Ci-Lock: Cipher Induced Logic Locking Resistant Against SAT Attacks

Akashdeep Saha · Sayandeep Saha · Debdeep Mukhopadhyay · Bhargab B Bhattacharya

Abstract Protection of intellectual property (IP) cores is one of the most practical security concern for modern integrated circuit (IC) industry. Albeit being well-studied from a practical perspective, the problem of safeguarding gate-level netlists from IP-theft is still an open issue. State-of-the-art netlist protection schemes, popularly known as logic locking, are mostly ad-hoc and their security claims are based on experimental evidences and the power of heuristics used for security evaluation. Observing this fact, in this paper we propose a novel logic locking approach, for which the security claims are based on the hardness of well-studied cryptographic primitives. More precisely, for the first time we show that the mapping realized by a circuit netlist (or at least a part of it) can be hidden inside a block cipher by setting a proper secret key. Moreover, this hiding scheme can be realized in a systematic manner with fairly simple heuristics. We claim that extracting the actual mapping is equivalent to a key recovery attack on the cipher, which is believed to be hard for standard block ciphers. The proposed scheme also attains SAT attack resistance directly from the block ciphers which are known to be SAT-hard, in general. Experimental evaluation on ISCAS-85 benchmarks establishes that even for small circuits like C17 (having 6 gates), the proposed approach can successfully throttle SAT-attacks. Further, we argue that the hiding a circuit inside a block cipher is interesting by its own from a theoretical perspective, and may have several useful applications in the domain of security.

A. Saha, S. Saha, D. Mukhopadhyay, B. B. Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
E-mail: {akashdeepsaha95, sayandeep.iitkgp, debdeep.mukhopadhyay, bhargab.bhatta}@gmail.com

1 Introduction

One of the major threats to modern integrated circuit (IC) industry comes from the risk of intellectual property (IP) theft and illegal overproduction. Modern IC manufacturing chain is global and involves several untrusted or partially trusted parties. It is fairly difficult (if not impossible) to control malicious activities within such diverse and complex production network unless some standard security measures are adopted. However, conventional cryptographic primitives and protocols are not directly applicable in the context of IC-supply chain, mainly due to the fact that one cannot simply "encode" a circuit netlist without hindering its actual functionality. This inherent challenge in hardware IP protection has given rise to an entirely new class of security measures popularly known as logic locking. Although logic locking bears some similarity with the extensively studied concept of obfuscation, their connection has never been established formally, and the progress of these two happened in a rather independent manner.
Several alternative suggestions for logic locking have been proposed till date. The core concept behind most of these schemes is to judiciously insert some extra key gates inside the actual circuit netlist, so that original mapping can only be realized if the correct key is provided. The adversary is supposed to obtain this locked netlist without the key. Moreover, she may also have a black-box access to an activated IC implementing the same mapping as the target netlist. The adversary is considered successful if she can somehow extract or realize the actual mapping from the locked netlist. Extracting the key used for locking is the most straightforward approach for doing so. Initial logic locking schemes suggest that the key gates should be inserted either randomly, or at certain specific locations for which the output corruption would be the maximum, upon the application of a wrong key. However, most of these schemes were eventually broken with the introduction of so-called SAT-attack, which intelligently prunes the key space of a locked netlist by exploiting the structural features of the circuit. Till date, SAT attack remains the most standard adversarial technique against logic locking schemes targeting key recovery. However, there exist relatively trivial attack strategies like removal of locking circuitry and use of bypass circuits for mitigating the output corruption due to wrong key guesses. There also exists Side-Channel Analysis (SCA) assisted techniques for key recovery. However, SCA against logic locking is a relatively less exported area and few contributions can be found in literature.
Recent proposals in logic locking describe several alternative strategies for throttling SAT (and other) attacks. The most notable among them are based on one simple observation that *the number of calls to the SAT solver in a SAT attack becomes exponential, if corresponding to each wrong key there exists exactly one input pattern resulting in a wrong output. For the rest of inputs the output will be correct even with the wrong key.* Albeit being simple, one of the major complains against these schemes is their low output corruptibility. Further, the extra logic incorporated for implementing

the locking scheme is often remains distinguishable and removable. Further, in [10] it was shown that such schemes are inherently susceptible to the addition of a bypass circuit for correcting the only wrong output corresponding to a wrong key. In such attacks the netlist can be unlocked even with an arbitrary wrong key. Although some of the recent schemes like [18] claim to provide security against such attacks, in most of the cases the output corruptibility remains low which is not desirable from a security perspective. It has been observed that low output corruptibility is an essential criteria to achieve an exponential blowup in the number of SAT solver calls in SAT-attack. Another alternative strategy for throttling SAT-attack is to make each SAT call sufficiently hard and time consuming for the SAT solver. One great advantage of this alternative approach is that the relation between output corruptibility and attack efficiency does not become a shortcoming anymore. In other words, such an approach is able to provide sufficiently high output corruptibility while still providing resilience against SAT-attacks. Surprisingly, this direction for throttling SAT attack is almost unexplored. One potential example for such is scheme is [3], where it was shown that addition of SAT-hard sub-circuits for logic locking makes each iteration (i.e., SAT call) of SAT-attack prohibitively difficult.

Our Contributions: The present proposal is somewhat motivated by the above mentioned strategy for throttling SAT-attacks. However, instead of embedding ad-hoc SAT-hard sub-blocks inside the circuit, we take an entirely different strategy. The main observation is that *a given block cipher structure is able to hide any arbitrary combinational circuit (with some reasonable bounds in circuit size).* Here, the hiding is achieved as follows: *the original mapping realized by the combinational circuit is activated only if the correct key is provided to the block cipher. For all other cases, the block cipher realizes a key dependent pseudo-random mapping with roughly 50% output corruptibility.* We note that hiding a circuit within a block cipher is an entirely new idea by itself, and may found usage in several other contexts. Some of the fascinating advantages from a logic locking perspective can be listed as follows:

- Block ciphers are fairly well-explored structures from security point of view. Using block cipher for logic locking inherently adopts many of the well-established security claims, and hence provides a higher confidence compared to the ad-hoc schemes proposed till date.
- The hiding operation happens without any significant structural modification of the original block cipher circuit. In some sense, the hiding is similar to the placement and routing of a circuit on an FPGA-like network. In essence, no trace of the actual netlist remains in the locked circuit, which significantly reduces the chances of trivial structural attacks like removal attack [14] or sensitivity analysis attack [14], [5], [6].
- Block ciphers are well-known to be SAT-hard. Hiding within block ciphers thus achieves SAT-attack resilience by default.

– The heuristics for embedding a circuit within a block cipher us fairly simple and efficient. Moreover, *it can be proved that any arbitrary combinational circuit can be embedded in a given block cipher structure and the number of iterative cipher rounds required is polynomial with the gate count of the embedded circuit.*

In this initial draft of the proposal, we only provide some proof-of-concept examples of the proposed approach. More specifically, we show that how small circuits like C17 from ISCAS-85 benchmark suite can be embedded on a lightweight block cipher called PRESENT. SAT-attack results on the locked circuit is also discussed which clearly establishes that the proposed approach is highly promising. The rest of the report is organized as follows. In the next section (Sec. 2), we provide a high level overview of the existing schemes and attacks. The fundamental building blocks for achieving the cipher embedding of a circuit are presented in Sec. 3. Proof-of-concept evaluations on PRESENT and C17 will be provided in Sec. 4 with experimental evidences for SAT-attack resilience. We conclude in Sec. 5.

2 Preliminary

2.1 Logic Locking: The Concept

Logic locking is a technique to hide the functionality of an IC. For an IC to function as desired, correct key is required else the IC behaves anomalously. Early logic locking methods [7] achieved its purpose by inserting key-gates at random position in the circuit. Later Yasin et al. [17] proposed to insert key gates (XOR/XNOR) into cautiously chosen locations such that various shortcomings of randomly inserting key-gates are overcome. The authors have also proposed metric for inserting key gates into the circuit with complex interference among them so that the attackers' effort maximizes in extracting the key from the circuit. They propose to insert key gates in such a way that they form cliques in the circuit graph representation. They aim to maximize the size of these cliques. As these cliques of key-gates require brute force to solve for the key.
After the powerful SAT-based attack was proposed, a new genre of logic locking schemes were developed which aimed to mitigate the SAT attack. They mainly achieved their purpose by increasing the number of SAT iterations or by increasing the time required for each iteration of SAT attack. These techniques are briefly discussed in section 2.3.

2.2 The SAT Attack

The SAT attack [9] works by iteratively eliminating the equivalence class of keys which produce the wrong output value for at-least one input pattern

until the correct key is left behind. It assumes that the attacker has input-output access to an operational chip (as oracle) obtained from the market and the obfuscated circuit netlist. The attack flow is represented in Fig. 1.

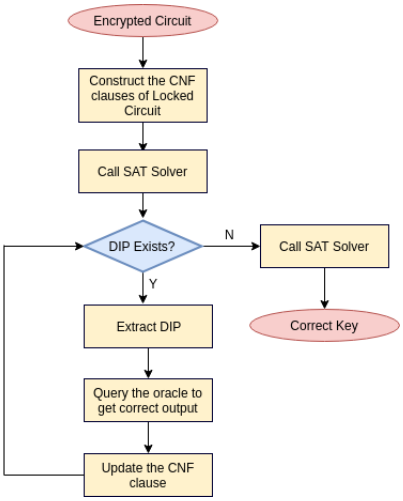


Fig. 1 The SAT attack flow.

The attack begins with the CNF C(X,K,Y) representing the obfuscated circuit where X is the input, K is key and output Y. The attack works by iteratively finding the assignment to the CNF clause C(X,K_1, Y_1) $\wedge$ C(X,K_2, Y_2) $\wedge$ ($Y_1 \neq Y_2$) until it is not satisfiable. Input X_i is called a discriminating input pattern iff for a particular input (X_i) two different keys (K_1 and K_2) produces two different outputs (Y_1 and Y_2), of which either one is correct or both are incorrect. The correctness of an output for a particular input and key combinations is verified using the oracle (an activated chip procured from the market). The clause C(X_i,K_1, Y_i) $\wedge$ C(X_i,K_2, Y_i) is appended to the existing CNF clause and fed back to the SAT solver.
When the CNF becomes unsatisfiable, it signifies that all the wrong key classes has been eliminated. Now the SAT solver finds the solution which satisfies the current constraints of the CNF and finds the correct key.

2.3 Related Works on SAT Attack Resiliency

With the advent of SAT based attacks on logic locking, the focus has been shifted towards developing countermeasures to mitigate SAT attack. Techniques like SARLock [11], Anti-SAT [15], CamoPerturb [13] etc., have been proposed to mitigate SAT attacks. The main crux of these attack, is to increase the number of iterations required by the SAT attack in extracting the correct key so that it becomes exponential. In SARLock each distinguishing

input pattern (DIP) can eliminate a single wrong key. This is achieved by inverting the output for a single input pattern corresponding to each wrong key. It consists of a couple of blocks namely a *mask block* and a *comparator block* and a *XOR gate*. The protection circuitry is integrated into the original circuit to make it SAT resilient as the number of iteration required becomes exponential. Similarly, Anti-SAT consists of two complimentary blocks connected to an AND gate. The blocks share the same input but have different key values. When the correct key is applied, the blocks produce complementary outputs for all inputs producing 0 as the AND gate output else it produces 1 for a particular input pattern, resulting in incorrect output. Stripped-functionality logic locking (SFLL), proposed in [18], in SFLL-HD^h the circuit to be protected is stripped of its original functionality for a certain number of inputs. Inputs having Hamming Distance h with the key are protected. SFLL-FLEX protects user specified input patterns. In this scheme, the protected input patterns act as secret keys which can be later used to get back the original functionality of the circuit. However, SFLL-HD^h is vulnerable to functional reverse engineering attack [8]. This attack can successfully identify the perturb and the restore unit of SFLL-HD^h and recover the original circuitry. Full-lock [3] are logic locking schemes that prevent SAT attack in a different manner. Instead of increasing the number of iterations, it increases the time required for each iteration of the SAT solver. Previously Yasin et al. proposed to insert AES cipher into the circuit [17] to be protected to complicate the circuit and make it SAT resistant.

However several above mentioned schemes are vulnerable to certain kinds of attacks like the removal attack [14]. As the original circuitry is kept untouched in SARLock after appending the protection circuitry, the attacker can peel-off the protection circuitry (XOR gate, mask and comparator block), and retrieve the original circuit. Anti-SAT is vulnerable to signal probability skew (SPS) attack [16]: Given a netlist locked by Anti-SAT, the AND gate can be easily identified by calculating the Absolute Difference of probability Skew (ADS) of the inputs for all gates. The AND gate of Anti-SAT block exhibits the highest ADS value as it has opposite skewed inputs. So the Anti-SAT block can be isolated from this AND gate by replacing the AND gate output to either 0 or 1 in the de-obfuscated netlist. To mitigate removal attack, the original circuit design in SARLock is slightly modified to produce TTLock [12]. This is done by flipping the output for a single input pattern (called the protected input pattern) with respect to the original circuit. So even if the XOR gate and the other block are isolated with the application of the removal attack, the attacker is left with the modified circuit. SFLL-HD^0 is functionally equivalent to TTLocks as it protects a single input pattern.

In addition SARLock and Anti-SAT are vulnerable to Bypass attack [10]. This attack works by finding the DIP that produces wrong output for the wrong key using SAT attack. Then a bypass circuit is added which flips the output for this DIP.

3 Ci-Lock: Cipher Based logic locking

3.1 Motivation

Various logic locking schemes that can mitigate SAT based attacks or more correctly stated reduce SAT attack to simple brute force attack, rely on increasing the number of SAT iterations. As already stated, the number of DIPs required to find the correct key is used as a metric for evaluating the logic locking technique against SAT attack.
In this way, SARLock is successful in increasing the number of DIPs required for finding the correct key to exponential order as each DIP eliminates only one possible wrong key combination. The problem with this scheme is that the error rate (output corruptibility) is exponentially low even though their key values are wrong there is a chance of getting the correct output for some input pattern.
Another approach of SAT attack mitigation is to increase the time required for each iteration of the SAT attack. This is achieved by forming SAT-hard instances. SAT hardness of formula is defined by the metric called variable per clause ratio. [4] investigates the SAT hardness of formulae and it concludes as follows - If a formula has this ratio less than 3, then it is said to be under-constrained. An under-constrained formula has many assignments and hence quickly solved by a SAT solver whereas if this ratio is above 6 for a formula, then the formula is said to be over-constrained. An over-constrained formula has fewer assignments and more contradictions, hence they are also quickly solved by SAT solvers.
But formulae with variable per clause ratio lying in the range 3 to 6 are found to be SAT-hard instances. SAT-hard instances have very few assignments and the SAT solver takes significant time for verifying such assignments and to solve such formulae. Logic locking using SAT-hard instances is proposed in [3], where SAT-hard instances are created using fully Programmable Logic and Routing blocks (PLR) that replace parts of the circuit to be obfuscated. These PLRs are non-blocking and SAT-hard instances by construction and they are constructed using a Logarithmic-based Network which is key-Configurable (CLN) and they are inserted into the circuit. The CLNs are constructed using MUXes.
As mentioned, CLN provides the required SAT hardness but to prevent other kinds of attacks (like removal attack), logic gates before the CLN are replaced using LUTs. However the CLNs suffers from few shortcomings, they can be expressed as an affine transformation function of input X (the affine form $y = AX + B$) and can be successfully de-obfuscated. Further we see that gates are replaced with LUTs (non volatile memory) which is not commonly used in standard ASIC design flow and are relatively slow.
Almost all logic locking schemes proposed till date are key-based. The security they provide is solely dependent on the key. But to secure real world data, we trust cryptographic constructs like block ciphers. Block ciphers are tested by cryptographers across the globe and are known for their rich

theory and strong mathematical background. They are by default secure against SAT based attacks, so we aim to utilize their properties and propose a logic locking scheme which is immune to known SAT based attacks.

In such a situation, recovering the key of the locked circuit for the purpose of de-obfuscation will be equivalent to the key extraction of a standard block cipher. The output corruption for the wrong key is also high for block ciphers. Even minor variations can create a significant number of output bits to change. So it is practically infeasible for the adversary to extract the key by sensitizing the key values to the output.

3.2 Can a Cipher Hide a Circuit?

Block ciphers $\mathcal{C}$ are pseudo-random permutation. It can be represented as $\mathcal{C}$: $\mathcal{P} \times \mathcal{K} \rightarrow \mathcal{E}$ where $\mathcal{K} = \{0,1\}^k$ is the key space $\mid \mathcal{K} \mid = k$, $\mathcal{P} = \{0,1\}^n$ is the plaintext and $\mathcal{E} = \{0,1\}^n$ is the encrypted text both of size n. It is fairly difficult to tune $\mathcal{C}$ to realize the desired mapping as required. The input-output mappings realized by $\mathcal{C}$ is unique based on a $\mathcal{K}$ and given a significant number of input-output pairs, it is not possible to guess the key that realizes the desired mapping. In other words, there is no correlation between $\mathcal{P}$, $\mathcal{E}$ and $\mathcal{K}$. $\mathcal{C}$ is secure against various attacks mentioned previously and also known to have high output corruptibility. We need to explore the possibility of hiding a circuit in $\mathcal{C}$ to utilize its security firmness.

A combinational circuit is a circuit, where the output is a pure function of the present input only. It produces specified output for each input pattern. The mapping realized by a combinational circuit can be represented as $\mathcal{F} : I \longrightarrow O$, where $I = \{0,1\}^x$ and $O = \{0,1\}^y$ with x inputs and y outputs. But the question is how can we realize $\mathcal{F}$ using $\mathcal{C}$ partially. A simple straight forward approach to do the same would be to have a unique key for every I in $\mathcal{F}$. But this is impractical as in this case the key would vary with each input pattern and becomes dependent on I. For $|I| = x$, we shall have $|\text{key}| = 2^x$. But for the purpose of logic locking, we need to have a unique key $m \in \mathcal{K}$ so that $\mathcal{C}$ can realize $\mathcal{F}$. For the correct key (m_c) we have $\mathcal{C}_{m_c} \equiv \mathcal{F}$ and $\forall m_i \neq m_c$ we have $\mathcal{C}_{m_i}$ not equivalent to $\mathcal{F}$.

A circuit consists of logic gates and wires as its fundamental components. It does not involve any memory elements. If we can create these fundamental components using the existing components of $\mathcal{C}$ then we can embed a circuit into $\mathcal{C}$. We explored this idea and found a novel approach of constructing gates and switch boxes using the Sbox of $\mathcal{C}$ (namely PRESENT [2] and GIFT [1]). The methodology for using the Sbox of PRESENT cipher as various logic gates and routing switch boxes is elaborated in detail in the following subsections.

3.3 Sbox as Logic Gates

Sboxes are the basic component of $\mathcal{C}$ which are non-linear and performs substitution. It takes n bit input and transforms them into m bit output (where n is not necessarily equal to m). Generally, Sboxes are implemented using look-up table.
Our main idea is to use the Sboxes as various logic gates by fixing some of the input bits of the Sboxes. Let us consider the PRESENT cipher. PRESENT cipher Sboxes are 4×4 mappings. For example, let us see how a Sbox can be configured as logic NAND. Fig. 2 shows a PRESENT cipher Sbox (4×4) configured as a NAND gate. In order to make a Sbox behave as a NAND gate, we need to set the last two input bits of the Sbox (2^{nd} and 3^{rd} bit) as logic 1 and connect the inputs for NAND gate (A and B) to the first two input bits of the Sbox (0^{th} and 1^{st} bit). With the above configuration, we obtain the NAND output of signal A and B at the 1^{st} output bit of the Sbox (0^{th} bit).

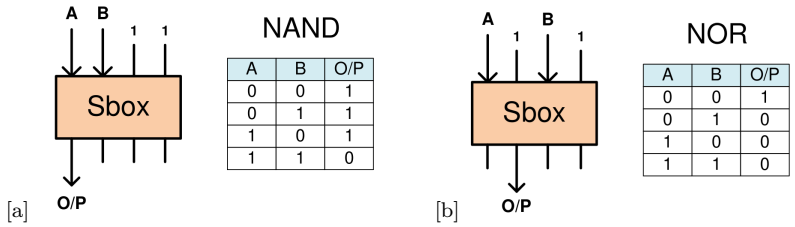


Fig. 2 Gates using Sbox (a) Nand gate (b) Nor gate, where A and B are gate inputs and O/P is the gate output.

Similarly logic NOR can be constructed from Sbox by setting the 1^{st} and 3^{rd} input bit (0^{th} and 2^{nd} bit) of the Sbox as logic 1 and connecting the inputs for NOR gate (A and B) to 2^{nd} and 4^{th} bit (1^{st} and 3^{rd} bit) of the Sbox input and the output of NOR gate is obtained at the 2^{nd} output bit.
In this way, other 2-input logic gates like AND, OR, XOR can be constructed using the PRESENT Sbox by setting two of the four input bits with 0 or 1 (as required) and connecting the inputs for the gate to the remaining two Sbox inputs. The output is obtained at one of the four output bits of the Sbox (this output bit remains fixed for a particular gate formed by that Sbox).
One input inverter gate can also be constructed using the above-mentioned method with a difference that the for a Sbox to behave as inverter gate, we need to fix three out of the four input bits with logic 0 or 1 (as required) and the inverted output of the input signal is obtained at one of the four output bit of the Sbox (which is fixed).
We know that NAND and NOR are universal logic gates. All other logic gates can be constructed using these two gates individually. Since a PRESENT cipher Sbox can be designed to function as NAND and NOR gates, proves

Gate	Sbox input bits				Sbox output bits			
	0	1	2	3	0	1	2	3
XOR	A	0	B	1			O/P	
	A	1	B	1		O/P		O/P
	1	A	B	0		O/P		
	A	B	0	0				O/P
	A	B	1	0	O/P			
AND	0	1	A	B		O/P		
	A	1	0	B			O/P	
OR	1	0	A	B	O/P			
	A	0	1	B	O/P			
NOR	1	A	1	B		O/P		
	A	1	0	B	O/P			
NAND	A	B	1	1	O/P			
	0	0	A	B		O/P		
NOT	0	A	1	0		O/P		
	A	0	0	0	O/P			

Table 1 Present Cipher Sbox configurations for various logic gates. 'A' and 'B' are the gate inputs and O/P is the gate output.

the universal nature of the Sbox as well. A Sbox or a cascaded set of Sboxes can be made to function as any required logic gates.
We have also observed that various logic gates (NAND, NOR, OR, AND, XOR, NOT) can be constructed using (4×4) GIFT cipher Sbox. Table 1 summarizes the configuration of various Sbox bits for the purpose of constructing different gates from PRESENT cipher Sbox. 'A' and 'B' are the input signals to the gate and 'O/P' is the gate output. We obtain the output at a specific Sbox output bit (one out of four) and the remaining three bits signals are redundant. There are many possibilities of single input not gate (only two mentioned in the table) that can be constructed using this method.

3.4 Routing the Signals

We have seen how the PRESENT cipher Sbox can be configured and used as various logic gates. We aim to embed a circuit into the PRESENT cipher without altering its original structure. PRESENT cipher is a Substitution-Permutation network having a linear permutation layer. Every layer of this cipher has 16 Sboxes. The output from these Sboxes are connected to the input of 16 Sboxes in next layer based on the fixed permutation network. An actual circuit can have multiple fan-outs and situations like output of gates feeding to other gates which are separated by multiple topological levels. Further, we see in the above section that to use a Sbox as gate we need to apply input signal to specific input bits of the Sboxes. All these situations are needed to be handled and for this purpose, we propose to use Sbox as switches for routing a signal from one PRESENT layer to another and also to act as dummy Sbox to supply multiple fan-out as required in the circuit.

These Sboxes behaving as dummies are required to keep the permutation layer and hence the structure of the cipher intact.
The Fig. 3 shows a few cases as how a Sbox can be configured as a dummy to simply pass signals from one layer to another and to handle multiple fan-out. The first and the second Sbox in Fig. 3 shows how the signal 'A' is distributed among three out of the four output Sbox bits whereas the third Sbox shows that the signal is simply passed through the Sbox and the fouth Sbox shows the signal getting distributed among two output bits. Sboxes, where the signal gets distributed into two or three bits can be used to handle fan-outs in the circuit and Sboxes where no such distribution occurs and the signal is simply passed to one of its output bit can be used as a routing wire.

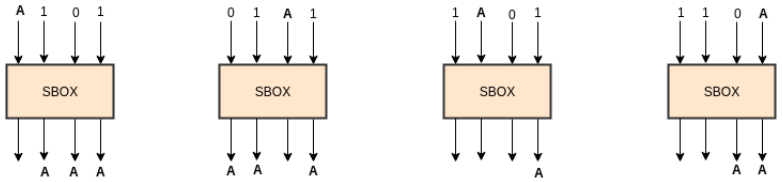


Fig. 3 Sbox configured as switch boxes for routing and handling multiple fan-out

,

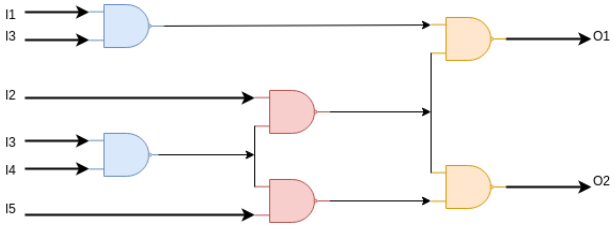


Fig. 4 C17 circuit of ISCAS-85 benchmark.

3.5 Embedding a Circuit into the PRESENT Cipher

So far we have seen how a Sbox can be configured into logic gates and routing wires by setting the input bits as required. Now let us see how a simple circuit can be embedded into $\mathcal{C}$ without disturbing the structure of $\mathcal{C}$. As an example, we consider C17 circuit of ISCAS-85 benchmark. It is a 5-input 2-output circuit consisting of 6 two-input NAND gates (ref. Fig. 4). Each input and output is of one bit.
Fig. 5 shows C17 circuit embedded into the PRESENT cipher. In this case, we need 7 layers of PRESENT to embed the entire C17 circuit.The initial

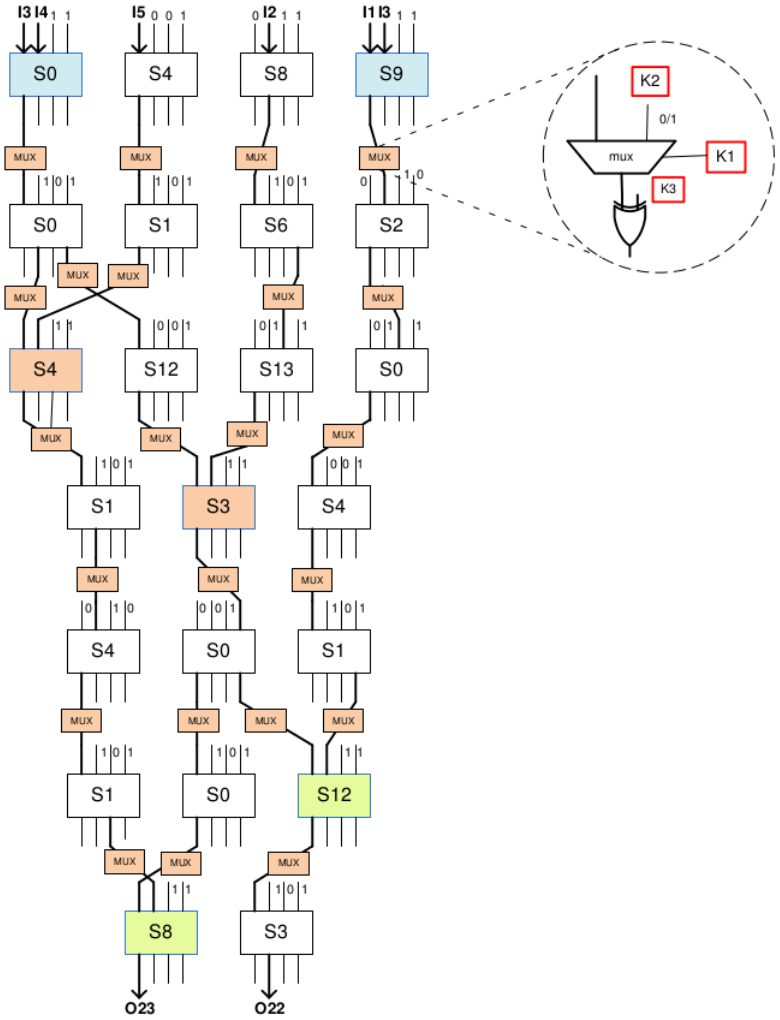


Fig. 5 C17 circuit embedded in Present cipher. The blue sboxes function as nand gate and the remaining sboxes are simply used for routing purpose. MUXes are placed for each bit for the purpose of setting the bits or passing the signal from the above layer (as required).

layer has two NAND gates (namely Sbox0 and Sbox9) and the second layer is a routing layer. The C17 circuit has a couple of multiple fan-outs and they are handled in the second layer by Sbox0 and in the fifth layer by Sbox0. The figure also demonstrates how a signal is passed from one topological

layer to another as required by the circuit. Finally the output is obtained at the last layer from the Sboxes.

The Sboxes in order to perform the desired function (gate or routing purpose) for embedding a circuit needs to be configured correctly. It is evident that the functionality of a Sbox depends on input bits and their corresponding set value (0 or 1). We see that in every layer of the figure, the Sbox inputs are fixed but this is not possible in a cipher directly. The inputs to various bits of an Sbox in a layer is based on the Sbox output in the layer above that is permuted to its input. But we need to fix the input bits of the intermediate Sbox layer in order to achieve the desired circuit mapping. We achieve this by inserting MUXes. PRESENT is a 64 bit block cipher and in each layer we place a MUX for each bit and pass the signal from the layer above through this MUX. These MUXes are key controlled. As a result of which we can either allow the signal from the above level to pass or set a desired input bit (0/1).

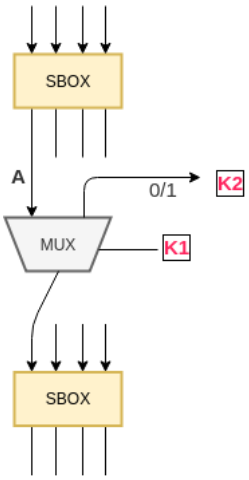


Fig. 6 MUX inserted for each bit in between each layer to set the input bits (if required).

For example in Fig. 6, K1 is the select line for the MUX and K2 is the value that is to be set into the next layer Sbox input. K1 can assume a value 0 or 1 based on which either 'A' is passed or K2 is passed through the MUX. Similarly K2 can also assume a value 0 or 1, which is to be set in the next layer. Both K1 and K2 can act as keys. Now if the Sbox in the upper layer is a gate Sbox then we need to pass signal 'A' to the next layer in that case we need to set K1 as 0 and K2 is in don't care condition. But if we need to set logic 0 to the input bit of the Sbox in the lower layer then we set K1 as 1 and K2 as 0. In this way we have control of each and every input bit of the Sboxes in the subsequent layer. The keys K1 and K2 are of length 64 bits each as there are 64 MUXes in each layer (one per bit) this adds complexity

to our proposed technique. In this way we can easily set a Sbox input bit or pass on a signal from one layer to any subsequent layer below.

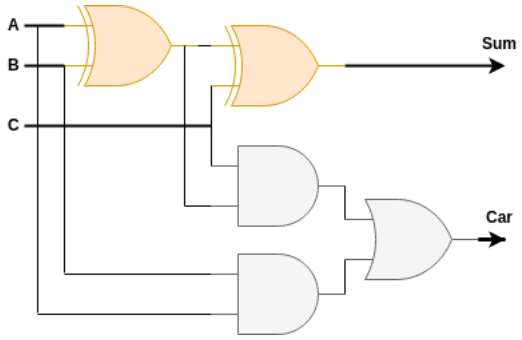


Fig. 7 A full adder circuit. 'A', 'B', 'C' are the inputs. 'Sum' and 'Car' are the sum and carry output of the circuit respectively.

Likewise, we have successfully embedded the full adder circuit by using this proposed technique. A full adder is a simple circuit which performs addition. It adds three one-bit binary numbers and outputs - one sum bit another carry bit. It is widely used in ALU and DSP blocks. Fig. 7 shows the gate level representation of the full adder. It took five layers of the PRESENT cipher to embed the full adder circuit using the proposed methodology. The obfuscated structure takes input 64 bits out of which 6 bits are used as the input bits of the full adder and the rest of the bits are randomly chosen. The output of the 5 layer PRESENT cipher is also 64 bits of which only 2 specific bits gives the desired output of the adder circuit - the sum and the carry.

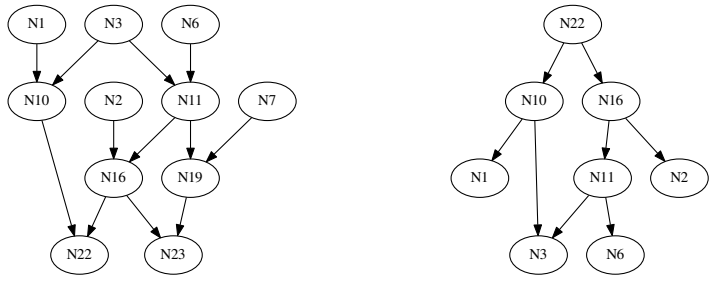
Algorithm 1 Cone nodes extraction algorithm

```
Input: Circuit Netlist(C_N)
Output:Cone with Node of maximum assigned colours (N)
G = parser(C_N) // DAG representation of C_N
Initialization: Output Nodes in G(N_S) = [1 ··· N]
Assign: unique colours to all elements in N_S
Define: Queue Q, Cone_nodes[], ColorsDict[N] // N ← # of nodes in G
for each i in N_S do
    Enqueue(Q, i)
    while (!Empty(Q)) do
        tmp ← Dequeue(Q)
        for each j in parents[tmp] do
            ColorsDict[j] ← colour[i]
            Enqueue(Q, j)
        end
    end
end
N = findMaxColor(ColorsDict[N]) // Returns the node with maximum colour
if N ≥ 2 then
    for each i in N do
        if fan_in(i) ≤ 3 then
            Cone_nodes ← Cone_nodes ∪ BFS(i) upto 3 levels upward
            return Cone_nodes
            break
        end
    end
end
```

3.6 Automating the Embedding

So far we have seen how a circuit can be successfully embedded into the PRESENT cipher. But can all circuits be embedded into the cipher? We see that the width of the cipher or the number of Sboxes in each layer is fixed for $\mathcal{C}$. So there is definitely a constraint on the type of circuit that can be obfuscated using this proposed technique. For simplicity we assume that we will extract a part of a circuit (a cone of reasonable depth from the circuit), we shall embed this cone into $\mathcal{C}$ using the above methodology and hence this part of the circuit remains obfuscated.
Initially we take a circuit netlist and represent it graphically. Fig. 8(a) shows the DAG representation of the circuit C17. This helps us to apply various graph algorithms in the circuit required for processing. The DAG representation ensures that the nodes in the circuit in already levelized. In order to extract a cone from the circuit graph, we run BFS from desired source node. BFS helps to extract a cone as shown in Fig. 8(b). The parent nodes in this cone which have 2 children are the NAND gates and the nodes (especially at the bottom level) which does not have any children are the input signal and they does not have any logic gate associated with them.

Nodes with highest influence ($\mathcal{N}$) : Since we propose to obfuscate a part of a circuit, we need to find the most influential part of the circuit and ob-



[a] [b]

Fig. 8 (a) Graphical representation of C17 circuit (b) Cone extracted from the C17 circuit graph using BFS with N22 as the source node.

fuscate it. $\mathcal{N}$ can be defined as a set of nodes which causes the maximum output corruptibility in the circuit to be obfuscated. Thus $\forall n_i \in \mathcal{N}$, output corruptibility of n_i is maximum. This asserts that the input signals to almost all the final output nodes in the circuit passes through n_i. $\mathcal{N}$ is extracted using Algo. 1. The algorithm uses a simple graph colouring approach to extract the nodes from the graphical representation of the netlist. These nodes, when obfuscated using the proposed approach will achieve the maximum possible output corruptibility. According to Algo. 1, we associate a unique colour to every output node. Now we find out the dependent nodes for each output node and assign the colour (corresponding to that output node) to all the nodes on which that output node depends. This process continues for all the output nodes and when all the nodes in the circuit graph has been assigned one or more colours based on its influence on the number of output nodes, we obtain the nodes with maximum colour assignment ($\mathcal{N}$). In this way we can find nodes that has the highest output corruptibility and hence cones containing one or more such nodes (provided it is a 2 or 3 input gate) are obfuscated to secure the circuit.
Now we have to embed the extracted cone into $\mathcal{C}$. We propose to do the same in a bottom-up manner. For example, for embedding the cone in Fig. 8(b). We will initially choose a Sbox randomly and make it as node N22 (the root of the cone). We know that N22 is a NAND gate so we need to supply the signals from N10 and N16 to the first two input bits of the Sbox chosen as N22 (since a Sbox forms a NAND gate when the first two bits are input and the other two bits are set as logic 1). It is evident that N10 and N16 are also NAND gates and they are in the same topological level. So we have to route the gate Sbox outputs of N10 and N16 to the input of gate Sbox N22. In order to do this, we often come across a problem that a Sbox output bit does not reach a desired Sbox input bit due to the presence of the permutation layer of $\mathcal{C}$. As we shall not alter the existing permutation layer, we deal with this issue by placing routing Sboxes in subsequent layer so that the required signal is ultimately routed to the desired bit position.

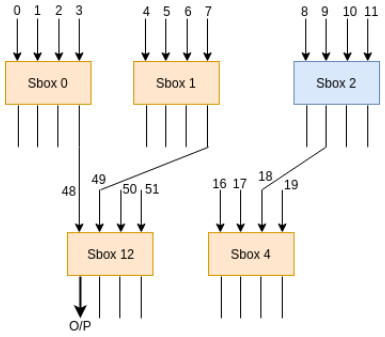


Fig. 9 If a particular Sbox is not available in a layer then we add a routing Sbox (Sbox 2)

The routing Sboxes also deals with the issue of non-availability of a particular Sbox in a layer. For example in Fig. 9 if a situation arises where both Sbox 12 and Sbox 4 are needed to be supplied at the first two of its input bits. It is not possible to simultaneously supply signals to first two bits of both the Sboxes as every layer has a single copy of every Sbox. Sbox 0 and Sbox 1 is already used to supply Sbox 12 so in such a situation, we place a routing Sbox (Sbox 2) to pass the signal to Sbox 4 and form the required gate in subsequent layers of PRESENT.
In this way by placing routing Sboxes based on availability in the layer above we embed the part of the circuit into $\mathcal{C}$.

4 Experimental Results

In this section we shall see the performance of our proposed obfuscation technique against SAT attack. We have implemented our scheme on various benchmark circuits of ISCAS-85 and also we have encrypted a simple full adder circuit using this scheme. The experiments are conducted on Intel Xeon processors running at 2.4GHz with 256 GB of RAM. The circuits were analyzed and graphically represented using python3 libraries. Netlists were generated using the DC compiler. The experimental results shows that all our obfuscated designs were SAT resistant. Table 2 summarizes the results of SAT attack on various circuits.

5 Conclusion

In this paper we have approached the problem of logic obfuscation in a entirely new dimension. So far logic encryption was achieved and various SAT based attacks were mitigated by inserting key gates, circuit distorting logic blocks or by sat hard constructions, into the circuit. But this paper highlights how block ciphers can be used to achieve the purpose of logic obfusca-

Circuits	# Gates	#I/Os	Cone Details	SAT-Resilient
Full Adder	5	3/2	Entire Circuit	✓
c17	6	5/2	Entire Circuit	✓
c432	160	36/7	4 nodes	✓
c499	202	41/32	7 nodes	✓
c880	383	60/26	5 nodes	✓
c1908	880	33/25	7 nodes	✓
c2670	1269	233/140	7 nodes	✓
c3540	1669	50/22	6 nodes	✓
c5315	2307	178/123	8 nodes	✓
c6288	2416	32/32	7 nodes	✓
c7552	3513	207/108	4 nodes	✓

Table 2 SAT Resilience on ISCAS-85 benchmark circuits

tion by embedding a part or the whole circuit into it. We have also demonstrated that our proposed scheme throttles SAT attack on various obfuscated circuits. Block ciphers are known to be resistant against SAT attacks so obfuscating circuits using them prevents SAT and other known forms of attacks against the logic locked circuits.

References

1. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: Gift: a small present. In: International Conference on Cryptographic Hardware and Embedded Systems, pp. 321–345. Springer (2017)
2. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: Present: An ultra-lightweight block cipher. In: International workshop on cryptographic hardware and embedded systems, pp. 450–466. Springer (2007)
3. Kamali, H.M., Azar, K.Z., Homayoun, H., Sasan, A.: Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks. In: Proceedings of the 56th Annual Design Automation Conference 2019, p. 89. ACM (2019)
4. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of sat problems. In: AAAI, vol. 92, pp. 459–465 (1992)
5. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Security analysis of logic obfuscation. In: Proceedings of the 49th Annual Design Automation Conference, pp. 83–89. ACM (2012)
6. Rajendran, J., Sam, M., Sinanoglu, O., Karri, R.: Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 709–720. ACM (2013)
7. Roy, J.A., Koushanfar, F., Markov, I.L.: Epic: Ending piracy of integrated circuits. In: Proceedings of the conference on Design, automation and test in Europe, pp. 1069–1074. ACM (2008)
8. Sirone, D., Subramanyan, P.: Functional analysis attacks on logic locking (2018)
9. Subramanyan, P., Ray, S., Malik, S.: Evaluating the security of logic encryption algorithms. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 137–143. IEEE (2015)
10. Xu, X., Shakya, B., Tehranipoor, M.M., Forte, D.: Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks. In: International Conference on Cryptographic Hardware and Embedded Systems, pp. 189–210. Springer (2017)

11. Yasin, M., Mazumdar, B., Rajendran, J., Sinanoglu, O.: Sarlock: Sat attack resistant logic locking. pp. 236–241 (2016). DOI 10.1109/HST.2016.7495588
12. Yasin, M., Mazumdar, B., Rajendran, J.J., Sinanoglu, O.: Ttlock: Tenacious and traceless logic locking. In: 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 166–166. IEEE (2017)
13. Yasin, M., Mazumdar, B., Sinanoglu, O., Rajendran, J.: Camoperturb: Secure ic camouflaging for minterm protection. In: 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2016). DOI 10.1145/2966986.2967012
14. Yasin, M., Mazumdar, B., Sinanoglu, O., Rajendran, J.: Removal attacks on logic locking and camouflaging techniques. IEEE Transactions on Emerging Topics in Computing (2017)
15. Yasin, M., Mazumdar, B., Sinanoglu, O., Rajendran, J.: Security analysis of anti-sat. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 342–347 (2017). DOI 10.1109/ASPDAC.2017.7858346
16. Yasin, M., Mazumdar, B., Sinanoglu, O., Rajendran, J.: Security analysis of anti-sat. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 342–347. IEEE (2017)
17. Yasin, M., Rajendran, J.J., Sinanoglu, O., Karri, R.: On improving the security of logic locking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**(9), 1411–1424 (2015)
18. Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J.J., Sinanoglu, O.: Provably-secure logic locking: From theory to practice. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, pp. 1601–1618. ACM, New York, NY, USA (2017). DOI 10.1145/3133956.3133985. URL http://doi.acm.org/10.1145/3133956.3133985