Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked

Fangfei Yang, Ming Tang, *Member, IEEE*, and Ozgur Sinanoglu, *Senior Member, IEEE*

***Abstract*— Logic locking is a technique that has received significant attention. It protects a hardware design netlist from a variety of hardware security threats, such as tampering, reverse-engineering, and piracy, stemming from untrusted chip foundry and end-users. This technique adds logic and inputs to a given design netlist to make sure that the locked design is functional only when a *key* is applied from the new inputs; an incorrect key makes the design produce incorrect outputs. The new inputs, referred to as the key inputs, are driven by a tamper-proof memory on the chip, which stores the secret key. Research in this field has shown that this technique, if not implemented properly, may be vulnerable to attacks that extract the *key* of logic locking. Recently, a logic locking technique called stripped functionality logic locking (SFLL) has been proposed and shown to withstand all known attacks in a provably secure manner. SFLL strips some functionality from the original design by corrupting its output corresponding to a number of "protected" input patterns. In one version of SFLL, referred to as SFLL-hd, these protected patterns are all of a certain hamming distance *h* to the *key*. The modified design is accompanied by additional logic that fixes the output for each protected input pattern only when the *key* is in the tamper-proof memory. In this paper, we present an attack that breaks SFLL-hd within a minute. Our attack exploits structural traces left behind in the locked design due to the functionality strip operation and is capable of identifying some of the protected patterns. We also present a theoretical framework that helps us develop two different techniques to complete our attack. In the first technique, we use the Gaussian elimination technique to solve a system of equations that we form based on *k*-identified protected patterns in $O(k^3)$ time in the best case, where *k* is the number of key bits in *key*. The second technique uses one identified protected pattern to query the oracle *k* times. In both techniques, we successfully recover the *key* from the protected pattern(s). We show that our attacks work on the SFLL-locked microprocessor design (more than 50 K gates) that the authors of SFLL made available to the public; we extract the 256-bit key within a minute and reveal it in this paper. We also test our attacks on a few other SFLL-hd benchmarks provided by SFLL authors.**

***Index Terms*— Hardware, security, logic gates, hamming distance, reverse engineering.**

I. Introduction

FABLESS IC companies take the advantage of outsourcing the fabrication of the chips in order to overcome the growing cost of semiconductor fabrication. However, the introduction of third-party manufacturers into the IC supply chain also brings security threats because IC designers must provide GDSII layout or other design files to the manufacturers. Through reverse engineering, the malicious manufacturers or attackers can reveal the gate-level netlist and have further understanding of the design, exposing the design to the risk of piracy [1], counterfeiting [2] or hardware trojans [3].

A. Logic Locking

Logic locking is a technique that protects the design against the untrustworthy IC supply chain. Compared to other techniques such as hardware metering [4], [5] or split manufacturing [6], logic locking can cover the entire supply chain with minor impact to the original design structure, timing or performance.

Logic locking protects the design by adding some ***key*** related circuitry/gates into the original design, so the circuit will not work without a correct ***key*** and it hides the original design functionality [7]–[16]. As a result, unauthorized foundries are not able to pirate this circuit or analyze the original design directly. The designer can activate this chip by loading the ***key*** on its tamper-proof memory after manufacturing to get functional products.

Currently, this kind of protection is vulnerable to different attacks such as SAT based attack [17]–[19], sensitization attack [9] and signal skew analysis/removal attack [20]. All these attacks aim at recovering the ***key*** of logic locking. The SAT attack is a key search space pruning technique while the sensitization attack aims at reading the key bits one at a time from the outputs by using input patterns that can sensitize the key bits. The signal skew analysis attack exploits the structural vulnerability of Anti-SAT technique [15].

Manuscript received July 2, 2018; revised December 10, 2018 and January 27, 2019; accepted March 7, 2019. Date of publication March 13, 2019; date of current version June 14, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61202386, Grant 61332019, and Grant 61472292, in part by the National Basic Research Program of China (973 Program) under Grant 2014CB340601, in part by the Key Technology Research of New-generation High-speed and High-level Security Chip for Smart Grid under Grant 526816160015, in part by the Technological Innovation of Hubei Province (Major Special Project) under Grant 2018AAA046, in part by the NSFC-General Technology Basic Research Joint Foundation under Grant U1636107, and in part by the Defense Advanced Research Projects Agency Obfuscated Manufacturing for GPS (OMG) Program. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jean-Luc Danger. *(Corresponding author: Ming Tang.)*

F. Yang is with the School of Computer, Wuhan University, Wuhan 430072, China.

M. Tang is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: m.tang@126.com).

O. Sinanoglu is with the Department of Computer Engineering, New York University Abu Dhabi, Abu Dhabi, United Arab Emirates.

Digital Object Identifier 10.1109/TIFS.2019.2904838

The threat model in most of the attacks on logic locking is that the attacker can get the protected (i.e., locked) gate-level netlist and have an oracle (i.e., a working chip embedded with the ***key***) that yields correct output. The attacker can only exercise the oracle by applying inputs and observing its outputs but cannot access the ***key*** embedded in the oracle; she controls/observes the I/O pins (scan chains & functional I/Os) through JTAG. The attacker uses the protected design netlist to find so-called distinguishing input patterns [17] (DIPs) by using a SAT solver; a DIP is an input pattern that can effectively prune the key space. The attacker can verify the outputs by applying the same DIP to the oracle; all the keys that lead to a different output for this DIP can be eliminated, thus reducing the size of possible key space dramatically. This process is repeated until the key search space contains only the ***key***.

Some protection structures (SFLL [8], Anti-SAT [15], SAR-Lock [16]) have been proposed to resist the SAT attack by adding a specialized logic block which restrains the effectiveness of key pruning in traditional SAT attack, raising the actual complexity of SAT solver.

Also, some new attacks have recently been proposed as well. These are AppSAT [19], Double DIP [18], and Bypass attacks [21].

Compared with the other logic locking techniques, SFLL provides a quantifiable and provable resilience trade-off between all known and anticipated attacks [8]. It modifies a given circuit into a Functionality Stripped Circuit (FSC) by flipping the output corresponding a set of input patterns, which are referred to as the *protected patterns*; a protected pattern is an input pattern for which the original circuit and the FSC produce different outputs. In SFLL-hd, all of these protected input patterns have a hamming distance of h to the ***key***. To restore the functionality of the original circuit from the FSC on the final chip that embeds the ***key***, a restore unit detects a protected input pattern by checking its hamming distance to the ***key*** and restores the output to its correct value. SFLL-hd technique is based on a secret ***key*** and a set of protected patterns dictated by the ***key*** and the hamming distance h; the protected patterns are also a secret, but h is not [8]. SFLL-hd is provably secure against key-pruning attacks such as the SAT attack [8].

We are interested in targeting SFLL since it is a very recent technique that thwarts all attacks such as SAT, sensitization and removal attack. This paper [8] also serves as a valuable tutorial on logic locking, as it explains the weaknesses of other logic locking solutions in the literature.

B. Weakness in SFLL-hd and Attack Approach

SFLL threat model assumes that the attacker has access to a netlist and a working chip while the key and the protected patterns are secret. In fact, SFLL authors admit that if the protected patterns were leaked, the secret key can be recovered as well; however, the paper provides no information as to how this can be accomplished. Our attack therefore identifies protected pattern(s) through a structural analysis of the netlist and then recovers the key from this information. We exploit a structural vulnerability in the SFLL-hd implementation to identify a number of protected patterns. This structural vulnerability is an end result of the particular way that the functionality strip operation is implemented. Current design synthesis tools are all security-oblivious; SFLL-hd too suffers from this, leaving traces in the circuit about the key (visible to reverse engineers). Once we identify a number of protected patterns, we perform one of the two following techniques to find the ***key*** based on these protected patterns. We have developed these techniques based on the hamming distance relationship between each protected pattern and the ***key***:

1) We create a system of linear equations from a set of k protected patterns and utilize Gaussian elimination to solve for the ***key***. Without querying the oracle, we can recover the ***key*** in $O(k^3)$ in the best case.
2) Given a single protected pattern, we apply inputs derived from this protected pattern to the oracle. Based on the oracle responses, we classify all these patterns as protected or non-protected. By analyzing the distribution of these patterns, we recover the ***key*** in $O(k)$.

C. Structure of the Paper

In section II, we briefly introduce SFLL-hd. Next, in section III, we explain the attack framework. In section IV, we explain our reverse engineering based structural analysis to identify traces in the netlist that help us obtain the protected patterns. In section V, we discuss the properties of the hamming distance function and provide two different algorithms to recover the ***key*** from a set of protected patterns. Finally in section VI, we show the application of the attack to recover the ***key*** in less than a minute from the benchmark circuit provided by the authors of SFLL on Github [22]. **This is the only benchmark circuit that is publicly available** and is a controller that mainly consists of a ARM Cortex-M0 microprocessor [23] design locked by SFLL-hd with a configuration of $h = 32$, $k = 256$. We further run our attack and show its success on a few other SFLL-hd benchmarks that we obtained from SFLL authors.

D. Novelty and Contribution

We present in this work a vulnerability that we identified in the state-of-the-art logic locking technique SFLL-hd. This vulnerability is an end-result of the fact that existing synthesis tools are security-oblivious. It is clear from the netlists we analyzed that the SFLL authors indeed resynthesized their netlists to hide the protected patterns; however, our structural attack is able to identify the protected pattern(s) in all the cases, in turn extracting the secret ***key***. Until a truly security-aware synthesis tool is developed, any defense that relies on conventional CAD tools will be vulnerable. By eliciting a vulnerability in a state-of-the-art logic locking technique that has been unbroken until now, our work emphasizes a major shortcoming, i.e., the need for the development of a security-aware synthesis tool, thus identifying a very important research direction.

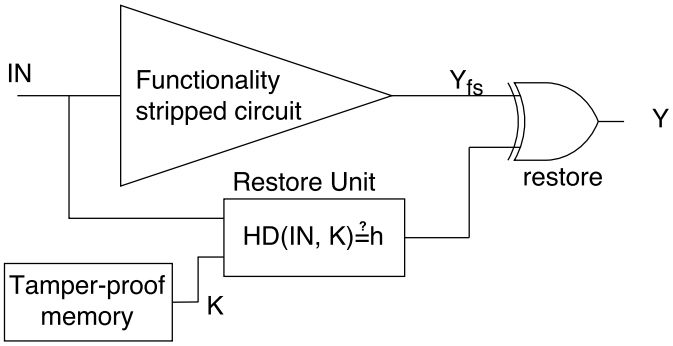

Fig. 1. SFLL-hd protection scheme [8].

II. Introduction of SFLL-hd

The SFLL-hd scheme modifies the original circuit into FSC as shown in figure Figure 1. It focuses on a specific set of k bits of the n-bit inputs and flips the response of $2^{n-k}\binom{k}{h}$ input patterns; each one of these protected patterns has a hamming distance of h to the ***key***, as far as the k bits of the input patterns are considered. The original circuit with $2^{n-k}\binom{k}{h}$ errors in its output space now becomes the Functionality Stripped Circuit (FSC). The restore unit under the FSC calculates the hamming distance between current input and the ***key*** stored in tamper-proof memory and checks whether this distance is h; if so, the current input is identified as one of the protected patterns, and thus, the restore unit flips the FSC output Y_{fs} accordingly and restores the correct output Y. The restore unit can properly fix the output for all the protected patters only if the correct ***key*** is loaded on the tamper-proof memory.

The authors of SFLL proved that under this design, SFLL-hd can thwart the SAT attack, removal attack, and sensitization attack by adjusting h. Their security analysis shows that SFLL-hd is $\left(k - \left\lceil log_2\binom{k}{h}\right\rceil\right) - secure$ against SAT attack, $(2^{n-k}\binom{k}{h}) - secure$ against removal attack, and $k - secure$ against sensitization attack [8]. For the SAT attack, $\lambda - security$ is defined to be the probability of an attacker success in reconstructing the original circuit for a polynomial number of queries $q(\lambda)/2^{\lambda}$. For the removal attack, the security is defined as the number of protected input patterns.

III. Possible Weakness in SFLL

A. The Security Impact of h

The authors of SFLL suggest that h can be adjusted to trade resilience to one attack for resilience to another [8]. When $h \to 0$ or $h \to k$ this protection has the best resistance toward SAT and other key-pruning attacks. In contrast, it achieves the best resistance to removal attacks when $h \to \frac{k}{2}$.

Hence, when h approaches to $\frac{k}{2}$, there would be more input vectors for which FSC produces incorrect output. In this scenario, the removal of the restore unit would corrupt the circuit functionality maximally. However, it is more likely for the attacker to get an input that causes the FSC to produce incorrect outputs and reveal some information about the ***key***. For example, when $k = 256, h = 128$, the possibility of getting such an input would be 4.9819%.

When h approaches to 0, there is a low possibility that FSC generates an incorrect output so the attacker can hardly learn about the ***key***; the number of protected patterns is very few. But, by removing the restore unit, the attacker can get an almost identical-functionality circuit as the original design, since the restore unit has so little impact on the circuit. For example, the authors of SFLL used $k = 256, h = 32$ in their benchmark and if we remove the restore unit, the FSC would have a possibility of $5.03 \times e^{-37}$ to produce an error for any given input pattern. The possibility is so low that it is almost impossible for FSC to generate a wrong output.

We believe that it is difficult for a hardly triggered protection to actually protect a circuit. So it seems that either we need a relatively large h, or select the ***key*** and h carefully so that the FSC produces an error more frequently. However, can these tricks really improve security of the protection?

B. Attack Framework

The first step in our attack is to identify the protected pattern(s). We reverse engineer the netlist and then perform a connectivity analysis to identify a signal specific to the SFLL-hd implementation. The activation condition of this signal provides information about the protected pattern(s); we use a SAT solver to compute the secret protected pattern(s).

We next rely on the two features of hamming distance to break the protection of SFLL-hd:

1) The logic expression of hamming distance can be converted to a linear equation and solved by Gaussian elimination.
2) By manipulating the bits of a protected pattern and querying the oracle, we can identify the bits of the protected pattern that contributes to the hamming distance of h to the ***key***.

This way, we can recover the ***key*** of SFLL-hd from its protected pattern(s).

In a nutshell, we can recover the ***key*** as long as we identify a few protected patterns. We rely on our earlier assumption that the FSC should create an error frequently enough for the protection to be meaningful. And for such scenarios, our attack will break SFLL-hd.

IV. Identifying Protected Patterns in SFLL-hd

We first analyze the topology structure of the circuit. The basic structure of SFLL-hd is shown in Figure 1, which comprises mainly the Functionality Stripped Circuit and the restore unit. In this section, we use the publicly available locked microprocessor design as an example to illustrate this part of the attack; later we show that the same attack is successful on other SFLL-hd benchmarks provided by the SFLL authors.

The dfx_sfll_k256_h32.bench [22] contains more than a simple SFLL-hd. It contains many useless nodes, inputs and outputs that conceal the restore unit and the FSC. From Figure 1, we know that only the restore unit receives the ***key***, which comes from the tamper-proof memory and is marked with a prefix 'keyinput' in the benchmark. We can separate the restore unit from the rest of the circuit by tracing these

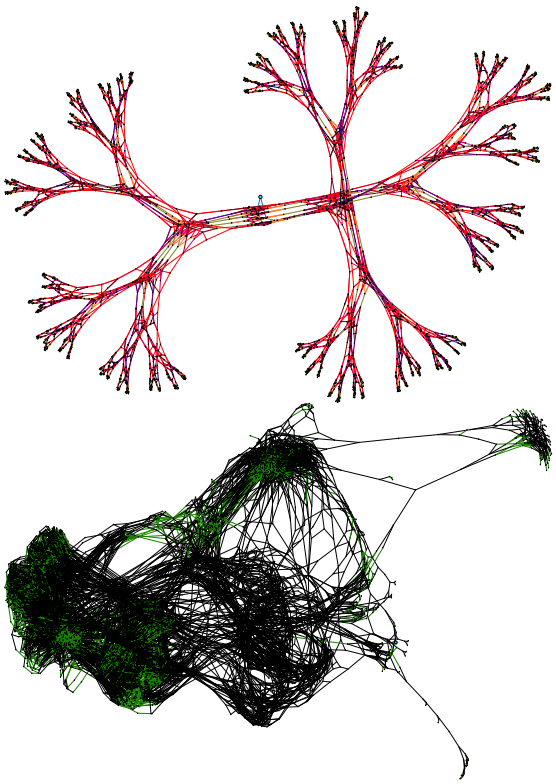

Fig. 2. Logic cone of two signals; at the top, the signal perturb, and at the bottom, another arbitrary signal n34544. Green nodes are input, Cyan nodes are output. These connectivity graphs have been generated from the implementation in Github [22].

keyinput nodes. Then, we follow $keyinput_i$ and figure out the only output node that is affected by *keyinput*, which is N29051 in this benchmark. This is the output protected by SFLL-hd in the circuit.

Next, we trace back from the protected output (node N29051) to compute its fan-in logic cone, i.e., the inputs and gates that feed this output. This way, we can isolate this part of the circuit from the rest, and obtain the protected logic cone in Figure 1. Because the restore unit (i.e. the *keyinput* related circuit) contains no information about the real ***key***, we further separate the logic cone into two parts: the restore unit and the functionality stripped circuit (FSC); we focus on FSC as the vulnerabilities we are looking for are embedded there. This way, we can identify signals connecting the two parts.

We identify a few such signals; for two such signals, we provide a graph that shows their fan-in cone in Figure 2. In these graphs, the nodes are gates, primary inputs or outputs and the edges are nets connecting them. One of these signals is rather attractive; it's called perturb in the benchmark and it has a topologically symmetrical and tree-like structure (the graph at the top of Figure 2). After comparing other signals we traced (for example, the other graph for n34544 at the bottom of Figure 2) we suspect that the functionality stripping is achieved by this logic cone which is a restore unit-like circuit synthesized with hardcoded values of ***key*** that generates a flip signal, which we denote as PP, to flip

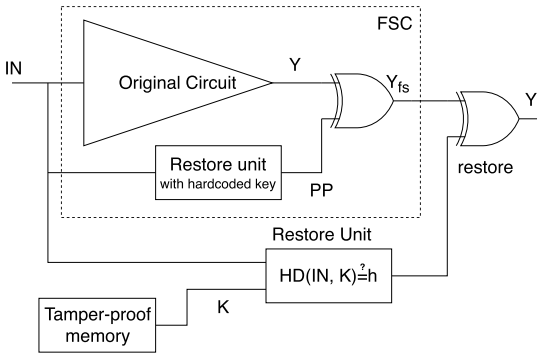

Fig. 3. Speculated implementation of functionality stripped circuit from the original circuit.

the output of the original circuit for the protected patterns. This particular implementation XORs PP with the original circuit to strip its functionality and produce FSC. We depict this understanding of the FSC implementation from the original circuit in Figure 3. While the logic synthesis steps hide this exact implementation and the values of the ***key*** from us, PP signal remains as part of the protected netlist to be utilized by our attack.

With this understanding of FSC structure in mind, the PP signal must have the following properties:

1) it should be around to the boundary between the restore unit and FSC, since it flips the output of the original circuit and passes it to the restore unit.
2) it must be driven by all the relevant inputs, but not by any *keyinput* node.

Then, we confirm that this signal is a very low activity signal; it is likely to be 0 most of the time. We use a SAT Solver [24] on this logic cone to set this signal to a 1 and produce one of the protected patterns. To confirm that this is a valid (protected) pattern, we simulate FSC with this pattern and expect that it produces a different output than the oracle.

A previously proposed signal skew analysis/removal attack [20] identifies a security-critical signal in the Anti-SAT defense. This attack aims at finding two oppositely skewed signals that converge at an AND gate. In our attack, on the other hand, we try to locate a signal PP specific to SFLL. Our analysis is based on graph connectivity rather than signal probabilities. Both attacks are defense-specific.

One may also consider a hypothetical enhancement of SFLL-hd based on the enhancements suggested earlier on Anti-SAT [25]. However, enhancement of Anti-SAT was towards fixing its specific vulnerability, i.e., to nullify the signal probability skew; this is of no use for SFLL-hd, as our attack is not based on signal probabilities. The other technique that enhances Anti-SAT increases the connectivity between the newly added block and the original logic to prevent simple removal attacks. This enhancement technique is of no use for SFLL-hd either; our attack expects and searches for traces stemming from a hamming distance checker embedded into the original circuit for functionality stripping. Any added artificial connectivity would not be able to hide information about the protected patterns.

TABLE I
TABLE OF NOTATION

$\boldsymbol{a}[i]$	$i-th$ entity of vector $\boldsymbol{a}$
$\boldsymbol{A}_{i,j}$	$i-th$ row and $j-th$ column of a matrix $\boldsymbol{A}$
k	Size of $\boldsymbol{key}$ vector used by SFLL-hd
h	Hamming distance used by SFLL-hd
$\boldsymbol{key}$	$\boldsymbol{key}$ used by SFLL-hd
$\boldsymbol{x}$	Candidate for $\boldsymbol{key}$ used by SFLL-hd
$oracle$	Function $oracle$ that produces a correct output for any input
$lock$	Function $lock$ implemented by the Functionality Stripped Circuit (FSC)
s	The number of protected patterns identified through structural analysis of the netlist
IN	Primary inputs to the design
K	k-bit signal which is equal to $\boldsymbol{key}$ in an activated chip
PP	One-bit signal that is one only when one of the protected patterns is applied through inputs IN
Y	The output of the original design
$\boldsymbol{Y}_{fs}$	The output of the functionality stripped design
$\boldsymbol{a}$	A protected pattern vector of k-dimension consisting only of the k bits of interest
$\boldsymbol{r}$	A k-dimension vector $\boldsymbol{r}$ with a hamming weight of h such that $\boldsymbol{a} = \boldsymbol{key} \oplus \boldsymbol{r}$
$\boldsymbol{A}$	Matrix made up of vectors $\boldsymbol{a}_i$
$\boldsymbol{B}$	Matrix of coefficients obtained from matrix $\boldsymbol{A}$
$\boldsymbol{b}$	k-dimension row vector from $\boldsymbol{B}$
$\boldsymbol{g}$	k-dimension matrix of constants obtained from matrix $\boldsymbol{A}$
J	Function that tests whether an input vector is a protected pattern
$eqset$	A set that contains i that $J(\boldsymbol{a}_i) = 0$
$neqset$	A set that contains i that $J(\boldsymbol{a}_i) = 1$
hd	Hamming distance function $hd(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \boldsymbol{x}[i] \oplus \boldsymbol{y}[i]$
$\oplus$	XOR operation

V. RECOVERING *Key* FROM PROTECTED PATTERN(S)

In this paper, we provide two different attacks for a ***key*** size of k:

1) The first attack needs s protected input patterns. By utilizing Gaussian elimination, we can solve the ***key*** in $O(k^3 + 2^{k-s}k)$. And in the best case, $s = k$; only $O(k^3)$ of time is needed. The first attack does not need to query the oracle.
2) The second attack needs only one protected pattern and also a working oracle. By querying the oracle with slightly manipulated versions of the protected pattern, we can identify whether a particular bit of the protected pattern is contributing to hamming distance h to ***key***. This way, the attack can recover the ***key*** in $O(k)$ and in the process queries the oracle k times.

A. Recovering *Key* by Gaussian Elimination

According to the structure of SFLL-hd, we know that a protected pattern vector $\boldsymbol{a} \in \{0,1\}^k$ will satisfy the Equation 1.

$$\boldsymbol{a} = \boldsymbol{key} \oplus \boldsymbol{r} \tag{1}$$

where $\boldsymbol{r} \in \{0,1\}^k$ is a vector with a hamming weight of h, in other words, it has exactly h bits of '1's.

Thus, if we test that the ***key*** is $\boldsymbol{x}$, then the hamming distance between $\boldsymbol{x}$ and every $\boldsymbol{a}_i$ would be $hd(\boldsymbol{x}, \boldsymbol{a}_i) = h$, where $\boldsymbol{a}_i$ means $i-th$ protected pattern vector. Consider these k equations,

$$\begin{cases} hd(\boldsymbol{a}_1, \boldsymbol{x}) = h \\ hd(\boldsymbol{a}_i, \boldsymbol{x}) = h \\ \dots \\ hd(\boldsymbol{a}_k, \boldsymbol{x}) = h \end{cases} \tag{2}$$

We can recover ***key*** by solving this system of equations, which may be hard to accomplish directly; the hamming distance can be represented as $hd(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \boldsymbol{x}[i] \oplus \boldsymbol{y}[i] = \sum_i \boldsymbol{x}[i](1 - \boldsymbol{y}[i]) + \boldsymbol{y}[i](1 - \boldsymbol{x}[i])$. Equation 2 can be transformed to a linear system-like equations as in Equation 3, where $\boldsymbol{A}_{i,j}$ is the $j-th$ binary bit of $i-th$ input vector,

$$\begin{cases} \sum_j \boldsymbol{A}_{1,j}(1 - \boldsymbol{x}[j]) + (1 - \boldsymbol{A}_{1,j})\boldsymbol{x}[j] = h \\ \sum_j \boldsymbol{A}_{i,j}(1 - \boldsymbol{x}[j]) + (1 - \boldsymbol{A}_{i,j})\boldsymbol{x}[j] = h \\ \vdots \\ \sum_j \boldsymbol{A}_{k,j}(1 - \boldsymbol{x}[j]) + (1 - \boldsymbol{A}_{k,j})\boldsymbol{x}[j] = h \end{cases} \tag{3}$$

Considering that $\boldsymbol{A}_{i,j}$ is constant, Equation 3 is a linear system and all variables are limited to $\{0, 1\}$. It is still hard to solve, but if we relax the binary constraint on the variables, this linear system can be solved directly by Gaussian elimination.

We can obtain a unique solution of a linear system of k variables as long as we have k linearly independent equations, and we know that the original equations must have a root that satisfies $x \in \{0,1\}^k$. Furthermore, based on the uniqueness of the root in linear system and that the relaxation of the binary constraint will only add more possible roots for the equations, this unique root must be the root of the original equations.

Because $\boldsymbol{A}_{i,j} \in \{0,1\}$, each entity in the summation $\boldsymbol{A}_{i,j}(1 - \boldsymbol{x}[j]) + (1 - \boldsymbol{A}_{i,j})\boldsymbol{x}[j]$ has only two possible expansions:

$$\boldsymbol{A}_{i,j}(1 - \boldsymbol{x}[j]) + (1 - \boldsymbol{A}_{i,j})\boldsymbol{x}[j] = \begin{cases} -\boldsymbol{x}[j] + 1 & \text{if } \boldsymbol{A}_{i,j} = 1 \\ \boldsymbol{x}[j] & \text{if } \boldsymbol{A}_{i,j} = 0 \end{cases} \tag{4}$$

Also, in Equation 4, $\boldsymbol{x}[i]$ is multiple by a factor of ± 1 based on $\boldsymbol{A}_{i,j}$. After putting all these factors into a matrix $\boldsymbol{B}$ and moving all constant 1s to the right side of the equations to form a vector $\boldsymbol{g}$, we have a matrix equation,

$$\boldsymbol{B}\boldsymbol{x} = \boldsymbol{g}. \tag{5}$$

where,

$$\boldsymbol{B}_{i,j} = \begin{cases} 1 & \text{if } \boldsymbol{A}_{i,j} = 0 \\ -1 & \text{if } \boldsymbol{A}_{i,j} = 1 \end{cases} \tag{6}$$

$$\boldsymbol{g}[i] = h - \sum_j \boldsymbol{A}_{i,j}. \tag{7}$$

As Terence Tao and Van Vu discussed, the upper bound for a random ± 1 matrix to be singular is $.939^n$ [26]. So, in most cases where h is not too small, k different protected input vectors are enough to solve the equations. When $h = k/2$, we can only find $k - 1$ protected input vectors, but

by enumerating the free variable, we can find only two roots that contain only 0 and 1; it must be the two complementary roots and either one can be used as ***key***.

So, if we assume we can obtain exactly k or $k-1$ protected input vectors, the complexity of this attack will be $O(k^3)$.

And, in a practical setting, if we can get only s protected input patterns, we can still enumerate the free variables and complete this attack with a complexity of $O(k^3 + 2^{k-s}k)$; but we may get more than one possible solution.

B. Recovering ***Key*** *by Querying the Oracle*

The oracle in logic locking is a blackbox that provides the correct output for a given input. The attacker cannot see the intermediate process and operations inside the oracle.

The starting point of this attack is a protected pattern $\boldsymbol{a}$. As $\boldsymbol{a} = \boldsymbol{r} \oplus \boldsymbol{key}$ where ***key*** is the vector we try to recover and $\boldsymbol{r}$ is a vector that contains exactly h '1's, we need to find the location of these h '1's in $\boldsymbol{r}$, so we recover ***key***.

Because we have such an oracle, we can design a function $J(IN)$ to check whether the oracle and the FSC produce the same output, and thus, check whether IN meets the requirement of the $\boldsymbol{a}$; i.e., whether IN is a protected pattern.

$$J(IN) = \begin{cases} 0 & if\ oracle(IN) = lock(IN), \\ 1 & if\ oracle(IN) \neq lock(IN). \end{cases} \tag{8}$$

Let $\boldsymbol{a}_0$ be the vector which flips $\boldsymbol{a}[0]$ in $\boldsymbol{a}$, $\boldsymbol{a}_0 = \left\{\overline{\boldsymbol{a}[0]}, \boldsymbol{a}[1], \ldots, \boldsymbol{a}[k-1]\right\}$. Because the hamming distance between $\boldsymbol{a}$ and ***key*** is the hamming weight of $\boldsymbol{r}$, the hamming distance between $\boldsymbol{a}_0$ and ***key*** will be ± 1 depending on whether $\boldsymbol{r}[0]$ is 0 or 1:

$$hd(\boldsymbol{a}_0, \boldsymbol{key}) = \begin{cases} hd(\boldsymbol{a}, \boldsymbol{key}) + 1 & \boldsymbol{r}[0] = 0, \\ hd(\boldsymbol{a}, \boldsymbol{key}) - 1 & \boldsymbol{r}[0] = 1. \end{cases} \tag{9}$$

In either case, the distance will not be h, so $J(\boldsymbol{a}_0) = 0$.

Then, we flip each bit $\boldsymbol{a}_0[i], i > 0$, in $\boldsymbol{a}_0$ and obtain $\boldsymbol{a}_i$, which is different than $\boldsymbol{a}$ in exactly two bits in locations 0 and i. Now, we can split i into two sets based on $J(\boldsymbol{a}_i)$: $eqset = \{i | J(\boldsymbol{a}_i) = 0\}\ and\ neqset = \{i | J(\boldsymbol{a}_i) = 1\}$.

We have two possible cases, depending on $\boldsymbol{r}[0]$:

1) If $\boldsymbol{r}[0] = 0$, $hd(\boldsymbol{a}_0, \boldsymbol{key}) = h + 1$. When $J(\boldsymbol{a}_i) = 1$, $hd(\boldsymbol{a}_i, \boldsymbol{key}) = h$, if and only if $\boldsymbol{r}[i] = 1$. So, $neqset$ contains all the h positions that $\boldsymbol{r}[i] = 1$.
2) If $\boldsymbol{r}[0] = 1$, $hd(\boldsymbol{a}_0, \boldsymbol{key}) = h - 1$. When $J(\boldsymbol{a}_i) = 0$, $hd(\boldsymbol{a}_i, \boldsymbol{key}) = h - 2$, if and only if $\boldsymbol{r}[i] = 1$. So, $eqset$ contains all the h positions that $\boldsymbol{r}[i] = 1$.

By querying the oracle, the attacker can learn the value of $J(\boldsymbol{a}_i)$ and classify i accordingly into $neqset$ and $eqset$. As h is public information, so long as $h \neq k/2$, the size of $neqset$ and $eqset$ will be different in either case. The set that contains h elements will reveal the bit locations where $\boldsymbol{a}$ and ***key*** differ (locations of '1's in $\boldsymbol{r}$); by flipping these bits of $\boldsymbol{a}$, ***key*** can be recovered. $h = k/2$ can also be handled by repeating the process above for a second time, this time with another bit in $\boldsymbol{a}$; however, as $h = k/2$ is the maximally SAT attack vulnerable case for SFLL-hd that would never be implemented, we do not explain the details of such a procedure.

TABLE II

EXAMPLE OF $\boldsymbol{r}[0] = 1, \boldsymbol{key} = 01011000, h = 3$

	0	1	2	3	4	5	6	7	Result
$\boldsymbol{a}$	1	0	0	1	1	0	1	0	$h = 3, J = 1$
$\boldsymbol{r}$	1	1	0	0	0	0	1	0	
$\boldsymbol{key}$	0	1	0	1	1	0	0	0	
$\boldsymbol{a}_0$	0	0	0	1	1	0	1	0	$h = 2, J = 0$
$\boldsymbol{a}_1$	0	1	0	1	1	0	1	0	$h = 1, J = 0$
$\boldsymbol{a}_2$	0	0	1	1	1	0	1	0	$h = 3, J = 1$
$\boldsymbol{a}_3$	0	0	0	0	1	0	1	0	$h = 3, J = 1$
$\boldsymbol{a}_4$	0	0	0	1	0	0	1	0	$h = 3, J = 1$
$\boldsymbol{a}_5$	0	0	0	1	1	1	1	0	$h = 3, J = 1$
$\boldsymbol{a}_6$	0	0	0	1	1	0	0	0	$h = 1, J = 0$
$\boldsymbol{a}_7$	0	0	0	1	1	0	1	1	$h = 3, J = 1$

TABLE III

EXAMPLE OF $\boldsymbol{r}[0] = 0, \boldsymbol{key} = 01011000, h = 3$

	0	1	2	3	4	5	6	7	Result
$\boldsymbol{a}$	0	0	0	1	0	0	1	0	$h = 3, J = 1$
$\boldsymbol{r}$	0	1	0	0	1	0	1	0	
$\boldsymbol{key}$	0	1	0	1	1	0	0	0	
$\boldsymbol{a}_0$	1	0	0	1	0	0	1	0	$h = 4, J = 0$
$\boldsymbol{a}_1$	1	1	0	1	0	0	1	0	$h = 3, J = 1$
$\boldsymbol{a}_2$	1	0	1	1	0	0	1	0	$h = 5, J = 0$
$\boldsymbol{a}_3$	1	0	0	0	0	0	1	0	$h = 5, J = 0$
$\boldsymbol{a}_4$	1	0	0	1	1	0	1	0	$h = 3, J = 1$
$\boldsymbol{a}_5$	1	0	0	1	0	1	1	0	$h = 5, J = 0$
$\boldsymbol{a}_6$	1	0	0	1	0	0	0	0	$h = 3, J = 1$
$\boldsymbol{a}_7$	1	0	0	1	0	0	1	1	$h = 5, J = 0$

Table II gives an example of the attack process where $\boldsymbol{r}[0] = 1, \boldsymbol{key} = 01011000$, $h = 3$ and the protected input pattern $\boldsymbol{a} = 10011010$. We show that i is classified into $\{0, 1, 6\}$ and $\{2, 3, 4, 5, 7\}$ by querying the oracle repeatedly. As we know $h = 3$, we can figure out $\boldsymbol{r} = 11000010$ and recover ***key*** by $\boldsymbol{a} \oplus \boldsymbol{r} = 01011000$.

Table III provides another example of the attack process where, this time, $\boldsymbol{r}[0] = 0$ for another protected input pattern $\boldsymbol{a} = 000100010$; the ***key*** and h are the same as the previous example. This time, i is classified into $\{1, 4, 6\}$ and $\{0, 2, 3, 5, 7\}$ by querying the oracle repeatedly. By flipping the bits in positions $\{1, 4, 6\}$ in $\boldsymbol{a}$, we obtain the ***key*** again.

C. Comparison of the Two Attacks

The first attack requires a larger number of protected patterns but it can work without an oracle. It is therefore more effective than the second attack when it is hard to gain access to a physical working chip, but the attacker can somehow (e.g., illegal means) obtain protected patterns from designers.

The second attack, on the other hand, is easier to launch as it only requires one protected pattern; but at the same time, it requires a working chip to be repetitively queried as an oracle. However, almost all the other attacks in the literature assume access to an oracle.

VI. EXPERIMENTAL RESULTS

We create an automatic analysis tool that performs all these analyses, including the reverse engineering and tracing of the locked netlist. The tool will first try to isolate the restore unit and find $keyinput$-related nodes and the sub-circuit that contains information about ***key***. Then it uses a SAT Solver to compute the protected patterns, which will be used to recover

TABLE IV
PERFORMANCE OF THE ATTACKS

Target				Time (seconds)						
Name	Size	H	K	Preprocessing	Oracle Based SAT	Oracle Based Queries	Oracle Based Sum	Gaussian elimination SAT	Gaussian elimination Equations	Gaussian elimination Sum
dfx_sfll_k256_h32.bench	49412	32	256	0.9080	1.3739	40.2994	42.5813	33.6596	0.0805	34.6481
b14_128_bits_MP_0_enc_RTL.v	3449	0	128	0.2807	0.2241	1.5165	2.0213	Unavailable	Unavailable	Unavailable
b14_128_bits_MP_12_enc_RTL.v	3458	12	128	0.2803	0.1688	1.3736	1.8227	3.7060	0.0678	4.0541
b14_128_bits_MP_4_enc_RTL.v	3454	4	128	0.2275	0.1777	1.3530	1.7582	Unavailable	Unavailable	Unavailable
b14_128_bits_MP_8_enc_RTL.v	3513	8	128	0.2870	0.1682	1.3271	1.7823	3.7831	0.0091	4.0793
b15_128_bits_MP_0_enc_RTL.v	4346	9	128	0.2974	0.1745	1.8967	2.3686	Unavailable	Unavailable	Unavailable
b15_128_bits_MP_12_enc_RTL.v	4526	12	128	0.3403	0.1682	1.9296	2.4381	3.8031	0.0080	4.1515
b15_128_bits_MP_4_enc_RTL.v	4468	4	128	0.3023	0.1905	1.8874	2.3802	Unavailable	Unavailable	Unavailable
b15_128_bits_MP_8_enc_RTL.v	4485	8	128	0.3316	0.1680	1.9379	2.4375	3.8146	0.0092	4.1554
b17_128_bits_MP_0_enc_RTL.v	13168	0	128	0.9402	0.1831	1.7643	2.8876	Unavailable	Unavailable	Unavailable
b17_128_bits_MP_12_enc_RTL.v	13184	12	128	0.9512	0.1787	1.7560	2.8860	3.8453	0.0090	4.8055
b17_128_bits_MP_4_enc_RTL.v	13279	4	128	0.9248	0.1664	1.8332	2.9243	Unavailable	Unavailable	Unavailable
b17_128_bits_MP_8_enc_RTL.v	13344	8	128	0.9696	0.1645	1.8020	2.9362	3.8758	0.0089	4.8543
b18_128_bits_MP_0_enc_RTL.v	37675	0	128	2.8295	0.1704	4.2601	7.2600	Unavailable	Unavailable	Unavailable
b18_128_bits_MP_4_enc_RTL.v	37482	4	128	2.8607	0.2488	5.7847	8.8941	Unavailable	Unavailable	Unavailable
b18_128_bits_MP_8_enc_RTL.v	36352	8	128	2.6872	0.1700	3.5959	6.4531	3.9755	0.0113	6.6739

the ***key*** by the Gaussian elimination attack and the oracle based attack respectively. We make this tool available as a python script on Github.

A. Results on Publicly Available Locked Microprocessor Design

The authors of SFLL provided a locked circuit ($h = 32$, $k = 256$) netlist in Github [22]. This is a microcontroller that uses ARM Cortex-M0 microprocessor [23] and has more than 50K gates. The microcontroller also includes ARM AHB-Lite as its BUS, UART interface, and 64 KB of SRAM. As they also provided an oracle (software model for their working chip), the first step of our analysis is to find a protected input pattern. We show that we can achieve this by reverse engineering the given netlist and analyzing its structure; we then apply our attacks in the previous section to recover ***key***. To complete our attack, we reorder the bits of ***key*** based on the XOR operation with the inputs. Finally, we acquire $keyinput0$ to $keyinput255$:

8ef4d4dd81aa036ab431b8f7ec6c4055a990bd138a77e4f4ec
7187ffd287cffb

We run the attack outlined here on an Intel(R) Core(TM) i5-5200U CPU @2.20GHz computer. It takes less than a minute to finish either of the two attacks; the detailed execution time for each attack is shown in Table IV.

B. Results on Other SFLL-hd Benchmarks

Here we report the results of our attack on the other SFLL-hd benchmarks provided to us by the SFLL authors; we are grateful for their help. These benchmarks are different from the public one. They are synthesized netlists in verilog with a key size of 128 and varying h values. These benchmarks are significantly smaller than the public one, and naturally, our attack also runs much faster.

In Table IV, we report the benchmark, its gate count, and the SFLL-hd parameters (h value used and the key size). Regarding our attacks, we report the preprocessing time to read in the netlist and create data structures, and the attack times; we report the computational time for the SAT solver to extract the protected pattern(s), if it can, and the time it takes to either solve equations through Gaussian Elimination or query the oracle repeatedly. We show the execution time of each step individually; we also report the total execution time for the two attacks.

The performance data of Gaussian elimination in $h \leq 4$ cases is unavailable, since in such cases the matrices $\boldsymbol{B}$ might be singular. Nevertheless, our attack that requires a single protected pattern runs successfully by querying the oracle in such cases.

We extract the secret key successfully in all cases and have confirmed this with SFLL authors as well.

C. Expected Attack Success on Other SFLL Benchmarks

Our attack on SFLL-hd was successful as the functionality strip operation was implemented in a way that left structural traces about the ***key***. We expect our attack to be successful on any other SFLL benchmark with this type of implementation, which allows us to first identify a protected input pattern, and then extract the ***key***.

The other version of SFLL is SFLL-flex. In SFLL-flex, the designer can choose the protected patterns freely irrespective of any key and hamming distance [8]. For an attack to be fully successful on this version, there are two conditions to be met.

First, the functionality strip operation should be implemented by leaving structural traces, so that the protected patterns can be identified. We expect this to always be the case unless truly security-aware design synthesis tools are used; we argue that any post-processing (resynthesis) on the logic will leave traces, i.e., new nets that can be structurally identified (e.g., low activity), revealing information about a protected pattern.

And, second, the attack should identify *all* protected patterns. Developing an attack that satisfies the second condition is an open research problem. We expect our attack to be less successful on SFLL-flex where the protected patterns are selected judiciously.

SFLL-fault is the other version of SFLL, where logic is removed from the design by inserting a fault; the keys are the test patterns detecting the fault [27]. This defense necessitates a very different attack, as logic removal does not leave any structural traces.

VII. Conclusion

In this paper, we perform a security analysis of the latest provably secure logic locking solution SFLL-hd. We identify a weakness in the structural implementation of the functionality strip operation in the locked design and exploit it to compute some of the protected input patterns, which are normally a secret. We would like to note that the creators of SFLL-hd utilize the existing design synthesis tools to implement the functionality strip operation. However, current design tools/flow are all security-oblivious; SFLL-hd too suffers from this, leaving traces in the circuit about the key (visible to reverse engineers). They rely on the transformations of the logic synthesis tool hoping that these traces will be difficult to find. Until truly security-aware synthesis tools are developed, these vulnerabilities will exist, undermining the security of any methodology that relies on these tools. Our attack highlights this reality, which we think is a call for a very important line of research: development of security-aware logic synthesis tools.

Then we present two different techniques that can extract the SFLL-hd ***key*** from the identified protected input patterns. One technique is based on creating a linear system of equations from the identified k protected input patterns and solving this system to extract the ***key***. This technique does not require an oracle access. The other technique requires only one protected input pattern and queries the oracle with k manipulated versions of this pattern to extract the ***key***. Finally, we execute our attack on the SFLL-hd benchmark circuit made publicly available [8]. This is a microcontroller that consists of an ARM Cortex-M0 microprocessor, with more than 50K gates, protected by SFLL-hd with $h = 32$ and $k = 256$. We successfully recover the 256-bit ***key*** by two attacks in 76.32 seconds. We also show that our attack is consistently successful on various other SFLL-hd benchmarks as well. Furthermore, we make this attack tool available to public. We expect our attack to be successful on any design locked with SFLL-hd, and more so on designs with a larger number of protected patterns. Our future work is to develop an attack on SFLL-flex [8] and SFLL-fault [27], if possible.

References

[1] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[2] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[3] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.

[4] Y. M. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proc. 16th USENIX Secur. Symp. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, Aug. 2007, pp. 20:1–20:16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362923

[5] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 51–63, Feb. 2012.

[6] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. 22nd USENIX Conf. Secur.* Washington, DC, USA: USENIX, Aug. 2013, pp. 495–510. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson

[7] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Des., Automat. Test Eur.*, Mar. 2008, pp. 1069–1074.

[8] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Aug. 2017, pp. 1601–1618.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. DAC Des. Automat. Conf.*, Jun. 2012, pp. 83–89.

[10] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[11] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2014, pp. 270–271.

[12] 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, Jun. 2015.

[13] 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, Feb. 2010.

[14] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Proc. Des., Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2014, pp. 1–6.

[15] Y. Xie and A. Srivastava, *Mitigating SAT Attack on Logic Locking* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2016, pp. 127–146.

[16] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 236–241.

[17] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[18] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proc. Great Lakes Symp. VLSI*, May 2017, pp. 179–184.

[19] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 95–100.

[20] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, to be published.

[21] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, *Novel Bypass Attack and BDD-Based Tradeoff Analysis Against All Known Logic Locking Attacks Lecture Notes in Computer Science*. Cham, Switzerland: Springer, 2017, pp. 189–210.

[22] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, (2017). *dfx_sfll_k256_h32. Bench*. [Online]. Available: https://github.com/DfX-NYUAD/CCS17/blob/master/benchmarks/sfll_hd/df_sfll_k256_h32.bench

[23] ARM. (2013). *Cortex-M0 Processor*. [Online]. Available: https://developer.arm.com/products/processors/cortex-m/cortex-m0

[24] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Germany: Springer, 2008, pp. 337–340.

[25] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.

[26] T. Tao and V. Vu, "On random ± 1 matrices: Singularity and determinant," *Random Struct. Algorithms*, vol. 28, no. 1, pp. 1–23, Jan. 2006. doi: 10.1002/rsa.20109.

[27] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "ATPG-based cost-effective, secure logic locking," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, Apr. 2018, pp. 1–6.

Fangfei Yang was born in 1996. He is currently a computer science student with the School of Computer, Wuhan University.

Ming Tang was born in 1976. She received the Ph.D. degree from the School of Computer, Wuhan University, China, in 2007.

She has conducted teaching and research work on cryptography and information security and has specific contributions in side channel analysis on cryptographic chips. She is currently a Professor with the School of Cyber Science and Engineering, Wuhan University. Her research interests include cryptography, secure design of cryptography chips, lightweight countermeasures against side channel analysis, and systematic research on side channel analysis.

Ozgur Sinanoglu received the B.S. degrees in electrical and electronics engineering and in computer engineering from Boğaziçi University, Turkey, in 1999, and the M.S. and Ph.D. degrees in computer science and engineering from the University of California San Diego, in 2001 and 2004, respectively.

He has industry experience at TI, IBM, and Qualcomm and has been with New York University Abu Dhabi since 2010, where he is currently a Professor of electrical and computer engineering and is also the Director of the Design-for-Excellence Lab. He has given more than a dozen tutorials on hardware security and trust in leading CAD and test conferences, such as DAC, DATE, ITC, VTS, ETS, ICCD, and ISQED. His recent research in hardware security and trust is being funded by the U.S. National Science Foundation, U.S. Department of Defense, Semiconductor Research Corporation, Intel Corp, and Mubadala Technology. His research interests include design-for-test, design-for-security, and design-for-trust for VLSI circuits, where he has more than 180 conference and journal papers, and 20 issued and pending U.S. patents.

Dr. Sinanoglu received the IBM Ph.D. fellowship award twice, during his Ph.D. He was also a recipient of the Best Paper Awards at the IEEE VLSI Test Symposium 2011 and ACM Conference on Computer and Communication Security 2013. He is serving as a track/topic chair or technical program committee member in about 15 conferences and as (Guest) Associate Editor for the IEEE TIFS, IEEE TCAD, ACM JETC, IEEE TETC, Elsevier MEJ, JETTA, and IET CDT journals.