Mitigating SAT Attack on Logic Locking

Yang Xie and Ankur Srivastava

University of Maryland, College Park, USA
{yangxie, ankurs}@umd.edu

Abstract. Logic locking is a technique that has been 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 [12] even for a reasonably large number of keys. This attack iteratively solves SAT formulas which progressively eliminate the incorrect keys till the circuit unlocked. In this paper, we present a circuit block (referred to as Anti-SAT block) to thwart the SAT attack. We show 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. Through our experiments, we illustrate the effectiveness of our approach to securing modern chips fabricated in untrusted foundries.

Keywords: Logic Locking; SAT Attack; Hardware IP Protection

1 Introduction

Outsourced fabrication of integrated circuit (IC) enables IC design companies to access advanced semiconductor technology at a low cost. Although it is cost-effective, the outsourced design faces various security threats since the offshore foundry might not be trustworthy. Without close monitoring and direct control, the outsourced designs are vulnerable to various attacks such as Intellectual Property (IP) piracy [10] and counterfeiting [3]. The malicious foundry can reverse-engineer a GDSII layout file to obtain its gate-level netlist and claim the ownership of the hardware IP design, or it can overbuild the IC and sell illegal copies into the market. These security threats (also known as supply chain attacks) pose a significant economic risk to most commercial IC design companies.

Logic locking 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

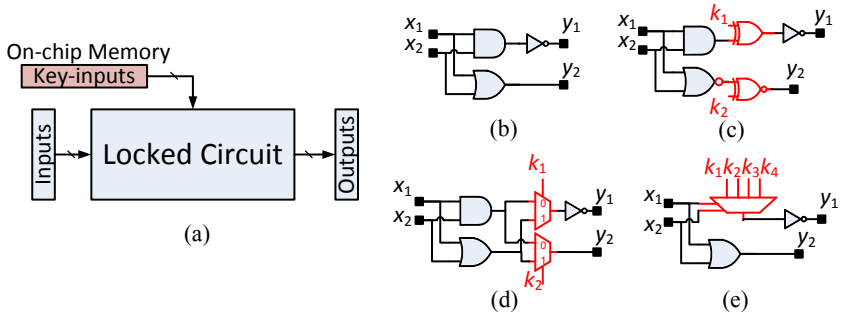

Fig. 1. Logic locking techniques: (a) Overiew; (b) An original netlist; (c) XOR/XNOR based logic locking; (d) MUX based logic locking; (e) LUT based logic locking.

its original functionality, as shown in Fig. 1. The key-inputs are connected to the on-chip memory and 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 internal signals of a running chip, a tamper-proof chip protection shall be implemented. 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 [8, 9, 11], MUX based logic locking [7, 9, 13] and Look-Up-Table (LUT) based logic locking [1, 5, 6], as shown in Fig. 1 (b-e). 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, such as fault-analysis based insertion [9] and interference-analysis based insertion [8]. *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.*

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. To learn the correct key, Subramanyan *et al.* [12] proposed a satisfiability checking based attack (*SAT attack*) algorithm that can effectively break most logic locking techniques proposed in [1, 2, 8, 9, 11] within a few hours even for a reasonably large number of keys. The insight of SAT attack is to infer the correct key using a small number of carefully selected input patterns and their correct outputs observed from an activated functional chip (which can be obtained from the open market). This set of correct input/output pairs together ensures that only the correct key will be consistent with these observations. The process of finding such input/output pairs is iteratively formalized as a sequence of SAT formulas that can be solved by state-of-the-art SAT solvers. In each of these iterations, the SAT formulation rules out a bunch of wrong key combinations till it reaches a point where all the wrong keys have been removed.

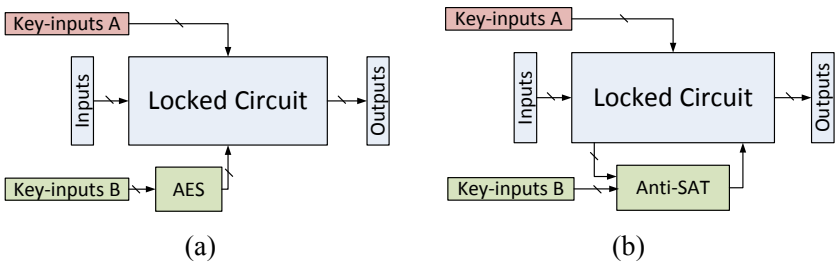

Fig. 2. SAT attack mitigation techniques: (a) Adding an AES circuit to increase the time for solving a SAT formula [14]; (b) Adding our proposed Anti-SAT circuit block to increase the number of SAT attack iterations.

The SAT attack is powerful as it guarantees that upon termination it can always reveal the correct key. This guarantee can't be achieved by other attacks on logic locking such as the EPIC attack [7]. Hence in this paper we focus on the SAT attack on logic locking.

Since the SAT attack needs to iteratively solve a set of circuit-based SAT formulas to reveal the correct key, its efficiency is determined by two aspects: a) the execution time for solving a SAT formula in one iteration and b) the number of iterations required to reveal the correct key. The first aspect depends on whether a locked circuit is easily solvable by a SAT solver (*i.e.,* finding a satisfiable assignment for the SAT formula based on this circuit). Based on this idea, Yasin *et al.* [14] proposed adding an AES circuit (with a fixed AES key) to enhance a locked circuit's resistance to the SAT attack. The insight underlying this proposal is shown in Fig. 2(a). A portion of key-inputs is firstly connected to the AES inputs and the outputs of the AES are the actual key-inputs into the locked circuit. As the AES circuit is hard to be solved by a SAT solver, the SAT attack will fail to find a satisfiable assignment for the SAT attack formula within a practical time limit. Although this approach can effectively increase the SAT attack execution time, the AES circuit results in a significant performance overhead since a standard AES circuit implementation requires a large number of gates [4]. This makes the approach in [14] impractical.

In this paper, we propose a relatively lightweight circuit block (referred to as Anti-SAT block) that can be embedded into a design to efficiently mitigate the SAT attack. The basic structure of our Anti-SAT block is shown in Fig. 2(b). 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 key-learning of 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.* Therefore, it can be integrated into a design to enhance its resistance to the SAT attack. The contributions of this paper are summarized 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 is integrated into a circuit to increase its resistance to the SAT attack. To prevent the Anti-SAT block from being identified (and removed by an attacker) we apply obfuscation techniques to hide the functionality and structure of the Anti-SAT block.
– Rigorous analysis and experiments on 6 circuits from ISCAS85 and MCNC benchmark suites have been conducted to validate the effectiveness of our proposed technique against the SAT attack.

2 Background: SAT Attack

2.1 Attack Model

The SAT attack model [12] 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 design details. The locked netlist is represented as $\boldsymbol{Y} = f_l(\boldsymbol{X}, \boldsymbol{K})$ with primary inputs $\boldsymbol{X}$, key inputs $\boldsymbol{K}$ and primary outputs $\boldsymbol{Y}$. Its SAT formula in conjunctive normal form (CNF) is represented as $C(\boldsymbol{X}, \boldsymbol{K}, \boldsymbol{Y})$.
– 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 $\boldsymbol{Y} = eval(\boldsymbol{X})$.

2.2 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 (I/O) pairs*. Each distinguishing I/O pair 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 distinguishing I/O pairs must be the correct key. The crux of the SAT attack is to find this set of distinguishing I/O pairs by solving a sequence of SAT formulas.

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

Definition 2: (Distinguishing input/output 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 distinguishing I/O pair $(\boldsymbol{X}_i^d, \boldsymbol{Y}_i^d)$, if it can identify a "unique" subset of wrong key combinations that cannot be identified by the previous $i-1$ distinguishing I/O pairs, *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 distinguishing I/O pair at i-th iteration.

The crux of the SAT attack algorithm relies on finding the distinguishing I/O pairs iteratively to identify *unique* wrong key combinations (see definition 2) until no further ones can be found. At this point, the set of all distinguishing I/O pairs *together* identifies all wrong key combinations thereby unlocking the circuit. Then, the correct key is the one that satisfies the following SAT formula G:

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

where $(\boldsymbol{X}_i^d, \boldsymbol{Y}_i^d)$ is the distinguishing I/O pair from i-th iteration and λ is the total number of iterations. Basically it finds a key $\boldsymbol{K}$ which satisfies the correct functionality for all the identified distinguishing I/O pairs. This must be the correct key since no other distinguishing I/O pairs exist (see definition 2).

Take the XOR/XNOR based locked circuit in Fig. 1(c) as an example. At first iteration, the I/O pair $(\boldsymbol{X}_1^d, \boldsymbol{Y}_1^d) = (00, 10)$ is a distinguishing I/O pair because it can rule out wrong key combinations $\boldsymbol{K} = (01)$, (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 $\boldsymbol{K} = (00)$. In general, a small number of correct I/O pairs (compared to all possible I/O pairs) are usually enough to infer the correct key [12]. As a result, the SAT attack is efficient because it only requires a small number of iterations to find these distinguishing I/O pairs.

2.3 Attack Algorithm

As noted above, the central theme of SAT attack algorithm is to iteratively find distinguishing I/O pairs till no new ones can be found. To find such distinguishing I/O pairs, the SAT attack algorithm iteratively formulates a SAT formula that can be solved by SAT solvers. The SAT formula F_i at i-th iteration is:

$$\begin{aligned} F_i :=& C(\boldsymbol{X}, \boldsymbol{K}_1, \boldsymbol{Y}_1) \wedge C(\boldsymbol{X}, \boldsymbol{K}_2, \boldsymbol{Y}_2) \wedge (\boldsymbol{Y}_1 \neq \boldsymbol{Y}_2) \\ & \Big(\bigwedge_{j=1}^{j=i-1} C(\boldsymbol{X}_j^d, \boldsymbol{K}_1, \boldsymbol{Y}_j^d)\Big) \wedge \Big(\bigwedge_{j=1}^{j=i-1} C(\boldsymbol{X}_j^d, \boldsymbol{K}_2, \boldsymbol{Y}_j^d)\Big) \end{aligned} \tag{2}$$

Algorithm 1 SAT Attack Algorithm [12]

Input: C and $eval$
Output: $\boldsymbol{K}_C$

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

where $C(\boldsymbol{X}, \boldsymbol{K}, \boldsymbol{Y})$ is the SAT formula (CNF form) for a locked circuit and $(\boldsymbol{X}^d_{\{1\ldots i-1\}}, \boldsymbol{Y}^d_{\{1\ldots i-1\}})$ are the distinguishing I/O pairs that are found in previous $i-1$ iterations. If satisfiable, an assignment for variables $\boldsymbol{X}$, $\boldsymbol{K}_1$, $\boldsymbol{K}_2$, $\boldsymbol{Y}_1$, $\boldsymbol{Y}_2$ will be generated. The first line in the formula (2) evaluates the circuit functionality for a specific $\boldsymbol{X} = \boldsymbol{X}_i^d$ at two different key values $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ such that the outputs are different (see $\boldsymbol{Y}_1 \neq \boldsymbol{Y}_2$). This guarantees that the input $\boldsymbol{X} = \boldsymbol{X}_i^d$ is capable of identifying two keys $\boldsymbol{K}_1$, $\boldsymbol{K}_2$ which produce different outputs. Hence at least one of the two keys must be wrong. This in itself is not enough to call $\boldsymbol{X} = \boldsymbol{X}_i^d$ as a distinguishing input because previous iteration may have found another input assignment that could have differentiated between $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$. According to definition 2, a distinguishing input in the i-th iteration must find "unique" wrong key combinations that have not been identified by previous $i-1$ distinguishing I/O pairs. This condition is checked by the SAT clauses in the second line. In the second line $\boldsymbol{X}_j^d$ is the distinguishing input identified in the previous j-th iteration and $\boldsymbol{Y}_j^d$ is the corresponding correct output. This correct output is know from the activated functional chip obtained from the open market. The clauses in the second line guarantee that the keys $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ which result in "different" outputs in line 1 of this formula produce the "correct" outputs for all previous distinguishing I/O pairs. Hence in this iteration we could identify at least one incorrect key combination which previous iterations could not. Therefore by definition 2 the input $\boldsymbol{X}_i^d$ (obtained from the SAT solver) and the corresponding "correct" output $\boldsymbol{Y}_i^d = eval(\boldsymbol{X}_i^d)$ (obtained from the activated chip) represent the i-th distinguishing I/O pair.

The SAT attack algorithm is shown in Algorithm 1. Basically it starts by first solving the line one of the formula (2) and as iterations progress it adds the clauses comprised in line two of the formula (2). It stops when the resulting SAT formula is unsatisfiable indicating no further distinguishing I/O pairs exist. The correct key is obtained by finding a key value which satisfies the correct I/O behavior of all the distinguishing I/O pairs. This algorithm is guaranteed to find the correct key. Please refer to [12] for any further theoretical details.

3 Efficiency Analysis of SAT Attack

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

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

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.

The SAT solving time t_i is dependent on benchmark characteristics as well as the efficiency of the SAT solver used. To increase t_i, Yasin *et al.* [14] proposed to add an AES circuit to protect the locked circuit, as shown in Fig. 2(a). As the AES circuit is hard to be solved by a SAT solver, the SAT attack will fail to find a satisfiable assignment for the SAT attack formula. Although this approach is effective, the AES circuit leads to a large performance overhead since a standard AES circuit implementation requires a large number of gates [4].

Increasing the number of iterations λ is another approach to mitigate the SAT attack. 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 [12], even with large number of keys (50% area overhead), for six previously proposed key-gate insertion algorithms [1, 2, 8, 9, 11], 86% benchmarks on average can still be unlocked in 10 hours.

4 Anti-SAT Block Design

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. Fig. 3(a) and Fig. 3(b) illustrate two configurations of the proposed Anti-SAT block. They consist of two logic blocks $B_1 = g_{l1}(\boldsymbol{X}, \boldsymbol{K}_{l1})$ and $B_2 = \overline{g_{l2}(\boldsymbol{X}, \boldsymbol{K}_{l2})}$. These two logic blocks share the same set of inputs $\boldsymbol{X}$ and their original functionalities (before locking) g and $\overline{g}$ are complementary, but are locked with different keys ($\boldsymbol{K}_{l1}$ and $\boldsymbol{K}_{l2}$) at different locations ($l1$ and $l2$). The one-bit output Y is the AND (for Fig. 3(a)) or OR (for Fig. 3(b)) operation of two logic blocks.

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 (*e.g.* always equals to 0 for Fig. 3(a) or 1 for Fig. 3(b)). Otherwise, Y can output either 1 or 0 depending on the inputs. This property enables it to be integrated into the original circuit. As shown in Fig. 3(c), the inputs of Anti-SAT block $\boldsymbol{X}$ are from the wires in the original circuit. The output Y is connected into the original circuit using an XOR gate (or a XNOR gate + inverter). When a correct key is provided, the output Y always equals to 0 (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

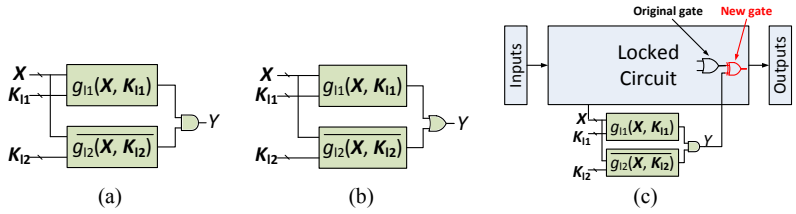

Fig. 3. Anti-SAT block configuration: (a) An Anti-SAT block that always outputs 0 if key values are correct; (b) An Anti-SAT block that always outputs 1 if key values are correct. (c) Integrating the Anti-SAT block into a circuit.

be 1 for some inputs (XOR gate behaves as an inverter) and thus can produce a fault in the original circuit.

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 give a provable lower bound to the number of SAT attack iterations. For some constructions of g, this lower bound is exponential in the number of keys thereby making the SAT-attack complexity very high. In the remaining of this paper, we take Fig. 3(a) as the configuration in our analysis and experiments (without loss of generality).

4.1 Construction of Anti-SAT Block

Now we describe how the Anti-SAT block can be constructed. Note that this construction may not be unique and other constructions may also be feasible. Consider the circuit illustrated in Fig. 4(a). Here a set of key-gates (XORs) are inserted at the inputs of two logic blocks, so $B_1 = g(\boldsymbol{X} \oplus \boldsymbol{K}_{l1})$ and $B_2 = \overline{g(\boldsymbol{X} \oplus \boldsymbol{K}_{l2})}$, where $|\boldsymbol{K}_{l1}| = |\boldsymbol{K}_{l2}| = n$. Hence the key-size is $2n$. The outputs B_1 and B_2 are fed into an AND gate and produce an output Y. As a result, we have $Y = g(\boldsymbol{X} \oplus \boldsymbol{K}_{l1}) \wedge \overline{g(\boldsymbol{X} \oplus \boldsymbol{K}_{l2})}$.

Note that here we are using only XOR gates as key-gates for the sake of ease of explanation. The key-gates used in Fig. 4(a) could be either XOR or XNOR gates based on a user-defined key. Similar to conventional XOR/XNOR base logic locking [9], if a correct key-bit is 0, the key-gate can be XOR or XNOR+inverter. If the key-bit is 1, the key-gate can be XNOR or XOR+inverter. 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 now be either 0 or 1). Moreover, as discussed in [9], 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. Besides, the XOR/XNOR gates can be synthesized using other gate types. Combined with obfuscation techqniues which will be discussed in Section 4.4, the attacker cannot obtain the correct key-values based on the types of gates connected to the key-inputs.

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. Because

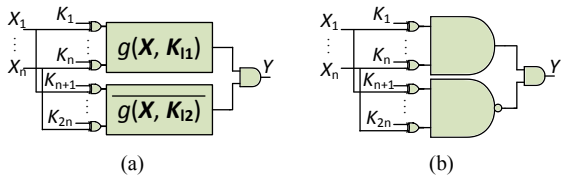

Fig. 4. Anti-SAT block construction: (a) basic Anti-SAT block construction and (b) one possible construction to ensure large number of SAT attack iterations.

correct key input (for Fig. 4(a)) happens when i-th key from $\boldsymbol{K}_{l1}$ and i-th key from $\boldsymbol{K}_{l2}$ have the same value, the number of correct key combinations $c = 2^n$.

4.2 SAT Attack Complexity Analysis

Here we analyze the complexity of a SAT attack on the Anti-SAT block construction of Fig. 4(a) (assuming this is the circuit being attacked to decode the $2n$ key bits).

Terminology Given a Boolean function $g(\boldsymbol{L})$ with n inputs, assuming there exists p input vectors that make g equal to one (denote p as *output-one count*, $1 \leq p \leq 2^n - 1$), we can classify the input vectors $\boldsymbol{L}$ into two groups L^T and L^F, where one group makes $g = 1$ and another makes $g = 0$:

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

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

Theorem 1: *Assuming the output-one count 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 (see λ in equation 3) to decipher the correct key is lower bounded by 2^n.*

Proof:

As described in Section 2, the SAT attack algorithm will iteratively find a distinguishing I/O pair ($\boldsymbol{X}_i^d$, Y_i^d) to identify wrong key combinations in the Anti-SAT block until all wrong key combinations are identified. In the i-th iteration, the corresponding distinguishing I/O pair can identify a subset of wrong key combinations, denoted as WK_i. Notice that for any input combinations (including the distinguishing inputs $\boldsymbol{X}_i^d$), the correct output (when provided the correct key) is 0. Therefore, a wrong key combination $\boldsymbol{K} = (\boldsymbol{K}_{l1}, \boldsymbol{K}_{l2}) \in WK_i$ which was identified by ($\boldsymbol{X}_i^d$, Y_i^d) must produce the Anti-SAT block output as 1. This condition is described below.

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

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

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

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

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

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. The reason 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" incorrect keys UK_i identified in iteration i is bounded by $p \cdot (2^n - p)$. This is noted in the equation below.

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

where UK_i is the number of unique incorrect keys identified at iteration i. 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 incorrect key combinations, its value must be $= 2^{2n} - c$. The equation above can be rewritten as follows.

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

Here λ_l is the lower bound on λ. As noted in Fig. 4(a) the correct key happens when the i-th bit from K_1 and i-th bit from K_2 have the same value. Hence $c = 2^n$. When $p \to 1$ or $p \to 2^n - 1$, we have the lower bound as follows:

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

Hence Proved.

Therefore, 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. One possible choice of g is indicated in Fig. 4(b) where g is chosen to be a simple AND gate. For AND gates $p = 1$ which clearly results in exponential complexity of SAT attack. Experimental results to indicate that shall be indicated subsequently. 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 $p = 2^n - 1$.

4.3 Anti-SAT block location

When the Anti-SAT block is integrated into a circuit, a set of wires in the original circuit are connected to the inputs $\boldsymbol{X}$ 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. 3(c)). If $\boldsymbol{X}$ 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 input of the Anti-SAT block. The location for Y is also important. An incorrect key causes $Y = 1$ for some inputs. This incorrect output must impact the overall functionality of the original circuit. Otherwise the logic will continue to function correctly despite wrong key inputs. In conclusion, the best location of the Anti-SAT block is such the inputs $\boldsymbol{X}$ are highly independent and Y has high observability at the POs (*i.e.,* change in Y can be observed by the POs of the original circuit). The impact of Anti-SAT block location on the overall security shall be evaluated in the experiments.

4.4 Anti-SAT Block Obfuscation

Since the Anti-SAT block is independent 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. Note that a similar criticism is possible for the AES based logic locking approach [14] which is very strong in presence of SAT attack but might be easily circumvented by an attacked due to its large footprint. To prevent such an identification and nullification, we apply both functional and structural obfuscation techniques to obfuscate the Anti-SAT block such that it can be covertly embedded in the original design.

Functional Obfuscation In the Anti-SAT block, 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. To prevent such attacks, we propose to insert additional key-gates at internal wires of two logic blocks (and the internal wires of $2n$

XOR/XNORs at the inputs since they can be synthesized using other gates) to obfuscate their functionalities. With an incorrect key, the functionalities of these two logic blocks are different and an attacker will fail to find the complementary pairs of signals through simulation. Besides, the logic blocks g and $\overline{g}$ and the key-gates can be synthesized using different logic gates to reduce their similarity.

Structural Obfuscation Besides functional obfuscation, we need to obfuscate the structure of the Anti-SAT block to prevent any structure-based attacks which are independent of functionality. 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. In order to prevent such attacks, we utilize the MUX-based logic locking (as shown in Fig. 1(d)) to increase the inter-connectivity between the locked circuit and the Anti-SAT block. A set of two-input MUX gates is utilized to connect these two parts. One input of the MUX is from a random wire w_1 of the locked circuit and the other input is from a random wire w_2 of the Anti-SAT block. The output of MUX replace the original signal (either w_1 or w_2) and is connected to the fan-outs of either w_1 or w_2. The selection bit of MUX is the key-input. After inserting the MUXes, the whole circuit is re-synthesized so as to obscure the boundary between two parts. With more MUXes inserted, 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, which will be validated in Section 5.3.

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 [12] 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.

Table 1. Impact of output-one count p on the security level of the $n = 16$-bit baseline Anti-SAT block. Timeout is 10 hours.

p	1	81	243	2187	30375	63349	65293	65455	65535
# Iteration	-	10675	4760	901	273	898	4647	-	-
Time (s)	timeout	16555.8	8746.12	174.743	3.24	307.104	12932.3	timeout	timeout

Table 2. Impact input-size n on the security level of the baseline Anti-SAT block (output-one count $p = 1$). Timeout is 10 hours.

n	8	10	12	14	16
# Iteration	255	1023	4095	16383	-
Time (s)	1.14569	20.024	324.727	4498.03	timeout

5 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. The SAT attack tools and benchmarks used are from [12]. The CPU time limit is set to 10 hours as [12]. The experiments are running on an Intel Core i5-2400 CPU with 16GB RAM.

5.1 Anti-SAT Block Design

We firstly evaluate the security level of the Anti-SAT block with respect to different design parameters: a) the input-size n and output-one count p of function g and b) the Anti-SAT block location. The **n-bit baseline Anti-SAT (BA) block** is constructed using a n-input AND gate and a n-input NAND gate (output-one count $p = 1$) as g and $\overline{g}$ 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 4.2, other function g that has sufficiently large n and sufficiently small (or large) p can also guarantee large number of iterations. The key-gates (XOR/XNOR) are inserted at the inputs of g and $\overline{g}$ with key-size $|\boldsymbol{K}_{l1}| = |\boldsymbol{K}_{l2}| = n$. Obfuscation techniques proposed in Section 4.4 are not applied here but they will be evaluated in the Section 5.3 when the Anti-SAT block is integrated into a circuit.

Input-size n and output-one count p As shown in equation (8), the lower bound of SAT attack iterations λ_l to unlock the Anti-SAT block is related to the input-size n and output-one count p of function g.

If n is fixed, λ_l is maximized when $p \to 1$ or $p \to 2^n - 1$. To evaluate the impact of p, we replace some 2-input AND gates with 2-input OR gates in a $n = 16$-bit baseline Anti-SAT block to gradually increase p. Table 1 illustrates the impact of p on the security level of the 16-bit Anti-SAT block. For $p = 1$ and $p = 2^{16} - 1 = 65535$, 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 ($\approx 2^{16}$) to rule out all the incorrect key combinations. As $p \to 2^{16}/2$ (the worst case),

Table 3. Impact of Anti-SAT block location on security level of the baseline n-bit Anti-SAT blocks ($n = 8, 12, 16$) inserted at c1355 circuit. Timeout is 10 hours. The random case is averaged over 5 trials.

	$\|\boldsymbol{K}_{l1}\| = \|\boldsymbol{K}_{l2}\| = n$	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

the SAT attack begins to succeed using less and less iterations and execution time. This result validates that when p is very small or very large for a fixed n, the iterations λ will be large and the SAT attack will fail within a practical time limit.

Moreover, as described in Section 4.2, 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 2 shows the exponential relationship between λ and n when $p = 1$ for five baseline Anti-SAT block ($n = 8, 10, 12, 14, 16$). 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 4.2.

Anti-SAT block location As noted in Section 4.3, the Anti-SAT block location may impact the its security in terms of SAT attack iterations and execution time. 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 $\boldsymbol{X}$ 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 $\boldsymbol{X}$ 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 latter topological order than that of the wires for $\boldsymbol{X}$ to prevent combinational loop. Table 3 compares two integration approaches when three baseline Anti-SAT block of different sizes ($n = 8, 12, 16$) are integrated into the c1355 circuit from ISCAS85. It can be seen that for three Anti-SAT blocks, secure integration is more secure 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.

5.2 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 4. We compare three logic locking configurations as follows:

Table 4. Benchmark information of 6 circuits from ISCAS85 and MCNC benchmark suites and the key-sizes of three logic locking configurations.

Circuit	#PI	#PO	#Gates	Key-size		
				TOC13 (5%)	n-bit BA	n-bit OA
c1355	41	32	546	29	2n	4n
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		

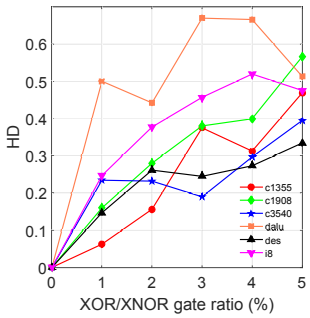

Fig. 5. HD v.s. key-gate ratio for TOC13 logic locking.

- **TOC13**: The original circuit is locked using TOC13 logic locking algorithm [9] which inserts XOR/XNOR gates into the circuit to obfuscate its functionality. Fig. 5 shows that TOC13 is effective in increasing the output corruptibility (in terms of the Hamming distance (HD) between the output of an original circuit and a locked circuit given a random key). Also, it can be seen that 5% overhead (ratio between # key-gates and # original gates) is roughly enough to approach 50% HD for all benchmarks.
- **TOC13(5%) + n-bit BA**: In this configuration, the original circuit is locked with TOC13 with 5% overhead. Besides, we integrate a n-bit baseline Anti-SAT (BA) block into the locked circuit using the *secure integration*, *i.e.,* the n inputs of the Anti-SAT block are connected to n PIs of the original circuit, and the output of the Anti-SAT block is connected to a wire in the original circuit which is randomly selected from the wires that have the top 30% observability. For a n-bit BA, its key-size is $k_{BA} = 2n$ because $2n$ keys are inserted at the inputs of g and $\overline{g}$.
- **TOC13(5%) + n-bit OA**: In this configuration, the obfuscation techniques proposed in Section. 4.4 are applied to make the baseline Anti-SAT block less distinguishable from the locked circuit. In our experiment, we insert n MUXes to increase the inter-connectivity between a n-bit baseline Anti-SAT block and the locked circuit. Besides, we insert additional n XOR/XNOR gates at random internal wires of the logic blocks g and $\overline{g}$ to obfuscate their functionality to prevent the detection of complementary pairs of signals. Thus, the keys in a n-bit obfuscated Anti-SAT (OA) block is $k_{OA} = 4n$.

We compare the security level of three 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 TOC13, the increased keys are inserted to the original circuit. For TOC13(5%) + n-bit BA/OA, the increased keys are used in the Anti-SAT block and increasing the key-size also indicates increasing the Anti-block size (in terms of input-size n) because we construct the BA and OA with $k_{BA} = 2n$ and $k_{OA} = 4n$, respectively. In this experiment, we

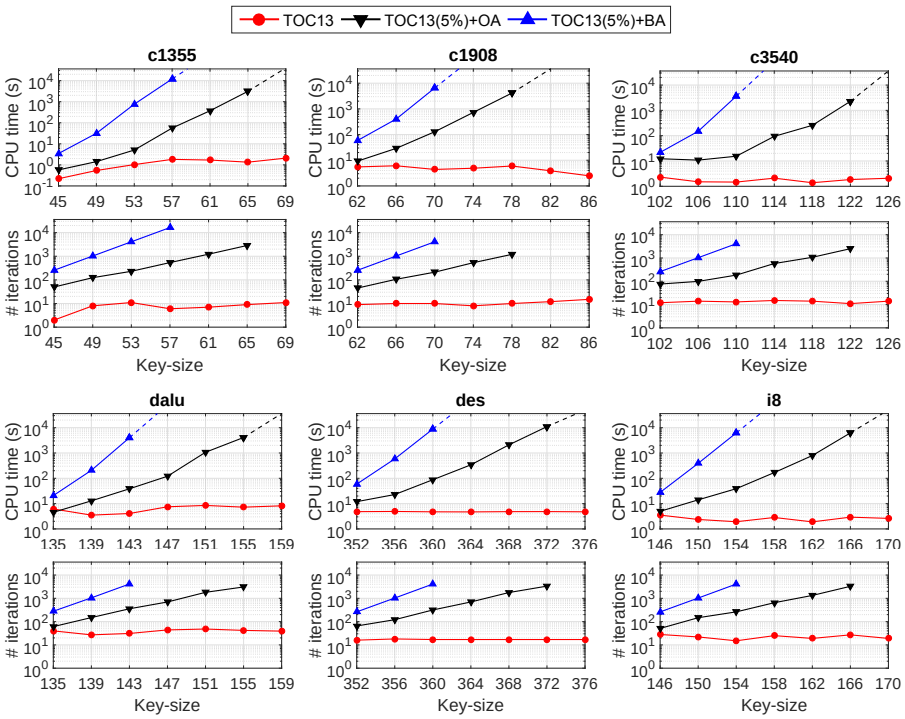

Fig. 6. SAT attack results on 6 benchmarks with three logic locking configurations: TOC13 only, TOC13(5%)+BA, and TOC13(5%)+OA. Timeout is 10 hours (3.6×10^4 s). The dashed line in the top figure (execution time) is the curve fitting result when the SAT attack has time-outed after certain key-size.

integrate the baseline Anti-SAT blocks of input-size $n_{BA} = 8, 10, 12, 14, 16, 18, 20$ and the obfuscated Anti-SAT blocks of input-size $n_{OA} = 4, 5, 6, 7, 8, 9, 10$. The input-size of BA is twice the size of OA ($n_{BA} = 2n_{OA}$) when their key-sizes are the same $k_{BA} = k_{OA}$.

The SAT attack result on three configurations w.r.t increasing key-size are shown in Fig. 6. 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 TOC13, increasing the key-size cannot effectively increase SAT attack complexity. For all benchmarks locked with TOC13, they can be easily unlocked using at most 48 iterations and 8.48 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. This holds for both the baseline Anti-SAT block and the obfuscated Anti-SAT block. Also the results show that with the same key-size, the grow rate of n-bit OA is slower than n-bit BA. This is because that for OA, a portion of keys are utilized to obfuscate the design and the resulting OA is half the size as the BA (in terms of n) as described earlier. Finally, we can see that for all benchmarks, the SAT

Table 5. Percentage of a 14-bit obfuscated Anti-SAT block isolated to a small partition when a min-cut partitioning algorithm is utilized. Area estimation error is percentage error of an attacker's estimation about the size of the Anti-SAT block.

Circuits	Area Estimation Error							
	0%		5%		15%		25%	
	NO MUX	With MUX	No MUX	With MUX	No MUX	With MUX	No MUX	With MUX
c1355	99.07%	0.00%	99.07%	0.93%	40.74%	0.93%	0.00%	0.00%
c1908	99.07%	0.93%	99.07%	0.00%	0.00%	0.93%	0.00%	0.00%
c3540	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
dalu	99.07%	0.00%	99.07%	0.00%	99.07%	0.00%	0.00%	0.00%
des	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
i8	100.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Average	66.20%	0.16%	66.20%	0.15%	23.30%	0.31%	0.00%	0.00%

attack fails to unlock the circuits within 10 hours when a 14-bit BA ($k_{BA} = 28$) is inserted or when a 10-bit OA ($k_{OA} = 40$) is inserted.

5.3 Anti-SAT Obfuscation

In Section 4.4, we claim that the Anti-SAT block has a separable structure because it has few interconnections with the locked circuit. When the size of the Anti-SAT block is roughly known, it's possible for an attacker to utilize a min-cut 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, an attacker will have less difficulty to identify the Anti-SAT block. In order to prevent the Anti-SAT block from being identified, we propose to use MUX-based logic locking to increase the inter-connectivity between the locked circuit and the Anti-SAT block. Table 5 shows the percentage of gates of a $n = 14$-bit OA (with or without MUXes) that are isolated to the small partition after min-cut partitioning. The reason for using a 14-bit OA is because that it can achieve a sufficiently large SAT attack time (according to the fitting result in Fig. 7 which will be discussed later). Because the Anti-SAT block can be synthesized with different types of gates, it's difficult for an attacker to obtain the exact size of the Anti-SAT block. Therefore, we perform the partitioning algorithm while assuming a certain area estimation error for the Anti-SAT block and analyze its impact on the attack results.

As shown in Table 5, when the area estimation error is 0% and no MUXes are inserted, the Anti-SAT block in 4 circuits (c1355, c1908, dalu, and i8) can be isolated. In these circuits, the percentages of gates of the Anti-SAT block isolated by the partitioning algorithm are almost 100%. However, with the increase of area estimation error, the partitioning fails to isolate the Anti-SAT block. Besides, when $n = 14$ MUXes are inserted to increase the inter-connectivity between the Anti-SAT block and the locked circuit, the percentages of isolated Anti-SAT block are almost 0% for four assumptions of area estimation error. This is because that the number of interconnections between the Anti-SAT block and the locked circuit is increased and partitioning them will result in a large cut-size, so the partitioning algorithm will avoid to separate the Anti-SAT block.

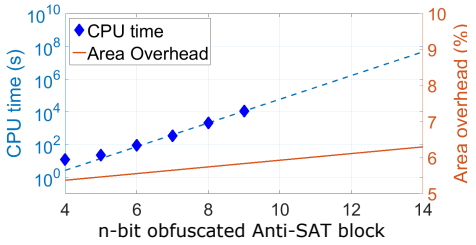

Fig. 7. SAT attack execution time (in log scale) and area overhead for the des circuit integrated with n-bit obfuscated Anti-SAT. The original circuit is locked with TOC13 (5% overhead). The blue dashed line is the fitting curve for CPU time.

5.4 Performance Overhead of the Anti-SAT Block

In our construction, a n-bit obfuscated Anti-SAT block consists of logic blocks g and $\overline{g}$, a 2-input AND gate, $3n + 1$ XOR/XNOR gates and n 2-input MUX 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 $\overline{g}$. The estimated SAT attack complexity and area overhead of inserting a n-bit OA into the des circuit are shown in Fig. 7. The overhead is evaluated in terms of the ratio between the total size of extra gates and the total size of original gates. It can be seen that a slight increase in area overhead can result in exponential increase in SAT attack's computation complexity. It's also important to notice that the design of the Anti-SAT block does not scale with the benchmark size; it only scales with the attacker's computation power. Compared to the technique proposed in [14] which inserts an AES circuit to defend the SAT attack, our proposed technique has much less overhead.

6 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 against SAT attack. Compared to adding a large hard-SAT circuit (*e.g.* AES), our proposed Anti-SAT block has a much smaller overhead, which makes it a cost-effective technique to mitigate the SAT attack.

Acknowledgments

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

References

1. Baumgarten, A., Tyagi, A., Zambreno, J.: Preventing IC piracy using reconfigurable logic barriers. IEEE Design & Test of Computers (2010)
2. Dupuis, S., Ba, P.S., Di Natale, G., Flottes, M.L., Rouzeyre, B.: A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International. pp. 49–54. IEEE (2014)
3. Guin, U., Huang, K., DiMase, D., Carulli, J.M., Tehranipoor, M., Makris, Y.: Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proceedings of the IEEE 102(8), 1207–1228 (2014)
4. HelionTechnology: High performance AES (Rijndael) cores for ASIC. http://www.heliontech.com/downloads/aes_asic_helioncore.pdf (2015)
5. Khaleghi, S., Da Zhao, K., Rao, W.: IC piracy prevention via design withholding and entanglement. In: Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific. pp. 821–826. IEEE (2015)
6. Liu, B., Wang, B.: Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In: Proceedings of the conference on Design, Automation and Test in Europe. p. 243. European Design and Automation Association (2014)
7. Plaza, S.M., Markov, I.L.: Solving the third-shift problem in IC piracy with test-aware logic locking. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 34(6), 961–971 (2015)
8. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Security analysis of logic obfuscation. In: Proceedings of the 49th Annual Design Automation Conference. pp. 83–89. ACM (2012)
9. Rajendran, J., Zhang, H., Zhang, C., Rose, G.S., Pino, Y., Sinanoglu, O., Karri, R.: Fault analysis-based logic encryption. Computers, IEEE Transactions on 64(2), 410–424 (2015)
10. Rostami, M., Koushanfar, F., Karri, R.: A primer on hardware security: models, methods, and metrics. Proceedings of the IEEE 102(8), 1283–1295 (2014)
11. Roy, J.A., Koushanfar, F., Markov, I.L.: Epic: Ending piracy of integrated circuits. In: Proceedings of the conference on Design, Automation and Test in Europe. pp. 1069–1074. ACM (2008)
12. Subramanyan, P., Ray, S., Malik, S.: Evaluating the security of logic encryption algorithms. In: Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on. pp. 137–143. IEEE (2015)
13. Wendt, J.B., Potkonjak, M.: Hardware obfuscation using PUF-based logic. In: Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design. pp. 270–277. IEEE Press (2014)
14. Yasin, M., Rajendran, J., Sinanoglu, O., Karri, R.: On improving the security of logic locking. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on (2015)