount of time. For example, a small circuit with 20 input

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3319495

Table 1: Classification of KGs in Sensitization Attack.

Term	Description	Strategy used by attacker
Runs of KGs	Back-to-Back KGs	Replacing by a Single KG
Isolated KGs	No Path between KGs	Finding Unique Pattern per KG (Golden Pattern (GP))
Dominating KGs	$k1$ is on Every Path between $k0$ and POs	Muting $k0$, Sensitizing $k1$
Concurrently Mutable Convergent KGs	Convergent at a Third Gate Both can be Propagated to POs	Muting $k0/k1$, Sensitizing $k1/k0$
Sequentially Mutable Convergent KGs	Convergent at a Third Gate One can be Propagated to POs	Determining $k1$ by GP, Update the Netlist, Target $k0$
Non-Mutable Convergent KGs	Convergent at a Third Gate None can be Propagated to POs	Brute Force Attack

pins, which is obfuscated with 80 key gates poses 2^{100} possible test pattern. An attacker can reduce the number of test patterns using functional test or random test, in which the exponential impact of $|PI|$s will be eliminated, and only $2^{|KI|} \times (func\_test\_patterns)$ is required for brute force attack. But even with this change, the attack time is exponential with respect to the number of key gates. EPIC [6] used a random KG insertion policy referred to as random logic locking (RLL). Using RLL, EPIC reasoned that by replacing a small percentage of gates (or insertion of KGs), the obfuscation can resist brute force attacks.

2.2 Sensitization Attack

After introducing EPIC [6], Rajendran *et al.* [7] proposed a *sensitization* attack, which determines individual key values, in a time linear to the |KI|, by applying patterns that sensitize key values to the output. As its name implies, sensitization of an internal wire (key bit) **L** to an output **O** means that the value of **L** can be propagated to **O** and thus any change on **L** is observable on **O**. After determining an input pattern that propagates the value of the key-bit to the output, the attacker applies the input pattern to a functional IC (An IC activated and programmed with the correct key that could be obtained from market). The correct key value will be propagated to an output by applying this pattern to the functional IC. The attacker observe and record this output as the value of the sensitized key-bit. The propagation of a key-bit to the output is heavily depending on the location of the KGs, hence, they classify KGs based on their location and discuss corresponding attack strategies for each case. The summary of strategies and techniques used in the sensitization attack is reflected in Table 1. To prevent sensitization attack they proposed SLL, in which the KGs are inserted in locations with maximum mutual interference. In SLL the attacker cannot sensitize the key-bit values to a primary output. Similar to SLL, several prior-art methods in the literature, including fault-analysis (FLL), LUT-based locking, etc. [1, 3, 7, 8, 24], tried to maximize the complexity of obfuscation using different KGs replacement strategies.

2.3 Random-based Hill-Climbing Attack

Plaza *et al.* [29] developed a new algorithmic attack that uses test patterns and observe responses. Unlike sensitization attack [7], their proposed approach does not require netlist access. They propose a randomized local key-searching algorithm to search the key that can satisfy a subset of correct input/output patterns. The algorithm proposed in [29] is iterative in nature. At first, it selects random value for key bits and then at each iteration, the key bits, which are selected randomly, are toggled one by one. The target is to minimize the frequency of differences between the observed and expected responses. Hence, a random key candidate is gradually improved based on observed test responses. If no solution is found in one iteration, the algorithm resets the key to a new random key value. However, the complexity of this attack quickly increases with increasing number of KGs.

3 STAGE 2: SAT ATTACK

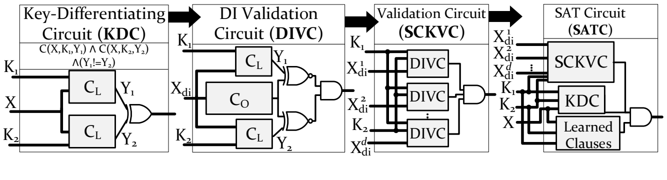
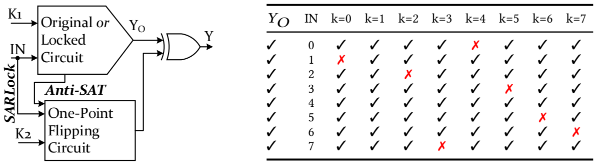

Key-Differentiating Circuit (KDC)
C(X,K₁,Y₁) ∧ C(X,K₂,Y₂) ∧(Y₁!=Y₂)
K1
X
K2
CL
CL
Y1
Y2
DI Validation Circuit (DIVC)
K1
Xdi
K2
CL
CO
CL
Y1
Y2
Validation Circuit (SCKVC)
K1
X1di
X2di
K2
Xddi
DIVC
DIVC
DIVC
SAT Circuit (SATC)
X1di
X2di
Xddi
K1
K2
X
SCKVC
KDC
Learned Clauses

Figure 2: SAT Attack Iterative Flow.



Y_O	IN	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
✓	0	✓	✓	✓	✓	✗	✓	✓	✓
✓	1	✗	✓	✓	✓	✓	✓	✓	✓
✓	2	✓	✓	✗	✓	✓	✓	✓	✓
✓	3	✓	✓	✓	✓	✓	✗	✓	✓
✓	4	✓	✓	✓	✓	✓	✓	✓	✓
✓	5	✓	✓	✓	✓	✓	✓	✗	✓
✓	6	✓	✓	✓	✓	✓	✓	✓	✗
✓	7	✓	✓	✓	✗	✓	✓	✓	✓

Figure 3: Flipping Structure of SARLock and Anti-SAT.

In 2015, Subramanyan *et al.* [20] proposed a new and powerful attack using Boolean satisfiability (SAT) solver, called SAT attack, that effectively and quickly broke all previously proposed logic locking techniques. As an "*oracle-guided*" attack, SAT attack requires a reverse-engineered but locked netlist (C_L), and a functionally activated chip (C_O). A circuit view of steps taken in a SAT attack is shown in Fig. 2. For this attack, the attacker first replicate the obfuscated circuit and builds a double circuit which is used for finding an input (X_d[i]) that for two different key values generates two different outputs. Such input is referred to as Discriminating Input Pattern(DIP). Each X_d[i] is used to create a DI validation circuit (DIVC). The validation circuit, as shown in Fig. 2 assures that for a previously found X_d[i], two different keys generate the same output value. Each iteration of the SAT attack finds a new (X_d[i]), and add a new DI validation circuit. The DIVCs are ANDed together to form a Set of Correct Key Validation Circuit (SCKVC). In each iteration, the SAT solver try to find a new X_d[i] and two key values that satisfy the double circuit (KDC) and the Validation Circuit (SCKVC). The key values and the X_d[i], as illustrated in Alg. 1 (line 5), is found by a SAT query. This means the new key generate two different values for the new X_d[i], but generate the same value for all previously found X_ds for both key values. This process continues until the SAT solver cannot find a new X_d[i] (line 4). At this point any key that generates the correct output for the set of found X_ds is the correct key (line 9).

Algorithm 1 SAT-based Attack Algorithm [20]

```
1: function SAT_Attack(Circuit $C_L$, Circuit $C_O$)
2:     $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2)$;
3:     while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
4:         $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
5:         $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$; $i \leftarrow i+1$;
6:     $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

For all prior obfuscation schemes, even those resistant to sensitization attack, the SAT attack was able to rule out a significant number of key values at each iterations (by finding each DIP). Hence, In order to thwart SAT attack, the first attempted approach was to weaken the strength of the DIPs to reduce its pruning power. SARLock [15] and Anti-SAT [32] were the first prior-art methods that accomplished this. Both SARLock and Anti-SAT engaged one-point flipping function, demonstrated in Fig. 3. Using this obfuscation scheme, each DIP is able to rule out only one incorrect key. Hence, the SAT attack requires to apply all $2^{|KI|}$ to retrieve the correct functionality. However, this method results in obfuscation circuits that for all but one output work as the original circuit, and the output corruption upon application of a wrong key is quite low.

4 STAGE 3: POST-SAT ATTACKS

As discussed, the proposed SAT-resilient solutions suffered from low output corruption. This however could have been addressed by combining a SAT-hard solution with a traditional obfuscation solution, such as RLL or SLL, that exhibits high level of output corruption. Although SAT-resilient logic locking schemes provided a defense against SAT attack, researchers found new vulnerabilities associated with this class of obfuscation techniques resulting in the development of many new attacks on the presumed SAT-hard logic locking schemes described in this section.

4.1 Removal Attack

As shown in Fig 3, in bare implementation of one-point flipping circuit, the locking circuitry is completely decoupled from the original circuit. A removal attack identifies and removes/bypasses the locking circuitry to retrieve the original circuit and to remove dependence on key values [17]. The removal attack, presented in [17], was used to detect and remove SARLock [15]. In presence of removal attack, researchers investigated SAT-hard solutions that are hard to detect (preventing removal by pure structural analysis), an example of which was Anti-SAT [32].

4.2 Signal Probability Skew (SPS) Attack

The Signal Probability Skew (SPS) attack [18] leverages the structural traces in Anti-SAT block to identify and isolate the Anti-SAT block [32]. *Signal probability skew* (SPS) of a signal x is defined as $s = P_r[x = 1] - 0.5$, where $P_r[x = 1]$ indicates the probability that signal x is 1. The range of s is $[-0.5, 0.5]$. If the SPS of signal x is closer to zero, an attacker have lower chance of guessing the signal value by random. For a 2-input gate, the signal probability skew is the difference between the signal probability of its input wires. The flipping-circuit in the Anti-SAT is constructed using two complementary circuits, g and $\overline{g}$, in which the number of input vectors that make the function g equal to 1 (p) is either close to 1 or $2^n - 1$. These two complementary circuits converge at an AND gate G. Considering this structure, the *absolute difference of the signal probability skew* (ADS) of the inputs of gate G is quite large, noting that the SAT resilience is ensured by this skewed p. Algorithm 2 shows the SPS attack, which identifies the Anti-SAT block's output by computing signal probabilities and searching for the skew(s) of arriving signals to a gate in a given netlist.

Algorithm 2 SPS Attack Algorithm [18]

```
function SPS_ATTACK(Circuit $C_L$)
    $ADS_{arr} \leftarrow \{\}$;
    for each $gate \in C_L$ do
        $ADS_{arr}(gate_i) \leftarrow$ Compute_ADS($C_L$, $gate_i$);
    $G \leftarrow$ Find_Maximum($ADS_{arr}$);
    $Y \leftarrow$ Find_value_from_skew($G$);          ▷ Correct value of Anti_SAT output
    $C_{Lock} \leftarrow$ remove_TFI($C_L$, $G$, $Y$);          ▷ Transitive FanIn of the gate $G$
    return $C_{Lock}$          ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

4.3 Bypass Attack

Although SARLock and Anti-SAT break the SAT attack, their respective output corruptibility is very low if they are not mixed with traditional logic locking, such as SLL. Observing and relying on the very low level of output corruption in such SAT-hard solutions, the bypass attack [25] was introduced. The bypass attack instantiates two copies of the obfuscated netlist using two randomly selected keys, and build a miter circuit that evaluates to 1 only when the output of two circuits is different. The miter circuit is then fed to a SAT solver looking for such inputs. The SAT returns with minimum of two inputs for which the outputs are different. These input patterns are tested using an activated IC (golden IC) validating the correct output. Then a bypass circuit is constructed using a comparator that is stitched to the primary output of the netlist which is unlocked using the selected random key, to retrieve the correct functionality if that input pattern is applied. The Bypass attack works well when the SAT-hard solution is not mixed with traditional logic locking mechanism since its overhead increases very quickly as output corruption of logic locking increases. This observation motivated researchers to look at possibilities of approximate attacks to retrieve the key values associated to non SAT-hard obfuscation solutions that are mixed with SAT-hard solutions.

4.4 AppSAT Attack

So far, defences solution to mitigate the SAT attack, are based on the assumption that the attacker needs an exact attack on logic locking. However, Shamsi *et al.* [9] proposed a new attack (AppSAT), which relax this constraint. AppSAT shown in Algorithm 3, is an approximate attack on logic locking based on the SAT attack and random testing. The authors use *probably-approximate-correct* (PAC) model for formulating approximate learning problem. The exact SAT attack continues to find DIPs until no more DIPs can be found. However, the AppSAT will be terminated in any early step in which the error falls below the certain limit. If this condition happens, the key value will be considered as an approximate key with specified error rate; otherwise, the random sampling that resulted in a disagreement will be added to a SAT formula as a new constraint. In AppSAT, heuristic methods for estimating the error is used for large functions, to avoid any computation complexity.

4.5 Double-DIP Attack

Double-DIP [30] is another approximate attack, shown in Algorithm 4. Double-DIP is an extension of SAT attack in which during each iteration, the discriminating input should eliminate at least two wrong keys. To illustrate its effectiveness, researchers used double-DIP to target SARLock+SSL, representing a compound of SAT-hard and high output corruption obfuscation. When the double-DIP attack terminates, the key of the traditional logic locking (SSL) is guaranteed to be correct. As a result, the compound logic locking will be reduced to a single SAT attack resilient technique, that could then be attacked using bypass attack.

Algorithm 3 AppSAT Attack Algorithm [9]

```
function APPSAT_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2)$;
    while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$; $i \leftarrow i+1$;
        every $n$ rounds do
        for each ($x \in$ Random Patterns) do
            if $C_L(X, K_1, Y) \neq C_O(X)$ then
                $FailedPatterns \leftarrow FailedPatterns + 1$;
                $F_{i+1} \leftarrow F_{i+1} \wedge (C_L(X, K_1, Y) = C_O(X))$; $i \leftarrow i+1$;
        if error < ErrorThreshold then
            return $K_1$ as an approximate key
    $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

Algorithm 4 Double-DIP Attack Algorithm [30]

```
function DOUBLEDIP_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2) \wedge C_L(X, K_3, Y_1) \wedge C_L(X, K_4, Y_2)$ ;
    while $SAT(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$;
        $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \bigwedge_{j=1}^{4} C_L(X_d[i], K_j, Y_d[i])$; $i \leftarrow i+1$;
    $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

4.6 Bit-Flipping Attack

The Bit-flipping attack [26] is yet another attack against compound logic locking schemes in which a SAT-hard solution such as SARLock is combined with a traditional logic locking to guarantee both high error rates and resilience to the SAT-based attack. In Bit-flipping attack, the keys are first separated into two groups (k_1 and k_2) by counting DIPs for two keys with hamming distance equal to one. The attack is motivated from the observation that in traditional logic locking, wrong key causes substantial wrong input-output pattern. However, the error rate of bit-flipping function is usually very small. As shown in Algorithm 5, after separation of keys, this attack fixes SAT-resilient keys, k_2, as a random number, and uses a SAT solver to find the correct key values for k_1. After finding k_1, the bypass attack is applied to retrieve the original circuit.

Algorithm 5 Bit-flipping Attack Algorithm [26]

```
function BitFlipping_Attack(Circuit $C_L$, Circuit $C_O$)
    for each $j$ < Fixed-iteration do
        $K_A \leftarrow$ a random key;
        for each bit b $\in K_A$ do
            $K_B \leftarrow K_A$ while bit b flipped;
            $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_A, Y_A) \wedge C_L(X, K_B, Y_B)$;
            while $SAT(F_i \wedge (Y_A \neq Y_B))$ do
                $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_A \neq Y_B))$;
                $F_{i+1} \leftarrow F_i \wedge (X \neq X_d[i])$; $i \leftarrow i+1$;
                if i > Threshold then
                    b is in $K_1$,
                    break;
        $j \leftarrow j + 1$;
    $K_2 \leftarrow$ all key bits / $K_1$;        ▷ Seperation is Done. Then, fix $K_2$ as a random number.
    $K_1 \leftarrow$ SAT_ATTACK $(C_L, C_O)$;                    ▷ Find Traditional Keys using SAT.
    $C^*_L \leftarrow$ update_netlist$(C_L \mid K_1)$
    return (BYPASS_ATTACK$(C^*_L)$);
```

4.7 AppSAT Guided Removal Attack

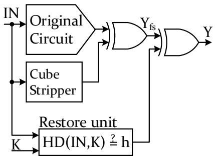
AppSAT Guided Removal (AGR) attack targets compound logic locking, particularly Anti-SAT + traditional logic locking [17]. This attack integrates AppSAT with a simple structural analysis of the locked netlist (a post-processing steps). Unlike AppSAT, the AGR attack recovers the correct key. In this attack, first the AppSAT is used to find the key of the traditional obfuscation scheme (used as a part of compound lock). Then, AGR targets the remaining key bits belong to the SAT-resilient logic locking, such as Anti-SAT block, through a simple structural analysis. As shown in Algorithm 6, in the post-processing steps, AGR finds the gate (G) at which most of the Anti-SAT key bits converge. AGR finds G by tracing the transitive fanout of the Anti-SAT key inputs, where all the Anti-SAT key bits converge. The ratio of key bits converging at each of the inputs of the gate G, are close to 0.5, which is shown as the *selected property* in line 7 of Algorithm 6. AGR identifies the candidates for gate G by checking this property for all gates in the circuit, and then sort these candidate based on the number of key inputs that converge at a gate and pick the gate G from all candidates, which has the most number of key inputs converge to that gate. Then the attacker re-synthesize the design with the constant value for the output of G gate and retrieving the correct functionality.

Algorithm 6 AGR Attack Algorithm [17]

```
 1: function AGR_Attack(Circuit $C_L$, Circuit $C_O$)
 2:     #Cand $\leftarrow$ num_gates$(C_L)$
 3:     while (#Cand > 1 and !Timeout) do
 4:         AppSAT_Attack();                                          ▷ 4 times
 5:         Candidates $\leftarrow \{\}$;
 6:         for each gate $\in C_L$ do
 7:             if $gate_i$ has the selected property then
 8:                 Candidates $\leftarrow$ Candidates + 1;
 9:     $G \leftarrow$ Find_Max_key_count(Candidates);
10:     $C_{Lock} \leftarrow$ remove_TFI$(C_L, G)$;     ▷ remove Transitive FanIn of the gate $G$
11:     return $C_{Lock}$;                  ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

4.8 Sensitization Guided SAT Attack

While the one-point flipping circuit in Anti-SAT and SARLock are completely decoupled from the original netlist, Li *et al.* [13] proposed the AND-tree Insertion (ATI), as a SAT-resilient logic locking, which embeds AND trees inside the original netlist. It not only makes all aforementioned attack less effective, it also decreases the implementation overhead. Additionally, the input of AND-tree are camouflaged by inserting INV/BUF camouflaged gates, which can be replaced with the XOR/XNOR gates in order to lock the AND-tree. However, this defense was broken by a new attack that was coined as Sensitization Guided SAT (SGS) attack [17]. The SGS attack is carried out in two stages: (1) *sensitization* that exploits bias in input patterns which allows an attacker to apply only a subset of DIPs, i.e., those that bring unique values to the AND-tree inputs. (2) *SAT attack* using the patterns discovered in the first stage.



Y_{fs}	IN	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
✓	0	✓	✓	✓	✓	✗	✓	✓	✓
✓	1	✗	✓	✓	✓	✓	✓	✓	✓
✓	2	✓	✓	✗	✓	✓	✓	✓	✓
✓	3	✓	✓	✓	✓	✓	✗	✓	✓
✗	4	✗	✓	✗	✗	✗	✗	✗	✗
✓	5	✓	✓	✓	✓	✓	✓	✗	✓
✓	6	✓	✓	✓	✓	✓	✓	✓	✗
✓	7	✓	✓	✓	✗	✓	✓	✓	✓

Figure 4: SFLL-HD while $h = 0$.

4.9 Functional Analysis Attack

Aiming to provide a defense that resists all previously formulated attacks led to the introduction of Stripped-Functionality Logic Locking (SFLL) [16]. In SFLL the original circuit is modified for at-least one input pattern (cube) using a *cube stripping unit*, demonstrated in Fig. 4. As shown, Y_{fs} is the output of the stripped circuit, in which the output corresponding to at-least one input pattern is flipped. The restore unit not only generates the flip signal for one input pattern per each wrong key, it also restores the stripped output, (e.g. $IN = 4$ in Fig. 4) to recover the correct functionality on Y. Note that applying removal attack on restore unit recovers Y_{fs}, which is not the correct functionality. In addition, SFLL-HD is able to protect $\binom{k}{h}$ input patterns that are of Hamming Distance (HD) h from the k-bit secret key, and accordingly uses Hamming Distance checker as a restore unit (e.g. $h = 0$ in Fig. 4 is also called TTLock [19]).

Although SFLL was resilient against all previously formulated attacks, it was exploited using a newly formulated attack, called Functional Analysis on Logic Locking (FALL) attack [31]. In this attack model, the adversary is assumed to be a malicious foundry that knows the locking algorithm and its parameters, e.g. h in SFLL-HD. A FALL attack is carried out in three main stages and relies on structural and functional analyses to determine potential key values of a locked circuit. First, FALL attack tries to find all nodes which are the results of comparing an input value with a key input. It is done by a comparator identification. Such nodes ($nodes_{RU}$), which contains these particular comparators, are very likely to be part of the functionality restoration unit. The set of all inputs that appear in these comparators, should be in the fan-in cone of the cube stripping unit. Then, it finds a set of all gates whose fan-in-cone is identical to the members of $nodes_{RU}$. This set of gates must contain the output of the cube stripping unit. Second, the attacker applies functional analysis on the candidate nodes suggested by and collected from the first stage to identify suspected key values. Broadly speaking, the attacker uses functional properties of the cube stripping function used in SFLL, to determines the values of the keys. Based on the author's view, this function has three specific properties. So, they have proposed three attacks algorithms on SFLL, which exploit unateness and Hamming distance properties of the cube stripping functions. The input of these algorithm is circuit node c, that computed from the first stage, and the algorithm checks if c behaves as a Hamming distance calculator in the cube stripping unit of SFLL-HD. If the attack is successful, the return value is the protected cube. Third, they have proposed a SAT-based key confirmation algorithm using a list of suspected key values and I/O oracle access, that verifies whether one of the suspected key values computed from the second stage, is correct.

4.10 CycSAT Attack

Considering the strength of all previously formulated attacks, some of the researchers started seeking solutions that fundamentally violated the assumptions of these attacks with respect to the nature of locked circuits. One of such attempts was the introduction of cyclic logic locking [10][23], was first proposed in [10]. In this obfuscation technique each deliberately established cycle is designed to have more than one way to open. The requirement for having more than one way to open each cycle assures that even if the original netlist has no cycle by itself, the cycles remains irreducible by means of structural analysis. The cyclic obfuscation resulted in an obfuscation with high level of output corruption, while it was able to break the SAT attack either by 1) trapping it in an infinite loop, or 2) forcing it to exit with a wrong key depending on weather the introduced cycles make the circuit stateful or oscillating.

The promise of secure cyclic obfuscation was shortly after broken by CycSAT attack [5]. In CycSAT, the key combinations that result in formation of cycles are found in a pre-processing step. These conditions are then translated into problem augmenting CNF formulas, denoted as cycle avoidance clauses, satisfaction of which guarantee no cycle in the netlist. The cycle avoidance clauses are then added to the original SAT circuit CNF and the SAT attack is executed. The validity of this attack, however, was challenged in [23], as researchers illustrated that the pre-processing time for CycSAT attack is linearly dependent on the number of cycles in the netlist. Hence, by building an exponential relation between the number of feedback, and the number of cycles in the design, the pre-processing step of CycSAT will face exponential runtime.

4.11 Behavioral SAT (BeSAT) Attack

Inability to analyze all cycles in the prepossessing step of CycSAT results in missing cycles in the pre-processing step of CycSAT, leading to building a statefull or oscillating circuit, trapping the SAT stage of the CycSAT attack. BeSAT [27] remedies this shortcoming by augmenting the CycSAT attack with a run-time behavioral analysis. As shown in Algorithm 7, by performing behavioral analysis at each SAT iteration, BeSAT detects repeated DIPs when the SAT is trapped in an infinite loop. Also, when SAT cannot find any new DIP, a ternary-based SAT is used to verify the returned key as a correct one, preventing the SAT from exiting with an invalid key.

Algorithm 7 BeSAT Attack on Cyclic Locked Circuits [27]

```
function BeSAT_Attack(Circuit $C_L$, Circuit $C_O$)
    $W = (w_0, w_1, \dots w_m) \leftarrow$ FindFeedback($C_L$);
    for each ($w_i \in W$) do
        $F(w_i, w_i') \leftarrow$ no_ structural_ path($w_i$);
    $i \leftarrow 0$; $NC(K) = \wedge_{i=0}^{m} F(w_i, w_i')$
    $C_L^*(X, K, Y) \leftarrow C_L(X, K, Y) \wedge NC(K)$; $F_0 \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
    while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$;
        if ($X_d[i]$ in DIP) and ($C_L(X_d[i], K_1) \neq Y_d[i]$)) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_1) \wedge (K_2 \neq \hat{K}_1)$;
        else if ($X_d[i]$ in DIP) and ($C_L(X_d[i], K_2) \neq Y_d[i]$) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_2) \wedge (K_2 \neq \hat{K}_2)$;
        $i \leftarrow i+1$;
    while $SAT_{K_1}(F_i)$ do                                        ▷ Correct Key: $\hat{K}_c$
        if Ternary_SAT($F_i$, $K_c$) then
            $F_i \leftarrow F_i \wedge (K_1 \neq \hat{K}_c)$
        else
            $K^* \leftarrow \hat{K}_c$; break;
```

5 STAGE 4: SMT ATTACK

As discussed previously, many of the attacks proposed at post-SAT attack stage were formulated by adding a pre-processing step to the original SAT attack, and/or extending the SAT attack to co-process and check additional features in each iteration. In other terms, to break many of the post-SAT era obfuscation techniques, attackers relied on compound attacks by combining SAT solvers by pre-processors (e.g. in CycSAT) and co-processors (e.g. in BeSAT) to extend its modeling reach. Motivated by this trend, the need for having pre- co- and post- processors along with a SAT solver in an attack was realized and addressed in [12] and a new and extremely powerful attack, coined as Satisfiability Module Theory (SMT) attack was introduced. The strength of SMT attack, as the superset of SAT attack, comes from its ability to combine SAT and Theory solvers. The SMT attack could be invoked with any number and combination of theory solvers, and a SAT solver, which allow the attacker to express constraints that are difficult or even impossible to express using CNF, including timing, delay, power, arithmetic, graph and many other first-order theories in general. To showcase the modeling capability of SMT attack, the authors used the SMT attack 1) to break a new breed of obfuscation that relied on locking the delay information in netlist (by generating setup and hold violations), 2) to formulate an accelerated attack (to reduce the attack time) with means of approximate exit (if trapped with SAT hard solutions).

In pursuit of obfuscation schemes that could not be attacked by SAT motivated attackers, some researchers tried to extend the locking mechanism to aspects of a circuit's function that cannot be translated to CNF. For example, Xie *et al.* proposed a timing obfuscation scheme, denoted as delay logic locking (DLL), in [28]. The Goal of DLL obfuscation scheme is introducing setup and hold violation if the correct key is not applied. In this case, the obfuscation attempts to change both logical and behavioral (timing) properties. A functionally-correct but timing-incorrect key will result in timing violations, leading to circuit malfunctions. Considering that timing is not translatable to CNF, the SAT solver remains oblivious to the keys used for timing obfuscation. Authors in [12], however, illustrated that the SMT attack could easily deploy a graph theory solver, provide timing constraints to the theory solver (in terms of required min and max delay to meet the hold and setup time), and use the theory solver in parallel with the internal SAT solver to break both logic and delay obfuscation. They additionally show that the theory solver could be initiated as a pre-processor (Eager SMT approach) or as a co-processor (Lazy SMT approach) to break the same problem, showcasing the strength of SMT attack. The *lazy* mode of this attack is illustrated in Algorithm 8. Although at about the same time Chakraborty proposed TimingSAT to attack the DLL [2], similar to many prior SAT-based attack, it was by deploying a pre-processor for analysis of graph timing, and generating helper clauses for the subsequent call to the SAT attack.

Algorithm 8 SMT Attack on DLL (Lazy Approach) [12]

```
function SMTLazy_Attack(Circuit $C_L$, Circuit $C_O$)
    $C_L^* \leftarrow$ toBOOLEAN($C_L$);                              ▷ Replace TDK with Buffer
    $i \leftarrow 0$; $F \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
    $G_L^* \leftarrow$ toGRAPH($C_L$);                        ▷ Wires = Edges, Gates = Vertices
    $F_T \leftarrow$ GenTCE($G_L^*$)                                ▷ Theory Learned Clauses
    $F_{SMT} \leftarrow F \wedge F_T$;                                          ▷ SMT Clauses
    while $SMT(F_{SMT})$ do                                     ▷ $X_d[i]$, $K_1$, $K_2$, CC
        $Y_d[i] \leftarrow C_O(X_d[i])$; $F \leftarrow F \wedge C_L^*(X_d[i], K_1, Y_d[i]) \wedge C_L^*(X_d[i], K_2, Y_d[i])$;
        $F_{SMT} \leftarrow F \wedge CC$; $i \leftarrow i+1$;
    $K^* \leftarrow$ smt_assignment$_{K_1}(F_{SMT})$;

    function GenTCE(Graph $G_L^*$)
        $Inputs \leftarrow$ find_start_points($G_L^*$); $Outputs \leftarrow$ find_end_points($G_L^*$); $T_{CE}(K) \leftarrow []$;
        for each ($(Sp, Ep) \in (Inputs, outputs)$) do
            Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));        ▷ Hold Violation
            Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);             ▷ Setup Violation
            $Range(Sp,Ep)(K) \leftarrow$ Lower(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K);
            $T_{CE}(K) \leftarrow T_{CE}(K) \cup Range(Sp,Ep)(K)$;
        return $T_{CE}(K)$
```

The ability of SMT solver to instantiate and integrate different theory solver makes it a suitable attack platform for modeling and formulating very strong attacks. As an example of the strength of SMT attack, the authors in [12] formulated and presented an accelerated SMT attack with ability of detecting the presence of SAT-hard obfuscation and switching to an accelerated approximate attack. As shown in Algorithm 9, this was done by invoking a *BitVector* theory solver to constrain the SMT solver for finding keys that result in highest output corruption first. This could be done by constraining the required HD between the output of double circuit when two different keys for the same discriminating input is being tested. The required HD starts from a large value, and every time that the SMT solver return UNSAT, the constraint is relaxed until HD of 1 is reached. This leads to the guaranteed discovery of keys for traditional logic locking first. After N tries (*Rep* in Algorithm 9) for HD of 1, the SMT attack exits, notes that there exist a SAT-hard obfuscation, which now could be addressed by the Bypass attack.

Algorithm 9 Accelerated SMT Attack on Compound Locking [12]

```
function AccSMT_Attack(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $HD_h \leftarrow sizeof(\text{output})$; $HD_l \leftarrow HD_h$ - 1;
    $TimeOut \leftarrow 20$; $Rep \leftarrow 20$; $HD_R \leftarrow 1$; $R_{cnt} \leftarrow 0$;
    $C_L^* \leftarrow$ toBOOLEAN($C_L$);                         ▷ Everything is Boolean.
    $F \leftarrow C_L^*(X, K_1, Y_1) \land C_L^*(X, K_2, Y_2)$;
    $BV_L^* \leftarrow$ toBITVECTOR($C_L$);                    ▷ Define BITVECTOR on output.
    $BVs_L^*(X, K_1, K_2) \leftarrow$ ONEs($BV_L^*(X, K_1) \oplus BV_L^*(X, K_2)$);
    $F_T \leftarrow (BVs_L^*(X, K_1, K_2) \geqslant HD_l) \land (BVs_L^*(X, K_1, K_2) \leqslant HD_h)$;
    $F_{SMT} \leftarrow F \land F_T$;                                  ▷ SMT Clauses
    while $HD_l \geqslant 1$ do
        while $SMT(F_{SMT} \mid TimeOut)$ do                     ▷ $X_d[i], K_1, K_2$, CC
            $Y_d[i] \leftarrow C_O(X_d[i])$;
            $F \leftarrow F \land C_L^*(X_d[i], K_1, Y_d[i]) \land C_L^*(X_d[i], K_2, Y_d[i])$; $F_{SMT} \leftarrow F \land CC$;
            if $HD_l \leqslant HD_R$ then
                if $R_{cnt} == Rep$ then
                    break;
                $R_{cnt}$++;
        $HD_l$--;
    $K^* \leftarrow$ smt_assignment$_{K_1}(F_{SMT})$;
```

6 DISCUSSION & OPPORTUNITIES

Table 2 compares the effectiveness of the attacks discussed in this paper against most notable obfuscation schemes. As illustrated the combination of FALL, Bypass and SMT attack can break all existing solutions, pointing us to a need for a new direction for generating non-bypassable SMT hard obfuscation solutions. The dilemma is that SAT-hard solutions have extremely low output corruption, and are prone to Bypass, FALL, Removal and SPS attack. On the other hand, the traditional logic locking schemes have high output corruption, but could be easily broken with SAT/SMT attack. The compound logic locking solutions that combine the SAT-hard solutions for resistance against SAT and SMT attack, and traditional logic locking for resistance against Bypass, FALL, Removal and SPS attack are also prone to approximate SAT and SMT attacks. What is really desired, is a SMT-hard logic locking scheme with high degree of output corruption. As a step in this direction, few very recent research papers have focused on increasing the execution time of each SAT/SMT iteration rather than the total execution time [4, 11]. The Full-Lock in [4] is argued that the strength of SAT/SMT solvers come from their *Conflict-Driven Clause Learning* (CDCL) ability, which is resulted by recursively calling *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm. Hence, the Full-Lock creates an obfuscation method that results in very deep recursive call trees. They argue that the SAT/SMT attack execution time can be expresses by formula 1, in which N denotes the number of iterations (DIPs) of the SAT/SMT attack, $T_{DPLL}(\Phi)$ is the execution time of recursive calls for DPLL algorithm on CNF Φ, and t is the execution time of the remaining book keeping code executed at each iteration.

$$T_{Attack} = \sum_{i=1}^{N} T(i) = \sum_{i=1}^{N} (t + T_{DPLL}(\Phi)) = \sum_{i=1}^{N} \sum_{j=1}^{M} (T_{DPLL}^{Avg}) \simeq MN \times T_{DPLL}^{Avg} \quad (1)$$

Authors argue that M in formula 1 denotes the number of recursive DPLL calls. Accordingly, the execution time of SAT attack could also become unfeasible by building an exponential relation between the percentage gate inserted (area overhead) and M. The strong aspect of this alternative solution is that (1) the problems posed at each iteration of SAT/SMT attack is a SAT-hard problem, (2) the output corruption of this methods is significantly higher than obfuscating solution relying on increasing the N, (3) it is not susceptible to SPS, removal, bypass, approximate attack, to name a few. The hardness of SAT/SMT attack in the solution posed by Full-Lock cannot be assessed/formulated similar to that of SFLL. Moving towards this new direction for generating SAT-hard problems with high level of output corruption can be generalized more, where an obfuscation solution in this direction can engineer the number of recursive calls, pushing the number of recursive call to be an exponential function of added gates counts (area overhead).

Table 2: Comparison of proposed attacks/defenses.

Defenses Attacks	RLL [6]	FLL [8]	SLL [7]	Anti-SAT [32]	SARLock [15]	Compound [15]	SFLL [16]	Cyclic [10]	SRCLock [23]	DLL [28]
Brute Force	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Sensitization[7]	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Hill-Climbing[29]	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
SAT[20]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
SPS+Removal[17][18]	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
Bypass[25]	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
AppSAT[9]	✓	✓	✓	✗	✗	**P**	✗	✗	✗	✗
Double-DIP[30]	✓	✓	✓	✗	✗	**P**	✗	✗	✗	✗
Bit-Flipping[26]	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
AGR[17]	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
FALL[31]	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
CycSAT[5]	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
BeSAT[27]	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
TimingSAT[2]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓
SMT[12]	✓	✓	✓	✗	✗	**P**	✗	✓	✓	✓

✓: Attack Success, ✗: Fail to Attack, **P**: Only removes the key to the traditional locking in Compound Defense.

REFERENCES

[1] A. Baumgarten *et al.* 2010. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers* 27, 1 (2010), 66–75.

[2] A. Chakraborty *et al.* 2018. TimingSAT: timing profile embedded SAT attack. In *ICCAD*. 6.

[3] H. M. Kamali *et al.* 2018. LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection. In *ISVLSI*. 405–410.

[4] H. M. Kamali *et al.* 2019. Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks. In *DAC*. 6.

[5] H. Zhou *et al.* 2017. CycSAT: SAT-based attack on cyclic logic encryptions. In *ICCAD*. 49–56.

[6] J. Roy *et al.* 2010. Ending piracy of integrated circuits. *Computer* 43, 10 (2010), 30–38.

[7] J. Rajendran *et al.* 2012. Security analysis of logic obfuscation. In *DAC*. 83–89.

[8] J. Rajendran *et al.* 2015. Fault analysis-based logic encryption. *IEEE Trans. on Comp.* 64, 2 (2015), 410–424.

[9] K. Shamsi *et al.* 2017. AppSAT: Approximately deobfuscating integrated circuits. In *HOST*. 95–100.

[10] K. Shamsi *et al.* 2017. Cyclic obfuscation for creating sat-unresolvable circuits. In *GLSVLSI*. 173–178.

[11] K. Shamsi *et al.* 2018. Cross-Lock: Dense Layout-Level Interconnect Locking using Cross-bar Architectures. In *GLSVLSI*. 147–152.

[12] K. Z. Azar *et al.* 2019. SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks. *IACR Trans. on CHES (TCHES)* (2019), 97–122.

[13] M. Li *et al.* 2017. Provably secure camouflaging strategy for IC protection. *IEEE Trans. on CAD (TCAD)* (2017).

[14] M. Rostami *et al.* 2014. A primer on hardware security: Models, methods, and metrics. *Proc. IEEE* 102, 8 (2014), 1283–1295.

[15] M. Yasin *et al.* 2016. SARLock: SAT Attack Resistant Logic Locking. In *HOST*. 236–241.

[16] M. Yasin *et. al.* 2017. Provably-secure logic locking: From theory to practice. In *ACM-CCS*. 1601–1618.

[17] M. Yasin *et al.* 2017. Removal attacks on logic locking and camouflaging techniques. *IEEE Trans. on Emerging Topics in Computing* 1 (2017), 1–1.

[18] M. Yasin *et al.* 2017. Security analysis of anti-sat. In *ASP-DAC*. 342–347.

[19] M. Yasin *et al.* 2017. What to lock?: Functional and parametric locking. In *GLSVLSI*. 351–356.

[20] P. Subramanyan *et al.* 2015. Evaluating the security of logic encryption algorithms. In *HOST*. 137–143.

[21] P. Tuyls *et al.* 2006. Read-proof hardware from protective coatings. In *CHES*. 369–383.

[22] S. Roshanisefat *et al.* 2018. Benchmarking the capabilities and limitations of SAT solvers in defeating obfuscation schemes. In *IOLTS*. 275–280.

[23] S. Roshanisefat *et al.* 2018. SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware. In *GLSVLSI*. 153–158.

[24] T. Winograd *et al.* 2016. Hybrid STT-CMOS designs for reverse-engineering prevention. In *DAC*. 6.

[25] X. Xu *et al.* 2017. Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks. In *CHES*. 189–210.

[26] Y. Shen *et al.* 2018. SAT-based bit-flipping attack on logic encryptions. In *DATE*. 629–632.

[27] Y. Shen *et al.* 2019. BeSAT: behavioral SAT-based attack on cyclic logic encryption. In *ASP-DAC*. ACM, 657–662.

[28] Y. Xie *et al.* 2017. Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction. In *DAC*. 9.

[29] S. M. Plaza and I. L. Markov. 2015. Solving the third-shift problem in IC piracy with test-aware logic locking. *IEEE Trans. on CAD (TCAD)* 34, 6 (2015), 961–971.

[30] Y. Shen and H. Zhou. 2017. Double dip: Re-evaluating security of logic encryption algorithms. In *GLSVLSI*. 179–184.

[31] D. Sirone and P. Subramanyan. 2018. Functional Analysis Attacks on Logic Locking. *arXiv preprint arXiv:1811.12088* (2018).

[32] Y. Xie and A. Srivastava. 2016. Mitigating sat attack on logic locking. In *CHES*. 127–146.

[33] A. yeh. 2012. Trends in the global IC design service market. *DIGITIMES research* (2012).