Anti-SAT: Mitigating SAT Attack on Logic Locking

Yang Xie, *Member, IEEE* and Ankur Srivastava, *Senior Member, IEEE*

***Abstract*—Logic locking is a technique that's proposed to protect outsourced IC designs from piracy and counterfeiting by untrusted foundries. A locked IC preserves the correct functionality only when a correct key is provided. Recently, the security of logic locking is threatened by a new attack called SAT attack, which can decipher the correct key of most logic locking techniques within a few hours [1] even for a reasonably large key-size. This attack iteratively solves SAT formulas which progressively eliminate the incorrect keys till the circuit is unlocked. In this paper, we present a circuit block (referred to as Anti-SAT block) to enhance the security of existing logic locking techniques against the SAT attack. We show using a mathematical proof that the number of SAT attack iterations to reveal the correct key in a circuit comprising an Anti-SAT block is an exponential function of the key-size thereby making the SAT attack computationally infeasible. Besides, we address the vulnerability of the Anti-SAT block to various removal attacks and investigate obfuscation techniques to prevent these removal attacks. More importantly, we provide a proof showing that these obfuscation techniques for making Anti-SAT un-removable would not weaken the Anti-SAT block's resistance to SAT attack. Through our experiments, we illustrate the effectiveness of our approach to securing modern chips fabricated in untrusted foundries.**

***Index Terms*—Logic Locking, SAT Attack, Hardware IP Protection, Hardware Security**

I. Introduction

NOWADAYS, chip design and fabrication are normally conducted by different facilities in a global supply chain. Most integrated circuit (IC) design companies are now adopting a fab-less model: they outsource the chip fabrication to offshore foundries while concentrating their effort and resource on the chip design. IC fabrication outsourcing enables IC design companies to access advanced semiconductor technology at a low cost. However, the outsourced design faces various security threats since the offshore foundry might not be trustworthy. Attacks by untrusted foundries on the outsourced IC design can take on many forms, such as Intellectual Property (IP) piracy [2], counterfeiting [3] and hardware Trojans [4]. These security threats (also known as supply chain attacks) pose a significant economic risk to most IC design companies.

Logic locking [5] is a technique that is proposed to thwart the aforementioned supply chain attacks. The basic idea is to insert additional key-controlled logic gates (key-gates), key-inputs and an on-chip memory into an IC design to hide its original functionality, as shown in Fig. 1. The key-gate can be implemented using XOR/XNOR gates [5], [6], [7], MUX gates [7], [8], [9], [10] and Look-Up-Tables (LUTs) [11], [12], [13]. The key-inputs are connected to the on-chip memory and

Y. Xie and A. Srivastava are with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA (email: yangxie@umd.edu; ankurs@umd.edu).

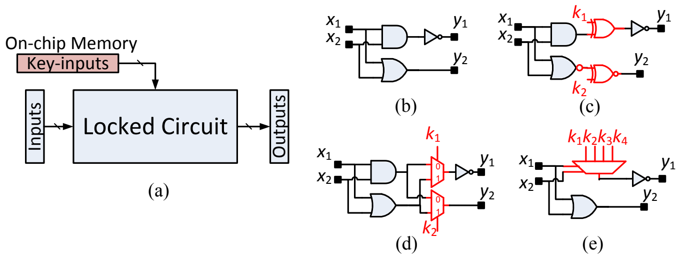

Fig. 1. Logic locking techniques: (a) Overview; (b) An original netlist; (c) XOR/XNOR based logic locking; (d) MUX based logic locking; (e) LUT based logic locking.

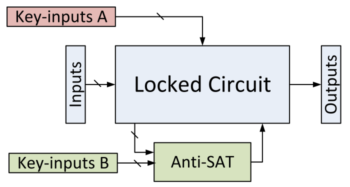

Fig. 2. A basic overview of Anti-SAT block enhanced logic locking.

the locked IC preserves the correct functionality only when a correct key is set to the on-chip memory. To prevent the untrusted foundry from probing the key values of a running chip, a tamper-proof chip protection shall be implemented.

A. Motivation

The security of logic locking is threatened if the correct key values into the key-gates are accessible to or can be learned within a practical time by the malicious foundries. Recently, Subramanyan *et al.* [1] proposed a satisfiability checking based attack (*SAT attack*) that can effectively break most logic locking techniques proposed in [5], [6], [7], [11], [14] within a few hours even for a reasonably large key-size. The insight of the SAT attack is to iteratively solve a sequence of SAT formulas which can progressively eliminate the incorrect keys till all the wrong key combinations have been eliminated. The SAT attack is powerful because it guarantees that upon termination it can always reveal the correct key. This severe security threat motivates this work to develop a secure logic locking design which can thwart the SAT attack.

B. Main Contribution

In this work, we have developed a relatively lightweight circuit block (referred to as Anti-SAT block) that can be embedded into a design to mitigate the SAT attack. The basic structure of our Anti-SAT block is shown in Fig. 2. While a portion of keys (key-inputs A) is connected to the original circuit to obfuscate its functionality, another portion of keys (key-inputs B) is connected to the Anti-SAT block to thwart the SAT attack. *The Anti-SAT block is designed in a way that the total number of SAT attack iterations (and thus the total execution time) to reveal the correct key in the Anti-SAT block is an exponential function of the key-size in the Anti-SAT block.* A slight increase in key-size can lead to exponentially increase in SAT attack complexity. Therefore, the Anti-SAT block can be integrated into a design to efficiently enhance its resistance to the SAT attack. The contribution of this work is as follows.

- We propose an Anti-SAT circuit block to mitigate the SAT attack on logic locking. We illustrate how to construct the functionality of the Anti-SAT block and use a mathematically rigorous approach to prove that if chosen correctly, the Anti-SAT block makes SAT attack computationally infeasible (exponential in key-size).
- The Anti-SAT block can be integrated into the original netlist to enhance its resistance to the SAT attack. A secure integration method is proposed which ensures that the SAT attack complexity is maintained after integration.
- The Anti-SAT block might be subject to attacks that intend to identify and nullify it, which are called *removal attacks*. In this work we highlight the unique functional and structural attributes of the Anti-SAT block that could be exploited for removal attacks. Based on that we investigate a unified obfuscation technique to hide the functionality and structure of the Anti-SAT block. More importantly, we provide a proof showing that the obfuscation technique would not weaken the Anti-SAT block's resistance to SAT attack.
- Rigorous analysis and experiments on 6 circuits from ISCAS85 and MCNC benchmark suites have been conducted to validate the effectiveness of our proposed technique in improving the security of existing logic locking techniques.

The rest of the paper is organized as follows. Section II provides a review on logic locking and SAT attack. Section III proposes the Anti-SAT block design and demonstrates its resistance against the SAT attack. Section IV outlines the unique functional and structural attributes of the Anti-SAT block that could be exploited for removal attacks and investigates corresponding obfuscation methods. Section V describes the overall Anti-SAT based logic locking design flow. Experiments and results are shown in Section VI. Section VII discusses related works, followed by conclusion in Section VIII.

II. Background

A. Logic Locking

Logic locking is a technique that inserts a set of key-gates and key-inputs into an IC design to protect it from supply chain attacks by untrusted foundries. Recent years have seen various logic locking techniques based on different key-gate types and key-gate insertion algorithms. According to the key-gate types, they can be classified into three major categories: XOR/XNOR based logic locking [5], [6], [7], MUX based logic locking [7], [8], [9], [10] and LUT based logic locking [11], [12], [13], as shown in Fig. 1 (b-e). Other key-gate types such as AND/OR gates [14] and special logic cones [16] have also been investigated. Among all, the XOR/XNOR based logic locking has received the most attention mainly due to its simple structure and low performance overhead. Various XOR/XNOR based logic locking algorithms have been proposed to identify the optimal locations for inserting the key-gates. The simplest algorithm is the random insertion [5], where key-gates are distributed randomly into the netlist. In [7], a fault-analysis based insertion algorithm was proposed which inserts the key-gates at locations that can maximally affect the primary outputs. It ensures that the functionality of the locked circuit (with a wrong key) would deviate substantially from the original one. In [6], an interference-analysis based insertion algorithm was proposed which places the key-gates at locations with complex interferences among them such that a correct key cannot be easily sensitized to the primary outputs. In [10], a logic-cone analysis based insertion algorithm was proposed to increase the number of key-gates per logic cone to thwart an exhaustive search based key-learning attack. The security objective of these logic locking techniques is to increase the output corruptibility (*i.e.,* produce more incorrect outputs for more input patterns) given an incorrect key, and to prevent effective key-learning attacks.

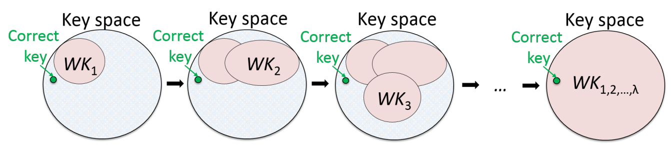

Fig. 3. Illustration of the iterative SAT attack process. Wrong key combinations are iteratively identified by a set of DIOs till no new ones exist. WK_i is the set of wrong key combinations identified by i-th DIO.

B. SAT Attack

SAT attack [1] is a newly proposed attack on logic locking which can effectively break most existing logic locking techniques. Here we introduce the attack model and describe the insight and algorithm of the SAT attack.

1) Attack Model: The SAT attack model [1] assumes that the attacker is an untrusted foundry whose objective is to obtain the correct key of a locked circuit. The malicious foundry has access to the following two components:

- A locked gate-level netlist, which can be obtained by reverse-engineering a GDSII layout file. This is available because the fabrication is done by the untrusted foundry which has the layout details provided by the designer. The locked netlist is represented as $\vec{Y} = f_l(\vec{X}, \vec{K})$ with primary inputs $\vec{X}$, key inputs $\vec{K}$ and primary outputs $\vec{Y}$. Its SAT formula in conjunctive normal form (CNF) is represented as $C(\vec{X}, \vec{K}, \vec{Y})$.
- An activated functional chip, which can be obtained from open market. This IC can be used to evaluate a set of input patterns and observe their correct output patterns as a black box model $\vec{Y} = eval(\vec{X})$.

2) Attack Insight: The key idea of the SAT attack is to reveal the correct key using a small number of carefully selected inputs and their correct outputs observed from an activated functional chip. These special input/output pairs are referred to as *distinguishing input/output (DIO) pairs*. Each DIO can identify a subset of *wrong key combinations* and all together they guarantee that only the correct key can be consistent with these correct I/O pairs. This implies that a key that correctly matches the inputs to the outputs for all the DIOs must be the correct key.

Definition 1 (**Wrong key combination**)**.** Consider the logic function $\vec{Y} = f_l(\vec{X}, \vec{K})$ and its CNF SAT formula $C(\vec{X}, \vec{K}, \vec{Y})$. Let $(\vec{X}, \vec{Y}) = (\vec{X}_i, \vec{Y}_i)$, where $(\vec{X}_i, \vec{Y}_i)$ is a correct I/O pair. The set of key combinations WK_i which result in an incorrect output of the logic circuit (*i.e.,* $\vec{Y}_i \neq f_l(\vec{X}_i, \vec{K})$, $\forall \vec{K} \in WK_i$) is called the set of wrong key combinations identified by the I/O pair $(\vec{X}_i, \vec{Y}_i)$. In terms of SAT formula, it can be represented as $C(\vec{X}_i, \vec{K}, \vec{Y}_i) = False$, $\forall \vec{K} \in WK_i$.

Definition 2 (**Distinguishing input/output (DIO) pair**)**.** As noted above, the SAT attack shall solve a set of SAT formulas iteratively. In each iteration, it shall find a correct I/O pair to identify a subset of wrong key combinations until none of these are left. An I/O pair at i-th iteration is a DIO, denoted as $(\vec{X}_i^d, \vec{Y}_i^d)$, if it can identify a *unique* subset of wrong key combinations that cannot be identified by the previous $i-1$ DIOs, *i.e.,* $WK_i \not\subset (\cup_{j=1}^{j=i-1} WK_j)$, where WK_i is the set of wrong key combinations identified by the DIO at i-th iteration.

The crux of the SAT attack algorithm relies on finding the DIOs iteratively to identify *unique* wrong key combinations (see Definition 2) until no further ones can be found. At this point, the set of all DIOs *together* can identify all wrong key combinations thereby revealing the correct one. An illustration of the SAT attack process is shown in Fig. 3. In each iteration, the SAT attack will find a new DIO that can rule out a subset of wrong key combinations WK_i. Notice that each iteration can identify unique wrong key combinations that are not belong to the ones discovered previously, *i.e.,* $WK_i \not\subset (\cup_{j=1}^{j=i-1} WK_j)$. The attack terminates when all wrong key combinations are identified.

Take the XOR/XNOR based locked circuit in Fig. 1(c) as an example. At first iteration, the I/O pair $(\vec{X}_1^d, \vec{Y}_1^d) = (00, 10)$ is a DIO because it can rule out wrong key combinations $\vec{K} = (01)$, (10), and (11) as these key combinations will result in incorrect outputs $(y_1 y_2) = (11)$, (00) and (01), respectively. Since this single I/O observation has already ruled out all incorrect key combinations, we have revealed the correct key $\vec{K} = (00)$. In general, a small number of DIOs (compared to all possible I/O pairs) are usually enough to infer the correct key [1]. As a result, the SAT attack is efficient because it only requires a small number of iterations to find these DIOs.

3) Attack Algorithm: As noted above, the central theme of SAT attack algorithm is to iteratively find DIOs to rule out wrong key combinations till no new ones can be found. Here we briefly introduce the SAT attack algorithm for finding the DIOs. Firstly, the algorithm will formulate a SAT formula that can be solved by state-of-the-art SAT solvers. The SAT

Algorithm 1 SAT Attack Algorithm [1]

```
Input: $C$ and $eval$
Output: $\vec{K}_C$
1: $i := 1$;
2: $G_i := True$;
3: $F_i := C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2) \wedge (\vec{Y}_1 \neq \vec{Y}_2)$;
4: while sat[$F_i$] do
5:     $\vec{X}_i^d :=$ sat_assignment$_{\vec{X}}[F_i]$;
6:     $\vec{Y}_i^d := eval(\vec{X}_i^d)$;
7:     $G_{i+1} := G_i \wedge C(\vec{X}_i^d, \vec{K}, \vec{Y}_i^d)$;
8:     $F_{i+1} := F_i \wedge C(\vec{X}_i^d, \vec{K}_1, \vec{Y}_i^d) \wedge C(\vec{X}_i^d, \vec{K}_2, \vec{Y}_i^d)$;
9:     $i := i + 1$;
10: end while
11: $\vec{K}_C :=$ sat_assignment$_{\vec{K}}(G_i)$;
```

formula F_i at i-th iteration is:

$$F_i := C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2) \wedge (\vec{Y}_1 \neq \vec{Y}_2) \\ (\bigwedge_{j=1}^{j=i-1} C(\vec{X}_j^d, \vec{K}_1, \vec{Y}_j^d)) \wedge (\bigwedge_{j=1}^{j=i-1} C(\vec{X}_j^d, \vec{K}_2, \vec{Y}_j^d)) \tag{1}$$

where $\vec{X}, \vec{K}_1, \vec{K}_2, \vec{Y}_1$ and $\vec{Y}_2$ are variables, $(\vec{X}_j^d, \vec{Y}_j^d), j = 1, ..., i-1$ are DIOs found in previous $i-1$ iterations. A detailed explanation of this formula can be found in [1]. Basically, this formula determines whether there still exist unique wrong key combinations that haven't been identified by all previous $i-1$ DIOs . If F_i is satisfiable, it means that such wrong key combinations still exist. By solving the SAT formula, the SAT solver will generate an assignment to the input variable $\vec{X} = \vec{X}_i^d$, which is the distinguishing input needed to form a new DIO. After $\vec{X}_i^d$ is obtained, it is fed into an activated functional chip obtained from the open market and the correct output $\vec{Y}_i^d$ is observed. This correct I/O pair forms the i-th DIO ($\vec{X}_i^d$, $\vec{Y}_i^d$) which can be used to eliminate new wrong key combinations. The processing of finding DIOs is continued till no new ones can be found (assuming after λ iterations). At this point, a correct key can be obtained by solving the following SAT formula G:

$$G := \bigwedge_{i=1}^{\lambda} C(\vec{X}_i^d, \vec{K}, \vec{Y}_i^d) \tag{2}$$

Basically it finds a key $\vec{K}$ which satisfies the correct functionality for all the DIOs. This must be the correct key since no other DIOs exist at this point (see Definition 2).

The SAT attack algorithm is shown in Algorithm 1. It starts by first solving the line one of the SAT formula (1) and as iterations progress it adds the clauses comprised in line two of the formula (1). It stops when the resulting SAT formula is unsatisfiable indicating no further DIOs exist. The correct key is obtained by finding a key value which satisfies the correct I/O behavior of all the DIOs (SAT formula (2)). This algorithm is guaranteed to find the correct key. Please refer to [1] for any further theoretical details.

III. Anti-SAT Block Design

The efficiency of SAT attack can be evaluated by the total execution time:

$$T = \sum_{i=1}^{\lambda} t_i \tag{3}$$

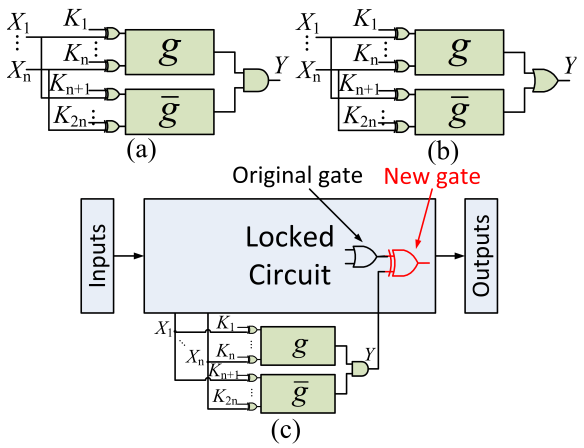

Fig. 4. Anti-SAT block configuration: (a) Type-0 Anti-SAT: always outputs 0 if key values are correct; (b) Type-1 Anti-SAT: always outputs 1 if key values are correct. (c) Integrating the Type-0 Anti-SAT block into a circuit.

where λ is the total number of SAT attack iterations and t_i is the SAT solving time for i-th iteration. Consequently, the SAT attack can be mitigated if t_i is large and/or λ is large. λ depends on the key-size and key location in the locked circuit. However, simply increasing the key-size or trying different key locations may not effectively thwart the SAT attack. As shown in the SAT attack results [1], even with large number of keys (50% area overhead), for six previously proposed key-gate insertion algorithms [5], [6], [7], [11], [14], 86% benchmarks on average can still be unlocked in 10 hours.

To mitigate the SAT attack, we propose to insert a relatively light-weight circuit block (referred to as Anti-SAT block) that can efficiently increase the number of iterations λ so as to increase the total execution time T.

A. *Configurations of Anti-SAT*

Fig. 4(a) and Fig. 4(b) illustrate two configurations of the proposed Anti-SAT block, referred to as *type-0 Anti-SAT* and *type-1 Anti-SAT*. They consist of two logic blocks g and $\overline{g}$, which share the same set of inputs $\vec{X} = (X_1 ... X_n)$. The functionalities of g and $\overline{g}$ are complementary. A set of key-gates (XORs [1]) are inserted at the inputs of two logic blocks, denoted as $\vec{K}_{l1} = (K_1 ... K_n)$ and $\vec{K}_{l2} = (K_{n+1} ... K_{2n})$. Hence the key-size is $2n$. The output of g and $\overline{g}$ are fed into an AND2 gate (for Fig. 4(a)) or an OR2 gate (for Fig. 4(b)) to form the final single-bit output Y. As a result, we have $Y = g(\vec{X} \oplus \vec{K}_{l1}) \wedge \overline{g(\vec{X} \oplus \vec{K}_{l2})}$ for type-0 Anti-SAT and $Y = g(\vec{X} \oplus \vec{K}_{l1}) \vee \overline{g(\vec{X} \oplus \vec{K}_{l2})}$ for type-1 Anti-SAT.

1) Constant-output Property: one basic property of Anti-SAT block is that when the key vector is correctly set, the output Y is a constant. Specifically, given a correct key, Y always outputs value 0 for type-0 Anti-SAT (Fig. 4(a)) and always outputs value 1 for type-1 Anti-SAT (Fig. 4(b)). Otherwise, when a wrong key is given, Y can output either 1 or 0 depending on the inputs $\vec{X}$. This property enables it to be integrated into the original circuit. Fig. 4(c) shows an example of integrating a type-0 Anti-SAT into a circuit. As seen, the inputs of Anti-SAT block $\vec{X}$ are from the wires in the original circuit. The output Y is connected into the original circuit using an XOR gate. When a correct key is provided, the output Y always equals to 0 (so the XOR gate behaves as a buffer) and thus will not affect the functionality of the original circuit. If a wrong key is provided, Y can be 1 for some inputs (so the XOR gate behaves as an inverter) and thus can produce a fault in the original circuit. Similarly, the type-1 Anti-SAT block can be integrated into the original circuit using an XNOR gate.

2) Correct Keys: Since the Anti-SAT block has $2n$ keys, the total number of wrong key combinations is $2^{2n} - c$, assuming there exists c correct key combinations. To ensure the constant-output property, the correct keys for the Anti-SAT block would be the ones that make type-0 Anti-SAT always output 0 and type-1 Anti-SAT always output 1. This happens when i-th key-bit from $\vec{K}_{l1}$ and i-th key-bit from $\vec{K}_{l2}$ have the same value, so the number of correct key combinations $c = 2^n$ for both types of Anti-SAT blocks and the number of wrong key combinations is $2^{2n} - 2^n$.

In the subsequent sections, we provide details on constructing the Anti-SAT block (*i.e.,* the functionality of g) and its impact on SAT attack complexity. We provide a rigorous mathematical analysis which gives a provable lower bound to the number of SAT attack iterations. For some constructions of g, this lower bound is exponential in the key-size thereby making the SAT-attack complexity very high.

B. *Complexity Analysis of SAT Attack on Anti-SAT*

Here we analyze the complexity of SAT attack on the Anti-SAT block (assuming this is the circuit being attacked to decipher the $2n$ key bits).

Terminology: Given a Boolean function $g(\vec{L})$ with n inputs, assuming there exists p input vectors that make g equal to one $(1 \leq p \leq 2^n - 1)$, we can classify the input vectors $\vec{L}$ into two groups L^T and L^F, where

$$\begin{aligned} L^T &= \{\vec{L} | g(\vec{L}) = 1\}, \quad (|L^T| = p) \\ L^F &= \{\vec{L} | g(\vec{L}) = 0\}, \quad (|L^F| = 2^n - p) \end{aligned} \tag{4}$$

We denote L^T as the *on-set* of function g, L^F as the *off-set* of function g, and p as the size of on-set.

The function g and its complementary function $\overline{g}$ are used to construct the Anti-SAT block as shown in Fig. 4.

Theorem 1. *Assuming the size of on-set p of function g is sufficiently close to 1 or sufficiently close to $2^n - 1$, the number of iterations needed by the SAT attack to decipher the correct key is lower bounded by 2^n.*

Proof for Type-0 Anti-SAT: As shown in Section II-B, the SAT attack algorithm will iteratively find a DIO $(\vec{X}_i^d, Y_i^d)$

[1]Note that here we are using only XOR gates as key-gates for the sake of ease of explanation. The key-gates used could be either XOR or XNOR gates (+ inverters) based on a user-defined key [7]. The usage of inverters can remove the association between key-gate types and key-values (*e.g.* the correct key into an XOR gate can be either 0 or 1). Besides, the synthesis tools can "bubble push" the inverters to their fan-out gates and an attacker cannot easily identify which inverters are part of the key-gates [7]. Therefore, the attacker cannot obtain the correct key-values by simply inspecting the key-gate types.

to identify wrong key combinations in the Anti-SAT block until all wrong key combinations are identified. In the i-th iteration, the corresponding DIO can identify a subset of wrong key combinations, denoted as WK_i. Notice that for any input combinations (including the distinguishing inputs $\vec{X}_i^d$), the correct output (when provided the correct key) is 0 for type-0 Anti-SAT. Therefore, a wrong key combination $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2}) \in WK_i$ which was identified by ($\vec{X}_i^d$, Y_i^d) must produce the Anti-SAT block output incorrectly as 1. This condition is described below.

$$\begin{aligned} & Y_i^d = g(\vec{X}_i^d \oplus \vec{K}_{l1}) \wedge \overline{g(\vec{X}_i^d \oplus \vec{K}_{l2})} = 1 \\ \Leftrightarrow\ & (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 1) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 0) \\ \Leftrightarrow\ & ((\vec{X}_i^d \oplus \vec{K}_{l1}) \in L^T) \wedge ((\vec{X}_i^d \oplus \vec{K}_{l2}) \in L^F) \end{aligned} \tag{5}$$

Basically Equation (5) states that the wrong key identified in the i-th iteration must be such that its output Y_i^d should be 1. This implies that both g and $\overline{g}$ must evaluate to 1. This means that the input to g, which is $\vec{X}_i^d \oplus \vec{K}_{l1}$, should be in L^T and the input to $\overline{g}$, which is $\vec{X}_i^d \oplus \vec{K}_{l2}$, should be in L^F.

Since $\vec{X}_i^d \oplus \vec{K}_{l1}$ is the input vector to g, for any given $\vec{X}_i^d$, we can always find a key $\vec{K}_{l1}$ such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$. Basically $\vec{X}_i^d \oplus \vec{K}_{l1}$ flips some of the bits of $\vec{X}_i^d$ (for which corresponding $\vec{K}_{l1}$ bits are 1) while keeping other bits the same (for which corresponding $\vec{K}_{l1}$ bits are 0). Hence for a given $\vec{X}_i^d$, we can always choose $\vec{K}_{l1}$ such that the resulting input to g is in L^T. However note that $|L^T| = p$ in Equation (4). Hence for any given $\vec{X}_i^d$, we can select $\vec{K}_{l1}$ in p different ways such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$.

Similarly, for any given $\vec{X}_i^d$, we can always find a key $\vec{K}_{l2}$ such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$. Note that $|L^F| = 2^n - p$ in Equation (4). Hence for any given $\vec{X}_i^d$, we can select $\vec{K}_{l2}$ in $2^n - p$ different ways such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$.

Now, as noted above, for a given $\vec{X}_i^d$, a wrong key $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2})$ should be such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$ and $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$. The total number of ways in which we can select such a wrong key is $p \cdot (2^n - p)$.

Now in any given iteration i, for a given X_i^d, the *maximum* number of incorrect keys identified is $p \cdot (2^n - p)$. This follows naturally from the discussion above. This is the *maximum* number because it is very much possible that some of these keys were identified in *previous* iterations. Hence the total number of *unique* wrong keys UK_i identified in iteration i is upper-bounded by $p \cdot (2^n - p)$. This is noted in the equation below.

$$p \cdot (2^n - p) \geq UK_i \tag{6}$$

The SAT attack works by iteratively removing all incorrect keys till only the correct ones are left (assuming after λ iterations). Hence the following holds true.

$$\lambda(p \cdot (2^n - p)) \geq \sum_{i=1}^{\lambda} UK_i \tag{7}$$

Since $\sum_{i=1}^{\lambda} UK_i$ is the total number of wrong key combinations, its value is $2^{2n} - 2^n$ as discussed in Section III-A2. Equation (7) can be rewritten as follows.

$$\lambda \geq \frac{2^{2n} - 2^n}{p(2^n - p)} \tag{8}$$

We denote this lower bound on λ as λ_l. When $p \to 1$ or $p \to 2^n - 1$, we have the lower bound as follows:

$$\lambda_l = \frac{2^{2n} - 2^n}{p(2^n - p)} \to \frac{2^{2n} - 2^n}{1 \times (2^n - 1)} = 2^n \tag{9}$$

■

Proof for Type-1 Anti-SAT: For type-1 Anti-SAT, the correct output (when provided the correct key) is always 1. Therefore, a wrong key combination $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2}) \in WK_i$ which was identified by ($\vec{X}_i^d$, Y_i^d) must produce the incorrect output as 0. This condition is described below.

$$\begin{aligned} & Y_i^d = g(\vec{X}_i^d \oplus \vec{K}_{l1}) \vee \overline{g(\vec{X}_i^d \oplus \vec{K}_{l2})} = 0. \\ \Leftrightarrow\ & (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 0) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 1) \\ \Leftrightarrow\ & ((\vec{X}_i^d \oplus \vec{K}_{l1}) \in L^F) \wedge ((\vec{X}_i^d \oplus \vec{K}_{l2}) \in L^T) \end{aligned} \tag{10}$$

Based on the discussion in the proof for type-0 Anti-SAT, we know that for any given $\vec{X}_i^d$, we can select $\vec{K}_{l1}$ in $2^n - p$ different ways such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^F$. Also, for any given $\vec{X}_i^d$, we can select $\vec{K}_{l2}$ in p different ways such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^T$. As noted in Equation (10), for a given $\vec{X}_i^d$, a wrong key $\vec{K} = (\vec{K}_{l1}, \vec{K}_{l2})$ should be such that $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^F$ and $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^T$. The total number of ways in which we can select such a wrong key is $p \cdot (2^n - p)$, which is exactly the same as the one for type-0 Anti-SAT. Therefore, the subsequent analysis would be the same as the analysis for type-0 Anti-SAT and we can obtain the same lower bound λ_l as shown in Equation (9).

■

As seen in Equation (9), if we choose a g function such that p is either very low or very high then the SAT attack would at least require an exponential number of iterations in n. Since the key-size of Anti-SAT is $2n$, the number of SAT attack iterations is also an exponential number in the key-size of Anit-SAT when g is correctly configured. One possible choice of g is indicated in Fig. 5(a) where g is chosen to be a simple n-input AND gate. For AND gates $p = 1$ which clearly results in exponential complexity of SAT attack in n. Experimental results to indicate that shall be shown in Section VI-A. Moreover, we can see that the lower bound λ_l is tight when $p = 1$ or $p = 2^n - 1$. This is because that for a n-input Anti-SAT block, the total number of input combinations is 2^n so the number of iterations to find distinguishing inputs is upper-bounded $\lambda \leq 2^n$. This combined with the Equation (9) shows that the lower bound is tight when $p = 1$ or $p = 2^n - 1$.

C. Integrating A Circuit with Anti-SAT

When the Anti-SAT block is integrated into a circuit, a set of wires in the original circuit are connected to the inputs

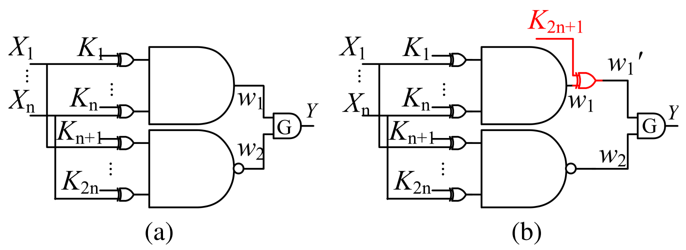

Fig. 5. Anti-SAT block design and obfuscation. (a) one possible construction of function g to ensure large number of SAT attack iterations; (b) an additional key-gate is inserted for functional obfuscation.

$\vec{X}$ of the Anti-SAT block and the output Y of the Anti-SAT block is integrated to a wire in the original circuit (as shown in Fig. 4(c)). If $\vec{X}$ are connected to wires that are highly correlated (*e.g.* two nets with identical logic), then the overall security of the block shall be reduced because less possible input combinations can occur at the inputs of the Anti-SAT block. The location for Y is also important. An incorrect key causes $Y = 1$ for some inputs (for type-0 Anti-SAT). This incorrect output must impact the overall functionality of the original circuit. Otherwise the logic will continue to function correctly despite of wrong key inputs. In conclusion, the best location of the Anti-SAT block is such that the inputs $\vec{X}$ are highly independent and Y has high observability at the POs (*i.e.,* changes in Y can be observed by the POs of the original circuit). Here we propose a *secure integration* method: n inputs of the Anti-SAT block $\vec{X}$ are connected to n PIs of the original circuit. The output Y is connected to a wire which is randomly selected from wires that have the top 30% observability. The randomness of the location of Y can assist in hiding the output wire of the Anti-SAT block and preventing it from being identified and nullified. The impact of the Anti-SAT integration location on the overall security shall be evaluated in the experiments (Section VI-A3).

D. *Combined with Conventional Logic Locking Techniques*

As noted before, conventional logic locking techniques as indicated in Fig. 1 try to avoid an unauthorized user who does not have a key from accessing the chip's functionality. They attempt to insert key gates in a way to force the chip to deviate substantially from the actual functionality whenever a wrong key is provided. These techniques are not immune to SAT attack (as noted in [1] and also indicated in our simulations). While our Anti-SAT block can provide provable measures to increasing the SAT attack complexity, they may not necessarily cause substantial deviation in the chip functionality for incorrect keys. Hence an unauthorized end user may still be able to use the chip correctly for "many" inputs (but not all). Therefore, conventional logic locking techniques need to be combined with our Anti-SAT block designs for achieving foolproof logic locking. Moreover, the key-gates inserted at the original circuit can make the Anti-SAT block less distinguishable with the original circuit. Without these key-gates in the original circuit, an attacker has less difficulty to locate the Anti-SAT block by inspecting the only key-inputs into the Anti-SAT block.

In this work, the original circuit is locked using the secure logic locking (SLL), a interference-based logic locking algorithm [6]. This technique has been shown to be secure against ATPG attack [6] while obfuscating the original functionality.

IV. Anti-SAT Block Obfuscation

Since the Anti-SAT block is independent (logically) of the locked circuit, it may be removed or nullified by an attacker if it is identified, thereby leaving only the locked circuit. Then, the SAT attack can be launched to unlock the circuit without the Anti-SAT block. These attacks are referred to as *removal attacks*. In this section, we first highlight the unique functional and structural attributes of the Anti-SAT block which might reveal its location. Then, we summarize potential removal attacks which exploit these attributes to identify and nullify the Anti-SAT block. Finally we investigate a unified obfuscation based on [12] which try to hide the functional and structural traces of the Anti-SAT block at the same time. Most importantly, we provide a rigorous proof showing that the obfuscation method would not weaken the resistance of the Anti-SAT block to SAT attack.

A. *Removal Attacks on Anti-SAT*

1) Functional Attributes Based Removal Attacks: In Anti-SAT, the logic blocks g and $\overline{g}$ have complementary functionality. An attacker can simulate the circuit and find potential complementary pairs of signals leading to potential identification of the Anti-SAT block. Moreover, in order to guarantee exponential number of SAT attack iterations, the function g shall be configured to have very small on-set size p. Assuming $p = 1$, the outputs of $g/\overline{g}$ would be 0/1 for most of the time even when wrong keys are provided for the Anti-SAT block. In other words, the outputs of g and $\overline{g}$ will have very high signal skews of opposite polarities. This functional attribute is exploited by Signal Probability Skew (SPS) attack [15]. The basic idea of SPS attack is to find a gate whose inputs have high signal skews of opposite polarities. Such gate is of great possibility to be the output gate G in the Anti-SAT block as shown in Fig. 5(a). Thus, the functional attribute of Anti-SAT provides a hint for attackers to identify the location of the gate G and its output signal Y of the Anit-SAT block.

2) Structural Attributes Based Removal Attacks: In the Anti-SAT block, the internal wires in g and $\overline{g}$ do not have connections with the locked circuit. This makes the Anti-SAT block a relatively isolated and separable structure. When the size of the Anti-SAT block is roughly known, it's possible for an attacker to utilize a partitioning algorithm to partition the whole circuit into two parts while ensuring that small partition has about the same size as the Anti-SAT block. If a large portion of gates of the Anti-SAT block is moved to the small partition, then the attacker will have less difficulty to identify the Anti-SAT block.

B. *Unified Anti-SAT Obfuscation Technique*

To mitigate the vulnerability of Anti-SAT to various removal attacks, we apply a unified obfuscation technique based on

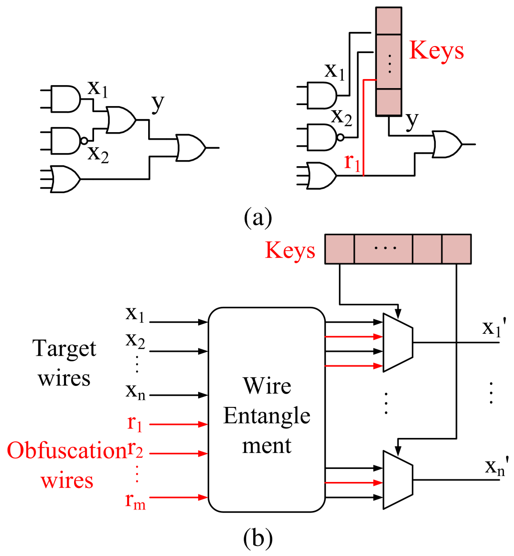

Fig. 6. Design withholding and entanglement technique: (a) design withholding and (b) wire entanglement.

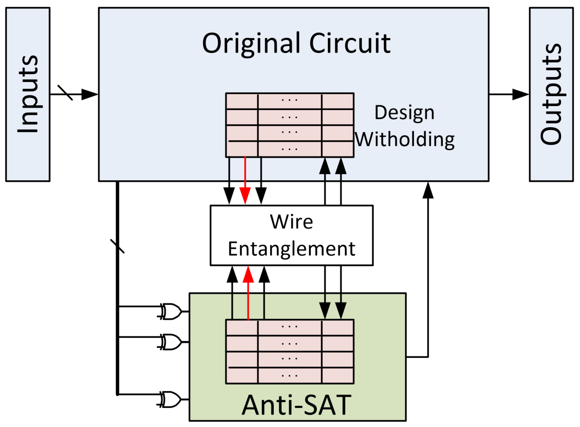

Fig. 7. Anti-SAT obfuscation based on design withholding and wire entanglement.

design withholding and entanglement as proposed in [12]. Fig. 6 illustrate the basic idea of design withholding and entanglement.

In design withholding (Fig. 6(a)), a portion of design is replaced with a set of LUT's to ensure that the original design detail is not available to the untrusted foundry. Hence, design withholding technique can be used to hide both the functionality and implementation detail of the Anti-SAT so that removal attacks that exploits the functionality attributes (*e.g.* signal skews) cannot be performed.

Design entanglement is another obfuscation technique that aims at obfuscating the interconnect structure of an IC design by using a wire-entanglement module as shown in Fig. 6(b). As shown, the basic idea of wire-entanglement module is to entangle n target wires with r obfuscation wires using MUX-based interconnect network. When the selection bits of the MUXes are correctly configured, the wire-entanglement module will represent the original interconnection. The wire-entanglement module is useful for obfuscating the interconnect structure between the Anti-SAT block and the original netlist. The target wires and obfuscation wires can be selected from both the original netlist and the Anti-SAT block and a correct key into the MUXes selection bits will recover the original interconnection. With the wire-entanglement module, the interconnections between the Anti-SAT block and the locked circuit will be increased and it's difficult for an attacker to partition and isolate the Anti-SAT block from the locked circuit.

Fig. 7 illustrates the overall obfuscation for Anti-SAT based on design withholding and wire entanglement. The design withholding technique is used to hide the functionality of the Anti-SAT block and part of the original netlist so that SPS attack which is based on signal skew analysis cannot be performed. Moreover, wire-entanglement technique is used to obfuscate the interconnection between the original circuit and the Anti-SAT block to prevent the partitioning-based attack. These two obfuscation techniques will inevitably increase the performance overhead. However, we present it as the first unified obfuscation technique to make various removal attacks on the Anti-SAT block harder. A more light-weight solution may be explored in future research.

C. SAT-attack Resistance of Anti-SAT After Obfuscation

In Section IV-B, a unified obfuscation technique for Anti-SAT block based on [12] is discussed. It basically obfuscates the Anti-SAT block by adding additional logic gates and key-inputs. *Here we use a rigorous proof to shows that the resistance of Anti-SAT block would not be weakened after when obfuscation technique is applied. In other words, adding addition key-gates and key-inputs will not reduce the number of SAT attack iterations required to decipher the Anti-SAT block.*

The outline of the proof is as follows:

1) We show that after adding one extra key-gate at internal wires of the Anti-SAT, the number of SAT-attack iterations required to unlock the Anti-SAT block would not be reduced.
2) We then show that the proof in 1) can be extended to the case when n_{obf} extra key-gate are added to the Anti-SAT block for obfuscation.

Now we first show that adding one addition key-gate for obfuscation will not reduce the number of SAT attack iterations required for unlocking the Anti-SAT. Without loss of generality, we insert the extra key-gate as shown in Fig. 5(b)

Theorem 2. *Assuming a new key-gate K_{2n+1} is inserted into the Anti-SAT block (with $p = 1$) for obfuscation (as shown in Fig. 5(b)), the number of SAT attack iterations needed by the SAT attack to decipher the correct key will remain to be 2^n.*

Proof: To show that the total number of SAT attack iterations would still be 2^n, here we show that any input combination (with decimal value from 0 to 2^n-1) is indeed a distinguishing input $\vec{X}_i^d$. It means that each of these input combinations can identify a unique set of wrong keys that can *only* be identified by it. In other words, to identify all wrong key combinations, the SAT attack requires 2^n DIOs (*i.e.,* 2^n iterations). The outline of the proof is as follows. We first derive an equation

	$\vec{X}_1^d = 00\ldots00,\ \vec{Y}_1^d = 1$		
	K_{l1}	K_{l2}	K_{2n+1}
WK_1	11...111	00...000 00...001 ... 11...101 11...110 ~~11...111~~	0

(a)

	$\vec{X}_2^d = 00\ldots01,\ \vec{Y}_2^d = 1$		
	K_{l1}	K_{l2}	K_{2n+1}
WK_2	11...110	00...000 00...001 ... 11...101 ~~11...110~~ 11...111	0

(b)

Fig. 8. Analysis of SAT attack on Anti-SAT in Fig. 5(b) when an extra key-gate is inserted for functional obfuscation. (a) Wrong key combinations (with $\vec{K}_{2n+1} = 0$) identified when $\vec{X}_i^d = (00...00)$; (b) Wrong key combinations (with $\vec{K}_{2n+1} = 0$) identified when $\vec{X}_i^d = (00...01)$.

(Equation (12)) which represents the wrong key combinations that can be identified by a DIO $(\vec{X}_i^d, \vec{Y}_i^d)$. Based on this equation, we figure out a one-to-one matching between an input combination (anyone from 0 to $2^n - 1$) and a set of unique wrong key combinations (Equation (15)). Therefore, every input combination (from 0 to $2^n - 1$) is a distinguishing input and the SAT attack requires 2^n iterations to use all DIOs to identify all the wrong key combinations.

Let's first derive the equation which represents the wrong key combinations that can be identified by a DIO $(\vec{X}_i^d, \vec{Y}_i^d)$. Notice that for any input combinations (including the distinguishing inputs $\vec{X}_i^d$), the correct output (when provided a correct key) is 0 for type-0 Anti-SAT. Therefore, a wrong key combination which is identified by $(\vec{X}_i^d, \vec{Y}_i^d)$ must result in incorrect output $\vec{Y}_i^d = 1$. This condition ($\vec{X}_i^d$, $\vec{Y}_i^d = 1$) is equivalent to:

$$\begin{aligned}&[(K_{2n+1} = 0) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 1) \wedge (\overline{g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 1})]\\ \vee&[(K_{2n+1} = 1) \wedge (g(\vec{X}_i^d \oplus \vec{K}_{l1}) = 0) \wedge (\overline{g(\vec{X}_i^d \oplus \vec{K}_{l2}) = 1})]\end{aligned} \tag{11}$$

This is equivalent to

$$\begin{aligned}&[(K_{2n+1} = 0) \wedge (\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T) \wedge (\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F)]\\ \vee&[(K_{2n+1} = 1) \wedge (\vec{X}_i^d \oplus \vec{K}_{l1} \in L^F) \wedge (\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F)]\end{aligned} \tag{12}$$

Basically, Equation (12) indicates that any key combinations satisfying this equation would be a wrong key combination because it will result in a wrong output $Y_i^d = 1$ for a given distinguishing input $\vec{X}_i^d$. Notice that because the function g in Fig. 5(b) is an n-input AND gate, we have

$$L^T = \{(11...11)\}, \quad L^F = \mathbb{B}^n \setminus L^T \tag{13}$$

where $\mathbb{B}^n$ is the set of all n-bit boolean vectors, and $\mathbb{B}^n \setminus L^T$ means every combinations of an n-bit Boolean vector except (11...11).

Now we show the one-to-one matching between an input combination (from 0 to $2^n - 1$) and a set of unique wrong key combinations. Based on Equation (12), we can see that when $K_{2n+1} = 0$, to satisfy Equation (12), we need to ensure $\vec{X}_i^d \oplus \vec{K}_{l1} \in L^T$ and $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$. Since L^T has only one vector (11...11), for any $\vec{X}_i^d$, we only have one way of selecting $\vec{K}_{l1}$ to make $\vec{X}_i^d \oplus \vec{K}_{l1} = (11...11)$, that is $\vec{K}_{l1} = \neg\vec{X}_i^d$ (bit-wise negation), *i.e.,*

$$\vec{K}_{l1}[j] = \neg\vec{X}_i^d[j], \quad j = 1...n \tag{14}$$

On the other hand, since $L^F = \mathbb{B}^n \setminus L^T$, for any $\vec{X}_i^d$, we have $2^n - 1$ ways to select $\vec{K}_{l2}$ such that $\vec{X}_i^d \oplus \vec{K}_{l2} \in L^F$, those are $\vec{K}_{l2} \in \mathbb{B}^n \setminus \neg\vec{X}_i^d$.

Therefore, when $K_{2n+1} = 0$, the set of wrong key combinations identified by $(\vec{X}_i^d, Y_i^d)$ is:

$$(\vec{K}_{l1} = \neg\vec{X}_i^d, \vec{K}_{l2} \in (\mathbb{B}^n \setminus \neg\vec{X}_i^d), K_{2n+1} = 0) \tag{15}$$

To illustrate above equation, Fig. 8 shows the wrong key combinations (with $\vec{K}_{2n+1} = 0$) identified by two DIOs $(\vec{X}_1^d = 00...00, Y_1^d = 1)$ and $(\vec{X}_2^d = 00...01, Y_2^d = 1)$. From Equation (15) and Fig. 8, we can see that since $\vec{K}_{l1} = \neg\vec{X}_i^d$, there exists an one-to-one matching between each pair of $\vec{X}_i^d$ and $\vec{K}_{l1}$. In other words, any $\vec{X}_i^d$ value (from 0 to $2^n - 1$) can identify a unique set of wrong key combinations in a form of Equation (15). It's unique because that $\vec{K}_{l1} = \neg\vec{X}_i^d$ and different $\vec{X}_i^d$ would result in different $\vec{K}_{l1}$. Therefore, every input combination (from 0 to $2^n - 1$) is a distinguishing input because it can identify a unique set of wrong key combinations that can only be identified by it. Thus, the SAT attack requires 2^n DIOs (*i.e.,* 2^n iterations) to identify all wrong key combinations.

■

Theorem 3. *Assuming n_{obf} new key-gates are inserted into the Anti-SAT block (with $p = 1$) for obfuscation, the number of SAT attack iterations needed by the SAT attack to decipher the correct key will be at least 2^n.*

The proof for Theorem 2 shows that after adding a new key-gate to the Anti-SAT block for obfuscation(Fig. 5(b)), the number of SAT attack iterations remains to be 2^n. Notice that this conclusion is also true when n_{obf} additional key-gates are inserted to the Anti-SAT for obfuscation Let's denote the extra keys for obfuscation as $\vec{K}_{obf}$ and its correct key is $\vec{K}_{obf}^C$. Based on Eqn. (15), we can conclude that any Anti-SAT input combination is a distinguishing input $\vec{X}_i^d$ because it can identify a unique set of wrong keys which is:

$$(\vec{K}_{l1} = \neg\vec{X}_i^d, \vec{K}_{l2} \in (\mathbb{B}^n \setminus \neg\vec{X}_i^d), \vec{K}_{obf} = \vec{K}_{obf}^C) \tag{16}$$

and this set of wrong keys can only be identified by this input $\vec{X}_i^d$. Hence the SAT attack requires at least 2^n DIOs (*i.e.,* 2^n iterations) to identify all wrong key combinations. Therefore, adding additional key-gates at different locations for obfuscation will not weaken the Anti-SAT's resistance to the SAT attack.

■

V. Design Flow of Anti-SAT based Logic Locking

Fig. 9 shows the overall design flow of Anti-SAT based logic locking. Firstly, an original netlist is locked using conventional logic locking techniques to form a locked netlist. The key-gates inserted at the original netlist can cause substantial

TABLE I
IMPACT OF p ON THE SECURITY LEVEL OF ANTI-SAT (WHEN $n = 16$)

	p	1	81	243	2187	30375	63349	65293	65455	65535
Type-0 Anti-SAT	# Iterations	-	10675	4760	901	273	898	4647	-	-
	Time (s)	timeout	16555.8	8746.12	174.743	3.24	307.104	12932.3	timeout	timeout
Type-1 Anti-SAT	# Iterations	-	-	4853	877	285	881	4691	-	-
	Time (s)	timeout	timeout	3559.96	55.108	3.148	187.896	1048.19	timeout	timeout

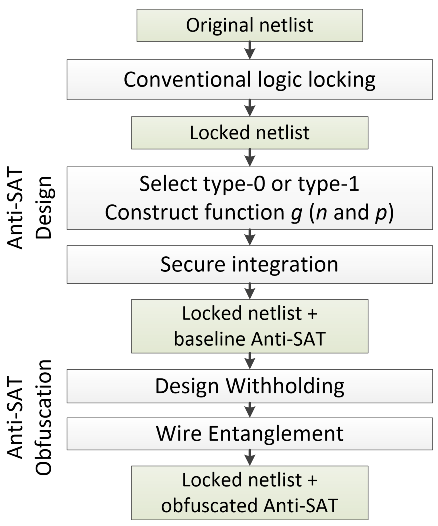

Fig. 9. Design flow of Anti-SAT based Logic Locking.

TABLE II
IMPACT OF n ON THE SECURITY LEVEL OF ANTI-SAT (WHEN $p = 1$)

	n	8	10	12	14	16
Type-0 Anti-SAT	# Iterations	255	1023	4095	16383	-
	Time (s)	1.14569	20.024	324.727	4498.03	timout
Type-1 Anti-SAT	# Iterations	255	1023	4095	16383	-
	Time (s)	1.06	14.612	273.1	3658.76	timeout

TABLE III
COMPARISON BETWEEN SECURE AND RANDOM INTEGRATION.

	n	8	12	16
Random	Avg. # Iteration	151	1748	11461
	Avg. Time (s)	1.4296	162.529	10272.4
Secure	# Iteration	255	4095	-
	Time (s)	3.452	759.924	timeout

functionality deviation for incorrect keys. Besides, it makes Anti-SAT block less distinguishable with the original circuit since key-gates are now inserted at both the original circuit and the Anti-SAT block. Secondly, an Anti-SAT block is designed by selecting type-0 or type-1 and tuning n and p of the logic block g. In this work, we select type-0 Anti-SAT and construct an **n-bit baseline Anti-SAT block (n-bit BA)** using an n-input AND gate ($p = 1$) as the logic block g and an n-input NAND gate as $\overline{g}$ to ensure large number of iterations. However notice that this is not the only possible choice for g and $\overline{g}$. As we have shown in Section III-B, other function g that has sufficiently large n and sufficiently small (or large) p can also guarantee large number of iterations. Then, the baseline Anti-SAT is integrated into the locked netlist using secure integration (as described in Section III-C). After that, the Anti-SAT block is obfuscated to counter potential removal attacks using design withholding and wire entanglement techniques.

VI. EXPERIMENTS AND RESULTS

In this section, we evaluate the security level of our proposed Anti-SAT blocks. The security level is evaluated by the number of *SAT attack iterations* as well as the *execution time* to infer the correct key. SAT attack tools and benchmarks used are from [1]. The SAT attack tool uses the Lingeling [18] SAT solver. The CPU time limit is set to 10 hours as [1]. The experiments are running on an Intel Core i5-2400 CPU with 16GB RAM.

A. Anti-SAT Block Design

We firstly evaluate the security level of an Anti-SAT block (Fig. 4) with respect to the on-set size p and the input-size n of function g. Both type-0 Anti-SAT (Fig. 4(a)) and type-1 Anti-SAT (Fig. 4(b)) are evaluated. Then, we integrate the Anti-SAT blocks into a circuit and validate the effectiveness of our proposed secure integration approach by comparing it with a random integration approach.

1) On-set size p: As shown in Equation (9), the lower bound of SAT attack iterations λ_l required to unlock the Anti-SAT block is related to both n and p. If n is fixed, λ_l is maximized when $p \to 1$ or $p \to 2^n - 1$. Table I illustrates the impact of p on the security level of 16-bit Anti-SAT blocks (type-0 and type-1). For both types of Anti-SAT, when $p \to 1$ and $p \to 2^{16} - 1 = 65535$, the SAT attack algorithm fails to unlock the Anti-SAT block in 10 hours. This is because that it requires a large number of iterations to rule out all the incorrect key combinations. As $p \to 2^{16}/2$ (the worst case), the SAT attack begins to succeed using less and less iterations and execution time for both types of Anti-SAT. This result validates that for a fixed n, when p is very small or very large, λ will be large and the SAT attack will fail within a practical time limit.

2) Input-size n: As shown in Equation (9), λ_l is an exponential function of n when p is very low ($p \to 1$) or very high ($p \to 2^n - 1$). Table II shows the exponential relationship between λ and n when $p = 1$ for type-0 and type-1 Anti-SAT block. It can be seen that as n increases, the simulated SAT iterations and execution time grows exponentially. Besides, the number of iterations validates that the lower bound λ_l is tight when $p = 1$, as discussed in Section III-B.

Comparing type-0 and type-1 Anti-SAT blocks, we can see that although the number of SAT attack iterations needed to decrypt these two types of Anti-SATs (for same n and p) are almost the same, the execution time varies. The type-0 Anti-SAT generally requires more CPU time than the type-1 Anti-SAT. This could be related to the SAT solver used, which might solve the SAT formula of type-1 Anti-SAT more

TABLE IV
BENCHMARK INFORMATION OF 6 CIRCUITS FROM ISCAS85 AND MCNC.

Circuit	#PI	#PO	#Gates	Key-size	
				SLL (5%)	n-bit BA
c1355	41	32	546	29	2n
c1908	33	25	880	46	
c3540	50	22	1669	86	
dalu	75	16	2298	119	
des	256	245	6473	336	
i8	133	81	2464	130	

efficiently. Therefore, we use type-0 Anti-SAT to construct the n-bit BAs and n-bit OAs in the following experiments.

3) Secure Integration of Anti-SAT: Here we compare two approaches of integrating the Anti-SAT block with the original circuit, namely *secure integration* and *random integration*. For the *secure integration*, n inputs of the Anti-SAT block $\vec{X}$ are connected to n PIs of the original circuit. The output Y is connected to a wire which is randomly selected from wires that have the top 30% observability. The randomness of the location of Y can assist in hiding the output of the Anti-SAT block. For the *random integration*, the inputs $\vec{X}$ are connected to random wires of the original circuit, and the output Y is connected to a random wire. For both cases, the wire for Y has a later topological order than that of the wires for $\vec{X}$ to prevent combinational loop. Table III compares two integration approaches when three n-bit BA ($n = 8, 12, 16, p = 1$) are integrated into the c1355 circuit from ISCAS85. It can be seen that for three Anti-SAT blocks, secure integration is better than random integration as the former requires more iterations ($\sim 2\times$) and execution time ($\sim 3\times$) for the SAT attack algorithm to reveal the key. Therefore, in the following experiments, we adopt the secure integration as the way to integrate the Anti-SAT block into a circuit.

B. Anti-SAT Block Application

We evaluate the security level of the Anti-SAT block when it's applied to 6 circuits of different sizes from ISCAS85 and MCNC benchmark suites. The benchmark information is shown in Table IV. We compare two logic locking configurations as follows:

- **SLL**: The original circuit is locked using the secure logic locking (SLL), a interference-based logic locking algorithm [6]. This technique has been shown to be secure against ATPG attack [6] while obfuscating the original functionality.
- **SLL (5%) + n-bit BA**: In this configuration, the original circuit is locked with SLL with 5% area overhead. Besides, we integrate an n-bit BA into the locked circuit using the secure integration (described in Section III-C). For an n-bit BA, its key-size is $k_{BA} = 2n$ because $2n$ keys are inserted at the inputs of g and $\overline{g}$.

We compare the security level of two configurations when the same number of keys are used in each configuration. We investigate the sensitivity of SAT attack complexity on the increase of key-size. For SLL, the extra key-gates are inserted to the original circuit. For SLL(5%) + n-bit BA/OA, the extra key-gates are used in the Anti-SAT block and increasing the key-size also indicates increasing the input-size n because we construct the BA with $k_{BA} = 2n$. In this experiment, we experiment the n-bit BA of input-size $n_{BA} = 8, 10, 12, 14, 16, 18, 20$. The key-sizes are shown in Table IV.

The SAT attack results on two configurations w.r.t increasing key-size are shown in Fig. 10. For each benchmark, the top figure shows the SAT attack execution time and the bottom figure shows the number of SAT attack iterations, both in log scale. It can be seen that for SLL, increasing the key-size cannot effectively increase SAT attack complexity. For all benchmarks locked with SLL, they can be easily unlocked using at most 67 iterations and 1070.85 seconds. On the other hand, when the Anti-SAT blocks are integrated, the SAT attack complexity increases exponentially with the key-size in the Anti-SAT block. Finally, we can see that for all benchmarks, the SAT attack fails to unlock the circuits within 10 hours when a 16-bit BA is integrated (as shown by the fifth data point w.r.t. x-axis).

C. Performance Overhead of the Anti-SAT Block

In our construction, a n-bit baseline Anti-SAT block consists of logic blocks g and $\overline{g}$, an AND2 gate, $2n+1$ XOR/XNOR gates. These extra logic gates will introduce performance overhead such as area, power and delay. Different implementation of g and $\overline{g}$ will result in different overhead. The performance overhead will be increased when sub-optimal synthesis is utilized to obfuscate the Anti-SAT block. In our experiments, we utilize a n-bit AND gate and a n-bit NAND gate to implement the function g and $\overline{g}$, each consists of $n-1$ AND2 gates. The estimated area for a n-bit BA is $4n$ additional gates while the number of SAT attack iteration required is 2^n. It can be seen that a slight increase in area overhead of the Anti-SAT block can result in exponential increase in SAT attack's computation complexity. Besides, the size of the Anti-SAT block does not scale with the benchmark size. It only scales with the attacker's computation power.

To counter removal attacks, we investigate both the design withholding and entanglement techniques. These two obfuscation techniques will inevitably increase the performance overhead. However, we present it as the first unified obfuscation technique to make various removal attacks harder. A more light-weight solution may be explored in future research.

VII. RELATED WORK

A. SAT Attack Resilient Logic Locking

To increase the execution time of SAT attack, [19] proposed to add an AES circuit (with a fixed AES key) into a locked circuit. As the AES circuit (behaves as an one-way function) is hard to be solved by a SAT solver, the SAT attack will fail to find a satisfiable assignment within a practical time limit. Although this approach is effective, the AES circuit leads to a significant performance overhead since a standard AES circuit implementation requires a large number of gates [20]. On the contrary, we have shown that the Anti-SAT would result in a much smaller overhead. A slight increase in the overhead for Anti-SAT can make the SAT attack exponentially harder.

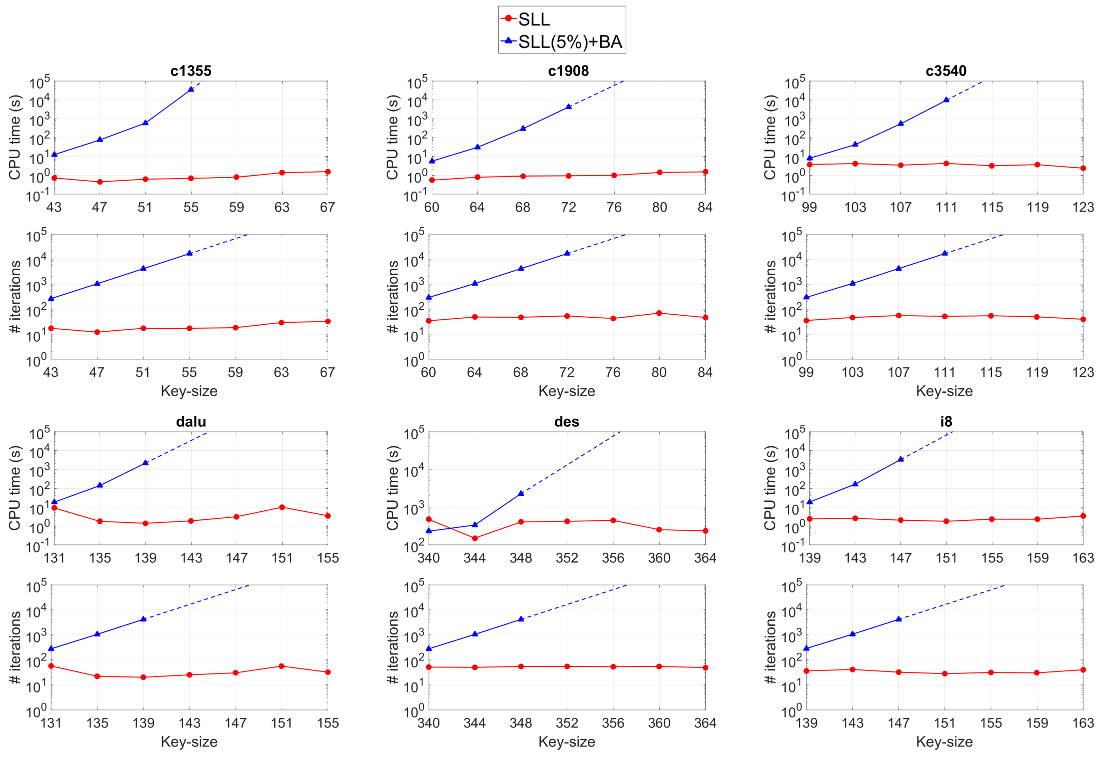

Fig. 10. SAT attack results on 6 benchmarks with three logic locking configurations: SLL and SLL(5%) + n-bit BA. Timeout is 10 hours (3.6×10^4 s). The dashed lines are the curve fitting results when the SAT attack has time-outed after certain key-size.

In [21], a SAT attack resistant logic locking (SARLock) technique was proposed which can make the attack iterations grow exponentially in key-size. Though [21] and this work are developed independently, they share a similar idea to defend the SAT attack, which is to integrate an additional SAT-attack resistant logic block into the locked circuit. Here we compare two techniques and highlight the advantages of our proposed Anti-SAT based logic locking. Firstly, the Anti-SAT block can be configured differently (by tuning the functionality of logic block g) while the SARLock has only one fixed configuration. We can construct different Anti-SAT blocks by tuning the parameters n and p to achieve the same lower bound of SAT attack iterations. The AND/NAND configuration is a simple illustration. The flexibility in configuration is a great advantage because we don't require a fixed structure like SARLock, and hence its less likely to be identified. Secondly, in this work we have studied the impact of locations for integrating the Anti-SAT block and proposed a secure integration method (Section III-C) which takes into account the need of hiding the location of output wire Y of the Anti-SAT block. Last but not least, we have proposed both functionality and structural obfuscation techniques (Section IV) to protect the Anti-SAT from the removal attacks such as the SPS attack and the partitioning based attack.

B. SAT Attack on IC Camouflaging

IC camouflaging is a reverse-engineering prevention technique that hides a circuit's functionality with camouflaging cells. Camouflaging cells are logic cells that look alike but have different functionalities (*e.g.* NAND, NOR and XOR). Since each camouflaging cell can be modeled as a key-controlled MUX gate with a set of possible functionalities (*e.g.* NAND, NOR and XOR) as inputs, SAT attack can also be applied to recover the functionality of the camouflaging cells [22], [23]. To counter the SAT attack, various countermeasures have been proposed [24], [25], which aim at increasing the de-camouflaging effort exponentially harder in the number of camouflaged gates.

- For [24], although only one n-input AND tree is inserted, there exists a worst case such that only one iteration can eliminate all wrong key combinations. This happens when the distinguishing input for the AND tree is ($\vec{X}$=11...11). This will eliminate all wrong key combinations because any key that is not (00...00) will result in incorrect output 0. To avoid having this worst case, we choose to use a complementary g and $\overline{g}$ structure.
- For [25], the basic idea is to add or remove a min-term in the original circuit and then add a logic block (called CamoFix) to provide a complementary min-term to fix it. This is basically the complementary structure as the Anti-SAT. A min-term is basically an AND tree. The technique might only need to added one AND tree. However, to add or remove a min-term in the original circuit, the original circuit shall be re-synthesized and it might result in extra overhead.

In terms of removal attack, both [24] and [25] would subject to removal attacks if the AND tree structure is not obfuscated. For [24], if the output of the AND tree is identified, it can be simply nullified. For [25], removal attack is harder because simply nullifying the AND tree output will result in error for one min-term. But since the error is only for one min-term it might be tolerable for many applications. Therefore, some degree of obfuscation to hide the AND tree structure is necessary for all three techniques.

VIII. Conclusion

In this paper, we present a circuit block called Anti-SAT to mitigate the SAT attack on logic locking. We show that the iterations required by the SAT attack to reveal the correct key in the Anti-SAT block is an exponential function of the key-size in the Anti-SAT block. The Anti-SAT block is integrated to a locked circuit to increase its resistance to the SAT attack. A unified obfuscation techniques has been proposed to protect the Anti-SAT block from removal attacks such as the SPS attack and the partitioning based attack. Overall, our proposed Anti-SAT based logic locking can effectively thwart the SAT attack, the SPS attack and the partitioning based attack.

Acknowledgments

This work was supported by NSF under Grant No. 1223233 and AFOSR under Grant FA9550-14-1-0351.

References

[1] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 137–143.

[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[3] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.

[4] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, 2010.

[5] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proceedings of the conference on Design, Automation and Test in Europe*. ACM, 2008, pp. 1069–1074.

[6] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 83–89.

[7] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *Computers, IEEE Transactions on*, vol. 64, no. 2, pp. 410–424, 2015.

[8] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2014, pp. 270–277.

[9] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in IC piracy with test-aware logic locking," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 34, no. 6, pp. 961–971, 2015.

[10] Y.-W. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *2015 16th Latin-American Test Symposium (LATS)*. IEEE, 2015, pp. 1–6.

[11] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design & Test of Computers*, 2010.

[12] S. Khaleghi, K. Da Zhao, and W. Rao, "IC piracy prevention via design withholding and entanglement," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 821–826.

[13] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Proceedings of the conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2014, p. 243.

[14] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*. IEEE, 2014, pp. 49–54.

[15] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-sat," Cryptology ePrint Archive, Report 2016/896, 2016, http://eprint.iacr.org/2016/896.

[16] R. S. Chakraborty and S. Bhunia, "Harpoon: an obfuscation-based soc design methodology for hardware protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[17] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 127–146.

[18] A. Biere, "Lingeling, plingeling and treengeling entering the sat competition 2013," *Proceedings of SAT Competition*, vol. 2013, 2013.

[19] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2015.

[20] HelionTechnology, "High performance AES (Rijndael) cores for ASIC," http://www.heliontech.com/downloads/aes_asic_helioncore.pdf, 2015.

[21] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 236–241.

[22] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes." in *22nd Annual Network and Distributed System Security Symposium (NDSS)*, 2015.

[23] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental sat-based reverse engineering of camouflaged logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[24] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably secure camouflaging strategy for ic protection," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 28.

[25] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Camoperturb: secure ic camouflaging for minterm protection," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 29.