Functional Analysis Attacks on Logic Locking

Deepak Sirone Pramod Subramanyan
Indian Institute of Technology, Kanpur
{*dsirone, spramod*}@cse.iitk.ac.in

***Abstract*—This paper proposes Functional Analysis attacks on state of the art Logic Locking algorithms (FALL attacks). FALL attacks use structural and functional analyses of locked circuits to identify the locking key. In contrast to past work, FALL attacks can often (90% of successful attempts in our experiments) fully defeat locking by only analyzing the locked netlist, *without oracle access* to an activated circuit. Experiments show that FALL attacks succeed against 65 out of 80 (81%) of circuits locked using Secure Function Logic Locking (SFLL), the only combinational logic locking algorithm resilient to all known attacks.**

I. INTRODUCTION

Globalization of the semiconductor supply chain has resulted in IC design houses becoming increasingly reliant on potentially untrustworthy offshore foundries. This reliance has raised the spectre of integrated circuit (IC) piracy, unauthorized overproduction, and malicious design modifications by adversarial entities that may be part of these contract foundries [4, 7, 19]. These concerns have both financial [5] and national security implications [14].

A potential solution to these problems is logic locking [1, 11]: a set of techniques that introduce additional logic and new inputs to a digital circuit in order to create a "locked" version of it. The locked circuit operates correctly if and only if the new inputs – called "key inputs" and typically connected to a tamper-proof memory – are set to the right values. The circuit is activated by programming the correct key values after manufacturing and prior to sale. The security assumption underlying logic locking is that the untrusted foundry does not know the correct key inputs *and cannot compute it*.

Initial proposals for logic locking did not satisfy this assumption and were vulnerable to attack [8, 9, 17, 23, 26]. Rajendran et al. [9] used automatic test pattern generation (ATPG) algorithms to reveal the key bits. Subramanyan et al. [17] developed the SAT attack which defeated all combinational logic encryption algorithms known at the time. The attack uses a Boolean satisfiability solver to iteratively find inputs that distinguish between equivalence classes of keys. For each such input, an activated IC (perhaps purchased from the market) is queried for the correct output. This is fed back to the SAT solver to compute the next distinguishing input. The practicality of this attack depends on the number of equivalence classes of keys present in the locked circuit.

Much subsequent work has focused on SAT-attack resilient logic locking [20, 21, 24, 27, 28] that ensures the number of equivalence classes of keys is exponential in the key length. Broadly speaking, these proposals all share the structure shown in Figure 1. They introduce a circuit which "flips" the output of the original circuit for a particular cube or set of cubes. We refer to this component as the *cube stripping unit*. This flipped output is then inverted by a key-dependent circuit that we refer to as the *progammable functionality restoration unit*. This latter circuit is guaranteed to have an exponential number of equivalence classes of keys and ensures SAT attack resilience. (Note these methods "hard code" the locking key in the cube stripping unit which leads to a vulnerability that we exploit.) Initial proposals along these lines were Anti-SAT [20, 21] and SARLock [24]. However, Anti-SAT was vulnerable to the signal probability skew (SPS) [24] attack while SARLock was vulnerable to the Double DIP [13] attack and the Approximate SAT [12] attack. Both schemes are vulnerable to removal and bypass attacks [22, 25]. Subsequently, Yasin et al. proposed TTLock [28] and Secure Function Logic Locking (SFLL) [27]. To the best of our knowledge, SFLL is the only combinational logic locking scheme resilient to all of the above attacks.

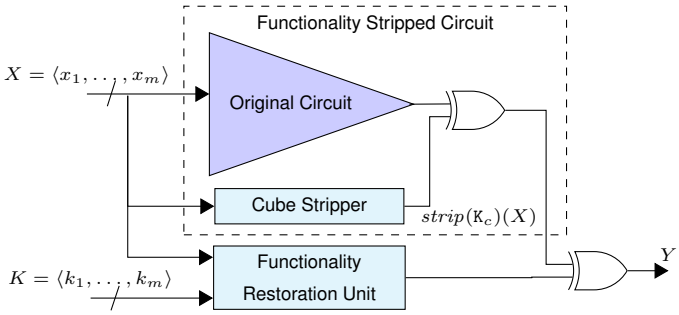

Fig. 1: Overview of SAT attack resilient locking algorithms. Note: we show a single-output circuit for simplicity.

A. Contributions

In this paper, we introduce a novel class of attacks: Functional Analysis attacks on Logic Locking, abbreviated FALL attacks. FALL attacks defeat locking methods which use cube stripping and programmable functionality restoration.

Our approach is to identify sub-circuits corresponding to the cube stripping module, and then extract the key using functional analysis of these nodes. This is challenging for two reasons. First, testing whether a sub-circuit is equivalent to the cube stripping function for some key value is a Quantified Boolean Formula (QBF) instance. Note QBF is PSPACE-complete and so a naïve approach based on encoding to QBF does not scale. We tackle this by introducing a set of poly-time structural analyses that identify the cube stripping unit.

The second challenge is extracting the key from the cube stripping unit. For this, we prove novel Boolean functional properties of the cube stripping function for TTLock and SFLL and introduce SAT-based analyses that determine locking keys using these properties. These structural and functional analyses defeat SFLL, which to the best of our knowledge, is the only combinational locking method resilient to all known attacks.

We present a thorough experimental analysis of FALL attacks. The attacks succeed on 65 out of 80 benchmark circuits (81%) in our evaluation. Among these 65, our attacks shortlist exactly one key for 58 circuits (90% of successful attempts). This shows FALL attacks are extremely practical to carry out as they do not even require Oracle access to an unlocked IC.

II. PRELIMINARIES

This section describes the adversary model for the FALL attack and the notation used in the rest of this paper.

A. Adversary Model

Our adversary is a malicious foundry with layout and mask information. The gate level netlist can be reverse engineered from this [18]. The adversary knows the locking algorithm, associated parameters and can distinguish between key inputs and circuit inputs. We follow [9, 17, 27] etc. and restrict our attention to combinational circuits.[1] The above assumptions are standard and in both logic locking defenses and attacks [9–13, 17, 20, 24, 27, 28]. We depart from past work in one important way. We do *not* assume that the adversary has access to an activated circuit which can be used to observe the output for a specific input, ruling out Oracle-guided attacks [9, 12, 13, 17]. This makes FALL attacks more practical than past work while also being more successful on SAT-resistant locking schemes.

B. Notation

$\mathbb{B} = \{0, 1\}$ is the Boolean domain. A combinational logic circuit is modeled as a directed acyclic graph $G = (V, E)$. Nodes in the graph correspond to logic gates and input nodes. Edge $(v_1, v_2) \in E$ if v_2 is a fanin (input) of the gate v_1.

Given a node $v \in V$, define $\mathit{fanins}(v) = \{v' \mid (v, v') \in E\}$. $\#\mathit{fanins}(v)$ is the cardinality of $\mathit{fanins}(v)$. For $v \in V$ such that $\#\mathit{fanins}(v) = n$, $\mathit{nodefn}(v)$ is the n-ary Boolean function associated with the node; $\mathit{nodefn}(v) : V \rightarrow (\mathbb{B}^n \rightarrow \mathbb{B})$. For example, if v_1 is a 2-input AND gate, $\mathit{nodefn}(v_1) = \lambda ab.\ a \wedge b$. For input nodes, $\mathit{nodefn}(v)$ is an uninterpreted 0-ary Boolean function (or equivalently a propositional variable). The circuit function of node v, $\mathit{cktfn}(v)$ is defined recursively as: $\mathit{cktfn}(v) = \mathit{nodefn}(v)\big(\mathit{cktfn}(v_1), \ldots, \mathit{cktfn}(v_n)\big)$ where $v_i \in \mathit{fanins}(v)$. The transitive fanin cone of a node v, $\textsc{Tfc}(v)$, is the set of all nodes v_j such that $(v, v_j) \in E$ or there exists some $v_i \in V$ such that $(v_i, v_j) \in E$ and $v_i \in \textsc{Tfc}(v)$. The support of a node, denoted by $\mathit{Supp}(v)$, is the set of all nodes v_j such that $v_j \in \textsc{Tfc}(v)$ and $\#\mathit{fanins}(v_j) = 0$. In a locked netlist, define the predicate $\mathit{isKey}(v)$ such that $\mathit{isKey}(v) = 1$ iff node v is a key input. Given two bit vectors $X^1 = \langle x_1^1, \ldots, x_m^1 \rangle$ and $X^2 = \langle x_1^2, \ldots, x_m^2 \rangle$, define $HD(X^1, X^2) \doteq \sum_{i=1}^{m} (x_i^1 \oplus x_i^2)$ to be their Hamming distance ($\oplus$ is the eXclusive OR operator). Given a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$, the function obtained by setting $x_i = 1$ in f, $f(x_1, \ldots, 1, \ldots, x_n)$, denoted as f_{x_i} is called a positive cofactor of f. $f(x_1, \ldots, 0, \ldots, x_n)$ denoted $f_{\neg x_i}$, is a negative cofactor of f.

[1] Sequential circuits can viewed as combinational by treating flip-flop inputs and outputs as combinational outputs and inputs respectively.

III. FUNCTIONAL ANALYSIS ATTACKS

This section describes FALL attacks. We first describe structural analyses to identify nodes that may be the output of the cube stripping unit. We then develop functional properties of the cube stripping function used in SFLL and develop algorithms based on these attacks. Due to a lack of space, the descriptions below are terse. More detailed explanations can be found in the associated technical report [16].

A. Identifying the Cube Stripping Unit

The first step in identifying the cube stripping unit is comparator identification. This finds all nodes in the circuit which are the result of comparing an input value with a key input. These nodes are likely to be part of the functionality restoration unit. This is done by identifying all gates v_i whose circuit function is equivalent to $(z \oplus x_i) \iff k_i$ for some z. Here x_i is a circuit input, k_i must be a key input and z captures whether k_i is being compared with x_i or $\neg x_i$. The result of comparator identification is the set $\mathit{Comp} = \{\langle v_i, x_i, k_i \rangle, \ldots\}$ where each tuple $\langle v_i, x_i, k_i \rangle$ is such that $\mathit{Supp}(v_i) = \{x_i, k_i\}$, $\mathit{isKey}(x_i) = 0$, $\mathit{isKey}(k_i) = 1$, and either $\mathit{cktfn}(v_i) \iff x_i \oplus k_i$ or $\mathit{cktfn}(v_i) \iff \neg(x_i \oplus k_i)$.

The set of all input nodes x_i that appear in Comp should be the support of the cube stripping unit. Given the set $\mathit{Comp} = \{\langle v_i, x_i, k_i, \rangle, \ldots\}$, define the projection Comp_x as $\mathit{Comp}_x = \{x_i \mid (v_i, x_i, k_i) \in \mathit{Comp}\}$. The set Cand is set of all gates whose support is identical to Comp_x: $\mathit{Cand} = \{u_i \mid \mathit{Supp}(u_i) = \mathit{Comp}_x\}$. This set of gates contains the output of the cube stripping unit.

B. Functional Properties of Cube Stripping

Cube stripping involves the choice of a protected cube, represented by the tuple $\mathtt{K}_c = \langle \mathtt{k}_1, \ldots, \mathtt{k}_m \rangle$ where $m = |\mathit{Comp}|$ and $\mathtt{k}_i \in \mathbb{B}$. A stripping function $\mathit{strip} : \mathbb{B}^m \rightarrow (\mathbb{B}^m \rightarrow \mathbb{B})$ is parameterized by this protected cube. The output of the functionality stripped circuit (the dashed box in Figure 1) is inverted for the input $X = \langle x_1, \ldots, x_m \rangle$ when $\mathit{strip}(\mathtt{K}_c)(X) = 1$. For a given locked circuit and associated key value, the value of $\mathtt{K}_c$ is "hard-coded" into the implementation of strip. The attacker's goal is to learn $\mathtt{K}_c$.

In this paper we study functional properties of the following cube stripping function: $\mathit{strip}_h(\mathtt{K}_c)(X) \doteq HD(\mathtt{K}_c, X) = h$. strip_h flips the output for input patterns exactly Hamming distance h from the protected cube $\langle \mathtt{k}_1, \ldots, \mathtt{k}_m \rangle$. This is the cube stripping function for SFLL-HD^h and the special case of $h = 0$ corresponds to the cube stripping function for TTLock. This function has three specific properties that can be exploited to determine the value of $\mathtt{K}_c$.

1) Unateness (TTLock/SFLL-HD^0): We say that a Boolean function $f : \mathbb{B}^m \rightarrow \mathbb{B}$ is positive unate in the variable x_i if $f_{\neg x_i} \leq f_{x_i}$. We say that f is negative unate in the variable x_i if $f_{x_i} \leq f_{\neg x_i}$. f is said to be unate in x_i if it is either positive or negative unate in x_i. ($a \leq b$ is defined as $\neg a \vee b$.)

Lemma 1: The cube stripping function for TTLock/SFLL-HD^0 is unate in every variable x_i. Further, it is positive unate in x_i if $\mathtt{k}_i = 1$ and negative unate in x_i if $\mathtt{k}_i = 0$.

Let $\langle \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3 \rangle = \langle 1, 0, 1 \rangle$. $strip_0(\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3)(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge x_3$. This is positive unate in x_1 as $0 \leq \neg x_2 \wedge x_3$, and negative unate in x_2 as $0 \leq x_1 \wedge x_3$.

Algorithm ANALYZEUNATENESS: Lemma 1 leads to the following attack on SFLL-HD^0. Check if the circuit function of a node $c \in Cand$, $cktfn(c)$, is unate in all of its inputs. If so, a potential locking key $\mathtt{K}_c$ is obtained using Lemma 1. Finally, verify that the $cktfn(c) \iff strip_0(\mathtt{K}_c)(X)$ is valid.

2) Non-Overlapping Errors Property (SFLL-HD^h): In the definition of $strip_h$, let $\mathtt{K}_c = \langle \mathtt{k}_1, \ldots, \mathtt{k}_4 \rangle = \langle 1, 1, 1, 1 \rangle$ and $h = 1$. Consider the two input values $\mathtt{X}^1 = \langle 1, 1, 1, 0 \rangle$ and $\mathtt{X}^2 = \langle 0, 1, 1, 1 \rangle$. $strip_1(\mathtt{K}_c)(\mathtt{X}^1) = 1 = strip_1(\mathtt{K}_c)(\mathtt{X}^2)$. $\mathtt{X}^1$ and $\mathtt{X}^2$ are Hamming distance 2 from each other and distance 1 from $\mathtt{K}_c$. This means that the values of x_i on which the two patterns agree – x_2 and x_3 – must be equal to $\mathtt{k}_2$ and $\mathtt{k}_3$ respectively. The "errors" (bit-positions where $x_i \neq \mathtt{k}_i$) in $\mathtt{X}^1$ and $\mathtt{X}^2$ cannot overlap as they are Hamming distance $2h$ apart. Generalizing this observation leads to the following result.

Lemma 2: Suppose $\mathtt{X}^1 = \langle \mathtt{x}^1_1, \ldots, \mathtt{x}^1_m \rangle$, $\mathtt{X}^2 = \langle \mathtt{x}^2_1, \ldots, \mathtt{x}^2_m \rangle$, $\mathtt{K}_c = \langle \mathtt{k}_1, \ldots, \mathtt{k}_m \rangle$ and $strip_h(\mathtt{K}_c)(\mathtt{X}^1) = 1 = strip_h(\mathtt{K}_c)(\mathtt{X}^2)$. If $HD(\mathtt{X}^1, \mathtt{X}^2) = 2h$, then for every j such that $\mathtt{x}^1_j = \mathtt{x}^2_j$, we must have $\mathtt{x}^1_j = \mathtt{x}^2_j = \mathtt{k}_j$.

Algorithm DISTANCE2H: An attack algorithm based on Lemma 2 is applicable when $4h \leq m$; m being the number of key inputs. Given a node $c \in Cand$, we use a SAT solver to find two satisfying assignments of $cktfn(c)$ that are Hamming distance $2h$ apart. This determines $m - 2h$ key bits in $\mathtt{K}_c$. Next, we constrain the $2h$ input variables which were different among these two assignments to be identical and find another pair of satisfying assignments Hamming distance $2h$ apart. This yields the remaining key bits in $\mathtt{K}_c$. The final step is checking whether $cktfn(c) \iff strip_h(\mathtt{K}_c)(X)$ is valid.

3) Sliding Window Property (SFLL-HD^h): Let us revisit the example from the non-overlapping errors property. Let $\mathtt{K}_c = \langle \mathtt{k}_1, \ldots, \mathtt{k}_4 \rangle = \langle 1, 1, 1, 1 \rangle$ and $h = 1$. For the input value $\mathtt{X}^1 = \langle 1, 1, 1, 0 \rangle$, we have $strip_1(\mathtt{K}_c)(\mathtt{X}^1) = 1$. Notice that *there cannot exist* another assignment $\mathtt{X}^2 = \langle \mathtt{x}^2_1, \ldots, \mathtt{x}^2_4 \rangle$ with $\mathtt{x}^2_4 = 0$, $HD(\mathtt{X}^1, \mathtt{X}^2) = 2$ and $strip_1(\mathtt{K}_c)(\mathtt{X}^2) = 1$. This is because $\mathtt{x}^2_4 \neq \mathtt{k}_4$, so the remaining bits in $\mathtt{X}^2$ must be equal to $\mathtt{K}_c$ so that $strip_1(\mathtt{K}_c)(\mathtt{X}^2) = 1$. But this forces the Hamming distance between $\mathtt{X}^1$ and $\mathtt{X}^2$ to be 0 (and not 2 as desired). This observation leads to the following result.

Lemma 3: Let $\mathtt{X}^1 = \langle \mathtt{x}^1_1, \ldots, \mathtt{x}^1_m \rangle$, $\mathtt{X}^2 = \langle \mathtt{x}^2_1, \ldots, \mathtt{x}^2_m \rangle$, and $\mathtt{K}_c = \langle \mathtt{k}_1, \ldots, \mathtt{k}_m \rangle$. $strip_h(\mathtt{K}_c)(\mathtt{X}^1) \wedge strip_h(\mathtt{K}_c)(\mathtt{X}^2) \wedge HD(\mathtt{X}^1, \mathtt{X}^2) = 2h \wedge \mathtt{x}^1_j = \mathtt{x}^2_j \wedge \mathtt{x}^1_j = \mathtt{b}$ is satisfiable iff $\mathtt{b} = \mathtt{k}_j$.

Algorithm SLIDINGWINDOW: Lemmas 2 and 3 lead to an attack on SFLL-HD^h for $h < \lfloor m/2 \rfloor$ where m is the number of key inputs. Given a node $c \in Cand$, we use a SAT solver to find two assignments $\mathtt{X}^1$ and $\mathtt{X}^2$ to $cktfn(c)$ which are Hamming distance $2h$ apart. The indices j for which $\mathtt{x}^1_j = \mathtt{x}^2_j$ are also equal to $\mathtt{k}_j$ by Lemma 2. The remaining bits in $\mathtt{K}_c$ are determined by iterated application of Lemma 3.

IV. EVALUATION

This section describes our experimental evaluation of FALL attacks. We describe the evaluation methodology, then present the results of the functional analyses, after which we present our evaluation of the key confirmation attack.

A. Methodology

ckt	#in	#out	#keys	# of gates		
				Original	SFLL	
					min	max
ex1010	10	10	10	2754	2783	2899
apex4	10	19	10	2886	2938	3058
c1908	33	25	33	414	1322	1376
c432	36	7	36	209	1119	1155
apex2	39	3	39	345	1367	1407
c1355	41	32	41	504	1729	1746
seq	41	35	41	1964	3177	3187
c499	41	32	41	400	1729	1750
k2	46	45	46	1474	2890	2903
c3540	50	22	50	1038	2591	2595
c880	60	26	60	327	2338	2368
dalu	75	16	64	1202	3284	3312
i9	88	63	64	591	2981	3015
i8	133	81	64	1725	3609	3637
c5315	178	123	64	1773	4076	4108
i4	192	6	64	246	2261	2289
i7	199	67	64	663	3038	3066
c7552	207	108	64	2074	4076	4105
c2670	233	140	64	717	2733	2775
des	256	245	64	3839	7229	7257

TABLE I: Benchmark circuits. #in, #out and #key refer to the number of inputs, outputs and keys respectively.

We evaluated FALL attacks on MCNC and ISCAS'85 benchmarks circuits, details of which are shown in Table I. We intentionally select smaller circuits to stress test the locking algorithms – a locking algorithm that only claims security for large circuits cannot be considered secure. We implemented the TTLock and SFLL locking algorithms for varying values of the Hamming distance parameter h and maximum key size of 128 bits. Locked netlists were optimized using ABC v1.01 [6] to minimize any structural bias introduced by our locking implementation. The circuit analyses were implemented in Python and use the Lingeling SAT Solver [3]. Attack source code is available at [15].

Our experiments were conducted on the CentOS Linux distribution version 7.2 running on 28-core Intel® Xeon® Platinum 8180 ("SkyLake") Server CPUs. Algorithms were run with a time limit of 1000 seconds.

B. Results

Figure 2 show the performance of FALL attacks. Four graphs are shown: the left most of which is for SFLL-HD^0 while the remaining are for SFLL-HD^h with varying values of the Hamming Distance h. For each graph, the x-axis shows execution time while the y-axis shows the number of benchmark circuits decrypted within that time.

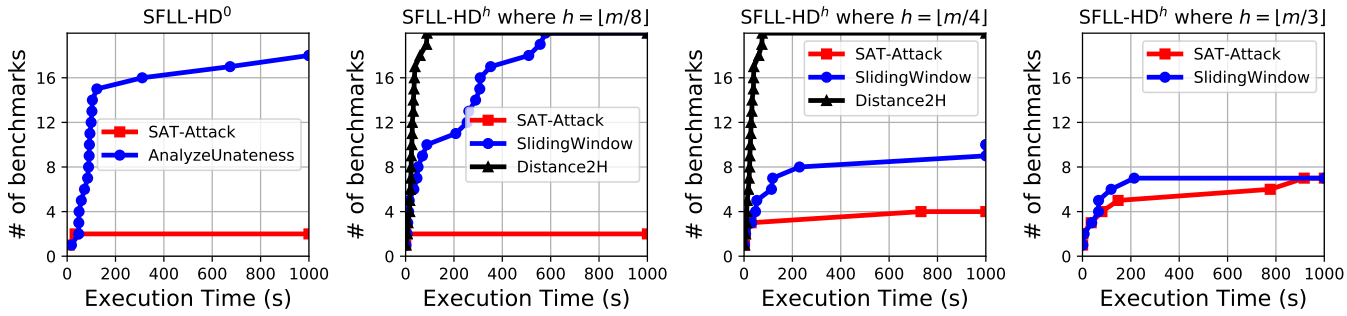

Fig. 2: Circuit analyses: execution time vs number of benchmarks solved in that time.

The **DISTANCE2H attack defeats all SFLL-HD^h locked circuits for $h = \lfloor m/8 \rfloor$ and $h = \lfloor m/4 \rfloor$**. We repeated this experiment for **the seven largest circuits with a key size of 128 bits and the DISTANCE2H attack defeated all of these locked circuits**. ANALYZEUNATENESS is able to defeat 18 out of 20 SFLL-HD^0/TTLock circuits. SLIDINGWINDOW is able to defeat all locked circuits for $h = \lfloor m/8 \rfloor$, but does not perform as well for larger values of h. This is because the SAT calls for larger values of h are computationally harder as they involve more adder gates in the Hamming Distance computation. In summary, **65 out of 80 circuits (81%) are defeated** by at least one FALL attack.

Among these 65 circuits for which the attack is successful, **a unique key is identified for 58 circuits (90%)**. This means 58 out of 80 circuits were **fully defeated without oracle access**. Among the seven circuits for which multiple keys were shortlisted, the attack shortlists two keys which are bitwise complements of each other for four circuits, three keys are shortlisted for two other circuits. One corner cases occurs for c432: 36 keys are shortlisted, this is still a huge reduction from the initial space of 2^{36} possible keys.

C. Discussion

Our results reinforce the observation that all logic locking schemes appear to be vulnerable to attack. We assert this is because the logic locking community has not adopted notions of provable security from cryptography. For instance, consider an adaptation of indistinguishability under chosen plaintext attacks (IND-CPA) [2] to logic locking. In this game, the adversary picks two different circuits and the defender encrypts of them with a random key. The adversary wins if they can determine which of the two circuits was encrypted. It is easy to see that the adversary always wins this game for SFLL-HD^h as the original circuit is largely unchanged by locking. To the best of our knowledge, the adversary would win the game described above for all logic locking schemes proposed so far. Truly secure logic locking will need the development of a methodology that can win this game.

V. CONCLUSION

This paper introduced Functional Analysis attacks on state of the art Logic Locking algorithms (FALL attacks). FALL attacks use structural and functional analyses of locked circuits to identify the locking key. Experiments showed that FALL attacks succeeded against 65 out of 80 (81%) of circuits locked using Secure Function Logic Locking (SFLL), the only combinational locking algorithm resilient to all known attacks. In contrast to past work, FALL attacks often (90% of successful attempts) fully defeat SFLL even *without* oracle access to an unlocked circuit, implying that logic locking attacks may be much easier to carry out than was previously believed.

REFERENCES

[1] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design and Test*, 27(1), Jan 2010.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. IEEE, 1997.

[3] A. Biere. Lingeling, Plingeling and Treengeling. In A. Balint, A. Belov, M. Heule, and M. Järvisalo, editors, *Proceedings of the SAT Competition*, 2013.

[4] Defense Science Board Task Force on High Performance Microchip Supply. http://www.acq.osd.mil/dsb/reports/ADA435563.pdf, 2005.

[5] IHS Technology Press Release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry. https://technology.ihs.com/405654/top-5-most-counterfeited-parts-represent-a-169-billion-potential-challenge-for-global-semiconductor-market, 2012.

[6] Alan Mishchenko. ABC: System for Sequential Logic Synthesis and Formal Verification. https://github.com/berkeley-abc/abc, 2018.

[7] M. Pecht and S. Tiku. Bogus! Electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE Spectrum*, May 2006.

[8] S.M. Plaza and I.L. Markov. Solving the third-shift problem in ic piracy with test-aware logic locking. In *IEEE Transactions on CAD of Integrated Circuits and Systems*, 2015.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security Analysis of Logic Obfuscation. In *Proceedings of the Design Automation Conference*, 2012.

[10] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault Analysis-Based Logic Encryption. *IEEE Transactions on Computers*, 64(2), Feb 2015.

[11] J. A. Roy, F. Koushanfar, and I. L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *Proceedings of Design, Automation and Test in Europe*, 2008.

[12] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin. Appsat: Approximately deobfuscating integrated circuits. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017.

[13] Yuanqi Shen and Hai Zhou. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.

[14] Semiconductor Industry Association: Anti-Counterfeiting Whitepaper One-Pager. http://www.semiconductors.org/clientuploads/directory/DocumentSIA/Anti%20Counterfeiting%20Task%20Force/ACTF%20Whitepaper%20Counterfeit%20One%20Pager%20Final.pdf, 2013.

[15] Deepak Sirone and Pramod Subramanyan. Fall Attacks Source. https://bitbucket.org/spramod/fall-attacks, 2018.

[16] Deepak Sirone and Pramod Subramanyan. Functional Analysis Attacks on Logic Locking. *CoRR*, abs/1810.01234, 2018.

[17] P. Subramanyan, S. Ray, and S. Malik. Evaluating the Security Logic Encryption Algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015.

[18] R. Torrance and D. James. The State-of-the-Art in IC Reverse Engineering. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, 2009.

[19] J. Villasenor and M. Tehranipoor. The Hidden Dangers of Chop-Shop Electronics. *IEEE Spectrum*, Sep 2013.

[20] Y. Xie and A. Srivastava. Mitigating SAT Attack on Logic Locking. In *International Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[21] Y. Xie and A. Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[22] X. Xu, B. Shakya, M.M Tehranipoor, and D. Forte. Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks. In *Cryptology ePrint Archive*, 2017.

[23] M. Yasin, B. Mazumdar, S.S. Ali, and Sinanoglu O. Security Analysis of Logic Encryption against the Most Effective Side-Channel Attack: DPA. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2015.

[24] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[25] M. Yasin, B. Mazumdar, O. Sinanoglu, and Rajendran J. Removal Attackson Logic Locking and Camouflaging Techniques. In *IEEE Transactions on Emerging Topics in Computing*, 2017.

[26] M. Yasin, S.M. Saeed, J. Rajendran, and O. Sinanoglu. Activation of logic encrypted chips: Pre-test or post-test? In *Design, Automation Test in Europe*., 2016.

[27] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2017.

[28] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan (JV) Rajendran. What to lock?: Functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.