2019 IEEE International Test Conference in Asia (ITC-Asia)

Characterization of Locked Sequential Circuits via ATPG

Danielle Duvalsaint[1], Zeye Liu[1], Ananya Ravikumar[2] and R. D. (Shawn) Blanton[1]
[1]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
[2]Department of Electronics and Communication Engineering, PES University, Bangalore, India
Advanced Chip Test Laboratory (www.ece.cmu.edu/~actl)
{dduvalsa, zeyel, rblanton}@andrew.cmu.edu

***Abstract*—Hardware security-related threats such as the insertion of malicious circuits, overproduction, and reverse engineering are of increasing concern in the IC industry. To mitigate these threats, various design-for-trust techniques have been developed, including sequential logic locking. Sequential logic locking protects a non-scanned design by employing a key-controlled entrance FSM, key-controlled transitions, or a combination of both techniques. Current methods for characterizing (attacking) the security of sequentially locked circuits do not have the scalability to be applicable to modern circuits. In addition, current methods often require the use of an oracle, which is a working, unlocked circuit that is assumed to be fully initializable and controllable. In this work, an oracle-free, ATPG-based approach is proposed for characterizing the security of a locked sequential circuit. This method is of several in a tool box called CLIC-A (Characterization of Locked ICs via ATPG). Experiments using CLIC-A demonstrate it is effective at recovering the key sequence from various sequentially locked circuits that have been locked using different locking methods.**

***Index Terms*—Hardware Security, Logic Locking, Obfuscation**

I. Introduction

The globalization of integrated circuit (IC) design, fabrication, testing, and packaging has led to an increase in the outsourcing of IC tasks to various third parties [1]. This state of the industry has led to many security-related threats that include, for example, intellectual property (IP) piracy, insertion of hardware Trojans, overproduction, and reverse engineering. IP piracy is the illegal copying of a design [2]. This is achievable at an untrusted foundry that manufactures various customer designs. Hardware Trojans are malicious circuit modifications meant to degrade performance or leak information [3]. Overproduction of ICs and reverse engineering an IC instance [4] can both lead to the illegal sale of additional ICs, or the collection of intelligence in the latter case. In order to mitigate these attacks, various design-for-trust techniques have been developed.

Passive hardware metering [5] is a method to detect chip overproduction. This method uses slight differences in post-processing to create a unique identifier for each chip. Active hardware metering [6] uses the unique power up state of each device to create a unique key sequence that is required to place the device into a common reset state. In this method, the design does not include a reset pin so the key is needed to bring the device into the reset state. Watermarking [7] is a method to validate ICs by adding a permanent embedded signature in a design. This can be achieved by selecting a set of hardware constraints that a design must satisfy pre- and post-silicon. The constraints are selected in such a way that they can be used to identify a device without changing the functionality. To combat reverse engineering, IC camouflaging has been introduced [8]. IC camouflaging inserts cells whose functions are hidden at carefully-chosen circuit locations for the purpose of obfuscating the entire circuit function. Another design-for-trust technique is split manufacturing [9]. In split manufacturing an IC is fabricated partially in an untrusted facility and finished in a trusted facility. This technique makes it more difficult for the untrusted facility to gain access to a fully-functional design. Lastly, logic locking combats IC piracy, overbuilding and reverse engineering [2]. Logic locking can be divided into two categories, namely, sequential (non-scanned) and combinational (scanned) logic locking. Sequential logic locking [10], like active hardware metering, requires the input of a key sequence to take a circuit from its reset state and out of the locked portion of the finite state machine (FSM) to the functional/protected FSM. Combinational logic locking uses additional inputs to corrupt circuit outputs upon application of an incorrect key [2]. In an unlocked, operational circuit, the correct key is assumed to be stored in a tamper-proof memory.

In this paper, we focus on characterizing the lock strength for sequential (non-scanned) circuits using CLIC-A. CLIC-A (Characterization of Locked ICs via ATPG) is a set of ATPG-based methods for characterizing key-based locks used for securing both combinational and sequential circuits. We use the term "characterize" as opposed to "attack" because the goal is to determine if the key sequence is identifiable in a given compute time. These results can be provided to the lock designer so that the security-level specified (*e.g.*, unbreakable with 100 days of runtime) can be satisfied. The CLIC-A method invoked here is an oracle-free approach that only requires a netlist of the design. Moreover, CLIC-A does not require the key inputs be identified.

The rest of the paper is organized as follows. Background that provides context for the utility of CLIC-A is presented in Section II. Section III provides a detailed explanation of the CLIC-A method that is applicable to a locked sequential

978-1-7281-4718-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ITC-Asia.2019.00030

circuit. In Section IV, results of applying CLIC-A to different lock types are presented. In addition, an oracle-free approach for further validating the key is described and evaluated. We discuss ways to strengthen sequential locking to provide resilience to CLIC-A in Section V. Finally, Section VI concludes this paper and describes directions of our future work.

II. Related Work

A. Sequential Logic Locking

Active hardware metering [6] can be viewed as a precursor of sequential logic locking, where rather than the circuit being forced into a known start state, a specific input sequence is required to bring the circuit to a known start state. The most recent versions of sequential logic locking can be divided into two categories, ones using an entrance FSM [10], and ones intermixing functional and obfuscated states/transitions [11]. There are also locking methods that utilize both categories [12].

HARPOON (Hardware Protection through Obfuscation Of Netlist) introduces the notion of using an entrance FSM as a method for sequential, non-scan circuit protection [10]. An IC obfuscated with an entrance FSM powers up in the reset state of the entrance FSM. An entrance FSM requires a specific key-input sequence to traverse through a path of states that leads to the functional FSM. Applying an incorrect sequence will result in a transition back to the initial state, cycling around the entrance FSM, and/or trapping in a black-hole state. These three scenarios are depicted in Figure 1.

A variation on the use of an entrance FSM is the entanglement of added false states and transitions into the functional FSM [11]. In this locking scheme, the key value controls whether transitions and states are functional or non-functional. This locking method makes it more difficult to distinguish whether a state is a false state or functional state.

A combination of the two methods is proposed in [12]. This method uses an entrance FSM that has multiple paths which always lead to the functional FSM. The input sequence used to traverse a path to the functional FSM forms a key that is then used to control transitions in the functional FSM. As a result, if the entrance FSM is not traversed using the correct key sequence, the stored incorrect key causes incorrect states and transitions in the functional FSM.

B. Existing Attacks

Contrary to the vast amount of work performed in investigating the security of combinational locking methods, *e.g.*, [13-16], limited work has focused on sequential, non-scan locks. In [11], an attack on HARPOON is described where a reverse engineering tool [17] is used to create a graph of the FSM. The resulting graph is analyzed to determine which states belong to the entrance FSM. The method then uses a shortest-path algorithm to find a path from the reset state to the functional portion of the FSM. Additionally, [11] proposes a methodology where the circuit is converted into a combinational circuit through unrolling into multiple time frames. The SAT attack [14], an oracle-based attack developed to characterize the security of combinational locking, is then performed on the unrolled combinational circuit until a key sequence is found. Although these attacks are mentioned in [11], no experiment results related to uncovering a key are

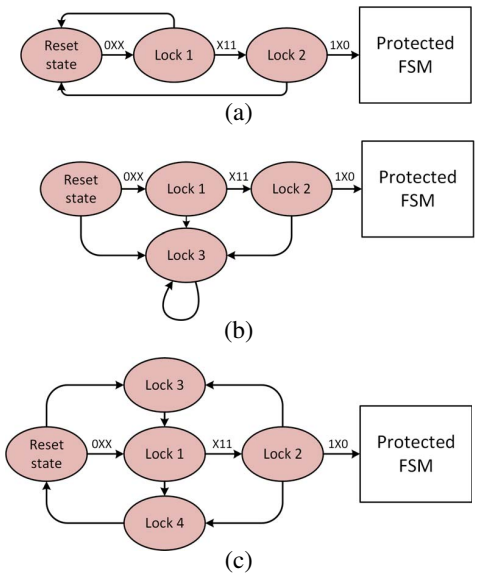


Fig. 1. Different schemes for forming an entrance FSM: (a) An entrance FSM that reverts to the reset state upon entry of an incorrect key. (b) An entrance FSM that traps upon entry of an incorrect key. (c) An entrance FSM cycles among non-functional states upon entry of an incorrect key. All entrance FSMs have the key input sequence: {0XX, X11, 1X0}.

presented. Moreover, the work of [11] is similar to the TimingSAT attack [18]. The TimingSAT attack is used to evaluate the security of delay locking [19]. This is a version of logic locking where entry of an incorrect key causes timing issues that produce incorrect functionality. In a TimingSAT attack, the circuit is unrolled N time frames and the SAT attack is performed iteratively on the locked circuit using boolean functions to capture timing information.

A significant challenge for FSM reverse engineering, employed in [11], is scalability. A modern design has a significant number of states and complex transitions, and distinguishing control from datapath is not trivial which means creating a state diagram is non-trivial. Moreover, current methods of FSM reverse engineering are not always 100% accurate when distinguishing control path from datapath, resulting in an FSM that is not an accurate representation of the design [17]. In addition, a commercial CAD tool for FSM reverse engineering is not readily available, making the method difficult to employ. Finally, for a SAT based attack on a locked sequential, non-scanned circuit, an oracle is required, and the key inputs must be known. None of these issues pose a challenge for CLIC-A.

For a locked scanned circuit, an oracle is an operational instance of the locked circuit with the correct key value applied to the key inputs. In the case of a sequential lock with an entrance FSM, an oracle is a locked non-scanned circuit that requires the correct sequence upon power up. However, in the case of a circuit locked with functional states and obfuscated

states intertwined, what constitutes an oracle is unclear. In addition, the dependence on an oracle leads to false security that threats are not present for designs not available on the open market, such as chips produced by nation states and other sensitive entities.

In this work, we describe an oracle-free, ATPG-based method within CLIC-A (Characterization of Locked Integrated Circuit via ATPG) that characterizes the security of locked unscanned, sequential circuits. ATPG is an ideal choice for this task because it is readily available from multiple commercial vendors. In addition, because of its development over the last half century, ATPG is able to handle extremely large circuits that contains 10s and even 100s of millions of gates ensuring that this characterization is applicable to real-world designs.

III. CLIC-A Methodology

The state diagrams shown in Figure 1 depict various schemes for locking a sequential, unscanned circuit with an entrance FSM. For each scheme, the lock portion of the circuit is shown shaded in red, and the protected FSM is unshaded. The insight used in CLIC-A is that there must exist numerous faults that require the key sequence for detection through sequential ATPG. This is true regardless of the logic-level implementation of the design or the type of fault analyzed. As a result, we use stuck-at faults due to their simplicity and ubiquitous nature for ATPG engines. By analyzing tests for these faults, the key sequence is extracted. Additionally, it is assumed that the only information available is the locked netlist. Lastly, CLIC-A also performs an oracle-free validation step to further build confidence that the correct key sequence has been derived.

A. ILA ATPG

Sequential circuit ATPG typically has a much higher complexity in comparison to ATPG for combinational or fully-scanned circuits. The implementation of scan chains has reduced the need to perform full sequential ATPG. However, with sequential-circuit obfuscation, scan is not implemented or is disabled so as to hinder an attacker from exploiting circuit state for deriving the key sequence using an oracle.

Sequential circuit ATPG is typically accomplished by unrolling the circuit to form a combinational-only model referred to as an ILA (iterative logic array). An ILA model mimics *N* clock cycles of behavior of a sequential circuit by making *N* copies of the combinational logic, and replacing memory elements with equivalent combinational circuits (*e.g.*, D-type flip flops are replaced with a single buffer and inverter). Moreover, the inputs to the flip-flops are added as secondary outputs to the circuit, and the outputs of the flip-flops are added as secondary inputs to the circuit. The secondary outputs of frame *i* (which corresponds to clock cycle *i*) are connected to the secondary inputs of frame *i* + 1. The secondary inputs for the first frame are held at the reset state, which must exist otherwise upon power up, the circuit could easily be operating within the protected FSM. Figure 2 illustrates an ILA model consisting of *N* frames.

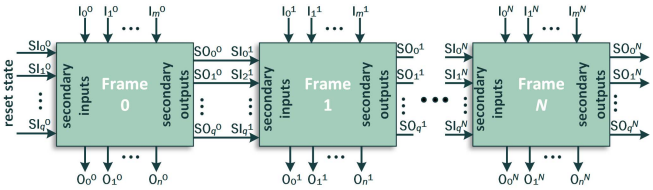


Fig. 2. An ILA model of a sequential circuit consisting of *N* frames where the first set of secondary inputs are tied to the reset state.

B. Fault Dropping

CLIC-A targets single stuck-at faults one ILA frame at a time, and drops detected faults from larger ILA models. More specifically, a fault is added or dropped based on its classification category. The fault categories considered include:

- *Detected (DT)*: fault is activated and its fault effect is propagated to at least one primary output.
- *ATPG Untestable (AU)*: fault is untestable due to constraints placed on ATPG.
- *Uncontrolled (UC)*: fault is not activated.
- *Unobserved (UO)*: fault effect is not propagated.
- *Unused (UU)*: fault location does not have a path to a primary output.
- *Redundant (RE)*: fault is deemed to be undetectable by ATPG.

Most of these categories are referenced in the fault dropping procedure described in Figure 3.

Procedure-1 begins with a user-defined guess for the number of time frames required to extract the full key sequence. The initial list of undetected faults (UD_0) includes all stuck-at faults within one frame. For each iteration through the while loop from $i = 1$ to $N + M$, all undetected faults are added to frame *i* (line 17 in Figure 3), and all unobserved faults are added to the frame they are first reported as unobserved (line 16 of Figure 3). Adding the unobserved faults in this way allows for detection of faults that may require more than one frame for error propagation. The unobserved faults (UO) and the undetected faults ($UD_{i-1} = UC_{i-1} + UU_{i-1} + AU_{i-1}$) are all targeted in the ILA model of size i (lines 15-18 of Figure 3). The results of ATPG are used to update the various fault categories (line 13). Procedure-1 terminates once the same key is derived for *M* consecutive iterations or $N + M$ is reached, where M is a user defined integer. This *M*-frame buffer builds confidence that the key found using the key derivation procedure (explained in the next section) is, in-fact, the correct key sequence. The complexity of the fault dropping procedure is O(*N*), where *N* is the number of frames, and the complexity of ATPG is well known to be NP-complete.

C. Key Derivation

Because we are operating on the assumption that there are faults that can only be detected when the circuit state reaches the functional FSM, deriving the key sequence requires searching the generated test sequences for a repeatedly appearing sequence of unknown length. Procedure-2 of Figure 4 indicates

```
1: procedure-1 FAULT DROPPING ALGORITHM
2:     N := user-defined guess of frames needed for key
3:     M := user-defined confidence buffer
4:     DT_0 := {}
5:     UO_0 := {}
6:     UD_0 := all stuck-at faults from frame 0
7:     Create a one-frame model, ILA_0
8:     Add UD_0 faults to frame
9:     i := 1
10:    run ATPG
11:    Update fault sets DT_0, UD_0 := UU_0+UC_0+AU_0, UO_0
12:    while i ≤ N + M do
13:        DT_i := {}
14:        UO_i := {}
15:        Create an i-frame model, ILA_i
16:        for j = 0 to i, add UO_j to frame j
17:        Add UD_{i-1} faults to frame i
18:        run ATPG
19:        Update fault sets DT_i, UD_i, UO_i
20:        key_i := key_derivation_algorithm(i)
21:        if key_i == key_j for j = (i − M) to i then
22:            return key_i
23:        end if
24:    end while
25: end procedure
```

Fig. 3. Pseudo-code for fault dropping within CLIC-A.

```
1: procedure-2 KEY DERIVATION ALGORITHM
2:     all_tests := test sequences generated in Procedure-1
3:     common_sequence := []
4:         ▷ array to hold most common value(s) for each i
5:     j := 0
6:     i := i value from Procedure-1
7:     while j ≤ i do
8:         test_values, frequency := []
9:             ▷ arrays to keep track of each input sequence
10:                and corresponding count
11:        for test in all_tests do
12:            if test[j] in test_values then
13:                increment frequency for test[j]
14:            else
15:                add test[j] to test_values
16:                set frequency for test[j] to 1
17:            end if
18:        end for
19:        most_common := value(s) with max frequency
20:        add most_common to common_sequence
21:    end while
22:    final_key := []
23:    for sequence in common_sequence do
24:        if sequence in all_tests then
25:            add sequence to final_key
26:        end if
27:    end for
28:    return final_key
29: end procedure
```

Fig. 4. Pseudo-code for key derivation within CLIC-A.

that key derivation begins by partitioning each generated test sequence into an array (variable all_tests in Figure 4), where each array index *i* corresponds to the sub-sequence of primary inputs $I_0^i, I_1^i, \ldots, I_m^i$ for frame *i*. A majority vote for each index is performed across all the test sequences test_1[*i*], test_2[*i*], ..., test_*n*[*i*] to find the most frequently-occurring sub-sequence for each frame i (lines 7-18 of Figure 4). If there is more than one frequently-occurring sub-sequence for a given frame i, then all the top-occurring sequences are kept.

To narrow down the number of possible key sequences, only key sequences found within in at least one generated test sequence are kept (lines 22-28 of Figure 4). For a key sequence equal to the user-defined guess of N, it is possible that Procedure-2 returns multiple possibilities for the key. There are two reasons why this situation may occur. First, there may be multiple correct keys for traversing the entrance FSM. In this case, all are correct and can be further validated using the oracle-free approach described later in this paper. Secondly, it could be the case that CLIC-A has not yet generated a sufficient number of sequences for confidently deriving the key. This is handled however using the user-defined value M, where ATPG is performed on M additional ILA models of length $N+1, N+2, \ldots, N+M$ to generate additional test sequences that contain the correct key. This occurs since faults not yet detected will indeed require the key for detection. The complexity of the key-derivation algorithm is O(*NM*), where *N* is the number of test sequences and *M* is the length of the test sequences.

IV. EXPERIMENTS

This section describes various experiments demonstrating the effectiveness of CLIC-A on sequential locks.

A. Setup

The CLIC-A experiments described in this section are all executed on an Intel Xeon processor running at 2.20 GHz, and all ATPG runs are executed using an off-the-shelf commercial tool. For all the experiments, N and M are set to 100 and 5, respectively. Instead of selecting a value for N arbitrarily, a user could alternatively chose a time limit for key derivation. Moreover, the value of M can be modulated to control the confidence-runtime trade-off. A higher value for *M* leads to more confidence in the key, however, this increase also leads to a higher runtime due to the need for more runs of ATPG.

B. Lock Types

CLIC-A is first applied to circuits that utilize the three different entrance FSM types depicted in Figure 1. Specifically, experiments are performed on the control unit of a GPS correlator [20] using all three lock types and different key lengths. The results in Table I demonstrate that CLIC-A is effective in finding the correct key for all entrance FSM locked types and key-sequence lengths. Specifically, column one indicates the lock type, while the second column of the table shows the key length, unknown to the user before

running the characterization. The third column shows how many frames are needed to derive the key. Finally, column four provides total run time for both ATPG and key derivation.

TABLE I
CLIC-A RESULTS FOR VARIOUS LOCK TYPES APPLIED TO A GPS CORRELATOR.

Type	Key length	No. of frames	Runtime (min)
Restart lock	4	14	13.3
Restart lock	8	17	21.6
Restart lock	16	26	55.4
Trap lock	4	16	13.4
Trap lock	8	18	26.3
Trap lock	16	26	58.1
Looping lock	4	20	14.1
Looping lock	8	24	27.1
Looping lock	16	23	20.8

C. HARPOON

Here, CLIC-A is applied to all the available circuits locked by the authors of HARPOON [10]. The results in Table II demonstrate that CLIC-A correctly derives long key sequences. Use of the ILA method causes circuit size to increase each iteration, which in turn causes a runtime increase. The increase in runtime is an issue for larger circuits. However, experiments demonstrate that the increased runtime does not prevent CLIC-A from generating test sequences that reach the protected FSM.

TABLE II
CLIC-A RESULTS FOR HARPOON CIRCUITS.

Circuit (gates, flip-flops)	Key length	No. of frames	Runtime (hr)
s9234 (733, 166)	31	40	1.0
s13207 (964, 358)	31	42	5.3
s35932 (5444, 1826)	31	39	48.3
s38417 (6444, 1652)	31	36	50.3
s38584 (5987, 1273)	31	36	128.2

D. Netlist Diversity

To verify that the effectiveness of CLIC-A across different gate-level implementations, CLIC-A is applied to the same design synthesized using four different standard-cell libraries. The design being used in this experiment is the s13207 benchmark circuit locked by the authors of HARPOON [10]. The results in Table III illustrate that CLIC-A is successful at extracting a key sequence regardless of its gate-level implementation.

TABLE III
IMPACT OF DIFFERENT NETLISTS SYNTHESIZED USING DIFFERENT STANDARD CELL LIBRARIES

Circuit	No. of gates	Key length	No. of frames	Runtime (hr)
Library 1	964	31	42	5.3
Library 2	1180	31	43	2.9
Library 3	1071	31	39	7.2
Library 4	1178	31	45	6.3

E. Entangled State Locks

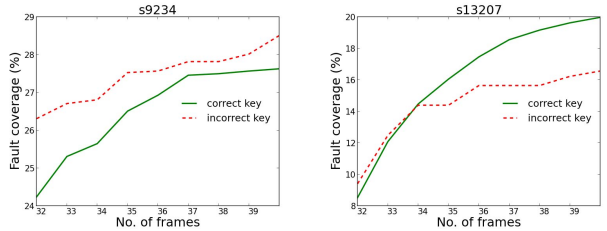
Lastly, CLIC-A is applied to circuits with entangled states [11], and a mix of entangled and entrance states [12]. Specifically, these locking methods are applied to the controller of a GPS correlator [20] and a modulo-11 counter to determine how CLIC-A performs on non-entrance FSM locks. Because these locks do not exhibit the same fault characteristics as the faults for an entrance FSM, CLIC-A is unable to extract the correct key sequence from these circuits. Thus, we conclude sequential locking methodologies using entangled states to be secure against this version of CLIC-A.

F. Key Confidence

Without the use of an oracle, and without knowing the correct key value, it is difficult to know if the found key sequence is the correct key sequence. In order to gain confidence that the key found using CLIC-A is the correct key, the fault coverage obtained with the input sequence derived by CLIC-A is compared with the fault coverage obtained with *any* other input sequence. Because it is likely that the protected FSM is significantly larger than the entrance FSM, we conclude fault coverage should be higher when the correct key sequence is applied. In order to test this hypothesis, ATPG is performed on the HARPOON circuits using two different constraints: (i) the inputs for the first 31 frames of the ILA model are constrained to the correct key sequence, and (ii) the inputs for the first 31 frames are constrained such that they can be any sequence except the correct one.

The plots in Figure 5 shows the results of this analysis. For four out of five circuits, the fault coverage is significantly lower without the correct key sequence. The one exception, s9234 which is the smallest circuit, has a larger portion of the protected functionality accessible without the correct key sequence. More specifically, 90% of the faults detected with the key sequence can be detected without the correct key sequence. For the remaining circuits the percentage is much lower, ranging between 57% and 75%.

Although this process does not guarantee the key sequence is correct, it demonstrates that ATPG does not find another input sequence that can reach the same coverage as the correct key sequence for most cases. If a key sequence found using CLIC-A exhibits this characteristic, this is strong evidence that it is the correct key sequence. However, as shown in Figure 5a, this is not always the case for the small percentage of faults detected.



(a) (b)

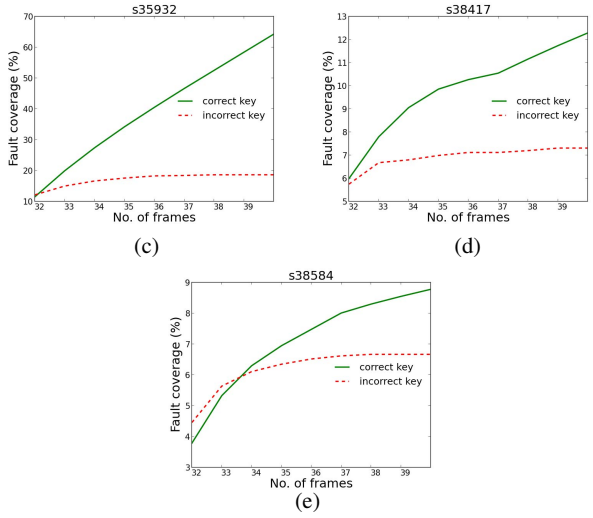


Fig. 5. Key-validation plots for five locked circuits based on HARPOON. The green solid line indicates the fault coverage with the correct key sequence applied and the red dashed line shows the fault coverage for any sequence except the correct one. Results are shown for (a) s9234, (b) s13207, (c) s35932, (d) s38412, and (e) s38584. For all circuits except the smallest (s9234), fault coverage is higher with the correct key sequence applied.

V. Strengthening Sequential Locking Methods

The results presented in Section IV demonstrate that the use of an entrance FSM alone lacks security against CLIC-A. If significant functionality is inaccessible in the entrance FSM, then a majority of tests generated for faults will require at least a partial key. One way to mitigate the effectiveness of the current version of CLIC-A is to employ the approaches described in [11,12] (Section IV-F). Another way to mitigate the effectiveness of CLIC-A would be to design the entrance FSM so that there are multiple keys, each of which unlocks different functionality. All the keys would be non-functional, except one, and appear almost equally in the test sequences. In this case, the current version of CLIC-A could not distinguish which key is correct. This mitigating approach however has a significant overhead cost due to the need for incorrect functionality per incorrect key but may be warranted depending on the application.

VI. Conclusion

The use of an entrance FSM in sequential locking is susceptible to attacks because it creates an imbalance of functionality between the locked and unlocked portions of the design, irregardless of the logic-level implementation. This property is exploitable through test sequence analysis. In this paper, we propose an oracle-free, ATPG-based key extraction method, within CLIC-A, which characterizes the security of sequential, non-scan locks. Through experiments that considered entrance FSMs, the key input sequence could always be derived from test sequences produced by ATPG. In addition, an oracle-free key confidence building approach is also included in CLIC-A that provides further confidence that the correct key sequence has been derived. In future work, we plan to expand CLIC-A to other sequential locking methods (*e.g.*, [11][12]). Additionally, we plan to investigate how parallelism can be used to significantly decrease the runtime required by CLIC-A.

Acknowledgment

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-1-7818.

References

[1] "Defense Science Board (DSB) Study on High Performance Microchip Supply," https://www.acq.osd.mil/dsb/reports/2000s/ada435563.pdf, March 2005.
[2] J. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30-38, 2010.
[3] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283-1295, 2014.
[4] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," *ACM/SIGSAC Conference on Computer and Communications Security*, pp. 709-720, 2013.
[5] F. Koushanfar, G. Qu, and M. Potkonjak, "Intellectual Property Metering," *Information Hiding Workshop*, pp. 81-95, 2001.
[6] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," *USENIX Security*, pp. 291-306, 2007.
[7] A. Kahng et al., "Watermarking Techniques for Intellectual Property Protection," *IEEE/ACM Design Automation Conference*, pp. 776-781, 1998.
[8] J. P. Baukus et al., "Camouflaging a Standard Cell Based Integrated Circuit," US Patent no. 8151235, 2012.
[9] R. Jarvis and M. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," US Patent no. 7195931, 2007.
[10] R. Chakaborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions for Computer Aided Design*, vol. 28, no. 10, pp. 1493-1502, 2009.
[11] T. Meade et al., "Revisit Sequential Logic Obfuscation: Attacks and Defenses," *IEEE International Symposium on Circuits and Systems*, pp. 1-4, 2017.
[12] A. Desai et al., "Interlocking Obfuscation for Anti-Tamper Hardware," *Cyber Security and Information Intelligence Research Workshop*, 2013.
[13] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *IEEE/ACM Design Automation Conference*, pp. 83-89, 2012.
[14] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 137-143, 2015.
[15] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, 2017.
[16] X. Xu, B. Shakya, M. 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.
[17] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, "Gate-level Netlist Reverse Engineering for Trojan Detection and Hardware Security," *IEEE International Symposium on Circuits and Systems*, pp. 1334-1337, 2016.
[18] A. Chakraborty, Y. Liu, and A. Srivastava, "TimingSAT: Timing Profile Embedded SAT Attack," *ACM International Conference on Computer-Aided Design*, 2016.
[19] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking Against IC Counterfeiting and Overproduction," *IEEE/ACM Design Automation Conference*, 2017.
[20] Massachusetts Institute of Technology, "GPS," *Common Evaluations Platform*, https://github.com/mit-ll/CEP, 2018.