Synthesis of Hidden State Transitions for Sequential Logic Locking

Kyle Juretus, *Student Member, IEEE*, and Ioannis Savidis, *Senior Member, IEEE*

***Abstract*—Oracle guided attacks, such as the satisfiability attack, are a significant concern when obfuscating an integrated circuit (IC). Partitioned finite state machine (FSM) based sequential logic locking techniques are much more resilient to oracle guided attacks due to the differences in the state space between the oracle and the IC under attack. However, susceptibility to structural attacks and the extraction of the transition state between the obfuscated and functional modes of an FSM threaten the efficacy of sequential logic locking. Therefore, a methodology to synthesize hidden state transitions (HSTs) into an FSM within an IC is developed. HSTs and logic cone modifications are utilized to further enhance the security of sequentially locked circuits by increasing the number of paths an adversary must search and reducing the susceptibility to structural attacks. An algorithm to insert hidden transitions and logic cone modifications into a netlist is developed that results in an average overhead of 6.79% in area, 7.78% in power, and 8.28% in performance across all of the ISCAS'89 sequential benchmark circuits. To modify the logic cone with two altered minterms, the average increase in area and power, beyond what is needed for the implementation of HSTs, is 26.46% and 30.30%, respectively, with no additional overhead in performance.**

***Index Terms*—Hardware security, hidden state transitions, sequential logic locking.**

I. Introduction

THE prevalence of third-parties within the integrated circuit (IC) design flow has drastically increased to meet time-to-market constraints and to reduce costs [1]. One of the most prominent strategies in IC production is the utilization of a commercial foundry that fabricates ICs for multiple clients [1], which is partially due to the multibillion-dollar investment required to build an advanced fabrication facility [2]. While more cost effective for a circuit design firm, the IC is vulnerable to an increased threat space as untrusted entities, including third-party intellectual property (IP) vendors, fabrication facilities, test facilities, and end-users are present in the supply chain.

Manuscript received September 9, 2019; revised January 9, 2020 and March 16, 2020; accepted April 23, 2020. Date of publication May 12, 2020; date of current version January 11, 2021. This work was supported in part by the Air Force Office of Scientific Research, National Defense Science and Engineering Graduate Fellowship, 32 CFR 168a, Drexel Ventures Innovation Fund, the Air Force Research Laboratory under Contract FA8075-14-D0025/DSTAT-15-1196, and in part by the National Science Foundation under Grant CNS-1751032. This article was recommended by Associate Editor Y. Makris. *(Corresponding author: Kyle Juretus.)*

The authors are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104 USA (e-mail: kjj39@drexel.edu; is338@drexel.edu).

Digital Object Identifier 10.1109/TCAD.2020.2994259

The wide variety of possible threats, including denial of service, theft of information, and/or corruption of functionality [3] require security solutions at the hardware level. Logic locking/encryption is one such physical level technique that secures an IC by producing incorrect functionality whenever an incorrect key is applied to the circuit [4]. By requiring a key to activate a circuit, threats that include IP theft, IC counterfeiting, IC overproduction, and hardware Trojan insertion all become increasingly difficult for an adversary to execute.

While logic locking presents an opportunity to protect against a wide threat space, oracle guided attacks have proven to be an efficient means of determining the key of the logic locked IC. The most prominent oracle guided attack on logic locked circuits is the satisfiability (SAT) attack introduced in [5], which uses a miter circuit of the locked IC to generate distinguishing input patterns (DIPs) for use on an oracle IC.

Multiple out-of-cone logic locking techniques have been developed that provide increased difficulty to an adversary executing a SAT attack by controlling the level of output corruption [6]–[9]. While the proposed out-of-cone techniques increase the difficulty of executing a SAT attack, new attack vectors such as described in [10], [11], [12], [13], and [14] have been developed to bypass much of the added security. In addition, out-of-cone techniques corrupt a limited number of the outputs of a circuit when an incorrect key is applied, which provides sufficient use of the IC for some adversaries.

Sequential logic locking is utilized in this article to reduce the threat space of an IC. As opposed to combinational logic locking, partitioned finite state machine (FSM) based sequential logic locking requires a key sequence to activate an IC. As the activated IC is in a different state space than the locked IC, oracle guided attacks present less of a threat to circuits secured with sequential logic locking.

However, attacks on the structure of a sequentially logic locked circuit or the determination of the obfuscated state transitions are possible by targeting the additional combinational logic used to secure the IC. Therefore, hidden state transitions (HSTs) are utilized in this article to reduce the leakage of key information through the combinational state transition logic. In addition, the logic cone is modified to increase the difficulty of executing an attack that exploits the combinational logic.

This article begins with an introduction to oracle guided attacks in Section II, providing both an overview of the SAT attack [5] and general principles to prevent a SAT attack. Threat models that apply to sequentially logic locked circuits are described in Section III. An overview of related

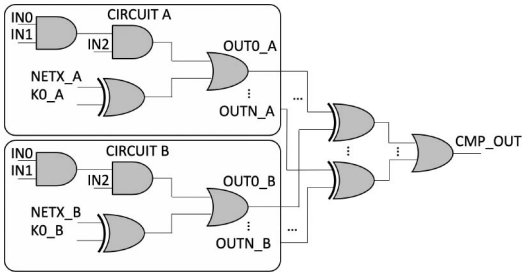


Fig. 1. Miter circuit with replicated *A* and *B* versions of a reverse engineered netlist that includes obfuscated gates. Corresponding outputs between the *A* and *B* versions of the circuit are XORed. Each XORed signal is then passed to an OR gate to check for any differentiating output.

work on sequential logic locking is provided in Section IV, describing 1) partitioned FSMs and 2) integration of incorrect state transitions into the FSM. Hidden state transitions (HSTs) are described in Section V. The algorithm that generates and integrates the HSTs into a circuit is described in Section VI. An analysis of the proposed HST methodology is performed through execution of the developed algorithm on the ISCAS'89 sequential benchmark circuits as described in Section VII. Concluding remarks are provided in Section VIII.

II. Oracle Guided Attacks

A capable adversary is able to reverse engineer an IC, which results in a locked netlist description of the circuit where the only unknown is the key of the logic locked IC. Determining the key is difficult when the adversary does not have access to an oracle circuit since correct input/output (I/O) pairs are not observed. However, with access to an oracle IC, the ability of the adversary to extract the key used for logic locking greatly increases as each obtained I/O pair significantly prunes the key space of the logic locked IC. The attack presented in [5] is an example of an efficient oracle guided attack that uses a SAT solver to generate I/O patterns that once applied to the oracle IC guarantee at least one key is removed from the search space. When the SAT attack is executed on the ISCAS'85 combinational benchmark circuits implemented with in-cone logic locking techniques, including for methods described in [4], [15], and [16], over 95% of the circuits are decrypted within 10 hours [5].

A. Overview of the SAT Attack

The efficiency of the SAT attack is dependent on the generation of DIPs from a miter circuit, with each DIP guaranteeing the elimination of at least one key from the set of all possible keys. An example of a miter circuit is provided in Fig. 1, which compares the outputs of two copies of the locked netlist, referred to as *A* and *B*. A DIP is generated whenever the output of the miter circuit is equal to one. The DIP is then applied to an oracle IC to ascertain the correct output response, which results in the determination of a correct I/O pair used to further constrain the SAT solver. The process of generating a DIP and then adding the response from the oracle to the solver as a constraint is repeated until no additional DIPs are found. Once no additional DIPs are generated, any key that satisfies the constraints is guaranteed to produce the correct circuit functionality as there are no additional inputs to the circuit that result in any differences at the outputs of the circuit due to the applied key.

B. Preventing the SAT Attack

While the SAT attack eliminates at least one key from the complete set of possible keys for each DIP, the attack is not privy to the total number of keys that are eliminated by each executed iteration of the SAT solver. Therefore, techniques such as described in [6], [7], [8], and [9] were developed to restrict the SAT attack from eliminating a large subset of keys for each DIP. The elimination of only a single key for each iteration forces the SAT attack to perform similar to a brute force attack, which results in a reduction in the efficiency of executing the SAT attack. While techniques to protect against the SAT attack exist [6]–[9], other attack vectors, including removal attacks [12], approximation attacks [10]–[11], and structural attacks [13], are capable of determining the key used for logic locking when executed.

Sequential logic locking techniques have also been developed that utilize time-based dependencies to reduce the susceptibility of an IC to a SAT-based attack. Techniques such as described in [17] split the IC into an obfuscated and functional mode. Since an activated IC only provides information regarding the functional mode operation of the circuit, a SAT-based attack is not capable of determining the activation sequence needed to transition from the obfuscated mode to the functional mode. Other techniques such as described in [18] only alter an FSM in locations that require an adversary to unroll the locked netlist of the sequential circuit many times. As each level of unrolling significantly increases the size of the model that represents the circuit and is provided to the SAT attack, the time to generate a DIP increases each iteration as well. Additional background on sequential logic locking, as well as vulnerabilities of the technique are discussed in Section IV.

III. Sequential Logic Locking Threat Models

There are multiple threat models to consider when deciding on whether to sequentially logic lock a circuit. The threats are broadly classified as either 1) unauthorized activation or 2) reverse engineering-based.

1) *Unauthorized Activation:* An adversary is able to gain access to the complete functionality of an IC without detection. Examples include IC counterfeiting and overproduction, where an adversary is able to sell or utilize the IC without the authorization of the IC designer.
2) *Reverse Engineering:* An adversary intends to steal critical IP from the IC. Under the given threat model, the adversary has access to the tools and knowledge to reverse engineer the design and obtain a gate-level netlist representation of the circuit.

Note that the threat model of reverse engineering also includes unauthorized use of an IC. Once an IC has been reverse engineered, the circuitry inserted to secure the IC is vulnerable to removal. Specifically, removal is a concern when the original circuit structure is unaltered and the modifications

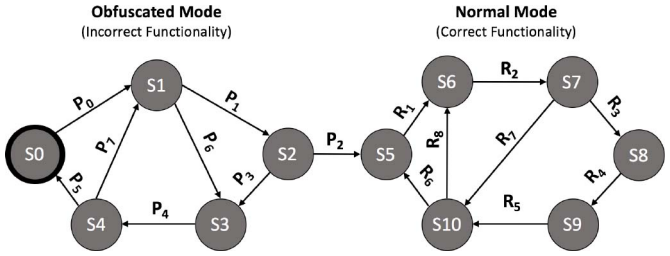


Fig. 2. Modified FSM with an enabling sequence of P0, P1, P2 required to enter normal circuit operation [17].

to the circuit implementing the security features are discernible and easily separated from the original functionality of the IC. The consideration of a given threat model, therefore, determines the required security of a circuit. The threat model considered in this article is that of an adversary attempting to reverse engineer the IC, while also assuming that the adversary has knowledge of the utilized security technique.

IV. Related Work

Sequential logic locking techniques are categorized as methods that either 1) partition the FSM or 2) produce incorrect state transitions within the original FSM. Prior work for each category is discussed in this section, as well as vulnerabilities of each approach.

A. Partitioned FSMs

An example of FSM partitioning is shown in Fig. 2, where correct operation is only achieved after the key sequence of P0, P1, and P2 is applied. The depicted partitioning is performed by applying the HARPOON sequential logic locking methodology [17], which utilizes modifying cells to alter the response of the combinational logic of the circuit when in obfuscated mode.

The resulting partitioned FSM is susceptible to vulnerabilities that include FSM extraction [19] and fault-injection attacks [20]–[22]. The vulnerability to FSM extraction is reduced as the size of the FSM increases since the number of state transitions that must be analyzed becomes prohibitively large.

Execution of a fault-injection attack without knowledge of the state transitions of the FSM is possible, as the objective is to reach the functional mode of the partitioned FSM. However, the adversary must have knowledge of the states that fall within the functional as well as the obfuscated state spaces. One functional state that an adversary is able to determine with access to an oracle IC is the starting state, which is found by resetting the IC and reading out the state in scan mode. Glitching the IC into a specific starting state from the obfuscated state space, however, becomes increasingly difficult as the number of states increases. In addition, even if an adversary is able to glitch into the starting state of the IC, the key sequence to activate the circuit has not been determined, which implies that the correct functionality of the circuit is only observable until the IC is returned to the obfuscated state after a power-off event.

When considering the threat of an unauthorized activation as described in Section III, where an adversary attempts to determine the activation sequence without knowledge of the state transition logic, fault-injection attacks are thwarted by introducing *black hole* states, which are never functionally exited, or *gray hole* states, which are difficult to functionally exit, to an FSM [23]. If an adversary glitches into a *black hole* or *gray hole* state, the IC does not function as intended. The inclusion of obfuscated and functional mode keys in a partitioned state machine is also possible, where even if an adversary is able to transfer into the functional mode, the incorrect key transitions the FSM back into the obfuscated mode [24].

The technique described in [25] prevents fault-injection attacks by requiring a code word for correct operation of the circuit when in the functional operating mode. The code is determined by the sequence applied during the obfuscated operating mode, which implies that if the correct code is not entered when bypassing the obfuscated mode of the FSM with a fault, then the functional mode does not result in proper circuit operation. When considering an adversary with full scan chain access to an oracle IC, the code word used in [25] is determined by executing the SAT attack. By determining the code word, the adversary now possesses the activation key sequence needed to reach the functional state space.

When considering an adversary that has reverse engineered the IC, the partitioned state machines require additional security against attacks that leverage the logical structure of the circuit. For example, in the case of HARPOON [17], the interconnects that modify the FSM are determined from the connections to the physically unclonable function (PUF) and the read-only memory (ROM). The ability to determine the added interconnects to the logic modifying cells provides an adversary with the information needed to remove the circuitry added to protect the IC from reverse engineering and IP theft. In addition, the determination of the state transition from the obfuscated mode to the functional mode is possible when the starting state only includes a small number of logical connections. For example, consider the example FSM shown in Fig. 2, where only $S10$ and $S2$ logically transition to the starting state $S5$. Since $S10$ and $S5$ both transition to $S6$, the only possible transition from the obfuscated mode to functional mode is $S2$ to $S5$.

B. Introducing Incorrect State Transitions Into the FSM

The techniques described in [18], [26], and [27] implement modifications to the functionality of the original FSM, which results in altered state transition logic that depends on the value of an applied key. A PUF is implemented in [27] that produces incorrect logical transitions when an incorrect key is applied, while assuming a threat model that includes remote activation. In [18], the state machine is not partitioned, but rather, incorrect state transitions are added during the normal operation of the circuit based on an applied key. The objective is to increase the number of times a logical function of the circuit must be unrolled, which increases the resources needed to execute the SAT attack [18]. The work in [26] utilizes dynamic state deflection based on an applied key vector to transition into a cluster of *black hole* states if an incorrect key is applied.

Assuming a threat model where all register values are accessible through the scan chain, all of the sequential logic locking techniques that introduce incorrect state transitions are vulnerable to SAT-based attacks. The vulnerability arises as the circuit is effectively only secured through combinational locking, with the in-cone key dependencies efficiently determined by SAT-based attacks due to the logical masking that results from the DIPs [28]. In addition, techniques described in [26] and [27] are vulnerable to structural attacks, where an adversary is able to identify the memory locations containing the key and to determine the added circuitry implementing the sequential locking. The circuitry added in [27] to secure the IC is vulnerable to removal, which allows an adversary to set the circuit to a known response. Similarly, the flip signals described in [26] are vulnerable to structural attack, and are then set to 0.

V. Creating FSM Hidden State Transitions

As discussed in Section IV, sequential logic locking techniques are vulnerable to various threats. In this article, the security of sequential logic locking is enhanced through a novel partitioned state machine methodology that 1) does not allow an adversary to observe the start-up sequence of the IC in obfuscated mode, 2) does not allow an adversary to determine the transition from obfuscated mode to functional mode by monitoring and attacking the combinational logic within the FSM, and 3) reduces the observable structural information of the added security elements in the circuit. The section includes a brief discussion on preventing access to the scan chain during startup in Section V-A and a means to measure the security of a sequential locking technique in Section V-B. The implementation of HSTs is described in Section V-C.

A. Preventing Scan Enable During Startup

If an adversary is able to restart the circuit and switch the IC to scan mode after each clock cycle, the key sequence is vulnerable to extraction, which limits the security provided by the applied logic locking technique. To prevent access to scan mode when the key sequence is applied, the circuit shown in Fig. 3 is utilized. The circuit checks that the number of applied transition signals to the sub-block matches the desired count, ensuring that the activation sequence has completed before scan mode is enabled. If an adversary inserts additional glitches to try to bypass the counter, the circuit transitions into a state outside of the startup sequence that generates an incorrect key response, which results in either the circuit remaining in obfuscated mode or starting the IC in an incorrect state. The utilization of a hidden state function, as described in Section VI-C, permits the designer to determine if the current state is in functional or obfuscated mode before allowing scan chain access, which further protects against adversarial attacks. The observability of the primary outputs is also an important consideration as attacks such as described in [29] and [30] do not require scan chain access. If the protected FSM is observable at the primary outputs, the outputs must also be masked during the start-up sequence to prevent an adversary from determining state transitions by monitoring the outputs of the circuit. As an example, an AND gate with a control signal

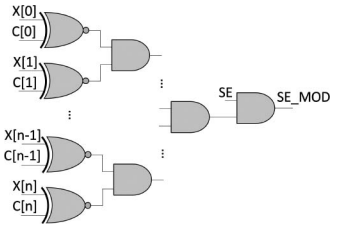


Fig. 3. Count register C is compared with a desired count X to determine the number of transitions that have occurred, which allows entrance to scan mode if correct.

from the comparator shown in Fig. 3 prevents the outputs from switching until the startup sequence has completed.

The functional testing of an IC is minimally impacted as scan chain access is only blocked during the application of the key sequence. Assuming that the testing facility is not trusted, the IC is provided for testing without the correct key. The scan chain becomes accessible after the IC is powered on and the test facility waits the required number of cycles for the secure key sequence to complete. After the scan chain is accessible, the testing facility is able to apply any sequence to the IC to verify the functionality of the device.

B. Security Overview Against Activation Sequence Extraction

Since access to an oracle IC only provides information regarding the state transitions of a circuit operating in the functional mode, the SAT attack no longer applies to the circuit topology developed in this article as the key sequence is not observable due to the included partitioned state machine. The adversary is, therefore, tasked with finding a path from the starting obfuscated state to the ending obfuscated state in s cycles, where s is the number of cycles set by the k-bit length key stored in tamper-proof memory or deduced from the run-time beginning from start-up until activation. The adversary is essentially performing a depth first search of the FSM, attempting each activation sequence while searching. Assuming the FSM has n inputs, f state flip-flops, and that a transition to each state exists, an adversary must search $2^{\min(n,f)*s}$ paths. However, the likelihood that each state transitions to every other state is low. Therefore, the number of paths is weighted by the α term, which results in a search space of $[\alpha * 2^{\min(n,f)}]^s$, where α is a value between 0 and 1. Each path is assumed to have a uniform chance of being the activation sequence, which results in the expected number of paths an adversary must traverse given by (1). An overview of the security against SAT-based attacks when functional mode FSM modifications are implemented is provided in Section VI-E.

$$E[X] = \sum_{i=1}^{\left[\alpha*2^{\min(n,f)}\right]^s} \frac{i}{\left[\alpha * 2^{\min(n,f)}\right]^s} = \frac{\left[\alpha * 2^{\min(n,f)}\right]^s}{2} \quad (1)$$

C. Hidden State Transitions

A methodology to eliminate the logical connection between the obfuscated mode and functional mode is provided in this section. The transition of the state machine once the logical representation in the state transition logic has been removed is referred to as a hidden state transition (HST). Two different methods to generate HSTs are described, which include

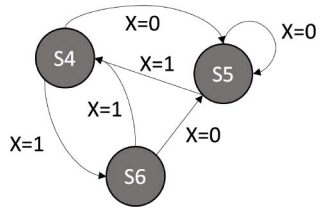


Fig. 4. Original state machine with single input X.

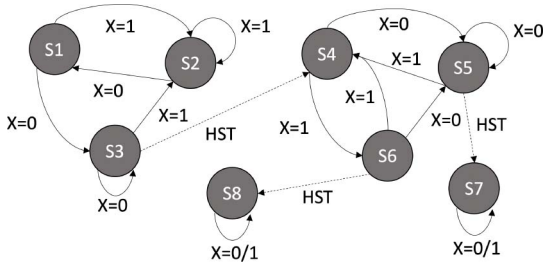


Fig. 5. State machine with HSTs from $S3$ to $S4$, $S6$ to $S8$, and $S5$ to $S7$ that activate when a strategic timing glitch is applied. A single input X is applied to the circuit.

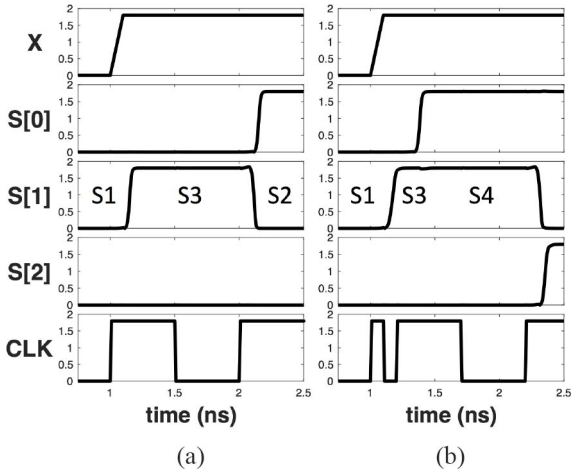


Fig. 6. Simulation of the state machine with (a) no frequency-based state transition and (b) a temporary increase in clock frequency to enable an HST to $S4$.

1) altering the clock frequency and 2) inserting logic to gate the clock signal, where both produce a strategic timing glitch in the circuit. There are two primary security advantages that HSTs add over traditional logic-based transitions. The first is the ability to increase the number of paths the adversary must search. Since an adversary is unaware of whether or not a HST occurs during a given clock cycle, the number of possible paths increases, which corresponds to an increase in the α term of (1). The second benefit is that the additional paths are not dependent on modifications to the original state transition function. The amount of structural information on the circuit corresponding to the execution of the HST is, therefore, limited.

1) Frequency-Based Hidden State Transitions: To demonstrate a frequency-based HST, consider the original state machine shown in Fig. 4. The desired transformation of the original FSM results in a structure similar to that depicted in Fig. 5. The HSTs are shown as dotted lines in Fig. 5, with the primary HST from the obfuscated FSM to the functional FSM occurring from $S3$ to $S4$. The transition from $S3$ to $S4$ is dependent on a modification (increase for this example) to the clock frequency, which prevents the state transition logic from propagating the correct logical value to a register of the FSM.

To demonstrate the execution of HSTs within a circuit, the FSM shown in Fig. 5 was implemented and simulated using SPICE. The results are shown in Fig. 6(b), which indicate that while in $S3$ and a pulse corresponding to a frequency of 5 GHz is applied, the circuit transitions to $S4$ ($S1$–$S3$–$S4$). However, when the pulse is not applied, the circuit transitions to $S2$ on the next clock edge, as shown in Fig. 6(a). By applying a clock pulse corresponding to a frequency of 5 GHz, the state transition logic for register $S[0]$ does not have enough time to switch to logic 1. Instead, $S[0]$ is held at logic 0, which places the FSM in the functional mode.

The timing slack of the path chosen for insertion of the HST determines the amount of variation that is tolerable by the circuit. If the path provides minimal timing slack, meaning that the propagation of the logic value requires the majority of the clock cycle, then the execution of the HST is possible at any time prior to the propagation of the signal(s) through the combinational logic. However, if the logical path includes significant slack, the arrival time of the key signal used to execute the HST is constrained to a shorter time interval. Path timing analysis is, therefore, required to ensure that the HSTs occur for all scenarios while accounting for process, voltage, and temperature (PVT) variations.

2) Clock Gating-Based Hidden State Transitions: In addition to relying on changes to the applied clock frequency, HSTs are also implemented by adding logic to the clock port of the registers to generate controlled glitches. Three different topologies that modify the clock signal based on the applied key are shown in Fig. 7. The first topology shown in Fig. 7(a) allows the key signal to gate the clock signal directly. However, a different topology is needed if the circuit already requires clock gating for power management, which includes an XOR connected to the control signal to allow for the execution of the clock gating technique, as shown in Fig. 7(b). As a result, the state of a given register is held when $KEY0$ is 0. If clock pulses are required, then the circuit topology shown in Fig. 7(c) is utilized, which implements an AND/OR gate [31], [32] that allows for the insertion of clock pulses when setting $KEY1$ to 1.

To analyze the acceptable variation in the arrival time of the key signal when executing the HST, the FSM shown in Fig. 5 is utilized. The circuit shown in Fig. 7(a) is connected to the clock input of register $S[0]$ and the arrival time of $KEY0$ is analyzed. The results of the sweep of the arrival time of $KEY0$ are shown in Fig. 8, which indicate a large range of acceptable arrival times of $KEY0$. The minimum arrival time is determined by the hold time of the register at the start of the current clock cycle, as shown at time $t = 0.25$ ns in Fig. 8. The maximum acceptable arrival time is determined by the hold time of the register for the next clock cycle, as shown at time $t = 2.25$ ns in Fig. 8. The $KEY0$ signal was modeled with a 30% variation in arrival time and for both the slow–slow (SS) and fast–fast (FF) corners. The results indicate that a significant portion of the clock cycle is available for execution

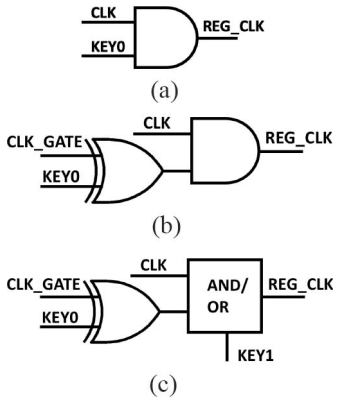


Fig. 7. Circuit topologies to implement logic-based HSTs: (a) key controlled gating of the clock signal, (b) clock signal gated with an AND gate, where the second input of the AND gate is the output of an XOR gate included to implement clock gating, and (c) clock signal gated with an AND/OR [31], [32] and with an XOR integrated to implement clock gating.

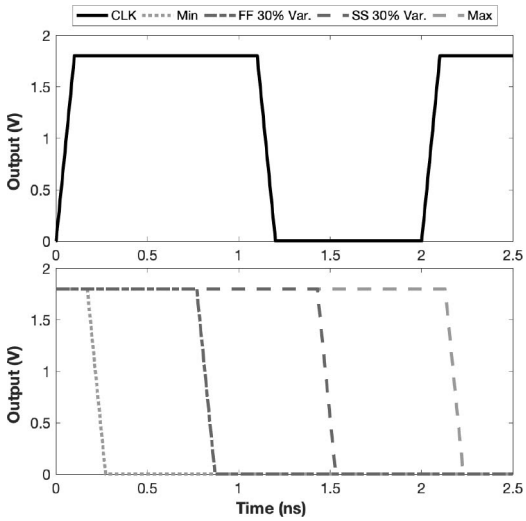


Fig. 8. Variability analysis of the arrival time of *KEY*0 for a HST implemented on register *S*[0] in the FSM shown in Fig. 5. The minimum and maximum arrival times for the *KEY*0 transition are shown, as well as results from analysis of the SS and FF corners, which indicate proper circuit functionality with 30% variation in the arrival time of *KEY*0.

of the HST when *KEY*0 is applied on the falling edge of the clock signal.

VI. Synthesis of Hidden State Transitions

In Section IV, structural attacks on sequential logic locking methods are described that allow for the identification of the circuitry added to implement a given security technique by determining the connections to the tamper-proof memory. Discerning the added structure(s) allows an adversary to determine the enable signals used in [17] or the flip signals used in [26]. The determination of the modification function further exacerbates security concerns as value assignment is static after a correct key is applied to the circuit. An adversary knows that the signals are set to logic 0 for correct operation, which provides a means to bypass/remove the circuitry or reduce the search space to determine the correct activation pattern. In order to reduce structural leakage from partitioned FSMs, the modified FSM must 1) be integrated within the original FSM and 2) not apply a static signal to the hidden state registers. An algorithm is, therefore, developed to incorporate HSTs into an FSM while avoiding static register values when inserting the additional components to the state machine. A flowchart of the primary steps of the algorithm, as shown in Fig. 9, begins with an unobfuscated FSM and upon completion results in a secure FSM.

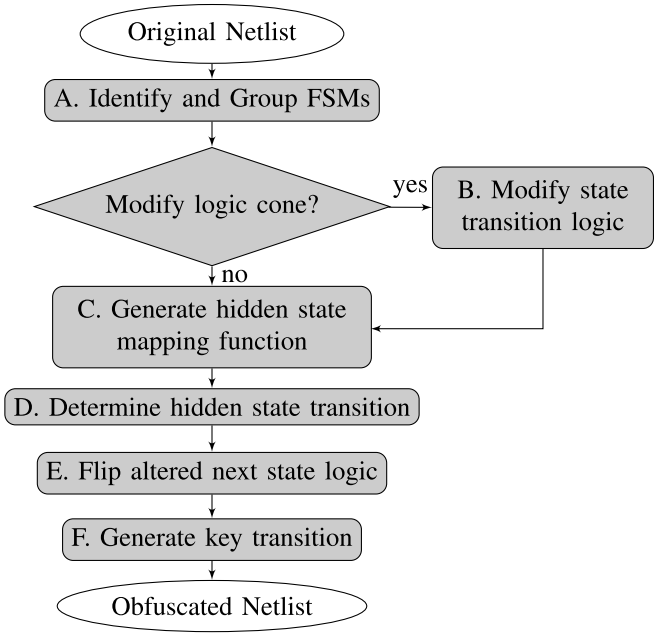


Fig. 9. Primary steps to transform an unobfuscated circuit into a circuit that includes a secure partitioned FSM with an HST. The procedural steps that include a letter are further described in Section VI.

A. Identification and Grouping of FSMs

The method described in [33] is utilized to identify and group the state registers of a netlist into FSMs. All registers are first examined to identify if any paths exist that form a closed loop, where the output from a given register returns to the input of the same register. The list of potential state registers is further filtered by grouping memory elements that share enable and clock gating signals [33]. The potential state registers are then grouped by examining the netlist for strongly connected components [34], where each subgroup includes a subset of the potential state registers that share state transition logic. The final outcome is a set of groupings of FSMs within the original netlist.

The worst-case time complexity, $O(|V|+|E|)$, occurs when the algorithm must explore the entire set of vertices $|V|$ and edges $|E|$ of the directed acyclic graph (DAG) representing the netlist of the circuit. If no FSM groupings are found, the algorithm terminates before any other stages of the algorithm are executed.

B. Modification of State Transition Logic

While a partitioned state machine allows for secure activation and protection against IC counterfeiting and overproduction, structural attacks, as pertaining to IP theft, are still a concern. A possibility exists that an adversary is able to identify the mapping function of the HST and determine the added

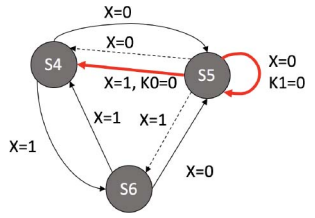
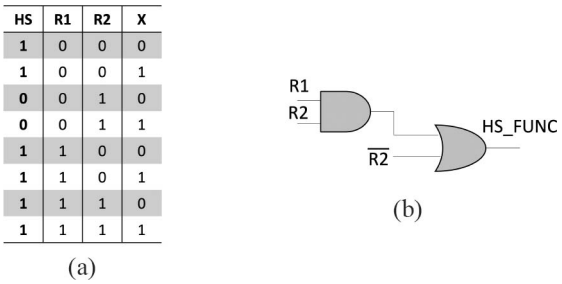


Fig. 10. Modified FSM that requires a temporally dependent key. Dotted lines represent incorrect edges that are corrected by asserting $K0$ or $K1$ in state $S5$ (correct transitions are shown as thick red lines).



HS	R1	R2	X
1	0	0	0
1	0	0	1
0	0	1	0
0	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

(a)

Fig. 11. Example of a hidden state mapping function where (a) depicts the logical remapping of the states in the form of a truth-table and (b) is a schematic of the circuit that implements the hidden state mapping function of an FSM.

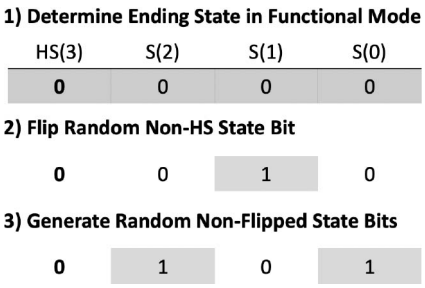


Fig. 12. Procedural steps to generate the HST values. The steps include 1) selection of the functional mode state, 2) flipping of a randomly chosen non hidden state (HS) bit, and 3) random selection of the bits not flipped in step two. The state bits that are modified during each step are highlighted in gray.

hidden state registers if the modifications to the state mapping are not integrated within the original state transition logic when synthesizing the netlist. If IP theft is a concern, then modification of the original state transition logic is needed. Consider the example FSM shown in Fig. 4. The transition from $S5$ to $S4$ when $X = 1$ is changed to the transition $S5$ to $S6$, and the transition from $S5$ to $S5$ when $X = 0$ is changed to $S5$ to $S4$. Modifying the conditions of the state transitions results in the altered state machine shown in Fig. 10.

The correct functionality when in state $S5$ is achieved by setting $K0$ to 0 when X is 1 and $K1$ to 0 when X is 0, as shown by the thick red transition arrows in Fig. 10. The required circuitry to execute the necessary corrections outside of the original logic cone is described in Section VI-E. The in-cone modifications include either an XOR if the function is altered in both the active and obfuscated mode, or the circuits shown in Fig. 7 and described in Section V-C if there is no modification required for the HST. When using the circuits shown in Fig. 7, the corresponding combinational logic does not incur an additional delay for the correction of the state mapping, which results in lower overhead in performance.

Each protected cube added to the FSM requires the addition of approximately $3 * n$ vertices to the netlist represented by a DAG, where n is the number of inputs to the FSM. Since the addition of a vertex to the data structure of the adjacency list graph is $O(1)$, the worst-case time complexity of adding the protected cubes is $O(c * n)$, where c is the number of added cubes. However, the number of cubes added is typically small as each additional protected cube incurs significant overhead in power consumption and area, which results in n as the dominant term. Therefore, the time complexity of adding the additional cubes is typically $O(n)$.

C. Generation of Hidden State Mapping Function

To avoid a static mapping of the hidden state register when the correct key sequence is applied, a novel mapping function of the hidden states is developed to ensure that the added hidden state register periodically switches between logic 0 and logic 1. The primary challenge is remapping the original states with the new hidden state bit. Applying a mapping of the old state to each new state becomes prohibitively costly in terms of run-time as the number of original states increases, with each additional register resulting in an increase in the total number of new states by a factor of 2. Instead, the use of a random logic function as the hidden state mapping results in a significant reduction in the time to remap the states into the new state space. However, applying a purely random function results in a large increase in the area and power of the circuit and potentially degrades the performance as well.

To avoid a significant degradation in the quality of results (QoRs) of the obfuscated FSM, the hidden state mapping function is derived from an internal net within the combinational logic used for the state transition, or is a function of the state registers of the FSM. The hidden states are then derived by XORing the hidden state mapping function. An example of the remapping of the state machine shown in Fig. 4 is provided in Fig. 11, where the hidden state function is $(R1 \cdot R2) + \overline{R2}$. In addition, a random subset of the outputs of the original FSM are flipped to mask the original transitions of the FSM. The modification of the outputs of the original FSM is further described in Sections VI-B and VI-E.

In the case where an internal net is utilized as the hidden state mapping function, the complexity is $O(1)$. If, however, a random function implements the hidden state mapping, then the complexity becomes $O(n)$, where n is the total number of registers included in the selected FSM. The $O(n)$ bound is a result of the random combination of the subset of registers chosen for the hidden state mapping function, which requires the random selection of a pair of registers until no further register pairs exist.

D. Determine Hidden State Transition

Once the hidden state mapping function is embedded into the FSM, the HST is generated via the pseudocode provided as Algorithm 1. To generate the hidden transition, at least one register is chosen as the hidden state register. For example, consider the process shown in Fig. 12, where the starting state

Algorithm 1: Determination of HST

```
Input: FSM netlist netlist;
Hidden state funciton info hs_func_info;
/* Determine random obfuscated end
   state                                  */
hst_end = get_hst_end(netlist, hs_func_info);
/* Determine obfuscated start based on
   end state                              */
hst_start = get_hst_start(netlist, hs_end);
/* Cut netlist at state registers         */
fsm_netlist = get_fsm_netlist(netlist);
/* Add flip signals to fsm_netlist        */
add_flip_signals(fsm_netlist);
/* Unroll fsm_netlist 2x                  */
for state_io in fsm_netlist do
    add_state_constraints(fsm_netlist, state_io);
end
is_sat = satisfy_contraints(fsm_netlist);
if is_sat then
    /* Extract states from SAT model      */
    return flipped_dffs, hst_start, hst_end;
else
    return None;
```

in functional mode is 0000 and the most significant state bit HS(3) is the hidden state bit. Another random state is generated with a one-bit hamming distance, which, for the given example, is 0010. The generated state serves as the ending state of the obfuscated FSM when the HST does not occur. The second least significant bit (LSB) of the starting state of the obfuscated partition must, therefore, equal the second most LSB of the starting state of the original FSM, but may differ in value for any other bit location. In the case of the given example shown in Fig. 12, assume the generated state is 0101. Now, when transitioning from state 0101 to 0010, if the second LSB is held low during capture, the FSM transitions into 0000 instead of 0010.

Once the desired states of the hidden transition and the starting state of the functional FSM are generated, a logical transition from state 0101 to 0010 does not exist within the FSM state transition logic. Note that the starting state of the functional FSM does not have to coincide with the ending state of the hidden transition. To allow for such transitions, an XOR gate is added at the end of the combinational logic path for each state register, with the other input to the XOR taken from the primary inputs of the original logic circuit, as shown in Fig. 13. In addition, the states chosen for the HST must be reachable from other states of the FSM. To ensure the states are reachable, the state transition logic is unrolled twice (more state unrolling is possible), as shown in Fig. 13, where the desired output of state m is the value of the ending state of the HST in the functional partition, and the desired output of state $m-1$ is the value of the starting state of the HST in the obfuscated partition. The only constraints for the states prior to $m-1$ are that the state values do not equal the starting or ending states of the HST, and that the state value of $m-3$ does not equal the state value of $m-2$. The constraints of

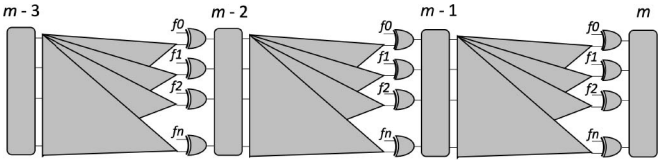


Fig. 13. Unrolling the combinational logic of the state machine. The ending state of the HST in the functional partition is m, the starting state of the hidden transition is $m-1$, and states $m-2$ and $m-3$ are previous states that are constrained to not equal states m or $m-1$. The $f0$ to fn inputs represent the flip signals determined by the SAT solver to meet the required constraints.

the FSM are provided to a SAT solver, which, when successfully completed, validates the inclusion of a given HST and determines if the output of a given register must be flipped. If the SAT solver returns UNSAT, the current HST is discarded and a new HST is generated and validated by the solver, with the process repeating until a valid transition is identified or the maximum number of attempts, as set by the designer, is reached. Reaching the maximum number of attempts indicates that the generated hidden state mapping function is not well suited for modification with an HST. The selection of a hidden state mapping function is, therefore, restarted to generate an HST with a new set of constraints.

The runtime for the generation of the HST is bounded by the use of the SAT solver, which is an NP-complete problem [35]. The generation of an UNSAT condition for a given set of constraints of the HST results in the formulation of a new set of constraints. In this article, the number of restarts is limited to 500; however, the limit is user specified. If during the generation of the HST an UNSAT condition occurs and the number of restarts has been exhausted, the algorithm terminates without generating the modified FSM.

E. Flip Outputs of Altered Registers

A list of the outputs of registers that must be flipped in either the obfuscated or functional mode is generated as a result of executing subroutines of the HST algorithm described in Sections VI-B and VI-D. If the state transition logic is not modified, then the output of the function generated as described in Section VI-C is utilized as the flip signal of the altered registers. The outputs of the modified registers are then simply connected to an XOR gate, with the other input to the gate connected to the flip signal.

If, however, the original state transition logic is modified, then the flip signal is generated from the hidden state function using the inputs to the FSM. The logical structure of the correction circuit is shown in Fig. 14, where the current state of the circuit is compared to a key value to determine if the output of the cone must be flipped or not. The correction circuit serves the function of a look-up-table (LUT) for the modified I/O pairs of the original circuit. Assuming that every register is accessible via the scan chain in functional mode, the sequentially locked circuit is now equivalent to a combinational structure vulnerable to an oracle guided attack. The estimated security against the SAT attack is then equivalent to SFLL-Flex [8], with each DIP given a uniform probability of $c/2^n$ to select a protected cube, where c is the number of cubes and n is the number of inputs to the logic cone. The probability of selecting a protected input cube is then given

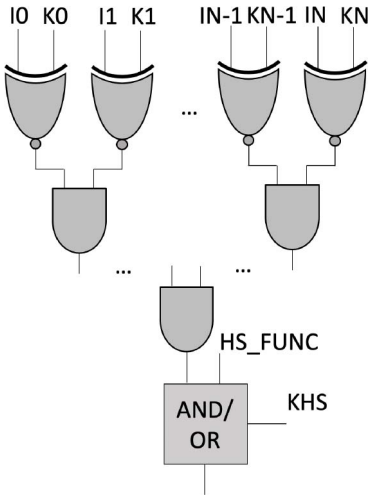


Fig. 14. Logical structure of the correction circuitry.

by $1/2^{n-\log_2 c}$, with the expectation for the number of DIPs required given as (2). Note that the metric is conservative, as the determination of a single cube is used to evaluate the security of the circuit. An adversary must, however, determine all c protected cubes to extract the key.

$$E[X] = \sum_{i=1}^{2^{n-\log_2 c}} \frac{i}{2^{n-\log_2 c}} = \frac{2^{n-\log_2 c}}{2} \tag{2}$$

As discussed in Section VI-B, the time complexity of modifying the netlist to include the protected cubes is $O(n)$. In addition to the circuitry required to add the protected cubes, up to m XOR gates are needed to allow for logical modifications to the outputs of the registers, where m is the total number of registers. Since the addition of each XOR gate is $O(1)$, the time complexity to add the XOR gates to flip the output of the registers is bounded to $O(m)$.

F. Generation of Key Transition

The key is generated based on a user defined number of key bits k. The algorithm iteratively executes to produce a sequence of transitions through the FSM beginning from the starting state of the HST to the starting state of the key sequence, which is equivalent to the starting state of the obfuscated partition. The netlist is first unrolled and a constraint on the input state of the circuit is applied to the SAT solver, which requires that the current state and the returned state from the solver are both reachable. The satisfying conditions of the previous state are determined using a SAT solver and are added to a list containing the states of the key sequence. In addition, the generated state is added to a hash table to track the number of times the generated state is visited. If the frequency of visits to a given state in successive iterations of the SAT solver is greater than a set threshold (set to two in this article), then an additional constraint is added to prevent the solver from returning to the given state. If the solver is unable to generate a solution with the additional constraint on the number of visits, then the constraint is removed and the solver is executed again. If the solver is still unable to produce a valid solution, then the algorithm exits without generating a key sequence.

Generating the key transition(s) is also bounded by the complexity of executing the SAT solver, which is an NP-complete problem [35]. If an UNSAT condition is produced by the SAT solver, the generation phase of the HST algorithm described in Section VI-D is executed again. If a key is not generated for a second time, the algorithm terminates and restarts the key generation process for another FSM to be secured. In this article, the number of attempts to generate a key is limited to four; however, the number is user defined. If the maximum number of attempts to generate a key is reached, the algorithm exits without securing a single FSM.

VII. Results

The algorithm described in Section VI is executed on the ISCAS'89 sequential benchmark circuits. The number of flip-flops within the modified FSM and the total number of inputs to the circuit, which affect the potential security of an FSM as described in Section V, are listed in Table I. A majority of the modified FSMs include at most 15 flip-flops, which are feasibly recoverable with FSM extraction tools, or even brute force analysis.

To estimate the α term from (1), an analysis utilizing a random walk is performed for each FSM. The α term is determined by starting with 50 random states and applying 100 random input sequences, where each input sequence must be different. The average percentage of unique paths over the 100 walks per state is listed in Table I, with most circuit benchmarks producing a random walk score of less than 10%. Results estimating the α term are provided in Table I for circuits that include FSMs modified with an HST, as well as for circuits that include no HSTs. In most cases, the FSMs with an HST produce an α term that is multiple percentage points greater than FSMs with no HSTs, which indicates that an adversary must search extra paths for the correct start-up sequence. For example, a change in α from 2.9% to 4.7% for the s382 benchmark circuit results in an increase in the average number of paths to search from 6.03E+08 to 7.15E+09. A further increase in the α term is possible by adding the circuit shown in Fig. 7(a) to multiple registers, rather than a single register as was done to produce the results listed in Table I. Applying the hidden state circuitry to all 14 state registers of the s382 benchmark circuit increases the α term to 35.46%.

The estimate of the average number of paths an adversary must traverse obtained from computing (1) indicates that circuits with a small number of both flip-flops and inputs only require the analysis of thousands of paths to correctly decrypt. However, when the number of both flip-flops and inputs is sufficiently large, the estimated average number of paths to analyze becomes computationally expensive, as shown for the s15850 benchmark circuit, which requires the analysis of an estimated 5.78E+28 paths to properly determine the obfuscated activation sequence. In order to enhance the security of netlists comprised of smaller FSMs, or FSMs with low unique path scores given by (1), multiple FSMs must be grouped together within the IC. However, some of the small benchmark circuits such as s27 only contain a single FSM. The small circuits are, therefore, not optimal for the implementation of sequential or combinational logic locking.

TABLE I
RESULTS FROM THE ANALYSIS OF THE ISCAS'89 BENCHMARK CIRCUITS, INCLUDING 1) THE NUMBER OF FLIP-FLOPS IN THE OBFUSCATED FSM, 2) THE NUMBER OF INPUTS TO THE FSM BEING OBFUSCATED, 3) THE HST RANDOM WALK SCORE (RANDOM WALK SCORE WITHOUT HSTS), 4) AN ESTIMATE OF THE AVERAGE NUMBER OF PATHS TO DETERMINE THE ACTIVATION SEQUENCE FROM (1), 5) THE NUMBER OF ITERATIONS TO GENERATE THE HST, AND 6) THE RUNTIME TO OBFUSCATE THE CIRCUIT

Benchmark	Number of FSM FFs	Number of Inputs	Random Walk Score (α)	Avg. Number of Paths	Number of HST Iterations	Runtime (s)
s27	4	4	35.5 (26.3)	5.87E+11	1	0.18
s13207	89	580	29.2 (27.3)	9.04E+25	1	13.08
s1423	51	41	25.86 (19.5)	1.13E+16	1	0.77
s1488	7	8	3.18 (2.0)	7.97E+04	9	5.48
s15850	99	484	18.24 (14.4)	5.78E+28	1	20.47
s298	11	7	5.54 (5.0)	5.60E+05	14	1.6
s344	16	9	16.36 (13.8)	3.51E+09	2	0.52
s35932	3	1473	16.00 (14.5)	6.40E-01	1	20.98
s382	14	11	4.7 (2.9)	7.15E+09	3	0.66
s38417	76	1589	3.0 (2.2)	1.13E+21	2	37.44
s38584	138	1321	8.4 (5.6)	1.46E+40	1	12.98
s386	7	7	2.44 (1.6)	1.66E+04	20	2.71
s400	14	11	4.68 (2.9)	6.99E+09	10	1.54
s420	17	19	3.3 (2.0)	4.24E+12	173	25.06
s444	14	11	4.52 (2.9)	5.85E+09	5	1.02
s510	7	19	2.54 (1.6)	1.66E+02	2	0.7
s526	14	11	3.08 (2.9)	8.21E+08	3	0.82
s641	16	37	56.64 (51.9)	5.43E+10	1	0.55
s713	16	37	59.24 (42.2)	6.05E+10	1	0.59
s820	6	18	8.84 (6.2)	5.18E+03	1	0.89
s832	6	18	8.16 (5.6)	3.36E+03	2	1.1
s838	33	35	3.26 (2.3)	3.98E+15	390	109.24
s9234	55	188	27.62 (23.9)	4.98E+15	3	11.85
s953	7	16	2.22 (1.4)	1.67E+02	5	2.08

TABLE II
AREA, POWER, AND PERFORMANCE ANALYSIS OF THE ISCAS'89 BENCHMARK CIRCUITS WITH AND WITHOUT SEQUENTIAL OBFUSCATION IMPLEMENTED WITH HSTS. THE PERCENTAGE OVERHEAD IN AREA, POWER, AND PERFORMANCE IS ALSO PROVIDED FOR EACH BENCHMARK CIRCUIT

Benchmark	Original			Obfuscated			Overhead		
	Area (um^2)	Power (mW)	Timing (ns)	Area (um^2)	Power (mW)	Timing (ns)	Area (%)	Power (%)	Timing (%)
s13207	3.95E+04	6.87	1.85	4.00E+04	7.39	1.87	1.41	7.10	1.07
s1423	1.04E+04	1.82	1.85	1.06E+04	1.85	1.85	1.99	1.76	0.00
s1488	6.18E+03	1.17	1.85	6.27E+03	1.18	1.87	1.45	0.93	1.07
s15850	5.13E+04	9.70	1.86	5.15E+04	9.72	1.86	0.44	0.20	0.00
s27	2.86E+02	0.03	0.49	4.83E+02	0.06	0.65	40.73	43.78	24.62
s298	1.65E+03	0.57	1.10	1.87E+03	0.65	1.23	11.89	12.54	10.57
s344	1.98E+03	0.35	1.16	2.18E+03	0.37	1.28	9.19	6.38	9.38
s35932	1.75E+05	55.54	0.70	1.77E+05	55.87	0.75	0.65	0.59	6.67
s382	2.31E+03	0.43	1.90	2.50E+03	0.49	1.92	7.41	11.80	1.04
s38417	1.83E+05	48.90	1.85	1.85E+05	53.24	1.87	0.80	8.15	1.07
s38584	1.60E+05	48.68	1.86	1.60E+05	50.41	1.86	0.27	3.43	0.00
s386	1.64E+03	0.32	0.96	1.86E+03	0.34	1.32	12.16	6.64	27.27
s400	2.38E+03	0.51	1.60	2.58E+03	0.56	1.84	7.90	8.80	13.04
s420	2.21E+03	0.38	0.77	2.35E+03	0.45	0.92	6.11	14.78	16.30
s444	2.38E+03	0.52	1.53	2.56E+03	0.61	1.70	6.95	15.45	10.00
s510	2.76E+03	0.64	1.50	2.93E+03	0.66	1.68	5.93	2.08	10.71
s526	2.40E+03	0.46	1.54	2.54E+03	0.48	1.72	5.51	4.42	10.47
s641	2.87E+03	0.47	1.86	3.09E+03	0.49	1.86	7.21	4.23	0.00
s713	2.86E+03	0.47	1.85	3.07E+03	0.48	1.94	6.77	1.58	4.64
s820	3.12E+03	0.48	1.45	3.27E+03	0.48	1.73	4.40	1.32	16.18
s832	3.13E+03	0.48	1.69	3.43E+03	0.49	1.76	8.69	1.84	3.98
s838	4.48E+03	0.67	0.85	4.66E+03	0.72	1.07	3.81	5.87	20.56
s9234	1.61E+04	2.20	1.86	1.74E+04	2.80	1.86	7.52	21.50	0.00
s953	5.18E+03	0.86	0.72	5.39E+03	0.87	0.80	3.86	1.58	10.00
Average	**-**	**-**	**-**	**-**	**-**	**-**	**6.79**	**7.78**	**8.28**

The runtime of the algorithm to determine and implement HSTs is also provided in Table I, with the algorithm completing for most of the large ISCAS'89 benchmark circuits in the order of 10's of seconds. The longest runtime was 109.242 seconds for the s838 benchmark circuit, which required 390 restarts of the algorithm to determine an HST, as described in

TABLE III
ADDITIONAL OVERHEAD IN AREA, POWER, AND PERFORMANCE BEYOND THE OVERHEADS REPORTED IN TABLE I FOR 1–5 MODIFIED MINTERMS C WITHIN THE STATE TRANSITION LOGIC. A TIMING OVERHEAD OF 0% IMPLIES THAT THE PERFORMANCE OF THE CIRCUIT IS CONSTRAINED BY THE STATE TRANSITION LOGIC AND NOT THE CORRECTION CIRCUITRY

Benchmark	C=1			C=2			C=3			C=4			C=5		
	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)	Area (%)	Power (%)	Timing (%)
s27	26.40	48.22	0.00	57.60	113.67	0.00	85.60	156.32	0.00	128.80	277.56	0.00	142.40	279.89	0.00
s13207	3.66	4.62	0.00	7.35	9.51	0.00	11.02	14.62	0.00	14.67	19.47	0.00	18.59	27.76	0.00
s1423	4.58	3.70	0.00	9.16	7.75	0.00	13.74	11.58	0.00	18.25	16.14	0.00	22.95	20.15	0.00
s1488	3.83	2.81	0.00	7.56	5.10	0.00	11.33	7.70	0.00	15.00	10.77	0.00	19.00	13.22	0.00
s15850	4.37	4.16	0.00	8.75	8.52	0.00	13.44	13.00	0.00	17.95	18.43	0.00	22.75	23.43	0.00
s298	5.58	3.16	0.00	13.95	10.00	0.00	18.45	9.88	0.00	27.90	19.53	0.00	30.69	17.37	0.00
s344	15.62	16.44	0.00	31.64	31.32	0.00	47.46	48.95	0.00	62.89	65.02	0.00	80.47	84.05	0.00
s349	15.62	16.44	0.00	31.64	31.32	0.00	47.46	48.95	0.00	62.89	65.02	0.00	80.47	84.05	0.00
s35932	0.61	7.27	0.00	1.23	15.46	0.00	1.87	23.28	0.00	2.52	33.02	0.00	3.15	44.58	0.00
s382	4.15	3.32	0.00	10.37	10.49	0.00	13.72	10.37	0.00	20.73	20.50	0.00	22.81	18.23	0.00
s38417	0.46	5.83	0.00	0.92	12.25	0.00	1.38	18.09	0.00	1.84	24.20	0.00	2.32	32.26	0.00
s38584	0.75	8.09	0.00	1.51	17.32	0.00	2.31	26.85	0.00	3.09	36.36	0.00	3.87	45.73	0.00
s386	14.08	13.20	0.00	29.11	26.11	0.00	43.66	39.30	0.00	58.45	54.98	0.00	74.88	69.88	0.00
s400	4.09	3.31	0.00	10.22	10.46	0.00	13.52	10.33	0.00	20.44	20.43	0.00	17.61	15.45	0.00
s420	28.88	27.48	0.00	57.60	55.86	0.00	86.64	89.81	0.00	115.19	120.03	0.00	146.91	150.60	19.74
s444	4.09	3.21	0.00	10.22	10.13	0.00	13.52	10.01	0.00	20.44	19.79	0.00	22.48	17.60	0.00
s510	21.93	19.39	0.00	43.73	39.44	0.00	65.78	63.42	0.00	88.21	86.45	0.00	111.53	106.13	0.00
s526	3.52	1.71	0.00	8.80	5.39	0.00	11.64	5.33	0.00	17.59	10.54	0.00	19.35	9.37	0.00
s641	40.26	45.89	0.00	80.65	97.50	0.00	122.06	146.81	0.00	163.48	207.56	0.00	206.71	282.47	0.00
s713	40.26	45.90	0.00	80.65	97.54	0.00	122.06	146.87	0.00	163.48	207.64	0.00	206.71	282.59	0.00
s820	17.55	13.77	0.00	34.99	28.80	0.00	52.65	46.34	0.00	69.99	61.80	0.00	88.08	76.80	0.00
s832	19.52	23.64	0.00	38.92	49.43	0.00	58.55	79.54	0.00	77.83	106.07	0.00	97.95	131.81	0.00
s838	25.98	28.85	0.00	52.04	61.37	0.00	78.77	92.32	3.66	107.49	132.61	10.98	133.39	173.23	15.85
s9234	7.30	9.09	0.00	14.71	19.56	0.00	22.42	29.55	0.00	30.00	42.61	0.00	37.55	51.17	0.00
s953	9.13	12.47	0.00	18.26	26.13	0.00	27.40	39.12	0.00	36.40	54.39	0.00	45.79	67.85	0.00
Average	**12.89**	**14.07**	**0.00**	**26.46**	**30.30**	**0.00**	**39.46**	**44.94**	**0.15**	**53.82**	**65.68**	**0.44**	**66.34**	**80.36**	**1.42**

Section VI-D. The difficulty of finding a satisfying condition of the constraint for the HST only occurs for a few benchmark circuits, with most only requiring a handful of iterations to successfully determine an HST that does not stem from an unreachable state.

The unobfuscated and obfuscated ISCAS'89 sequential benchmark circuits are characterized for area, power, and performance, with results provided in Table II. The characterization of the ISCAS'89 benchmark circuits is performed on a 180 nm IBM technology node using Synopsys DC compiler. The circuit shown in Fig. 7(a) that executes the HST is included in the analysis of the overhead in power, area, and performance. However, the overheads due to the tamper-proof memory are not considered as is common when analyzing logic locking [4], [8], [15]. The circuits that resulted in the most significant overheads in area and power once obfuscated are the smallest benchmark circuits, which includes s27 and s526. As the benchmark circuits increase in size, the area and power overheads of implementing the sequential locking technique for a majority of the obfuscated circuits is less than 5%, with most of the large benchmark circuits (i.e., s13207, s15850, and s38584) resulting in power and area overheads of less than 1%. A larger variation in the delay overhead is observed across the benchmark circuits, as the delay is dependent on 1) the depth of the combinational logic of the modified FSM and 2) whether the critical path is included as one of the flipped outputs as described in Section VI-E. Rerunning the algorithm and selecting a different HST, therefore, results in a different overhead in delay, as indicated by the results shown in Fig. 15. The variation in the delay due to the selection of the path through the state transition logic is characterized by

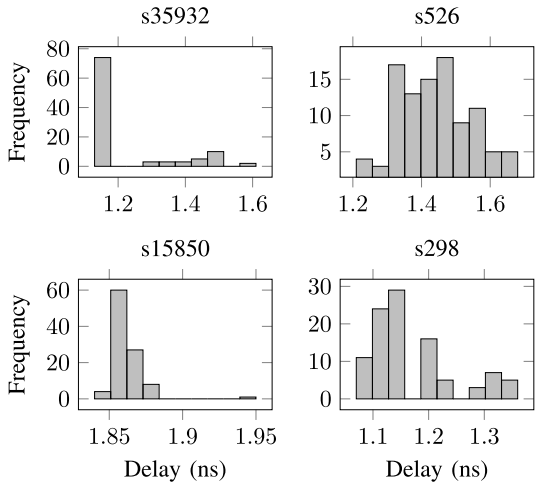


Fig. 15. Histogram of circuit timing (ns) for 100 instances of the s35932, s526, s15850, and s298 benchmark circuits.

running the HST algorithm on the s35932, s526, s15850, and s298 benchmark circuits 100 times.

The additional overhead in area, power, and performance required when the original logic cone is modified to secure against structural attack is listed in Table III. The overheads are analyzed for 1 to 5 cubes C, where the protected cubes are minterms of the given circuit. A lower overhead in power, performance, and area is obtained when using nonminterm protected cubes of the circuit. In addition, the corruption of the outputs increases and attacks such as described in [13] become more difficult to execute. For example, consider the s1423 benchmark circuit, which has 41 primary inputs. The

TABLE IV
ANALYSIS OF EXECUTION OF THE SAT ATTACK WITH A SET TIMEOUT OF 5 DAYS FOR THE ISCAS'89 SEQUENTIAL BENCHMARK CIRCUITS. ALL BENCHMARK CIRCUITS INCLUDE TWO PROTECTED CUBES. THE NUMBER OF ITERATIONS AND THE TOTAL NUMBER OF SAT CUBES ARE DETERMINED AT THE MOMENT A TIMEOUT TO OCCURS

Benchmark	# of Iterations	# of Cubes	CPU Time(s)
s13207	3.33E+04	7.40E+08	TO
s1423	6.94E+04	3.02E+08	TO
s1488	4.83E+03	2.55E+07	1913.23
s15850	4.47E+04	8.85E+08	TO
s27	6.70E+01	1.28E+04	0.11
s298	1.19E+05	9.82E+07	140143.00
s344	1.30E+05	1.74E+08	278422.00
s349	1.71E+05	2.42E+08	TO
s35932	1.91E+04	1.77E+09	TO
s382	1.46E+05	1.42E+08	TO
s38417	2.16E+04	8.81E+08	TO
s38584	1.97E+04	2.15E+09	309432.00
s386	1.62E+03	2.13E+06	35.50
s400	1.70E+05	1.78E+08	TO
s420	1.36E+05	2.18E+08	TO
s444	1.95E+05	1.93E+08	TO
s510	1.09E+05	2.49E+08	TO
s526	1.38E+05	1.60E+08	TO
s641	1.01E+05	2.08E+08	TO
s713	1.34E+05	2.77E+08	TO
s820	1.33E+05	3.18E+08	TO
s832	1.06E+05	2.55E+08	TO
s838	1.12E+05	3.50E+08	TO
s9234	2.87E+04	6.76E+08	TO

output corruption when a single protected cube is utilized is 9.1E–11%. However, if the protected cube constrains only 20 inputs, the corruption increases to 9.54E–5%, but the resilience to a SAT attack, as given by (2), is reduced to an average of 5.24E+5 iterations from 1.1E+12 iterations. For larger FSMs, the tradeoff between output corruption and provided security is acceptable. However, for smaller FSMs, the resulting decrease in the security of the circuit may not be sufficient to protect against the primary attack vectors including SAT.

Implementing a secure partitioned state machine to protect against unauthorized activation and modified logic to protect against reverse engineering forces an adversary to conduct two separate attacks. The adversary must determine the activation sequence, with a measure of the provided security given by (1). In addition, the adversary must utilize an oracle guided attack to determine the modifications to the functional mode FSM, with the measure of the provided security given by (2). The analysis of the resilience of all circuits to the SAT attack was conducted on a Xeon E52687W CPU running at 3 GHz with 95 GB of RAM, with a timeout set to 5 days (432,000 seconds). The results of the analysis of the resilience of the secured circuits to the SAT attack are listed in Table IV for the case when two minterm cubes are protected. Over 75% of the executing SAT attacks time out in five days, with the exception of the benchmark circuits that consist of a small number of inputs, as listed in Table I. The results also correlate with the estimated number of required attack iterations given by (2), which further demonstrates strong resilience against SAT-based attacks. Note that the execution of the SAT attack on s38584 returned the correct key despite a large security expectation as calculated by (2). The SAT attack was rerun with five different netlist orderings, as described in [28]. However, all but one trial resulted in a decryption of the key sequence as the locked cone has a large fanout but a shallow logic depth, which provides the SAT attack with increased probability of finding the modified logic cubes.

TABLE V
COMPARISON OF DIFFERENT SEQUENTIAL LOGIC LOCKING METHODOLOGIES AGAINST VARIOUS ATTACK VECTORS, INCLUDING 1) ORACLE GUIDED ATTACKS, 2) STRUCTURAL ATTACKS THAT ALLOW FOR THE BYPASS OF THE ADDED SECURITY FEATURES, 3) EXTRACTION OF THE TRANSITION FROM OBFUSCATED TO FUNCTIONAL MODE, AND 4) EXTRACTION OF THE COMBINATIONAL STATE TRANSITION LOGIC THROUGH REVERSE ENGINEERING. A ✔ MEANS THE TECHNIQUE IS NOT VULNERABLE TO THE ATTACK VECTOR AND A ✗ MEANS THE METHODOLOGY IS SUSCEPTIBLE TO THE ATTACK VECTOR

Obfuscation Methodology	Oracle Guided Attacks	Structural Attacks	Extraction of Obfuscated to Functional Mode Transition	Extraction of State Transition Logic
[17]	✔	✗	✗	✗
[25]	✗	✔	✗	✔
[18]	✗	✔	N/A	✔
[26]	✗	✗	N/A	✗
HST	✔	✔	✔	✗
HST-MOD CONE	✔	✔	✔	✔

A comparison of the security of circuit obfuscating methodologies against various threats is provided in Table V. The proposed HST algorithm is the only partitioned state machine algorithm that protects against extraction of the obfuscated to functional mode transition, which results in an increase in the number of paths an adversary must search to determine the key sequence that activates the circuit. In addition, a HST with modifications to the logic cone protects against extraction of the state transition logic.

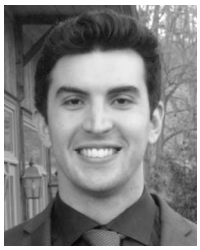
VIII. CONCLUSION

A methodology to determine and integrate HSTs into an FSM within a circuit is described in this article. The elimination of the logical transition between the obfuscated and functional modes of a circuit greatly increases the key space an adversary must search to determine the key sequence that activates the IC. In addition, the developed algorithm introduces modifications to the original state transition logic, which increases the difficulty of performing a structural attack on the FSM. The developed algorithm is analyzed on the ISCAS'89 sequential benchmark circuits, the implementation of which results in an average overhead of 6.79% in area, 7.78% in power, and 8.28% in performance across all of the sequential circuits. Modifying two minterms of the state transition logic leads to an additional overhead of 26.46% in area and 30.30% in power, but does not result in an overhead in performance. The results demonstrate that the developed HST-based algorithm provides greater resilience to oracle guided attacks at a low-cost in overhead, while providing a means to control the activation of ICs sent to third-party facilities.

REFERENCES

[1] *Trusted Integrated Chips (TIC) Program*, document IARPA-BAA-11-09, IARPA, Riverdale Park, MD, USA, Oct. 2011.

[2] *Trends in the Global IC Design Service Market*, DigiTimes, Taipei, Taiwan, Mar. 2012. [Online]. Available: http://www.digitimes.com/news/a20120313RS400.html?chid=2
[3] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.
[4] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. IEEE/ACM Design Autom. Test Eur. Conf.*, Munich, Germany, Mar. 2008, pp. 1069–1074.
[5] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust*, Washington, DC, USA, May 2015, pp. 137–143.
[6] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.*, Jun. 2016, pp. 127–146.
[7] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust*, McLean, VA, USA, May 2016, pp. 236–241.
[8] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Nov. 2017, pp. 1601–1618.
[9] M. Li *et al.*, "Provably secure camouflaging strategy for IC protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1399–1412, Aug. 2019.
[10] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust*, McLean, VA, USA, May 2017, pp. 95–100.
[11] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Proc. IEEE/ACM Great Lakes Symp. VLSI*, May 2017, pp. 179–184.
[12] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 21, 2017, doi: 10.1109/TETC.2017.2740364.
[13] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *Proc. IEEE Design Autom. Test Eur. Conf. Exhibit*, Florence, Italy, Mar. 2019, pp. 936–939.
[14] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "SigAttack: New high-level SAT-based attack on logic encryptions," in *Proc. IEEE Design Autom. Test Eur. Conf. Exhibit.*, Florence, Italy, Mar. 2019, pp. 940–943.
[15] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.
[16] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 66–75, Jan./Feb. 2010.
[17] S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.
[18] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *Proc. IEEE Int. Conf. Circuits Syst.*, Baltimore, MD, USA, May 2017, pp. 1367–1370.
[19] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *Proc. Asia South Pac. Design Autom. Conf.*, Macau, China, Jan. 2016, pp. 655–660.
[20] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proc. IEEE*, vol. 100, no. 11, pp. 3056–3076, Nov. 2012.
[21] C. H. Kim and J.-J. Quisquater, "Faults, injection methods, and fault attacks," *IEEE Design Test Comput.*, vol. 24, no. 6, pp. 544–545, Nov./Dec. 2007.
[22] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Muehlberger, "High level fault injection for attack simulation in smart cards," in *Proc. IEEE 13th Asian Test Symp.*, Kenting, Taiwan, Nov. 2004, pp. 118–121.
[23] Y. Alkabani and K. Farinaz, "Active hardware metering for intellectual property protection and security," in *Proc. 16th USENIX Security Symp.*, Aug. 2007, pp. 1–16.
[24] K. Juretus and I. Savidis, "Time domain sequential locking for increased security," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, May 2018, pp. 1–5.
[25] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in *Proc. ACM 8th Annu. Cyber Security Inf. Intell. Res. Workshop*, Jan. 2013, pp. 1–4.
[26] J. Dofe and Q. Yu, "Novel dynamic state-deflection method for gate-level design obfuscation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 273–285, Feb. 2018.
[27] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, Nov. 2007, pp. 674–677.
[28] K. Juretus and I. Savidis, "Characterization of in-cone logic locking resiliency against the SAT attack," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Jun. 27, 2019, doi: 10.1109/TCAD.2019.2925387.
[29] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Irvine, CA, USA, Nov. 2017, pp. 33–40.
[30] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *Proc. IEEE Design Autom. Test Eur. Conf. Exhibit.*, Florence, Italy, Mar. 2019, pp. 534–539.
[31] K. Juretus and I. Savidis, "Reduced overhead gate level logic encryption," in *Proc. IEEE/ACM Int. Great Lakes Symp. VLSI*, Boston, MA, USA, May 2016, pp. 15–20.
[32] K. Juretus and I. Savidis, "Reducing logic encryption overhead through gate level key insertion," in *Proc. IEEE Int. Conf. Circuits Syst.*, Montreal, QC, Canada, May 2016, pp. 1714–1717.
[33] Y. Shi, C. W. Ting, B. Gwee, and Y. Ren, "A highly efficient method for extracting FSMs from flattened gate-level netlist," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, France, May 2010, pp. 2610–2613.
[34] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.
[35] S. Cook, "The complexity of theorem-proving procedures," in *Proc. ACM Symp. Theory Comput.*, 1971, pp. 151–158.



Kyle Juretus (Student Member, IEEE) received the Bachelor of Science degree in computer and electrical engineering and the Masters of Science degree in computer engineering from Drexel University, Philadelphia, PA, USA, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree.

He is currently a Research Assistant with the Integrated Circuits and Electronics Laboratory, Drexel University. He is currently a National Defense Science and Engineering Fellow. His research interests include circuit level techniques to prevent intellectual property theft and counterfeiting, mitigating side-channel leakage of integrated circuit designs, and design automation for hardware security.

Ioannis Savidis (Senior Member, IEEE) received the B.S.E. degree in electrical and computer engineering and biomedical engineering from Duke University, Durham, NC, USA, in 2005, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Rochester, Rochester, NY, USA, in 2007 and 2013, respectively.

He joined the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA, USA, in 2013, where he is currently an Associate Professor and directs the Integrated Circuits and Electronics Design and Analysis Laboratory. His current research interests include analysis, modeling, and design methodologies for high performance digital and mixed-signal integrated circuits, power management for SoC and microprocessor circuits, hardware security, including digital and analog obfuscation and Trojan detection, and electric and thermal modeling and characterization, signal and power integrity, and power and clock delivery for heterogeneous 2-D and 3-D circuits.

Dr. Savidis was a recipient of the 2018 National Science Foundation Early Faculty (CAREER) Award. He also serves on the editorial boards of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, *Microelectronics Journal*, and the *Journal of Circuits, Systems and Computers*. He serves on the organizing committees of the IEEE International Symposium on Hardware Oriented Security and Trust, the ACM Great Lakes Symposium on VLSI, and the International Verification and Security Workshop. He is a member of the Institute of Electrical and Electronic Engineers, the Association of Computing Machinery, the IEEE Circuits and Systems Society, the IEEE Communications Society, and the IEEE Electron Devices Society.