

Fa-SAT: Fault-aided SAT-based Attack on Compound Logic Locking Techniques

Nimisha Limaye[§], Satwik Patnaik[†], and Ozgur Sinanoglu[‡]

[§]Tandon School of Engineering, New York University, USA

[†]Electrical & Computer Engineering, Texas A&M University, College Station, Texas, USA

[‡]Division of Engineering, New York University Abu Dhabi, UAE
{nsl278, ozgursin}@nyu.edu, satwik.patnaik@tamu.edu

Abstract—Logic locking has received significant traction as a one-stop solution to thwart attacks at an untrusted foundry, test facility, and end-user. Compound locking schemes were proposed that integrate a low corruption and a high corruption locking technique to circumvent both tailored SAT-based and structural-analysis-based attacks. In this paper, we propose *Fa-SAT*, a generic attack framework that builds on the existing, open-source SAT tool to attack compound locking techniques. We consider the recently proposed bilateral logic encryption (BLE [1]) and Anti-SAT [2] coupled with random logic locking [3] as case studies to showcase the efficacy of our proposed approach. Since the SAT-based attack alone cannot break these defenses, we integrate a fault-injection-based process into the SAT attack framework to successfully expose the logic added for locking and obfuscation. Our attack can circumvent these schemes' security guarantees with a 100% success across multiple trials of designs from diverse benchmark suites (ISCAS-85, MCNC, and ITC-99) synthesized with industry-standard tools for different key-sizes. Finally, we make our attack framework (as a web-interface) and associated benchmarks available to the research community.

Index Terms—Logic locking, Bilateral logic encryption, Anti-SAT, Fault-injection, SAT-based attacks, Compound locking.

I. INTRODUCTION

Logic locking is a one-stop solution to defend against *untrustworthy* entities in the supply chain from pirating and over-producing integrated circuits. Researchers directed their initial efforts towards finding appropriate insertion points for *key-gates* and ensuring high output corruption where the application of an incorrect key by an adversary can lead to maximal corruption at outputs [4]. However, these initial efforts were short-lived when a Boolean satisfiability (SAT)-based attack [5] was shown to break these locking techniques successfully. Consequently, researchers began to develop techniques to thwart the SAT-based attack using point-function-based schemes like *Anti-SAT* [2] and variants of *Stripped Functionality Logic Locking (SFL)* [6].¹ Although point-function-based techniques proved resilience against the SAT-based attack [5] by pushing it to its brute-force equivalent, such schemes were vulnerable to either tailored SAT-based attacks [7], [8] or structural-analysis-based attacks [9], [10].² Recently, researchers have explored using SAT-hard instances to thwart the SAT-based attack [11],

¹Point function such as AND tree outputs a '1' for only one input pattern; the output is '0', otherwise.

²The vulnerability to tailored attacks like *AppSAT* [7] and *Double-DIP* [8] was owing to the property of low output corruption, which is a manifestation of the usage of point functions. Regarding structural-analysis-based attacks [9], [10], the dissolution of the inserted AND-tree is relegated to security-oblivious synthesis tools, and the structural traces left behind help an opportune attacker.

TABLE I
ATTACKS VERSUS DEFENSES ON LOCKING TECHNIQUES. ✓ DENOTES ATTACK SUCCESS AGAINST THE DEFENSE. AGR: APPSAT GUIDED REMOVAL ATTACK. SPS: SIGNAL PROBABILITY SKEW ATTACK

Attack \ Defense	RLL	Anti-SAT	Anti-SAT+RLL	BLE
SAT-based attack [5]	✓	✗	✗	✗
AppSAT attack [7]	✓	✗	✗	✗
AGR attack [9]	✓	✓	✓	✗
SPS attack [9]	✗	✓	✗	✗
Fa-SAT (This work)	-	✓	✓	✓

AppSAT [7] recovers an approximate key, therefore we consider it unsuccessful in circumventing the defense.

[12].³ However, recently these techniques were shown to be vulnerable to advanced neural-network-based attack [13]. Other approaches included scan locking [14], [15], but they were prone to shift-and-leak and modeling-based attacks [16], [17].

To circumvent the drawbacks of all these standalone techniques, *compound locking* was adopted, where a low corruption locking technique (which thwarts SAT-based attack) is integrated with a high corruption locking technique (which prevents structural-analysis-based attacks). *Anti-SAT* [2] combined with random logic locking (RLL) [3] is an example of a compound locking scheme. Recently, bilateral logic encryption (BLE) [1] was proposed, where a low-corruption locking technique and a routing-based obfuscation scheme were integrated to thwart variants of exact (and approximate) SAT-based attacks and the structural-analysis-based attacks. The BLE technique thus stands unbroken; we explain BLE in more detail in Sec. II. The **primary contributions** of our work are enumerated as follows:

- 1) We propose a fault-injection-based attack (Fa-SAT) to assist the seminal SAT-based attack in tackling the otherwise resilient bilateral logic encryption (BLE) technique [1]. To that end, we make our attack framework (as a web-interface) and the associated benchmarks available.⁴
- 2) Fa-SAT attack is generic and we examine its generality, scalability, and efficacy by applying it to another technique comprising of Anti-SAT [2] and RLL [3].
- 3) We showcase the strength (100% key recovery) of our attack by conducting extensive experiments across multiple trials of designs from diverse benchmark suites

³SAT-hard structures aid in increasing the time significantly for each iteration of the SAT-based attack [12].

⁴<https://github.com/DfX-NYUAD/Fa-SAT>

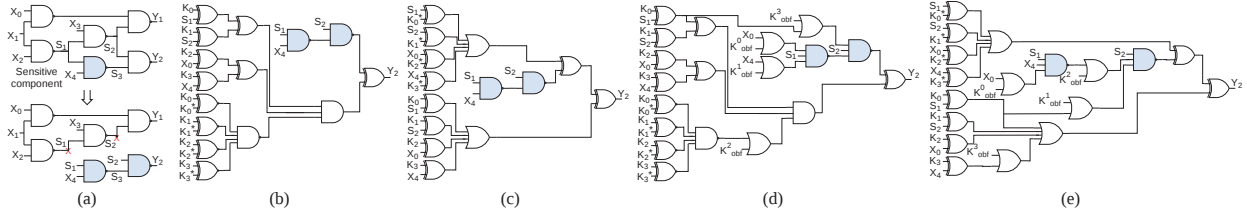


Fig. 1. Bilateral logic encryption [1]: Illustration of the stages viz. extraction, locking, and obfuscation of the ISCAS-85 benchmark c17.

(ISCAS-85, MCNC, and ITC-99) with different key-sizes synthesized with commercial, industry-grade tools.

Table I summarizes the attacks on various locking techniques and illustrates that Fa-SAT circumvents the security promises of the BLE technique [1] along with the Anti-SAT variants [2].

II. FUNDAMENTALS OF BILATERAL LOGIC ENCRYPTION

Bilateral logic encryption (BLE) [1] scheme incorporates logic locking to thwart the seminal [5] and tailored, advanced, approximate SAT-based attacks [7], [8]. It further incorporates obfuscation to thwart structural-analysis-based attacks by strategically obfuscating the locked circuit, rather than relying on security-agnostic synthesis tools to dissolve structural traces, as adopted in prior works [2], [6]. Primarily, BLE constitutes of four stages, viz., (i) extraction of sensitive component and deriving of to-be-protected sub-circuit, (ii) locking of extracted sub-circuit, (iii) obfuscation of the locked sub-circuit, and (iv) concatenation of the obfuscated locked sub-circuit with the remaining design; we explain the steps briefly next.

A. Steps Involved in Bilateral Logic Encryption

The first stage involves the extraction of a security-critical component.⁵ However, protecting only the security-critical component might not achieve the desired output corruption.⁶ Therefore, the authors advocate adding gates in the fan-out of this security-critical component (in a greedy manner) till reaching a primary output; this becomes the protected output port. Figure 1(a) shows the selection of the security-critical component (shown in light blue) and the subsequent formation of the to-be-protected sub-circuit (gates are shown in light blue) disconnected by crosses shown in red.

Next, this extracted sub-circuit is locked, either using standard locking (SL) (Fig. 1(b)) or stripped functionality circuit (SFC) locking (Fig. 1(c)). The authors use a n -bit key to lock the sub-circuit where n is approximately equal to the number of primary inputs (l) in the original design. The SL technique inserts a well-defined structure governed by Eq. 1 around the sub-circuit; $h(x, k)$ is the sub-circuit having x inputs and k key-bits. Note that k^* is the secret key hard-coded in the design and applying k equal to k^* unlocks the design. SL ensures an output error rate (OER) of $1/2^{(n/2)}$; for an incorrect key, the circuit fails for $2^{(n/2)}$ patterns. On the other hand, the SFC technique inserts a well-defined structure governed by Eq. 2 around the

sub-circuit; $h(x, k)$ is the sub-circuit having x inputs and k key-bits with secret key k^* hard-coded in the design. Further, applying k equal to k^* unlocks the design. Unlike SL, it has an even lower OER of $1/2^n$.

$$h(x, k) = \bigwedge_{i \in [0, n/2-1]} (x_{2i} \oplus k_{2i}) \oplus (x_{2i+1} \oplus k_{2i+1}) \quad (1)$$

$$k \neq k^* = \bigwedge_{i \in [0, n-1]} k_i \oplus k_i^*$$

$$h(x, k) = \bigvee_{i \in [0, n-1]} x_i \oplus k_i; \quad h(x, k^*) = \bigvee_{i \in [0, n-1]} x_i \oplus k_i^* \quad (2)$$

To evade the demerits of the locking technique, authors incorporate obfuscation of the locked sub-circuit, as shown in Fig. 1(d)-(e); obfuscation key-bits are denoted by K_{obf} . Signal routing is utilized to perform the task of strategic obfuscation. This obfuscation scheme aims to provide resilience against structural-analysis-based attacks *without requiring re-synthesis* [1]. Strategic obfuscation inherently introduces security without relying on security-agnostic synthesis tools to dissolve the structural traces of security-critical components. It leverages the concept of real nets and dummy nets to obfuscate the design. Routing obfuscation adopted in BLE is explained in detail next.

First, NAND/AND and NOR/OR gates in the design are chosen, and dummy nets are randomly inserted at the input of these gates. The obfuscation scheme inserts an AND gate controlled by an obfuscation key-bit at the input of NOR/OR gates present in the design. If the AND gate was inserted at an existing (real) net, then the obfuscation key-bit is '1', thus acting as a buffer. If the AND gate is inserted at a dummy net, then the obfuscation key-bit is '0', blocking the effect of the dummy net on the design's functionality. Similarly, when an OR gate is added at the input of an existing NAND/AND gate, for the real signal, the obfuscation key-bit will be '0' whereas the obfuscation key-bit will be '1' for the dummy signal. Equation 3 highlights all four transformations adopted in strategic obfuscation; p is the candidate net which can be either a real or a dummy net, q is a real net, and K_{obf}^i is the i^{th} obfuscation key-bit.

$$\begin{aligned} p \wedge q &\rightarrow ((p \vee K_{obf}^i) \wedge q) & p \overline{\wedge} q &\rightarrow ((p \vee K_{obf}^i) \overline{\wedge} q) \\ p \vee q &\rightarrow ((p \wedge K_{obf}^i) \vee q) & p \overline{\vee} q &\rightarrow ((p \wedge K_{obf}^i) \overline{\vee} q) \end{aligned} \quad (3)$$

The correct obfuscation key (K_{obf}) is "1001" and "1101" for designs shown in Fig. 1(d)-(e). In the final stage, the obfuscated-locked sub-circuit is concatenated with the rest of the original design.

⁵Note that, the authors in [1] have not specified how to choose the security-critical component; instead, they showcase their concept by choosing the sensitive component randomly, an approach which we also follow in our work.

⁶Locking a single component from the design may not have a transmissible effect on the primary output ports, thereby reducing the output corruption.

UP	O/P	k0	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10	k11	k12	k13	k14	k15	Pruned Key Values
0000	1	1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1	Iter 1: k5, k6, k9, k10
0001	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	1	1	Iter 4: k4, k7, k8, k11
0010	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	1	1	
0011	1	1	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1	
0100	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	
0101	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	
0110	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	Iter 3: k0, k3, k12, k15
0111	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	
1000	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	Iter 2: k1, k13, k14
1001	1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	
1010	1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	
1011	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	Iter 5: key found
1100	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	
1101	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	1	1	
1110	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	
1111	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	1	1	

(a)

UP	O/P	k0	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10	k11	k12	k13	k14	k15	Pruned Key Values
0000	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0001	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Iter 4: k1
0010	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	Iter 11: k2
0011	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	Iter 12: k3
0100	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Iter 8: k4
0101	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	Iter 13: key found
0110	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	Iter 9: k6
0111	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0
1000	1	1	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	Iter 1: k8
1001	1	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1	0	Iter 3: k9
1010	1	0	1	1	0	1	1	1	1	1	1	0	1	1	0	1	1	Iter 2: k10
1011	1	1	0	1	1	1	1	1	1	1	1	0	1	0	0	1	1	Iter 10: k11
1100	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	Iter 7: k12
1101	1	1	1	1	1	0	1	1	0	0	1	1	0	1	0	1	1	Iter 5: k13
1110	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	Iter 6: k14
1111	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	0	

(b)

Fig. 2. (a) Standard locking (SL) truth table (b) Stripped functionality circuit (SFC) locking truth table.

B. Security analysis

We examined the BLE scheme's components, viz., locking and obfuscation, and analyzed each of them for their security promises. As elucidated by authors and validated by our experiments, SL indeed takes $2^{(n/2)}$ iterations to recover the secret key (see Fig. 2(a)); each SAT iteration prunes out $2^{(n/2)}$ keys, where $n=4$. Further, for any incorrect key, the design will fail for $2^{(n/2)}$ patterns, thereby thwarting exact and approximate versions of SAT-based attack. SFC locking, as mentioned by authors and validated by our experiments (shown in Fig. 2(b)), prunes out exactly one key in each SAT iteration. The number of SAT iterations can reach $2^n - 1$ before retrieving the secret key. However, if the SAT solver picks the secret key as its next distinguishing pattern, then the attack terminates sooner with the secret key returned (shown in Fig. 2(b)), as also observed in other provably secure techniques. It is also vulnerable to approximate versions of the SAT-based attack [7].

Regarding the signal-routing based obfuscation scheme, we observed that the SAT solver recovers all the obfuscation key-bits *at once in the last iteration* before terminating, as opposed to recovering partial key-bits in each iteration [5]. Further, removing the obfuscated-locked component will result in an incomplete design (partial functionality), not useful to an attacker. *With such security promises, existing attacks fail to break the BLE scheme.*

III. FA-SAT: FAULT-AIDED SAT-BASED ATTACK

A. Adversarial model

We assume that the attacker is a rogue employee in an untrusted foundry and/or is an untrusted end-user, having access to the following resources to launch the Fa-SAT attack.

- 1) Functional chip or oracle (obtained from open market)
- 2) Reverse-engineered gate-level locked netlist (either from a foundry or by reverse-engineering a fabricated chip)

We assume that the attacker cannot distinguish the dummy nets from real nets [1]. Further, we assume that an attacker can perform any number of re-synthesis procedures or carry out structural analysis to decode the design functionality after

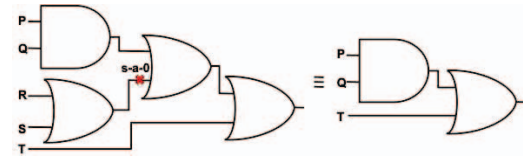


Fig. 3. (Left) Netlist with stuck-at-0 inserted. (Right) Reduced netlist as a result of fault-injection.

obtaining the reverse-engineered netlist. Next, we describe our proposed Fa-SAT attack in detail.

B. Fa-SAT attack

Our proposed Fa-SAT attack builds on the concept of fault-injection.⁷ We seek to identify the node in the netlist that will help the SAT solver return the secret key in a finite duration. We apply the fault-injection concept to obtain a sub-circuit, which, when fed to the SAT-based attack, will return the secret key in fewer iterations than the baseline SAT-based attack. Inserting a fault at the correct node aids this attack to return the correct key; however, inserting the fault at an incorrect node can result in false positives, i.e., the key returned by the SAT solver is not the secret key.

Attack methodology. In our (generic) attack framework, we start with the locked gate-level netlist, without performing any pre-processing (structural analysis) to identify the gates connected to key-inputs *or* identifying the structure corresponding to the locking scheme. We obtain a list of all fault locations in the locked netlist and store it in our candidate *faults* list. We insert a fault at these locations iteratively and feed it to the existing SAT-based attack framework. The SAT-based attack attempts to find a key for this fault-inserted netlist in a given timeout we set. We provide a timeout for each SAT run such that fault injection at the correct node reduces the circuit-size and hence its corresponding clauses and variables to return the secret key in fewer iterations. Accordingly, we provide a timeout of 10–90 seconds, depending on the size of the underlying design.

⁷Note that this principle is different from fault attacks where voltage glitches or lasers are used to induce targeted faults in the underlying design.

Algorithm 1: Fa-SAT algorithm.**Input:** locked netlist L , oracle O **Output:** secret key K , unlocked netlist U

$faults$ = faults in the design;
 FI = fault-inserted netlist;
 $f(x, y)$ = inserts fault y in netlist x ;
 $V(x, y, z)$ = functional verification with key x , locked netlist y , and oracle z ;

```

for  $i \leftarrow 1$  to  $faults$  do
   $FI \leftarrow f(L, i)$ 
   $cand\_key \leftarrow \text{timeout } 30s \text{ SAT\_solver}(FI, O)$ 
  if  $V(cand\_key, FI, O) == \text{equivalent}$  then
     $K \leftarrow cand\_key$ 
    return  $K, U(K, L)$ 
  
```

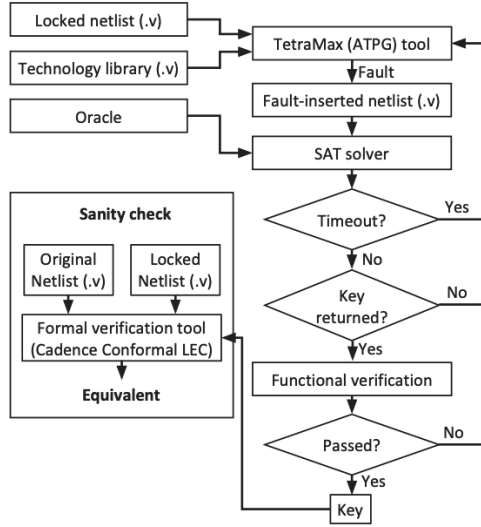


Fig. 4. Our Fa-SAT attack methodology.

Figure 3 showcases the effect of fault-injection in the netlist leading to a reduction in the circuit size. Further, Eq. 4 shows the Boolean formula of the original (fault-free) netlist and its corresponding conjunctive normal form (CNF). Inserting a fault at the output of the OR gate (as shown in Fig. 3(a)) reduces the CNF as illustrated in Eq. 5. Comparing Eqs. 4 and 5, we observe that the number of variables has reduced by two, but the number of clauses remains the same. Although the cumulative number of iterations increases with each fault's rejection, the variables and clauses reset to a minimum after each timeout, thereby not slowing down our attack.

$$[(P \wedge Q) \vee (R \vee S)] \vee T \rightarrow [(P \vee R \vee S \vee T) \wedge (Q \vee R \vee S \vee T)] \quad (4)$$

$$[(P \wedge Q) \vee (0)] \vee T \rightarrow [(P \vee T) \wedge (Q \vee T)] \quad (5)$$

After retrieving a key from the SAT solver, we functionally verify this key's correctness/fidelity using random input patterns. We apply these random input patterns on the unlocked netlist (with key returned by the SAT solver) and the oracle simultaneously. If the oracle output matches the unlocked

netlist's output for all these applied input patterns, we conclude to have successfully verified the correctness of the key; else, the attack selects the next *fault* from the list. To validate our attack success, we verify the recovered key using a formal verification tool (e.g., *Cadence Conformal LEC*). Algorithm 1 outlines the steps used to apply the Fa-SAT attack on a compound locking scheme. Figure 4 shows the flowchart, including the tools leveraged in the Fa-SAT attack.

Case study. We launch our attack on BLE, which inserts a point-function-based circuit (SL, SFC locking techniques) in the original design to make it differ for exactly one protected input pattern. Figure 5(a) shows a fault-inserted sub-circuit locked using SL and its corresponding revised circuit. Inserting a stuck-at-1 fault in this circuit aids the SAT solver to terminate with the correct key in just a single iteration. Equation 6 shows the Boolean equation of the SL-locked sub-circuit and its corresponding equation post-fault-insertion. Figure 5(b) shows fault-inserted sub-circuit locked using SFC and its corresponding revised circuit. Inserting a stuck-at-0 fault in this circuit aids the SAT solver in terminating with the correct key in just five iterations as opposed to 14 iterations taken by the seminal SAT-based attack [5]. Equation 7 shows the Boolean equation of the SFC-locked sub-circuit and its corresponding equation post-fault-insertion.

$$\begin{aligned}
 & [(S_1 \oplus k_0) \oplus (S_2 \oplus k_1) \oplus (X_0 \oplus k_2) \oplus (X_4 \oplus k_3)] \\
 & \wedge [(S_1 \oplus k_0^*) \oplus (S_2 \oplus k_1^*) \oplus (X_0 \oplus k_2^*) \oplus (X_4 \oplus k_3^*)] \\
 & \oplus [(S_1 \bar{\wedge} X_4) \bar{\wedge} S_2] \xrightarrow{\text{stuck-at-1}} \\
 & [(1) \wedge [(S_1 \oplus k_0^*) \oplus (S_2 \oplus k_1^*) \oplus (X_0 \oplus k_2^*) \oplus (X_4 \oplus k_3^*)] \\
 & \oplus [(S_1 \bar{\wedge} X_4) \bar{\wedge} S_2] \quad (6)
 \end{aligned}$$

$$\begin{aligned}
 & [(S_1 \oplus k_0^*) \vee (S_2 \oplus k_1^*) \vee (X_0 \oplus k_2^*) \vee (X_4 \oplus k_3^*)] \oplus [(S_1 \bar{\wedge} X_4) \bar{\wedge} S_2] \\
 & \oplus [(S_1 \oplus k_0) \vee (S_2 \oplus k_1) \vee (X_0 \oplus k_2) \vee (X_4 \oplus k_3)] \\
 & \xrightarrow{\text{stuck-at-0}} \\
 & [(S_1 \oplus k_0^*) \vee (S_2 \oplus k_1^*) \vee (X_0 \oplus k_2^*) \vee (0)] \oplus [(S_1 \bar{\wedge} X_4) \bar{\wedge} S_2] \\
 & \oplus [(S_1 \oplus k_0) \vee (S_2 \oplus k_1) \vee (X_0 \oplus k_2)] \quad (7)
 \end{aligned}$$

With the fault-injection approach tested and verified to aid the SAT-based attack, we move to experimental analysis next.

IV. EXPERIMENTAL INVESTIGATION

A. Setup

Defense Implementation. We implement BLE scheme [1] using *Synopsys Design Compiler* tool leveraging the *Nangate 45nm Open Cell Library* [18]. After locking the ISCAS-85 and MCNC benchmarks, we additionally verify the security claims by leveraging the SAT-based attacks [5], [7] to check the correctness of our implementation. Benchmark statistics, along with key-bits used for locking, are illustrated in Table II. We lock ITC-99 benchmarks with 64-bit key using standalone Anti-SAT and construct compound locking variant using 32 and 64-bit RLL combined with Anti-SAT technique. We verify the correctness of our implementation by launching SAT-based and *AppSAT* attacks using the open-source tools [5], [7]. All these tools are executed on a 128-core Intel Xeon processor running at 2.2 GHz with 256 GB of RAM.

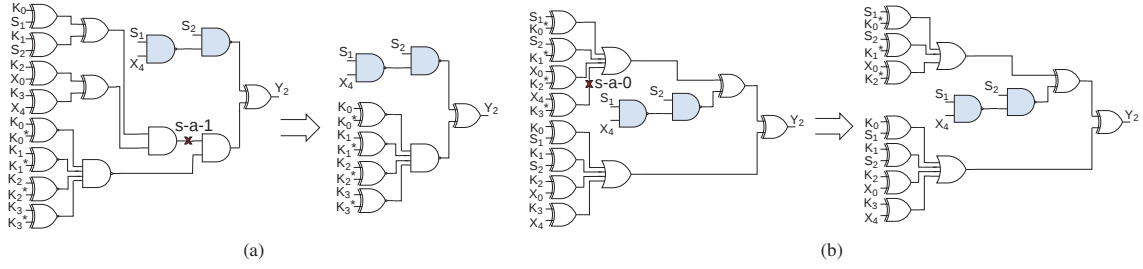


Fig. 5. Injecting a fault at the correct location (node) aids the SAT solver to terminate sooner, resulting in the extraction of the secret key. (a) For the SL-locked design, SAT iterations required to recover the secret key are reduced from 4 to 1. (b) For the SFC-locked design, SAT iterations required to recover the secret key are reduced from 14 to 5.

TABLE II
BENCHMARK STATISTICS FOR BLE

Benchmark	# Inputs	# Keys	# Gates
c1355	39	40 + 40 = 80	188
c1908	33	34 + 34 = 68	227
c5315	178	178 + 178 = 356	1,226
c880	60	60 + 60 = 120	275
apex2	39	40 + 40 = 80	263
dalu	75	76 + 76 = 152	624
k2	46	46 + 46 = 92	831

For # Keys: $x + y$ means x locking keys and y obfuscation keys.
Gates correspond to original design.

Fa-SAT Implementation. We implement our attack using customized *TCL* and *BASH* scripts. We utilize *Synopsys Tetra-max* to extract faults from the synthesized Verilog netlists. We launch our attack on 15 trials for four ISCAS-85, three MCNC benchmarks (total 210 circuits) locked with BLE schemes (SL and SFC). Additionally, we also test the efficacy of our attack on six ITC-99 benchmarks locked using standalone Anti-SAT and Anti-SAT combined with RLL. The randomness in our trials is due to the random selection of the sensitive component, as mentioned in [1]. For BLE, we illustrate results only on the pre-synthesized locked designs, *since authors suggest refraining from re-synthesizing the obfuscated locked designs* [1]. The correctness of the recovered key using Fa-SAT is verified using *Cadence Conformal LEC*. We extend our attack to tackle the Anti-SAT defenses; we show experiments on pre-synthesized and post-synthesized locked benchmarks. We analyze our attack's scalability on total 228 locked designs using node number and iteration count metrics.

B. Definitions

Node number. This denotes the position of the fault in the candidate faults list obtained from *Synopsys Tetramax*.⁸

Iteration. This denotes the number of iterations required by the SAT solver to recover the secret key successfully.

C. Attack results on BLE

We launch our Fa-SAT attack on BLE [1]; we analyze the strength of our attack on pre-synthesized locked netlist. We present our attack results on a total of seven benchmarks and 210 designs. We set a timeout of 90 seconds for the SAT runs.

Fa-SAT on Obfuscated, SL-locked designs. Figure 6(a) shows the box plots for node numbers corresponding to the

⁸Note that this metric is different from depth/level of a logic gate in a design.

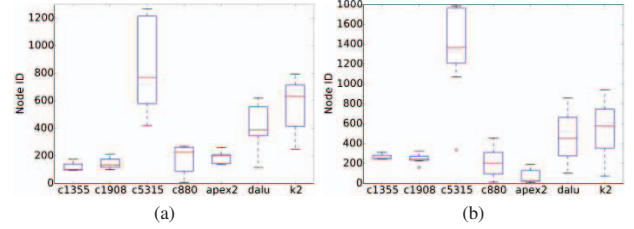


Fig. 6. Box plot for ISCAS-85 and MCNC benchmarks for 15 trials showing the node number at which the fault is inserted, aiding the SAT-based attack to circumvent BLE techniques. (a) Attack on obfuscated SL-locked benchmarks. (b) Attack on obfuscated SFC-locked benchmarks. Each box's upper and lower limits range from the 5th to 95th percentile for the respective data set. The whiskers represent the minimal and maximal values, and the red bars inside the boxes represent the median, and the red dots reflect the outliers.

fault. Figure 7(a) shows the box plots for the number of iterations taken by the SAT solver to retrieve the secret key of the locked circuits successfully. We observed that at max, SAT solver required nine iterations to retrieve the secret key. The farthest node in the locked netlist at which the fault was inserted was 1,267th node. All these results are computed on *pre-synthesized designs*.

Fa-SAT on Obfuscated, SFC-locked circuits. Figure 6(b) shows the box plots for node numbers corresponding to the fault. Figure 7(b) shows the box plots for the number of iterations required for the SAT solver to retrieve the secret key successfully. We use the same set of benchmarks as used in SL and observe a maximum of 241 iterations to recover the secret key and the farthest node in the locked netlist on which fault was inserted being 1,783rd node. Again, all these results are computed on *pre-synthesized designs*.

D. Attack results on Anti-SAT

Setup. We lock ITC-99 benchmarks' *BENCH* files using Anti-SAT technique. For post-synthesis analysis, we convert the *BENCH* files to RTL Verilog using *ABC* and then synthesize them using *Synopsys Design Compiler*. To mimic our attack on post-synthesized files, we convert these Verilog files to *BENCH* format. We launched our Fa-SAT attack on six ITC-99 benchmarks locked with the standalone Anti-SAT scheme and when AntiSAT is combined with RLL.

Fa-SAT on standalone Anti-SAT. Table III outlines the results of our Fa-SAT attack on the standalone Anti-SAT technique. To that end, we locked six ITC-99 benchmarks with a security level of 64, leading to a key-size of 128 [2]. In

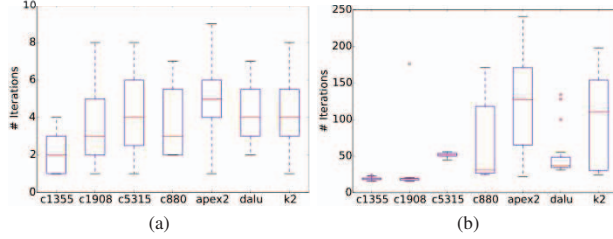


Fig. 7. Box plot for ISCAS-85 and MCNC benchmarks for 15 trials showing iteration count taken by our Fa-SAT attack to break BLE techniques. (a) Attack on obfuscated SL-locked benchmarks. (b) Attack on obfuscated SFC-locked benchmarks. The details about the box plots are same as Fig. 6.

the worst-case scenario, for pre-synthesized Anti-SAT locked designs, a fault inserted at 1,290th node resulted in the correct SAT run to recover the secret key. Across all the runs, the SAT solver required at max 26 iterations to return the secret key; the SAT run terminated within 30 seconds. We found the node at 21,107th location for post-synthesized Anti-SAT locked designs, but the SAT solver returned the secret key within a single iteration for this node.

Fa-SAT on Anti-SAT + RLL. Table III shows the success of our attack in breaking the Anti-SAT + RLL defense to recover the exact key. We locked six ITC-99 benchmarks with a 64-bit Anti-SAT key along with 32 and 64-bit RLL keys. For pre-synthesized locked designs, the attack terminated at an earlier node, whereas for post-synthesized designs, the attack terminated at 22,382nd node in the worst-case scenario. The SAT solver took 29 iterations to retrieve the secret key.

Overall, we observe that the node location identified in pre-synthesized and post-synthesized versions of the same benchmark is different. This arises because the synthesis tools can merge the added locking circuitry with the underlying, original design to a certain extent. However, as our attack results demonstrate, the synthesis procedures employed by the considered synthesis tools have no bearing on our proposed attack's efficacy.

E. Attack scalability

We observe from our experiments that Fa-SAT linearly scales with the circuit size. We also observe that increasing the key-size does not affect the attack success exponentially as observed in seminal [5] and other advanced SAT-based attacks [7], [8]. As seen from Table III, when key-size doubles, it does not have any dependence on the node number, whereas the number of SAT iterations increases linearly.

V. CONCLUSION

In this paper, we question the security promise offered by the recently proposed BLE technique by proposing Fa-SAT. Our attack leverages the principle of fault-injection to aid the seminal SAT-based attack to recover the secret key sooner. Extensive experimental analysis on a plethora of benchmarks from diverse benchmark suites (ISCAS-85, MCNC, and ITC-99) illustrates the efficacy of our fault-aided SAT-based attack. We verified the attack's success by recovering the correct key in all scenarios for two case studies, viz., BLE and Anti-SAT variants locked with multiple keys and synthesized with

TABLE III
ATTACK SUCCESS ON ITC-99 BENCHMARKS LOCKED WITH STANDALONE ANTI-SAT AND WHEN COMBINED WITH RLL

Benchmark	# Keys	Pre-syn		Post-syn	
		Node	# Iter.	Node	# Iter.
b14_C	128/-	556	1	1,824	1
	128/32	6	11	1,535	8
	128/64	2	20	3,509	20
b15_C	128/-	1,290	1	348	1
	128/32	3	17	1,884	13
	128/64	2	18	531	15
b17_C	128/-	1	1	21,107	1
	128/32	2	17	22,382	17
	128/64	3	24	16,222	29
b20_C	128/-	141	1	37	1
	128/32	3	16	1,068	17
	128/64	3	20	2,268	20
b21_C	128/-	160	1	104	1
	128/32	3	14	3,390	12
	128/64	3	26	995	20
b22_C	128/-	1	1	3,966	1
	128/32	2	15	6,365	15
	128/64	2	22	9,116	22

For # Keys: x/y means x Anti-SAT keys and y RLL keys.
x/- denotes standalone Anti-SAT locked with x keys.

industry-standard tools. As a part of future work, we plan to tackle other compound locking techniques to further verify the efficacy of our attack.

ACKNOWLEDGEMENTS

This work was supported in part by the Center for Cyber Security (CCS) at New York University New York/Abu Dhabi. (NYU/NYUAD).

REFERENCES

- [1] A. Rezaei et al., "Rescuing logic encryption in post-sat era by locking & obfuscation," in *Design Automation Test in Europe Conference*, 2020, pp. 13–18.
- [2] Y. Xie et al., "Mitigating SAT attack on logic locking," in *International conference on cryptographic hardware and embedded systems*. Springer, 2016, pp. 127–146.
- [3] J. A. Roy et al., "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
- [4] A. Chakraborty et al., "Keynote: A disquisition on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 1952–1972, 2020.
- [5] P. Subramanyan et al., "Evaluating the security of logic encryption algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2015, pp. 137–143.
- [6] A. Sengupta et al., "Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, 2020.
- [7] K. Shamsi et al., "AppSAT: Approximately deobfuscating integrated circuits," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2017, pp. 95–100.
- [8] Y. Shen et al., "Double dip: Re-evaluating security of logic encryption algorithms," in *Great Lakes Symposium on VLSI*, 2017, pp. 179–184.
- [9] M. Yasin et al., "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517–532, 2020.
- [10] D. Sironi et al., "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.
- [11] K. Shamsi et al., "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Great Lakes Symposium on VLSI*, 2018, pp. 147–152.
- [12] H. M. Kamali et al., "Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks," in *56th Annual Design Automation Conference*, 2019, pp. 1–6.
- [13] K. Z. Azar et al., "NNgSAT: Neural network guided sat attack on logic locked complex structures," in *IEEE/ACM International Conference On Computer Aided Design*, 2020, pp. 1–9.
- [14] U. Guin et al., "Robust design-for-security architecture for enabling trust in IC manufacturing and test," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 818–830, 2018.
- [15] R. Karmakar et al., "A scan obfuscation guided design-for-security approach for sequential circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 3, pp. 546–550, 2019.
- [16] N. Limaye et al., "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access," in *IEEE/ACM International Conference on Computer-Aided Design*, 2019, pp. 1–8.
- [17] N. Limaye et al., "DynUnlock: Unlocking scan chains obfuscated using dynamic keys," in *Design, Automation Test in Europe Conference Exhibition*, 2020, pp. 270–273.
- [18] (2011) NanGate FreePDK45 Open Cell Library. Nangate Inc. [Online]. Available: http://www.nangate.com/?page_id=2325