SARLock: SAT Attack Resistant Logic Locking

Muhammad Yasin[†], Bodhisatwa Mazumdar[‡], Jeyavijayan (JV)[ξ] Rajendran and Ozgur Sinanoglu[‡]
yasin@nyu.edu, bm105@nyu.edu, jv.ee@utdallas.edu, ozgursin@nyu.edu
† Electrical and Computer Engineering, NYU Tandon School of Engineering, NY, USA
ξ Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas, TX, USA
‡ Electrical and Computer Engineering, New York University Abu Dhabi, Abu Dhabi, U.A.E.

Abstract—**Logic locking is an Intellectual Property (IP) protection technique that thwarts IP piracy, hardware Trojans, reverse engineering, and IC overproduction. Researchers have taken multiple attempts in breaking logic locking techniques and recovering its secret key. A Boolean Satisfiability (SAT) based attack has been recently presented that breaks all the existing combinational logic locking techniques. In this paper, we develop a lightweight countermeasure against this and other attacks that aim at gradually pruning the key search space. Our proposed logic locking technique, referred to as SARLock, maximizes the required number of distinguishing input patterns to recover the secret key. SARLock thwarts the SAT attack by rendering the attack effort exponential in the number of bits in the secret key, while its overhead grows only linearly.**

I. Introduction

The evolving complexity of Integrated Circuits (ICs) and the skyrocketing costs of building or maintaining a semiconductor foundry have propelled the globalization of IC design and manufacturing flow [1]. Design houses may purchase Intellectual property (IP) cores from third-party IP vendors to reduce their design effort and meet strict time-to-market constraints. Many companies, such as Apple, operate fabless and outsource the fabrication to offshore foundries. In a globalized IC supply chain, untrusted agents may obtain access to the valuable IP or the physical IC, which gives rise to security threats. These malicious agents can pirate the IP, overbuild ICs for illegal sale, tamper the ICs to insert malicious circuitry in the form of Hardware Trojans (HTs), or reverse engineer the netlist from an IC for unlicensed use [2].

Countermeasures such as IC camouflaging [3], split manufacturing [4], and IC metering [5] have been developed to thwart these attacks. *Logic locking*[1] is a set of techniques that thwart IP piracy, overbuilding, and reverse engineering attacks by locking a design with a secret key [6], [10]–[12]. To enable chip-locking features, additional logic, e.g., XOR/XNORs gates referred to as *key gates*, is added to the original netlist to obtain a *locked netlist*.

Figure 1(a) shows an example netlist, and Figure 1(b) shows its locked version through three XOR/XNOR key gates. One of the inputs of each key gate is driven by a wire in the original design, while the other input, referred to as *key input*, is driven by a key bit stored in a tamper-proof memory. The inverters in the netlist can be moved around to increase the obfuscation complexity as shown in Figure 1(c): An attacker will not know whether the inverter is a part of original design or added for logic locking. This locked netlist passes through the untrusted design/fabrication stages. A locked IC (or a locked netlist) will not generate correct output unless it is activated using the correct key. Even if an attacker gets access to the locked netlist with the key gates, either by stealing in design house or by reverse engineering the masks in the foundry, the netlist could implement one of exponentially many functions determined by the key value unknown to the attacker.

Several researchers have exploited the weaknesses of different combinational logic locking techniques and developed attacks that recover the secret key [8], [11], [13], [14]. An introduction of the logic locking techniques and a summary of the recent attacks are presented in Section VI. While specific attacks focus on specific weak points of a logic locking technique, a more general attack was presented at HOST 2015 that breaks all existing combinational logic locking techniques, recovering the secret key within a few hours for most of the locked circuits [14].

The attack uses Boolean satisfiability (SAT) based algorithms and we refer to it as *SAT attack* [14]. In the threat model of the attack, the attacker is a malicious agent in the foundry with access to the following:

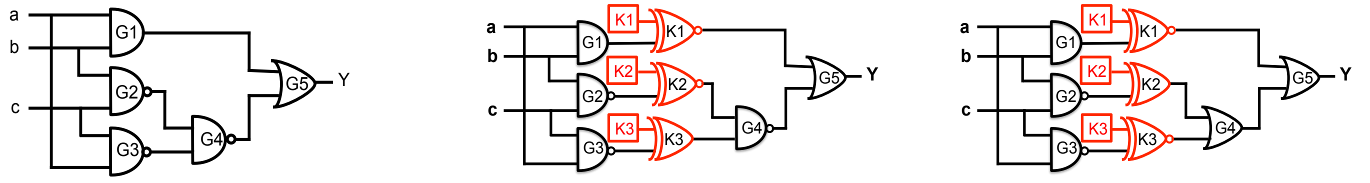
1) a locked netlist with the key gates.
2) a functional IC with the correct key embedded inside. The attacker can apply inputs to the IC and observe the outputs.

The objective of the attacker is to extract the secret key. The SAT attack uses modern SAT solvers to compute special *distinguishing input patterns (DIP)* [14]. These patterns, along with the correct output collected from the functional IC, are used to reduce the key search space by eliminating the incorrect keys. A single distinguishing input pattern may eliminate/discriminate multiple incorrect key values. The attack is successful when all incorrect key values are eliminated.

This paper focuses on defense against the SAT attack [14]. The specific contributions of this paper are as follows:

1) We analyze the strengths and weaknesses of the SAT attack [14]. Based on the discriminating ability of individual input patterns, we present a scenario that is hardest for the SAT attack to break.
2) We present a lightweight ***S**AT-**A**ttack **R**esistant Logic **Lock**ing (SARLock)* technique that thwarts key-distinguishing attacks. The proposed technique adds only a few XOR/XNOR gates; however, the attack effort increases exponentially with the number of key bits. We demonstrate the effectiveness of our approach using empirical attack results.
3) We demonstrate that SARLock can be used in conjunction with existing logic locking techniques to protect against a wide spectrum of attacks.

[1] Researchers have previously used the terms "logic obfuscation" [6]–[9] and "logic encryption" [10] for this purpose. However, echoing the call for consistent terminology in [11], we use the term "logic locking."



(a) Example circuit: majority of inputs [14].

(b) Circuit locked using XOR/XNOR key gates. The correct key is value is 110.

(c) Locked circuit with inverters absorbed by the key gates. The correct key value is 110.

Fig. 1: Logic locking using XOR/XNOR gates [12]. This technique is vulnerable to SAT attack [14].

4) We present an analysis of the lower bound on the number of distinguishing input patterns ($\#DIPs$) needed for a successful SAT attack.

II. Preliminaries: The SAT attack [14]

The SAT attack iteratively rules out incorrect key values using distinguishing input patterns [14]. A distinguishing input pattern X_d is an input value for which at least two different key values, $k1$ and $k2$, produce differing outputs, $o1$ and $o2$, respectively. Since $o1$ and $o2$ are different, at least one of the key values or both of them are incorrect. It is possible for a single DIP to rule out multiple incorrect key values.

The DIPs are found by constructing a miter-like circuit as illustrated in Figure 2. The primary inputs are common to the two copies of the locked circuit, while the key inputs are left independent. The corresponding outputs of two circuits are XORed and then ORed to generate *diff* signal. The conjunctive normal form (CNF) of the resultant circuit is generated and passed to a SAT solver. The SAT solver finds a DIP X_d for which *diff=1*, i.e., the outputs of the two circuits are different. X_d is applied to the functional IC, and correct output I_d is obtained. The input-output pair (X_d, I_d) is used to identify incorrect key values. However, a single pattern may not rule out all incorrect keys. To rule out the remaining incorrect key values, more distinguishing patterns are needed. A new pair (X_d, I_d) is added to the SAT formula in each iteration and the SAT formula is updated. **The attack is successful when no further DIP is found, which implies that all incorrect key values have been pruned.**

Example. Let us consider the application of the SAT attack on the example locked circuit in Figure 1. Figure 3 presents the output of the original circuit in column Y, and the output of the locked circuit for different key values in the following columns. For three key inputs, there are eight possible key values, which are represented as k0, k1,..., k7. When the SAT attack is launched on the locked circuit, it takes four DIPs to identify the correct key [14]. In iteration 1, the DIP 011 is used. For this DIP, the key value k4 alone produces a wrong output as highlighted in red. Thus, only one incorrect key is ruled out in the first iteration. In the second and third iterations, key values k1 and k7 are ruled out, using the patterns 111 and 101, respectively. The pattern 100, used in the fourth iteration, eliminates all incorrect keys and the attack successfully identifies the correct key as k6.

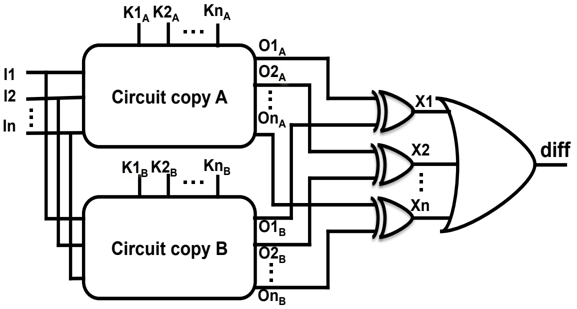


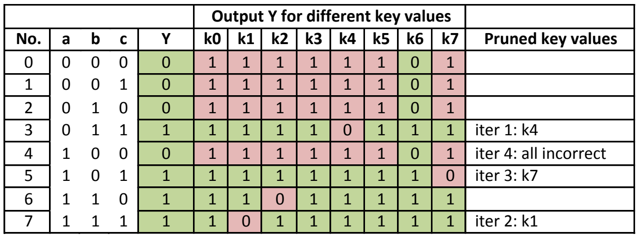
Fig. 2: Miter-like circuit to determine DIPs [14].

The attack could have succeeded in the first iteration with a single DIP 100, if this input pattern was tried first. Thus, the execution time of the attack depends on the order in which the input patterns are applied for the SAT attack. The SAT attack, however, chooses the DIPs arbitrarily [14]. The larger the number of incorrect key values ruled out per DIP, the fewer the patterns needed for the attack, which implies a smaller execution time of the attack algorithm. **We propose to use $\#DIPs$ needed for a successful attack as a metric for evaluating the resilience of a logic locking technique against the SAT attack.**

III. Resisting the SAT attack

The SAT attack is successful against the existing logic locking techniques as the $\#DIPs$ needed for the attack against these techniques is relatively small; 250 or fewer patterns are needed for a successful attack on 90% of the circuits in the study in [14]. The existing logic locking techniques fail to take into account the discriminating ability of individual input patterns and are thus vulnerable to the SAT attack. For example, in Figure 3, all incorrect key values k0-k5 and k7 produce an incorrect output for the DIP 100, and the SAT attack could identify the correct key value using a single input pattern. This represents the best-case scenario for the SAT attack.

The worst-case scenario for the SAT attack arises when the attack can discriminate at most one incorrect key value with each DIP. The truth table in Figure 4 illustrates SAT attack resistant locking for three primary inputs and three



					Output Y for different key values								
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7	Pruned key values
0	0	0	0	0	1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	1	1	1	1	0	1	
2	0	1	0	0	1	1	1	1	1	1	0	1	
3	0	1	1	1	1	1	1	1	0	1	1	1	iter 1: k4
4	1	0	0	0	1	1	1	1	1	1	0	1	iter 4: all incorrect
5	1	0	1	1	1	1	1	1	1	1	1	0	iter 3: k7
6	1	1	0	1	1	1	0	1	1	1	1	1	
7	1	1	1	1	1	0	1	1	1	1	1	1	iter 2: k1

Fig. 3: Analysis of the SAT attack against logic locking [14]. Columns k0-k7 show the locked circuit's output for different key values. Red entries in each row denote an incorrect output. The correct key is k6.

					Output Y for different key values							
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7
0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0	0
3	0	1	1	1	1	1	1	0	1	1	1	1
4	1	0	0	0	0	0	0	0	1	0	0	0
5	1	0	1	1	1	1	1	1	1	1	1	1
6	1	1	0	1	1	1	1	1	1	0	1	1
7	1	1	1	1	1	1	1	1	1	1	1	0

Fig. 4: Resisting the SAT attack by controlling the discriminating ability of input patterns. At most one incorrect key value corrupts the output for any input pattern.

key inputs. In each row, there is at most one key value that generates an incorrect output. As dictated by the truth table in Figure 4, when the SAT attack is launched on a circuit with $|K|$ key bits, $\#DIPs \geq 2^{|K|} - 1$. **Thus, the attack effort grows exponentially with $|K|$.**

A. SARLock: SAT attack-resilient logic locking

To build a SAT attack resistant circuit that implements a truth table similar to the one shown in Figure 4, in a lightweight and scalable fashion, we use a small comparator circuit. The comparator generates a *flip* signal that is asserted for specific input and key value combinations. The *flip* signal will be XORed with one of the primary outputs as shown in Figure 5. To prevent the *flip* signal from being asserted for the correct key value, such as 110 in Figure 4, a small mask logic is inserted. The resulting locked circuit achieves the desired resistance against the SAT attack at minimal overhead. We refer to the proposed IP protection logic as *SARLock*.

For $|K|$ key bits, the additional logic will consist of $|K|+1$ two-input XOR/XNOR gates and $2|K| + 1$ two-input AND gates. On increasing the key size, $|K|$, the area overhead grows linearly, while there is an exponential increase in the $\#DIPs$.

B. Results

In this section, we report the effectiveness of SARLock against the SAT attack using empirical attack results[2]. We compare SARLock with strong logic locking (SLL) [8], since amongst the existing (SAT attack vulnerable) logic locking techniques, SLL offers the best resilience to the other attacks on logic locking [6], [8], [11], [13] (see Section VI).

Table I reports the $\#DIPs$ and the execution time of the attack on SLL [8] for different key sizes $|K|$. SLL can be broken with only a few DIPs. Moreover, the execution time of the attack is below one second for all the benchmark circuits, which demonstrates how vulnerable SLL [8] and existing locking techniques are to the SAT attack.

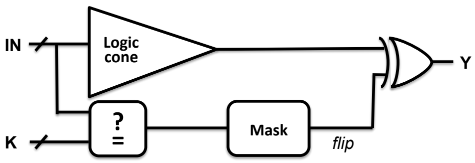


Fig. 5: SAT attack resistant circuit. The *flip* signal is asserted upon a match between an input value and a key value.

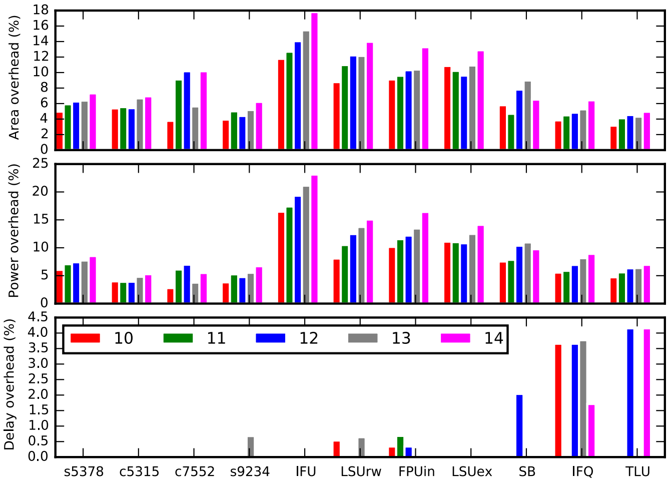


Fig. 6: Area, power, and delay overhead of SARLock for different values of $|K|$.

SARLock, however, exhibits strong resilience against the SAT attack. As shown in Table II, the $\#DIPs$ for SARLock grows exponentially with $|K|$, across all the benchmarks. While the $\#DIPs$ doubles with each increment in $|K|$, the change in execution time can be higher – $3\times$ to $4\times$ for most of the circuits. The execution time varies across the benchmark circuits, because the number of clauses in the SAT formula of each circuit is different. Although the key size $|K|$ used in Table I and Table II is small, these key sizes are considered for comparing $\#DIPs$ and execution time of the SAT attack empirically. While the attack completes within a minute for most circuits for $|K| = 10$, the attack takes about $4-5$ hours for the OpenSPARC controller circuits for $|K| = 14$.

Figure 8 shows that the area and power overhead of SARLock increases linearly with $|K|$, as the number of gates inserted by SARLock grows linearly with $|K|$. However, the benefit from the security perspective is exponential. The average delay overhead of SARLock is less than 0.7%.

C. Provably secure obfuscation

We now provide a security analysis of the SARLock from the provable obfuscation perspective. The inputs to this logic block are primary input IN and the key input K, while the output is a single bit called *flip*. This logic block can be represented as the Boolean function *flip* = $F(IN, K)$.

For each incorrect key guess, K_{incorr}, to the SARLock block, the output bit is different at only one input w.r.t. the correct key input K_{corr}. In other words, the function $F(IN, K_{corr}) \oplus F(IN, K_{corr} \oplus \alpha)$, $\alpha \in \{0,1\}^n \setminus \{0^n\}$ is a one-point function[3]. The incorrect key K_{incorr} is defined as $K_{corr} \oplus \alpha$. The study in [17] shows that the implementation of this one-point function can be provably obfuscated, giving the attacker no advantage beyond having a black-box access to the implemented netlist. For instance, the one-point function can be obfuscated as:

$$F(IN, K_{corr}) \oplus F(IN, K_{corr} \oplus \alpha) = \begin{cases} 1, & \text{if } r^{IN} = r^{K_{corr} \oplus \alpha} \\ 0, & \text{otherwise} \end{cases}$$

[2] The experiments are conducted using ISCAS85 benchmark circuits and OpenSPARC microprocessor controllers [15]. The SAT attack is executed on a server with 6-core Intel Xeon W3690 CPU, running at 3.47GHz, with 24 GB RAM [14], [16].

[3] A point function is a Boolean function that produces the output value 1 at exactly one point.

TABLE I: $\#DIPs$ and the execution time (s) of the SAT attack [14] to break SLL [8] for different values of $|K|$.

	$\#DIPs$					Execution time (s)				
Benchmark	**10**	**11**	**12**	**13**	**14**	**10**	**11**	**12**	**13**	**14**
s5378	8	9	9	10	13	0.2	0.2	0.2	0.2	0.2
c5315	4	3	4	5	3	0.3	0.3	0.3	0.3	0.3
c7552	8	9	9	9	12	0.7	0.5	0.5	0.5	0.5
s9234	7	13	13	10	12	0.2	0.3	0.3	0.3	0.3
IFU	8	8	9	13	11	0.1	0.1	0.1	0.1	0.1
LSUrw	4	5	5	7	9	0.1	0.1	0.1	0.1	0.1
FPUin	6	7	8	5	9	0.1	0.1	0.1	0.1	0.1
LSUex	5	5	8	8	6	0.1	0.1	0.1	0.1	0.1
SB	7	5	6	6	6	0.1	0.1	0.1	0.1	0.1
IFQ	9	7	9	9	8	0.2	0.2	0.2	0.2	0.2
TLU	7	6	7	9	10	0.3	0.3	0.4	0.3	0.4

TABLE II: $\#DIPs$ and the execution time (s) of the SAT attack [14] to break SARLock for different values of $|K|$.

	$\#DIPs$					Execution time (s)				
Benchmark	**10**	**11**	**12**	**13**	**14**	**10**	**11**	**12**	**13**	**14**
s5378	1023	2047	4095	8191	16383	62.6	177.9	996.1	2710.2	9374.6
c5315	1023	2047	4095	8191	16383	79.6	247.4	1188.1	3441.4	11122.5
c7552	1023	2047	4095	8191	16383	73.1	311.1	1276.6	3561.3	11761.5
s9234	1023	2047	4095	8191	16383	77.7	279.0	1235.2	3491.2	12330.9
IFU	1023	2047	4095	8191	16383	32.5	104.3	589.6	2216.8	6650.8
LSUrw	1023	2047	4095	8191	16383	37.3	120.5	618.0	1762.8	6133.0
FPUin	1023	2047	4095	8191	16383	34.9	112.8	650.0	1898.6	6390.0
LSUex	1023	2047	4095	8191	16383	36.0	130.4	690.4	1941.0	7104.9
SB	1023	2047	4095	8191	16383	50.8	149.3	825.2	2412.9	7845.9
IFQ	1023	2047	4095	8191	16383	67.9	232.6	1135.9	2958.1	10521.1
TLU	1023	2047	4095	8191	16383	81.2	271.7	1198.7	3341.4	11628.5

In the exponentiation operation, r is a generator of a multiplicative cyclic group $\mathbb{G}$ and IN, $K_{corr} \oplus \alpha$ are elements of $\mathbb{G}$. If the chosen family of groups is $\mathbb{Z}_p^*$, where p and q are primes and $p = 2q + 1$, then recovering $K_{corr} \oplus \alpha$ from $r^{K_{corr} \oplus \alpha}$ becomes a discrete logarithm (DL) problem. For large sizes of the primes p and q (say 1024 bits), the DL problem is computationally hard. Thus, an attacker cannot extract $K_{corr} \oplus \alpha$ from $r^{K_{corr} \oplus \alpha}$. Under a variant of decisional Diffie-Hellman (DDH) assumption[4] [18], this construction provably satisfies the strong definition of obfuscation. Hence, this technique can be used to hide K_{corr} in the logic block F. However, the implementation of 1024-bit modular exponentiation incurs a large area overhead [19] and is, thus, impractical.

D. Is SARLock alone sufficient?

SARLock successfully thwarts the SAT attack [14]; however, it may not protect against other existing attacks, such as the removal attack and the sensitization attack[5] [10] (see Section VI). In removal attacks, for instance, an attacker identifies the components/gates that belong to the lock circuitry and removes them from the locked netlist. An attacker can obtain the locked netlist either by reverse engineering or by stealing it in the design house. Since, the SARLock logic is isolated and not intertwined with the original circuit, the attacker can easily separate it from the original circuit. To defend against such attacks, SARLock should be coupled with other defenses.

IV. Two-layer logic locking

To provide protection against a wide spectrum of attacks, we propose to integrate SARLock with one of the existing logic locking techniques [8], [10], [12] that are resilient against reverse engineering attacks. From the existing (SAT attack vulnerable) logic locking techniques, we choose SLL [8] since, to the best of our knowledge, SLL offers the maximum protection against the known attacks [6], [8], [11], [13] on logic locking. SLL inserts XOR/XNOR key-gates with increased interference among them and protects a circuit from reverse engineering and sensitization attacks [8] (see Section VI). By intertwining the functional gates and the key gates, SLL hides the implementation of a netlist, thwarting the removal attack.

To lock a particular circuit output, we propose *SARLock+SLL*, a technique that integrates SARLock with SLL. Given $|K| = |K1| + |K2|$ key bits, SARLock+SLL will:

1) lock the logic cone (transitive fan-in of the output) with SLL using $|K1|$ key bits,
2) scramble K2 with K1, and protect the circuit with SARLock using the scrambled K2 bits.

The key K1 has two roles: it serves as the key for SLL, and it scrambles the key K2. Scrambling K2 is important, since otherwise, all the flips will occur in pre-determined combinations of input and key values. Moreover, the scrambler creates a dependency between the K1 and K2, thwarting the removal attack, as will be explained further in Section IV-A. SARLock+SLL is illustrated in Figure 7.

The scrambler logic can be chosen by the designer based on the permissible overheads. For instance, a bus-based IC protection technique [20] authorizes activation of each individual chip by leveraging bit permutations as scrambler for the bus data using a key unique to each IC. Some other techniques comprise logic locking techniques [13] such as XOR gates, arithmetic transformations such as addition and subtraction, bit permutations using Benes network [20], and

[4] The assumption mentions that there is no efficient probabilistic algorithm that, given any triplet $< g^a, g^b, g^c >$ in $\mathbb{G}^3$, outputs "true" if $a = bc$, and "false", otherwise.

[5] Sensitization of an internal line l to an output O implies that any change on l will be observable on O.

TABLE III: $\#DIPs$ and the execution time (s) of the SAT attack [14] to break SARLock+SLL for different values of $|K2|$.

	$\#DIPs$					Execution time (s)				
Benchmark	**10**	**11**	**12**	**13**	**14**	**10**	**11**	**12**	**13**	**14**
s5378	1024	2048	4096	8191	16384	54.1	190.6	619.7	4351.8	10250.7
c5315	1024	2049	4096	8191	16383	75.4	252.9	829.1	4778.2	15874.9
c7552	1025	2049	4096	8191	16386	78.3	234.1	757.0	3165.3	14573.1
s9234	1027	2049	4102	8195	16386	77.2	247.9	864.1	3225.7	15532.3
IFU	1023	2056	4100	8206	16389	55.2	166.7	789.5	2309.8	10258.7
LSUrw	1025	2049	4096	8194	16383	58.2	152.0	626.9	1802.6	7466.6
FPUin	1025	2049	4097	8194	16384	28.4	135.0	1359.6	4497.6	15457.2
LSUex	1024	2049	4096	8194	16384	52.8	268.3	1137.2	3101.3	16707.1
SB	1026	2050	4099	8194	16386	69.2	257.4	1416.6	3304.6	19193.7
IFQ	1024	2048	4098	8192	16384	63.3	290.8	1644.7	4185.4	14563.1
TLU	1027	2052	4099	8195	16385	57.2	227.0	2238.7	3507.6	18760.3

crossbar switches [21].

A. Protection against attacks

The scrambler performs mixing of the keys K1 and K2 to prevent the comparator and the mask circuits from leaking explicit information about K1 and K2. The attacker needs to determine the value of K1 to recover K2 from the scrambler. To determine K1, the attacker needs to perform SAT attack on the locked logic cone. This, in turn, depends on K2 as the IC's output is a function of both K1 and K2, and K2 is the key for the SARLock circuit that resists the SAT attack. This circular dependency not only thwarts the removal attacks, but also enables SARLock+SLL to inherit the protection that SLL offers against sensitization attacks [8].

B. Results

For the SAT attack on SARLock+SLL, we assume that $|K1| = |K2| = |K|/2$. Table III reports the $\#DIPs$ for different values of $|K2|$. Since, SLL [8] can be broken with a small $\#DIPs$, the $\#DIPs$ is increased only minimally by integrating with SLL [8]. The same table shows that the execution time of the SAT attack on SARLock+SLL is, on average, 1.29 times higher than that for SARLock.

The average area, power and delay overhead for SARLock+SLL is 21.27%, 33.3%, and 4.95%, respectively, for $|K2| = 14$. As shown in Figure 8, the overhead grows only linearly with $|K2|$. The average delay overhead for SARLock+SLL is, however, higher compared to SARLock.

Table IV presents a comparison of SARLock+SLL with existing locking logic techniques, random logic locking (RLL) [12] and SLL [8], for $|K| = 64$. While the SAT attack [14] can break RLL [12] and SLL [8] within a second, it will take about 3.1×10^9 seconds $\approx$ 100 years to break SARLock+SLL. The average area, power and delay overhead of SARLock+SLL is 35.2%, 61% and 9.3% respectively, which is comparable to that of RLL [12] and SLL [8].

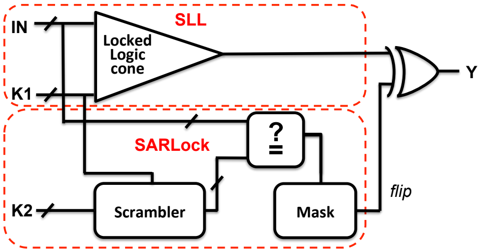


Fig. 7: SARLock+SLL: two layer logic locking. $|K1|$ key bits are used for SLL [8] and $|K2|$ key bits for SARLock.

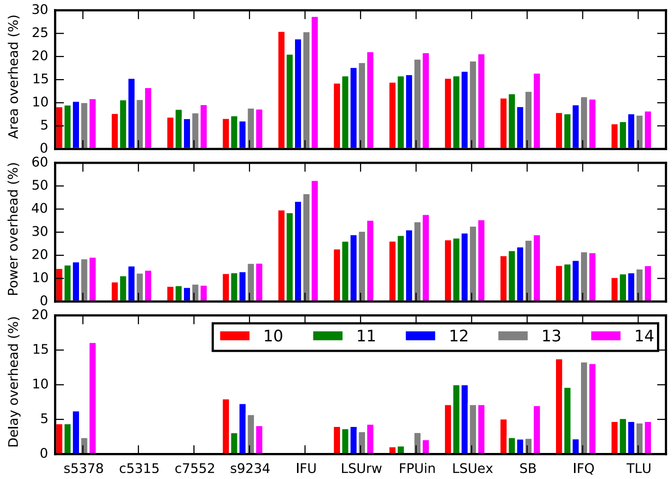


Fig. 8: Area, power, delay overhead for SARLock+SLL for different values of $|K2|$.

TABLE IV: Comparison of SARLock+SLL with RLL [12] and SLL [8] for $|K| = 64$. $\#DIPs$ and execution time are extrapolated based on empirical data.

	RLL [12]	SLL [8]	SARLock+SLL
$\#DIPs$	19	26	4.3E09
Exec. time (s)	0.4	0.7	3.1E09
Area (%)	29.6	32.2	35.2
Power (%)	45.0	59.2	61.0
Delay (%)	15.1	17.2	9.3

V. Discussion

SAT attack resistance vs. corruptibility. SARLock thwarts the SAT attack by corrupting/flipping the output bits selectively, resulting in a small Hamming distance (HD) at the outputs on applying incorrect keys [10]. There is a dichotomy between the two security metrics, $\#DIPs$ and HD. A designer can decide the optimal values of the $\#DIPs$ and the HD, based on the application and the threat model. SARLock is suitable for protecting the output of the control units in microprocessors, where a single bit flip can vastly corrupt the overall sequence of operations [22].

Flipping multiple output bits. SARLock aims to protect the critical logic cones in a circuit while offering the maximum resistance to the SAT attack. Flipping multiple output bits simultaneously increases the HD, and must be traded off carefully with the $\#DIPs$. In a setting where SARLock is designed to flip multiple outputs, its integration with logic locking techniques that increase the HD [10] can be investigated, and is part of our future work.

TABLE V: Comparing logic locking threat models.

Attack	Attacker's assets	Attack method
Sensitization [8]	1) Locked netlist 2) Activated IC	Sensitization of key bits to outputs
Logic cone analysis [6]	1) Locked netlist 2) Activated IC	Brute force on individual logic cones
SAT [14]	1) Locked netlist 2) Activated IC	SAT-based algorithm that rules out incorrect keys iteratively
Hill climbing [11]	1) Locked netlist 2) Test patterns and responses	Key bit flipping guided by the Hamming distance between test response and circuit output

Choice of the key sizes. SLL aims at maximizing the *clique size* in a graph of key gates, dictating $|K1|$. The desired $\#DIPs$ dictates $|K2|$.

Low-overhead SARLock. To reduce the overhead, SARLock can be selectively applied on only the crucial parts of the design. Controllers typically represent the most valuable IP in processors while at the same time occupying a small area ($\approx 1\%$ [5]). In resource-constrained settings, protecting the controllers alone can help achieve the security objectives with a minimal overhead on the overall system.

VI. Related Work

A. Logic locking techniques

- Random logic locking inserts XOR/XNOR key gates at random locations in a netlist [12].
- Fault analysis based logic locking addresses the limitations of RLE and locks an IC such that a random incorrect key corrupts maximum output bits [10].
- Strong logic locking inserts key gates such that it becomes difficult to sensitize key bits to primary outputs on an individual basis [8].

B. Attacks against logic locking

Table V presents a summary of existing attacks against logic locking and the corresponding threat models. The sensitization attack generates key-leaking input patterns by analyzing the locked netlist. These patterns are applied to the functional IC to sensitize the key bits to the primary outputs [8]. The logic cone analysis attack is based on a divide and conquer approach [6]. It identifies the logic cone with the smallest number of key bits and employs brute force to recover the secret key. SARLock can be complemented with SLL to defend against these attacks.

The SAT attack uses Boolean satisfiability techniques to prune the incorrect key candidates (see Section II) [14]. The hill climbing search attack uses test data information to guess the secret key for ICs that are activated prior to the manufacturing test [11]. Unlike the SAT attack, where an attacker can choose I/O pairs of his choice, hill-climbing attack only uses I/O pairs in the test data set. SARLock also defends against the hill-climbing attack, since it increases the $\#DIPs$ to be greater than the number of I/O pairs in the test data set (see Table II and Table III).

VII. Acknowledgement

This work was supported by the National Science Foundation, Computing and Communication Foundations (NSF/CCF) under Grant 1319841.

VIII. Conclusion

We propose a logic locking technique, SARLock+SLL, that thwarts key distinguishing attacks, and in particular, the SAT attack [14], in addition to thwarting all the attacks that SLL protects against [8], [11]. SARLock+SLL increases the required number of distinguishing input patterns exponentially with the key size, by reducing the number of key values filtered in each iteration of the attack. Thus, the execution time of the attack grows exponentially with the key size. The extra hardware inserted by our proposed technique is provably obfuscated to resist reverse engineering attacks. As part of our future work, we plan to explore if SARLock logic alone can be intertwined with the circuit, with minimal loss in $\#DIPs$.

References

[1] "Defense Science Board (DSB) study on High Performance Microchip Supply," 2005, [March 16, 2015]. [Online]. Available: www.acq.osd.mil/dsb/reports/ADA435563.pdf

[2] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[3] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," in *Proc. ACM/SIGSAC Conference on Computer & Communications Security*, 2013, pp. 709–720.

[4] R. W. Jarvis and M. G. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," March 2007, US Patent 7,195,931.

[5] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *Proc. USENIX Security*, 2007, pp. 291–306.

[6] Y.-W. Lee, N. Touba *et al.*, "Improving Logic Obfuscation via Logic Cone Analysis," in *Proc. IEEE Latin-American Test Symposium*, 2015, pp. 1–6.

[7] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, 2009.

[8] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," in *Proc. IEEE/ACM Design Automation Conference*, 2012, pp. 83–89.

[9] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Des. Test. Comput.*, vol. 27, no. 1, pp. 66–75, 2010.

[10] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, 2015.

[11] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 961–971, 2015.

[12] J. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proc. Design, Automation and Test in Europe*, 2008, pp. 1069–1074.

[13] G. K. Contreras, M. T. Rahman, and M. Tehranipoor, "Secure Split-Test for Preventing IC Piracy by Uuntrusted Foundry and Assembly," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2013, pp. 196–203.

[14] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *Proc. IEEE International Symposium on Hardware Oriented Security and Trust*, 2015, pp. 137–143.

[15] "OpenSPARC T1 Processor,," 2015, [Nov 1, 2015]. [Online]. Available: {http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html}

[16] "Decryption tool binaries and benchmark circuits." 2015, [Sep 30, 2015]. [Online]. Available: {https://bitbucket.org/spramod/host15-logic-encryption}

[17] R. Canetti, "Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information," in *Proc. Annual International Cryptology Conference*, 1997, pp. 455–469.

[18] D. Boneh, "The Decision Diffie-Hellman Problem," in *Algorithmic Number Theory*. Springer, 1998, pp. 48–63.

[19] G. D. Sutter, J.-P. Deschamps, and J. L. Imaña, "Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem Based on Digit Serial Computation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3101–3109, 2011.

[20] J. A. Roy, F. Koushanfar, and I. L. Markov, "Protecting Bus-Based Hardware IP by Secret Sharing," in *Proc. IEEE/ACM Design Automation Conference*, 2008, pp. 846–851.

[21] R. Naik and D. Walker, "Large Integrated Crossbar Switch," in *IEEE International Conference on Wafer Scale Integration*, 1995, pp. 217–227.

[22] N. Karimi, M. Maniatakos, A. Jas, and Y. Makris, "On the Correlation between Controller Faults and Instruction-Level Errors in Modern Microprocessors," in *Proc. IEEE International Test Conference*, 2008, pp. 1–10.