Expanding In-Cone Obfuscated Tree for Anti SAT Attack

RuiJie Wang*, Li-Nung Hsu*, Yung-Chih Chen†, TingTing Hwang*,
*Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
†Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan
wrj651121@gmail.com, kartd0109153@gmail.com, ycchen.ee@mail.ntust.edu.tw, tingting@cs.nthu.edu.tw

***Abstract*—Logic locking is a hardware security technology to protect circuit designs from overuse, piracy, and reverse engineering. It protects a circuit by inserting key gates to hide the circuit functionality, so that the circuit is functional only when a correct key is applied. In recent years, encrypting the point function, e.g., AND-tree, in a circuit has been shown to be promising to resist SAT attack. However, the encryption technique may suffer from two problems: First, the tree size may not be large enough to achieve desired security. Second, SAT attack could break the encryption in one iteration when it finds a specific input pattern, called remove-all DIP. Thus, in this paper, we present a new method for constructing the obfuscated tree. We first apply the sum-of-product transformation to find the largest AND-tree in a circuit, and then insert extra variables with the proposed split-compensate operation to further enlarge the AND-tree and mitigate the remove-all DIP issue. The experimental results show that the proposed obfuscated tree can effectively resist SAT attack.**

***Index Terms*—Hardware security, Logic locking, SAT attack, AND-Tree**

I. Introduction

Today's IC supply chain involves many secret information and data. The main concern to all parties is the silicon intellectual property (IP) protection. Due to the high R&D cost and complex chip manufacturing process, very few companies can provide a one-stop chip manufacturing solution, including wafer production, packaging and testing. Most IC design companies outsource their IC manufacturing to high-end foundries. Although such a business model is cost-effective, it also exposes IC design companies to the risks of IC/IP threats, such as unauthorized use, piracy, overproduction, reverse engineering, hardware Trojans, and counterfeiting. A report from the industry association Semiconductor Equipment and Materials International (SEMI) states that IP infringements cost the industry an estimated 4 billion dollars annually [1].

To avoid such hazardous situations, many countermeasures have been developed. **Logic locking** is one of the promising countermeasures [2], which encrypts a circuit by inserting key gates such that if the secret/correct key is not applied, the circuit cannot function normally. In early days, key gates were placed at the carefully-selected locations to prevent sensitization-based attacks, which cracked the secret key by propagating it to the circuit output. After the powerful SAT-based attack technique [3], called SAT attack, was proposed, many recent protection techniques aimed to defend against SAT attack.

SAT attack finds the correct key by iteratively pruning error keys with SAT solving. According to the defense principle, we can divide the protection techniques into two categories: 1) The SAT-hard techniques which exploit the characteristics of the SAT solver to increase the solving time of each iteration [4]–[6]. 2) The techniques which encrypt the *point function* to increase the number of required iterations by SAT attack [7]–[11].

A point function can be an *in-cone* or *out-of-cone* point function. The former originally exists in the circuit [7], [8], and the latter is newly added into the circuit [10], [11]. Both types of point functions are effective to mitigate SAT attack but have their own characteristics. For the in-cone point function, it may not be large enough to provide desired resistance against SAT attack. Additionally, there exists a specific input pattern, called remove-all DIP, which allows SAT attack to crack the secret key immediately. For the out-of-cone point function, although a sufficiently large function can be inserted, it is vulnerable to Removal attack [12]. An attacker can break the encryption by identifying the added structure and directly removing it without needing the secret key. Additionally, encrypting an out-of-cone point function also suffers from the remove-all DIP problem.

In order to defend against Removal attack, the corrupt-and-correct locking method was proposed by [13], [14]. It first corrupts the original function, and then adds an out-of-cone structure to restore the function. When the introduced structure is removed, the original function can never be restored. Thus, the secret key is required to break the encryption. However, the method also suffers from the remove-all DIP problem.

In this paper, we propose a new encryption method to construct an obfuscated AND-tree for defending against SAT attack and Removal attack simultaneously. We introduce an operation, called *split-compensate*, to expand the size of the in-cone AND-tree and reduce the number of remove-all DIPs. In the experiments, we applied the proposed method to a set of ISCAS'85 [15] and MCNC benchmarks [16]. The experimental results show that the split-compensate operation can successfully expand the size of the in-cone AND-tree and mitigate the remove-all DIP problem. Additionally, the encryption method is effective to resist SAT attack. 6 out of 7 encrypted benchmarks cannot be broken by SAT attack within a time limit of 3 hours.

The rest of this paper is organized as follows: Section II reviews some background on the point function-based logic encryption. Section III presents our motivation. Section IV introduces the relevant properties of an obfuscated AND-tree to support the proposed method. Section IV presents the proposed encryption method. Section V shows the experimental results. Finally, the conclusion is presented in Section VI.

II. PRELIMINARIES

Given a Boolean function $F : I \rightarrow O$, where $I = \{0,1\}^n$ and $O = \{0,1\}^m$, its obfuscated circuit has a Boolean function $F^{obfuscated} : I \times K \rightarrow O$, where K denotes the key inputs

TABLE I: Truth Table of 2-input obfuscated AND-tree, $T^{obfuscated} = (x_1 \oplus k_1)(x_2 \oplus k_2)$.

			$T^{obfuscated}$			
	x_1x_2	T	$\hat{k_1}$	$\hat{k_2}$	$\hat{k_3}$	$\hat{k_4}$
$\hat{in}_1$	00	0	0	0	0	**1**
$\hat{in}_2$	01	0	0	0	**1**	0
$\hat{in}_3$	10	0	0	**1**	0	0
$\hat{in}_4$	11	1	1	**0**	**0**	**0**

and $K = \{0,1\}^q$. A key $\hat{k_s} \in K$ is said to be a secret/correct key, if $F^{obfuscated}(\hat{in}, \hat{k_s}) = F(\hat{in})$ for all $\hat{in} \in I$.

A single-output Boolean function F is said to be a *one-point function*, if there exists only one input pattern evaluating F to 1. For example, an AND function is a one-point function. Previous work [9] has demonstrated that encrypting a circuit by obfuscating the existing AND-tree, i.e., an AND function, is effective to defend against SAT attack.

An n-input AND-tree T can be expressed as $T = \wedge_{i=1}^{n} x_i$. An n-input *obfuscated* AND-tree $T^{obfuscated}$ encrypted with XOR/XNOR gates can be represented as $T^{obfuscated} = \wedge_{i=1}^{n}(x_i \oplus / \odot k_i)$, where x_i and k_i denote the i^{th} tree input and key input signals, respectively. Both XOR and XNOR can be used as the key gates. For ease of explanation, we use only XOR as the key gate to describe the proposed method in the rest of this paper.

SAT attack [3] is an attack technique to break logic locking. It iteratively identifies a *distinguish input pattern (DIP)* to prune error keys with SAT solving. A DIP is an input pattern that can cause two different keys to generate different outputs. Thus, when a DIP is found, at least one key is incorrect and can be pruned. When all the error keys are pruned, the correct key can be cracked successfully.

Let us use an example to demonstrate why an obfuscated AND-tree is resistant to SAT attack. Table I shows the truth table of a 2-input obfuscated AND-tree, $T^{obfuscated} = (x_1 \oplus k_1)(x_2 \oplus k_2)$. The correct key is $\hat{k}_1$. In this example, all the input patterns, $\hat{in}_1 \sim \hat{in}_4$, can be DIPs. If SAT attack identifies a DIP in the ascending order, 4 iterations are required to prune all the error keys. For an obfuscated AND-tree with n inputs, the number of required iterations is $2^n - 1$.

However, in fact, $2^n - 1$ is the worst-case complexity, since there exists a special DIP, called *remove-all* DIP, which can cause SAT attack to prune all the error keys in one iteration. In the above example, $\hat{in}_4$ is the remove-all DIP. Since SAT attack finds a DIP arbitrarily, when n is large enough and there exist only few remove-all DIPs, the worst-case complexity can be used as a measure of security.

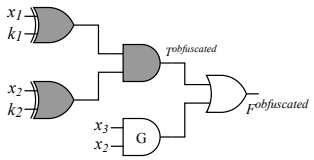
III. Motivation

The one-point function is an effective defense structure against SAT attack. However, the methods of obfuscating one-point functions may suffer from some issues. For the in-cone one-point function, its size, i.e., the number of inputs, may not be large enough for providing sufficient security. For the out-of-cone one-point function, the extra logic is introduced on a large scale. As a result, a re-synthesis process is needed to simplify and hide the added structure, which could affect the original design essence. Moreover, the remove-all DIP problem affects both of their security.

Table II shows the benchmarks, in which the size of the largest AND/OR-tree identified by the tree detection method [9] is less than 20. For these benchmarks, encrypting the AND/OR-tree could not provide sufficient resistance against SAT attack. Additionally, if an internal node of the tree has multiple fanout, the resistance would be greatly reduced, since the key effect can be propagated through the fanout.

TABLE II: Sizes of largest AND/OR-trees and numbers of remove-all DIPs.

Circuit	AND/OR-tree detection [9] Tree input size	#Remove-all DIPs
c1355	9	172
c5315	18	495616
ex5	9	14
i7	4	32
i9	15	416
apex2	9	109080
i4	7	18597788326912



(a) Example of an locked circuit.

			$F^{obfuscated}$			
	$x_1x_2x_3$	F	$\hat{k_1}$	$\hat{k_2}$	$\hat{k_3}$	$\hat{k_4}$
$\hat{in}_1$	000	0	0	0	0	**1**
$\hat{in}_2$	001	0	0	0	0	**1**
$\hat{in}_3$	010	0	0	0	**1**	0
$\hat{in}_4$	011	1	1	1	1	1
$\hat{in}_5$	100	0	0	**1**	0	0
$\hat{in}_6$	101	0	0	**1**	0	0
$\hat{in}_7$	110	1	1	**0**	**0**	**0**
$\hat{in}_8$	111	1	1	1	1	1

(b) Truth table of the locked circuit.

Fig. 1: Example of the equivalence class of an locked circuit.

Table II also shows the numbers of remove-all DIPs for the obfuscated largest AND/OR-tree. For some benchmarks, there exist a large number of remove-all DIPs, indicating that SAT attack may have a high probability of finding a remove-all DIP to prune all error keys in one iteration. Note that since a tree input may not be primary input (PI), the number of remove-all DIPs could be larger than the number of all possible input combinations of the tree inputs.

The mentioned issues motivate us to develop a new locking method for constructing an obfuscated tree, which can expand the in-cone AND-tree and reduce the impact of remove-all DIPs to improve security.

IV. Obfuscated AND-Tree Construction

In this section, we first introduce three basic definitions. Next, we present some propositions on the properties of an obfuscated AND-tree. Then, we discuss how to improve an obfuscated AND-tree to resist SAT attack. Finally, we present the proposed method.

A. Basic Definitions

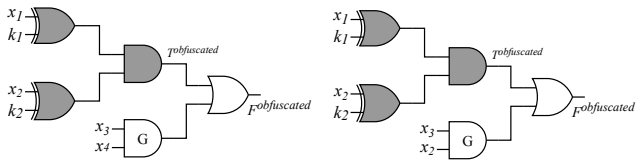
Definition 1. *(Error key set). Given a Boolean function F and its obfuscated function $F^{obfuscated}$, the error key set of an input pattern $\hat{in}$ is defined as:*

$$EK_{\hat{in}} = \{\hat{k_e} | F^{obfuscated}(\hat{in}, \hat{k_e}) \neq F(\hat{in})\}.$$

Definition 2. *(Equivalence class of error key sets). We say that two input patterns $\hat{in}_1$ and $\hat{in}_2$ are in the same equivalence class of error key sets, if and only if their error key sets are equivalent, i.e., $EK_{\hat{in}_1} = EK_{\hat{in}_2}$.*

Definition 3. *(Number of members in an equivalence class). Let $[\hat{in}]$ denote the equivalence class of error key sets with $\hat{in}$ as one of the members. The number of members in $[\hat{in}]$, denoted as $MEK([\hat{in}])$, is the cardinality of $[\hat{in}]$.*

Fig. 1(b) shows the truth table of the locked circuit in Fig. 1(a). $\hat{in}_1$ and $\hat{in}_2$ are in the same equivalent class, and $\hat{in}_5$ and $\hat{in}_6$ are in the same equivalence class.



(a) Disjoint. (b) Non-disjoint.

Fig. 2: Effect of an obfuscated AND-tree $T^{obfuscated}$ operating with a function G.

			$F^{obfuscated}$			
	$x_1x_2x_3x_4$	F	$\hat{k}_1$	$\hat{k}_2$	$\hat{k}_3$	$\hat{k}_4$
$\hat{in}_1$	0000	0	0	0	0	**1**
$\hat{in}_2$	0001	0	0	0	0	**1**
$\hat{in}_3$	0010	0	0	0	0	**1**
$\hat{in}_4$	0011	1	1	1	1	1
$\hat{in}_5$	0100	0	0	0	**1**	0
$\hat{in}_6$	0101	0	0	0	**1**	0
$\hat{in}_7$	0110	0	0	0	**1**	0
$\hat{in}_8$	0111	1	1	1	1	1
$\hat{in}_9$	1000	0	0	**1**	0	0
$\hat{in}_{10}$	1001	0	0	**1**	0	0
$\hat{in}_{11}$	1010	0	0	**1**	0	0
$\hat{in}_{12}$	1011	1	1	1	1	1
$\hat{in}_{13}$	1100	1	1	**0**	**0**	**0**
$\hat{in}_{14}$	1101	1	1	**0**	**0**	**0**
$\hat{in}_{15}$	1110	1	1	**0**	**0**	**0**
$\hat{in}_{16}$	1111	1	1	1	1	1

(a) Truth table of Fig. 2(a).

			$F^{obfuscated}$			
	$x_1x_2x_3$	F	$\hat{k}_1$	$\hat{k}_2$	$\hat{k}_3$	$\hat{k}_4$
$\hat{in}_1$	000	0	0	0	0	**1**
$\hat{in}_2$	001	0	0	0	0	**1**
$\hat{in}_3$	010	0	0	0	**1**	0
$\hat{in}_4$	011	1	1	1	1	1
$\hat{in}_5$	100	0	0	**1**	0	0
$\hat{in}_6$	101	0	0	**1**	0	0
$\hat{in}_7$	110	1	1	**0**	**0**	**0**
$\hat{in}_8$	111	1	1	1	1	1

(b) Truth table of Fig. 2(b).

Fig. 3: Truth tables of the circuits in Fig. 2.

Note that for an obfuscated function, the number of equivalence classes is the worst-case complexity and the number of members in an equivalence class of remove-all DIPs represents the number of remove-all DIPs. Thus, our encryption objective is to increase the number of equivalence classes and minimize the number of members in the equivalence class of remove-all DIPs.

B. Properties of an Obfuscated AND-Tree

In general, an obfuscated AND-tree could be randomly embedded in a circuit. In addition to the correlation between the tree inputs and the PIs, whether the effect of an obfuscated AND-tree can be propagated to a primary output (PO) affects the resistance to SAT attack as well. To analyze the security of an obfuscated AND-tree, we first present the following proposition.

Proposition 1. *For an n-input obfuscated AND-tree $T^{obfuscated}$, there exist 2^n equivalence classes of error key sets.*

Proof: Omitted. ■

For example, the 2-input obfuscated AND-tree shown in Table I has 4 equivalence classes.

Next, let us consider the effect of an obfuscated AND-tree operating with a subfunction G in a circuit, where G is an arbitrary function ORed with the obfuscated AND-tree. There should be another case, in which G and the obfuscated AND-tree are ANDed. However, in the proposed method, we can transform the circuit so that only OR gates are left on the path from the obfuscated AND-tree toward the PO.

For ease of explanation, in the following example, we assume that the obfuscated AND-tree $T^{obfuscated}$ has 2 inputs, the correct key is $\hat{k}_1 = 00$, and G is an AND gate. Fig. 2 shows different combinations of $T^{obfuscated}$ and G. They are different in whether the support sets of G and $T^{obfuscated}$ are disjoint or not.

The truth tables of the two circuits in Fig. 2 are shown in Table 3. First, let us consider the circuit in Fig. 2(a), in which the support sets are disjoint. $T^{obfuscated}$ can be propagated to the PO under the input patterns with $(x_3, x_4) = \{(0,0),(0,1),(1,0)\}$. According to the truth table in Fig. 3(a), the obfuscated circuit $F^{obfuscated}$ still has 4 equivalence classes and each class has 3 members. Next, for the circuit in Fig. 2(b), in which the support sets are non-disjoint, input patterns $(x_2, x_3) = \{(0,0),(0,1),(1,0)\}$ sets input of OR gate to non-controlling value. Thus, the obfuscated tree $T^{obfuscated}$ can be propagated to the output. The truth table in Fig. 3(b) shows that although $F^{obfuscated}$ still has 4 equivalence classes, there is only **one** remove-all DIP.

The example shows that the number of remove-all DIPs is affected by whether the support sets of $T^{obfuscated}$ and G are disjoint or not. Based on the above observation, we present the following proposition. Here, let $ON(F)$ and $OFF(F)$ denote the ON-set and the OFF-set of a Boolean function F, respectively.

Proposition 2. *Suppose that an obfuscated AND-tree $T^{obfuscated}$ is embedded in a circuit $F^{obfuscated}$, where $F^{obfuscated} = T^{obfuscated} \vee G$, and G is an arbitrary function. Then, for $F^{obfuscated}$, given a key $\hat{k}$ which makes $T^{obfuscated}(\hat{k}, \hat{in}_i) = 1$ for all $\hat{in}_i$ in an equivalence class $[\hat{in}]$, $[\hat{in}]$ can be computed with the following equation.*

$$[\hat{in}] = ON(T^{obfuscated}(\hat{k})) \cap OFF(G). \quad (1)$$

Proof: Omitted. ■

$\hat{k}$ is either a correct key or an error key. If $\hat{k}$ is the correct key, $[\hat{in}]$ is also the remove-all DIP set. For example, in Fig. 2(b), $ON(T^{obfuscated}(\hat{k_1}))$ is $\{110, 111\}$ and $OFF(G)$ is $\{000, 001, 010, 100, 101, 110\}$. $ON(T^{obfuscated}(\hat{k_1})) \cap OFF(G)$ is $\{110\}$, which is the equivalence class of remove-all DIPs, as shown in Figure 3(b). Furthermore, $ON(T^{obfuscated}(k_4))$ is $\{000, 001\}$. $ON(T^{obfuscated}(\hat{k_4})) \cap OFF(G)$ is $\{000, 001\}$. Either 000 or 001 can be a DIP to prune $\hat{k_4}$.

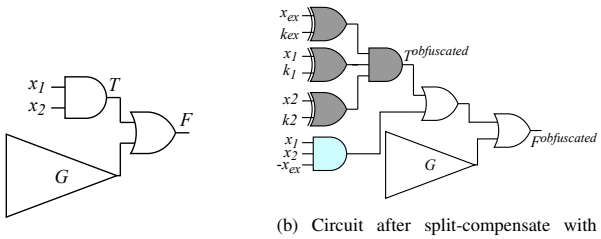
The following proposition states a method to measure the number of remove-all DIPs for $F^{obfuscated}$.

Proposition 3. *Given an obfuscated AND-tree $T^{obfuscated}$ embedded in a circuit $F^{obfuscated}$, where $F^{obfuscated} = T^{obfuscated} \vee G$ and G is an arbitrary function, the number of remove-all DIPs for $F^{obfuscated}$ is equivalent to the number of test patterns that can detect the stuck-at-0 fault at the output of $T^{obfuscated}(\hat{k_c})$, where $\hat{k_c}$ is the correct key.*

Proof: Omitted. ■

We note that previous tree-based encryption methods may have poor resistance to SAT attack due to the small size of the in-cone AND-tree and the large number of remove-all DIPs. Thus, we propose an operation, called *split-compensate*, to expand the AND-tree and reduce the number of remove-all DIPs with extra variables.

Definition 4. *(split-compensate). Given an AND-tree T embedded in a circuit F, where $F = T \vee G$ and G is an arbitrary function, the operation* **split** *performs Shannon's expansion on*



(a) Original circuit.

(b) Circuit after split-compensate with x_{ex} and encryption.

Fig. 4: Example of split-compensate.

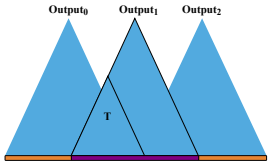


Fig. 5: External variables and internal variables.

an extra variable x_{ex} to T and selects one product term for encryption, and the operation **compensate** *inserts the other product terms after the split operation into F to recover the function.*

Fig. 4 shows an example. We add X_{ex} as an extra variable to T with Shannon's expansion, $T = x_{ex} \cdot T + x_{ex}{}' \cdot T = x_{ex}x_1x_2 + x_{ex}{}' x_1x_2$. Then, we encrypt the product term $(x_{ex}x_1x_2)$ to be the obfuscated AND-tree and insert the product term $(x_{ex}{}' x_1x_2)$ to recover the function. Note that the split-compensate operation can be repeatedly conducted to expand $T^{obfuscated}$ by adding more extra variables.

We can use extra variables to enlarge the obfuscated AND-tree. However, the extra variables may correlate with G. We classify the variables into *external variables* and *internal variables*, and analyze their effect in the following subsections.

C. Expanding Obfuscated AND-Tree with External Variables

Definition 5. *(External variable). For an AND-tree T, the external variable set of T, denoted as $external(T)$, is defined as:*

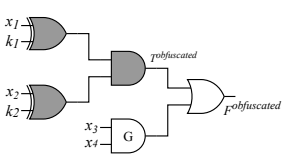
$$external(T) = I - SUP(O_{io}).$$

where I denotes the PI set, O_{io} denotes the set of POs in the transitive fanout cone of T, and $SUP(O_{io})$ denotes the support set of O_{io}.

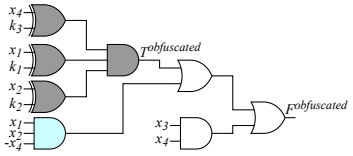
Fig. 5 shows an example of external variables, where $external(T)$ are highlighted with the orange color.

The input size of an obfuscated AND-tree can be increased by q after introducing q external variables. When we introduce q external variables through the split-compensate operation, we insert a new circuit structure, which can be seen as G, into $F_{obfuscated}$ for compensating the function. As discussed in the examples in Fig. 2, G may affect the effect propagation of the obfuscated AND-tree. The following proposition states the number of equivalence classes we can obtain, when we introduce q external variables through the split-compensate operation.

Proposition 4. *Given an AND-tree T embedded in a circuit F, where $F = T \vee G$ and G is an arbitrary function. Suppose that if we encrypt T directly without the split-compensate operation, the number of equivalence classes of $F^{obfuscated}$ is $const_G$ less than that of $T^{obfuscated}$ due to the influence of G. Then, when we perform the split-compensate operation to encrypt T with q external variables, the number of equivalence classes of $F^{obfuscated}$ has $2^{n+q} - 2^q - const_G \cdot 2^q + 1$ equivalence classes.*



(a) Original obfuscated circuit: $F^{obfuscated}$.



(b) $F^{obfuscated}$ after split-compensate with internal variable x_4.

Fig. 6: Example of split-compensate with the internal variable x_4.

		$F^{obfuscated}$				
	$x_1x_2x_3x_4$	F	$\hat{k}_1$	$\hat{k}_2$	$\hat{k}_3$	$\hat{k}_4$
$\hat{in}_1$	0000	0	0	0	0	**1**
$\hat{in}_2$	0001	0	0	0	0	**1**
$\hat{in}_3$	0010	0	0	0	0	**1**
$\hat{in}_4$	0011	1	1	1	1	1
$\hat{in}_5$	0100	0	0	0	**1**	0
$\hat{in}_6$	0101	0	0	0	**1**	0
$\hat{in}_7$	0110	0	0	0	**1**	0
$\hat{in}_8$	0111	1	1	1	1	1
$\hat{in}_9$	1000	0	0	**1**	0	0
$\hat{in}_{10}$	1001	0	0	**1**	0	0
$\hat{in}_{11}$	1010	0	0	**1**	0	0
$\hat{in}_{12}$	1011	1	1	1	1	1
$\hat{in}_{13}$	1100	1	1	**0**	**0**	**0**
$\hat{in}_{14}$	1101	1	1	**0**	**0**	**0**
$\hat{in}_{15}$	1110	1	1	**0**	**0**	**0**
$\hat{in}_{16}$	1111	1	1	1	1	1

(a) Truth table of Figure 6(a).

		$F^{obfuscated}$								
	$x_1x_2x_3x_4$	F	$\hat{k}_1$	$\hat{k}_2$	$\hat{k}_3$	$\hat{k}_4$	$\hat{k}_5$	$\hat{k}_6$	$\hat{k}_7$	$\hat{k}_8$
$\hat{in}_1$	0000	0	0	0	0	0	0	0	0	**1**
$\hat{in}_2$	0001	0	0	0	0	0	0	0	**1**	0
$\hat{in}_3$	0010	0	0	0	0	0	0	0	0	**1**
$\hat{in}_4$	0011	1	1	1	1	1	1	1	1	1
$\hat{in}_5$	0100	0	0	0	0	0	0	**1**	0	0
$\hat{in}_6$	0101	0	0	0	0	0	**1**	0	0	0
$\hat{in}_7$	0110	0	0	0	0	0	0	**1**	0	0
$\hat{in}_8$	0111	1	1	1	1	1	1	1	1	1
$\hat{in}_9$	1000	0	0	0	0	**1**	0	0	0	0
$\hat{in}_{10}$	1001	0	0	0	**1**	0	0	0	0	0
$\hat{in}_{11}$	1010	0	0	0	0	**1**	0	0	0	0
$\hat{in}_{12}$	1011	1	1	1	1	1	1	1	1	1
$\hat{in}_{13}$	1100	1	1	1	1	1	1	1	1	1
$\hat{in}_{14}$	1101	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\hat{in}_{15}$	1110	1	1	1	1	1	1	1	1	1
$\hat{in}_{16}$	1111	1	1	1	1	1	1	1	1	1

(b) Truth table of Figure 6(b).

Fig. 7: Truth tables of split-compensate with internal variable x_4.

Proof: Omitted. ■

Proposition 4 shows that the number of equivalence classes increases exponentially with respect to the number of introduced external variables. Thus, it is effective to increase the number of equivalence classes by introducing external variables with the split-compensate operation.

D. Expanding Obfuscated AND-Tree with Internal Variables

Next, let us discuss how to mitigate the remove-all DIP issue. Our objective is to minimize the members of the equivalence class of remove-all DIPs. Proposition 2 states that when the support sets of $T^{obfuscated}$ and G are non-disjoint, the members of the equivalence classes are affected by the size of $OFF(G)$. When we introduce the compensation function, which can be seen as G, we can minimize the number of remove-all DIPs by manipulating the compensation function.

Definition 6. *(Internal variable). For an AND-tree T, the internal variable set of T, denoted as $internal(T)$, is defined as:*

$$internal(T) = SUP(O_{io})$$

where O_{io} denotes the set of POs in the transitive fanout cone of T, and $SUP(O_{io})$ denotes the support set of O_{io}.

Figure 5 shows an example of internal variables, where $internal(T)$ are highlighted by the purple color.

Proposition 5. *Given an AND-tree T embedded in a circuit F, where $F = T \vee G$ and G is an arbitrary function, if we perform the split-compensate operation to T by adding internal variables, the number of remove-all DIPs can be reduced.*

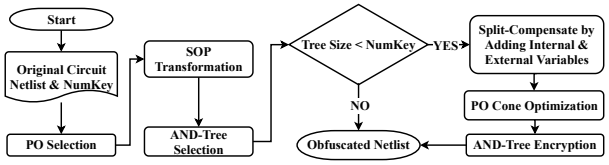


Fig. 8: Flow of the proposed method.

Proof: Omitted. ■

Fig. 6 shows an example of using an internal variable to effectively reduce the number of remove-all DIPs. Fig. 6(a) and (b) show the original obfuscated circuit and the resultant obfuscated circuit after we perform the split-compensate operation with the internal variable x_4, respectively. Their truth tables are shown in Fig. 7(a) and (b). As you can see, the number of remove-all DIPs is reduced from 3 to 1. Thus, it is effective to reduce the number of remove-all DIPs by introducing internal variables with the split-compensate operation.

E. Overall Flow

Since an AND-tree corresponds to a product term, the most straightforward method to find/construct an AND-tree is to find the a product term. Given a function to be obfuscated, we first collapse it into a 2-level sum-of-product (SOP) form. Then, we select one product term as the AND-tree and encrypt it with the split-compensate operation.

The overall flow of the proposed method is shown in Fig. 8. In the first step, **PO Selection**, we select the PO function with the lowest probability to be 1. In the second step, **SOP Transformation**, we use the Berkeley ABC tool [17] to collapse the selected PO function to the SOP form. In the third step, **AND-Tree Selection**, we find the largest AND-Tree with the minimum number of remove-all DIPs for encryption.

Next, we compute the input size of the selected AND-tree. If the tree input size is smaller than the number of key inputs specified by the user, we entrance the fourth step, **Split-Compensate by Adding Internal & External Variables**.

Then, in the fifth step, **PO Cone Optimization**, we optimize the selected PO function except the AND tree with the Berkeley ABC tool, Finally, we encrypt the AND tree with randomly selected XOR/XNOR key gates to obtain the obfuscated netlist.

In the following subsection, we only explain the step of **Split-Compensate by Adding Internal & External Variables** in details, since the other steps are more straightforward.

F. Split-Compensate by Adding Internal and External Variables

Fig. 9 shows the flow of **Split-Compensate by Adding Internal and External Variables**. First, we compute the number of remove-all DIPs of the selected AND-tree $T^{obfuscated}$ by computing the number of test patterns that can detect the stuck-at-0 fault at the output of $T^{obfuscated}$. Then, we check whether the number of remove-all DIPs is less than a threshold value specified by the user. If not, internal variables are selected to perform the split-compensate operation to reduce the number of remove-all DIPs until the desired number is met.

Based on Proposition 3, The method of selecting internal variables is as follows: First, we collect all the test patterns, denoted as $\{t_p\}$, that can detect the stuck-at-0 fault at the output of $T^{obfuscated}$ by the ATPG tool. Next, for each variable x_i in

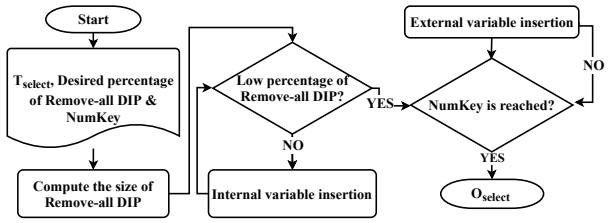


Fig. 9: Flow of the split-compensate by adding internal & external variables.

TABLE III: Test patterns generated by ATPG.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
$\hat{t}_1$	1	1	0	0	0	X	0	X
$\hat{t}_2$	1	1	0	0	1	0	0	X
$\hat{t}_3$	1	1	0	0	0	X	1	0
$\hat{t}_4$	1	1	0	0	1	0	1	0

$\{t_p\}$, we compute the number of test patterns with $x_i = v$ (v is 0 or 1), denoted as $\#t_p(x_i, v)$. Then, we select the variable x_i with the smallest non-zero $\#t_p(x_i, v)$ as the internal variable x_{select} for performing the split-compensate operation. Finally, x_{select} with its selected phase v is added to $T^{obfuscated}$ and the test patterns with $x_{select} = v'$ are removed from $\{t_p\}$.

The reason to select the variable and the phase with a smaller test pattern count is that we would like to eliminate more test patterns from $\{t_p\}$ in one iteration, i.e., more remove-all DIPs. Since we use x_{select} with the phase v to expand the tree, the patterns with $x_{select} = v'$ are not test patterns anymore due to the phase conflict. Furthermore, we choose the non-zero $\#t_p(x_i, v)$ to avoid eliminating all the remove-all DIPs for defending against Removal attack. When we keep one remove-all DIP, it indicates that $T^{obfuscated}$ has one minterm of the selected PO function, which is not covered by the compensation function. An attacker who directly removes $T^{obfuscated}$ based on Removal attack cannot recover the minterm.

The internal variable selection is demonstrated as follows: Suppose that the test patterns generated by the ATPG tool are listed in Table III, and x_5, x_6, x_7 and x_8 are the internal variables to be selected. In the first iteration, $x_7 = 1$ is selected, and $\hat{t}_1$ and $\hat{t}_2$ are eliminated. Next, $x_6 = 0$ is selected and the remove-all DIP count is reduced to 1.

When the remove-all DIP count is less than the threshold value, we then repeatedly select a random external variable and add it to $T^{obfuscated}$ with the split-compensate operation, until the desired key size is reached.

V. Experimental Results

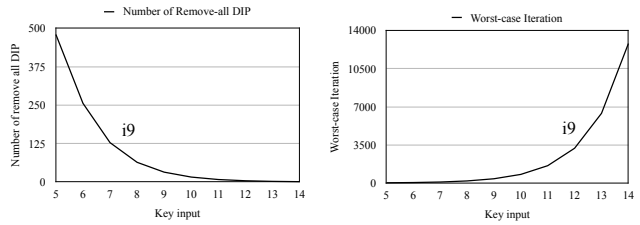
In the experiments, we evaluate the effectiveness of the proposed method to resist SAT attack. We use the Berkeley ABC tool [17] for synthesis and the Atalanta ATPG tool [18] to generate test patterns. SAT Attack is adopted from [3], and we set two time limits of 3 hours and 5 hours to SAT attack. All the experiments were conducted on a CentOS machine with two Intel Xeon(R) Gold 6226R 2.90GHz CPUs and 128G memory. The benchmarks are chosen from ISCAS'85 [15] and MCNC benchmarks [16].

A. Effectiveness of SOP Transformation

We conduct the SOP Transformation to find the largest AND-tree in each benchmark circuit and compare the results to that found by the tree detection method [9]. The experimental results are shown in Table IV. Columns 2 and 3 show the statistics of the benchmarks. Columns 4 and 5 show the sizes of

TABLE IV: Sizes of the largest AND/OR-trees.

Circuit	#PI/#PO	#Gate	Input size of largest AND/OR-tree		#Support PI of tree [9]
			[9]	SOP Transformation	
c1355	41/32	546	9	-	41
c1908	33/25	880	27	30	33
c2670	233/140	1193	34	40	108
c3540	50/22	1669	24	29	35
c5315	178/123	2307	18	26	29
c7552	207/108	3512	35	22	94
ex5	8/63	1055	9	8	8
i7	199/67	1315	4	4	6
k2	46/45	1815	175	20	41
i9	88/63	1035	15	8	10
apex2	39/3	610	9	21	11
i4	192/6	338	7	16	47
des	256	245	31	17	13



(a) Effectiveness of internal variables. (b) Effectiveness of external variables.

Fig. 10: Effectiveness of extra variables.

the largest AND/OR-trees found by the tree detection method and the SOP transformation, respectively. For 6 out of 13 benchmarks, the tree sizes found by the SOP Transformation are larger than that found by the tree detection method. However, for c1355, the SOP Transformation cannot successfully collapse the selected PO function into the SOP form, since the benchmark contains a lot of arithmetic logic. For the ex5 and k2 benchmarks, the input counts of the AND/OR-trees found by the tree detection method are even larger than the PI counts. It indicates that the inputs of the AND/OR-trees are highly correlated, and thus the security of the trees could be over-estimated. As you can see in the last column, for the k2 benchmark, the support PI count of the tree found by the tree detection method is much smaller than the tree input count. As for the SOP Transformation, all the inputs of the found trees are PIs. Encrypting the trees could lead to better security.

B. Effectiveness of Extra Variables

We conduct the experiment on the i9 benchmark to evaluate the effectiveness of inserting internal variables to minimize the number of remove-all DIPs. Fig. 10(a) shows the relation between the number of inserted internal variables and the number of remove-all DIPs. When the number of key inputs increases from 5 to 14 by adding internal variables, the number of remove-all DIPs dramatically drops. Furthermore, Fig. 10(b) shows the effectiveness of adding external variables to expand the tree size.

C. Effectiveness of Anti-SAT Attack

In this experiment, we use the proposed method to encrypt the benchmarks and apply SAT attack to decrypt them. Table V shows the experimental results. Column 1 lists the benchmarks. Column 2 shows the numbers of inserted key inputs. Column 3 shows the numbers of iterations spent by SAT Attack when it reaches the time limit or ends early. Columns 4 and 5 show the SAT Attack execution time under the 3- and 5-hour limits, respectively, "TLE" indicates that the benchmark cannot be

TABLE V: Resilience against SAT attack with 32 key bits.

Circuit	KI	Our method			
		IT	$T_{3h}(s)$	$T_{5h}(s)$	Area (%)
i4	32	>35732	TLE	TLE	31.952
i9	32	23025	TLE	15554.1	25.507
i7	32	5033	811.131	811.131	10.494
des	32	>12238	TLE	TLE	3.661
apex2	32	>20270	TLE	TLE	234.262
c5315	32	>11316	TLE	TLE	181.554
c7552	32	>11784	TLE	TLE	91.625

successfully decrypted by SAT attack within the time limit. Note that the i9 and i7 benchmarks are cracked in a relatively short time, which is due to the fact that we retain one remove-all DIP and the probability of finding the DIP is related to the heuristic of the SAT Solver. Column 6 represents the area overhead. Note that the area overhead for apex2, c5315 and c7552 is significantly large. It is mainly contributed by the SOP transformation. Functions of c5315 and c7552 are 9-bit ALU and 32-bit adder/comparator, respectively. For Apex2, collapsing one of PO cones results in 276 product terms.

VI. Conclusions

In this paper, we present a new method for constructing obfuscated trees. We first use the SOP Transformation to find the largest AND-tree in the circuit and then insert extra variables by the split-compensate operation to further expand the AND-tree and reduce the number of remove-all DIPs. The experimental results show that the proposed method can find larger AND-trees than a previous tree detection method for most benchmarks. Additionally, the constructed obfuscated trees can effectively resist SAT attack.

References

[1] A. Mutschler, "SEMI: Semi equipment industry stands to lose up to 4B annually due to IP infringement", *Electronic News*, 2008, 54(18):17-22.

[2] J. Roy, *et al*. "EPIC: Ending piracy of integrated circuits.", In *DATE*, pp. 1069-1074, 2008.

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

[4] A. Alaql and S. Bhunia. "Saro: Scalable attack-resistant logic locking", *IEEE TIFS*, 16:3724–3739, 2021.

[5] H. M. Kamali, *et al*. "Full-Lock: Hard Distributions of SAT instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," In *DAC*, pp. 1-6, 2019.

[6] A. Saha, *et al*. "LoPher: SAT-Hardened Logic Embedding on Block Ciphers," In *DAC*, pp. 1-6, 2020.

[7] M. Yasin, *et al*. "Camoperturb: Secure ic camouflaging for minterm protection", In *ICCAD*, pp. 1–8, 2016.

[8] Y. Chen. "Smartlock: Sat attack and removal attack-resistant tree-based logic locking", *IEICE TFECCS*, E103.A(5):733–740, 2020.

[9] M. Li, el al. "Provably secure camouflaging strategy for ic protection", *IEEE TCAD*, 38(8):1399–1412, 2019.

[10] Y. Xie and A. Srivastava. "Anti-sat: Mitigating sat attack on logic locking", *IEEE TCAD*, 38(2):199207, 2019.

[11] M. Yasin, *et al*. "Sarlock: Sat attack resistant logic locking", In *HOST*, pp. 236–241, 2016.

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

[13] M. Yasin, *et al*. "Provably-secure logic locking: From theory to practice", In *ACM CCS*, pp. 1601–1618, New York, NY, USA, 2017.

[14] M. Yasin, *et al*. "What to lock? functional and parametric locking", In *GLSVLSI*, pp. 351–356, New York, NY, USA, 2017.

[15] F. Brglez, *et al*. "Combinational profiles of sequential benchmark circuits", in *ISCAS*, pp. 1929–1934.1989.

[16] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0", *MCNC*, New York, NY, USA, Rep. 1991.

[17] ABC: Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification".

[18] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits", *Technical Report* No. 12−93, Dep't of Eng., Virginia Polytechnic Institute and State University.