Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks

Xiaolin Xu*, Bicky Shakya*, Mark M. Tehranipoor, and Domenic Forte

ECE Department, University of Florida
{xiaolinxu,tehranipoor,dforte}@ece.ufl.edu bshakya@ufl.edu

Abstract. Logic locking has emerged as a promising technique for protecting gate-level semiconductor intellectual property. However, recent work has shown that such gate-level locking techniques are vulnerable to Boolean satisfiability (SAT) attacks. In order to thwart such attacks, several SAT-resistant logic locking techniques have been proposed, which minimize the discriminating ability of input patterns to rule out incorrect keys. In this work, we show that such SAT-resistant logic locking techniques have their own set of unique vulnerabilities. In particular, we propose a novel "bypass attack" that ensures the locked circuit works even when an incorrect key is applied. Such a technique makes it possible for an adversary to be oblivious to the type of SAT-resistant protection applied on the circuit, and still be able to restore the circuit to its correct functionality. We show that such a bypass attack is feasible on a wide range of benchmarks and SAT-resistant techniques, while incurring minimal run-time and area/delay overhead. Binary decision diagrams (BDDs) are utilized to analyze the proposed bypass attack and assess tradeoffs in security vs overhead of various countermeasures.

1 Introduction

With the globalization of semiconductor industry, many companies have relocated the fabrication of their integrated circuits (ICs) from trusted on-shore foundries to untrusted off-shore foundries. As a result of this realignment, companies as well as government agencies are now facing threats of intellectual property (IP) theft/piracy, counterfeiting, and IC overproduction [1]. Therefore, there is a critical need to develop technologies that tackle the threats associated with untrusted foundries. Towards this end, various countermeasures such as split manufacturing [2], IC metering [3] and logic locking [4][5] have been developed. Among these techniques, logic locking has emerged as a low-cost and effective

* Indicates equal contribution.

solution. Basic logic locking works by embedding extra *key-gates* into the netlist of the circuit design. Proper operation of the circuit can only be ensured in the presence of the correct unlocking key. However, recent work has shown that early logic locking techniques are all vulnerable to *Boolean satisfiability (SAT) based attacks* [6]. In these SAT attacks, a small set of *discriminating input patterns (DIPs)* are obtained from the locked circuit netlist and incorrect keys that do not satisfy the DIP and the corresponding correct output are ruled out. In order to mitigate SAT attacks, several SAT-resistant countermeasures have been recently proposed [7] [8].

In this paper, we show that the cutting-edge SAT-resistant logic locking techniques: SARLock and Anti-SAT, also possess their own critical vulnerability. In particular, we show that for any logic locking technique which is highly resistant to SAT attacks, it becomes more vulnerable to "bypass attacks" that can easily circumvent the effect of the SAT resistant locking scheme. In this novel yet simple attack, the logic locked circuit is embedded with a low-overhead bypass circuitry that enables the circuit to operate *even in the presence of an incorrect* key. Our main contributions in this paper can be summarized as follows:

- We present the bypass attack, which can be applied to recently proposed SAT-resistant logic locking techniques. Our attack uses the *same* set of assumptions/adversarial models as regular SAT attacks and can make the circuit operate correctly with any arbitrary key.
- We present the complete flow of the attack and show that it can thwart the state-of-the-art logic locking techniques: SARLock, Anti-SAT and hybrid versions of SARLock. We execute the attack on several benchmark circuits protected with these SAT-resistant logic locking methods. Further, we show that the original functionality of the circuit can be restored with area overheads linear to the number of patterns to bypass, and with minimal runtime required to execute the attack.
- We analyze logic locking techniques and SAT-resistant countermeasures in terms of existing attacks and the proposed attack. We show that bypass attack possesses a tradeoff with SAT attack, i.e., resistance to bypass decreases the resistance to SAT and resistance to SAT decreases the area overhead of the proposed attack. This leads to an interesting new way of assessing the security of logic locking schemes.
- Binary decision diagrams (BDDs) are introduced as a method to determine whether there exists a feasible complexity/overhead/attack resistance tradeoff for secure logic locking. The benefits and future challenges associated with BDD-based logic locking approaches are also discussed.

The rest of the paper is organized as follows. Section 2 reviews the background of conventional logic locking and the countermeasures against SAT attacks. Section 3 explains our bypass attack; in particular, the feasibility/scalability of our attack on different logic locking techniques is shown. Section 4 presents experimental results (delay/area overhead, computation time) of the attack on various benchmarks. Our proposed attack is also compared with the state-of-the-

art. Section 5 presents the BDD-based approach for logic locking and tradeoff analysis. Section 6 concludes the paper.

2 Background and Related Work

Logic locking techniques modify the netlist of a circuit design by adding extra key controlled logic such that the circuit will only work correctly when the correct key (or keys) is applied to it; otherwise, the circuit's output is corrupted. The insertion of additional key gates into the original netlist obfuscates the functionality of the IC to an untrusted foundry and potentially prevents them from engaging in overproduction or IC piracy. Several techniques have been proposed over the years in order to perform logic locking, such as random locking [4] and fault analysis-based techniques [5]. Unfortunately, all these approaches are vulnerable to SAT attacks, as discussed below.

2.1 SAT Attacks on Logic Locking

In the SAT attack model [6], an attacker has access to: 1) *Logic Locked Netlist*: Such a netlist can be obtained from a malicious foundry or through reverse-engineering [9]. Simulations can also be readily performed on the netlist. 2) *Unlocked IC*: Such an IC can be purchased from the open market or through a malicious insider in the trusted design house. This IC can be used by the attacker as an oracle, i.e., one can check whether the output for a given key from the locked netlist is correct. In order to perform this attack within reasonable time, an attacker seeks to apply the minimum number of input patterns to the IC. Note that only combinational circuits (or sequential circuits in which all flip-flops are assumed to be accessible through the scan chain) are considered in such attacks [6].

Various attacks have been proposed based on this attack model to minimize the number of required input patterns. For example, in [10], automatic test pattern generation (ATPG) tools [11] are used to generate a set of inputs that can propagate (sensitize) the correct key to observable outputs in the circuit. In SAT-based attacks, such propagations are not required. Instead, the attacker iteratively finds a set of *distinguishing input patterns* (DIPs) for which two copies of the locked netlist, loaded with two wrong keys, produce different outputs. Since the unlocked IC is available to the attacker, he or she can then apply this pattern to the unlocked IC and find the correct output. The algorithm then iteratively uses these DIPs to guide a SAT solver to a correct key value. The algorithm terminates when no more DIPs can be found, which means that the remaining key is guaranteed to be the correct key. The results in [6] show that the algorithm quickly converges in little to no time, with a fairly small amount of DIPs.

2.2 Notation and Terminology

- A bold variable means a set of elements, and |.| is used to denote the number of elements in a set. For example $\mathbf{K}$ stands for a key set with $|\mathbf{K}|$ possible keys, and K_i represents the i^{th} element in this set;
- We denote the input/output relationship of the obfuscated logic circuit with: $\mathbf{Y} = F(\mathbf{X},\ \mathbf{K})$, where $\mathbf{Y}$ denotes the primary output space of the circuit, $\mathbf{X}$ denotes the primary input space and $\mathbf{K}$ denotes the key input space; similarly, $Y = F(X,\ K)$ means that one primary output Y is generated by the circuit fed with one input vector X and key K;
- To keep it consistent with common SAT notation, an obfuscated logic circuit is expressed in conjunctive normal form (CNF) as $C(\mathbf{X}, \mathbf{K}, \mathbf{Y})$. $SAT(C(\mathbf{X}, \mathbf{K}, \mathbf{Y}))$ is used to evaluate whether the CNF $C(\mathbf{X}, \mathbf{K}, \mathbf{Y})$ is true or false. $\mathbf{X} = SAT\_Assignment(C(\mathbf{X}, \mathbf{K}, \mathbf{Y}))$ refers to calling a SAT solver to find satisfying assignments $\mathbf{X}$ for the CNF $C(\mathbf{X}, \mathbf{K}, \mathbf{Y})$.
- The evaluation operation with X on the unlocked IC (i.e., applying DIPs and observing the correct output) is denoted by $eval(X)$.

2.3 SAT-Resistant Logic Locking

To strengthen the security of logic locking, various SAT-resistant techniques have been recently developed, most notably SARlock [7] and Anti-SAT [8]. Both these techniques attach additional logic to the circuit in order to reduce the number of wrong keys that can be ruled out by each DIP and, therefore, force the SAT attack to take an exponential number of iterations to find the correct key.

SARLock. In SARLock [7], *at most* one incorrect key value is ruled out by each DIP. This effect is brought about by a small comparator circuit that flips the circuit output for only one input pattern for a given (wrong) key. SARLock results in the worst case scenario for the attacker, as shown in the truth table of Fig. 1. For this particular circuit/Boolean function, there are, in total, $2^3 = 8$ possible key values: K_0-K_7. When the input pattern $\{1, 1, 1\}$ is applied, only K_7 can be identified as incorrect. To find the correct key, one has to iteratively search through 6 more DIPs and rule out the other wrong keys (K_0-K_5). On the other hand, it is possible to rule out all incorrect keys with one input pattern $\{1, 1, 0\}$ for a regular logic locked design.

SARLock+SLL. Though SARLock possesses strong resistance against SAT attacks, it cannot protect the circuit against other attacks that exploit its mode of implementation. For example, in a *removal attack*, an attacker can analyze the netlist and then identify and remove the SARLock gates from the design. To mitigate this vulnerability, the authors in [7] proposed a two-layer or hybrid logic-locking mechanism: SARLock + strong logic locking (SLL)[10]. This hybrid technique combines SARLock with regular logic locking (i.e., embedding of XOR/XNOR/MUX key-gates into the netlist), and also intertwines the two keys (SARLock key and SLL key) using permutations.

Input Patterns	Golden output	Output patterns for different keys							
		K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
000	0	1	0	0	1	0	0	0	0
001	0	0	0	1	0	0	0	0	0
010	0	0	1	0	0	1	0	0	0
011	1	1	1	1	0	1	1	1	1
100	0	0	0	0	0	1	0	0	0
101	1	1	1	1	1	1	1	1	1
110	1	0	0	0	0	0	0	1	0
111	1	1	1	0	1	1	1	1	0

(a) Truth table of regular logic-lock design

Input Patterns	Golden output	Output patterns for different keys							
		K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
000	0	1	0	0	0	0	0	0	0
001	0	0	0	1	0	0	0	0	0
010	0	0	1	0	0	0	0	0	0
011	1	1	1	1	0	1	1	1	1
100	0	0	0	0	0	1	0	0	0
101	1	1	1	1	1	1	1	1	1
110	1	1	1	1	1	1	0	1	1
111	1	1	1	1	1	1	1	1	0

(b) Truth table of SARLock design

Fig. 1: Two truth tables of a logic design with 3-bit inputs. (a) shows that multiple wrong keys will be ruled out for each input pattern. (b) shows that with each input patterns, only one incorrect key value can be identified.

The SARLock+SLL scheme comprises of a $2n$-bit key, where n-bits are used for SARLock and n-bits are used for SLL. To understand the exact effect of such a hybrid scheme, we divide the whole key set (consisting of 2^{2n} keys) into *SLL set* and *SARlock set*. The SARLock set comprises of 2^n keys where the n SLL key bits are correct and the n SARLock key bits are incorrect. All the other keys ($2^{2n} - 2^n$) are classified into a SLL set, as shown in Fig. 2. From the table, it can be seen that a single DIP can rule out multiple wrong keys in the SLL set. However, if a wrong key is in the SARLock set, then only one DIP can be found and *at most* one key in the SARLock set can be ruled out per iteration. As shown in Fig. 2, we can see that the SAT attack can easily rule out the keys (K_0, K_1, K_2, K_3, K_4, K_5, K_6)[1] in the SLL set with a small number of DIPs. However, the keys (K_0^{SAR}, K_1^{SAR}, K_2^{SAR}, K_3^{SAR}) in the SARLock set can only be ruled out one at a time per input pattern. Therefore, the SAT resistance of the hybrid scheme is only brought about by keys in the SARLock set. The keys in the SLL set only add a negligible amount of DIPs for the attack.

Anti-SAT In Anti-SAT [8], an Anti-SAT block is integrated into the circuit (see Fig. 3), which is composed of a pair of sub-blocks $B_1 = g_{l1}(X, K^{l1})$ and $B_2 = \overline{g_{l2}(X, K^{l2})}$. The two blocks share a common input X but two different keys K^{l1} and K^{l2}. The functionality of the two blocks g_{l1} and $\overline{g_{l2}}$ are complementary. Hence, they can also be denoted by g and $\bar{g}$. There is a one-bit output Y for the Anti-SAT block, which is generated by ANDing B_1 and B_2. Similar to SARLock, a wrong key applied on the Anti-SAT block will enable $Y = 1$ for some input pattern(s), and flip the correct outputs, as depicted in Fig. 3(a). Assuming the Boolean function g has n inputs, we denote the number of input patterns that make g evaluate to "1" as p. The authors in [8] prove that the decryption capability of the SAT attack is greatly limited if p is sufficiently close to 1 (or $2^n - 1$). A properly designed Anti-SAT block satisfying $p = 1$ forces an attacker

[1] These sequential numbers are used to make it easier to visualize the entire key space.

Input Patterns	Golden output	Output patterns for different keys															
		SLL set					SARLock set						SLL set				
		K_0	K_1		K_2	K_3	K_0^{SAR}	K_1^{SAR}		K_2^{SAR}	K_3^{SAR}	K_4	K_5		K_6		
000…000	0	1	1		0	1	1	0		0	0	0	1		1		
000…001	1	0	0		0	0	0	0		0	0	0	0		0		
000…011	0	1	1		0	0	0	0		0	0	0	1		0		
…	…	…		…	…		…		…	…		…		…	…		
111…100	0	1	0		1		0	1		0	0	1	1		0		
111…101	1	1	0		1	0	1	1		1	1	1	0		0		
111…110	0	1	0		0	1	1	1		1	1	1	1		1		
111…111	1	1	1		0	0	1	1		1	1	1	0		1		

Fig. 2: A truth table example of the SARLock+SLL mechanism. The strength of the SARLock+SLL scheme against SAT attack is provided only by the keys in the SARLock set. (Note that the key space is divided into SLL and SARLock sets for simplicity. In practice, the keys of the two sets are mixed with each other.)

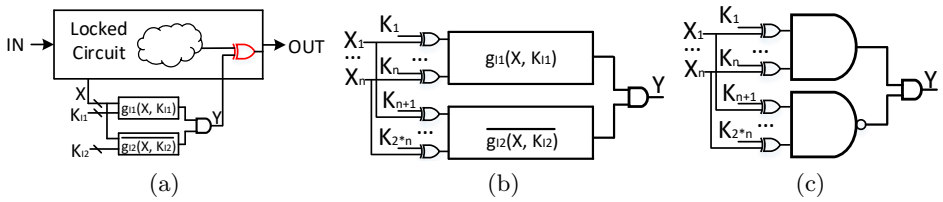


Fig. 3: Schematic of Anti-SAT: (a) shows the integration of Anti-SAT and a locked circuit. By using an XOR gate, the Anti-SAT block can flip the output if a wrong key is used. (b) illustrates the construction of Anti-SAT block, in which two complementary Boolean functions with n-bit inputs are employed. (c) shows an example of Anti-SAT implemented with AND and NAND gates.

to enumerate the largest number of possible keys to reveal the correct ones. They also note that natural candidates for g and $\bar{g}$ that satisfy $p = 1$ are AND and NAND respectively.

2.4 Other Attacks

Yasin *et al.* have proposed the use of cipher blocks (such as AES) for generating logic-locking keys [12], which are infeasible to break by SAT within reasonable time. However, due to the independence between the cipher block and the functional circuitry, it becomes trivial for the attacker to identify and circumvent the AES. To prevent similar vulnerabilities, Xie *et al.* propose functional and structural obfuscation techniques to enhance the security of Anti-SAT block [8]. However, it has been recently shown that although the Anti-SAT block can be hidden in the whole netlist, the attacker can still identify the flip signal Y generated by the Anti-SAT block, by analyzing the signal probability skew of the g and $\bar{g}$ blocks in the circuit [13]. This allows the attacker to set the flip signal of the Anti-SAT block to 0 and then apply the conventional SAT attack.

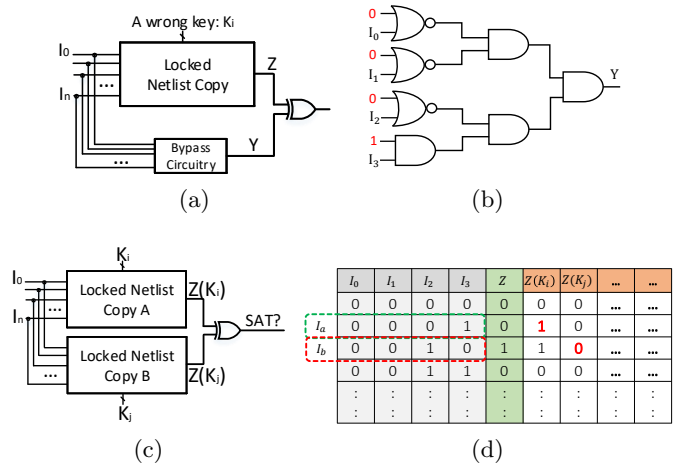


Fig. 4: (a) shows that for a locked netlist, a bypass circuit can be inserted to detect the DIP for the wrong key K_i. (b) shows an example bypass circuit block for correcting the flipped output in (d). When the input pattern (I_0, I_1, I_2, I_3) is (0,0,0,1), a logic "Y=1" will be generated to flip the original wrong output. (c) denotes the construction of miter circuit, which will be then applied to the SAT Solver. (d) shows an example truth table for finding the DIPs.

3 Bypass Attack: Definition and Methodologies

3.1 Adversarial Model/Capabilities

In this work, we follow the *same* adversarial model considered in most attacks on logic locking [6], i.e., the malicious party is in possession of the following: (1) The locked netlist; and (2) An unlocked IC, on which the attacker can apply input patterns and observe outputs. In practice, the attacker treats the locked netlist as a black box, and seeks to unlock the functionality of the design so that it can be pirated/overproduced.

3.2 Our Method: Bypass Attack

The main purpose of SAT attack is to reveal the correct key by iteratively applying DIPs. However, once all DIPs for any wrong key are known, an alternative for the attacker is to reverse the incorrect outputs instead of continuing with the search for the correct key(s). Taking the schematic in Fig. 4(a) as an example, if the DIPs that cause an incorrect output for a wrong key are known, then one can simply stitch a "bypass circuit" to monitor those DIPs and reverse the output back to the correct one. Such a bypass circuitry can be constructed with a comparator, which is stitched to the primary output of the circuit/logic cone. An example bypass circuit that monitors the DIP=$(0, 0, 0, 1)$ is shown in Fig.

4(b). When the circuit encounters this DIP, it can be used to trigger a signal $Y = 1$ that inverts the incorrect output. In summary, a bypass circuit ensures that the incorrect output can be inverted back; thereby nullifying the effect of a wrong key.

Miter Construction. The first step in our proposed bypass attack is constructing a miter circuit that can be fed into a SAT solver. The miter is constructed with two circuit copies: the first is a copy of the locked netlist with an incorrect key K_i and the second is the same locked netlist with another incorrect key K_j, as shown in Fig. 4(c). A SAT solver can then be used to find a DIP that causes the miter to evaluate to 1 (where the output of copy A does not equal the output of copy B). In the example in Fig. 4(d), the SAT solver should find and return the input pattern $I_a = (0, 0, 0, 1)$ or $I_b = (0, 0, 1, 0)$ where $Z(K_i) \neq Z(K_j)$, where Z is the output of the circuit copy. Further, calling the SAT solver again (while banning the previous solution) should return both input patterns I_a and I_b. Note that any input pattern which causes both $Z(K_i)$ and $Z(K_j)$ to evaluate to the same wrong logic value (e.g., $Z(K_i) = Z(K_j) = 0$ when $Z = 1$) will not be discovered by this miter construction.

Querying Unlocked IC. Once I_a and I_b are found, they can be applied on the unlocked IC to find the correct outputs. In Fig. 4(d), $Z = 0$ for I_a, and $Z = 1$ for I_b. With these observations, we can now see that for the locked netlist with key K_i, only input pattern I_a produces the incorrect output. Provided that standalone SARLock or Anti-SAT is applied (no SLL or structural/functional obfuscation), we can be certain that this is the only input pattern for which the circuit with wrong key K_i produces the wrong output. Similarly, for the locked netlist with wrong key K_j, I_b is the only pattern that produces the wrong output.

Bypass Circuitry Overhead. In terms of gates, the bypass circuitry overhead is a linear function of the number of DIPs N_{DIP} for the wrong key found above and the number of output bits flipped by the DIPs. Consider a circuit with N primary inputs. It would need N XNOR gates (or AND/NOR) for checking the inputs for the single DIP, $(N - 1)$ two input AND gates for determining a match between the DIP and input, and one XOR gate to flip the primary output when the input matches the DIP. In case the flip signal from the Anti-SAT/SARLock is not connected directly to the primary output (and instead, to an internal net), we can evaluate the number of primary outputs in the fan-in cone of the key input (say N_{out}), and embed N_{out} XOR gates into the N_{out} primary outputs. Thus we have the following expression:

$$Overhead = (2N - 1) \times N_{DIP} + N_{out} \tag{1}$$

The overhead across a set of benchmarks will be shown in Section 4.

In the sections below, we show how to apply this attack on SARLock, SARLock+SLL, and Anti-SAT.

3.3 Bypass Attack on SARLock

In SARLock, there is only one DIP corresponding to each wrong key. In other words, though the wrong key is applied, the functionality of the circuit is just slightly different from that of an unlocked IC. This favors our bypass attack. Simply put, we can just apply any random key[2], and then identify the lone DIP with a SAT solver. By simply reversing the flipped output with a bypass circuitry, we can make the circuit (fed with a wrong key) regain its correct input-output behavior.

3.4 Bypass Attack on SARLock+SLL

Following the methodology of bypass attack on SARLock, we can pick up a random wrong key, identify all the DIPs and reverse them for SARLock+SLL. However, this is not a good choice in practice because for each key in the SLL set (as mentioned in Section 2.3), the number of DIPs is not a constant value. This would increase the overhead of the bypass circuit. Further, we would not be able to guarantee the correct functionality of the bypassed circuit (more on this will be discussed in Section 4). By analyzing the truth table in Fig. 2, we can make two conclusions:

1. For any two random keys K_i^{SAR} and K_j^{SAR} from the SARLock set, the Hamming Distance $HD(F(X, K_i^{SAR}), F(X, K_j^{SAR}))$ between their outputs[3] is 1 if the input X is a DIP. Here, $F(X, K_i^{SAR})$ denotes the output of the design for a primary input X and key input K_i^{SAR}. In other words, for any two (wrong) keys in SARLock set, *at most* 2 DIPs can be observed.
2. In a single iteration of the SAT attack, *at most* 1 incorrect key from the SARLock set can be ruled out using one DIP, but ≥ 2 wrong keys from SLL set can be ruled out.

These two observations imply that our approach should now be to first find a wrong key in the SARLock set and then implement our bypass attack. To realize this, we propose Algorithm 1[4], which is a modified version of the original SAT algorithm presented in [6]. In the original attack, the algorithm terminates when no further DIPs can be found. For the purpose of executing our bypass attack, the algorithm should instead terminate when all the wrong keys in the SLL set have been ruled out. In other words, the new algorithm stops when *no more DIPs which can rule out at least 2 wrong keys in a single iteration are found*[5].

The modified attack is shown in Algorithm 1. The main difference between this and the original SAT attack [6] lies between lines 2 and 11. In the modified

[2] The probability of getting the correct key in the first random try is extremely low, thus we do not consider this situation.

[3] "1" means the number of flipped outputs, not the number of flipped bits.

[4] Note that a paper recently accepted to GLSVLSI 2017 proposed a similar algorithm [14]. We developed Algorithm 1 independently.

[5] Note that when this condition is satisfied, some keys in the SARlock set might also have been ruled out, but all the keys in SLL set are already ruled out.

Algorithm 1 Ruling out the wrong keys in SLL set.

Prerequisite: C and *eval* (as defined in Section 2.2)
Ensure: A wrong key candidate K^{SAR} in SARlock set
1: $i := 1$
2: $F_1^1 = C(X_1, K_1, Y_1) \wedge C(X_1, K_2, Y_2)$
3: $F_1^2 = C(X_1, K_3, Y_1) \wedge C(X_1, K_4, Y_2)$ {K_1, K_2, K_3 and $K4$ are 4 random key candidates}
4: $F_1 = F_1^1 \wedge F_1^2$ {F_1 is a SAT formula composed by 2 parts: F_1^1 and F_1^2}
5: **while** $SAT\big[F_i \wedge (Y_1 \neq Y_2) \wedge (K_1 \neq K_3) \wedge (K_2 \neq K_4)\big]$ **do**
6: $\quad X_i^d := SAT\_Assignment\Big((F_i \wedge (Y_1 \neq Y_2) \wedge (K_1 \neq K_3) \wedge (K_2 \neq K_4))\Big)$
7: $\quad Y_i^d := eval(X_i^d)$
8: $\quad F_{i+1}^1 = F_i^1 \wedge C(X_i^d, K_1, Y_i^d) \wedge C(X_i^d, K_2, Y_i^d)$
9: $\quad F_{i+1}^2 = F_i^2 \wedge C(X_i^d, K_3, Y_i^d) \wedge C(X_i^d, K_4, Y_i^d)$
10: $\quad i \leftarrow i + 1$
11: $\quad F_i = F_i^1 \wedge F_i^2$
12: **end while**
13: $K^{SAR} = K_1$ {when the algorithm terminates, any key remaining should be in *SARLock set*}
14: **return** K^{SAR}

attack, a combinational miter is formed between four copies of the locked netlist, each with keys K_1, K_2, K_3, K_4, the same input X_i and outputs Y_1, Y_2 (lines 2, 3 and 4). A SAT solver is called to find a DIP X_i^d, that causes the four circuit copies to produce outputs such that $Y_1 \neq Y_2$ (line 6). This X_i^d is then applied on the unlocked circuit to obtain the correct output Y_i^d (line 7). After X_i^d and Y_i^d are obtained, these are added as constraints to the conjunctive normal form (CNF) circuit formula, so that in the next iteration, the keys K_1, K_2, K_3, K_4 will be chosen such that they are consistent with all the X_i^d and Y_i^d inputs/outputs observed thus far on the unlocked IC (line 8 and 9). In contrast to the original SAT algorithm, this algorithm will terminate when no more than 2 wrong keys can be ruled out within a single iteration (with one single DIP X_i^d). This implies that all the wrong keys in SLL set have been ruled out, and any key(s) left behind (K^{SAR}) should now be in the SARlock set. As stated earlier, in the SARlock set, the key bits corresponding to SLL gates are correct and the key bits corresponding to the SARLock block may or may not be correct. After this, K_{SAR} can be used to implement our bypass attack as previously discussed in Section 3.3 for SARLock. Note that once K_{SAR} is obtained, the area overhead required for the bypass attack will be the same as that of standalone SARLock.

3.5 Bypass Attack on Anti-SAT

In [8], two different modes of integration of the Anti-SAT block were proposed: *secure integration (SI)* and *random integration (RI)*. In *secure integration* mode, the n-bit inputs X of the Anti-SAT block are directly connected with the n-bit primary inputs (IN) of the original circuit, and output Y of the Anti-SAT block is

connected to a randomly selected wire in the circuit that has high observability. In *random integration* mode, the inputs X and output Y of Anti-SAT are connected to several random internal wires of the original circuit. The authors also showed that the Anti-SAT block implemented with secure integration was more resistant to SAT attacks than random integration. In Appendix A, we describe the secure integration mode in more detail and also show that using secure integration makes it easier to apply the bypass attack. More specifically, we show that if an Anti-SAT block is implemented using secure integration (where $p = 1$), there exists one and only one DIP for any wrong key. This then implies that our bypass attack can be implemented on Anti-SAT in the exact same manner as on SARLock. However, for $p > 1$, the number of DIPs causing bit flips (N_{DIP}) increases and therefore, the overhead of the bypass attack increases (see Equation 1). Thus, there is tradeoff between SAT-resistance attack complexity and bypass attack overhead.

In random integration mode, it cannot be guaranteed that there exists only one DIP per wrong key. Internal nets in wires are often correlated (to varying degrees), which prevents all possible input patterns from occurring at the input of the Anti-SAT block. Therefore, the *one bit flip per wrong key* assumptions holds only for a very limited subset of the entire input space. This brings about two effects.

- The SAT attack becomes easier, as only a limited subset of the entire input space triggers the Anti-SAT block. Therefore, the number of DIPs as well as the time required to execute the attack decrease significantly. Further, a large number of keys could turn out to be correct, because of the failure of the Anti-SAT block to trigger. This explanation is also supported by the results in [8], where it was shown that random integration was broken in far fewer iterations/less time than secure integration. We also performed a few experiments on random integration, where we varied the nodes chosen (as well as the number of nodes chosen) as inputs to the Anti-SAT block. While SAT attack execution time increased with the number of nodes chosen, it also varied significantly with the choice of nodes. For example, for the C3540 benchmark, a 32 bit Anti-SAT key resulted in a SAT attack time of 89 s (941 iterations) for one choice of 16 random nodes, and 616 s (2615 iterations) for yet another choice of nodes.
- Bypass attack becomes harder (or less feasible), as setting a random wrong key in the locked circuit could result in multiple bit flips for multiple input patterns. Depending on which wrong key is randomly chosen, the number of patterns (and therefore, the number of gates required to implement the bypass circuitry) could be prohibitively high. For example, when querying the miter circuit for the C3540 benchmark, we found that for some wrong keys, the SAT solver returned UNSAT immediately, indicating that no distinguishing patterns existed between the two circuits with the two wrong keys. For other key pairs, however, we found that the solver returned more than 50K patterns as distinguishing.

In summary, our bypass attack works very good against secure integration (SI). Although the bypass attack also works on random integration (RI), its scalability in terms of area overhead depends on which internal nodes are selected.

4 Experimental Results and Discussion

In this section, we evaluate the performance and overhead of our approach. We also compare our technique with the current state-of-the-art.

4.1 Experimental Setup

We evaluate our method with a subset of benchmarks from the ISCAS, MCNC and EPFL benchmark sets [15][16]. For each benchmark, a primary output with at least 8 inputs in its transitive fan-in cone was chosen and all gates in such a cone were extracted to create a logic cone for locking. SARLock/Anti-SAT were implemented on the output cone, and then the bypass circuitry was embedded on the locked cone. As for the key length, for a benchmark with N inputs, the SARLock key length is N whereas the Anti-SAT (SI) key length is $2N$. We excluded random integration (RI) for Anti-SAT because of the aforementioned scalability issues of our bypass attack. For SARLock+SLL, we added 32 randomly inserted key gates which makes the key length $N + 32$. In terms of tools, we employed the Python extension of Cryptominisat [17] for finding the DIPs to bypass, and used the ABC synthesis tool [18] to estimate the area/delay overhead of the final bypassed circuit (after optimizing/resynthesizing them using the commands *strash* $\rightarrow$ *refactor* $\rightarrow$ *rewrite*).

Bypass Circuitry Overhead The basis of our attack is that we are able to embed a bypass circuitry to circumvent SAT-resistant logic locking. However, the area/delay overhead consumed by the bypass circuitry itself cannot go unnoticed. Therefore, from an attacker's perspective, the relevant metrics for attack efficacy would be area and delay overhead from the bypass circuitry. Area and delay overhead are estimated by the increases in design gate count and number of levels in the output cone, respectively, from the original as well as locked design.

Table 1 shows the area/delay overhead from integrating the bypass circuitry on designs locked with SARLock and Anti-SAT. For most of the benchmarks, we can see that the there is actually a considerable improvement in area/delay overheads (compared to the locked designs). This is because we applied resynthesis to the bypassed circuit [6]. Since the bypassed design has hard-coded SARLock/Anti-SAT key values, resynthesis leads to a considerable portion of the locking circuitry being automatically eliminated/merged with other gates. However, there is a slight increase in area/delay overheads compared to the original design (as seen in the columns under "over original"). Note that these overheads scale mostly as

[6] Note that if resynthesis were not applied, we can expect to see an area overhead in line with Equation 1, as shown in Fig. 5(b).

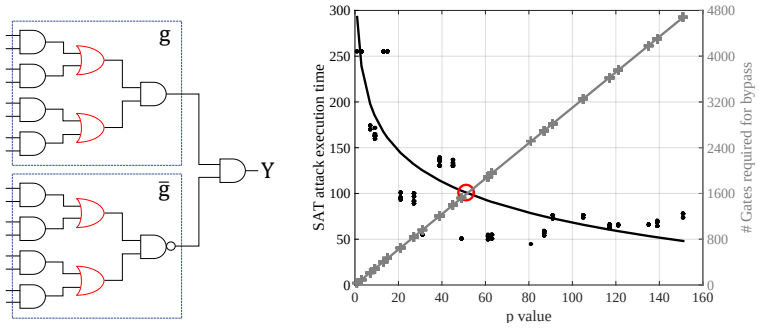
Benchmark	Gate Count	Cone Gate Count	SARLock						Anti-SAT					
			Locked Cone		Bypass (over locked)		Bypass (over original)		Locked Cone		Bypass (over locked)		Bypass (over original)	
			Area %	Cone Delay %	Area %	Cone Delay %	Area %	Cone Delay %	Area %	Cone Delay %	Area %	Cone Delay %	Area %	Cone Delay %
C432	160	105	83.13	29.03	-43.69	-32.5	3.13	-12.9	394.74	29.03	-23.23	-15	48.75	9.68
C880	383	80	44.19	110.53	-25.19	-52.5	0.78	0.00	50.5	111.11	-9.72	-10.53	26.11	88.89
C1908	880	522	15.36	37.04	-11.19	-27.03	0.11	0.00	18.11	37.04	-4.74	-21.62	9.55	7.41
C3540	1669	354	7.02	178.57	-4.38	-66.67	0.84	-7.14	8.26	178.57	-3.94	-30.77	2.22	92.86
C5315	2297	184	6.6	166.67	-5.86	-62.5	0.00	0.00	7.38	166.67	-6.51	-62.5	0.00	0.00
C7552	3512	493	10.08	476.47	-8.56	-77.55	-0.2	29.41	11.68	476.47	-9.76	-77.55	-0.2	29.41
apex2	1522	583	11.04	11.76	-3.68	-10.53	-0.33	0.00	11.67	11.76	-3.87	-10.53	-0.33	0.00
sqrt	16998	884	0.63	0.86	-0.61	-1.7	-0.01	-0.86	0.7	0.86	-0.33	0.43	0.33	1.29

Table 1: Area, delay overheads for implementing bypass circuitry on SAT-resistant circuits.

a function of the number of primary inputs N in the circuit (see Equation 1). For designs with few primary inputs and large number of gates, the overhead becomes negligible (e.g., $\approx 1\%$ area/delay overheads for benchmarks *apex2, sqrt*).

Attack Time. From the attacker's perspective, execution time is also important. The execution time to generate the DIPs to bypass for SARLock and Anti-SAT is < 2 seconds for all the benchmarks. Note that the scalability/run-time of our attack is limited only by the number of variables/clauses (from the circuit's CNF representation) that can be handled by the SAT solver (which only needs to be called twice for the two DIPs). Arbitrarily large sequential circuits could also be bypassed (provided there is scan access), because the SAT-resistant scheme is only applied to a few combinational logic cones in the circuit. These are usually much smaller than the entire circuit.

We also implemented Algorithm 1 using the Python wrapper for Cryptominisat, and used it to extract a bypass key for the hybrid version of SARLock (i.e., SARLock + SLL, with 32 bit XOR keys inserted randomly into the netlist). For the locked output cone of the C3540 benchmark, the code converged to the final key with the correct SLL portion in 442 iterations (i.e., 442 input-output observations). Similarly, for the C432 benchmark, the code took 651 iterations. For apex2, the number of iterations was 820. The run-time for the SAT solver on these benchmarks was on the order of 5-15 minutes. The run-times were higher as we used the Python wrapper for Cryptominisat (not the native C++ version). We do not present area/delay results for bypass attack on hybrid SARLock, as they are identical to the results for standalone SARLock (the bypass circuit only depends on the no. of inputs).



(a) Varying g and $\bar{g}$ (b) Tradeoff between SAT and bypass

Fig. 5: (a) Alternative Anti-SAT construction for g and $\bar{g}$ to vary p (by changing few AND gates to OR gates) (b) Trade-off between SAT resistance and bypass circuitry overhead on varying p. Each data point is a 16 key bit Anti-SAT block with varying p. Maximum value of p is 255. However, the graph is symmetric before and after $p \approx 127$. Therefore, only the first half is shown.

4.2 Comparison to State-of-the-Art

Table 2 shows a comparison of various logic locking countermeasures and applicable attacks, where a ✓ (✗) denotes that an attack can (can not) break a particular logic locking method. The table shows that SAT attack applies only to SLL. Removal attacks can apply to Anti-SAT and SARLock [13]. Bypass attack applies to all of the techniques except SLL. Note that the bypass attack may or may not scale to Anti-SAT (RI), which is why it has a ✓as well as ✗. Furthermore, bypass attack is complementary to both SAT and removal attacks.

– *SAT Attack:* The parameter p for Anti-SAT is directly proportional to N_{DIP} [7]. As discussed earlier, a low (high) value of p (and therefore, N_{DIP}) implies higher (lower) SAT resistance. However, the overhead of the bypass attack (see Eq. 1) increases linearly with N_{DIP} (and therefore, p). This implies that there is a tradeoff between these two attacks, which can be seen in Fig. 5. As one attack becomes more effective (i.e., time complexity of SAT decreases, bypass circuit overhead decreases), the other attack becomes less effective (i.e., time complexity of SAT increases, bypass circuit overhead increases).
It should also be noted that when p is modified by changing the construction of the Anti-SAT block (as shown in Figure 5a), there is a chance that some patterns can be missed by the miter construction (as explained in Section 3).

[7] Note that in [8], p refers to the output one count of the function g. When p is very low (i.e., 1) or very high ($2^N - 1$, where N is the number of inputs to the Anti-SAT block), SAT attack becomes difficult. For values of p between 1 and $2^N - 1$, SAT resistance decreases. In the discussion here, a high value of p refers to $p \approx \frac{2^N-1}{2}$.

Attacks	*Countermeasures*				
	Regular Logic Lock (SLL)	SARLock	SARLock + SLL	Anti-SAT (SI)	Anti-SAT (RI)
SAT	✓	✗	✗	✗	✗
Removal	✗	✓	✗	✓	✓
Bypass	✗	✓	✓	✓	✗/✓

Table 2: A comparison of various logic locking techniques, attacks and countermeasures.

The number of patterns remaining undetected will depend on the key chosen for bypass, and the boolean function obtained by the modified Anti-SAT block. In any case, the trade-off observation still holds. A higher p value implies a higher chance of undetected patterns, higher overhead for bypass but also much lower SAT resistance.

– *Removal Attack:* Anti-SAT (RI) cannot always be efficiently attacked using bypass attack. However, it is vulnerable to removal attacks, if the Anti-SAT block is not obfuscated using additional key gates. Further, SAT resistance is also lowered as discussed in Section 3.5.

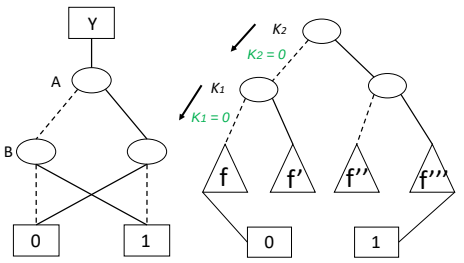
Therefore, for any secure logic locking scheme, all the aforementioned attacks need to be considered in unison.

5 Countermeasure Exploration and Trade-off Assessment

5.1 Binary Decision Diagram

In order to better understand the tradeoffs discussed above (complementary nature of the attacks), we propose logic locking at the functional level using binary decision diagrams (BDDs) instead of at the netlist level. BDDs are graph-representations of Boolean functions that have been extensively used in the past decade for synthesis and formal verification. A BDD is able to represent the entire input space of a Boolean function in a compact form. An example of a BDD for a simple XOR function $Y = A \oplus B$ is shown in Fig. 6(a), where the variables A, B are represented as nodes. Dashed lines represent a variable (A, B) equaling logic '0' and solid lines represent a variable (A, B) equaling logic '1'.

Given a BDD representation of a combinational circuit, a simple logic locking scheme is shown in Fig. 6(b). K_1, K_2 are new variables added to the BDD. In this scheme, application of the correct key $\{K_1 = 0, K_2 = 0\}$ allows the BDD to exert the original circuit functionality f. Application of any other (wrong) key causes the circuit to perform functions f', f'', f''', and so forth which are different from the original function f, as shown in Fig. 6(c). In order to develop SAT attack resistance at the BDD level (for $p = 1$), we need to make sure that every wrong key value leads to a function f', f'', etc. that has Hamming Distance from f equal to 1. This causes a 1-bit flip when the wrong key is used. Further, any arbitrary values of p (or N_{DIP}) can be accommodated by the BDD.



(a) (b)

A	B	key inputs and functions			
		00	01	10	11
		f	f'	f''	f'''
0	0	0	0	0	0
0	1	1	**0**	1	1
1	0	1	1	**0**	1
1	1	0	0	0	1

(c)

Fig. 6: (a) BDD representation of an XOR function. (b) Logic locking at the BDD Level. (c) Every wrong key value leads to a function that has Hamming Distance from f equal to 1.

We summarize the benefits of BDD-based logic locking below.

- *Balancing Bypass and SAT Resistance:* As shown in Fig. 5, there is clearly a tradeoff between SAT attack execution time and bypass attack feasibility. Since BDDs permit arbitrary values of p, it would be possible to find the point of intersection in Fig. 5. As a designer, this is the best-case scenario for logic locking, since it balances the highest SAT attack execution time with the highest bypass cost for the attacker. In addition, by knowing this point of intersection, the designer can determine whether logic locking provides enough protection against piracy.
- *Removal and Sensitization Attacks*: Unlike Anti-SAT/SARLock which inverts the circuit at a single net, BDD-based obfuscation represents the Boolean function as a digraph, embeds key gates and introduces the obfuscated functions as part of the original logic circuit. It would not be possible to isolate the original function f from the obfuscated functions f', f'', f'''. Therefore, there is no tradeoff involved for mitigating removal attacks. Further, sensitization attacks that try to propagate a single key value to the output are also difficult [10], as (i) all key values converge to the same BDD output and (ii) a key vector appears as a graph traversal path (not as individual key gates).

Therefore, BDDs could be viewed as a platform for simultaneously assessing all known threats against logic locking.

However, we've also identified a shortcoming of BDD-based logic locking – area overhead. Table 3 shows the results from applying the proposed BDD-based logic locking scheme with SAT attack resistance. The BDD transformation of the original circuit and subsequent embedding of key inputs (10 bits long) was performed in the CUDD environment, using iterative ITE operations [19]. From the table, we can observe that the SAT attack tool takes a number of iterations that is, at the least, exponential in the size of the SAT-resistant key-length (i.e., # iterations $\geq 2^{10}$ for 10-bit key). Unfortunately, the area overheads are also observed to be extremely high. This is expected because for SAT resistance, every

Benchmark	Hybrid BDD Obfuscation			
	Area Overhead /%	Iterations for SAT Attack	SAT Attack Time (s)	Build/Lock Time (s)
C880	4090.72	1457	3049.3	1.08
C1908	3314.89	1268	1839.5	0.56
C3540	1286.9	1034	2161.3	3.18
dalu	1171.99	1075	821.6	0.56
apex2	535.58	1028	1789.9	0.37

Table 3: BDD-based Logic Locking with SAT Resistance: Each benchmark was logic locked for SAT resistance with BDDs (w/ 10 bit key) and 32 key gates were then introduced to increase the key length. *Build/Lock Time* indicates the total time required to build the BDD for the selected output logic cone of the benchmark, and to introduce the 10-bit SAT-resistant locking.

wrong key value $(2^n - 1)$ leads to a separate BDD with a unique DIP. Although several BDD size reduction techniques exist (e.g., changing the variable orders as they appear in the BDD, BDD-based logic optimization), we noticed that for SAT resistance, the size of the locked BDD is almost always exponential in the key length, as seen in Fig. 7. Also, the BDD tool could read in and build a BDD for all the benchmarks in the ISCAS'85 benchmark set (with the exception of C6288, which is a multiplier). The node count for the BDD of the largest benchmark (C7552) was 16K nodes, with regular sifting-based reordering and *without* any resynthesis of the BDD. Since this is clearly much bigger than the original gate count (3.5K), it is recommended that BDD-based locking be performed on a per-output basis (i.e., extract transitive fan-in cone of an output, convert the cone to BDDs, lock and then merge with the cones of the other outputs which have not been converted to BDDs).

In order to further combat the area overhead limitation, three avenues can be pursued.

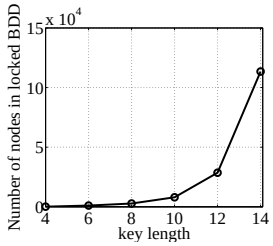


Fig. 7: Growth of no. of nodes (area) as a function of key length for SAT attack resistant BDD locking on the C5315 benchmark

– The SAT-resistant key can be shortened, and regular logic locking (i.e., embedding XOR, MUX key gates) can be performed on the circuit generated from BDD-based obfuscation. This prevents an attacker from brute forcing a

short key. In fact, for the results in Table 3, we incorporated 32-bit XOR-based logic locking into the resultant circuit after BDD-based locking. However, the extra key gates introduced *do not* increase the circuit's SAT resistance capabilities. The SAT attack tool's solving time is only limited as a function of the Anti-SAT keys, not the regular logic locking keys (as these keys only add a minimal number of DIPs for the attack). Further, the attack in Algorithm 1 could be used to directly obtain keys in the SAT-resistant key space, which can then be used for bypass.

- Another option is to embed the BDD-based obfuscation on multiple outputs of the circuit, with a short key dedicated to each output. This allows the key length of the circuit to increase without exponential area blow-up. However, the number of DIPs required by the SAT attack will now be linear in the size of the key. For example, a circuit with outputs Y_1 and Y_2 is locked using the BDD-based approach. Output Y_1 and Y_2 are locked with keys K_1 and K_2 respectively. Provided that these outputs do not share any DIPs, the number of iterations required by the SAT solver will now be lower bounded by $(2^{|K_1|} - 1) + (2^{|K_2|} - 1)$ iterations, and not $2^{|K_1|+|K_2|} - 1$. However, area will only grow linearly as a function of the number of locked outputs. Due to these reasons, there is again, an inherent tradeoff in SAT resistance and area overhead when doing BDD-based logic locking.
- The area overhead from BDD-based logic locking is also a direct result of sub-optimal logic synthesis from BDDs. Note that BDDs can be further optimized by better variable ordering or logic decomposition [20]. Unfortunately, techniques and tools for synthesizing circuits from BDDs are still scarce. More research in this domain could make BDD-based logic locking more feasible.

5.2 Parametric Tests

As shown in Fig. 4(a), the bypass attack is implemented by adding extra circuit to decrypt the locked netlist. The area and delay overhead of the bypassed IC copies would be different from the original (locked) ones, and therefore can be potentially identified by so called parametric tests such as side-channel measurements. As the original IC is larger, the detection becomes possible since the size of the bypass circuit also increases, as depicted in Fig. 5. However, there are several issues that prohibit the implementation of these parametric tests in practical scenarios:

1. The existence of process variations between different ICs would introduce uncertainty into side-channel leakage and limit the effectiveness of the parametric tests.
2. Motivation for consumers in the market to undertake such an effort is weak. Consumers usually want the cheapest chip, regardless of whether it contains pirated IP. Our results in Table 1 show that the pirated IP can perform even better (in terms of overhead) after re-synthesis than the obfuscated circuit.
3. It is already common practice for design houses to use reverse-engineering (RE) companies (e.g., TechInsights) to physically RE the IP of competitors for litigation purposes, which would be more effective than parametric tests.

With the reasons above, we argue that although it may be possible to use parametric tests as a countermeasure against bypass attacks, practical concerns like detection accuracy and cost would likely limit their applicability.

6 Conclusion

In this paper, we presented a novel bypass attack that can thwart SAT-resistant logic locking schemes. The only overhead from our attack is a small bypass logic that can be stitched onto the SAT-resistant circuit. We also assessed how all existing attacks on logic locking can complement each other. Specifically, high SAT attack resistance corresponds to low bypass resistance and vice-versa. The only Anti-SAT locking technique that is somewhat resistant to our bypass attack is still vulnerable to removal attacks. We also introduced a BDD-based logic locking approach for analyzing these competing attacks and simultaneously balancing them. Finally, we highlighted the challenges and future work required to make BDD (and in general, secure logic locking approaches) more practical.

Acknowledgment

This research is supported in part by Cisco Systems Inc, and by the AFOSR award number FA9550-14-1-0351.

References

1. M. M. Tehranipoor, U. Guin, and D. Forte. Counterfeit integrated circuits. In *Counterfeit Integrated Circuits*, pages 15–36. Springer, 2015.
2. K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi. Efficient and secure intellectual property (ip) design with split fabrication. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 13–18. IEEE, 2014.
3. Y. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX security*, pages 291–306. Boston MA, USA, 2007.
4. J. A. Roy, F. Koushanfar, and I. L. Markov. Epic: Ending piracy of integrated circuits. volume 43, pages 30–38. IEEE, 2010.
5. J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault analysis-based logic encryption. *IEEE Transactions on Computers*, 64(2):410–424, 2015.
6. 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*, pages 137–143. IEEE, 2015.
7. M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. Sarlock: Sat attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, May 2016.
8. Y. Xie and A. Srivastava. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 127–146. Springer, 2016.

9. R. Torrance and D. James. The state-of-the-art in ic reverse engineering. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 363–381. Springer, 2009.
10. J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference*, pages 83–89. ACM, 2012.
11. M. Bushnell and V. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2004.
12. M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1411–1424, 2016.
13. M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. Security analysis of anti-sat. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 342–347. IEEE, 2017.
14. Y. Shen and H. Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 179–184, New York, NY, USA, 2017. ACM.
15. F. Brglez. A neutral netlist of 10 combinational benchmark circuits and a target translation in fortran. In *ISCAS-85*, 1985.
16. L. Amarú, P.-E. Gaillardon, and G. De Micheli. The epfl combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, number EPFL-CONF-207551, 2015.
17. M. Soos. Cryptominisat–a sat solver for cryptographic problems. *URL http://www. msoos. org/cryptominisat4*, 2009.
18. R. Brayton and A. Mishchenko. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pages 24–40. Springer, 2010.
19. F. Somenzi. Cudd: Cu decision diagram package release 2.3. 0. *University of Colorado at Boulder*, 1998.
20. C. Yang and M. Ciesielski. Bds: A bdd-based logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):866–876, 2002.

A Bypass Attack on Anti-SAT with Secure Integration

Note that in this proof, we follow the notation and terminology in Section 2.2. Following the notation in [8], we denote the n-bit inputs to the Anti-SAT block with $\mathbf{X}$. In the *secure integration* mode, $\mathbf{X}$ are directly connected to the primary inputs ($\mathbf{IN}$, width of which might be larger than n) of the netlist, as shown in Fig. 3(a). If $(|\mathbf{IN}| - |\mathbf{X}|) > 0$, then those input wires not connected with Anti-SAT block become "don't cares" for it. The existence of such "don't cares" makes it easier for attackers, since $|\mathbf{X}|$ is not maximized, which means $|\mathbf{K}|$ is not maximized since $|\mathbf{K}| = |\mathbf{X}|$(as shown in Fig. 3(b) and (c)). If our attack works when $|\mathbf{IN}| = |\mathbf{X}|$ then it should also work when $|\mathbf{IN}| > |\mathbf{X}|$. Therefore, in the following discussion we shall simply assume that $|\mathbf{IN}| = |\mathbf{X}|$.

Lemma 1. *Given a wrong key to Anti-SAT of secure integration mode, for all n-bit input patterns: $\boldsymbol{X} = \mathbb{B}^n$, $\mathbb{B} = \{0, 1\}$, there exists one and only one DIP.*

Proof. First of all, note that to make it more understandable, our proof is based on the same notation and terminology as [8]. Given a Boolean function $g(\mathbf{L})$ with n-bit inputs, we can divide the input vectors $\mathbf{L}$ into two sets: $\mathbf{L}^1$ and $\mathbf{L}^0$, which represent the inputs that make the Boolean function g equal to 1 and 0. If we denote $|\mathbf{L}^1| = p$, we can get:

$$\begin{array}{l} \mathbf{L}^1 = \{\mathbf{L} | g(\mathbf{L}) = 1\}, \quad (|\mathbf{L}^1| = p) \\ \mathbf{L}^0 = \{\mathbf{L} | g(\mathbf{L}) = 0\}, \quad (|\mathbf{L}^0| = 2^n - p) \end{array} \tag{2}$$

We define all $2n$-bit keys for Anti-SAT with $\mathbf{K} = < \mathbf{K}^{l1}, \mathbf{K}^{l2} > = \mathbb{B}^{2n}$, $\mathbb{B} = \{0, 1\}$, in which $\mathbf{K}^{l1}$ and $\mathbf{K}^{l2}$ stand for two n-bit key inputs connected to the Anti-SAT components g and $\overline{g}$ ($l1$ and $l2$ refer to the locations of g and its complementary function $\overline{g}$ in the netlist, as shown in Fig. 3). Assuming $\mathbf{X}^d$ denotes a set of DIPs, and $\mathbf{Y}^d$ stands for corresponding outputs of Anti-SAT, then for the wrong key set $\mathbf{WK}_i = < \mathbf{K}_i^{l1}, \mathbf{K}_i^{l2} >$ ruled out at the i^{th} iteration of SAT attack by a DIP X_i^d and Y_i^d, we can get:

$$Y_i^d = g(X_i^d \oplus \mathbf{K}_i^{l1}) \wedge \overline{g(X_i^d \oplus \mathbf{K}_i^{l2})} = 1 \tag{3}$$

From Eq. 2 and 3, we can deduce that:

$$(X_i^d \oplus \mathbf{K}_i^{l1}) \in \mathbf{L}^1 \quad and \quad (X_i^d \oplus \mathbf{K}_i^{l2}) \in \mathbf{L}^0 \tag{4}$$

Note that X_i^d is a input vector, thus $|\mathbf{K}_i^{l1}| = |\mathbf{L}^1| = p$. By defining the elements in $\mathbf{K}_i^{l1}$ as $\{K_{i\_1}^{l1}, K_{i\_2}^{l1}, \dots K_{i\_p}^{l1}\}$, and corresponding XORed results in $\mathbf{L}^1$ as $\{L_1^1, L_2^1, \dots L_p^1\}$, we can get:

$$(X_i^d \oplus K_{i\_o}^{l1}) = L_o^1 \in \mathbf{L}^1, \quad o \in [1, 2, \dots p] \tag{5}$$

In Eq. 5, $K_{i\_o}^{l1}$ stands for a wrong key vector for g, thus according to the properties of XOR operation, the following equation holds true, $\forall X_j \in \mathbf{X}$, if $X_j \neq X_i^d$:

$$(X_i^d \oplus K_{i\_o}^{l1}) \neq (X_j \oplus K_{i\_o}^{l1}), \quad o \in [1, 2, \dots p] \tag{6}$$

As proven in [8], if the output-one count p of Anti-SAT block g is sufficiently close to 1, attackers are forced to iterate *at least* 2^n keys to reveal the correct one(s). That is, the SAT-resistance capability of Anti-SAT is maximized when p is 1. The authors of [8] proposed to use AND and NAND gates to realize this goal, as shown in Fig. 3(c). This implies that $|\mathbf{K}_i^{l1}| = |\mathbf{L}^1| = 1$, if this wrong key $K_{i\_1}^{l1}$ is applied on the Boolean function g, then output becomes:

$$g(X \oplus K_{i\_1}^{l1}) = \begin{cases} 1, & \text{if } X = X_i^d \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

The total number of 0 outputs from $g(\mathbf{X} \oplus K_{i\_1}^{l1})$ is $2^n - 1$, this means that for $2^n - 1$ input vectors of $\mathbf{X}$, the outputs $\mathbf{Y}$ of Anti-SAT block will be 0, i.e., the original outputs are not flipped. Note that there must exist *at least* one input corresponds to an output $Y = 1$, since otherwise, it violates the definition of a wrong key.

Conclusion: in *secure integration* mode, there exists one and only one DIP for any wrong key.