TGA: An Oracle-less and Topology-Guided Attack on Logic Locking

Yuqiao Zhang*
yuqiao.zhang@auburn.edu
Auburn University
Auburn, Alabama, USA

Pinchen Cui†
pinchen@auburn.edu
Auburn University
Auburn, Alabama, USA

Ziqi Zhou*
ziqi.zhou@auburn.edu
Auburn University
Auburn, Alabama, USA

Ujjwal Guin*
ujjwal.guin@auburn.edu
Auburn University
Auburn, Alabama, USA

ABSTRACT

Due to the outsourcing of semiconductor design and manufacturing, a number of threats have emerged in recent years, and they are overproduction of integrated circuits (ICs), illegal sale of defective ICs, and piracy of intellectual properties (IPs). Logic locking is one method to enable trust in this complex IC design and manufacturing processes, where a design is obfuscated by inserting a lock to modify the underlying functionality so that an adversary cannot make a chip to function properly. A locked chip will only work properly once it is activated by programming with a secret key into its tamper-proof memory. Over the years, researchers have proposed different locking mechanisms primarily to prevent Boolean satisfiability (SAT)-based attacks, and successfully preserve the security of a locked design. However, an untrusted foundry, the adversary, can use many other effective means to find out the secret key. In this paper, we present a novel oracle-less and topology-guided attack denoted as *TGA*. The attack relies on identifying repeated functions for determining the value of a key bit. The proposed attack does not require any data from an unlocked chip, and eliminates the need for an oracle. The attack is based on self-referencing, i.e., it compares the internal netlist to find the key. The proposed graph search algorithm efficiently finds a duplicate function of the locked part of the circuit. Our proposed attack correctly estimate a key bit very efficiently, and it only takes few seconds to determine the key bit. We also present a solution to thwart *TGA* and make logic locking secure.

CCS CONCEPTS

• **Security and privacy → Hardware attacks and countermeasures**;

KEYWORDS

Logic locking, Boolean functions, overproduction, directed graph, depth-first search.

ACM Reference Format:
Yuqiao Zhang, Pinchen Cui, Ziqi Zhou, and Ujjwal Guin. 2019. TGA: An Oracle-less and Topology-Guided Attack on Logic Locking. In *3rd Attacks and Solutions in Hardware Security Workshop (ASHES'19), November 15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3338508.3359576

1 INTRODUCTION

The globalization of the design and manufacturing of integrated circuit (IC) shifts the semiconductor industry to adopt horizontal integration from the vertical one, where a system on a chip (SoC) designer acquires intellectual properties (IPs) from many different IP vendors and sends their design to an offshore fabrication unit (e.g., foundry or fab) for manufacturing. As a result of this globalization, different threats have emerged in recent years and they are -(*i*) overproduction of ICs [3, 7, 13, 16, 31] – an untrusted foundry produces more chips and sells them in the open market without the consent of the SoC designers, (*ii*) sourcing of defective (e.g., out-of-specification or rejected) ICs in the market [13, 14, 27], and (*iii*) piracy of IPs [6, 37, 38] – an entity in the supply chain illegally obtains a functional IP. It can either use it, or sell it to a different entity in the supply chain.

Over the years, researchers proposed different technologies to prevent these aforementioned attacks. These solutions can be broadly categorized into IC metering [2, 3, 20, 31], logic locking [13, 28, 31], hardware watermarking [9, 18, 26], and split manufacturing [17, 40]. However, logic locking gains popularity in recent years to address these attacks. In logic locking, the original design of a circuit is transformed to a different one. The primary objective of this technique is to obfuscate the inner details of the circuit so that an adversary cannot reconstruct the original netlist. The original functionality can only be reversed, when a secret key is programmed into the chip. Different logic locking techniques (see Figure 1) have been proposed over the years, and they are -(*i*) XOR-based, where a set of XOR/XNOR gates are inserted to change the functionality [13–15, 28, 31], (*ii*) MUX-based [21, 24, 29], (*iii*) LUT-based [4, 19, 23],

ASHES'19, November 15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6839-1/19/11…$15.00
https://doi.org/10.1145/3338508.3359576

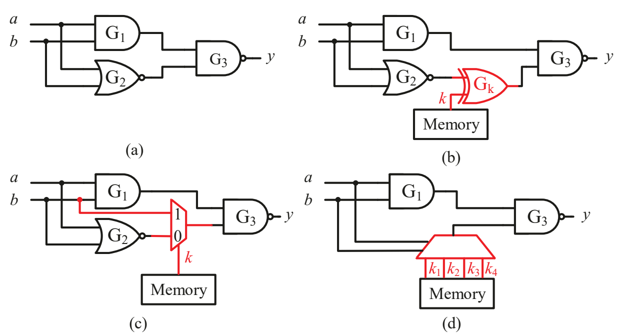

Figure 1: Logic locking methods: (a) An original netlist (b) XOR/XNOR-based logic locking (c) MUX-based logic locking (d) LUT-based logic locking.

and (*iv*) state space based [8]. However, XOR-based logic locking becomes popular due to its simplicity.

Boolean Satisfiability (SAT)-based attack, proposed in [35], has shown that logic locking techniques can be broken very effectively. Since then, many different solutions [13–15, 41, 43, 46] have been proposed to prevent SAT-based attacks. Unfortunately they have been broken subsequently [22, 32–34, 42, 44, 45]. Even though the wide popularity of SAT attacks among the research community, these attacks may have practical limitations for determining the secret key of a locked netlist. As the attacks require an oracle to discard equivalent class of incorrect keys, an adversary must possess a working chip to launch the attack. An untrusted foundry cannot unlock the netlist, when it receives the layout information (GDSII or OASIS files [30]) from the SoC designer. It needs to wait for an unlocked chip, an oracle, to be available in the market. This can be challenging as many of the chips used in critical or DoD applications are highly unlikely to be circulated (unless it is a commercial-off-the-shelf, COTS part) in the market. Second, it is yet to be demonstrated that SAT-attacks can be launched in industrial designs. The attack even fails to estimate the correct key for a small benchmark circuit (e.g., *c6288*, see the details in [35]).

As logic locking was proposed to address the threats from the untrusted manufacturing, where the adversary is an untrusted foundry, it can use many other effective techniques rather than to use SAT to break logic locking. In this paper, we proposed a novel attack on logic locked circuits to determine the key without having oracle access to the chips. ***Can we determine the secret key simply by analyzing the circuit topology?*** Contrary to the SAT-based attacks, we present a novel attack that does not require an oracle to break logic locking. We denote our proposed attack as ***TGA***: An Oracle-less and ***T***opology-***G***uided ***A***ttack on logic locked circuits. By using our proposed *TGA*, the secret key can be estimated efficiently even for the circuits that SAT attack fails (see in Section 5 for *c6288* circuit). In addition, an untrusted foundry can unlock any netlists using our proposed *TGA* without waiting for a working chip available in the market. The contributions of this paper are as follows:

- We present a novel attack, *TGA*, which is based on function search. The basic functions in a logic cone are generally repeated multiple times in a circuit. In this paper, we denote these functions as unit functions (*UFs*). If a key gate is placed in an instance of repeated *UFs* during the locking of a circuit, the original netlist can be recovered by searching the equivalent unit functions (*EUFs*), which are constructed with all hypothesis key values. As the *UFs* are constructed in few layers of gates (see Section 3.3 for details), the number of key gates to be placed in them is limited, which limits the *EUF* search combinations. Simulation results (see Section 5) show that we can determine majority of the key bits using our proposed *TGA*. Note that no oracle (unlocked chip) is required to launch our proposed *TGA*.
- We develop an efficient algorithm that uses Depth-First-Search (*DFS*) for finding the equivalent unit functions in the locked netlist under attack. An adversary first constructs a directed graph [36] from the netlist to launch the attack. Note that each gate can be represented as a vertex, and each wire can be modelled as an edge in the graph that represents the netlist. In this paper, we demonstrate and implement a DFS-based *UF* search algorithm to determine the correct value of a secret key. The average time to determine a secret key bit is in the order of seconds. As a result, a locked circuit can be broken in few minutes, when they are locked with few hundred/thousand key gates.
- We also present a solution to prevent this proposed *TGA*. As an adversary performs *EUF* search in the netlist for self-referencing, *TGA* can be prevented if the search produces contradictory (or no) results with different hypothesis keys. *TGA* resistivity can be achieved if we lock all the repeated instances of an *UF*. The same DFS-based search algorithm can be used to identify all repeated instances of an unit function. If a key gate is placed in a repeated *UF*, it is necessary to lock all of such *UFs* so that an adversary cannot reach to a decision about the actual value of the key bit by comparing with its unlocked version. The key gates can also be placed in those unique *UFs* that are not repeated in the circuit.

The rest of the paper is organized as follows. In Section 2, the details of XOR-based logic locking technique is presented. Section 3 introduces our proposed *TGA*. The countermeasure is presented in Section 4. The simulation results are described in Section 5. In Section 6, we describe our future work and finally conclude the paper in Section 7.

2 XOR-BASED LOGIC LOCKING

XOR-based logic locking is a popular locking technique due to its simplicity. When an XOR gate is inserted to obfuscate the inner details of a circuit, the overall functionality remains the same when the correct key is programmed into the chip, and alters for some input patterns when a wrong key is applied.

Figure 2 shows an example to lock a circuit, which has three inputs (X_1, X_2 and X_3) and one output (Y). Let us assume that the circuit is locked using one key gate with key input k. Figure 2.(a) shows the original circuit. There can be two possible key values, $k = 0$ and $k = 1$. For $k = 0$, an XOR gate can directly be placed

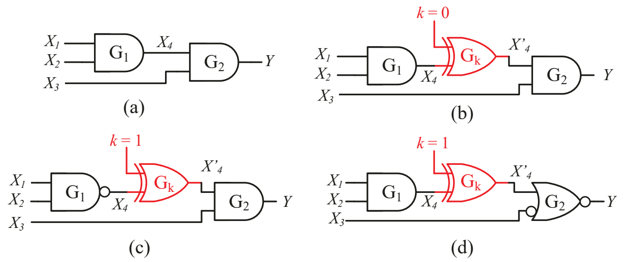

Figure 2: Logic locking using Exclusive OR (XOR) gates. (a) Original netlist. (b) Locked netlist when $k = 0$. (c) *Case-I*: Locked netlist when $k = 1$. (d) *Case-II*: Locked netlist when $k = 1$ (using DeMorgan's Theorem).

at node X_4, which is shown in Figure 2.(b). However, for $k = 1$, two possible scenarios may occur. One can invert the previous stage functionality, which is shown in Figure 2.(c). It is also possible to modify successive stage function using DeMorgan's Theorem, shown in Figure 2.(d).

In this example, the original function of the circuit $Y = X_3 \cdot X_4$, where $X_4 = X_1 \cdot X_2$. It is not necessary to change the functionality of the preceding or succeeding stages of the XOR gate, when $k = 0$.

$$X_4^{'} = X_4 \oplus 0 = X_4 = X_1 \cdot X_2 \quad (1)$$

To preserve the original functionality for $k = 1$, it is required either to invert the functionality of the preceding stage (Figure 2.(c)) or compensate the functionality of the following stage (Figure 2.(d)) of the added XOR gate. For the first case, the original functionality preserves as $X_4^{'} = 1 \oplus \overline{X_4} = X_4$. For the second case, DeMorgan's transformation is necessary and shown below:

$$Y = \overline{\overline{X_3} + X_4^{'}} = \overline{\overline{X_3}} \cdot \overline{X_4^{'}} = X_3 \cdot \overline{(1 \oplus X_4)} = X_3 \cdot X_4 \quad (2)$$

Note that only XOR gates are used in the example to lock the netlist. However, one can also use XNOR gates for such purposes. It is important to remember that one cannot use XOR gate for $k = 0$ and XNOR gate for $k = 1$ for every key bit. It is then trivial for an adversary to determine the secret key just by simply observing the key gates.

3 PROPOSED TOPOLOGY GUIDED ATTACK ON LOGIC LOCKING

The research community primarily focuses on evaluating the security of a logic locking technique through SAT-based analysis after the seminal attack presented in [35]. However, the SAT-attacks may pose few practical limitations to an adversary (e.g., untrusted foundry). An adversary must possess a working chip to launch the SAT attacks as it is required an oracle to discard equivalent class of incorrect keys. Note that an untrusted foundry needs to wait for an unlocked chip available in the market, and simply cannot unlock it after receiving the layout information (GDSII or OASIS files [30]) from an SoC designer. Many of the chips used in critical or DoD applications are rarely circulated in the market. Moreover, the attack even fails to estimate the correct key for a small benchmark circuit (e.g., *c6288*), and yet to be validated in large industrial designs. In this section, we present our proposed oracle-less and topology-guided attack, *TGA*, to break a logic locked circuit without possessing an unlocked chip.

3.1 Adversarial Model

The secure logic locking relies upon the fact that an adversary cannot determine or estimate the secret key from the locked netlist or an unlocked chip. The secret key is stored in a secure and tamper-proof memory so that an adversary cannot access the key values directly from an unlocked chip. In the attack model, the adversary is assumed to be an untrusted foundry and has the access to the following:

- *Gate-level netlist:* As the foundry is the primary attacker, it can have the access to the gate-level netlist of a locked circuit. The SoC designers typically send the circuit layout information using GDSII or OASIS files [30] to a foundry for chip fabrication. A foundry can extract the gate level netlist from the GDSII/OASIS files with the help of advanced tools [39].
- *Location of the key gates:* An adversary has the capability to determine the location of key gates. The key gates are connected either directly or through temporary storage elements to the tamper-proof memory. An adversary can easily track the routing path from the tamper-proof memory to the corresponding gates to determine their locations.
- *Locked unit function:* It is trivial for an untrusted foundry to construct equivalent unit functions *EUFs* for launching *TGA*, as it has the netlist and locations of the key gates.

3.2 Motivation for Designing TGA

The basic idea of launching our proposed attack is based on the repeated functionality that exists in a circuit. The Boolean functions are generally not unique in a circuit and repeated multiple times to implement its overall functionality. The majority of circuits are constructed based on small functional units. For example, several small functions (we describe as 'unit functions' or *UFs*) are repeated in an arithmetic logic unit (ALU) of a processor, adders, multipliers, advance encryption standards (AES), RSA, and many other digital circuits. If any of such unit functions are not obfuscated during the logic locking process, all the locked functions will be unlocked simply by comparing them with their unlocked version.

Figure 3 shows a four-bit ripple carry adder circuit to illustrate our concept of attacking the logic locked circuit. The adder consists of eight identical one bit half adders (*HA*) with inputs (*P* and *Q*) and outputs (*S* and *C*). It is clear from the figure that *HA* is repeated in the design and can be treated as a *UF*. If one of these half adders is locked using an XOR gate, an adversary only need to find an original *HA*, and then match this with the locked *HA* to recover the key value (see details in Section 3.5).

3.3 Construction of Equivalent Unit Function

The objective of this attack is to find the key value without performing traditional SAT-based analysis that requires an oracle. When an untrusted foundry receives the layout and mask information from the designer, it can reconstruct the gate-level netlist of the locked circuit by reverse engineering [39]. It can then easily identify the

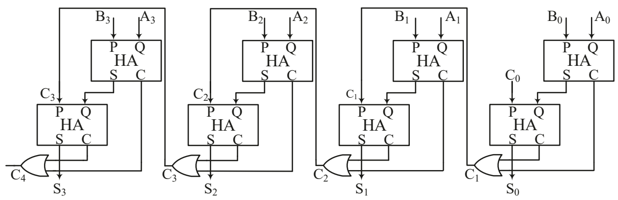

Figure 3: A 4-bit binary full adder (*FA*) consists of 8 half adders (*HA*). An adversary can recover the netlist for an locked *HA* by comparing with other *HAs*.

key gates by tracking the routes originated from the tamper-proof memory, where the secret key will be stored.

Our proposed attack constructs an equivalent unit function (*EUF*) using a hypothesis key bit, and searches that *EUF* in the entire netlist to find a match. The hypothesis key bit will be the actual secret key bit if a match is found. Otherwise, it constructs another *EUF* using the complementary hypothesis key bit and search the netlist again. The attack for determining a key bit fails, when the search fails to find a match.

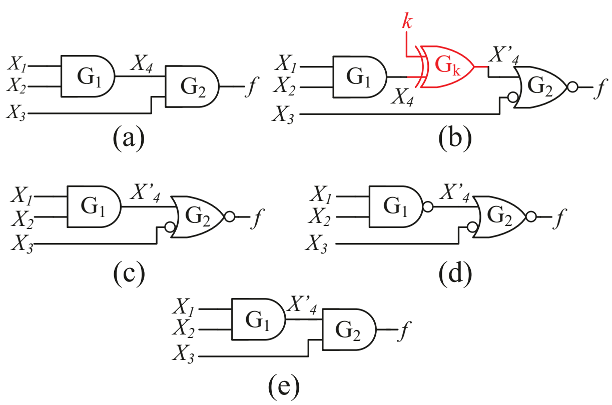

Figure 4: Equivalent unit functions for different hypothesis keys. (a) Original netlist. (b) Locked netlist with key value $k = 1$. (c) *EUF* for hypothesis key $k_h = 0$. (d) *EUF* for hypothesis key $k_h = 1$ (*Case-I*). (e) *EUF* for hypothesis key $k_h = 1$ (*Case-II*).

Figure 4 shows an example for constructing equivalent unit functions, which can be used to launch the function search attack. Figure 4.(a) represents an original unit function to be locked using a secret key $k = 1$. The locked circuit is shown in Figure 4.(b). The adversary does not know the value of the key, simply by observing the key gate. It first makes an assumption for $k_h = 0$, and constructs the *EUF*, which is shown in Figure 4.(c). It then searches this function in the locked circuit to find a match. If no match is found (as the actual key is 1), it constructs another *EUF* for $k_h = 1$. Two possible scenarios may occur. For *Case-I*, the output of the previous stage needs to be inverted (shown in Figure 4.(d)). On the other hand, DeMorgan's transformation needs to be carried out to obtain the *EUF* for $k_h = 1$ for *Case-II*, which is shown in Figure 4.(e).

3.4 Unit Function Search using DFS Algorithm

In order to launch the attack, we develop an efficient search algorithm, which performs a search of the *EUFs* in the locked netlist. Since the structure of a circuit can be transformed and represented by a directed graph, all the algorithms that could be used to search the component in the directed graphs, could also be applied to search the *EUF*. Therefore, we propose to use the Depth-First-Search (DFS)-based algorithm to launch the attack. Generally, the DFS method follows the rule: in the graph traverse, always select the next edge from the most recently reached and connected vertex that still has unexplored edges [36]. In addition, when the problem comes to find the specific component in the circuit, some preprocessing of the data structure is necessary.

The procedure of DFS-based search is described in Algorithm 1. For a given netlist, we first need to define a data object structure for all the gates. The gate object needs to have the following attributes: gate type (e.g., XOR, AND, etc), name of the gate (i.e., its identification in the netlist), an array that contains its preceding gates (i.e., its inputs), and an array contains its following gates (i.e., its outputs). Then the circuit structure could be transformed into a dictionary, in which the keys are the types of the gates and the values are corresponding gate objects. Dictionary is basically a data structure that stores mappings and relationships of data [10]. The benefit of using a dictionary is that it makes the searching of specific type of gates becomes efficient. When a specific *UF* need to be searched in this netlist, we define the last gate of the *UF* as the root gate (Line 2 in the Algorithm 1). An example root gate is G_2 in the Figure 4). All the gates are searched, which are of the same type with the root gate in the dictionary (Line 3) and store them into an array. The *DFS* is then performed on all these found gates (Line 3-7). Finally, all the *UFs* in the netlist will be found and the count of the *UF* will be returned as the output (Line 8).

The detailed implementation of the DFS, which is used for searching of *UFs*, is demonstrated in Lines 9-38. The general idea can be described as follows: for every gate that is the same type with the root gate of the *UF*, we traverse all its preceding gates to check whether the existence of the same structure. We implement the entire function using Python 2.7 [25]. The worst case time complexity of the search algorithm is $O(n * u)$, where n is the size of netlist and u is the size of a unit function. This is an acceptable complexity result, since it is shown that the subgraph isomorphism problem is an NP-complete problem and its time complexity is quadratic in the number of nodes [12, 30]. Note that, the optimization of the algorithm complexity is not the major objective of this paper. However, our search strategy slightly reduces the search complexity by using a dictionary to locate root gates. In this case, the algorithm performs similar to a substree isomorphism search (or a sequence of tree isomorphism searches), which complexity is known to be at least subquadratic [1]. Reading the netlist and transforming it into a dictionary may have different complexity, and the complexity we mentioned does not consider the complexity of constructing a dictionary.

3.5 Proposed *TGA* using Unit Function Search

The objective of *TGA* is to find the value of a secret key bit using unit function search. To determine the value of a key bit k_i, different

Algorithm 1: Function *UFS*
Unit Function search based on DFS Algorithm.

```
Input  : The gate-level netlist of a circuit (C), Unit Function (UF)
Output : Result List (L_R)

Read C and UF, and transform them into dictionaries, O and T;
R ← UF.root; L_S ← O[R.type]; L_R ← ϕ ;
for each gate G in L_S do
    if DFS(R, G) then
        L_R.append(G);
    end
end
return L_R;

Function DFS(r, g):
    F ← True;
    L_1 ← r.PrecedingGates; L_2 ← g.PrecedingGates;
    T_1 ← L_1.types; T_2 ← L_1.types;
    if L_1 is empty then
        return True;
    end
    for each gate type T in T_1 do
        if gate type T not in T_2 then
            return False;
        else
            T_2.remove(T)
    end
    for each gate R_N in L_1 do
        L_T ← ϕ ;
        for each gate G_T in L_2 do
            if G_T.type = R_N.type then
                L_T.append(G_T);
            end
        end
        F_T ← False;
        for each gate G_N in L_T do
            if DFS(R_N, G_N) then
                F_T ← True;
                break
            end
        end
        F ← F * F_T;
    end
return F
```

Algorithm 2: *TGA*

```
   Input  : Locked Circuit Netlist (C*)
   Output : List of predicted key values (K_P), Success Rate (SR)

1  Read the netlist C*;
2  Determine the location and number |K| of key gates;
3  Initialization for correct prediction counter, p_c ← 0 ;
4  for i ← 1 to |K| do
5      Initialization for layer counter, l ← 1;
6      Construct equivalent unit functions: EUF_0 for k_h = 0 and
       {EUF_1^1, EUF_1^2} for k_h = 1;
7      r_0 = UFS(C*, EUF_0).sz();
8      r_1 = UFS(C*, EUF_1^1).sz() + UFS(C*, EUF_1^2).sz();
9      if r_0 > 0 and r_1 > 0 then
10         l ← l + 1 and go to Line 6;
11     else if r_0 > 0 or r_1 > 0 then
12         if r_0 > 0 then
13             k_i ← 0;
14         else if r_1 > 0 then
15             k_i ← 1;
16         end
17         Write k_i into K_P; p_c ← p_c + 1;
18     else
19         k_i ←X and write k_i into K_P;
20     end
21     if Key gate is placed in a fan-out net then
22         k_i = FV( );
23         Update k_i into K_P;
24     end
25 end
26 Compute success rate, SR ← p_c/|K| × 100%;
27 Output K_P, SR;

28 Function FV( ):
29     Construct different EUFs for the fanout paths;
30     Search EUFs for each path and make key prediction ;
31     if Opposite predictions for different paths then
32         k_i ←X ;
33     else if Same predictions for different paths then
34         k_i ← {0 or 1};
35     end
36 return k_i
```

unit functions are constructed corresponding to the hypothesis key $k_h = 0$ or $k_h = 1$. If a match is found in the netlist, the corresponding key will be the secret key.

Algorithm 2 describes our proposed *TGA* using *UFS* search. It takes the locked circuit (C^*) as the input and results the predicted key (K_P) and the success rate (*SR*). K_P contains the value of each key gates, which is 0, 1, or X. The X represents an unknown value when the search fails to find a match. The locations of the key gates can be found by tracking the routes originated from the tamper-proof memory, and their numbers can be determined, $|K|$. For the key gate i, three different unit functions, EUF_0 for $k_h = 0$, and EUF_1^1 and EUF_1^2 for $k_h = 1$, are constructed, (see Figure 4 for details) and shown in Line 6. The unit function search (*UFS*) need to be performed to determine the repeated instances of that *EUF* (Lines 7-8). r_0 and r_1 represent the count values (obtained by using *sz()* function) for two different key assumptions. If both the r_0 and r_1 are non-zero, it is necessary to increase the size of the equivalent unit function by increasing the layer. Here, l denotes how many layers are considered to construct the unit function. By default, this value is 1 (Line 5), which is shown in Figure 4. In the case of both the $r_0 > 0$ and $r_1 > 0$, the current $EUFs$ will not help the attacker to make a decision on the key value, the l needs to be increased by 1 (Line 10) and the algorithm will construct new equivalent unit functions with more inputs (Line 6).

If one of the r_0 or r_1 is greater than 0, the attacker makes a prediction on the key value, and the key value will be written into K_P while the prediction counter (p_c) will be increased by 1. The value of k_i will be 0 if $r_0 > 0$, which represents that there is a match for EUF_0 in C^* (Line 12). On the other hand, the value of k_i will be 1 if $r_1 > 0$, which represents that there is a match for either EUF_1^1 or EUF_1^2 in C^* (Line 14). No decision will be made if both of them are equal to 0 when the unit function is unique in the circuit and the adversary cannot make a prediction on the key value. Thus, an unknown value (X) is assigned to the corresponding key bit location (Line 19). It is also necessary to verify the key value assignment when a key gate is placed in a fan-out net (Lines 21-24). In addition, when a key gate is inserted at the fan-out, the function, $FV(\ )$ verifies the key decision on each path. It may happen that different paths for the same key gate may have different key predictions. No prediction will be made in case of any two (or more) paths provides opposite key predictions (Line 32). Correct predictions will be considered if these different paths make the same prediction (Line 34).

$$SR = \frac{p_c}{|K|} \times 100\% \quad (3)$$

Finally, the success rate is computed using Equation 3. The *TGA* attack algorithm finally reports predicted key K_P and *SR* (Line 27).

The proposed *TGA* may also lead to incorrect predictions. For example, it is possible that the actual key bit is 1 when *TGA* estimates it as 0, and vice versa. It is thus necessary to measure the accuracy of the proposed attack. The misprediction rate (*MR*) of *TGA* can be described as the ratio of the incorrect predictions to the key size and is presented using the following equation:

$$MR = \frac{p_i}{|K|} \times 100\% \quad (4)$$

where, p_i represents the total number of incorrect predictions.

4 COUNTERMEASURE FOR *TGA* ATTACK

As the *TGA* relies on self-referencing, it can be prevented if the insertion of the keys are carried out in such a way that the search always returns null. In other words, the attack will fail if we choose to place a key gate in a unique *UF* in the netlist or lock all the same *UFs* simultaneously. In this section, we present an automated key insertion algorithm that performs *UF* search in the netlist before placing a key gate into the netlist.

Algorithm 3 illustrates our proposed algorithm to prevent against *TGA*. The inputs of the algorithm are the original unlocked netlist (*C*) and key size (<K_{min}, K_{max}>), which indicates the number of key gates that need to be inserted. The algorithm reports the locked circuit netlist (C^*) with the secret key K^*. In the algorithm, n denotes the number of key gates that has been already inserted in the circuit and initialized to be 0 (Line 1). A gate is selected randomly from the original and unlocked netlist as the root gate and then the unit function is created (Lines 3-4). The $UFS(C, UF)$ and *.sz()* functions are executed, which returns r (Line 5). Here, r denotes the number of this selected unit function repeated in the circuit. A key gate is inserted in this *UF*, if $r = 0$ which represents that it is unique in the netlist (Line 6). Note that the *UF* will be modified randomly based on one of the modifications mentioned in Figure 2. The key bit value is written in the respective location of K^*, and the value of n will be increased by 1 (Lines 8-9).

Algorithm 3: Key gate insertion

Input : Gate level netlist of a circuit (C), Key size (<K_{min}, K_{max}>)
Output : Locked netlist (C^*) and Key value (K^*)

```
1  Initialization: n ← 0, r ← 0;
2  while n < K_min do
3      Select a root gate randomly from C;
4      Construct the unit function, UF;
5      r ←UFS(C, UF).sz();
6      if r = 0 then
7          Insert the key gate and assign key value, k_i;
8          Write key value, K*[n] ← k_i;
9          n ← n + 1;
10     else if 0 < r ≤ K_max − n then
11         Lock all the UFs;
12         Write key values to K*[n + r : n];
13         n ← n + (r + 1);
14     end
15 end
16 Output C* and K*;
```

The algorithm chooses a different gate (Line 3) if $r > K_{max} - n$, otherwise, it locks all the instances of this *UF* (Line 11). The respective key bit locations in K^* are written with the key value (Line 12). Note that it is not necessary to lock all these instances with one key value, i.e., all 0s or all 1s. One can choose a combination of 1s and 0s (circuits shown in Figures 2.(b) - (d)). However, it is required to lock all the instances. Finally, the value of n is increased by $r + 1$. Note that the Algorithm 3 is only designed to prevent *TGA*. Additional countermeasures [14, 15] focused on SAT attacks need to be considered simultaneously to make logic locking secure.

5 SIMULATION RESULTS AND DISCUSSIONS

The effectiveness of our proposed *TGA* is presented in this section. We provide an in-depth analysis for key prediction accuracy of *TGA* on ISCAS'85 [5] and ITC'99 [11] benchmark circuits. We use a HP server with Intel Xeon Silver 4116 @2.10GHz processor and 64 GB of RAM to launch the *TGA*.

5.1 Performance Analysis

Four different benchmark circuits, *c6288*, *c5315*, *b15*, *b17* are first selected for determining the success rate (*SR*) and misprediction rate (*MR*) of our proposed *TGA*. We created 100 instances of the locked circuit for each benchmark, where 128 key gates are placed randomly, and then attacked using Algorithm 2. For each locked circuit, the success rate (*SR*) is computed using Equation 3, and the misprediction rate *MR* is calculated by using Equation 4.

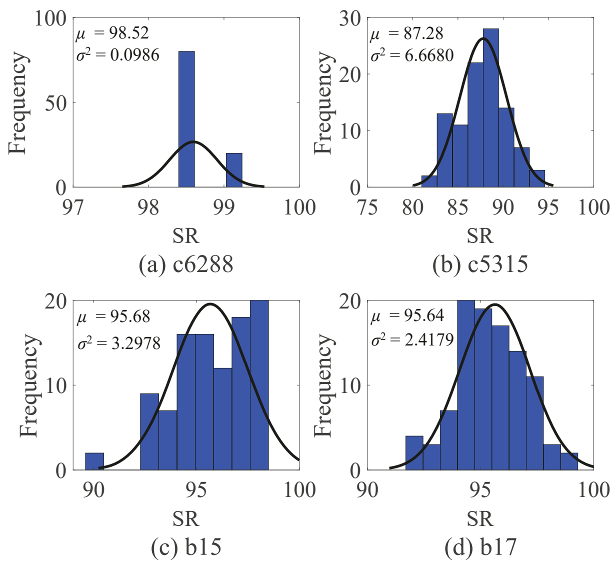

Figure 5: Histogram plots of the *SR* for different benchmark circuits with 128 key bits: (a) *c6288* (b) *c5315* (c) *b15* (d) *b17*

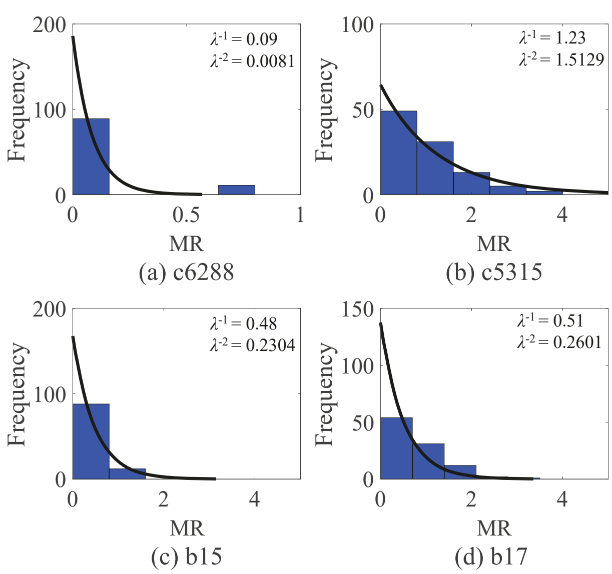

Figure 6: Histogram plots of the *MR* for different benchmark circuits with 128 key bits: (a) *c6288* (b) *c5315* (c) *b15* (d) *b17*

Figure 5 shows the histogram plots of *SR* metrics for four benchmark circuits. For benchmark circuit *c6288*, we estimate majority of the key bits (Figure 5) as this multiplier consists of many half and full adders. 126 out of 128 key bits can be predicted successfully, which results a minimum *SR* of 98%. Figure 5.(b) shows the *SR* distribution for *c5315* circuit. We observe a Gaussian distribution with a mean (μ) of 87.28% and variance (σ^2) of 6.6680 for this circuit. Similar behavior is observed for other two benchmark circuits (Figure 5.(c) and Figure 5.(d)). Note that the overall variance of the *SR* distribution is decreased when increasing the size of the benchmark circuits due to the increase of the *EUF* search space in the circuit graph, which makes our proposed *TGA* more effective for extracting keys in real designs.

The histogram plots of *MR* for the same benchmark circuits are shown in Figure 6. For *c6288* benchmark circuit, the key bits can be predicted correctly with a 0% *MR* in majority of cases. One key bit is predicted incorrectly, and thus the maximum value of *MR* is less than 1%. As for *c5315*, we observe an exponential distribution with a mean (λ^{-1}) of 1.23% and variance (λ^{-2}) of 1.5129 for this circuit. Similar behavior can be observed for *b15* and *b17* (Figure 6.(c) and Figure 6.(d)) benchmark circuits. In general, the mean and variance are presented by λ^{-1} and λ^{-2} for exponential distributions, whereas they are represented by μ and σ^2 for Gaussian distributions. Based on the observation, both mean and variance of *MR* are decreased with the increase of the size of the benchmark circuits, which makes *TGA* more accurate for larger designs.

Table 1 shows the success rate (*SR*) and misprediction rate (*MR*) of the proposed *TGA* attacks on ISCAS'85 and ITC'99 benchmark circuits. The number of logic gates and inserted key gates are presented in Columns 2 and 3, respectively. The total area overhead due to the inserted number of key gates is constrained to 10% to insert 128 key gates. However, the overhead added by the key gates can be negligible for larger designs with thousands of gates. Columns 4, 5, and 6 show the minimum, average, and maximum *SR* values (see Equation 3) by analyzing 100 locked instances for each benchmark circuit to determine the accuracy of *TGA* (see Algorithm 2 for details). For *c7552* benchmark, 128 key gates are inserted randomly in the netlist with 3512 logic gates. The minimum accuracy of 69.53% is observed, where the attack predicts 89 out of 128 key value correctly and the maximum prediction accuracy attained is 88.28%, where the attack identifies 113 key bits. Similar analysis can be performed for all the benchmarks shown in each row. For the larger benchmark circuits, the average success rate *SR* can be increased over 90% because of the increased search space, which makes our proposed *TGA* efficient for larger designs. Note that, although *SAT* fails on benchmark c6288, *TGA* provides better accuracy (average of 98.52%) for benchmark *c6288* due to its special topology – it is a multiplier, which consists of 225 full adders and 15 half adders. Therefore, an adversary can choose *TGA* as an alternate of SAT attacks.

The accuracy of the proposed *TGA* is evaluated as well, as it is necessary to determine the correctness of estimated *SR*. The minimum, average, and maximum misprediction rate, *MR*, are calculated using Equation 4 and provided at Columns 7, 8, and 9, respectively of Table 1. We observe an exponential distribution (see Figure 6) for *MR*. The average *MR* is less than 1% for majority of benchmark circuits, which makes *TGA* very effective for determining the secret key. Note that it can reach to a higher value for some benchmark circuits (e.g., 4.69% for *c7552*, where 6 key bits are predicted incorrectly). Our future work will be analyzing higher *MR* values to increase the accuracy of *TGA*.

Table 1: Success rate (*SR*) and misprediction rate (*MR*) for estimating keys for locked benchmark circuits.

Benchmark	# Total Gates	# Key Gates	Success Rate (*SR*)			Misprediction Rate (*MR*)		
			Min.	Avg.	Max.	Min.	Avg.	Max.
c3540	1669	128	75.22%	79.61%	87.50%	0.00%	1.76%	3.12%
c5315	2307	128	81.25%	87.80%	94.53%	0.00%	1.23%	3.91%
c6288	2406	128	98.44%	98.52%	99.22%	0.00%	0.09%	0.08%
c7552	3512	128	69.53%	79.87%	88.28%	0.00%	2.03%	4.69%
b14	3461	128	85.16%	93.38%	97.66%	0.00%	0.52%	3.12%
b15	6931	128	89.85%	95.68%	98.44%	0.00%	0.48%	1.56%
b20	7741	128	92.97%	96.39%	99.22%	0.00%	0.25%	1.56%
b21	7931	128	89.06%	94.62%	98.44%	0.00%	0.35%	1.56%
b22	12128	128	92.97%	95.56%	98.44%	0.00%	0.37%	1.56%
b17	21191	128	92.19%	95.64%	99.22%	0.00%	0.51%	3.12%

5.2 Complexity Analysis

The time complexity of a typical SAT-resistant locking method is exponential to the size of secret key [43], since it considers all possible key combinations. However, our proposed *TGA* does not require to compare any input and output pairs, and all the inserted key gates would be analyzed individually. Therefore, the time complexity of *TGA* itself is simply linear to the key size, namely, $O(|K|)$. Note that, our attack algorithm is based on *UFS*, the actual overall complexity is $O(|K| * n * u)$ where n and u represent the size of the netlist and average size of the unit functions, respectively. Thus the complexity could be considered as linear for a particular circuit, since the netlist size is fixed, and the size of *UF* normally ranges from 3-10 depending on the key gate location.

Table 2: Attack Effort of TGA

Benchmark	# Total Gates	Approximate Attack Effort (*AE*)		
		$\|K\|$= 128	$\|K\|$= 256	$\|K\|$= 512
c3540	1669	2^{18}	2^{19}	2^{20}
c5315	2307	2^{19}	2^{20}	2^{21}
c6288	2406	2^{19}	2^{20}	2^{21}
c7552	3512	2^{19}	2^{20}	2^{21}
b14	3461	2^{19}	2^{20}	2^{21}
b15	6931	2^{20}	2^{21}	2^{22}
b20	7741	2^{20}	2^{21}	2^{22}
b21	7931	2^{20}	2^{21}	2^{22}
b22	12128	2^{21}	2^{22}	2^{23}
b17	21191	2^{22}	2^{23}	2^{24}

Table 2 shows an estimated attack effort (*AE*) for different benchmark circuits. *AE* is determined by the number of gate search that an adversary needs to perform. For *b17* benchmark, it is required approximately 2^{22} searches to determine the complete 128 key bits, whereas, it takes 2^{23} and 2^{24} searches for 256 and 512 key bits, respectively. Note that the number of searches increases linearly for a circuit as we expected. For a modern computer, it takes only few minutes to complete these searches for launching *TGA*.

6 FUTURE WORK

In the future, we plan to evaluate the effectiveness of our proposed attack on the state-of-the-art SAT-resistant countermeasures [14, 15, 41, 43, 46, 47]. As many of today's SAT-resistant techniques use conventional locking using XORs/MUXes to modify the functionality, determining these keys will collapse the security provided from SAT-resistant blocks. For example, SARLock [43] and Anti-SAT [41] use one-point functions to obtain resilience against SAT attack. Both the techniques are dependent on additional blocks that protects the original locked netlist by inverting/corrupting the logic value at internal node or primary output (*PO*) for incorrect key value. As both the techniques rely on traditional logic locking to lock the functionality of the original netlist, which makes them vulnerable to this proposed attack.

7 CONCLUSION

In this paper, we presented a novel oracle-less and topology guided attack (*TGA*) that uses function search to break an existing secure logic locking technique. The unit functions are generally instantiated multiple times in a netlist. If a key gate is placed in one of these instances, an adversary can perform a search with the *EUF* formed using a hypothesis key bit. If a match is found in the netlist, the hypothesis key becomes the actual key bit. As this proposed attack does not require any input/output data from an unlocked chip, SAT resistant solutions cannot prevent an adversary launching this attack. The success rate (*SR*) and misprediction rate (*MR*) metrics are proposed to evaluate the effectiveness of this attack. The simulation results show that we can accurately determine majority of the secret key bits. Note that the complexity of launching *TGA* is linear with the key size, which makes it very effective for any designs. We also present a solution to prevent *TGA*, where it is required to lock all the repeated instances of an *UF*. Note that this solution can only be used to prevent *TGA*. To design a secure logic locking technique, one needs to select an existing secure logic locking technique along with our proposed solution. Our future work is to evaluate the performance of *TGA* on the state-of-the-art secure logic locking techniques.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under grant number CNS-1755733.

REFERENCES

[1] Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. 2018. Subtree isomorphism revisited. *ACM Transactions on Algorithms (TALG)* 14, 3 (2018), 27.

[2] Yousra Alkabani, Farinaz Koushanfar, and Miodrag Potkonjak. 2007. Remote activation of ICs for piracy prevention and digital right management. In *Proc. of IEEE/ACM international conference on Computer-aided design.* 674–677.

[3] Yousra M. Alkabani and Farinaz Koushanfar. 2007. Active hardware metering for intellectual property protection and security. In *Proc. of USENIX Security Symposium.* 20:1–20:16.

[4] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. 2010. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers* (2010), 66–75.

[5] David Bryan. 1985. The ISCAS'85 benchmark circuits and netlist format. *North Carolina State University* 25 (1985).

[6] Encarnación Castillo, Uwe Meyer-Baese, Antonio García, Luis Parrilla, and Antonio Lloris. 2007. IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores. *IEEE Trans. Very Large Scale Integr. Syst.* (2007), 578–591.

[7] R.S. Chakraborty and S. Bhunia. 2008. Hardware protection and authentication through netlist level obfuscation. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design.* 674 –677.

[8] Rajat Subhra Chakraborty and Swarup Bhunia. 2009. HARPOON: an obfuscation-based SoC design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2009), 1493–1502.

[9] E. Charbon. 1998. Hierarchical watermarking in IC design. In *Custom Integrated Circuits Conference.* 295–298.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms.* MIT press.

[11] Scott Davidson. 2019. https://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[12] Peter J Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. 2003. On graphs with unique node labels. In *International Workshop on Graph-Based Representations in Pattern Recognition.* Springer, 13–23.

[13] Ujjwal Guin, Qihang Shi, Domenic Forte, and Mark M Tehranipoor. 2016. FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. *ACM Transactions on Design Automation of Electronic Systems* (2016).

[14] U. Guin, Ziqi Zhou, and A. Singh. 2017. A novel design-for-security (DFS) architecture to prevent unauthorized IC overproduction. In *Proc. of the IEEE VLSI Test Symposium (VTS).* 1–6.

[15] Ujjwal Guin, Ziqi Zhou, and Adit Singh. 2018. Robust design-for-security architecture for enabling trust in IC manufacturing and test. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2018), 818–830.

[16] Jiawei Huang and J. Lach. 2008. IC activation and user authentication for security-sensitive systems. In *Proc. of IEEE International Workshop on Hardware-Oriented Security and Trust.* 76 –80. https://doi.org/10.1109/HST.2008.4559056

[17] Richard Wayne Jarvis and Michael G McIntyre. 2007. Split manufacturing method for advanced semiconductor circuits. US Patent 7,195,931.

[18] A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, Huijuan Wang, and G. Wolfe. 2001. Constraint-based watermarking techniques for design IP protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2001), 1236–1252.

[19] Soroush Khaleghi, Kai Da Zhao, and Wenjing Rao. 2015. IC piracy prevention via design withholding and entanglement. In *The 20th Asia and South Pacific Design Automation Conference.* 821–826.

[20] F Koushanfar and Gang Qu. 2001. Hardware metering. In *Proc. IEEE-ACM Design Automation Conference.* 490–493. https://doi.org/10.1109/DAC.2001.156189

[21] Yu-Wei Lee and Nur A Touba. 2015. Improving logic obfuscation via logic cone analysis. In *Latin-American Test Symposium (LATS).* 1–6.

[22] Nimisha Limaye, Abhrajit Sengupta, Mohammed Nabeel, and Ozgur Sinanoglu. 2019. Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access. *arXiv preprint arXiv:1906.07806* (2019).

[23] Bao Liu and Brandon Wang. 2014. Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In *Proceedings of the conference on Design, Automation & Test in Europe.* 243.

[24] Stephen M Plaza and Igor L Markov. 2015. Solving the third-shift problem in IC piracy with test-aware logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2015), 961–971.

[25] Python-2.7. 2019. https://www.python.org/download/releases/2.7/.

[26] Gang Qu and Miodrag Potkonjak. 2003. *Intellectual property protection in VLSI designs: theory and practice.* Springer Science & Business Media.

[27] Md Tauhidur Rahman, Domenic Forte, Quihang Shi, Gustavo K Contreras, and Mohammad Tehranipoor. 2014. CSST: an efficient secure split-test for preventing IC piracy. In *IEEE North Atlantic Test Workshop.* 43–47.

[28] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. 2012. Security analysis of logic obfuscation. In *Proc. of ACM/IEEE on Design Automation Conference.* 83–89.

[29] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. 2015. Fault analysis-based logic encryption. *IEEE Transactions on Computers* (2015), 410–424.

[30] Alfred J Reich, Kent H Nakagawa, and Robert E Boone. 2003. OASIS vs. GDSII stream format efficiency. In *23rd Annual BACUS Symposium on Photomask Technology*, Vol. 5256. 163–174.

[31] J.A. Roy, F. Koushanfar, and I.L. Markov. 2008. EPIC: Ending Piracy of Integrated Circuits. In *Proc. on Design, Automation and Test in Europe.* 1069 –1074. https://doi.org/10.1109/DATE.2008.4484823

[32] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. 2017. AppSAT: Approximately deobfuscating integrated circuits. In *Int. Symp. on Hardware Oriented Security and Trust.* 95–100.

[33] Yuanqi Shen and Hai Zhou. 2017. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI.* 179–184.

[34] Deepak Sirone and Pramod Subramanyan. 2019. Functional Analysis Attacks on Logic Locking. *in Proc. DATE* (2019).

[35] P. Subramanyan, S. Ray, and S. Malik. 2015. Evaluating the security of logic encryption algorithms. In *Int. Symp. on Hardware Oriented Security and Trust.* 137–143.

[36] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* (1972), 146–160.

[37] Mohammad Tehranipoor and Cliff Wang. 2012. *Introduction to Hardware Security and Trust.* Springer.

[38] Mark (Mohammad) Tehranipoor, Ujjwal Guin, and Domenic Forte. 2015. *Counterfeit Integrated Circuits: Detection and Avoidance.* Springer.

[39] Randy Torrance and Dick James. 2009. The state-of-the-art in IC reverse engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems.* 363–381.

[40] Kaushik Vaidyanathan, Renzhi Liu, Ekin Sumbul, Qiuling Zhu, Franz Franchetti, and Larry Pileggi. 2014. Efficient and secure intellectual property (IP) design with split fabrication. In *Int. Symp. on Hardware Oriented Security and Trust.* 13–18.

[41] Yang Xie and Ankur Srivastava. 2016. Mitigating sat attack on logic locking. In *Int. Conf. on Cryptographic Hardware and Embedded Sys.* 127–146.

[42] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. 2017. Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks. *International Conference on Cryptographic Hardware and Embedded Systems (CHES)* (2017).

[43] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *Int. Symp. on Hardware Oriented Security and Trust.* 236–241.

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

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

[46] Muhammad Yasin, Abhrajit Sengupta, M Ashraf, M Nabeel, J Rajendran, and Ozgur Sinanoglu. 2017. Provably-Secure Logic Locking: From Theory To Practice. In *ACM/SIGSAC Conference on Computer & Communications Security.* 1–1.

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