Characterization of Locked Combinational Circuits via ATPG

Danielle Duvalsaint, Xiaoxiao Jin, Benjamin Niewenhuis, and R. D. (Shawn) Blanton
Advanced Chip Test Laboratory (www.ece.cmu.edu/~actl)
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
{dduvalsa, rblanton}@andrew.cmu.edu

***Abstract*—Threats to integrated circuits exist due to the outsourcing of IC design and fabrication to third parties. As a result, various design-for-trust techniques have been developed including logic locking. Current methods of characterizing the security of logic locking require multiple tools and expertise for the various locking types now in existence. In this paper, we propose an ATPG-based toolbox called CLIC-A (Characterization of Locked Integrated Circuits via ATPG) that can be used to determine the level of security effectiveness for a given instance of a locked circuit. Experiments demonstrate that CLIC-A is effective across a multitude of locking methods.**

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

I. Introduction

It is now well known that world-wide outsourcing of IC design, fabrication, testing, assembly, etc. has introduced a number of threats that did not exist when ICs were completely created in-house [1]. Among the variety of threats, the most prominent include intellectual property (IP) piracy [2], [3], malicious circuit alteration (i.e., Trojans) [4]–[6], overproduction [3], and reverse engineering [7], [8]. The threats are so significant that the US government has announced a $1.5B electronics initiative that has a major focus on hardware security [9].

To detect overproduction and piracy, passive hardware metering and watermarking have been developed. In passive hardware metering [10], [11], the process variations inherently within chips are used to created a unique identifier for each device. Watermarking [12]–[14] validates an IC by adding a permanent embedded signature within the design. This can be achieved by selecting a set of hardware constraints that a design must satisfy before and after fabrication. To prevent overproduction and piracy, split manufacturing was developed. In split manufacturing, an IC is partially fabricated in an untrusted facility and finished in a trusted facility [15]–[17]. This method increases the difficulty for an untrusted facility to gain access to a fully-functional design. To thwart reverse engineering, IC camouflaging has been introduced. IC camouflaging inserts cells with unknown functionality at carefully-chosen circuit locations to obfuscate the circuit functionality [18], [19]. Lastly, to prevent a functional version of an IC from being obtained from either overproduction, reverse engineering, or piracy, active hardware metering and logic locking can be employed. Active hardware metering [20] uses the unique power up state of a device to create a key sequence that is required to place the device into the correct start state. In this method, the key sequence is needed to bring the device into the reset state because the design does not include a reset. Logic locking adds additional circuitry to a design such that it requires a correct key to make available the correct functionality [2]. Logic locking can be divided into two categories, sequential and combinational. Sequential logic locking [21] requires an input key sequence to make available the intended functionality. Combinational logic locking uses additional key inputs to corrupt circuit functionality upon application of an incorrect key [2]. In an operational circuit, the correct key is stored on chip within a tamper-proof memory.

II. Related Work

The first method for locking combinational or full-scan circuits inserts parity gates at randomly-chosen signal lines [2]. Subsequently, fault-based logic locking ensures that incorrect key-input values cause incorrect circuit-output values [22]. The introduction of the sensitization attack [12], which is effective at uncovering key-input values from random and fault-based lock techniques, led to the introduction of strong logic locking (SLL) [23] which provides resilience to key-input sensitization. However, the introduction of the SAT attack [24] revealed that a majority of circuits locked using random, fault-based, and SLL approaches are vulnerable. The SAT attack derives Boolean expressions from a locked design to generate circuit inputs that identify sets of keys that produce different outputs. Using this information, groups of keys that produce incorrect outputs are identified by comparing output values from an oracle[1] and design simulation; such sets of keys are then ruled out as incorrect keys.

The introduction of the SAT attack led to new logic-locking methods specifically meant to mitigate the effectiveness of SAT. Cyclic logic locking [25] is a locking method that creates key controlled logical loops in a circuit to prevent SAT from creating a representation of the circuit. However, this method

[1]An oracle is a functional circuit with the correct key stored in a tamper-proof memory where inputs can be controlled and outputs can be observed.

978-1-7281-4823-6/19/$31.00 ©2019 IEEE

was broken with the creation of CycSAT [26] which adds checks to the original SAT attack to avoid entering cycles.

Additionally, the introduction of the SAT attack led to the development of SARLock (SAT Attack Resistant Logic Locking) [27] and Anti-SAT [28]. These two logic-locking techniques each add a lock sub-circuit to the targeted circuit that reduces the capability of SAT to find and rule out incorrect keys. SARLock achieves this goal by ensuring that each input pattern can only rule out at most one incorrect key. Anti-SAT uses complementary logic functions that take in both signal lines from the circuit, and key inputs to ensure that only a small subset of keys will produce a correct output for all input patterns. Because the lock sub-circuits create easily-recognizable structures that are additions to the targeted circuit, both techniques are susceptible to structural attacks [29]–[31] which remove or bypass the external lock leaving behind the unprotected circuit. Additionally, because these methods typically result in low output corruption, approximation attacks have been developed that can extract an approximate key from these locking types which results in an almost completely functional circuit, *i.e.* [32].

To mitigate the effectiveness of structural attacks, logic-locking techniques TTLock (Tenacious and Traceless Logic Locking) [33], and SFLL (Stripped Functionality Logic Locking) [34], [35] have been introduced. Both TTLock and SFLL corrupt the functionality of the target circuit, and include additional circuitry that restores functionality when the correct key is applied. The main difference between the two techniques is that TTLock protects the output of a circuit for one protected input pattern while SFLL protects the output of a circuit for N input patterns, where N is based on the hamming distance from a selected pattern in the case of SFLL-HD. For SFLL-Flex, the protected patterns are chosen by the designer. Additionally, for SFLL-fault [35], the protected patterns are determined by the set of tests that detect a particular fault. The circuit is then resynthesized with that fault value and the set of tests are used as key values to restore correct functionality. Because the targeted circuit functionality is modified, the removal attack is not effective at recovering a circuit that possesses 100% correct functionality from TTLock or any variant of SFLL. However, both TTLock and SFLL-HD have been successfully broken in various attack methodologies which aim to extract the hard-coded protected pattern from the netlist [36], [37].

One general issue with the aforementioned attacks is that they are each aimed a particular locking technique, or at most, a subset of techniques. For example, the SAT attack is generally applicable to random, fault-based, and strong logic locking approaches. More, each attack is unique with varying requirements ranging from off-the-shelf commercial capabilities (e.g., ATPG), to less available techniques such as SAT, all the way to custom approaches such as netlist analysis used in the removal attack. One consequence of this landscape is that the level of expertise needed to characterize existing and emerging logic-locking techniques is extremely high. That is, it is unlikely there are designers/engineers sufficiently skilled to apply all of the existing logic-locking attacks for the purposes of characterizing the effectiveness of multiple locked designs. Related to this point, is the fact that none of the capabilities required for the attacks (e.g., SAT) are readily available with the exception of ATPG. Finally, even if there are mitigations for these concerns, there is still the issue of scale. That is, there has been little investigation on how well the various attacks scale with the size of the targeted circuit given that most published works have experimented with small benchmark circuits [38], [39]. For example, we have applied the SAT attack to various sub-circuits of the OpenSPARC T2 [40], and in each instance SAT ran for several days before exiting without finding a single key input value.

In this work, we demonstrate that ATPG can be universally used to characterize the effectiveness (or the lack thereof) of a majority of existing logic-locking mechanisms for both combinational and sequential circuits. Characterization of Locked Integrated Circuits via ATPG (CLIC-A) is a tool box of ATPG-based circuit analysis techniques that can be mixed-and-matched in different ways to uncover key-input values. CLIC-A consists of existing ATPG-based approaches, and new approaches, some which are also oracle-free. ATPG is an ideal choice for characterizing locked logic circuits due to its its ability to handle extremely large circuits that contains 10s and even 100s of millions of gates. Moreover, modern circuit complexity involving multiple clock domains, embedded memories, various power and operating modes, etc. ensures that CLIC-A is applicable to real designs [41]. Additionally, the continuous development of ATPG over the last 50 years by the EDA industry has ensured that ATPG meets the demands of modern circuits, and consequently has established significant user expertise in the design and test community.

The rest of the paper is organized as follows. Section III details the different methods included in CLIC-A. In Section IV, each existing locking technique is described in detail and the CLIC-A methods used to characterize the effectiveness of each technique is discussed. Additionally, Section IV presents a comparison of the performance of SAT and CLIC-A on each locking technique. Section V discusses ways of strengthening current locking methods to provide resilience, and Section VI concludes the paper.

III. CLIC-A Methods

CLIC-A uses methods oriented around commercial ATPG to uncover key-input values in locked combinational and sequential circuits. There are four methods currently included in CLIC-A. The first three are discussed in this paper and the forth method, which is applicable to sequential locking, is described in [41]. The first method is key-input sensitization, the second is constraint-based ATPG, and the third is targeting key-dependent faults. The first two methods require the use of an oracle and a netlist of the locked circuit. The third method is oracle-free, meaning it only requires the netlist.

A. Key-Input Sensitization

CLIC-A includes the key-input sensitization method described in [23]. In this approach, ATPG is applied to the locked netlist to generate tests that sensitize individual key inputs to one or more primary outputs. By sensitization, we refer to the automatic test pattern generation of a test input pattern that propagates the targeted key-input value to one or more primary outputs. To target each key input, a stuck at fault is considered at each key input, one at a time, while constraining all the other key inputs to a do-not-care (X) value. Constraining the other key inputs to X, mutes their effects on circuit functionality. If a test is found under these circumstances, it is then applied to the oracle circuit. Based on which fault is used, and whether the response between the oracle and locked netlist match or mismatch, the key-input value is easily deduced. For example, when a test is generated for a single stuck-at zero fault, if the outputs of the oracle and the netlist match for that test, the key-input value is one. If there is a mismatch, the key-input value is zero. Pseudo-code for this approach is given in Figure 1.

```
1:  procedure-1 KEY-INPUT SENSITIZATION
2:      while key-input values are found do
3:          for unsolved_key_input in netlist do
4:              add stuck-at zero fault to unsolved_key_input
5:              constrain other unsolved key inputs to X
6:              test, netlist_response := run ATPG on netlist
7:              if test exists then
8:                  oracle_response := apply test to oracle
9:                  if oracle_response == netlist_response
10:                 then
11:                     value of unsolved_key_input is 1
12:                 else
13:                 then
14:                     value of unsolved_key_input is 0
15:                 end if
16:             end if
17:         end for
18:     end while
19: end procedure
```

Fig. 1. Pseudo-code for key-input sensitization within CLIC-A.

Key-input sensitization is performed iteratively until ATPG reports no test can be found. Once this occurs, and still unknown key-input values exist, another approach from CLIC-A is invoked. The purpose of sensitization is to find individual key-input values that have at least one direct path to an output without the interference of other key inputs. Having key inputs that can be sensitized is most common for locked circuits using the random [2] or fault-based [22] locking techniques.

B. Constraint-based ATPG

CLIC-A also includes a novel ATPG constraint-based characterization method. This method utilizes the constraint functions built into commercial ATPG. An ATPG constraint is a logic function applied to an ATPG run where the signal-line values included in the constraint must satisfy that function for any test generated. If a test cannot be generated that satisfies the constraints, ATPG reports a failure. For example, if the constraint is $(s_1 + \overline{s_2} + \overline{s_3})$ then every test generated adhering to that constraint must have $s_1 = 1$, $s_2 = 0$, or $s_3 = 0$.

Constraint-based ATPG within CLIC-A begins by dividing the circuit into logic cones and ordering the cones from smallest to largest by the number key inputs in that cone. Starting with the smallest cone, constraint-based ATPG is performed on each key input, one at a time. The details of this approach are described in the pseudo-code given in Figure 2. Constraint-based ATPG targets a fault at each key input to generate a test which is then applied to the oracle (lines 3-7 in Figure 2). Based on whether the outputs match or mismatch, a constraint is constructed and applied to subsequent runs of ATPG for that cone (lines 8-14 in Figure 2).

```
1:  procedure-2 CONSTRAINT-BASED ATPG
2:      for unsolved_key_input in cone do
3:          add stuck-at zero fault to unsolved_key_input
4:          add constraints to ATPG
5:          test, netlist_response := run ATPG on netlist
6:          if test exists then
7:              oracle_response := apply test to oracle
8:              if oracle_response == netlist_response
9:              then
10:                 add match constraint to constraints
11:             else
12:             then
13:                 add mismatch constraint to constraints
14:             end if
15:             reduce constraints
16:             if single key-input value in constraints
17:             then
18:                 assign single key-input value
19:             end if
20:         end if
21:     end for
22: end procedure
```

Fig. 2. Pseudo-code for constraint-based ATPG within CLIC-A.

Logic constraints are constructed using the key-input values determined by ATPG when targeting key-input faults. If the outputs between the simulated netlist and oracle do not match, this means that the generated key-input values are incorrect. This results in a constraint function where all the key-input values are inverted and disjunctively combined. The constraint ensures that ATPG creates a subsequent test with key input values that differ from the one previously generated. Constraint generation is depicted in Figure 3. However, if instead the outputs between the simulated netlist and oracle match, it does not ensure that the key-input values generated are correct since matching can serendipitously occur for a variety of reasons. Therefore, a constraint is added where all key-input values, except the targeted key input, are inverted

and disjunctively combined. This new constraint ensures that ATPG can regenerate the same set of key-input values again if it is correct, while able to generate other key-input values as well. This scenario is depicted in Figure 4.

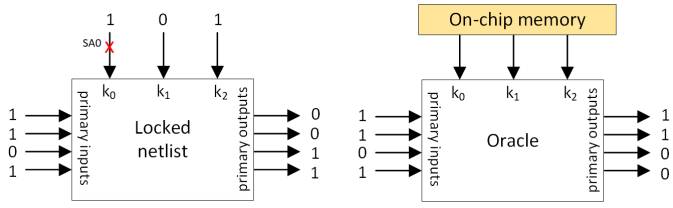

Fig. 3. A mismatch between a locked netlist and an oracle for a test generated by targeting the stuck-at zero fault k_0. The constraint created in this example is: $\tilde{}k_0 + k_1 + \tilde{}k_2$.

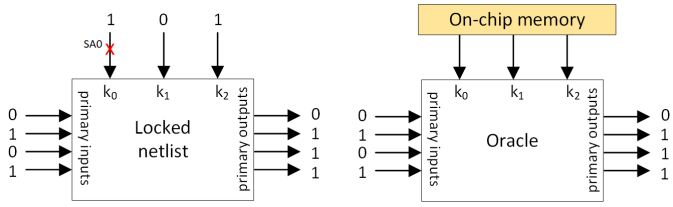

Fig. 4. A match between a locked netlist and an oracle for a test generated by targeting the stuck-at zero fault k_0. The constraint created in this example is: $k_0 + k_1 + \tilde{}k_2$.

Every newly-generated constraint is added to all the previous constraints which are reduced using a logic minimizer (line 15 in Figure 2). As more constraints are conjunctively added, the logic minimizer is able to more clearly analyze the relationships between the key inputs. Eventually, a clause consisting of just one key input may be factored out of the minimized clauses. A clause consisting of a single key input means satisfaction of that clause reveals the correct value for the corresponding key input. This process is repeated until there is no change in the constraints after iterating through every key input or until all key-input values are found. Once this stopping criterion is reached, constraint-based ATPG is performed on the next largest cone. Once all cones reach the stopping criterion, if there are still unknown key-input values, another CLIC-A method is invoked. Because this process derives key-input values based on the relationships between groups of key inputs, this CLIC-A method is most effective at solving the values of key inputs inserted using strong logic locking [23]. Additionally, this method is effective at solving keys in random [2], fault based [22], and cyclic [25] locked circuits.

C. Targeting Key-Dependent Faults

The last method included in CLIC-A targets faults that require multiple key inputs for sensitization. This novel method requires only the locked netlist without the use of an oracle. To find these faults, ATPG is first performed on the locked netlist with the unsolved key inputs all constrained to do-not-care values (X). Key-input values that have been identified from a previous CLIC-A method are constrained to their known values (lines 2-7 in Figure 5). Faults that require the key inputs for detection will be reported as an ATPG failure (line 10 in Figure 5). Each fault in this set of key-dependent faults is targeted using a second round of ATPG. The only constraints placed on ATPG in the second round is the known partial key, if that exists (lines 11-19 in Figure 5). If a test is found, the generated test is stored, along with the fault to be further analyzed. The test analysis method used for the generated tests differs depending on the lock type.

```
1:  procedure-3 TARGETING KEY-DEPENDENT FAULTS
2:      for key_input in key_inputs do
3:          if key_input value is solved then
4:              constrain key_input to solved value
5:          else
6:              constrain key_input value to X
7:          end if
8:      end for
9:      run ATPG for all faults in netlist
10:     get set of key_dependent_faults
11:     for fault in key_dependent_faults do
12:         for key_input in key_inputs do
13:             if key_input value is solved then
14:                 constrain key_input to solved value
15:             end if
16:         end for
17:         add fault to netlist location
18:         test := run ATPG on netlist
19:         if test exists then
20:             store test and fault
21:         end if
22:     end for
23: end procedure
```

Fig. 5. Pseudo-code for targeting key-dependent faults within CLIC-A.

The SAT-resistant locking methods typically have a similar structure in that there exists at least one signal line in which all or a majority of the key inputs converge upon. Such a signal line has a corresponding fault that requires all or a majority of the key inputs for detection. Analysis of the generated tests vectors locates these signal lines and extracts the correct key. This is possible because the correct key value is typically hard-coded into the circuit. The key-extraction method varies slightly between the different SAT-proof circuit types, however, they all follow this general technique. Details of this CLIC-A approach are further explained in Section IV, sub-sections E-H.

IV. EXPERIMENTS

This section gives a detailed description of which CLIC-A methods are applicable to the various lock types. Results are provided for various experiments that demonstrate the

effectiveness of CLIC-A compared to the effectiveness of the SAT attack. In this section, key-input sensitization is referred to as method one, constraint-based ATPG is referred to as method two, and targeting key-dependent faults is referred to as method three. With the exception of all but one netlist under SFLL-HD and one netlist under random locking, the experiments in this section are performed on locked benchmarks created by others. By using such benchmarks, we eliminate the possibility of creating our own locked circuits in in a way that would be beneficial to CLIC-A.

A. Setup

The experiments are performed using a machine with 64 CPU cores running at 2.20 GHz with 1,009GB of RAM. CLIC-A uses an off-the-shelf commercial ATPG tool and the results for the SAT attack stem from the publicly-available binaries reported in [24]. Additionally, the reported number of key-input values solved for both CLIC-A and SAT are all correct. Both methods either returned a correct value or no value for each key input.

B. Random Logic Locking

Random logic locking [2] inserts xor and xnor gates into a circuit at arbitrarily selected signal lines. The key-input values are typically determined as follows: a value of zero for an xor gate and a value of one for an xnor gate, unless inverters are added at the signal line where the gate is inserted. We assume that the attacker has a locked netlist, an oracle, and knows which netlist inputs are the key inputs.

Random gate insertion does not guarantee interference among key-input values. This makes random insertion particularly susceptible to the sensitization attack as demonstrated in [23]. Therefore to characterize this technique, method one is used to target key inputs which can be singularly sensitized, and method two is used to target key inputs that cannot.

Randomly-locked ISCAS85 [38] benchmarks obtained from [24] are used for experiments. Additionally, experiments are performed on the c6288 multiplier circuit obtained from Trust-Hub [42]. Lastly, experiments are also performed on the T2-NCU portion of the OpenSPARC [40] processor encrypted with all sequential elements removed and xor and xnor gates randomly inserted. Results from CLIC-A and SAT are given in Table I.

TABLE I
CLIC-A AND SAT COMPARISONS FOR RANDOM GATE INSERTION.

Circuit (gates, keys)	**SAT (keys solved, runtime)**	**CLIC-A (keys solved, runtime)**
apex2 (643, 31)	31, 4.31 seconds	31, 11.4 seconds
apex4 (5628, 268)	268, 4.31 seconds	268, 906 seconds
c3540 (1754, 83)	83, 1.41 seconds	83, 721 seconds
c6288 (2406, 128)	128, 2.3 days	128, 18 hours
OpenSparc T2-NCU (84189, 5507)	0, exits	4160, 27 hours

For smaller, low-complexity circuits, SAT outperforms ATPG in terms of runtime. For larger, more complex circuits ATPG solves circuits that SAT cannot. For example, the version of SAT from [24] cannot handle the OpenSparc T2-NCU (non-cachable unit) block, evidenced by its exit without finding any key-input values. On the other hand, ATPG identifies 75% of the key-input values. Method two terminates after identifying 75% of the key-input values because a cone with a large number of key inputs is encountered. When a large number of key bits exists in a cone, the number of constraints grows, preventing CLIC-A from performing constraint-based ATPG in a reasonable amount of time[2]. Additionally, for the multiplier circuit, c6288, ATPG identifies all the key-input values in less than half the time required by SAT. These results demonstrate that for larger, more complex circuits, ATPG is the preferred choice to characterize the security of randomly-locked circuits.

C. Fault-based Locking

Fault-based logic locking [22] selects key-gate locations based on fault testability. Specifically, faults located at the key gates must be testable which ensures an incorrect key value produces an incorrect output. This locking technique aims to achieve a 50% hamming distance between the correct outputs and incorrect outputs when a incorrect key is applied. In applying CLIC-A to fault-based locked circuits, we assume that the attacker has a locked netlist, an oracle, and knows which netlist inputs correspond to the key inputs.

CLIC-A methods one and two are employed to characterize fault-based logic locking. Both methods are dependent on faults at the input of a key gate having impact on at least one primary output. Because fault-based locking ensures this property, these methods are effective at uncovering key-input values. Locked ISCAS85 benchmarks [38] taken from [24] are use to explore the effectiveness of CLIC-A, and the results are shown in Table II. The comparative results on fault-based locking are similar to that of random locking. For smaller benchmark circuits, SAT outperforms CLIC-A in terms of compute time. However, because SAT has difficulty analyzing large and complex designs (see the last entry of Table I), CLIC-A would most likely outperform SAT with modern designs locked with fault-based locking. Unfortunately, because we do not have access to the fault-based locking method, experiments involving larger circuits locked with this approach could not be conducted.

D. Strong Logic Locking

Strong logic locking [23] performs key-gate insertion to ensure interference among multiple key gates. This insertion method results in a majority, or all of the key inputs to require at least one other key input for sensitization. We assume the attacker has a locked netlist, an oracle, and knows which netlist inputs are the key inputs.

[2]Running CLIC-A for a longer time, especially with parallelization, would identify the remaining key-bit values.

TABLE II
CLIC-A AND SAT COMPARISONS FOR FAULT-BASED GATE INSERTION.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
c432 (170, 10)	10, 0.11 seconds	10, 2.3 minutes
c499 (214, 12)	12, 0.11 seconds	12, 4.3 seconds
c880 (405, 22)	22, 0.09 seconds	22, 93.8 seconds
c1355 (546, 29)	29, 0.06 seconds	29, 6.7 seconds
c1908 (971, 91)	91, 0.08 seconds	91, 6.6 minutes

CLIC-A methods one and two are used to characterize strong logic locking. In most cases, method one is ineffective because the strength of this locking technique is that a majority of the key inputs cannot be sensitized without knowing other key-input values. Method two however is effective in deducing relationships between key inputs making this method the most suitable for identifying key-input values in strongly-locked circuits. Experiment results listed in Table III stem from analysis of strongly-locked benchmarks obtained from [24] and [42].

TABLE III
CLIC-A AND SAT COMPARISONS FOR STRONG LOGIC LOCKING.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
ex5 (1109, 53)	53, 31.9 seconds	10, 1.20 minutes
c432 (168, 8)	8, 0.06 seconds	8, 3.2 minutes
c499 (212, 10)	10, 0.1 seconds	10, 13.0 minutes
c5315 (2403, 64)	64, 1.38 seconds	14, 74.5 minutes
c7552 (3607, 64)	64, 0.84 seconds	30, 60.7 minutes

These results further demonstrate the point made in section IV-B about the efficiency of the logic minimizer. Due to the number of key inputs in the cone, the constraints could not be efficiently minimized. Currently, we consider these unsolved key-inputs to be secure against the current version of CLIC-A. However, we are working on implementing different methods to expand the abilities of method two, including parallelization.

E. Cyclic Locking

Cyclic logic locking [25] locks a circuit by adding key controlled paths into a circuit to create logical loops. In this locking scheme, the key is used as the select port in a multiplexer, where one input of the multiplexer selects the correctly functioning path in the circuit and the other selects a path that creates a feedback loop. To characterize cyclic logic locking, we assume that the attacker has a locked netlist, an oracle, and knows which netlist inputs correspond to the key inputs.

The CLIC-A methods used on cyclic locking are methods one and two. Locked ISCAS85 benchmarks [38] taken from [42] are use to explore the effectiveness of CLIC-A, and the results are shown in Table IV. Results demonstrate that CLIC-A effectively solves keys from cyclic locked circuits while SAT remains stuck in a infinite loop because it is unable to represent the circuit as a directed acyclic graph. CLIC-A can analyze cyclic circuits without oscillation because commercial ATPG tools include heuristics that detect and stabilize combinational cycles.

TABLE IV
CLIC-A AND SAT COMPARISONS FOR CYCLIC LOGIC LOCKING.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
c880 (349, 31)	0, inf. loop	31, 2.81 minutes
c3540 (793, 25)	0, inf. loop	25, 1.10 hours
c3540 (897, 56)	0, inf. loop	56, 5.20 minutes
c7552 (1428, 26)	0, inf. loop	26, 8.85 minutes
c7552 (1494, 50)	0, inf. loop	50, 8.42 minutes

F. Anti-SAT

Anti-SAT [28], depicted in Figure 6, locks a design using two complementary functions, g and $\overline{g}$, which take in two different XORed sets of key inputs and the same set of signal lines from the circuit. The outputs of these two functions are fed into an AND gate and used to invert a particular circuit output using an XOR gate. Only specific key values will make the Anti-SAT lock output a constant non-inverting value for any signal line value coming from the circuit. Correct keys for Anti-SAT circuits are where the i^{th} key input of the first key is equal to the i^{th} key input of the second key.

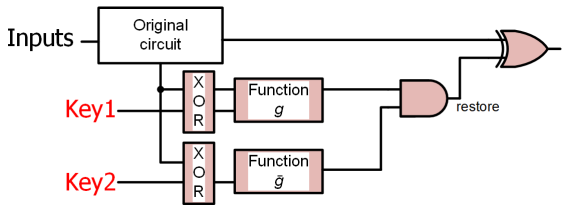

Fig. 6. Anti-SAT logic locking.

We assume that the attacker has access to the locked netlist, an oracle, and knows which netlist inputs are the key inputs. To characterize the security of Anti-SAT, all three CLIC-A methods are used. Anti-SAT is typically combined with another locking method such as strong or random locking to increase output corruption due to an incorrect key. CLIC-A methods one and two are used to identify key-input values that are not a part of the Anti-SAT lock.

Method three is used to find a key that results in the Anti-SAT lock always producing a constant value. Each fault that is reported untestable with the unsolved key inputs tied to X is detected multiple times (i.e., N-detected) with the unsolved key inputs unconstrained, and the keys generated by ATPG are stored. Because a valid key makes the Anti-SAT output a constant value, the generated key is tested with the fault value

polarity reversed. If ATPG cannot find a test for the reversed-polarity fault with the generated key, and all unsolved keys are in the fanin logic for that signal line, that key is concluded to be a viable key.

CLIC-A is tested using Anti-SAT ISCAS85 benchmarks [38] obtained from Trust-Hub [42], and the results are shown in Table V. These results demonstrate that CLIC-A is the preferred method for solving key-input values in circuits locked with the Anti-SAT method.

TABLE V
CLIC-A AND SAT COMPARISONS FOR ANTI-SAT.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
c432(332, 140)	0, 3 days	140, 4.8 minutes
c1355(747, 155)	0, 3 days	155, 2.1 hours
c5315 (2973, 566)	0, 3 days	566, 12.3 minutes
c6288(2577, 128)	0, 3 days	128, 12.7 minutes
c7552 (4337, 685)	0, 3 days	685, 13.5 minutes

G. SARLock

SARLock [27], SAT attack resistant logic locking (Figure 7), locks a design by adding external logic to the original circuit that inverts one or more outputs for specific input patterns. The logic compares a subset of the primary inputs with the key inputs and flips the output if they are equal. When the correct key is applied, the output is always the correct value.

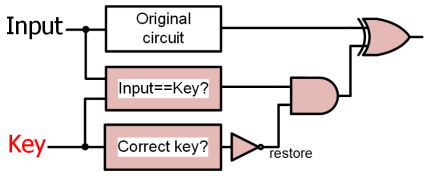

Fig. 7. SARLock logic locking.

The attacker is assumed to have access to a locked netlist and knows which netlist inputs are the key inputs. Additionally, because SARLock is typically combined with another locking mechanism such as random, fault-based, or SLL, an oracle is also required. A combination of all three CLIC-A methods are used to analyze circuits locked with SARLock.

Method three is first used to find the correct subset of key inputs that makes the output of the SARLock block a constant value of zero. The faults that are reported untestable with the key inputs constrained to X are N-detected using ATPG. Taking a majority vote over the key inputs of the resulting test patterns produces a correct partial key which deactivates the SARLock. This occurs because only one key value can produce a one at the output of the comparator between the protected inputs and the stored protected pattern. The correct key value is the value that is most likely to appear in the test vectors repeatedly. This has held true for all of the SARLock circuits we have analyzed. CLIC-A methods one and two are then applied to solve keys that are located in the logic of the circuit. Encrypted benchmarks locked by the authors of [32] are evaluated using CLIC-A and SAT and the results are shown in Table VI. The results demonstrate that CLIC-A is able to solve keys in SARLock circuits that SAT cannot.

TABLE VI
CLIC-A AND SAT COMPARISONS FOR SARLOCK.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
apex2 (739, 71)	0, 3 days	71, 41.9 seconds
ex5 (1190, 61)	0, 3 days	61, 2.64 minutes
i4 (973, 219)	0, 3 days	219, 4.3 minutes
i7 (1736, 275)	0, 3 days	275, 8.0 minutes
k2 (2048, 139)	0, 3 days	139, 2.81 minutes

H. TTLock

TTLock [33], tenacious and traceless logic locking, locks a circuit by inverting the output for one particular partial input pattern. The protected pattern is composed of a subset of the primary inputs and is chosen by the designer. This method is implemented by adding circuitry to the logic cone which determines if a subset of the primary inputs are equal to the protected pattern. When that case occurs, the output of the comparator is equal to one (signal line *flip* in Figure 8), inverting the output of the circuit. To restore the output to the correct value, additional circuitry is added which compares the key inputs to the primary inputs. If they are equal, the output of the comparator is one (signal line *restore* in Figure 8) and the output is restored. For the circuit to always exhibit correct functionality, the key must be equal to the protected pattern.

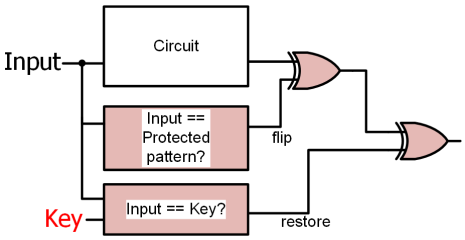

Fig. 8. TTLock logic locking.

The assumption made in this attack is that the attacker has the locked netlist. Additionally, it is assumed that the attacker knows what netlist inputs are the key inputs, but does not know what subset of primary inputs are being compared with the protected pattern. To characterize the security of TTLock, CLIC-A method three is used. Method three is applied to find which primary inputs are being compared to the protected pattern, *i.e.*, which primary inputs are the protected inputs. Because only the locked netlist is needed, this is an *oracle-free* analysis.

Method three is used to target faults that are untestable

with the key inputs constrained to X. Any fault that requires the key inputs for detection, requires the signal line $restore$ in Figure 8 to have a value other than X. For this to be true, the corresponding subset of inputs being compared with the protected pattern must be assigned in the tests generated by ATPG. Due to this property of TTLock, the subset of primary inputs used to compare with the protected pattern can be found by looking at the inputs that are always assigned in these test vectors.

Once the protected inputs are found, all faults with all of the protected inputs in their fanin are identified. Each fault in this set is then *N*-detected to obtain a set of tests for each fault. Taking a majority vote over these tests, specifically examining the values assigned for the subset of primary inputs, reveals the protected input pattern. This occurs because only one input value can produce a logical one at the output of the comparator between the protected inputs and the protected input pattern (signal line *flip* in Figure 8). The correct key-input value is the value that is most likely to appear in the test vectors repeatedly. When the protected input pattern is discovered, the correct key is identified.

CLIC-A is tested using ISCAS85 benchmarks [38] locked by the authors of [33], and the results are given in Table VII. With the exception of c7552 encoded with 32 key-bits, both CLIC-A and SAT are able to extract the correct key. In the case of c7552 with 32 key-bits, SAT was ran on the circuit for a total of 5 days without extracting a key while CLIC-A extracted the key in 1.4 minutes. Additionally, although SAT and CLIC-A can both extract keys from circuits locked with TTLock, the SAT attack requires and oracle while CLIC-A does not.

TABLE VII
CLIC-A AND SAT COMPARISONS FOR TTLOCK.

Circuit (gates, keys)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
c2670 (512, 32)	32, 10.1 seconds	32, 23.8 seconds
c5315 (1265, 64)	64, 47.9 seconds	64, 424.5 seconds
c7552 (1451, 32)	0, 5.0 days	32, 1.4 minutes
c7552 (1489, 128)	128, 6.01 seconds	128, 432.6 seconds
b18 (57362, 64)	64, 1.1 hours	64, 26.5 minutes

I. SFLL-HD

SFLL-HD [34], stripped functionality logic locking - hamming distance, works similarly to TTLock. The difference is that rather than protecting a single pattern, partial input patterns that are a hamming distance h from the protected pattern are also protected. This is depicted in Figure 9. The value of h is an integer between zero and the key length chosen by the designer. With a key length of k and a hamming distance of h, a total of $\binom{k}{h}$ patterns are protected. When h equals zero, there is one protected pattern, making SFLL-HD^0 functionally equivalent to TTLock. The assumptions made in this attack are the same as in TTLock, meaning it is also *oracle-free*.

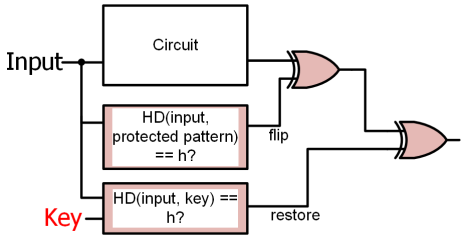

Fig. 9. SFLL-HD logic locking.

To characterize SFLL-HD, method three is applied to find which subset of primary inputs are being used in the hamming distance comparator with the protected pattern. Method three is used to target faults that are untestable with the key inputs constrained to the unknown value X. Any fault that requires the key inputs for propagation, requires the signal line $restore$ in Figure 9 to have a value other than X. As with TTLock, targeting these faults reveals the subset of primary inputs that are compared with the protected pattern.

To find the hamming distance, h, we simulate the circuit with all of the protected inputs tied to zero. The key inputs begin tied to zero and then each is flipped to one consecutively until all key inputs are set to one. By doing this, we are effectively simulating the circuit with each possible hamming distance, from 0 to k, looking for the value of h that inverts a bit in the output. The value of h that inverts an output bit is concluded to be the hamming distance of the circuit.

Similarly to TTLock, all signal lines with the protected inputs in their fanin are found and the corresponding faults associated with these signal lines are *N*-detected using ATPG. Taking all the protected inputs from the tests generated for one fault, CLIC-A uses an integer linear programming (ILP) formulation to find a pattern that is hamming distance h away from all input patterns. The ILP formulation is set up in such a way that the optimizer is tasked with minimizing a binary string with the constraint that it must be hamming distance h away from all the input patterns. If such a value is found, it is stored as a potential key. The most frequently occurring pattern is concluded to be the correct key value.

To demonstrate the effectiveness of CLIC-A on SFLL-HD circuits, testing is performed on ISCAS85 benchmark circuits [38] that we encrypted with varying values of k and h. Additionally, CLIC-A is applied to a circuit locked by the authors of [34] located at [43]. The results for this analysis are shown in Table VIII, where the circuit from [43] is in the last row of Table VIII.

The first four results in Table VIII demonstrate that for smaller key values, SAT and CLIC-A are able to solve the correct key. However, the last circuit demonstrates that for larger key values, CLIC-A identifies the correct key while SAT does not. Additionally, for the last circuit in Table VIII, CLIC-A was able to narrow down the key value to a total of 35 possible keys. We then used an oracle to determine which of these keys is the correct key.

TABLE VIII
CLIC-A AND SAT COMPARISONS FOR SFLL-HD.

Circuit (keys, HD)	SAT (keys solved, runtime)	CLIC-A (keys solved, runtime)
c880 (20, 5)	20, 8.06 seconds	20, 3.4 minutes
c1355 (32, 4)	32, 7.16 seconds	32, 4.1 minutes
c5315 (64, 19)	64, 2.04 minutes	64, 1.07 hours
l2b (128, 16)	128, 7.28 minutes	128, 1.20 days
dfx (256, 32)	0, 6.0 days	256, 3.0 days

V. STRENGTHENING COMBINATIONAL LOCKING

The introduction of the hard-coded "correct key" or hard-coded "protected pattern", as used in SARLock, TTLock, and SFLL, provide a particular point of attack that is exploited in CLIC-A. Because the key value is implemented as part of the design, when targeting faults that require those particular signal lines, the correct key/protected pattern has a high likelihood of appearing in test patterns.

The results from Anti-SAT, SAR-Lock, TTLock, and SFLL demonstrate the weaknesses with locks that function outside the circuit and are not actually entangled with the functionality. This was also demonstrated in the previous work [29]–[31], but because CLIC-A targets faults to deduce the key, rather than removing the external circuitry, CLIC-A is effective against TTLock and SFLL without losing functionality. These methods could be considered more vulnerable that the traditional locking methods because the key can be extracted without an oracle.

Additionally, strengthening the traditional locking methods (*i.e.*, random, fault-based, and strong), requires mitigating the effectiveness of method two in CLIC-A. The time required to solve key-input values using constraint-based ATPG is dependent on the number of key inputs in a logic cone. As the number of key inputs in a cone grow, so does the time it takes to identify key-input values. Therefore increasing the number of key inputs in a cone effectively mitigates this attack. However, increasing the number of key inputs in each cone does come at the expense of increasing design overhead.

VI. CONCLUSION

The various methods of combinational logic locking contain weaknesses that can be exploited using CLIC-A. Specifically, the experiments conducted demonstrate the weaknesses of traditional locking methods (*i.e.*, random, fault-based, strong). Other experiment results show the issues with using methods containing an hard-coded key value (*i.e.*, SARLock, TTLock, etc.) and how these attacks can be executed without the use of an oracle. Having a standard set of methods to evaluate combinational locked circuits via CLIC-A allows for designers to compare and contrast the security of each locking method. Future work is focused on improving the number of keys that can be solved using method two and speeding up CLIC-A with parallelism.

ACKNOWLEDGMENT

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-1-7818, and Honeywell under award number A023646.

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] K. Ramesh et al., "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *IEEE Computer*, vol. 143, no. 10, pp. 39–46, 2010.
[5] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.
[6] S. Bhunia et al., "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.
[7] R. Torrance and D. James, "The State-of-The-Art in Semiconductor Reverse Engineering," *IEEE/ACM Design Automation Conference*, 2011.
[8] P. Subramanyan, "Reverse Engineering Digital Circuits Using Functional Analysis," *Design, Automation, and Test in Europe Conference and Exhibition*, 2013.
[9] DARPA, "Electronics Resurgence Initiative," https://www.darpa.mil/news-events/2018-11-01a, 2018.
[10] F. Koushanfar, G. Qu and M. Potkonjak, "Intellectual Property Metering,"*Information Hiding Workshop*, pp. 81–95, 2001.
[11] S. Wei, A. Nahapetian and M. Potkonjak, "Robust Passive Hardware Metering," *International Conference on Computer Aided Design*, 2011.
[12] A. Kahng et al., "Watermarking Techniques for Intellectual Property Protection," *IEEE/ACM Design Automation Conference*, 1998.
[13] A. Kahng et al., "Robust IP Watermarking Methodologies for Physical Design," in *IEEE/ACM Design Automation Conference*, 1998.
[14] Y. Alkabani and F. Koushanfar, "Behavioral Synthesis Techniques for Intellectual Property Protection," in *Transactions on Design Automation of Electronic Systems*, vol. 10, no. 3, pp. 523–545, 2005.
[15] R. Jarvis and M. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," US Patent no. 7195931, 2007.
[16] M. Jagasivamani et al., "Split-Fabrication Obfuscation: Metrics and Techniques," in *International Symposium on Hardware-Oriented Security and Trust*, 2014.
[17] K. Vaidyanathan et al., "Building Trusted ICs Using Split Fabrication," in *International Symposium on Hardware-Oriented Security and Trust*, 2014.
[18] J. P. Baukus et al., "Camouflaging a Standard Cell Based Integrated Circuit," US Patent no. 8151235, 2012.
[19] J. Rajendran et al., "Circuit Camouflage Integration for Hardware IP Protection," in *IEEE/ACM Design Automation Conference*, 2014.
[20] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," *USENIX Security*, pp. 291–306, 2007
[21] R. Chakaborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer Aided Design*, vol. 28, no. 10, pp. 1493–1502, 2009.
[22] J. Rajendran et al., "Fault Analysis-Based Logic Encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2013.
[23] J. Rajendran, Y. Pino, O. Sinanoglu and R. Karri, "Security Analysis of Logic Obfuscation," in *IEEE/ACM Design Automation Conference*, 2012.
[24] 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.
[25] K. Shamsi et al., "Cyclic Obfuscation for Creating SAT-Unresolvable Circuits," in *Great Lakes Symposium on VLSI*, 2017.
[26] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based Attack on Cyclic Logic Encryptions," in *International Conference on Computer Aided Design*, pp. 49–56, 2017.

[27] 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.
[28] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016.
[29] M. Yasin et al., "Security Analysis of Anti-SAT," in *IEEE Asia and South Pacific Design Automation Conference*, 2016.
[30] M. Yasin et al., "Removal Attacks on Logic Locking and Camouflaging Techniques," To appear in: *IEEE Transactions on Emerging Topics in Computing*, 2017.
[31] X. Xu et al., "Novel Bypass Attack and BDD-based Tradeoff Analysis Against all Known Logic Locking Attacks," in *Cryptology ePrint Archive*, Report 2017/621, 2017.
[32] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," in *Great Lakes Symposium on VLSI*, 2017.
[33] M. Yasin et al., "What to Lock?: Functional and Parametric Locking," in *Great Lakes Symposium on VLSI*, 2017.
[34] M. Yasin et al., "Provably-Secure Logic Locking: From Theory To Practice," in *ACM/SIGSAC Conference on Computer Communications Security*, 2017.
[35] A. Sengupta et al., "ATPG-Based Cost-Effective, Secure Logic Locking," in *VLSI Test Symposium*, 2018.
[36] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," in *Design, Automation Test in Europe Conference*, 2019.
[37] F. Yang, M. Tang, and O. Sinanoglu, "Stripped Functionality Logic Locking with Hamming Distance-Based Restore Unit - Unlocked", in *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, 2019.
[38] F. Brglez and H. Fujiwara, "Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," *Proc. of the International Symposium on Circuits and Systems*, pp. 663–698, 1989.
[39] F. Brglez and D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. of the International Symposium of Circuits and Systems*, pp.1929–1934, 1989.
[40] "OpenSPARC T2," http://www.oracle.com/technetwork/systems/opensparc/, Oracle.
[41] D. Duvalsaint, Z. Liu, A. Ravikumar, and R. D. Blanton, "Characterization of Locked Sequential Circuits via ATPG," *International Test Conference Asia*, 2019.
[42] S. Amir et al., "Development and Evaluation of Hardware Obfuscation Benchmarks," *Journal of Hardware and Systems Security*, 2018.
[43] M. Yasin et al., "SFLL Attack Framework," https://github.com/DfX-NYUAD/CCS17, 2017.