Functional Reverse Engineering on SAT-Attack Resilient Logic Locking

Conference Paper · May 2019

DOI: 10.1109/ISCAS.2019.8702704

5 authors, including:

Lilas Alrahis
Khalifa University
52 PUBLICATIONS **749** CITATIONS
SEE PROFILE

Muhammad Yasin
Texas A&M University
43 PUBLICATIONS **2,687** CITATIONS
SEE PROFILE

Hani Saleh
Khalifa University
140 PUBLICATIONS **2,059** CITATIONS
SEE PROFILE

Baker Mohammad
Khalifa University
344 PUBLICATIONS **4,135** CITATIONS
SEE PROFILE

Functional Reverse Engineering on SAT-Attack Resilient Logic Locking

Lilas Alrahis[ϒ], Muhammad Yasin[†], Hani Saleh[ϒ], Baker Mohammad[ϒ] and Mahmoud Al-Qutayri[ϒ]
lilas.alrahis@ku.ac.ae, myasin@tamu.edu, hani.saleh@ku.ac.ae
ϒSystem on Chip center, Khalifa University, Abu Dhabi, U.A.E.
†Texas A&M University, TX, USA

Abstract— Logic locking is a solution that mitigates hardware security threats, such as Trojan insertion, piracy and counterfeiting. Research in this area has led to, in an iterative fashion, a series of logic locking defenses as well as attacks that circumvent these defenses by extracting the logic locking key. The most powerful attacks rely on a full access to a working chip/oracle that can be used to produce the input-output pairs utilized in recovering the secret key. A recently proposed technique Stripped Functionality Logic Locking (SFLL) provides resilience to all known attacks on combinational logic locking. In this paper, we propose a functional reverse engineering attack on SFLL: an attack that can detect the protection logic of SFLL which results in obtaining the original unlocked design with a high success rate. The restore and perturb blocks utilized by SFLL were detected with average coverage percentages of 93.95% and 85.42% respectively, proving that our attack is capable of breaking the state of the art logic locking technique.

I. Introduction

With the increasing cost of Integrated Circuit (IC) manufacturing, many leading edge design houses are outsourcing their fabrication, testing and packaging to potentially untrusted parties. Outsourcing the design makes it vulnerable to several supply chain attacks including IC piracy, counterfeiting, reverse engineering, and hardware Trojan insertion [1]. Many design-for-trust (DfTr) solutions have been developed to prevent such hardware security threats; Watermarking [2], IC metering [3], split manufacturing [4], IC camouflaging [5], and logic locking [6]–[12] are among the most popular solutions. Logic locking provides protection throughout the globalized IC supply chain and thus it is considered a comprehensive solution. Logic locking protects the functionality of the design via the insertion of additional key-controlled logic components (key gates). The newly added key gates are controlled by additional key inputs that are read from an on-chip secure memory. The key gates are used in order to lock the design during the untrusted phases of the design and manufacturing path. The correct functionality of the design is restored (unlocked) when a valid key is set on the secure memory. Logic locking can be combinational or sequential. In sequential logic locking, the Finite State Machine (FSM) of the design is modified, enforcing a correct key sequence to be entered for unlocking the design [8]. On the other hand, combinational logic locking inserts combinational key gates such as XOR/XNORs [6], [10], [11], or multiplexers (MUXes) [11], [13] to lock a design.

Earlier approaches in logic locking are based on finding the most suitable location for inserting key gates while maintaining a reasonable overhead. A few representative examples consist of random [6], fault analysis-based [11], [14], and strong logic locking [10]. These early efforts then spawned a series of attacks launched on these logic locking defenses; these attacks aim at extracting the logic locking key. A few representative examples consist of the sensitization attack [15], the hill-climbing based attack [13], and the satisfiability checking based (SAT) attack [16]. The SAT attack was able to break all the combinational locking techniques prior to its development. The SAT attack requires the attacker to have access to two components: (i) a working chip, which can be purchased from the open market and (ii) a locked design netlist, which can be obtained by reverse-engineering a chip. The Conjunctive Normal Form (CNF) of the locked netlist is generated and then fed to a SAT solver. The SAT solver tries to find a Distinguishing Input Pattern (DIP) for which at least two key combinations produce differing outputs. In order to rule out incorrect key values, the same DIP is then applied to the functional chip to get the correct output. The obtained input-output pair is then added as a constraint to the growing CNF formula in order to rule out all of the key values that are not able to generate the correct response. The SAT solver performs this procedure repeatedly until no more DIPs can be found and the valid key is returned.

The SAT attack urged the researchers to develop logic locking solutions resilient to the attack. The SAT attack iteratively solves the CNF formulation to reveal the secret key, and thus, its performance is defined by two factors: (i) the execution time for one iteration and (ii) the total number of iterations (DIPs) needed to eliminate all the wrong keys. For example, SAT Attack-Resilient Logic locking (SARLock) [17] increases the number of required DIPs exponentially by inserting a logic locking hardware that limits the effectiveness of DIPs. To achieve the same goal, Anti-SAT utilizes two complementary functions [12]. These techniques are later identified to be vulnerable to other attacks such as the Signal Probability Skew (SPS) driven removal attacks [18]. Later, Tenacious and Traceless Logic locking (TTLock) is developed in [19] to overcome the limitations of SARLock, thwarting removal attacks. In a more recent solution [9], SFLL generalizes TTLock, providing resilience against all

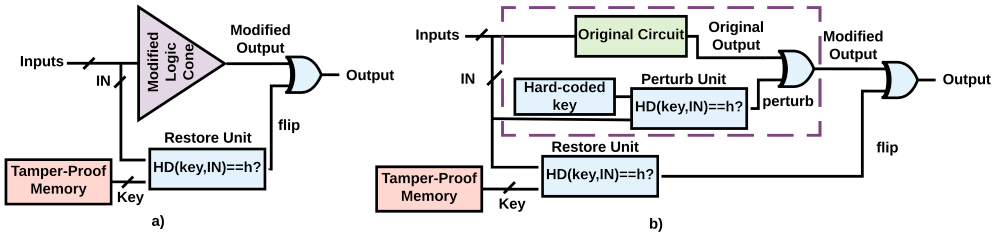

Fig. 1. a) SFLL-HD^h architecture consisting of the restore unit integrated with the modified logic cone using an XOR gate [9]. b) The modified logic cone structure (highlighted in a dotted rectangle) consisting of the perturb unit and the hard-coded secret key. The perturb unit generates the built-in errors. SFLL-HD^h expects that during synthesis the original circuit and the perturb block will be merged together.

known attacks, such as Double-DIP [20], AppSAT [21], and Bypass attack [22]. The idea is to have the hardware implementation of the design different than that of the original. The modified design is referred to as the modified logic cone. The modified logic cone protects a specific set of input patterns by generating wrong responses when they are applied. A restore unit is utilized in order to recover the correct output when a valid key is inserted.

In this paper, we propose a functional reverse engineering attack on SFLL, while we note that this attack can be launched on other SAT-attack resilient logic locking techniques such as SARLock. We focus on SFLL as it provides resilience against all known attacks on logic locking. The proposed attack requires (i) a locked design netlist and (ii) a functional reverse engineering tool. The attack methodology does not require a functional IC; although a functional IC if available may be used for verification purposes. The attack utilizes the functional reverse engineering tool in order to identify/isolate the protection logic of SFLL, thereby extracting the original design. The attack is proven to be successful against SFLL as the restore and perturb blocks utilized by the method were detected with average coverage percentages of 93.95% and 85.42% respectively.

II. Background

A. Stripped Functionality Logic Locking

SFLL [9] was proposed with a motivation to resist all known attacks on logic locking and to hide parts of the design from its hardware implementation in order to prevent removal attacks. Therefore, the design is no longer the same as the original. The original design is only restored when the correct key is entered. SFLL protects all input patterns that are of a Hamming distance h away from the secret key k_x and so it is referred to as SFLL-HD^h. The number of protected input patterns is $\binom{k}{h}$ where k refers to the key size. The architecture of SFLL-HD^h is shown in Fig. 1 a). It consists of a modified logic cone with the stripped functionality along with a restore unit (a Hamming distance checker). The modified logic cone is implemented in a way that produces built-in errors for the protected input patterns.

The modified logic cone structure is depited in Fig. 1 b). A perturb block is utilized in order to generate the built-in errors. The perturb logic is also a Hamming distance checker that generates a perturb signal when the applied input pattern N is h Hamming distance away from the "hardcoded" secret key (a protected input pattern). The perturb signal is then XORed with the original output in order to corrupt it. This output is referred to as the modified output. When a protected input pattern is applied while having the valid key set on the memory, the restore unit will generate a flip signal that is XORed with the modified output, thereby restoring the correct response. For the special case when $h = 0$, SFLL-HD^0 protects one secret input pattern which is the same as the locking key. Both the restore and perturb units can be reduced to k-bit comparators.

B. Functional Reverse Engineering

In functional reverse engineering, digital circuits are reverse-engineered utilizing a set of algorithms. A flattened and unstructured netlist can be transformed into a netlist with high level components such as adders/subtractors, comparators, register files and counters. BSIM algorithmic interface tool is an example of such technique [23]. This tool analyzes an unstructured gate level netlist using a specific set of algorithms and is capable of transforming a sea of unstructured gates into a set of high level interconnected modules. The functionality of unknown modules is identified by applying Boolean satisfiability analysis using a set of units with known functionalities. BSIM has a structural algorithm for detecting comparators. The algorithm looks for an array of bit by bit comparison structures (XNOR gates). The tool is also capable of detecting adders such as the ones used in the Hamming distance checker blocks.

III. Proposed Functional Reverse Engineering Attack on SFLL

As discussed earlier, a restore unit and a perturb unit are utilized in order to implement SFLL-HD^h. The restore/perturb unit consists of a Hamming distance checker. However, for the case of SFLL-HD^0, the restore/perturb unit can be reduced to a k-bit equality comparator. A Hamming distance checker consists of k XOR gates and a k-bit adder to compute h. The synthesis tool may retain the compartor and other blocks as is in the netlist in an attempt to minimize the power, performance, and area overhead.

In our work we propose a functional reverse engineering attack on SFLL-HD^h. The goal of the attack is to identify, isolate and remove the restore and perturb units of SFLL-HD and therefore our proposed attack can be classified among the structural/removal attacks on logic locking. Other removal attacks [18] can be launched on SFLL-HD^h, however the attacks will be able to isolate and remove the restore unit only. Therefore, the result is a modified logic cone with approx-

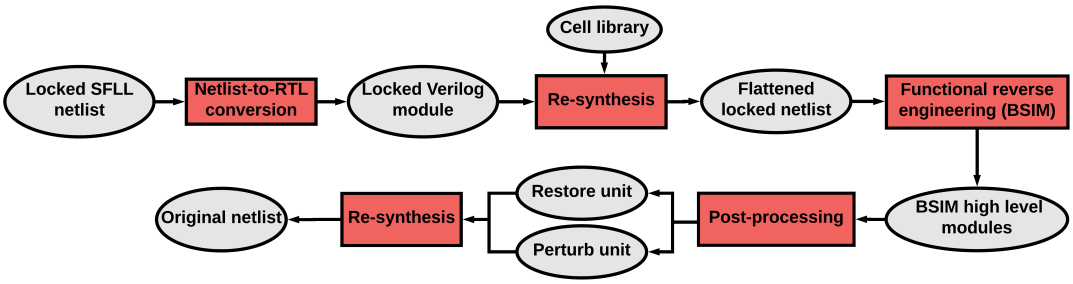

Fig. 2. The proposed functional reverse engineering attack flow.

imate functionality. In our proposed attack, both the restore and perturb units are detected and thus the original design is obtained. The proposed functional reverse engineering attack requires a locked netlist and a functional reverse engineering tool. The attack flow is presented in Fig. 2. BSIM tool is used in order to perform the functional reverse engineering. The tool is not foolproof however especially if there is optimization being performed on the comparator/adder structures. As a solution to this challenge, in our proposed attack we utilize a netlist-to-RTL parser in order to obtain the locked Verilog module from the locked optimized netlist. Then, the locked Verilog module can be re-synthesized using a smaller and more basic standard cell library. The newly obtained locked netlist is then fed to BSIM. Later on, post-processing is implemented on the obtained high level modules in order to detect which ones are related to the perturb and restore blocks. The post-processing traces the key inputs and protected inputs and checks if they are linked to any high level detected modules. The percentages of the key inputs and protected inputs linked to high level modules (forming the perturb and restore blocks) are then calculated in order to evaluate the detection coverage. The detected modules can then be removed from the locked netlist. If both units are retrieved correctly with a 100% detection rate, then the retrieved netlist is the same as the original design netlist. Otherwise, the obtained final netlist is an approximate netlist with approximate functionality.

IV. Experimental Results

A. Experimental Setup

In this section, we present the experimental results for the functional reverse engineering attack on several ISCAS-85 benchmark circuits [24] locked using SFLL-HD^h. We implemented SFLL-HD^h in a Perl framework on the three circuits with the highest number of primary inputs (in order to increase the key size), i.e., c2670, c5315, and c7552. The benchmark Verilog modules are locked using 32, 64, and 128-bit keys through SFLL-HD^h implementation; having Hamming distance values of 0, 1, and 2. We test our attack having the locked Verilog modules synthesized using Synopsys Design Compiler along with (i) ARM standard cells 65nm LPe library and with (ii) a smaller standard cell library containing only XOR, INV, BUF, NAND, and NOR gates. All the tests were performed on Ubuntu virtual machine utilizing one core of Intel $i7-3770$ CPU running at $3.40GHz$ with 10 GB of RAM.

B. Functional Reverse Engineering on SFLL

1) Case 1:

When launched on circuits locked using SFLL-HD^h and synthesized using ARM standard cells 65nm LPe library, the BSIM tool was not able to provide accurate detection. The results of this test case are shown in Fig. 3.

2) Case 2:

The locked netlists are then converted to Verilog using a netlist-to-RTL parser then are re-synthesized using a smaller standard cell library containing only XOR, INV, BUF, NAND, and NOR gates. The attack was then launched on the obtained locked netlists and the results are displayed in Fig. 4. Comparing those results with the ones presented in Fig. 3, we notice the expected jump in the detection coverage for both the restore and perturb units. For example, when $h = 0$ and $k = 32$, for case 1 the detection percentages of the restore blocks are all 0s for all three circuits and the detection percentages for the perturb blocks are again 0s except for the c7552 circuit for which it is 41.4%. For case 2 however, the restore unit detection percentages are increased to 100%, 100% and 93.75% for the c2670, c5315, and c7552 circuits. Moreover, the perturb block detection percentages are increased to 46.88%, 51.57% and 67.19%. Those results confirm the need for the netlist-to-RTL parser and the re-synthesis step in order to obtain higher detection coverage. We notice that the detection percentages for the restore unit are slightly higher of those for the perturb unit. This is expected as the perturb unit includes the hard-coded secret key. Therefore, the hardware implementation will not be the same as that of the restore unit which has external inputs only.

The execution time of the functional reverse engineering performed by BSIM for case 2 and for when $h = 1$ is presented in Table I. It is expected that when the circuit is larger, the execution time becomes higher due to the increase in the number of gates to be analyzed. Also, we notice that with the increase in the key size, the execution time increases and again this is expected as the number of gates and key inputs increases.

C. Discussion

- In our attack there is no manual intervention. However, the attacker can still be able to manually observe the detected modules and their interconnections in order to obtain higher detection percentages and be able to precisely remove both the restore and perturb units.

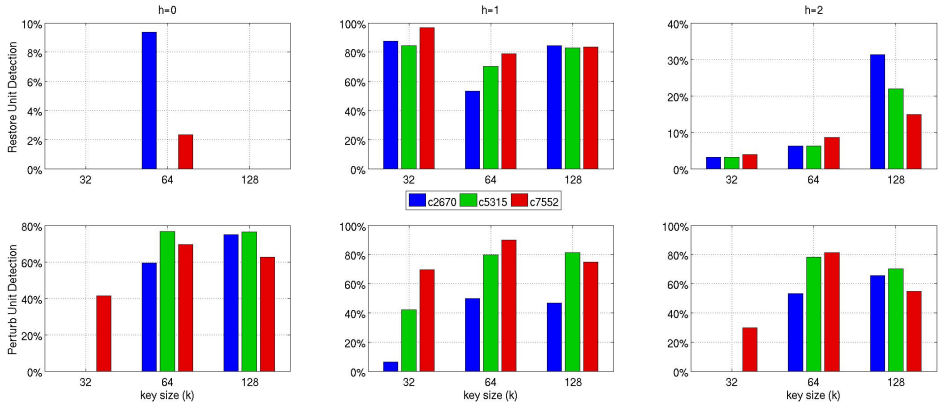

Fig. 3. Success rate of the functional reverse engineering attack on SFLL-HD^h for h={0,1,2}, and k={32,64,128} when synthesized using ARM standard cells 65nm LPe library; the detection percentage of the restore and perturb units are displayed.

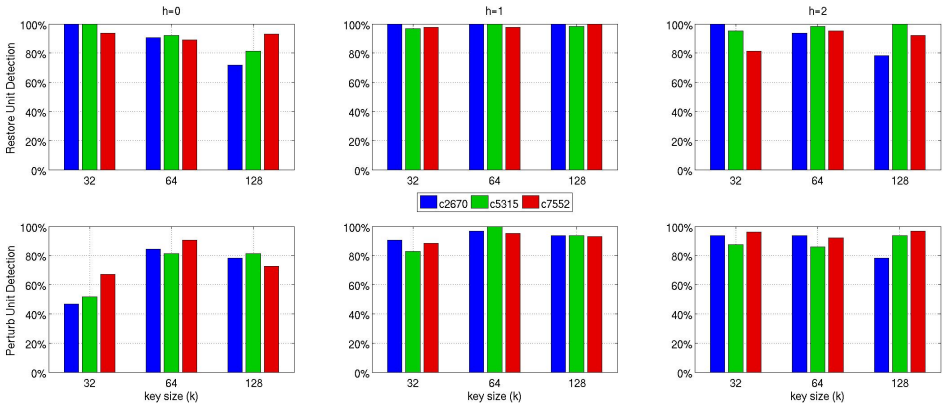

Fig. 4. Success rate of the functional reverse engineering attack on SFLL-HD^h for h={0,1,2}, and k={32,64,128} when synthesized using a basic and smaller cell library; the detection percentage of the restore and perturb units are displayed.

- Since the hardware implementation of the perturb unit depends on the hard-coded secret key, we expect the detection percentages to vary depending on the selected key. Finding an optimal key value that provides higher resilience against our functional reverse engineering attack is an open problem.
- Our attack is based on the fact that SFLL requires additional hardware (perturb unit) in order to introduce built-in errors. Our attack might fail against a recent effort [25] in which fault injection is utilized in order to modify the design. However, those techniques cannot protect a large number of patterns as SFLL-HD. Thus, developing a fully secure locking technique remains an open research problem.
- Security aware logic synthesis can be utilized in order to ensure that there are no structural traces in the locked netlist. This area has not been investigated much and remains part of our future work.

V. CONCLUSION

The SAT attack urged researchers to develop SAT-resilient logic locking techniques. SFLL technique modifies the logic cone of the circuit by utilizing a perturb unit generating built-in errors for specific input patterns. A restore unit is integrated in order to restore the correct functionality when a valid key is set. SFLL was shown resilient to all known attacks on logic locking. In this paper, we propose a functional reverse engineering attack on SFLL, detecting the perturb and restore blocks, thereby extracting the original design. The attack is evaluated by analyzing SFLL locked netlists when optimization is performed on the netlists and when it is not. We demonstrate that functional reverse engineering has high detection rate for the restore and perturb blocks for large key sizes and various Hamming distance values. The proposed attacks highlights the vulnerabilities associated with state-of-the-art logic locking and cautions against reliance on existing EDA tools in security-critical applications.

TABLE I
THE REPORTED EXECUTION TIME IN SECONDS FROM BSIM IN ORDER TO REVERSE ENGINEER THE CIRCUITS FOR CASE 2 HAVING $h = 1$.

Circuit	Key size (k)		
	32	64	128
c2670	11	11	14
c5315	23	35	37
c7552	34	39	39

REFERENCES

[1] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] A. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking Techniques for Intellectual Property Protection," in *IEEE/ACM Design Automation Conference*, 1998, pp. 776–781.

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

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

[5] J. Baukus, L. Chow, R. Cocchi, and B. Wang, "Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing," 2012, uS Patent no. 20120139582.

[6] J. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.

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

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

[9] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *ACM/SIGSAC Conference on Computer & Communications Security*, 2017, pp. 1601–1618.

[10] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.

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

[12] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016, pp. 127–146.

[13] S. Plaza and I. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[14] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic Encryption: A Fault Analysis Perspective," in *Design, Automation and Test in Europe*, 2012, pp. 953–958.

[15] ——, "Security Analysis of Logic Obfuscation," in *IEEE/ACM Design Automation Conference*, 2012, pp. 83–89.

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

[17] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2016, pp. 236–241.

[18] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 99, no. 0, p. PP, 2017.

[19] M. Yasin, A. Sengupta, B. Schafer, Y. Makris, O. Sinanoglu, and J. Rajendran, "What to Lock?: Functional and Parametric Locking," in *Great Lakes Symposium on VLSI*, 2017, pp. 351–356.

[20] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," Cryptology ePrint Archive, Report 2017/290, 2017, http://eprint.iacr.org/2017/290.

[21] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z., and Y. Jin, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *to appear in IEEE International Symposium on Hardware Oriented Security and Trust*, 2017, pp. 95–100.

[22] X. Xu, B. Shakya, M. Tehranipoor, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks," Cryptology ePrint Archive, Report 2017/621, 2017, http://eprint.iacr.org/2017/621.

[23] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse engineering digital circuits using functional analysis," *IEEE/ACM Design Automation and Test in Europe*, 2013.

[24] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

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