Noname manuscript No.
(will be inserted by the editor)

A Novel Topology-Guided Attack and Its Countermeasure Towards Secure Logic Locking

Yuqiao Zhang · Ayush Jain · Pinchen Cui · Ziqi Zhou · Ujjwal Guin

the date of receipt and acceptance should be inserted later

Abstract The outsourcing of the design and manufacturing of integrated circuits (ICs) in the current horizontal semiconductor integration flow has posed various security threats due to the presence of untrusted entities, such as overproduction of ICs, sale of out-of-specification/rejected ICs, and piracy of Intellectual Properties (IPs). Consequently, logic locking emerged as one of the prominent design for trust techniques. Unfortunately, these locking techniques are now inclined to achieve complete Boolean satisfiability (SAT) resiliency after the seminal work published in [47]. In this paper, we propose a novel oracle-less attack that is based on the topological analysis of the locked netlist even though it is SAT-resilient. The attack relies on identifying and constructing unit functions with a hypothesis key to be searched in the entire netlist to find its replica. The proposed graph search algorithm efficiently finds the duplicate functions in the netlist, making it a self-referencing attack. This proposed attack is extremely efficient and can determine the secret key within a few minutes. We have also proposed a countermeasure to make the circuit resilient against this topology-guided attack to progress towards a secure logic locking technique.

Keywords Logic locking · Boolean satisfiability · Boolean functions · piracy · overproduction · directed graph · depth-first search

Y. Zhang · A. Jain · Z. Zhou · U. Guin
Department of Electrical and Computer Engineering,
Auburn University, Auburn, AL, USA
E-mail:
{yuqiao.zhang, ayush.jain, ziqi.zhou, ujjwal.guin}@auburn.edu

P. Cui
Department of Computer Science and Software Engineering,
Auburn University, Auburn, AL, USA
E-mail: pinchen@auburn.edu

1 Introduction

The prohibitive cost of building and maintaining a foundry (fab) with advanced technology nodes has forced many design companies to become fabless and adopt the horizontal semiconductor integration model. Currently, majority of the design houses integrates intellectual properties (IPs) obtained from different third-party IP (3PIP) vendors along with its design and outsources the manufacturing to an offshore foundry resulting in a global supply chain with distributed vendors carrying out design, verification, fabrication, testing, and distribution of chips. The involvement of untrusted entities at various stages in the IC manufacturing and testing process has resulted in evident security threats, such as piracy or theft of IPs, overproduction of ICs, and sale of out-of-specification/rejected ICs [4,7,9,10,18,50]. Many design-for-trust techniques have been studied over the years as countermeasures against the aforementioned threats [4, 12, 18, 21, 23, 26, 33, 34, 37, 52].

Among the many, logic locking is the most widely accepted and studied design-for-trust technique to prevent threats from untrusted manufacturing and testing. Logic locking hides the circuit's inner details by incorporating key gates in the original circuit resulting in a key-dependent locked counterpart. The resultant locked circuit functions correctly once the secret key is programmed in its tamper-proof memory. Otherwise, it will produce erroneous outputs for the same input patterns, which makes it practically unusable. Over the years, different locking techniques are proposed, which can be primarily categorised based on key-insertion strategy (see Figure 1) and can be described as – (i) XOR-based [18–20,34,37], (ii) MUX-based [27,30,35], (iii) LUT-based [6, 25, 29], and (iv) state-space based [11]. However, XOR-based logic locking is popular due to its simplicity.

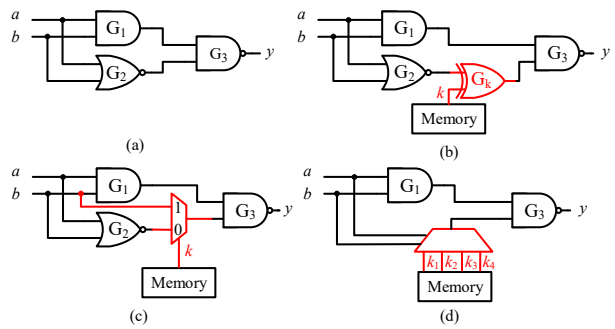

Fig. 1: Logic locking methods: (a) An original netlist (b) XOR/XNOR-based logic locking (c) MUX-based logic locking (d) LUT-based logic locking.

The research community is continuously driven to reveal logic locking vulnerabilities through attacks and to propose countermeasures in turn. The majority of early work was demonstrated vulnerable by oracle guided key-pruning attacks [47] and its variants [5, 42, 44, 45]. Since then, many SAT resilient solutions have been proposed [14, 18–20, 22, 24, 31, 39, 40, 53]. However, some of them have been broken as well [5, 13, 28, 41, 43, 46, 48]. Even though the SAT attacks are widely popular amongst the research community, the attack model assumes the availability of an oracle or a functionality correct (unlocked) IC pre-loaded with the correct key, and the adversary has the scan-chain access to obtain the input-output responses. This serves as the limitation 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 right after manufacturing. In addition, the concept of restricting scan-access has also been adopted to provide security against the SAT attacks. An adversary is not restricted to perform only SAT-based attacks as it may deploy other effective attacks to extract the secret key from a locked netlist. Therefore, it is necessary to consider and explore the different directions by which an untrusted foundry can exploit security vulnerabilities to undermine the security of logic locking.

In this paper, we propose a novel oracle-less attack on logic locked circuits to determine the key. Exploring the capabilities of an adversary, ***is it possible to determine the secret key simply by analyzing the circuit topology?*** The answer is yes, as the entire circuit topology is built from basic Boolean functions that are repeated multiple times. An adversary can determine the secret key by comparing the locked instances of these functions with the unlocked instances in the entire netlist. This proposed attack is an oracle-less self-referencing attack. We denote our proposed attack as ***TGA***: *T*opology-***G***uided ***A***ttack on logic locked circuits. By using our proposed attack, the secret key can be estimated efficiently even for the circuits that the SAT attack fails (see in Section 5 for *c6288* circuit). In addition, an adversary can unlock any netlist using our proposed attack without waiting for a working chip available in the market or with no scan access. This was further validated and demonstrated at *UF/FICS Hardware De-obfuscation competition at Trusted and Assured Microelectronics (TAME) forum [1, 2]*. The contributions of this paper are as follows:

1. *A novel oracle-less topology-guided attack on logic locking:* We proposed a topological function search attack that relies on identifying and searching the repeated functions in a netlist. We denote these basic functions as unit function *UF*, which are repeated multiple times in a circuit. If a key gate is placed in an instance of repeated *UF* 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, the number of key gates and key bits associated with a *UF* is limited, resulting in minimal *EUF* search combinations. The results in Table 1 show the efficiency of the proposed attack by recovering the majority of key bits correctly for ISCAS'85 and ITC'99 benchmark circuits locked with Random Logic Locking (*RLL*) and Secure Logic Locking (*SLL*). The effectiveness of our proposed *TGA* is also validated using locked benchmarks from TrustHub [38] (see Table 2). In contrast with the traditional oracle (unlocked chip) attacks, no oracle is required to launch our proposed attack.
2. *An efficient function search algorithm:* To perform the search, an efficient Depth-First-Search (*DFS*) based algorithm is developed to find the equivalent unit functions in a locked netlist. The complete netlist is first converted to a directed graph [49], where each gate in the netlist is represented as a vertex, and each wire is modeled as an edge. This paper demonstrates and implements a *DFS*-based *EUF* 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 a few minutes, locked with a few hundred/thousand key gates.
3. *A countermeasure against the proposed TGA attack:* As the proposed attack recovers the original design by performing the *EUF* search in the netlist, it can be prevented if the function search with hypothesis keys does not find results or produces contradictory results. This resiliency against the attack can be achieved by inserting the key gates in all the repeated instances of an *UF* as the adversary will not decide the actual value of the key bit by comparing it with its unlocked version. *DFS*-based search algorithm is again exploited to identify all repeated and unique instances of a unit function. Note that the key length can be variable in a range instead of a fixed value, which can increase both the efficiency of the key insertion and the security of the locked design.

The rest of the paper is organized as follows: the background of XOR-based logic locking is provided in Section 2. We present our proposed topology-guided attack methodology in Section 3. We present the countermeasure against the proposed attack in Section 4. We present the results for the implementation of the proposed attack on different logic locked benchmark circuits in section 5. Finally, we conclude our paper in Section 6.

2 XOR-based logic locking

To describe our proposed topology-guided attack based on function search, it is necessary to present XOR-based logic locking. Additionally, we need to analyze the resulting circuit modifications based on the selected correct key bit and the key gate type (either XOR or XNOR) to lock the original functionality. This will assist in building equivalent unit functions (*EUFs*) that will be searched in the netlist to perform the proposed attack.

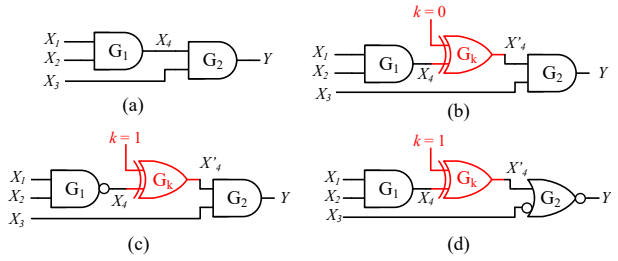

Fig. 2: Logic locking using Exclusive OR (XOR) gates. (a) Original netlist. (b) Locked netlist when k = 0. (c) *Case-I*: Locked netlist when k = 1. (d) *Case-II*: Locked netlist when k = 1 (using DeMorgan's Theorem).

Figure 2 shows an example to lock a circuit using an XOR gate, which has three inputs (X_1, X_2 and X_3) and one output (Y). One key gate with value k is selected to obfuscate the functionality of the circuit. The original circuit is shown in Figure 2.(a). There can be two possible key values, $k = 0$ and $k = 1$. For $k = 0$, an XOR gate can directly be placed 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 is $Y = X_3 \cdot X_4$, 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.

$$X_4^{'} = X_4 \oplus 0 = X_4 = X_1 \cdot X_2 \tag{1}$$

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 as shown below:

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

Note that only XOR gates are used in the example to lock the netlist. However, one can also use XNOR gates for such purposes, which has the opposite logic function compared with the XOR gate. It is important to remember that one cannot insert the XOR gate with $k = 0$ and XNOR gate with $k = 1$ for every key bit, as the adversary can determine the secret key just by observing the type of key gates.

3 Proposed Topology-Guided Attack on Logic Locking

The general locking strategy adopted to provide security in a circuit includes the placement of key gates either randomly or in some particular manner (e.g., pair-wise). Since, the secret key associated with the key gates is the same for all the chips manufactured with the same design, finding this key from one netlist undermines the security resulted from logic locking. In this section, we show how an adversary can easily extract the secret key for a key-based locked design using our proposed oracle-less and topology-guided attack, which is built on searching the hypothesis key-based equivalent unit function in the entire locked netlist. Moreover, this attack overcomes the limitations of SAT attacks that require an oracle with the scan access. For the same, we present the different steps involved in performing the proposed attack.

3.1 Adversarial Model

The specific objective is to undermine the security of a logic locking technique by determining the secret key. The secret key is stored in a secure and tamper-proof memory so that the adversary cannot access the key values directly from an unlocked chip. The adversarial model is presented to clearly state the resources and the assets possessed by an adversary. In our attack model, the adversary is assumed to be an untrusted foundry and has access to the following:

- *Gate-level netlist:* As the primary attacker, the foundry can have access to the gate-level netlist of a locked IC. The SoC designers typically send the circuit layout information using GDSII or OASIS files [36] to a foundry for chip fabrication. With the help of advanced tools, the foundry can extract the gate-level netlist from those provided GDSII/OASIS files [51].
- *Location of the key gates:* The location of key gates can be determined as these 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.

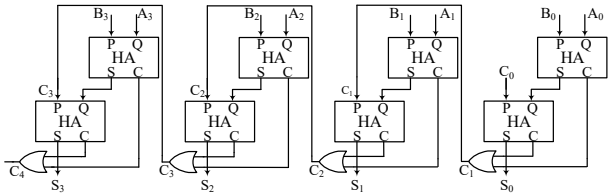

Fig. 3: Four-bit ripple carry adder consists of eight identical half adders (*HA*). If a *HA* is locked, an adversary can recover the original netlist by simply comparing it with other unlocked *HAs*.

– *Locked unit function:* It is trivial for an untrusted foundry to construct equivalent unit functions *EUFs* for launching the topology-guided attack, as it has the netlist and locations of the key gates.

3.2 Motivation

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 their 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 *UFs* are not obfuscated during logic locking, all the locked functions will be unlocked simply by comparing them with their unlocked version.

Figure 3 provides a four-bit ripple carry adder circuit as an example to illustrate the concept of our proposed attack. This full adder *FA* consists of eight identical one-bit half adders (*HA*) with inputs (P and Q) and outputs (S and C). Each individual half adder can be considered as a unit function *UF*, which is repeated multiple times inside this full adder. If one of these half adders is locked using an XOR/XNOR gate, an adversary only needs to find an original unlocked *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

Our proposed attack constructs an equivalent unit function to perform the search. While constructing the *EUF*, an adversary may encounter two different cases, either there is only one key gate, or there are multiple key gates in the *UF*. In either case, the (*EUF*) is constructed using one/more hypothesis key bits or a combination of hypothesis key bits, and searches that *EUF* in the entire netlist to find a match. The hypothesis key bits will be the correct secret key bits for the respective *UF* if a match is found corresponding to the *EUF*. Otherwise, it constructs another *EUF* using a different combination of values for the hypothesis key bits in both the cases and searches the netlist again. The number of *EUFs* depends on the number of key gates included in the *UF*. In this section, we show how *EUFs* are created to determine the secret key for both *RLL* and *SLL* circuits.

3.3.1 Random logic locking

In random logic locking (*RLL*), the key gates are inserted randomly inside the circuit that needs to be protected. In the large design with thousands of gates, it is highly unlikely that multiple key gates will be inserted adjacent to each other. Thus, the inserted key gates usually can be considered individually to construct the equivalent unit functions.

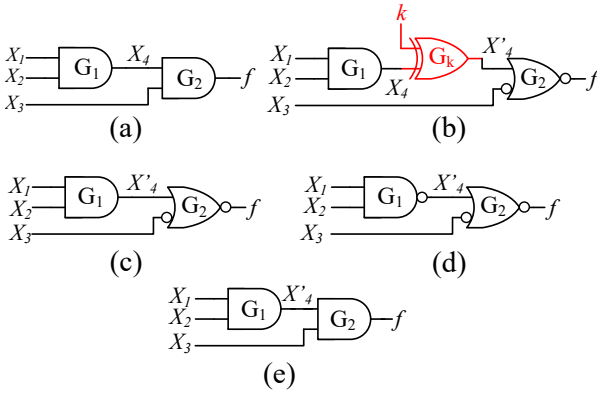

Fig. 4: *EUF* construction for different hypothesis key values. (a) Original unlocked netlist. (b) Netlist is secured with key value $k = 1$. (c) EUF_0 for hypothesis key $k_h = 0$. (d) EUF_1 for hypothesis key $k_h = 1$ (*Case-I*). (e) $EUF_{\hat{1}}$ for hypothesis key $k_h = 1$ (*Case-II*).

Figure 4 illustrates the construction of equivalent unit functions with a single key gate, which can be used to launch the function search attack. Figure 4.(a) represents an original unit function to be locked using a correct secret key $k = 1$. The locked circuit is shown in Figure 4.(b). The adversary cannot deduce 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). As inferred from the construction of the equivalent unit function, each key gate has two hypothesis keys with three transformations represented as: (i) EUF_0 where the hypothesis key $k_h = 0$, (ii) EUF_1 (*Case-I*) where the hypothesis key $k_h = 1$, and (iii) $EUF_{\hat{1}}$ (*Case-II*) where the

hypothesis key $k_h = 1$ but the modification is carried out using DeMorgan's Theorem. Three *EUFs* can be constructed for a single key bit. For a hypothesis key bit $k_h = 0$, a single implementation of *EUF* can be considered, whereas, two different implementations can be possible for hypothesis key bit $k_h = 1$. As a result, for an *UF* locked with a j-bit key, 3^j number of implementations can be possible. We need to perform all 3^j *EUF* search to determine the j-bit key.

3.3.2 Strong logic locking

The objective of strong logic locking (*SLL*) is to maximize the interference between different key gates to restrict key sensitization at the output [54]. In *SLL*, two or more key gates are inserted adjacent to each other so that their outputs converge at the next stage logic gate. The propagation of one of the key-bit will be possible only if certain conditions are forced on other key inputs or they are known. As these key inputs are not accessible by the attackers, they cannot force the logic values necessary to sensitize a key. As a result, the proposed *TGA* on *SLL* requires an equivalent unit function search with multiple keys instead of a single one for random logic locking.

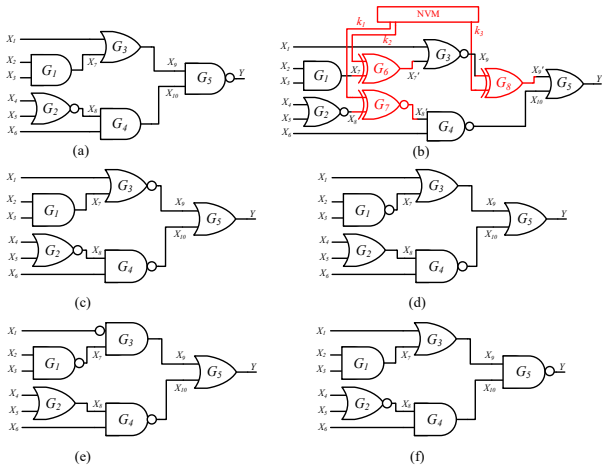

Fig. 5: Equivalent unit functions for multiple gates with different hypothesis keys. (a) Original netlist. (b) Locked netlist with key value $k_1k_2k_3 = 101$. (c) EUF_{100} for hypothesis key, $k_h = 100$. (d) EUF_{011} for $k_h = 011$. (e) $EUF_{0\hat{1}0}$ for $k_h = 010$. (f) $EUF_{10\hat{1}}$ for $k_h = 101$.

Figure 5 illustrates the construction of *EUFs* with multiple key gates that will assist in performing the function search. The original unit function (as shown in Figures 5.(a)) is locked with three key gates to increase the inter key-dependency. The locked unit function is shown in Figure 5.(b) with correct key $k_1k_2k_3 = 101$. As an adversary cannot extract the correct key value from the non-volatile memory directly, all the *EUFs* will be constructed and searched in the entire locked netlist. However, the number of constructed *EUFs* will increase due to the number of key gates and its combination in the locked *UF*. As mentioned earlier, each key gate results 3 different *EUFs* (e.g., EUF_0, EUF_1, and $EUF_{\hat{1}}$ based on the hypothesis key values (either 0 or 1). This will result in overall 27 *EUFs* (i.e., 3^3, as $j = 3$ is the number of key gates in the *UF*) for Figure 5.(b), amongst which only 4 of them are shown in Figure 5.(c)-(f). These *EUFs* are derived from different key combinations. For example, EUF_{100} in Figure 5.(c) is constructed with hypothesis key bits based transformation as EUF_1, EUF_0 and EUF_0 for key gates G_7, G_6 and G_8 respectively. Also, we construct EUF_{011} as shown in Figure 5.(d). if we transform based on the hypothesis key bits $k_h = 011$ for key gates G_7 (EUF_0), G_6 (EUF_1) and G_8 (EUF_1). Figure 5.(e) shows yet another *EUF* represented as $EUF_{0\hat{1}0}$, where the hypothesis key is 0, 1 (Case-II) and 0. Likewise, if we select the transformation as EUF_1, EUF_0 and $EUF_{\hat{1}}$ (Case-II) for key gates G_7, G_6 and G_8 respectively, then we will get the $EUF_{10\hat{1}}$ shown in Figure 5.(f). Once all the *EUFs* are constructed, all of them will be searched in the netlist to find a match. As Figure 5.(f) is identical to Figure 5.(a), the hypothesis key combination $k_h = 101$ should be the correct key value. If no such match is found for any of the *EUFs*, an adversary cannot make the prediction on the key combination resulting the *UF* being unique in the circuit.

3.4 Function Search using *DFS* Algorithm

An efficient search algorithm has been developed to search the *EUFs* in the locked netlist. The structure of a circuit can be transformed and represented as a directed graph, and all the algorithms that can be used to search the component in the directed graphs, can 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 procedure, the edge from the most recently reached and connected vertex that still has unexplored edges will always be selected as next edge [49]. Before performing the *DFS*-based search, a data object structure needs to be defined to store and transform the netlist as a directed graph. 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 can be transformed and stored 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 [15]. The use of a dictionary makes the search for specific type of gates more efficient.

The procedure of *DFS*-based search is described in Algorithm 1. The function *FS* finds matches for any function (*Fn*) in a circuit that it takes as an input. Whenever a specific *Fn* need to be searched in this netlist, we define the last gate

Algorithm 1: Function *FS*
Function search based on *DFS* Algorithm.

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

```
1  Read C and Fn, and transform them into dictionaries, O and
     T;
2  R ← Fn.root; L_S ← O[R.type]; L_R ← φ ;
3  for each gate G in L_S do
4      if DFS(R, G) then
5          L_R.append(G);
6      end
7  end
8  return L_R;
9  Function DFS (r, g) :
10     F ← True;
11     L_1 ← r.PrecedingGates; L_2 ← g.PrecedingGates;
12     T_1 ← L_1.types; T_2 ← L_1.types;
13     if L_1 is empty then
14         return True;
15     end
16     for each gate type T in T_1 do
17         if gate type T not in T_2 then
18             return False;
19         else
20             T_2.remove(T)
21     end
22     for each gate R_N in L_1 do
23         L_T ← φ ;
24         for each gate G_T in L_2 do
25             if G_T.type = R_N.type then
26                 L_T.append(G_T);
27             end
28         end
29         F_T ← False;
30         for each gate G_N in L_T do
31             if DFS(R_N, G_N) then
32                 F_T ← True;
33                 break
34             end
35         end
36         F ← F * F_T;
37     end
38 return F
```

of the *Fn* as the root gate (Line 2 in the Algorithm 1). An example root gate is G_2 in the Figure 4). All the gates that have the same type with the root gate (G_2) in the dictionary (Line 3) are stored 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 *F* will be returned as the output (Line 8). The detailed implementation of the *DFS* is demonstrated in Lines 9-38.

The algorithm is implemented with Python 2.7 [32]. The worst case time complexity of the search algorithm is $O(n * u)$, where n is the size of the netlist and u is the size of a unit function. This is an acceptable complexity, since it is known that the subgraph isomorphism problem is an NP-complete problem and its time complexity is quadratic in the number of nodes [17, 36]. 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 subtree isomorphism search (or a sequence of tree isomorphism searches), whose complexity is known to be at least subquadratic [3]. Reading the netlist and transforming it into a dictionary may have different complexity. The complexity analysis does not consider the complexity of constructing a netlist dictionary.

3.5 Proposed attack using Equivalent Unit Function Search

The objective of the proposed topology-guided attack is to recover the entire original netlist using the equivalent unit function search (*FS*). Algorithm 2 describes the proposed attack. The locked circuit (C^*) is given as the input, and the list of predicted key values (K_P) with the success rate (*SR*) will be returned as outputs. K_P contains the predicted value of each key gates, which can be either 0, 1, or X. The X represents an unknown value when the search fails to find a match and make the prediction. 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 as $|K|$. In order to determine the key value inside a particular unit function, different unit functions need to be constructed based on the number of key gates inserted in this unit function. In addition, each of the key gate comes with a hypothesis key value (either 0 or 1), and this also leads to the different hypothesis key combinations when there are multiple key gates inserted in a *UF*.

For each key gate k_i, the unit function will be constructed based on the value l. Here, l denotes how many layers of gates are considered when constructing the unit functions. The l is initialized as 1 at the beginning (Line 6), which is also shown in Figure 4. Next, the unit function based on the k_i and l will be generated (Line 7), and the number of key gates (includes k_i) in this unit function will be determined as j (Line 8). The hypothesis key combinations for all the key gates in this unit function will be generated and stored in a list J (Line 9). Note that the order of the keys has no relationship with the real sequence in the circuit, and the number of the combinations is 2^j. Once the key combination list is generated, all the possible *EUFs* will be constructed based on the hypothesis key combinations (Line 11). For each key gate, three different cases need to be considered (see Figure 4 for details), thus 3^j *EUFs* will be generated. The function search (*FS*) (described in section 3.4) is then performed to find the repeated instances of *EUFs* (Line 12-14). 2^j count values will be accumulated in a list R (initialized with all 0 in Line 10) for all key assumptions.

Upon finishing the search of all the *EUFs*, if only one count value in R is non-zero, this non-zero value corresponding *EUF* represents a correct key prediction. The hy-

pothesis key J' of this *EUF* will be written into K_P, and the prediction counter (p_c) will be increased by the length of this hypothesis key, j (Line 16-22). Note that, if the key gate is placed in a fan-out net, an additional process needs to be performed (Line 19-21). 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. As a result, no prediction will be made in case of any two (or more) paths provides the opposite key value predictions (Line 36-37). Correct predictions will only be made if different paths make the same prediction (Line 38-39).

On the other hand, if all of the elements in R are equal to 0, this means this unit function is unique in the circuit and the adversary cannot make a prediction on the key value. As a result, unknown value (X) is assigned to all the j key gates in this unit function, and the values are also stored in to K_P (Line23-25). In the case of multiple count values in R are non-zero, the adversary can neither make the key value prediction based on the current *EUF*. It is necessary to increase the size of the *EUF* by increasing the layer of gates considered in *EUF* constructions. Therefore, the l value needs to be increased by 1, and the entire searching procedure will be re-performed (26-28).

Algorithm 2: Topology-guided attack using *FS*

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 Initialize correct prediction counter, $p_c \leftarrow 0$;
4 **for** $i \leftarrow 1$ **to** $|K|$ **do**
5 **if** k_i *is not determined in* K_P **then**
6 Initialize layer counter, $l \leftarrow 1$;
7 Get the unit function for k_i based on l ;
8 Get number of key gates j in the function;
9 $J \leftarrow$ key combinations list, where the length of $J = 2^j$;
10 $R \leftarrow [0] * 2^j$;
11 Generate 3^j equivalent unit functions for the j-bit key ;
12 **for** *each generated EUF* **do**
13 $J' \leftarrow$ hypothesis key of EUF;
14 $R[J.index(J')] \leftarrow R[J.index(J')] + FS(C^*, EUF).sz()$;
15 **end**
16 **if** $R.nonzero$ = *1* **then**
17 $J' \leftarrow R.index(1)$;
18 Correct hypothesis key $k_j \leftarrow J[J']$;
19 **if** *Any key gate in* k_j *is placed in a fan-out net* **then**
20 $k_j = FV(\,)$;
21 **end**
22 Write k_j into K_P; $p_c \leftarrow p_c + j$;
23 **else if** $R.nonzero$ = *0* **then**
24 $k_1 ... k_j \leftarrow X$;
25 Write $k_1 ... k_j$ into K_P;
26 **else**
27 $l \leftarrow l + 1$, go to line 7
28 **else**
29 Continue
30 **end**
31 Compute success rate, $SR \leftarrow \frac{p_c}{|K|} \times 100\%$;
32 Output K_P, SR;
33 **Function** FV( ) **:**
34 Construct different *EUFs* for the fanout paths;
35 Search *EUFs* for each path and make key prediction ;
36 **if** *Opposite predictions for different paths* **then**
37 $k_i \leftarrow$X ;
38 **else if** *Same predictions for different paths* **then**
39 $k_i \leftarrow$ {0 or 1};
40 **end**
41 **return** k_i

$$SR = \frac{p_c}{|K|} \times 100\% \tag{3}$$

Finally, the success rate is computed using Equation 3. Here, $|K|$ presents the size of the key while p_c indicates the value stored in the correct prediction counter. The algorithm will finally report predicted key list K_P and *SR* (Line 27).

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

$$MR = \frac{p_i}{|K|} \times 100\% \tag{4}$$

where, p_i denotes the total number of incorrect predictions.

4 Countermeasure for *TGA*

In this section, we propose an effective key insertion algorithm, which can prevent the proposed topology-guided attack. As an adversary performs *EUF* search in the netlist to find out the reference *UF*, this attack can be prevented if the search of those key gates and *EUFs* always returns no results or contradictory values. The basic idea of the countermeasure is to lock all the repeated instances of *UFs* and insert the key gate(s) in all unique *UFs* in the circuit simultaneously. As a result, the adversary cannot predict and recover the correct key values by comparing the locked *UFs* with the unlocked version. In order to find all the repeated instances of selected *UF*, the *UF* search will be performed at the beginning before the key gates are placed into the netlist.

Algorithm 3 illustrates our proposed solution for key gate(s) insertion. The original unlocked netlist (C) will be provided as the initial input, along with the key size ($\langle\, K_{min}, K_{max}\, \rangle$), which indicates the range of number of key gates that needs to be inserted in the circuit. Finally, the locked circuit netlist (C^*) and the secret key K^* will

Algorithm 3: Insertion of key gates to prevent topology-guided attack

Input : Gate level netlist of a circuit (C),
Key size $(\langle K_{min}, K_{max} \rangle)$
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 ← FS(C, UF).sz();
6      if RLL then
7          if r = 1 then
8              Insert the key gate at one random input of root
                 gate and assign key value, k_n ∈ {0, 1};
9              Write key value, K*[n] ← k_n;
10             n ← n + 1;
11         else if 1 < r ≤ K_max − n then
12             Lock all the UFs;
13             Write key values to K*[n + r : n];
14             n ← n + r;
15         end
16     else if SLL then
17         if r = 1 then
18             Insert j key bits in the unique UF, and assign
                 key values, k_n, k_(n+1), ..., k_(n+j) ;
19             Write key value,
                 K*[n + j : n] ← [k_(n+j), ..., k_(n+1), k_n] ;
20             n ← n + j;
21         else if 1 < r ≤ K_max − n then
22             Lock all the UFs;
23             Write key values to K*[(n + r * j) : n];
24             n ← n + r * j;
25         end
26     end
27 end
28 Output C* and K*;
```

be the outputs of the algorithm. Here, n denotes the key index, which is the number of key gates that has been already inserted in the circuit and initialized to be 0 (Line 1). The entire process can be described as follows: First, a gate is selected randomly from the original unlocked netlist as the root gate (Line 3). Then, the unit function based on the root gate will be created (see Figure 4.(a)) for the *UF* search (Lines 4). Next, $FS(C, UF).sz()$ returns r , which denotes the number of this selected *UF* repeated in the circuit (Line 5). Depending on the value of r, whether 1 (unique) or greater than 1 (repeated), key gate(s) can be inserted in this *UF* in accordance with *RLL* or *SLL* techniques.

For *RLL*, $r = 1$ signifies the constructed *UF* is unique, and the *FS* function found only one instance (itself) in the netlist. As a result, a random key gate (either XOR or XNOR) will be inserted before the root gate and the *UF* will be modified randomly based on the key value. After the key gate insertion, the key bit value is written in the respective location of K^*, and the value of n will be increased by 1 (Lines 9-10). In the case of $r > K_{max} - n$ which represents that the number of this repeated *UF* is more than the maximum remaining number of key gates we expect to insert, the algorithm will randomly choose a different gate as the new root gate (Line 3). Otherwise, the algorithm will lock all the repeated instances of this constructed *UF* in the circuit (Line 12). The respective key bit locations in K^* are written with the key values (Line 13). Note that it is ineffective to lock all these instances with only one key value, i.e., all 0s or all 1s, as the attacker can recover the entire netlist by simply analyzing the type of the key gate. A combination of 1s and 0s (shown in Figures 2) will be a better option in order to provide enough security for the circuit. However, it is mandatory to lock all the repeated *UFs*. Finally, the value of n is increased by r.

Similarly, for $r = 1$, *SLL* can be carried out by inserting j key gates in the *UF*, namely $k_n, k_{n+1}, \ldots, k_{n+j}$ (line 18). After the insertion of the key gates, the value of these key bits is written in the respective location of K^*, and the value of n will be increased by j since j key gates has been inserted already (Lines 19-20). In the case of $r > K_{max} - n$ when the number of this repeated *UF* is more than the maximum remaining number of key gates, the algorithm will automatically choose a different gate as the new root gate (Line 3). Otherwise, the algorithm will lock all the repeated instances of this constructed*UF* with *SLL* (Line 23). The respective key bit locations and values is also updated in K^* (Line 24). At last, the value of n is increased by $r * j$.

When multiple identical *UFs* are selected, a designer may select to lock these *UFs* with a key of 0 or 1. However, an adversary can identify different locked versions of the same unit functions (*UFs*) and compare them to find the value of the secret key. For example, an adversary can construct the complete truth table of two locked *UFs* with $k_i = 0$ and $k_i = 1$, and compare their responses. As the correct key bit produces the same responses for both the functions, the key bits can be determined. As a result, if the same repeated *UF* is locked with different key bits, an adversary can determine their value by merely comparing it with the other locked version. An adversary, however, needs to identify the same repeated *UFs* for comparison. As a result, we propose to lock all the repeated UFs with the same key bit so that self-referencing becomes infeasible.

Another possible countermeasure against *TGA* can be developed by performing DeMorgan's transformation further from the key gate. When a key gate is inserted in a specific location, the required circuit modification due to the inserted key can be performed further away from it instead of modifying it is preceding or the following logic. As a result, the attacker may fail to make the correct key prediction when the *EUFs* constructed with few gates are searched in the entire circuit. However, an adversary constructs additional possible *EUFs* with DeMorgan's transformation further away from the key gate and searching the netlist. In our

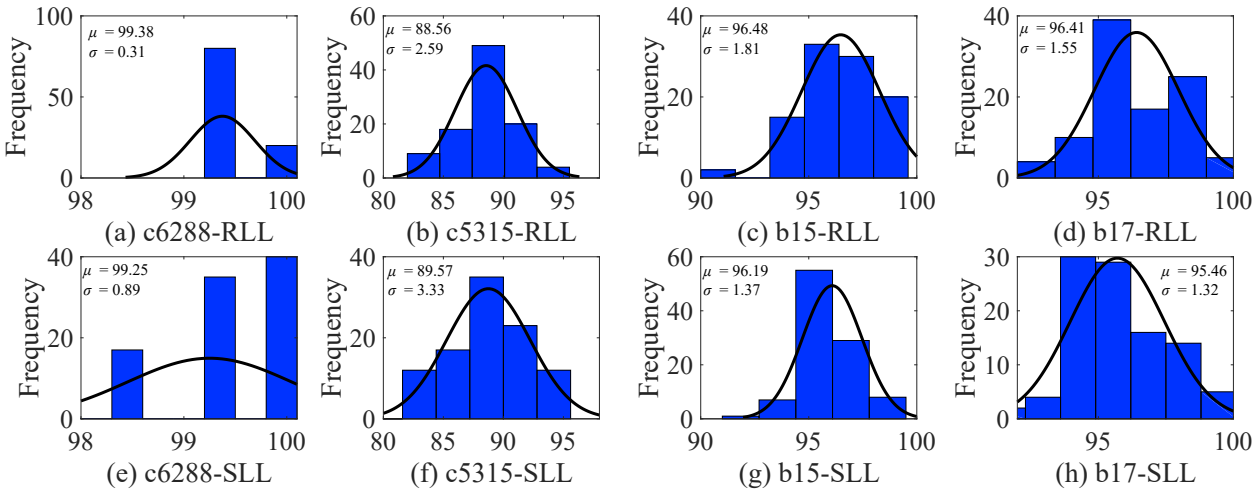

Fig. 6: Histogram plots of the *SR* for different benchmark circuits with 128 key bits: (a) *c6288-RLL* (b) *c5315-RLL* (c) *b15-RLL* (d) *b17-RLL* (e) *c6288-SLL* (f) *c5315-SLL* (g) *b15-SLL* (h) *b17-SLL*.

future work, we plan to explore the effectiveness of our proposed *TGA* with this countermeasure in place.

5 Simulation results and discussions

In this section, we present the results and evaluate the performance of our proposed topology-guided attack on different logic locking schemes. We provide an in-depth analysis for key prediction accuracy of the proposed attack on ISCAS'85 [8] and ITC'99 [16] benchmark circuits locked with *RLL* and *SLL* using our in-house script. In addition, we have validated our proposed attack on TrustHub benchmark circuits [38].

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 have created 100 instances of the locked circuit based on *RLL* and *SLL*) for each benchmark circuits, where 128 key gates are placed, and then attacked using Algorithm 2. For each locked circuit, the success rate (*SR*) is computed using Equation 3, while the misprediction rate *MR* is calculated using Equation 4. In general, the mean and standard deviation are presented by μ and σ for Gaussian distributions related to *SR*, whereas they are all represented by λ^{-1} for exponential distributions that is related to *MR* plots.

Figure 6 shows the histogram plots of *SR* metric for the four selected benchmark circuits based on *RLL* (see Figure 6.(a)-(d)) and *SLL* (shown in Figure 6.(e)-(h)). For benchmark circuit *c6288-RLL*, we estimate the majority of the key bits (Figure 6.(a)) as this multiplier consists of many half and full adders. 127 out of 128 key bits can be predicted successfully, which results in a minimum *SR* of 99.22%. Figure 6.(b) shows the *SR* distribution for *c5315-RLL* circuit. A Gaussian distribution is observed with μ of 88.56% and σ of 2.59. Similar behavior is observed for the other two benchmarks circuits as shown in Figure 6.(c) and Figure 6.(d). The μ for $b15-RLL$ and $b17-RLL$ are 96.48% and 96.41% with the σ values as 1.81 and 1.55 respectively. We observe a similar Gaussian distributions for the *SR* on locked circuits using SLL (see Figure 6.(e)-(h)).

The histogram plots of misprediction (*MR*) for the same selected benchmark circuits are presented in Figure 7. Figures 7.(a)–(d) present the *MR* plot for the circuits locked with *RLL*. For *c6288-RLL* benchmark circuit, all the key bits can be determined correctly with a 0% *MR* in majority of the cases. The worst case is one bit misprediction, resulting in maximum value of *MR* within 1%. As for *c5315-RLL*, we observe an exponential distribution with a mean and standard deviation (λ^{-1}) of 1.23. As observed from Figure 7.(c) and Figure 7, *b15-RLL* shows λ^{-1} of 0.48%, whereas *b17-RLL* shows λ^{-1} of 0.51. Likewise, a similar analysis can be done for *MR* for the same selected benchmark circuits locked with *SLL* plotted in Figure 7.(e)–(h).

Table 1 shows the success rate (*SR*) and misprediction rate (*MR*) of our proposed attack on different ISCAS'85 [8] and ITC'99 [16] benchmark circuits locked with *RLL* and *SLL* techniques. The number of logic gates in the circuit and inserted key gates are presented in Columns 2 and 3, respectively. The number of key gates is set to 32 for the first four benchmark circuits (e.g., c880, c1350, c1908, and c2670), while the remaining are inserted with 128 key gates. Columns 4 and 5 presents the mean value μ and standard deviation σ of *SR* values (see Equation 3) by analyzing 100 locked instances for each benchmark circuit to determine the accuracy of the proposed *TGA* (see Algorithm 2 for details)

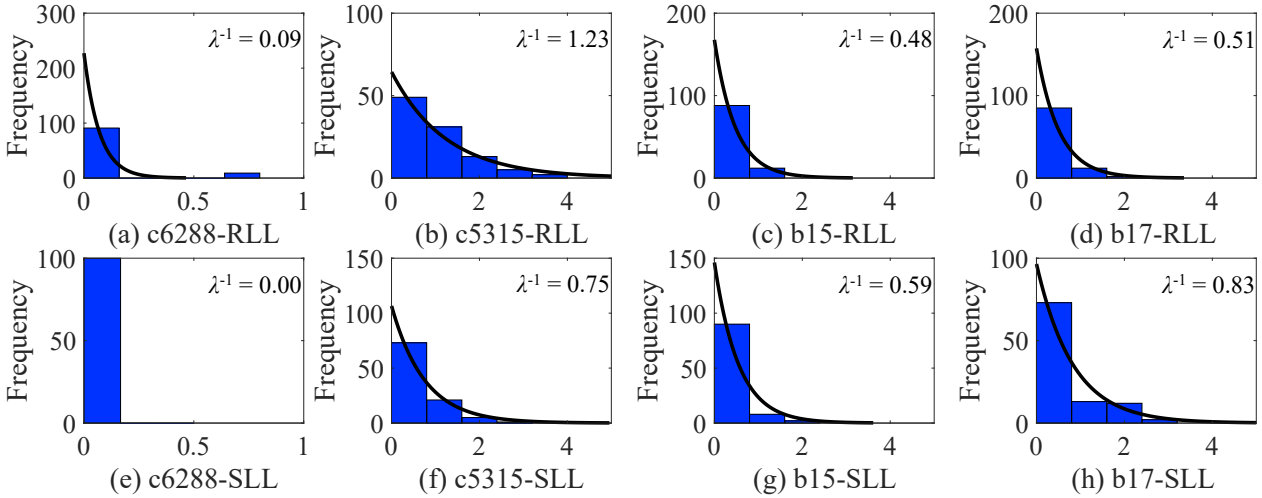

Fig. 7: Histogram plots of the *MR* for different *RLL* and *SLL* benchmark circuits with 128 key bits: (a) *c6288-RLL* (b) *c5315-RLL* (c) *b15-RLL* (d) *b17-RLL* (e) *c6288-SLL* (f) *c5315-SLL* (g) *b15-SLL* (h) *b17-SLL*

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

Benchmark	**# Total Gates**	**# Key Gates**	**RLL**			**SLL**		
			SR (%)		**MR (%)**	**SR (%)**		**(MR) (%)**
			μ	σ	λ^{-1}	μ	σ	λ^{-1}
c880	404	32	75.03	6.14	1.53	74.63	5.89	1.59
c1350	593	32	69.25	5.97	0.00	69.10	6.41	0.00
c1908	768	32	74.13	4.08	1.44	73.66	4.53	1.72
c2670	1193	32	75.22	4.98	1.13	74.78	5.07	1.06
c3540	1669	128	80.39	3.72	1.76	80.56	4.63	2.01
c5315	2307	128	88.56	2.59	1.23	89.57	3.33	0.75
c6288	2406	128	99.38	0.31	0.09	99.25	0.89	0.00
c7552	3512	128	91.08	3.17	2.03	90.21	2.79	0.97
b14	3461	128	94.16	2.00	0.52	93.67	2.10	0.92
b15	6931	128	96.48	1.81	0.48	96.19	1.37	0.59
b20	7741	128	97.17	1.44	0.25	96.95	1.48	0.84
b21	7931	128	95.40	1.71	0.35	94.50	1.86	0.63
b22	12128	128	96.34	1.25	0.37	95.78	1.62	0.77
b17	21191	128	96.41	1.55	0.51	95.46	1.32	0.83
b18	49293	128	90.25	2.54	0.29	89.36	2.96	0.80
b19	98726	128	89.56	3.06	0.45	88.11	3.55	0.95

for *RLL*. For *c5315* benchmark, 128 key gates are inserted randomly in the netlist with 2307 logic gates. The μ of success rate *SR* is 88.56%, and the σ is 2.59%, which means that the confidence of 99.7% (for $\pm 3\sigma$) can be observed in the range from 80.79% to 96.33%. A 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, our proposed attack provides better accuracy (average of 99.38%) 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 our proposed attack as an alternative to the SAT attacks.

Column 6 shows the mean (or standards deviation) *MR* of each benchmark circuit, calculated using Equation 4. Note that the mean and standards deviation are of the same value for an exponential distribution. The average *MR* is less than 1% for most benchmark circuits, which makes our attack very useful for determining the secret key. Table 1 also presents the *SR* and *MR* metrics for the same benchmark circuits locked with *SLL*, and shown in Columns 7 to 9. We also observe a similar trend like *RLL*.

In order to evaluate the accuracy of *SR* and *MR* with the circuit size, scatter plots of *SR* and *MR* are performed and shown in Figure 8. A least-squares trend line is added in every plot, while m represents the line's slope. In Figure 8.(a), we observe a slope m of 9.60, which indicates that the value of *SR* increases with the circuit size increase. A

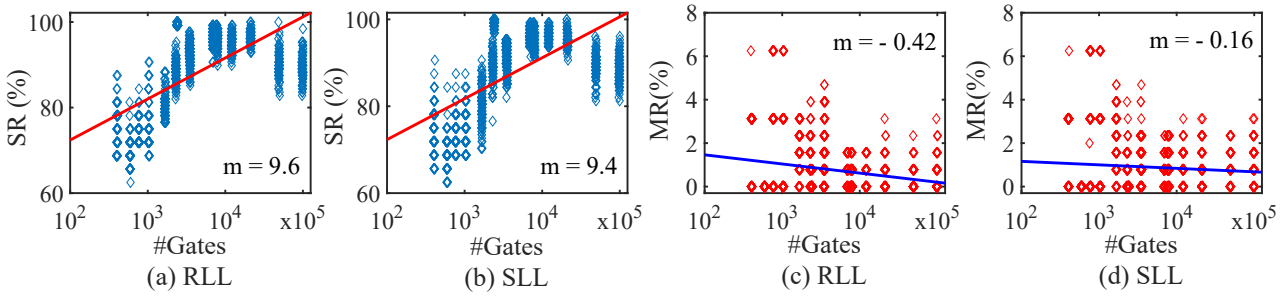

Fig. 8: Scatter plots of *SR* and *MR* versus number of gates on *RLL* and *SLL* benchmark circuits. (a) *SR* for *RLL* circuits, (b) *SR* for *SLL* circuits, (c) *MR* for *RLL* circuits, and (d) *MR* for *SLL* circuits.

similar trend is observed for the *SLL* locked circuits, and shown in Figure 8.(b). As for the *MR* shown in Figure 8.(c), we observe a negative slope of 0.42. A similar trend is found for *SLL* locked circuits and shown in Figure 8.(d). From this observation, we can conclude that the misprediction rate decreases with the increase of the circuit size. Overall, the accuracy of the proposed *TGA* increases for larger circuits.

Table 2: *SR* and *MR* for estimating keys for locked circuits from Trust-Hub.

Benchmark	# Total Gates	# Key Inputs	***SR*(%)**	***MR*(%)**
c880-SL320	404	32	87.50	3.13
c1350-SL320	593	32	78.13	0.00
c1908-SL320	768	32	84.38	3.13
c2670-SL320	1042	32	84.38	3.13
c3540-SL640	1546	64	82.81	1.56
c5315-SL640	2090	64	87.50	1.56
c6288-SL1280	2603	128	96.88	0.00
c7552-SL1280	3173	128	88.28	0.78

To reinforce our conclusion from Table 1, we also selected 8 different benchmark circuits from trust-Hub [38] and performed our topology-guided attack to evaluate the effectiveness. Table 2 presents the obtained results for the same. The selected benchmark is noted in Column 1 with the corresponding number of logic gates in the circuit shown in Column 2. Columns 3 presents the number of key inputs instead of key gates for each benchmark circuits as one key input may be fed into multiple key gates. The resultant *SR* and *MR* are concluded in Columns 4 and 5. For c3540-SL640, 64 key inputs are inserted in the circuit. The *SR* is 82.81%, which depicts 53 key inputs can be predicted with correct values. When comparing the results, 1 incorrect prediction is found, which produces a misprediction rate *MR* of 1.56%. As for the c6288-SL1280 benchmark circuit, 128 key inputs are inserted to protect the circuit. The *SR* of 96.88% can be observed which indicates that the majority of key inputs can be recovered based on our attack (125 key bits). The *MR* is 0.00%, which 0 key bit is mispredicted out of the entire 128 key inputs. We have emphasized *c6288* benchmark circuit as to present a clear comparison with the SAT attack, which was not efficient on this circuit.

Note that an adversary can recover the complete key from Table 1 with the help of an oracle (e.g., an unlocked chip). As the objective of logic locking is to modify the input-output response of a circuit, it produces incorrect responses for applying a wrong key. If an adversary finds out the key bit location (the unspecified, X, key bits from Algorithm 1, it is easy to determine its value by comparing it with an oracle). As there are few Xs, their permutations are limited and can easily be determined. Note that this could have a complex problem if we do not know the location of the wrong key bit(s). Then, the adversary needs to verify ${}^{|K|}C_N \times 2^N$, where N is the number of unspecified keys, and $|K|$ is the key size. These N unspecified key bits can come from any locations of the key K. On the contrary, we only need to verify 2^N cases to determine the complete key, which is a much simpler problem.

5.2 Complexity Analysis

SAT problem is an NP-complete problem, thus solving an SAT-resistant locking leads to an exponential worst-case complexity. However, our proposed topology-guided attack does not need to compare any input and output pairs, and all the inserted key gates are analyzed individually. Therefore, the time complexity of the attack itself is simply linear to the key size, namely, $O(|K|)$. Note that, our attack algorithm is based on *FS*, the actual overall complexity is $O(|K| * n * u)$ where n and u represent the size of the netlist and maximum 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 gates, depending on the key gate location. In Algorithm 2, once a key bit is predicted and written in the key list K_P, it will never be analyzed again as the value is recovered already. As a result, the computation complexity of launching the attack on *SLL* is the same as it is for *RLL*.

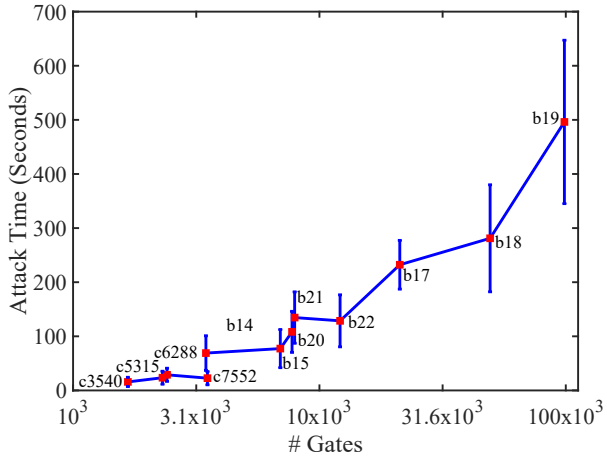

Fig. 9: Attack time for different *SLL* locked benchmark circuits with 128-bit keys . Each benchmark circuit is evaluated with 100 instances.

The attack time of the proposed *TGA* on different *SLL* locked circuits is illustrated in Figure 9. The plot shows the attack time versus the number gates for the 100 instances of each benchmark circuit (same circuits mentioned in Table 1). Note that we consider benchmark circuits inserted with 128 key gates only for the uniformity. The minimum, average, and maximum values of the attack time are displayed for each benchmark circuit. For example, the reported time for performing *TGA* on 128-bit *SLL* locked b17 benchmark circuit lies in a range from 178 seconds to 277 seconds with an average of 222.38 seconds. The location of inserted key gates causes this variation, and the corresponding number of *EUFs* searched during the attack. The x-axis is presented in the log scale to display all the different benchmarks clearly. The graph displays exponential behavior, and thus, it can be concluded that the attack time increases linearly with the increase of circuit size.

6 Conclusion

In this paper, we proposed a novel oracle-less topology-guided attack that is based on unit function search. Due to the repetitive usage of *UF* in a netlist, the key bits for a locked unit functions can be determined by constructing *EUFs* with hypothesis key bits and comparing them against the corresponding unlocked *UFs*. Compared to the traditional SAT-based attacks, the proposed topology-guided attack does not require input/output pairs or an activated chip. Moreover, SAT resistant countermeasures cannot prevent an adversary from launching this attack. To demonstrate the success of this attack, we presented the results on different benchmark circuits locked with random logic locking and strong logic locking techniques. We also validated our proposed attack on existing locked benchmark circuits from the trust-Hub. The success rate and misprediction rate metrics are proposed to evaluate the effectiveness of this attack. It is important to emphasize on the complexity of this attack which is linear with the key size on both *RLL* and *SLL*, which makes it very effective for circuits with larger key sizes. A countermeasure is also proposed as a solution to prevent this topology-guided attack. The basic idea is to insert the key gate in a unique unit function or lock all the instances repeated in the netlist. Note that this solution can only be used to prevent this topology-guided attack. To design a secure logic locking technique, one needs to select an existing secure logic locking technique along with our proposed solution.

Acknowledgement

This work was supported by the National Science Foundation under grant number CNS-1755733. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. UF/FICS Hardware De-obfuscation Competition (2019) Available: https://trust-hub.org/competitions/hwobfuscation1
2. Trusted and Assured Micro Electronics Forum (2019). Available: https://www.tameforum.org/
3. Abboud, A., Backurs, A., Hansen, T.D., Vassilevska Williams, V., Zamir, O.: Subtree isomorphism revisited. ACM Transactions on Algorithms (TALG) **14**(3), 27 (2018)
4. Alkabani, Y.M., Koushanfar, F.: Active hardware metering for intellectual property protection and security. In: Proc. of USENIX Security Symposium, pp. 20:1–20:16 (2007)
5. Alrahis, L., Yasin, M., Limaye, N., Saleh, H., Mohammad, B., Alqutayri, M., Sinanoglu, O.: ScanSAT: Unlocking Static and Dynamic Scan Obfuscation. Transactions on Emerging Topics in Computing (2019)
6. Baumgarten, A., Tyagi, A., Zambreno, J.: Preventing IC piracy using reconfigurable logic barriers. IEEE Design & Test of Computers pp. 66–75 (2010)
7. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann (2018)
8. Bryan, D.: The ISCAS'85 benchmark circuits and netlist format. North Carolina State University **25** (1985)
9. Castillo, E., Meyer-Baese, U., García, A., Parrilla, L., Lloris, A.: IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores. IEEE Trans. Very Large Scale Integr. Syst. pp. 578–591 (2007)
10. Chakraborty, R., Bhunia, S.: Hardware protection and authentication through netlist level obfuscation. In: Proc. of IEEE/ACM International Conference on Computer-Aided Design, pp. 674 –677 (2008)
11. Chakraborty, R.S., Bhunia, S.: HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems pp. 1493–1502 (2009)
12. Charbon, E.: Hierarchical watermarking in IC design. In: Custom Integrated Circuits Conference, pp. 295–298 (1998)

13. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: Genunlock: An automated genetic algorithm framework for unlocking logic encryption. In: ICCAD, pp. 1–8 (2019)
14. Chiang, H.Y., Chen, Y.C., Ji, D.X., Yang, X.M., Lin, C.C., Wang, C.Y.: LOOPLock: LOgic OPtimization based Cyclic Logic Locking. Transactions on Computer-Aided Design of Integrated Circuits and Systems (2019)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2009)
16. Davidson, S.: Itc'99 benchmark circuits-preliminary results. In: International Test Conference 1999. Proceedings (IEEE Cat. No. 99CH37034), pp. 1125–1125. IEEE Computer Society (1999)
17. Dickinson, P.J., Bunke, H., Dadej, A., Kraetzl, M.: On graphs with unique node labels. In: International Workshop on Graph-Based Representations in Pattern Recognition, pp. 13–23. Springer (2003)
18. Guin, U., Shi, Q., Forte, D., Tehranipoor, M.M.: FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. ACM Transactions on Design Automation of Electronic Systems (2016)
19. Guin, U., Zhou, Z., Singh, A.: A novel design-for-security (DFS) architecture to prevent unauthorized IC overproduction. In: Proc. of the IEEE VLSI Test Symposium (VTS), pp. 1–6 (2017)
20. Guin, U., Zhou, Z., Singh, A.: Robust design-for-security architecture for enabling trust in IC manufacturing and test. IEEE Transactions on Very Large Scale Integration (VLSI) Systems pp. 818–830 (2018)
21. Jarvis, R.W., McIntyre, M.G.: Split manufacturing method for advanced semiconductor circuits (2007). US Patent 7,195,931
22. Juretus, K., Savidis, I.: Increasing the SAT Attack Resiliency of In-Cone Logic Locking. In: International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2019)
23. Kahng, A., Lach, J., Mangione-Smith, W., Mantik, S., Markov, I., Potkonjak, M., Tucker, P., Wang, H., Wolfe, G.: Constraint-based watermarking techniques for design IP protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems pp. 1236–1252 (2001)
24. Karmakar, R., Chatopadhyay, S., Kapur, R.: Encrypt flip-flop: A novel logic encryption technique for sequential circuits. arXiv preprint arXiv:1801.04961 (2018)
25. Khaleghi, S., Da Zhao, K., Rao, W.: IC piracy prevention via design withholding and entanglement. In: The 20th Asia and South Pacific Design Automation Conference, pp. 821–826 (2015)
26. Koushanfar, F., Qu, G.: Hardware metering. In: Proc. IEEE-ACM Design Automation Conference, pp. 490–493 (2001). DOI 10.1109/DAC.2001.156189
27. Lee, Y.W., Touba, N.A.: Improving logic obfuscation via logic cone analysis. In: Latin-American Test Symposium (LATS), pp. 1–6 (2015)
28. Limaye, N., Sengupta, A., Nabeel, M., Sinanoglu, O.: Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access. arXiv preprint arXiv:1906.07806 (2019)
29. Liu, B., Wang, B.: Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In: Proceedings of the conference on Design, Automation & Test in Europe, p. 243 (2014)
30. Plaza, S.M., Markov, I.L.: Solving the third-shift problem in IC piracy with test-aware logic locking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems pp. 961–971 (2015)
31. Potluri, S., Aysu, A., Kumar, A.: Seql: Secure scan-locking for ip protection. arXiv preprint arXiv:2005.13032 (2020)
32. Python-2.7: Available: https://www.python.org/download/releases/2.7/ (2019)
33. Qu, G., Potkonjak, M.: Intellectual property protection in VLSI designs: theory and practice. Springer Science & Business Media (2003)
34. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Security analysis of logic obfuscation. In: Proc. of ACM/IEEE on Design Automation Conference, pp. 83–89 (2012)
35. Rajendran, J., Zhang, H., Zhang, C., Rose, G.S., Pino, Y., Sinanoglu, O., Karri, R.: Fault analysis-based logic encryption. IEEE Transactions on Computers pp. 410–424 (2015)
36. Reich, A.J., Nakagawa, K.H., Boone, R.E.: OASIS vs. GDSII stream format efficiency. In: 23rd Annual BACUS Symposium on Photomask Technology, vol. 5256, pp. 163–174 (2003)
37. Roy, J., Koushanfar, F., Markov, I.: EPIC: Ending Piracy of Integrated Circuits. In: DATE, pp. 1069 –1074 (2008). DOI 10.1109/DATE.2008.4484823
38. Salmani, H., Tehranipoor, M.: Trust-hub (2018). [Online]. Available: https://trust-hub.org/home
39. Sengupta, A., Nabeel, M., Limaye, N., Ashraf, M., Sinanoglu, O.: Truly stripping functionality for logic locking: A fault-based perspective. Transactions on Computer-Aided Design of Integrated Circuits and Systems (2020)
40. Shakya, B., Xu, X., Tehranipoor, M., Forte, D.: Cas-lock: A security-corruptibility trade-off resilient logic locking scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 175–202 (2020)
41. Shakya, B., Xu, X., Tehranipoor, M., Forte, D.: Defeating cas-unlock. IACR Cryptol. ePrint Arch. **2020**, 324 (2020)
42. Shamsi, K., Li, M., Meade, T., Zhao, Z., Pan, D.Z., Jin, Y.: AppSAT: Approximately deobfuscating integrated circuits. In: Int. Symp. on Hardware Oriented Security and Trust (2017)
43. Shamsi, K., Li, M., Pan, D.Z., Jin, Y.: Kc2: Key-condition crunching for fast sequential circuit deobfuscation. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 534–539. IEEE (2019)
44. Shamsi, K., Li, M., Plaks, K., Fazzari, S., Pan, D.Z., Jin, Y.: IP Protection and Supply Chain Security through Logic Obfuscation: A Systematic Overview. Trans. on Design Automation of Electronic Systems (TODAES) p. 65 (2019)
45. Shen, Y., Zhou, H.: Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In: Proceedings of the on Great Lakes Symposium on VLSI, pp. 179–184 (2017)
46. Sirone, D., Subramanyan, P.: Functional analysis attacks on logic locking. IEEE Transactions on Information Forensics and Security **15**, 2514–2527 (2020)
47. Subramanyan, P., Ray, S., Malik, S.: Evaluating the security of logic encryption algorithms. In: Int. Symp. on Hardware Oriented Security and Trust, pp. 137–143 (2015)
48. Tan, Q., Potluri, S., Aysu, A.: Efficacy of satisfiability based attacks in the presence of circuit reverse engineering errors. arXiv preprint arXiv:2005.13048 (2020)
49. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM journal on computing pp. 146–160 (1972)
50. Tehranipoor, M.M., Guin, U., Forte, D.: Counterfeit Integrated Circuits: Detection and Avoidance. Springer (2015)
51. Torrance, R., James, D.: The state-of-the-art in IC reverse engineering. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 363–381 (2009)
52. Vaidyanathan, K., Liu, R., Sumbul, E., Zhu, Q., Franchetti, F., Pileggi, L.: Efficient and secure intellectual property (IP) design with split fabrication. In: Int. Symp. on Hardware Oriented Security and Trust, pp. 13–18 (2014)
53. Wang, X., Zhang, D., He, M., Su, D., Tehranipoor, M.: Secure scan and test using obfuscation throughout supply chain. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(9), 1867–1880 (2017)
54. Yasin, M., Rajendran, J.J., Sinanoglu, O., Karri, R.: On improving the security of logic locking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**(9), 1411–1424 (2016)