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

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

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 [3,6,8,9,16,47]. Many design-for-trust techniques have been studied over the years as countermeasures against the aforementioned threats [3,11,16,19,21,24,31,32,35,49].

Amongst the many, logic locking is the most widely accepted and studied design-for-trust technique to prevent threats originated from untrusted manufacturing and test. Logic locking hides the inner details of the circuit 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 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 [16,18,32,35], (ii) MUX-based [25,28,33], (iii) LUT-based [5,23,27], and (iv) state-space based [10]. However, XOR-based logic locking is popular due to its simplicity.
A novel oracle-less topology-guided attack on logic locking: We propose 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 $UF$s are constructed in few layers of gates, the number of key gates and key bits associated with a $UF$ is limited resulting in very small $EUF$ search combinations. The results in Table 1 shows the efficiency of the proposed attack by recovering 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 [36] (see Table 5). In contrast with the traditional oracle (unlocked chip) attacks, no oracle is required to launch our proposed attack.

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 won’t reach a decision about the actual value of the key bit by comparing with its unlocked version. To achieve this, DFS-based $EUF$ 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.

\[ \text{Y} = \overline{X_3} + X_4 = \overline{X_3} \cdot \overline{X_4} = X_3 \cdot (1 \oplus X_4) = X_3 \cdot X_4 \quad (2) \]

Note that only XOR gates are used in the example to lock the netlist. However, one can also use XNOR gates for such purposes, 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 easily determine the secret key just by simply 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 oracle with scan access. For the same, we present the different steps involved in performing the proposed attack.

3.1 Adversarial Model

The unambiguous objective of an attacker 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 the access to the gate-level netlist of a locked IC. The SoC designers typically send the circuit layout information using GDSII or OASIS files 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.
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 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 UF) 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 1 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 or 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 the random logic locking (RLL), the secure key gates are inserted randomly inside the circuit that needs to be protected. In the large designs 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 illustrates the construction of the 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 3(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 3(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) \( EUF_1 \) (Case-II) where the hypothesis key \( k_h = 1 \) but the modification is carried out using DeMorgan’s Theorem. As a result, we can generalize and say that the number of EUFs for a given UF equals to \( 3^j \), where \( j \) represents the number of key gates included in the unit functions, which will be used in strong logic locking discussed in the following.

### 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 [32]. 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 equivalent unit function search with multiple keys instead of a single one for random logic locking.

![Figure 5](image_url)

**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) \( EUF_{110} \) for \( k_h = 011 \). (e) \( EUF_{010} \) for \( k_h = 010 \). (f) \( EUF_{101} \) 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 UF. As mentioned earlier, each key gate results 3 different EUFs (e.g., \( EUF_0 \), \( EUF_1 \), and \( EUF_1 \) 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, \( EUF_{101} \) in Figure 5(c) is constructed with hypothesis key bits based transformation as \( EUF_1 \), \( EUF_0 \) and \( EUF_0 \) for key gates \( G_7 \), \( G_8 \) and \( G_9 \) 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 \( EUF_{010} \), where the hypothesis key is 0, 1 (Case-II) and 0. Likewise, if we select the transformation as \( EUF_1 \), \( EUF_0 \) and \( EUF_1 \) (Case-II) for key gates \( G_7 \), \( G_6 \) and \( G_9 \) respectively, then we will get the \( EUF_{101} \) 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 Unit 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 [36]. 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 [13]. The use of a dictionary makes the search for specific type of gates more efficient.

### 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 (LR)

1. Read C and UF, and transform them into dictionaries, O and T;
2. R ← UF.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
4. end
5. return L_R;

**Function DFS(r, g):**

1. F ← True;
2. L_1 ← r.PrecedingGates; L_2 ← g.PrecedingGates;
3. T_1 ← L_1.types; T_2 ← L_1.types;
4. if L_1 is empty then
   5. return True;
5. end
6. for each gate type T in T_1 do
5. if gate type T not in T_2 then
6. T_2.remove(T);
7. end
8. end
9. for each gate R_N in L_1 do
   10. L_T ← φ;
   11. for each gate G_T in L_2 do
      12. if G_T.type = R_N.type then
         13. L_T.append(G_T);
      14. end
   15. end
   16. F_T ← False;
   17. for each gate G_N in L_T do
      18. if DFS(R_N, G_N) then
         19. F_T ← True;
         20. break
      21. end
   22. end
   23. if F_T = True then
      24. F ← F * F_T;
   25. end
   26. end
   27. return F

The procedure of DFS-based search is described in Algorithm 1. 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. Whenever 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 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 UF 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 [30]. The worst case time complexity of the search algorithm is $O(n \ast u)$, where $n$ is the size of 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 [13][33]. 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 sub-quadratic [2]. 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 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 equivalent unit function search (UFS). 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 an 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 $2^{1}$. 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 $3^{3}$ EUFs will be generated. The unit function search (UFS) (described in section 3.3) is then performed to search the repeated instances of EUFs (Line 12-14). $2^{1}$ 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 hypothesis key $J'$ of this EUF will be written into $K_P$, and 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, additional process needs to be performed (Line 19-21). Function $FV(\cdot)$ 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 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 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).

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

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 correct key list $K_P$ and $SR$ (Line 27).

The proposed attack may also cause the 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:

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

where, $p_i$ represents 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 UF and insert the key gate(s) in all unique UF in the circuit simultaneously. As a result, the adversary cannot predict and recover the correct key values by comparing the locked UF 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: Insertion of key gates to prevent topology-guided attack

\begin{verbatim}
Input : Gate level netlist of a circuit \((C)\).
Key size \((k_{min}, k_{max})\).
Output: Locked netlist \((C^*)\) and Key value \((K^*)\)

1. Initialization: \(n \leftarrow 0, r \leftarrow 0;\)
2. While \(n < k_{min}\) do
3. Select a root gate randomly from \(C\);
4. Construct the unit function, \(UF\);
5. \(r \leftarrow UF.SLS(C, UF.sz());\)
6. If \(RLL\) then
7. \(\quad\) If \(r = 1\) then
8. \(\quad\) Insert the key gate at one random input of root
9. \(\quad\) gate and assign key value, \(k_n \in \{0, 1\};\)
10. \(\quad\) Write key value, \(K^* [n] \leftarrow k_n;\)
11. \(\quad\) \(n \leftarrow n + 1;\)
12. Else if \(1 < r \leq k_{max} - n\) then
13. \(\quad\) Lock all the UF;
14. \(\quad\) Write key values to \(K^* [n + r : n];\)
15. \(n \leftarrow n + r;\)
16. Else if \(SLL\) then
17. \(\quad\) If \(r = 1\) then
18. \(\quad\) Insert \(j\) key bits in the unique \(UF\), and assign key
19. \(\quad\) values, \(k_n, k_{n+1}, \ldots, k_{n+j};\)
20. \(\quad\) Write key value, \(K^* [n + j : n] \leftarrow [k_{n+j}, \ldots, k_{n+1}, k_n];\)
21. \(\quad\) \(n \leftarrow n + j;\)
22. Else if \(1 < r \leq k_{max} - n\) then
23. \(\quad\) Lock all the UF;
24. \(\quad\) Write key values to \(K^* [(n + r * j) : n];\)
25. \(n \leftarrow n + r * j;\)
26. End
27. End
28. Output \(C^*\) and \(K^*\);
\end{verbatim}

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 \((k_{min}, k_{max})\), 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 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 for the search later (Lines 4). Next, \(UF.SLS(C, UF.sz())\) 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 \(UF\) 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 \(SLL\) in the circuit (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 \(r * j\).

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 [7] and ITC’99 [14] benchmark circuits locked with \(RLL\) and \(SLL\) using our in-house script. In addition, we
have validated our proposed attack on TrustHub benchmark circuits.

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 variance are presented by μ and σ² for Gaussian distributions related to SR, whereas they are represented by λ⁻¹ and λ⁻² 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 σ² of 6.7018. 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 3.2613 and 2.4179 respectively. We observe a similar Gaussian distributions for the SR on locked circuits using SLL (see Figure 6(e)-(h)). Note that the overall variance of the SR distribution is decreased when increasing the size of the benchmark circuits due to the increased EUF search space in the circuit, which makes our proposed attack more effective for extracting key value in large designs.

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 (λ⁻¹) of 1.23% and variance (λ⁻²) of 1.5129. As observed from Figure 7(c) and Figure 7(b15-RLL shows λ⁻¹ of 0.48% and λ⁻² of 0.2304, whereas b17-RLL shows λ⁻¹ of 0.51% and λ⁻² of 0.2601. Likewise, a similar analysis can be done for MR for the same selected benchmark circuits locked with SLL plotted in Figure 7(e)–(h). In general, both mean and variance of MR are decreased with the increase in the size of the benchmark circuits, which makes this attack more accurate for larger designs.

Table 1 shows the success rate (SR) and misprediction rate (MR) of our proposed attack on different ISCAS’85 and ITC’99 benchmark circuits locked with random logic locking. The number of logic gates in the circuit 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% such that 128 key gates are inserted randomly. 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 5).
by analyzing 100 locked instances for each benchmark circuit to determine the accuracy of proposed topology-guided attack (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 85.93% is observed, where the attack identifies 125 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 topology-guided attack 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 alternate of SAT attacks.

It is also necessary to evaluate the correctness of the calculated SR, so the accuracy of the attack is concluded as well. 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 7) for MR. The average MR is less than 1% for majority of benchmark circuits, which makes our attack 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).

In Table 2 we evaluated the same benchmark circuits but locked with SLL to evaluate SR and MR of our proposed topology-guided attack. For each benchmark is shown in Column 1; 100 different locked instances are implemented with 128 key gates insertion; and the results of SR and MR are concluded in Columns 4 to 9 with minimum, average and maximum rate. For b14 benchmark, the minimum SR of
Table 2: SR and MR for estimating keys for SLL circuits.

| Benchmark | # Total Gates | # Key Gates | Success Rate (SR) | Misprediction Rate (MR) |
|-----------|---------------|-------------|-------------------|------------------------|
|           |               |             | Min. | Avg. | Max. | Min. | Avg. | Max. |
| c3450     | 1669          | 128         | 80.31% | 90.63% | 96.88% | 0.00% | 2.01% | 4.68% |
| c6288     | 2406          | 128         | 90.21% | 95.31% | 99.25% | 0.00% | 0.00% | 0.00% |
| c7552     | 3512          | 128         | 90.25% | 100.00% | 99.25% | 0.00% | 0.97% | 4.69% |
| b14       | 3461          | 128         | 93.67% | 95.31% | 96.88% | 0.00% | 0.92% | 2.34% |
| b15       | 6931          | 128         | 94.50% | 99.22% | 96.19% | 0.00% | 0.59% | 2.34% |
| b20       | 7741          | 128         | 94.50% | 99.22% | 96.19% | 0.00% | 0.84% | 2.34% |
| b21       | 7931          | 128         | 94.50% | 99.22% | 96.19% | 0.00% | 0.63% | 2.34% |
| b22       | 12128         | 128         | 95.46% | 100.00% | 95.78% | 0.00% | 0.77% | 3.12% |
| b17       | 21191         | 128         | 90.62% | 95.46% | 95.78% | 0.00% | 0.83% | 3.12% |
| b18       | 49293         | 128         | 96.09% | 96.09% | 95.46% | 0.00% | 0.80% | 2.34% |
| b19       | 98726         | 128         | 96.01% | 96.01% | 95.46% | 0.00% | 0.95% | 3.12% |

Table 3: 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% | 3.12% |
| c1350-SL320| 593          | 32           | 78.13% | 0.00% |
| c908-SL320| 768           | 32           | 84.38% | 0.12% |
| c2670-SL320| 1042         | 32           | 84.38% | 0.12% |
| c5340-SL640| 1546         | 64           | 82.81% | 1.56% |
| c5317-SL640| 2090         | 64           | 87.50% | 1.56% |
| c6288-SL1280| 2603         | 128          | 96.88% | 0.00% |
| c7552-SL1280| 3173         | 128          | 88.28% | 0.78% |

88.28% is observed, where our proposed attack can at least predicts 113 out of 128 key value correctly. Also, 95.31% is attained as the maximum SR, where the attack can identify 122 key bits successfully. As for the MR, a maximum value 2.34% can be observed that depicts 3 bits were predicted incorrectly. Similar analysis can be studied on all the benchmark listed in the table. Following the earlier trend, as the size of the benchmark circuit increases, the overall average SR increases with lower MR. The overall performance evaluation for the same benchmark is similar to the result shown in Table 1. As a result, our proposed topology-guided attack can be performed on the SLL efficiently as well.

To reinforce our conclusion from Table 1 and Table 2, we also selected 8 different benchmark circuits from trust-Hub [36] and performed our topology-guided attack to evaluate the effectiveness. Table 3 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 c3450-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 inputs). 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.

5.2 Complexity Analysis

SAT problem is a NP-complete problem, thus solving a 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 UF, 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 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.

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 UF. Compared to the traditional SAT-based attacks, the proposed topology-guided at-
tack 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, https://trust-hub.org/competitions/hwobfuscation1, 2019
2. Abboud, A., Backurs, A., Hansen, T.D., Vassilevska Williams, V., Zamir, O.: Subtree isomorphism revisited. ACM Transactions on Algorithms (TALG) 14(3), 27 (2018)
3. 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)
4. Alrahis, L., Yasin, M., Limaye, N., Saleh, H., Mohammad, B., Alquatayri, M., Sinanoglu, O.: ScanSAT: Unlocking Static and Dynamic Scan Obfuscation. Transactions on Emerging Topics in Computing (2019)
5. Baumgarten, A., Tyagi, A., Zambreno, J.: Preventing IC piracy using reconfigurable logic barriers. IEEE Design & Test of Computers pp. 66–75 (2010)
6. Bhunia, S., Tehaniipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann (2018)
7. Bryan, D.: The ISCAS’85 benchmark circuits and netlist format. North Carolina State University 25 (1985)
8. 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)
9. 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)
10. 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)
11. Charbon, E.: Hierarchical watermarking in IC design. In: Custom Integrated Circuits Conference, pp. 295–298 (1998)
12. Chiang, H.Y., Chen, Y.C., Ji, D.X., Yang, X.M., Lin, C.C., Wang, C.Y.: LOOPLock: Low OpTimization based Cyclic Logic Locking. Transactions on Computer-Aided Design of Integrated Circuits and Systems (2019)
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2009)
14. Davidson, S.: (2019). https://www.cerc.utexas.edu/itc99-benchmarks/bench.html
15. 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)
16. Guin, U., Shi, Q., Forte, D., Tehaniipoor, M.M.: FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. ACM Transactions on Design Automation of Electronic Systems (2016)
17. 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)
18. 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)
19. Jarvis, R.W., McIntyre, M.G.: Split manufacturing method for advanced semiconductor circuits (2007). US Patent 7,195,931
20. 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)
21. 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)
22. Karmakar, R., Chatopadhyay, S., Kapur, R.: Encrypt flip-flop: A novel logic encryption technique for sequential circuits. arXiv preprint arXiv:1801.04961 (2018)
23. 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)
24. Koushanfar, F., Qu, G.: Hardware metering. In: Proc. IEEE-ACM Design Automation Conference, pp. 490–493 (2001). DOI 10. 1109/DAC.2001.156189
25. Lee, Y.W., Toubia, N.A.: Improving logic obfuscation via logic cone analysis. In: Latin-American Test Symposium (LATS), pp. 1–6 (2015)
26. 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)
27. 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)
28. 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)
29. Potluri, S., Kumar, A., Aysu, A.: SeqL: SAT-attack Resilient Sequential Locking (2020)
30. Python-2.7: https://www.python.org/download/releases/2.7/ (2019)
31. Qu, G., Potkonjak, M.: Intellectual property protection in VLSI designs: theory and practice. Springer Science & Business Media (2003)

32. 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)

33. 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)

34. 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)

35. Roy, J., Koushanfar, F., Markov, I.: EPIC: Ending Piracy of Integrated Circuits. In: DATE, pp. 1069 –1074 (2008). DOI 10.1109/DATE.2008.4484823

36. Salmani, H., Tehranipoor, M.: Trust-hub (2018). [Online]. Available: https://trust-hub.org/home

37. 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)

38. Shakya, B., Xu, X., Tehranipoor, M., Forte, D.: Defeating cas-unlock

39. Shakya, B., Xu, X., Tehranipoor, M., Forte, D.: Cas-lock: A security-corrupibility trade-off resilient logic locking scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 175–202 (2020)

40. 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)

41. 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)

42. 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)

43. 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)

44. Sirone, D., Subramanyan, P.: Functional analysis attacks on logic locking. IEEE Transactions on Information Forensics and Security 15, 2514–2527 (2020)

45. 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)

46. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM journal on computing pp. 146–160 (1972)

47. Tehranipoor, M.M., Guin, U., Forte, D.: Counterfeit Integrated Circuits: Detection and Avoidance. Springer (2015)

48. 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)

49. 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)

50. 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)