IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 January 2023, accepted 7 February 2023, date of publication 14 February 2023, date of current version 1 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3244902

RESEARCH ARTICLE

ReTrustFSM: Toward RTL Hardware Obfuscation-A Hybrid FSM Approach

M. SAZADUR RAHMAN, RUI GUO, HADI M. KAMALI, FAHIM RAHMAN, FARIMAH FARAHMANDI, (Member, IEEE), AND MARK TEHRANIPOOR, (Fellow, IEEE)
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA
Corresponding author: M. Sazadur Rahman (mohammad.rahman@ufl.edu)

ABSTRACT Hardware obfuscating is a proactive design-for-trust technique against IC supply chain threats, i.e., IP piracy and overproduction. Many studies have evaluated numerous techniques for obfuscation purposes. Nevertheless, de-obfuscation attacks have demonstrated their insufficiency. This paper proposes a register-transfer (RT) level finite-state-machine (FSM) obfuscation technique called ReTrustFSM that allows designers to obfuscate at the earliest possible stage. ReTrustFSM combines three types of secrecy: explicit external secrecy via an external key, implicit external secrecy based on specific clock cycles, and internal secrecy through a concealed FSM transition function. So, the robustness of ReTrustFSM relies on the external key, the external primary input patterns, and the cycle accuracy of applying such external stimuli. Additionally, ReTrustFSM defines a cohesive relationship between the features of Boolean problems and the required time for de-obfuscation, ensuring a maximum execution time for oracle-guided de-obfuscation attacks. Various attacks are employed to test ReTrustFSM's robustness, including structural and machine learning attacks, functional I/O queries (BMC), and FSM attacks. We have also analyzed the corruptibility and overhead of design-under-obfuscation. Our experimental results demonstrate the robustness of ReTrustFSM at acceptable overhead/corruption while resisting such threat models.

INDEX TERMS Hardware obfuscation, logic locking, FSM, RTL, structural analysis, BMC.

I. INTRODUCTION

Due to the ever-increasing costs/complexity of IC manufacturing, the rush time-to-market, and the ability to take advantage of cutting-edge technologies, many companies are adopting the horizontal model, where multiple independent entities fulfill various stages of the IC supply chain, forming a globally distributed supply chain [1]. Albeit highly beneficial, with outsourcing, the original equipment manufacturers take fewer precautions to meet the market demand (OEM). With no reciprocal trust and a lack of reliable monitoring, the control of OEMs and third-party IP vendors over the supply chain reduces drastically, resulting in numerous hardware security threats, including but not limited to IP piracy, IC overproduction, and counterfeiting [2], [3].

Many design-for-trust solutions (e.g., watermarking, IC metering, IC camouflaging, and hardware obfuscation) have been studied in the literature to address these threats [4], [5], [6], [7]. Over the last two decades, hardware obfuscation, a.k.a. logic locking, has garnered considerable attention as a trustworthy proactive method for securing IP [8]. Logic locking enables the IP/IC designers to support post-fabrication activation of the fabricated designs to recover the underlying functionality. The logic locking secret governs the resulting locked circuit functionality, i.e., the key, and is only known to authorized/trusted entities, e.g., IP owners or OEMs.

Since 2005 [7], numerous studies have focused on various methodologies for building sophisticated obfuscation techniques. However, with newer de-obfuscation attacks, the robustness of most locking solutions has been undermined. Functional input/output (I/O) query-based attacks that primarily rely on satisfiability (SAT) solvers [9] are widely applicable to different combinational logic locking techniques [10], [11]. Structural analysis-based attacks [12], [13], [14] on point function techniques [15], [16], [17], [18],

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Saleem.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

TABLE 1. Comparison of ReTrustFSM with the existing FSM obfuscation methods.

Method	Mechanism	Vulnerability	Exploited by
HARPOON [30]	Adds extra authentication and obfuscation states prior the original FSM	Obfuscated states start first and are loosely connected to the original part of FSM	RANE [28], Fun-SAT [29], Structural [31]
Interlocking [32]	Augments orignial FSM with obfuscation mode and code-word	Code-word doesn't impact output function	Fun-SAT [29], Structural [31]
State [33] Deflection	Verifies enabling key at each state and deflects to black holes	Obfuscated states are loosely connected to the orignial part of FSM	Structural [31]
Active [5] Metering	Adds extra obfuscation states and black holes for correct order	Obfuscated states are loosely connected to the original FSM	Fun-SAT [29], Structural [31]
JANUS [34, 35]	Concealment state register role using a configurable FF	Leaves scan chain open (leak of FF type)	BMC not tested (§II-D)

attacks relying on machine learning [19], [20] on routing-based techniques [21], [22], [23], and theory-based SAT attacks [24], [25] on non-Boolean logic locking [26], [27] are some of the evident examples showing the concerning state of the logic locking.

Logic locking can be categorized into *combinational* and *sequential*. Combinational locking targets the combinational functions (logic parts) [10], [11], [15], [16], [17], [18], [21], [22], [23], [26], [36], whereas sequential ones target the state transitions graphs (STG) of the circuits [28], [30], [31], [32], [33], [34], [35], [37], [38], [39], [40]. In almost all sequential logic locking techniques, instead of having a dedicated reference (secret) as the key (i.e., explicit external secrecy), a sequence of input patterns serves as the unlocking/activation condition (i.e., implicit external secrecy). So, combinational locking is also known as key-based, while sequential logic locking is a keyless one.

Over time, many keyless locking methods have been challenged by different techniques [29], [31], [41]. The {structural+functional} attack focuses on the topological flaws of these methods that allow the adversary to distinguish the original part of the STG from the obfuscated part, leading to recovering the correct FSM [31], [38]. More recent approaches show how these keyless locking, even while they are utilizing implicit representation of the secret [30], [32], [33], [34], [35], [37], can be remodeled and evaluated using I/O query-based attacks like the BMC attack [29], [41]. Table 1 shows a comparative summary of the notable FSM obfuscation methods, their vulnerabilities, and the attacks they are susceptible to (discussed in Sec. II-D).

This paper proposes a novel FSM-based logic locking, called **ReTrustFSM** that targets multiple forms of threats against logic locking. ReTrustFSM is an effort of logic locking at the RTL that leverages the behavioral state transition coding for obfuscation. In ReTrustFSM, both keyless (implicit external secrecy) and key-based (explicit external secrecy) types of locking have been utilized with a high correlation that boosts the robustness of the solution, against oracle-guided I/O query-based attacks and {structural+functional} attack. Also, ReTrustFSM formulates (models) the relationship between Boolean features and the expected de-obfuscation time. By using such a model, ReTrustFSM can be configured in a way that guarantees the targeted time. Additionally, by revealing the leakage possibility of a recent study on FSM-based obfuscation, we evaluate the requirements of the design-for-testability (DFT) structure (scan chain architecture) obfuscation once the FSM-based obfuscation is in place. Our main contributions are as follows:

(I) RT-Level FSM Obfuscation: In ReTrustFSM, the STG is extracted and obfuscated at RT-level.

(II) FSM Re-Encoding Via Explicit External Secrecy: ReTrustFSM uses a counter and LFSR-based FSM obfuscation technique tightly coupled with the original FSM, where the external secrecy (the key) determines how intermediate FSM states (and their encoding values) become dependent on the value/status of the counter/LFSR.

(III) FSM (In-the-Middle) Obfuscation Via Implicit External Secrecy: We utilize a keyless FSM obfuscation technique that conceals intermediate state transition functioning to boost the robustness by building deep obfuscation-oriented transitions directly dependent on the input patterns in a cycle-accurate manner.

(IV) Evaluation of Circuit Attributes for FSM Obfuscation: We equipped ReTrustFSM with an ML-based analysis that focuses on the core features of the obfuscated circuit. The benefit of such a mechanism is twofold: (a) It boosts the complexity of the obfuscated circuit against the existing attacks, particularly I/O query-based attacks, and (b) ReTrustFSM becomes capable of targeting the desired de-obfuscation time before configuring the obfuscation parameters and building the obfuscated circuit(s).

(V) Comprehensive Security/PPA Analysis: To show the comprehensiveness and robustness of the proposed approach, we conduct different analyses using a wide variety of state-of-the-art attacks on the proposed logic locking.

The rest of the paper is organized as follows: Section II covers backgrounds of key-based and keyless obfuscation, as well as FSM obfuscation. Section III provides the details of the proposed scheme. Detailed security analysis of ReTrustFSM is discussed in Section IV. Section V demonstrates the efficiency of ReTrustFSM experimentally. Finally, Section VI concludes the paper.

II. BACKGROUND

A. THREAT MODEL

FSM-based de-obfuscation attacks are considered successful if the adversary can (1) separate the original STG from the obfuscated part, or (2) retrieve external secrets (explicit/implicit), allowing reconstruction of the original STG. In de-obfuscation attacks, the threat model defines the adversary's capability at the time of the attack. This work considers the following assumptions for threat modeling [9]:

1) ADVERSARY DEFINITION

Any individual involved in the IC lifecycle, from the untrusted foundry to malicious end-users, can be an adversary. In this case, the adversary may have access to the (locked) GDSII (at an offshore foundry) and the activated chip. In both cases, the adversary can retrieve the *locked gate-level netlist* from either locked GDSII or activated chip by reverse engineering [8].

2) ACCESS TO THE ORACLE

The adversary has access to an activated/unlocked chip. So, I/O query-based attacks like SAT are applicable to ReTrustFSM. The adversary can also utilize oracle-less attacks, such as ML-based attacks.

3) ACCESS TO THE TEST INFRASTRUCTURE

The adversary can access the scan chain (not stitched to the obfuscation part). Section III-D describes how this access becomes selective, and limits the adversary to a non-combinational de-obfuscation attack, like a BMC-based attack.

4) BASIC INFORMATION ABOUT THE OBFUSCATION TECHNIQUE

The adversary knows the obfuscation method and the location of the key gates (LFSR/counter) related to explicit external secrecy. Also, the adversary knows that part of the secrecy comes from the PIs (implicit external secrecy).

B. EXPLICIT VS. IMPLICIT OBFUSCATION

The majority (more than 95%) [8] of logic locking techniques use the concept of **explicit external secrecy** for building their techniques. In this concept, for a circuit $\{c_{org} : I \rightarrow O\}$, in which $I = \{0,1\}^n \times \tau$ and $O = \{0,1\}^m \times \tau$ are the inputs/outputs sequence (sequence of n/m-bit inputs/outputs for τ cycles), after obfuscating using the explicit-based secrecy augmentation, the obfuscated circuit $\{c_{obf} : I \times K \rightarrow O\}$ requires an additional variable/input, referred to as the *key*. When the secrecy is applied to the circuit using explicit external input, it is called "key-based logic locking." In such cases, for any arbitrary obfuscated circuit (c_{obf}), $c_{obf}: I \times K \rightarrow O$, where $K = \{0,1\}^k$ is the *key* space, there exists $k_c \in K$ such that $\forall i \in I \Rightarrow c_{obf}(i, k_c) = c_{org}(i)$. On the contrary, some studies have evaluated the utilization of **implicit external secrecy** for obfuscation purposes. In this concept, for a circuit $\{c_{org} : I \rightarrow O\}$, after obfuscating using the implicit-based secrecy augmentation, the obfuscated circuit $\{c_{obf} : I \rightarrow O\}$ is still only dependent to the primary inputs, and there is no new variables/inputs, such as key input. For any arbitrary obfuscated circuit, $\{I_1, I_2\}$ is the expected composition of input patterns, in which I_1 serves as the unlocking/activation sequence, allowing the user to reach the normal mode(s) of the circuit. Since part of the input sequence is used as the unlocking condition, this model is widely used to obfuscate FSM as a keyless logic locking.

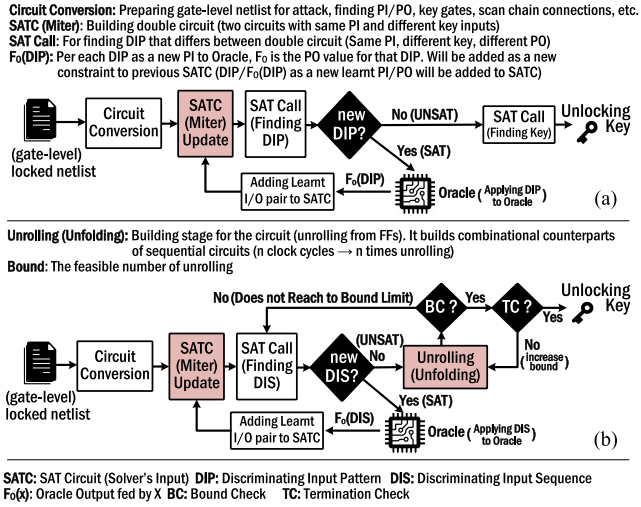
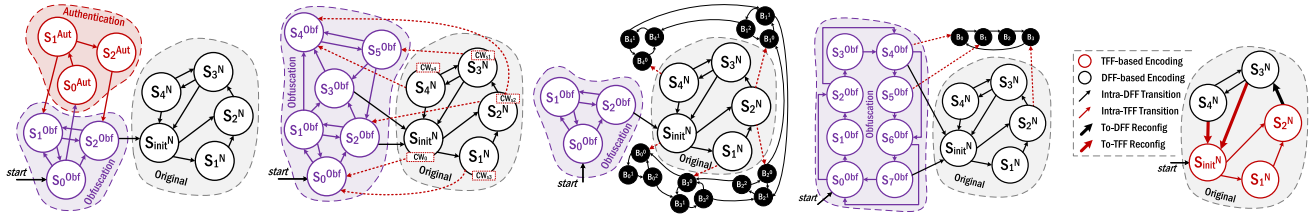


FIGURE 1. I/O query-based De-obfuscation Attacks: (a) Combinational De-obfuscation, (b) Sequential De-obfuscation.

C. I/O QUERY-BASED DE-OBFUSCATION: COMBINATIONAL VS. SEQUENTIAL

I/O query-based de-obfuscation attacks mainly focus on analyzing I/O pairs on locked/unlocked circuits to rule out the incorrect keys. Amongst them, the SAT-based techniques are most effective on a broader range of locking solutions, and they require access to the test infrastructure (scan chain) along with an unlocked/activated circuit (oracle) to target each combinational logic (CL) that is accessible directly via the scan chain, known as combinational de-obfuscation [9]. In a combinational SAT-based attack, as shown in Fig. 1(a), for each obfuscated combinational logic CL_i (*"locked netlist"*), the attacker converts the netlist to be understandable by parsers (defining key gates, adding key inputs, etc.). Then, a (distinguishing) miter (SAT) circuit (SATC) is built as $miter \equiv CL_{i-copy1}(dip, k_1) \neq CL_{i-copy2}(dip, k_2)$. The SAT call on this circuit returns a distinguishing input pattern (dip) that produces different output(s) for two different keys, k_1 and k_2. Later, through the scan chain, the dip is evaluated on CL_i of the unlocked chip ($oracle$, F_0), and the outcome is added as a constraint $CL_{i-copy1}(dip, k_1) = CL_{i-copy2}(dip, k_2) = F_O(dip)$ for the next iterations, and this continues until there is no new dip (UNSAT). Finally, with one more SAT call on all dips, there exists one key that satisfies the condition, which is the correct key.



(a) HARPOON [30] (b) Interlocking FSM [32] (c) State Deflection [33] (d) Active Metering [37] (e) JANUS(-HD) [34, 35]

FIGURE 2. Existing FSM obfuscation solutions.

The combinational SAT-based de-obfuscation is not applicable if there is no/partial access to the scan chain. As in this case, an adversary can only rely on primary input and output (PI/PO) for an I/O query-based (like SAT-based) de-obfuscation attack, the circuit state needs to be taken into account during the de-obfuscation. To do so, FFs will be cut off, and FFs' I/O becomes new circuit PO/PIs. So, in the *miter*, the attacker can apply the initial state directly to the FFs and read them out after capturing. In this case, *unrolling* (unfolding) cascades the design with one more copy to resemble clock cycle sequences. So, with SAT call, the attacker finds the distinguishing input sequences (DIS) for the whole design (*dis* with length equals with the number of unrolling). After finding all *dis*es with the current length, *unrolling* (unfolding) happens for finding *dis*es with the new length. This continues until the attack reaches the threshold bound (BC) or the following termination conditions (TC):

(i) *unique completion (UC)*:

$$\forall dis\ \exists! k_c\ [CU_{copy_i}(dis, k_c) = F_O(dis)]$$

(ii) *combinational equivalence (CE)*:

$$\forall dis\ \exists k_{1,2}\ [CU_{copy_1}(dis, k_1) = CU_{copy_2}(dis, k_2)]$$

(iii) *unbounded model check (UMC)*:

$$\nexists dis, k_{1,2}\ |_{\tau=\infty}\ [CU_{copy_1}(dis, k_1) \neq CU_{copy_2}(dis, k_2)]$$

Due to the duplication and unrolling factors in both attacks (see Fig. 1), the de-obfuscation is not scalable (particularly, the sequential one (Fig. 1(b)) [42]. Thus, limiting scan chain access is an effective approach to combat existing threats that make the problem more complex [43]. Also, the explicit declaration of external secrecy is crucial to successfully setting up and applying a de-obfuscation attack. This is the main incentive for FSM-based obfuscation techniques that mostly rely on keyless (implicit secrecy) obfuscation.

D. FSM OBFUSCATION: PRIOR ART

Prior FSM obfuscation schemes mostly manipulate the STG of the circuits' controller(s) using implicit external secrecy [28], [30], [31], [32], [33], [34], [35], [37], [38], [39]. The augmentation of the original STG can be done by expanding (i) the state space, (ii) the transition space, or (iii) a combination of both. Fig. 2 shows how these methods work.[1] *HARPOON*, depicted in Fig. 2(a), augments the targeted FSM by adding *obfuscation* and *authentication* modes, whose traversal is required *before* reaching out to the original states [30]. So, for the circuit obfuscated by HARPOON, $c_{obf}: I \rightarrow O$, the expected composition of input patterns must be like $I = \{I_{en}, I_{auth}, I_{norm}\}$, where I_{en} denotes the *unlocking sequence* ($S_0^{Obf} \rightarrow S_2^{Obf}$) for *obfuscation* mode, I_{auth} is the *authentication sequence* ($S_0^{Aut} \rightarrow S_2^{Aut}$) for *authentication* mode, and I_{norm} is the normal stimulus on the original FSM ($S_{init}^N, S_1^N, \ldots, S_4^N$). Similarly, approaches like *interlocking*, *dynamic state deflection*, and *active metering* augment the FSM by inserting new modes [31], [32], [33], [37], [38]. For instance, *interlocking*, shown in Fig. 2(b), augments the original FSM by adding the same preceding obfuscation states ($S_0^{Obf}, \ldots, S_5^{Obf}$) like HARPOON. However, they extend the FSM encoding as well as transition function by adding a code-word (CW_i) in a way each *{transition, code-word}* is dependent on the previous {*state, code-word*}, and there exists an initial code-word requirement ($CW_0 \rightarrow S_{init}^N$) that can be considered as the explicit external secrecy [32].

Some other studies use explicit external secrecy (key-based) for FSM obfuscation [28], [34], [35], [39], such as the usage of routing-based obfuscation or key-based counter enumeration for FSM retouching [28], [39]. Li et al. proposed JANUS(-HD), in which the encoding depends on the configuration of the state FFs. As shown in Fig. 2(e), The STG is split into two parts (red and black) whose transition and encoding are based on a specific FF type (D-FF or T-FF). Since it is built based on the different behavior in D-FF (data) and T-FF (toggle), crossing from one mode to another mode (red/T-FF $\leftrightarrow$ black/D-FF) needs an encoding update. So, based on the current state and input patterns, the configuration needs to be updated, and it is done by a configurable MUX (switching between modes). Since the configuration of the FFs stored in a tamper-proof-memory (input to the FSM circuitry), it acts as explicit secrecy.

E. EXISTING ATTACKS ON FSM LOCKING

I/O query-based de-obfuscation attacks, i.e., the SAT/BMC, are formulated in a way that makes them only applicable to key-based obfuscation techniques. However, different studies evaluated a {structural+functional} attack on obfuscated

[1]For few cases, such as *code-word* of interlocking FSM [32] or FF configuration in JANUS [34], [35], the external secrecy is defined explicitly.

FSMs [31], [38]. This attack relies on the fact that the obfuscation part is decoupled from the original part (Fig. 2(a-d)). So, if the adversary, which has access to the netlist, can detect the FFs of FSM circuitry and then tries to re-construct the STG, then the original part can be readily distinguished through the following steps:

1) TOPOLOGICAL/STRUCTURAL ANALYSIS

This step is for distinguishing and decoupling FSM FFs from other FFs (like datapath FFs). The topological analysis consists of steps like (i) identifying FFs whose input contains a combinational feedback path from their output (combinational logic for building next state logic), (2) grouping the FFs controlled by the same set of signals, and (3) finding strongly connected components (SCC) using Tarjan's algorithm [44], [45].

2) FUNCTIONAL ANALYSIS

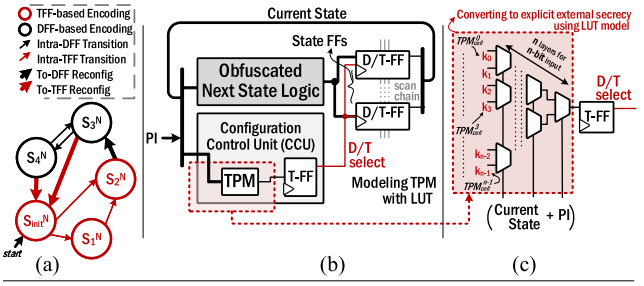
After extracting a pool of potential FSM FFs, the STG is built through functional analysis. This is done by first attempting to find the initial state, and then identifying the reachable states by creating a reduced binary decision diagram (BDD) or using a SAT solver.

3) MATCHING/EXTRACTING ORIGINAL FSM

After extracting the STG, the original part of FSM is retrieved based on some behavioral specifications. In most FSM obfuscation solutions, as demonstrated in Fig. II-D, distinguishing between the original part and the other modes is not very challenging, as these parts are not strongly connected components. In this step, for complex cases that are hard to distinguish between original and extra states, some random stimuli can be matched with the oracle.

More recent studies on FSM obfuscation show how even implicit external secrecy (keyless) can be modeled using I/O query-based attacks [29], [41]. In RANE [41], the secret(s) of HARPOON, which are S^N_{init}, I_{en}, and I_{auth} (Fig. 2(a)), are modeled as explicit secrecy and followed by the BMC invocation for finding the secrets that show more scalability versus the {structural+functional} attack. The attack consists of two main parts: (1) finding the initial state, and (2) finding the *unlocking+authentication* sequence. In FuSAT [29], the minimum number of unrolling needed to find the unlocking/authentication sequence (I_{en} and I_{auth}) is determined based on bounded-depth function corruptibility (FC) analysis that allow the attack to directly jump to the depth leading to the final satisfying assignment(s).

Similar methodology can even be deployed on newer FSM obfuscation techniques, e.g., JANUS or JANUS-HD [34], [35] (Fig. 3(a)), which tries to thwart the usage of any I/O query-based attacks. In this case, as shown in Fig. 3(a,b), the configuration of FFs (D/T *select*), as the secret, is stored in a tamper-proof-memory (TPM), and depending on the current state and the input patterns, this configuration can be updated (based on the value of corresponding TPM_{cell}).



D/T-FF: Configurable DFF or TFF; **TPM:** Tamper-proof Memory; **TPM$^i_{cell}$:** i^{th} Cell of Tamper-proof memory; **D/T select:** Config. value for D/T-FFs; **PI:** Primary Input Sequence; **k$_i$:** key input i designed as the i^{th} TPM cell

FIGURE 3. I/O query-based attack on FSM obfuscation: (a) JANUS(-HD) [34], [35], (b) JANUS(-HD) architecture [34], [35], (c) Converting memory cell contents (TPM^i_{cell}) to external explicit secrecy (k_i) using LUT (To build BMC attack model).

However, as shown in Fig. 3(b,c), one can formulate the whole configuration unit (CCU) module as a small fully configurable logic (e.g., look-up-table (LUT)), in which the current state + PI defines which configuration should be selected, and each configuration bit (k_i) serves as one TPM_{cell} (TPM^i_{cell}). So, considering the configuration of the LUT as the secret, it becomes an obfuscation with explicit external secrecy. Hence, the security of such a technique can be undermined by the invocation of SAT/BMC.[2]

III. PROPOSED SCHEME: ReTrustFSM

To build a robust FSM obfuscation against the attacks, a set of crucial requirements must be met, which are as follows:

REQ_1 Hybrid Approach With High Correlation Between Explicit and Implicit Secrecy: Existing attacks show how I/O query-based attacks can challenge implicit-based FSM augmentation. Also, they do not meet the SCC requirement. However, FSM complexity can be increased by implicit secrecy. On the other side, explicit-based ones follow SCC, but none of the existing ones are robust I/O query-based attacks as they cannot expand the search space enough. To have the best of both worlds, these two can be combined for FSM obfuscation by (i) obfuscating the internal FSM transition function via explicit secrecy and (ii) expanding the search space by implicit secrecy.

REQ_2 No Structural/Functional Traceable Information: The obfuscated design must provide the adversary with the least amount of information (indistinguishability needed for logic locking [8]). The original STG of all except JANUS's in Fig. 2 can be recovered after {structural+functional} analysis as the added states is not part of the SCC of the original FSM. Therefore, FSM obfuscation must be in a way that (1) it is challenging to distinguish FSM FFs, and (2) re-constructing STG becomes difficult or leads to an incorrect one.

REQ_3 Applicable to All FSMs: In techniques like JANUS(-HD), the targets of obfuscation must be a point that allows the designers to split the STG into two parts while the

[2]As this work assumed scan chain open, SAT can be applied directly to CCU. Since CCU is small (for a few transitions), the LUT counterpart will be small enough which makes the de-obfuscation time significantly shorter.

number of cutting edges is minimal to reduce the required size for the TPM that is part of CCU, as demonstrated in Fig. 3. Meeting such requirements always limits the applicability of a method. Therefore, any arbitrary FSM should be obfuscated with no challenge.

REQ_4 Boosting Robustness by Manipulating Original Behavior of FSM: FSM determines how the datapath is controlled, so manipulating its behavior can make de-obfuscation more complex while not affecting functionality. For example, the output of a circuit with valid/ready signals can be masked to 0 (no switching activity) when there is no valid data. Understanding such signals also helps the adversary narrow the modeling (skipping unrollings). It is possible to prevent such reduction by increasing the frequency of output updates (either valid is 1 or 0), based on the power budget, and enhancing the search space for pruning by using the handshaking/communication signals.

A. OVERVIEW OF ReTrustFSM

ReTrustFSM proposes multiple components that apply obfuscation at RTL to meet $REQs_{1-4}$. Fig. 4 shows how ReTrustFSM retouches the FSM of the circuit for obfuscation purposes. In ReTrustFSM, the original (black part) and obfuscated (purple part) FSMs are systematically twisted together (tightly coupled for building SCC in which the obfuscated FSM is in the middle of the original FSM). By doing so, it becomes difficult (almost impossible) for the adversary to distinguish between original and obfuscated states (no target for the initial state to separate these two parts). So, approaches like fun-SAT [29] and RANE [41] are not applicable as there is no initial state as the main target.

ReTrustFSM also utilizes both key-based (explicit) counter-based and LFSR-based obfuscation for STG traversals and implicit secrecy for the traversal of the obfuscated FSM. So, to have a correct connection between two splits of the original FSM (through obfuscated FSM), both explicit and implicit secrecy must be correct. Without correct traversal of the obfuscated FSM, the FSM is pushed towards the initial state, meaning that the expected (correct) functionality is not acquired (malfunctioned). Moreover, the expected behavior of the obfuscated FSM depends on both implicit and explicit external secrecy. Considering the example depicted in Fig. 4, the following shows the key components of ReTrustFSM and how they work:

1) MIXING OBFUSCATED STATES WITH ORIGINAL STATES

As shown previously in Fig. 2(a-d), in existing FSM obfuscation methods, the obfuscation states are separated from the original states and they do not create an SCC together [31], making them vulnerable to {structural+functional} attack. So, to deter such an attack, obfuscation states must be well-blended with the original FSM states in such a way that the Tarjan's algorithm [45], or any similar one, cannot distinguish the obfuscation states from the original states. Hence, in ReTrustFSM, the obfuscated region (S_0^{Obf}, S_1^{lfsr}, S_2^{cnt} and their related transitions in Fig. 4), called **encFSM**, is completely mixed and twisted as an intermediate sub-FSM into the original FSM. Based on the specification of encFSM, it may split the original FSM into three sub-FSMs, as we call them **preFSM**, **lockedFSM**, and **postFSM**. **preFSM** is the first fixed part of the original FSM as the preceding states of the **encFSM** (S_{init}^N, S_1^N, S_2^N and their edges in Fig. 4). **postFSM** is another fixed (could be dynamic as well) part of the original FSM as the succeeding states of the **encFSM** (S_4^N and their edges in Fig. 4). **lockedFSM** is the connecting state (and its outgoing transitions) between the **encFSM** and **postFSM** (S_3^N and their edges in Fig. 4). Additionally, all transitions between **encFSM** and **preFSM**, **lockedFSM**, and **postFSM** are populated (once the correctness of secrecies is not met) to build a unified SCC. Details of these terminologies and their roles in ReTrustFSM are discussed shortly in Section III-B.

2) MIXING KEYLESS AND KEY-BASED FSM OBFUSCATION

Since the key-based obfuscation techniques are vulnerable to SAT/BMC attacks, and keyless obfuscation methods are vulnerable to {structural+functional} attack [31], we introduce a sophisticated hybrid approach in ReTrustFSM combining both key-based (explicit external secrecy) and keyless obfuscation (implicit external secrecy). To do that, traversal of **encFSM**, that allows correctly connecting **preFSM** to **lockedFSM** and then to **postFSM**, ReTrustFSM requires both explicit external secrecy (key input) as the configuration of counter and LFSR and implicit external secrecy (primary input) as the internal transitions of the **encFSM**. With these two secrets, the connection between (**preFSM**, **lockedFSM**, and **postFSM**) will be established correctly ($S_2^N \rightarrow S_3^N$), thereby the original FSM will be reconstructed. As keyless and key-based obfuscation approaches are orthogonal, they individually combat the SAT/BMC and {structural+functional} attacks, respectively.

3) OBFUSCATION-ORIENTED STATES ENCODING

In ReTrustFSM, the encoding of **lockedFSM** (red parts including S_3^N and its outgoing transitions) is calculated by the **encFSM** at run-time. So, it becomes dependent on the execution of the obfuscated counter and LFSR that is located within **encFSM**. Without correct traversal of the **encFSM**, the succeeding states do not work correctly, leading to invalid signaling to the datapath (malfunctioning). The inclusion of state encoding makes ReTrustFSM resistant to removal attacks.

B. STATE ENCODE BY EXPLICIT EXTERNAL SECRECY

In ReTrustFSM, we associate lockedFSM and its outgoing transition function with internal computations of the encFSM. Fig. 4 shows how ReTrustFSM builds this concept by integrating key-based LFSR and counters. As shown, in encFSM, the key input as the explicit secrecy ($key_{0/1}$) determines how the LFSR/counter will operate in the encFSM, and based on their values, the encoding of lockedFSM will be calculated.

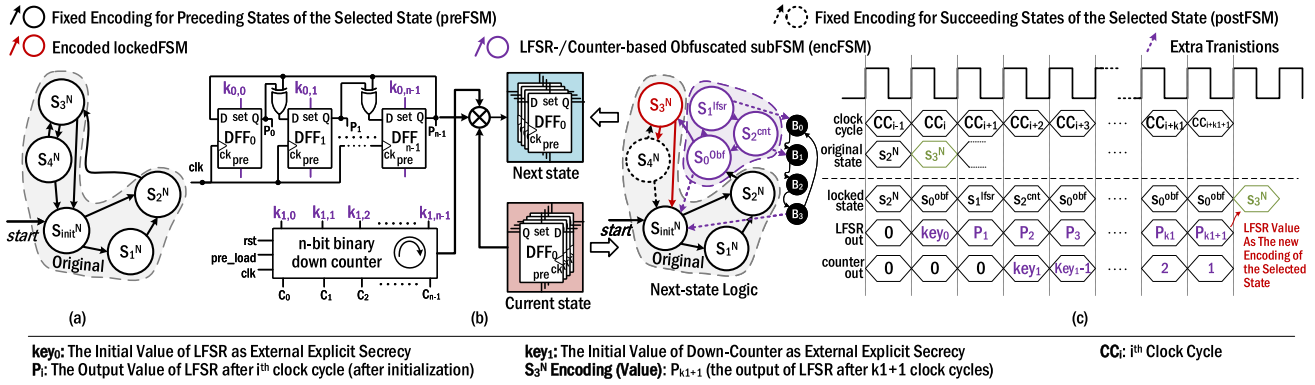


FIGURE 4. Proposed ReTrustFSM: (a) original FSM, (b) obfuscated FSM, (c) Activation: From LFSR/counter initialization to encoding.

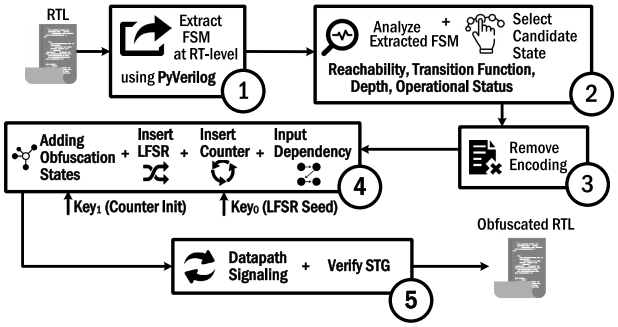


FIGURE 5. The proposed FSM obfuscation flow.

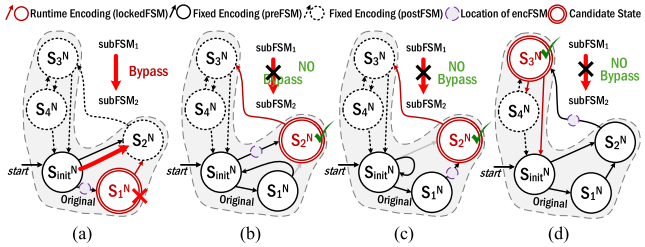


FIGURE 6. Selection of candidate in ReTrustFSM to avoid bypassing of encFSM.

Fig. 4(a,b) show the original vs. obfuscated FSM after applying the proposed scheme, respectively, and Fig. 4(c) shows a timing diagram of how activation works for correctly encoding encFSM (S_3^N). From preFSM to encFSM ($S_N^2 \rightarrow S_0^{cnt}$), the LFSR is initiated with part of explicit external secrecy (key_0). Similarly, during encFSM ($S_1^{lfsr} \rightarrow S_2^{cnt}$), the down counter is initiated with another part of explicit external secrecy (key_1). Then, once the down counter reaches 1, the current value of LFSR will be used as the encoding of the lockedFSM state (S_N^3).

Fig. 5 shows the main steps of ReTrustFSM obfuscation flow whose details are discussed below.

1) FSM EXTRACTION AT RTL

ReTrustFSM framework is equipped with *PyVerilog* [46], allowing us to extract all the crucial information of the original FSM, e.g., states space/encoding, initial state, and state transition function.

2) CANDIDATE STATE ANALYSIS + SELECTION

In ReTrustFSM, candidate states should possess the following properties:

Lemma 1: A candidate is the immediate next state after encFSM and must be the only connection between preFSM and postFSM. It means postFSM must have NO direct in-degree transition from the preFSM. This property ensures that the obfuscation states (encFSM) cannot be bypassed.

In ReTrustFSM, the candidate state is the immediate next state after encFSM. Since both preFSM and postFSM are implemented using fixed encoding, if there exists more than one path from preFSM to postFSM, the encFSM and lockedFSM might be bypassed while some part of the functionality is still available in postFSM (e.g., moving from decoding to execution in a processor). Following Lemma 1 enforces every run to first traverse encFSM (before lockedFSM and postFSM). Also, since the candidate must have only one in-degree transition from the preFSM, meeting this property guarantees that the candidate state is one of the intermediate states (the initial state will no longer be a candidate). Fig. 6 illustrates different scenarios and how bypassing might happen if the candidate does not possess Lemma 1. In Fig. 6(a), as ReTrustFSM selects S_1^N as the candidate, after the insertion of the encFSM on $S_{init}^N \rightarrow S_1^N$, there exists more than one (two) in-degree transitions from preFSM to postFSM (i.e., $S_1^N \rightarrow S_2^N$ and $S_{init}^N \rightarrow S_2^N$). So, the encFSM can be bypassed through the other transition which has no obfuscated state and the adversary might still achieve the functionality by bypassing state S_1^N. Fig. 6(b-d), shows three other scenarios where candidates (S_2^N or S_3^N) follows Lemma 1 and bypassing is no longer possible.

Meeting such property might look like a strict condition that limits the choices of the candidate state for the proposed framework. However, analysis of the benchmark circuits and real applications shows that a significant portion of FSM states in all circuits meet this criterion, and this is mostly because of the sequential nature of the controller's behavior.

Lemma 1 can be extended in ReTrustFSM from one state selection to multiple states selection. In this case, any state member of postFSM can be part of lockedFSM. For instance, based on Fig. 4, the encoding of both S_3^N and S_4^N can be determined by the LFSR/counter. In this case, for I/O query-based attacks, more iterations would be required for re-constructing both states' encoding vs. the case only the immediate state after encFSM is chosen.

Lemma 2: The lockedFSM with encoding should involve such states that feed the datapath circuitry. This property ensures that obfuscating the FSM using ReTrustFSM surely corrupts the datapath circuitry (like the flow of data) so that the design is dysfunctional without the correct secrets.

Based on how the FSM is operating, the controlling signals are defined and sent to the datapath side, which determines the functionality and behavior of the design. Hence, from both functionality as well as corruptibility points of view, the candidate selection would be a crucial step in the proposed framework. As an example in Fig. 4(a), assuming that S_3^N performs the execution step and configures necessary datapaths' registers, it would be a better candidate as it can affect a higher portion of computations within the datapath.

Lemma 3: The lockedFSM with encoding should involve such states that are located at the deepest stage of the FSM.

From the adversary's point of view, assuming that the adversary will apply the I/O query-based attack, e.g., SAT/BMC, targeting and obfuscating the latest stages of the FSM would be a booster as it requires deeper calculation, such as more unrolling for the combinational counterpart creation and larger bound checking. For instance, To observe the output of the circuit after arriving at S_3^N, the adversary requires at least 4 times unrolling (resembling 4 clock cycles). However, for states like S_2^N, it would be in the range of 1-2 cycles for the first observation. We experimentally validated the significance of these Lemmas 1, 2, and 3 in Table 3 by performing BMC attack on the ReTrustFSM obfuscated design of Fig. 6 for varying candidate states.

3) REMOVAL OF STATE ENCODING

Once the candidate states are selected, then their original state encoding is removed from the design RTL. We assume that the designer has manually defined the encoding using some `constant`/`parameter` definition with desired encoding format (binary, one-hot, gray, etc.). The encoding of these states is later driven by the LFSR state which is discussed below.

4) INSERTION OF OBFUSCATION STATES AND ALL RELATED COMPONENTS

This step adds the obfuscated states (encFSM) based on the selected candidate(s). In Fig. 4, S_3^N has been selected as the candidate, thereby ReTrustFSM will insert the encFSM on transition $S_2^N \rightarrow S_3^N$. The states of encFSM (the purple states) are the following three additional states:

a: S_0^{obf}

This state determines whether the traversal of the encFSM is required or not. In ReTrustFSM, to avoid performance/throughput degradation, the traversal of the encFSM will be accomplished only once after power-on (activation). So, if the encFSM is already traversed once, the encoding of the candidate state (S_3^N) has already been done by the LFSR, and the state machine can go directly to the candidate state. We implemented a one-bit flag check to see whether the traversal of the encFSM is needed or not. At power-on, the flag is set to *zero*. Once the flag is *zero*, the encFSM must be passed first to configure the lockedFSM register(s). Through the $S_2^{cnt} \rightarrow S_0^{obf}$ traversal, the flag will be set to *1*. Additionally, in this state, as also shown in Fig 4(c), ReTrustFSM initializes the LFSR circuitry by using the explicit external secrecy key_0. For an incorrect value of the external secrecy, the next states lead to an incorrect encoding for the candidate state, which results in returning back to S_{init}^N (becomes dysfunctional).

b: S_1^{lfsr}

In this state, as shown in Fig 4(c), ReTrustFSM starts updating the LFSR based on its initialized value (starting of the shift). In ReTrustFSM, to ensure that the obfuscation circuitry, e.g., LFSR, and down-counter, is strongly connected with the original FSM, the initialization of the LFSR is controlled by the next state circuitry while the provided key is incorrect. This allows us to fully twist the logic dedicated to the original FSM with that of the encFSM. This significantly helps to immobilize step 1 of {structural+functional} attack [31], [45], which is Tarjan-based topological analysis. To implement this, we do not need to insert a MUX-based encoding selection between correct/incorrect state value(s) for the correct/incorrect key. Since the design team is aware of the location of encFSM in the STG, the encoding of the preceding state (the latest state of preFSM) is known. For instance, as demonstrated in Fig. 4, S_2^N is the immediate state before encFSM with a known (fixed) encoding value. So, to build this twist, we just need to add XOR operation for the key value and the preceding state encoding, whose output produces the correct encoding. For instance, in Fig. 4, the value of key_0 can be defined in a way that $key_0 \oplus S_2^N$ results in producing the correct initial for the LFSR that led to the correct S_3^N encoding (after down counting). So, in this case, the XOR-based model does not reveal any combination of incorrect/correct keys to the attacker. Once initialized, the LFSR starts shifting/shuffling per each clock cycle, and the encFSM goes to the S_2^{cnt} state.

c: S_2^{cnt}

This state initializes the down-counter with the other part of explicit external secrecy (key_1) as the initial count value, and the counter starts down-counting until it reaches 1, as shown in Fig. 4(c). The counter drives the LFSR to encode the

candidate state (S_3^N) after key_1 clock cycles during run time. S_2^{cnt} state then goes to the S_0^{obf} state.

If the LFSR and the down-counter are initialized correctly (correct $key_{0/1}$), the down-counter counts correctly, allowing the LFSR to be in a current value for encoding the candidate state S_3^N. In the event of an incorrect external secret (incorrect $key_{0/1}$), the candidate state S_3^N is encoded with an incorrect value. Hence, they are discarded from the original state machine and the obfuscation states move the state machine to some incorrect states (e.g., black holes $B_{0:3}$ or initial state S_{init}^N in Fig. 4(b)). It is also noteworthy that, once the obfuscated FSM is unlocked, only S_0^{obf} state stays in the state machine during regular operation, thereby no throughput degradation will be experienced after unlocking. Additionally, ReTrustFSM can produce false responses at PO once the state machine is in state S_0^{obf} with no `valid` signal. This increases the frequency of output updates that enhance the robustness against I/O query-based attacks (REQ_4).

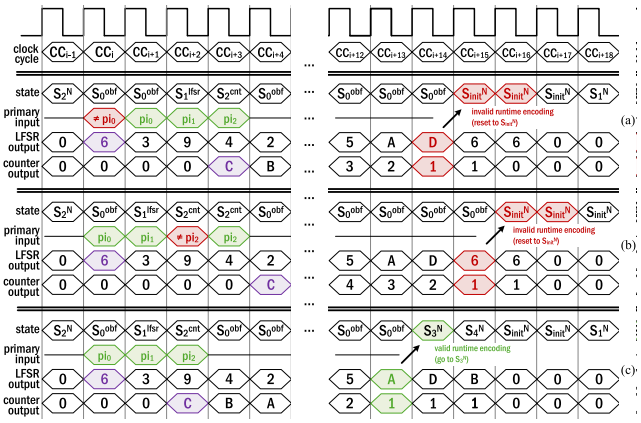


FIGURE 7. Implicit external secrecy for the example of Fig. 4. Correct $\{k_0, k_1\} = \{0110,1100\}$, and LFSR equation is $x^3 + 1$. Incorrect input pattern (cycle inaccurate) for (a) $S_0^{obf} \rightarrow S_1^{lfsr}$, (b) $S_0^{lfsr} \rightarrow S_1^{cnt}$, and (c) $S_0^{cnt} \rightarrow S_1^{obf}$.

5) FUNCTIONAL VERIFICATION WITH INCLUSION OF DATAPATH

Once the insertion of obfuscation states, LFSR, down-counter, and all extra populating transitions is done, ReTrustFSM will verify the functionality between the original and obfuscated circuit while the secrets are provided. In this case, the inclusion of the datapath is crucial as the signaling between the controller and datapath needs to be functionality verified while the encoding is updated and new intermediate states are added. Researchers have investigated how don't care states in FSM can be exploited to insert hardware Trojans and faults [47]. Therefore, ReTrustFSM inserts `default` case to define any don't care states in the design and eliminate any existing FSM vulnerabilities [48].

C. STATE EXPANSION USING IMPLICIT SECRECY

To boost the robustness of ReTrustFSM, particularly against I/O query-based attacks, we deploy a hybrid approach for FSM obfuscation, by establishing a deep correlation between the key-based (explicit external) encoding and keyless (implicit external) obfuscation. To accomplish this, the state transitions within encFSM ($S_0^{obf} \rightarrow S_1^{lfsr} \rightarrow S_2^{cnt}$ in Fig. 4) have been associated to the input patterns at specific clock cycles. So, for clock cycles $cc_{i,i+1,i+2}$, shown in Fig. 4(c), specific patterns must be observed at PI allowing encFSM to be traversed cycle accurately as expected.

Fig. 7 demonstrates how this implicit secrecy works in ReTrustFSM for a 4-bit FSM, while a 4-bit LFSR and counter are integrated. It shows that since the LFSR/counter initialization happens at different clock cycles, incorrect traversal of encFSM states results in having incorrect LFSR value for encoding while the down-counter reaches 1. So, the candidate state will be discarded from the FSM, making the circuit dysfunctional (Fig. 7(a,b)). The implicit secrecy that is dependent on patterns driven by the PI creates functional obscurity and protects the encFSM with the explicit secrecy for the state encoding. Using such a structure, the keyless and the key-based components of ReTrustFSM are correlated to each other during the unlocking and functional phase, to aid in the cycle-accurate functioning of the design.

It is evident that since the implicit external secrecy must be applied at specific clock cycles (in the middle of the first round of execution), the activation may look challenging. However, similar to other sequential FSM-based obfuscation techniques, the whole sequence of input patterns for one full FSM round with the inclusion of encFSM traversal will be the implicit external secrecy. For instance, for Fig. 4, any input pattern sequence allowing us to reach S_2^N, followed by the specific implicit external secrecy needed to pass encFSM, plus any input pattern sequence allowing us to reach S_{init}^N will be considered as a sequence for implicit external secrecy. Note that for applying a full round of input to the circuit, no additional circuitry is required, and the sequence of inputs will be applied directly (activation license).

As mentioned in Section III-B, ReTrustFSM removes the encoding of the candidate state and drives the candidate state by the LFSR state. The LFSR state defines the state transitions concerning the candidate state. However, the candidate state is defined by some `constant/parameter` which is a secret in this case. Having any secret value hard-coded in the RTL could make the design vulnerable to functional analysis-based structural attack [13]. To eliminate any structural trace from the design, stealthy opaque predicate [49] based dynamic secrecy generation can be deployed. To know more about how opaque predicate can be utilized for generating obfuscated constants in the RTL, we direct the readers to [49].

As mentioned previously, the traversal related to the LFSR and the down-counter (counting and shifting) will continue till the down-counter reaches `one`. This means that for the down-counter with greater initial values (greater exact value of external secrecy), it takes more time to calculate

the encoding value of the candidate state. This shows one of the unique features of ReTrustFSM, where it makes a relation between the exact value of the explicit external secrecy and the robustness of the approach. However, it may result in building a backdoor for attacking this mechanism. For instance, for correct case demonstrated in Fig. 7(c), the explicit secrecy pair for the LFSR and the down-counter ($\{k_{0,0:3}, k_{1,0:3}\}$) is set to {0110, 1100}, and for the correct traversal, the down-counter must be initialized two clock cycles after the LFSR (LFSR at cc_i and down-counter at cc_{i+2}). So, for any pair of the LFSR and the down-counter values demonstrated in Fig. 7(c) with two clock cycles difference, it can be considered as the correct value of the external secrecy. For instance, {0011, 1011}, which is the LFSR value at cc_{i+1} and down-counter at cc_{i+3}, could be another correct value of the external secrecy, that also needs one less clock cycle execution to reach down-counter to one. Having such knowledge allows the adversary to fix the down-counter to a small value, and run the I/O query-based BMC for the other part of the secrecy that is related to LFSR, and since it requires few traversals, the BMC will break it very fast.[3] So, for two purposes, which are (1) avoiding having multiple correct key pairs, and (2) avoiding the possibility of running *fix and run partial BMC*, we re-use one of the external secrecies with targeted value to obfuscate part of the design, e.g., the next state logic. It constrains the key value to the expected (targeted) one. Using such constraining limits the correct key to only *0110+1100*, and enforces the attacker to run the BMC to the depth as expected, and additionally, it reduces the number of possible correct keys to only one. All the experimental results demonstrated in Section V are captured after applying this constraint. Please note that adding constraint for one part of the external secrecy would be enough to completely mitigate this issue.

D. NECESSITY OF SCAN PROTECTION

Manufacturing testability provides access to internal circuit states to achieve high test coverage. Debug engineers (from the foundry or test facility) can inspect the concurrent state of the sequential elements in case of manufacturing errors. Capitalizing privilege access to internal state registers does not reveal any secret meta-data in existing FSM obfuscation techniques that use implicit secrecy since no explicit representation of secrecy exists, and the sequence of input patterns serves as unlocking/authenticating sequences. The adversary needs to check all possible transactions to reconstruct the obfuscated FSM if the state registers are known. As a result, while only implicit secrecy is maintained, the test infrastructure may not provide any clues to an adversary. Assume there is explicit secrecy in place. In that case, it may reveal secret meta-data (e.g., the current state of the FSM, the state registers' output/input) concerning the underlying obfuscation method, as the propagation of intermediate variables might reveal something about the explicit secrecy (information leakage). Therefore, for existing FSM obfuscation methods, they may rely on various scan chain securing methods [38]. As discussed in Section II-E, JANUS(-HD) [34], [35] performs FSM obfuscation by deploying reconfigurable D-T flip-flops where a control signal (stored in TPM) decides the role of the flip-flop (either D-type or T-type). The role of the individual reconfigurable state registers (D-type or T-type) is the main secret in the case of JANUS(-HD), and it is assumed that the attacker has full access to the chain. However, having such privileged access to the scan chain, the attacker can initiate a scan-based attack [50] to reveal this explicit secrecy that determines the state register type (D-type or T-type) of the reconfigurable FFs in JANUS(-HD). This can be done by flushing the scan chain with a known pattern and inspecting the location of bit flips in the scan-out response.

Since ReTrustFSM uses both key-based and keyless obfuscation, some registers dedicated to obfuscation may be targeted as sources of information leakage. In ReTrustFSM, security information is contained in register resources dedicated to state encoding, LFSR, and counters. To deter these scan-based attacks, ReTrustFSM assumes the design is full-scan capable and protected by state-of-the-art scan obfuscation techniques [51] to shield these register contents. To keep the scan chain as open as possible, we only obfuscate the chains that include the targeted registers. Despite being a closed scan chain, the DFT structure still makes different chains fully accessible. Our experiments demonstrate that this approach allows for a high test coverage when multiple scan chains are inserted. It also provides a higher level of observability/controllability to the designer while not compromising security for the obfuscated secrets.

IV. DETAILED SECURITY ANALYSIS OF ReTrustFSM

This section provides a detailed security analysis of ReTrustFSM. Considering the I/O query-based BMC attack as an algorithmic approach applicable to FSM obfuscation, and based on the combinatorial complexity of the constraint satisfaction problem, we define the security metric of ReTrustFSM and predicts the security complexity in terms of BMC attacks using a machine-learning (ML) model by extracting a set of features. Furthermore, we evaluate ReTrustFSM against a {structural+functional} attack [31], removal-based attack, functionally guided SAT attack [29], and an Oracle-less machine-learning-based attack [52].

A. OPERATIONAL MODEL AND BMC COMPLEXITY

The SAT and BMC attacks are algorithmic attacks defined on logic locking [9], [41]. In ReTrustFSM, since part of the secrecy is the implicit external secrecy provided by the primary inputs, even though the scan chain is fully accessible, the possibility of running SAT attack (modeling primary inputs as secret) is almost zero. Hence, this section evaluates the possibility of a BMC attack where the adversary needs (1) PI/PO access to the activated chip (oracle, and (2) PI/PI

[3]The same can happen by fixing the LFSR and running the BMC on the secret of the counter.

access to the gate-level netlist. The BMC helps find the best set of I/O pairs, leading to the secrets.

As discussed in Sec. III, the key-based state encoding of ReTrustFSM is driven by an LFSR/counter configuration. Assuming that the LFSR is k_0 bits wide, the *characteristic polynomial* of a k_0-bit LFSR with $c_0, c_1, \ldots, c_{k_0-1}$ feedback coefficient can be represented by the following Equ. 1.

$$P(l) = 1 + c_0 l_0^t + c_1 l_1^t + \ldots + c_{k_0-1} l_{k_0-1}^t \tag{1}$$

Eqn. 1 is a univariate monic polynomial that can be represented by a Frobenius companion matrix [51]. Equ. 2 and Equ. 3 show the companion matrix that can represent the succeeding states of the LFSR (based on the coefficients and its initial value) and its simplified form, respectively.

$$\begin{bmatrix} l_0 \\ l_1 \\ l_2 \\ . \\ . \\ l_{k_0-1} \end{bmatrix}^{t+1} = \begin{bmatrix} c_0 & c_1 & c_2 & . & . & c_{k_0-1} \\ 1 & 0 & 0 & . & . & 0 \\ 0 & 1 & 0 & . & . & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} l_0 \\ l_1 \\ l_2 \\ . \\ . \\ l_{k_0-1} \end{bmatrix}^{t} \tag{2}$$

$$L^{t+1} = A \cdot L^t \tag{3}$$

In ReTrustFSM, once the obfuscated circuit is powered on, the explicit secrecies, i.e. initial secret seed and initial counter value, is loaded into the LFSR and the down-counter, respectively. Let us consider that the seed of the LFSR is L^t and the initial count value of the down-counter is k_1. As discussed previously, in ReTrustFSM, the LFSR stops traversing once the down-counter reaches `one`. At this moment, the current value of the LFSR is utilized to encode the obfuscated state(s), as demonstrated in Fig. 7(c) at clock cycle cc_{i+14}. So, considering that the initial states of the LFSR and down-counter constitute the secret key $\{k_0, k_1\}$ of the circuit in ReTrustFSM, as demonstrated in Fig. 7, and assuming L^t as the initial seed of LFSR and k_1 as the number of clock cycles to reach the current state encoding, the LFSR state that performs correct encoding of the obfuscated FSM can be represented by the Equ. 4.

$$L^{t+k_1} = A^{k_1} \cdot L^t \tag{4}$$

For BMC attack, the model checker tool must visit all succeeding states of the LFSR and down-counter, i.e., the ReTrustFSM obfuscated circuit must be unrolled for at least $(k_1 + \delta)$ times to extract the secret seed of the LFSR, required for correct traversal. Here, δ is related to the other sequential depth of the obfuscated FSM and the obfuscated states, e.g., the number of states that must be traversed to reach the LFSR initialization state (S_1^{lfsr}). To enable ReTrustFSM by a machine learning model that predicts the security robustness against BMC attacks, we concentrate on the features that can be extracted from the formal representation of the problems (obfuscated circuits) based on *propositional logic formula*. This security evaluation consists of three steps: (1) feature engineering, (2) training, and (3) testing and validation.

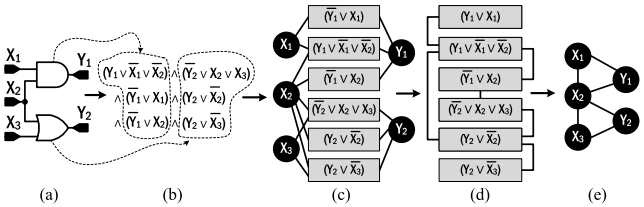


FIGURE 8. The CNF features predictive of the computational complexity of the SAT instance. (a) sample circuit CL_i, (b) corresponding CNF representation of CL_i, (c) VCG graph of CL_i CNF, (d) CG graph of CL_i CNF, and (e) VG graph of CL_i CNF.

Definition 1: A *propositional formula* is a conjunction of *clauses*. A *clause* is a disjunction of literals. A propositional variable or its negation is known as a *literal*.

1) FEATURE ENGINEERING

Knowing (about the source, propagation, and usage) the explicit secrecy in ReTrustFSM (k_0, k_1) allows the designer to generate the propositional formula (in conjunctive normal form (CNF)) of the SATC (as discussed in Section II-C) based on the required/selected unrolling number.[4] In a SAT instance, the number of clauses, variables, and their ratio is highly correlated with the instance's empirical difficulty [53]. The authors of [54] generated a set of features that are predictive of determining the computational complexity of a SAT instance. Some of these features are derived from well-known heuristics, e.g., the number of clauses, variables, and the ratio of clauses to variables, while some others are based on more complex combinatorial properties [53]. Fig. 8 shows a visual representation of the dominant features. A sample circuit and its corresponding CNF formula is shown in Fig. 8(a) and Fig. 8(b) respectively. From Fig. 8(b), it is noticeable that the number of clauses, variables, and their ratios is 6, 5, and $\frac{6}{5}$, respectively. These features capture the size of the SATC generated from the ReTrustFSM obfuscated circuit (after τ-time unrolling). The three undirected graphs in Fig. 8 correspond to three different graph representations of a SAT instance, which are defined as follows:

Lemma 4: A VCG graph G(V,E) is a bipartite graph of the vertex set V and edge set E, where each variable and clause $(v_i,\ c_j) \in V$. The occurrence of a variable v_i in a clause c_j represents an edge, $(v_i \rightarrow c_j) \in E$ in the graph G.

Lemma 5: A CG graph G(V,E) has node for each clause, $c_i \in V$. Whenever two clauses share a negated literal, an edge, $(c_i \rightarrow c_j) \in E$ is added to the graph G.

Lemma 6: A VG graph G(V,E) has node for each variable, $v_i \in V$. Whenever two variables occur together in at least one clause, an edge, $(v_i \rightarrow v_j) \in E$ is added to G.

Fig. 8(c-e) corresponds to the VCG, CG, and VG graph representations of the CNF instance from Fig. 8(b), respectively, by following Lemma 4, 5, and 6. Each of these graphs corresponds to a constraint graph associated with the obfuscated

[4]From SAT solver point of view, for sequential circuits, unrolling will be done as a pre-processing step to build the combinatorial counterpart.

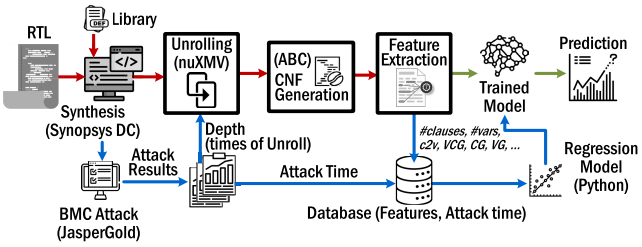


FIGURE 9. Linear regression based security complexity estimation model framework based on the predictive features of the ReTrustFSM obfuscated CNF instance.

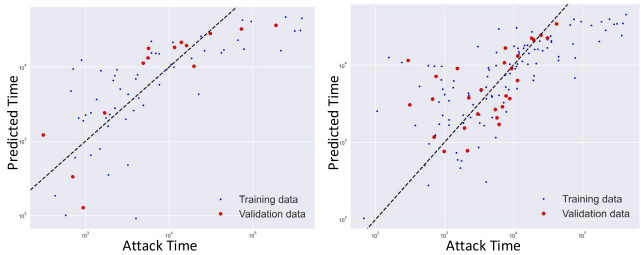


FIGURE 10. Accuracy of the linear regression-based BMC attack complexity prediction model for two different cases - (a) 70 and (b) 140 data points. Both (a) and (b) plot the predicted vs. actual BMC attack time where the blue and red dots represent the training and validation data set. The prediction accuracy metric of the linear regression model increased from (a) 0.21 to (b) 0.77 when the training size was doubled.

circuit. Therefore, they delineate the complex combinatorial structure of the SAT instance. For a detailed understanding of all the features of a SAT instance, we direct the readers to [54]. These features can be combined with a regression-based machine learning model to construct the hardness model of the ReTrustFSM obfuscated circuit.

2) MODEL TRAINING

Fig. 9 provides an overview of the ML framework that we developed for the complexity analysis of the ReTrustFSM obfuscated circuit. The blue steps in Fig. 9 are only required during the training phase while the green steps are for prediction phase. The brown steps are essential for both training and prediction. The details training phase of this ML framework is as follows:

- $\mathbf{ml}_0$ (*Synthesis*): The original (unlocked) and ReTrustFSM obfuscated RTL must be synthesized in a target library using open-source [55] or commercial (Synopsys Design Compiler, or Cadence Genus) EDA tools.
- $\mathbf{ml}_1$ (*Dataset Creation*): Once the original and obfuscated synthesized netlist is generated, we perform a BMC attack on the ReTrustFSM obfuscated circuit using the state-of-the-art BMC attack tool on sequentially obfuscated circuits [41]. If succeeded, the attack results provide the unlocking key, size of *dis* (as discussed in II-C), and time to extract the key. The time taken by the BMC attack fills the database as *target*, which the ML model later utilizes along with the feature set.
- $\mathbf{ml}_2$ (*Manual Unrolling of the De-obfuscated Circuits*): The size of the *dis* reported by the successful BMC attacks represents the number of cycles the sequential circuit must be unrolled (or the number of copies the *CU* of the sequential design needs to be added in the miter SATC as discussed in Section II-C) to perform the combinational SAT attack and deduce the secret key. During the training phase, we utilize the *dis* number along with an in-house developed functional corruptibility-guided [29] unrolling script to generate the miter SATC of the ReTrustFSM obfuscated circuit using an open-source model-checking tool *nuXMV* [56].
- $\mathbf{ml}_3$ (*Generation of Propositional Logic Formulas*): The τ-time[5] unrolled circuit generated by the unrolling tool is in `bench` format. We thereby make use of the ABC synthesis tool [55] to generate the *CNF* of the unrolled version of the ReTrustFSM obfuscated circuit.
- $\mathbf{ml}_4$ (*Feature Extraction of Generated CNFs*): At this step, we plug in the *CNF* representation of the ReTrustFSM obfuscated circuit to an open-source feature extraction tool [54] which generates features that are predictive of determining the computational complexity of the given SAT instance. In our analysis of ReTrustFSM, we extracted 46 features from the given CNF. These features constitute the feature set of the database.
- $\mathbf{ml}_5$ (*Regression for Training*): Once the database is ready, we deploy a linear regression algorithm to train the model based on the extracted CNF features and target BMC attack time. We utilized a *ridge regularizer* with $\alpha = 0.022$ as error metric.

We developed a database of roughly 140 BMC attack results (details of the experimental setup and attack results are discussed in Section V) and their associated features among which 90% of the data points were used to train the linear regression model and 10% (on never-seen circuits by the model, i.e., test data set is not part of the training data set) were kept for validation.[6]

3) TESTING AND VALIDATION

After training the ML model using the pre-generated dataset, the trained model is tested and validated on the targeted circuit(s) (To estimate the de-obfuscation time for a desired external secrecy). Fig. 10 shows the comparison between actual time and predicted time for both training (blue) and validation (red) data sets for two different training data sizes. We collected in total 140 data points and considered two cases for training, testing, and validation. In the first case, we considered half of the collected dataset (70 data points) while in the second case we considered full dataset (140 data points). For both the cases, we utilized 90% of the considered data for training (i.e., 63 data points in case one and 126 data

[5]Per obfuscated circuit, different numbers of unrolling are required.

[6]The size of training dataset can be increased over time, and by increasing the dataset size, higher accuracy can be achieved.

points in case two for training) the linear regression model. We validated the trained model using the residual 10% dataset of the considered data in each case (i.e., 7 data points in case one and 14 data points in case two). Accuracy of the BMC attack complexity prediction model for two different cases are presented in Fig. 10(a) and (b), respectively. The prediction accuracy metric, $R2$-score increased from 0.21 to 0.77 when the training data size was doubled. Hence, further expanding the training data set will increase the prediction accuracy beyond 0.77. It can also be observed from Fig. 10 that for the training data, the model mostly performed underestimation. However, the model was able to predict the validation data with an average of 90% accuracy.[7] It is noteworthy that the model can be trained offline with the results generated from the prior BMC attacks and its associated features. Enabling our proposed FSM obfuscation by this ML model estimator allows the designers to acquire the estimated de-obfuscation time before the accomplishment of obfuscation. Based on the desired external secrecy (down-counter initial value) and the minimum number of required unrolling (which can be identified by utilizing functional corruptibility [29]), the model (unrolled version of CNF) can be generated, and the ML framework can predict the de-obfuscation time with high accuracy. Therefore, the designer can estimate security complexity of ReTrustFSM obfuscated circuit for the given external secrecy without actually performing BMC attack.

B. FORMAL SECURITY EVALUATION

The authors of [57] and [58] provided formal definitions of the security posed by logic locking. The *indistinguishable logic locking* (IND−LL) requires that the adversary cannot distinguish between a locked and an unlocked (original) circuit [57]. The *simulation secure logic locking* (SIM−LL), entails that the adversary cannot leak any additional information about the original circuit from the locked one than an oracle simulator. Later, based on these definitions, the authors of [57] and [58] formally proved that universal circuits can achieve IND−LL secure logic locking. However, for a circuit of size n, the universal circuit requires a key of length $O(n \log n)$ [57] which significantly blows up the locked circuit size and suffers greatly from overhead. Therefore, to achieve IND−LL secure logic locking in practise, we contemplated two corollaries from [57] and [58].

Corollary 1: The IND−LL secure logic locking with smaller key size (i.e., reasonable overhead) can be achieved by the inclusion of a cryptographic assumption [57].

Corollary 2: Increasing the information leaked by the locked circuit can improve the locking overhead while provable security is still guaranteed [58].

Linear feedback shift registers (LFSR) are pseudo-random number generators vastly used for light-weight cryptographic applications. According to the Berlekamp-Massey [59], an adversary must observe at least $2n$ consecutive outputs from an LFSR to reconstruct the seed (explicit external secrecy of ReTrustFSM). However, in the case of ReTrustFSM, as explained in Section III-D, circuit elements dedicated to the state encoding, LFSR, and counter are protected by the scan obfuscation method to immobilize any chance of leakage from the LFSR. Moreover, LFSR defining the state encoding can drive the preFSM to any possible state (2^{n-1} as the best case), as shown in Fig. 7. Therefore, ReTrustFSM with its explicit external secrecy can achieve IND−LL security.

C. STRUCTURAL AND TOPOLOGICAL ANALYSIS

The structural and topological analysis attempts to reverse-engineer the FSM automatically from the obfuscated flattened gate-level netlist [31] as discussed in Section II-D. The adversary requires access to the locked gate-level netlist, and the attack is performed based on a set of structural and topological analyses. In some cases the adversary may require access to the activated chip as well for validating the attack. In the case of FSM obfuscation, structural and topological analysis means that the attacker can find the state registers in locked netlist. This step is a prerequisite for re-constructing the STG. The authors of [31] utilized fundamental properties of the state registers, e.g., register type, strongly connected components, combinational logic feedback paths, dependency, and control behavior metric to distinguish the state registers. This approach aided them in reverse-engineering several state-of-the-art FSM obfuscation schemes. One of the key observation from Fig. 2 as discussed in Section II-D is that, most of the existing FSM obfuscation methods (e.g., HARPOON [30], dynamic state deflection [33], active metering [5]) do not possess any transition from the original states back to the preceding obfuscation states and create a bipartite graph. Hence, the original states do not form a strongly connected component with the obfuscated states according to Tarjan's algorithm [45]. Therefore, obfuscated states along with their associated state registers can be partitioned from the original state registers. ReTrustFSM thwarts such structural and topological analysis-based attacks in the following manner.

- The existing FSM obfuscation methods insert the obfuscated states before the original STG starts and loosely connect them to the original states. However, ReTrustFSM inserts obfuscated states (S_0^{Obf}, S_1^{lfsr}, and S_2^{cnt} in Fig. 2) deeply rooted into the original FSM at the RTL by following Lemmas 1, 2, and 3. Therefore, in the ReTrustFSM obfuscated circuit, obfuscated states are tightly entangled with the original state machine.
- It can be observed from Fig. 2 that the proposed ReTrustFSM inserts obfuscated states that possess multiple back and forth transitions with the original states. Hence, the obfuscation states creates a strongly connected component with the original states according to Tarjan's algorithm [45]. Therefore, ReTrustFSM does

[7] ≈90% accuracy for the prediction of the attack time would be more than enough for the designer to decide about the value of external secrecies (e.g., the difference between actual and predicted would be in the order of a few days while the total de-obfuscation time is in order of few years).

not leave any structural footprint in the obfuscated design that can be exploited by the topological analysis [31].
- The obfuscation circuitry (LFSR, and down-counter) that performs encoding of the postFSM (S_3^N and optionally S_4^N in Fig. 2), are initialized by the states of the encFSM ($S_0^{obf} \rightarrow S_1^{lfsr} \rightarrow S_2^{cnt}$ in Fig. 2) after the traversal of preFSM. Therefore, the obfuscation circuitry in ReTrustFSM are a part of the FSM state registers which makes the search space even larger for structural analysis.
- As implicit secrecy is integrated as part of the operational mode of the encFSM, to mitigate the design vulnerability against the functional analysis-based structural attack [13] (for input pattern checking), stealthy opaque predicate [49] based dynamic secrecy generation has been deployed in ReTrustFSM construction.

Due to the points mentioned above, structural and topological analysis-based attacks [31] are ineffective on the ReTrustFSM obfuscated circuit. It is noteworthy that the circuit will be dysfunctional even if the attacker can reverse engineer the state machine and even distinguish the obfuscated state registers without knowledge of the correct encoding of the original states at RTL. In other words, removing the encFSM and black holes from the STG does not allow the adversary to reconstruct the original FSM, as the encoding of lockedFSM depends on traversal in encFSM.

D. REMOVAL ATTACK

In case of a removal attack on ReTrustFSM obfuscated circuitry (LFSR/counter), since a dependency between the obfuscated circuitry and the original functionality is built, the removal of logic dedicated for the obfuscated circuitry results in losing the correct functionality. In such cases, the transition(s) between preFSM to postFSM gets lost, and similar to what demonstrated in Fig. 7, it returns back to the `default` state, S_{init}^N. The following is some of the reasons why removal attack does not work on ReTrustFSM:

(i) With access to the netlist and having the basic information of the obfuscation method, the attacker can identify the LFSR/counter in the design and remove them. However, after removing LFSR/counter, the register(s) dedicated for state encoding of the lockedFSM state(s) become *unconnected* (no more connection to LFSR/counter). So, this register, which is part of the FSM circuitry, will be removed during synthesis optimization. This will be followed by removing more parts of the design that are directly connected to this state register. Then with the removal of part of the actual circuit, the removal ultimately corrupts the functionality.

(ii) The functionality of all three obfuscation FSM states is clearly defined and based on the threat model, the adversary has access to the locked gate-level netlist. However, in oracle, a sub-set of registers, including state registers, and LFSR/counter registers are not directly available. So, the adversary has to apply stimuli to primary inputs and observe primary outputs. In this case, there exists no connection between PI/PO pairs and the specification of internal registers as they are not observable. The adversary cannot distinguish between the state registers vs. the other registers. So, the removal of these three states needs a prerequisite in which the adversary must apply methods like Tarjan to see if they are able to distinguish between registers. To the best of our knowledge, there is no mechanism that can distinguish between these registers by 100% (guaranteed) success ratio [31], [38]. When the adversary cannot distinguish state registers, applying input patterns for re-constructing the whole STG, and then based on the whole STG (for detecting and removal of three obfuscation states) becomes clueless.

E. FUNCTIONAL CORRUPTIBILITY-GUIDED ATTACK

As discussed in Section II-C, functional corruptibility-guided SAT attack (Fun-SAT) [29] is an extended version of BMC (SAT) attack, in which the required minimum numbers of unrolling for de-obfuscation is estimated based on output corruptibility. Then, based on the number of unrolling, the satisfiability is invoked. Hence, to perform this attack, the adversary must have PI/PO access to the (1) activated chip (oracle), and (2) gate-level netlist. Fun-SAT attack [29] analyzes the keyless FSM obfuscation method [30] to estimate the minimum number of unrolling such that the functional equivalency between the unrolled circuit and the original sequential circuit is 100%. Once Fun-SAT deciphers the minimum number of required unrolling, it unrolls the obfuscated circuit and performs the regular SAT attack [9] to extract the secret enabling key ($\{I_{en}, I_{auth}\}$).

Although Fun-SAT efficiently works on a set of FSM obfuscation techniques that rely on implicit external secrecy, e.g., HARPOON [30], the corruptibility analysis fails to build the functional equivalency between the unrolled circuit and the original sequential circuit once explicit secrecy is in place, as without the correct external secrecy, there exists no equivalency between the obfuscated and the original circuit. Since ReTrustFSM is a hybrid approach with a strong correlation between implicit and explicit secrecy, Fun-SAT fails to break ReTrustFSM obfuscated circuit because the corruptibility analysis does not provide any advantage. The attack eventually reduces to a regular SAT or BMC attack, the analysis and results of which are presented in Section IV-A and V. Additionally, in ReTrustFSM, the obfuscated states are deeply twisted as an intermediate region with the original FSM. So, any pre-processing like corruptibility analysis that helps to reveal the required number of unrolling is not applicable on ReTrustFSM obfuscated circuits.

F. ORACLE-LESS MACHINE-LEARNING ATTACK

Oracle-less ML-based attacks on logic locking, e.g., SWEEP attack [52], rely on re-synthesizing the locked gate-level netlist with additional key-gates and then learning the optimization algorithm of the EDA tool. So, the adversary only require access to the locked gate-level netlist for applying the key value followed by re-synthesis. With the knowledge of

the re-synthesis-based optimization algorithm, the SWEEP attack then tries to revert to the original gate-level netlist before locking. These ML-based attacks [52], [60] focus on tracking down certain key gates, e.g., XOR, XNOR, or MUX [6], [10]. Therefore, locking methods that depend on specific key gates are extremely susceptible to these oracle-less attacks. Additionally, almost all ML-based attacks build their knowledge based on *guess and check* method, in which the guessing is always around the value for the explicit secrecy. Hence, in cases where implicit secrecy is in place, such attacks become ineffective.

ReTrustFSM resists all existing ML-based attacks for three reasons: (1) the obfuscation circuitry is inserted into the original design at the RT-level, which gets blended with the original design, unlike most of the existing obfuscation methods that integrate with the gate-level design. (2) Establishing an ML-based attack requires explicit guess and processing of the secret, and the training is accomplished on the post-processing analysis after constraining the explicit secret. However, we use implicit secrecy tightly correlated with the obfuscated sub-circuitry with explicit secrecy. (3) ReTrustFSM, which relies on the composition of implicit-explicit secrecy, has nothing to do with any specific gate type. Furthermore, the LFSR and counter circuitry does not rely on any specific gate type, which is a key requirement of oracle-less ML-based attacks [52], [60]. Hence, the oracle-less ML-based attacks are useless against ReTrustFSM. We performed an oracle-less ML attack [52] on the ReTrustFSM obfuscated netlist discussed in Section V.

G. TIMING SIDE CHANNEL ATTACK

ReTrustFSM focuses on the control behavior of the circuits (implemented by the FSM) and the obfuscation part (known to the adversary). By identifying timing side channel information, such as switching activity of POs and observable points, the adversary can identify obfuscation part traversal timing. Because this timing information is determined by the counter value (explicit external secrecy), the attacker can have a close guess. A partial external secret can be revealed by testing adjacent values to the guessed secret. This threat can be avoided by following REQ_4. In ReTrustFSM, we follow a simple rule in which we target the handshaking signals at the PO. For instance, in designs with valid/ready handshaking signals, the PO can be masked to 0 (no switching activity) while there is no valid data. However, we void the masking while we apply ReTrustFSM. Additionally, for signals like valid, we generate false positive valid for incorrect key values making it difficult to guess the duration of the obfuscation period. Moreover, there exist techniques that, with almost zero overhead, make the timing-side-channel attack impractical. For instance, a decoy counter can be used for a fake count once the obfuscation part configures. So, the adversary observes that for $c_1 + c_2$ clock cycles, the idleness is at the highest level, implying that the counter value (explicit external secrecy) is $c_1 + c_2$. However, c_2 corresponds to the number of clock cycles of the fake count. So, the adversary is deceived into guessing a wrong initial counter value.

V. EXPERIMENTAL RESULTS

This section evaluates ReTrustFSM by implementing in a set of ten different ITC'99 [61] and microprocessor benchmark circuits. We perform BMC attack [41] by varying the secret LFSR seed and count. We also assess ReTrustFSM against the oracle-less ML-based attack [52]. To investigate the industrial scalability of ReTrustFSM, we analyze power, performance, and area overhead. Finally, we check the corruptibility metric to test the quality of output confusion.

A. EXPERIMENTAL SETUP

To comprehensively evaluate the performance of ReTrustFSM, we implemented proposed hybrid FSM obfuscation method on benchmarks from ITC'99 [61], μcontrollers, and SoCs. Table 2 presents the specification of the benchmark circuits, e.g., the number of gates, input/output ports, flip-flops, state, state transitions, key sizes, and candidate states. The chosen benchmarks varies in sizes, number of states, transitions, and ports to evaluate ReTrustFSM for different corner cases. For each benchmark, we implemented ReTrustFSM for two different key sizes to exhaustively investigate its security and implementation overheads. We have chosen the key sizes based on the number of flip-flops in the original design. Table 2 also reports the number of candidate states available in the original FSM, based on the Lemma 1. For each benchmark, multiple states satisfied the candidate state selection requirements, and we picked the one that meets all three Lemmas. The effectiveness of Lemma 1, 2, and 3 in increasing BMC attack complexity is experimentally proved in Section V-B. All of our experiments are performed in a 32 core 2.6 GHz CPU and 64 GB memory. ReTrustFSM utilized Synopsys Design Compiler, Cadence JasperGold, nuXMV [56], ABC [55], and Python 3.8 for the experiments.

To implement ReTrustFSM, for lockedFSM states (states whose encoding is determined and calculated by the LFSR), we re-used the static (existing) encoding as the target state encoding. So, the explicit external secrecies are defined in a way that the correct calculations lead to the exact static (previously assigned) encoding for those specific states. Please note that it can be easily swapped with another state, making sure that the sequential ordering does not give any clue to the adversary for revealing the correct encoding. Since ReTrustFSM inserts additional (three) states for the encFSM part, it might be possible that we need to extend the state encoding size by at least $+1$. For instance, the RISCV benchmark has already 264 different states, meaning that with binary encoding, it needs an 8-bit register for state encoding values. So, by adding at-least three additional states (encFSM), it becomes 267 states that need a 9-bit register for state encoding. In such cases, we extended the whole encoding mapping into a larger bit size. Please note that since all obfuscation procedure has been done at RTL,

TABLE 2. Benchmark circuits specification and candidates per benchmark.

Circuit	#Gates	#FFs	#PI/PO	States	Transitions	key1, key2	Candidates*	Candidates* per States
Fib	589	100	12/8	5	7	8, 16	3	60%
B05	608	34	3/36	5	9	8, 16	4	80%
B07	382	51	3/8	9	13	8, 16	6	66.7%
B10	172	17	13/6	11	14	8, 16	7	63.6%
B11	366	30	9/6	10	16	8, 16	3	30%
B15	6931	447	37/70	18	42	8, 16	3	16.7%
Sayeh	5585	564	3/20	11	101	16, 32	4	36.4%
MSP430	11561	787	14/8	34	71	16, 32	4	11.8%
MIPSR2K	13846	548	11/272	20	39	16, 38	17	85%
RISCV	20251	1459	252/260	264	531	16, 32	260	98.5%

*Candidates are those states that meet the requirements defined by the Lemmas (Lemma 1).

extending bit size can be applied easily through the RTL representation.

B. BMC ATTACK RESULTS

In this subsection, we first experimentally validate the effectiveness of Lemma 1, 2, and 3 in the candidate state selection to increase the BMC attack complexity, as discussed in Section III-B. For the example demonstrated in Fig. 4(a), three states, S_2^N, S_3^N, *and* S_4^N, meet the Lemma 1. State S_2^N does not satisfy (i.e., not the best selection) Lemma 3 while S_4^N satisfies (i.e., the best selection) Lemma 3 but not Lemma 2. Therefore, state S_3^N is the best choice for obfuscation (deeper than S_2^N and the best connection with datapath). To validate this statement experimentally, we obfuscate the FSM shown in Fig. 6(a) with ReTrustFSM for four different cases where for the first three cases, we selected one state from S_2^N, S_3^N, *and* S_4^N as a candidate state. For the fourth case, we selected all the states mentioned as candidates. We perform a BMC attack on the ReTrustFSM obfuscated designs for all four cases and report the attack results in Table 3. The attack time is maximum when the candidate state has the best cover for all three Lemmas (meeting Lemma 1 and the best options for Lemmas 2 and 3). Please note that, though the number of unrolling is more when S_4^N is selected as a candidate state (since it is a deeper state), the less datapath corruptibility of state S_4^N helps BMC attack converge within fewer iterations/time. Based on the above observation, for selecting the best candidates (lockedFSM) for each benchmark, we start from the deepest state that satisfies Lemma 1 and go backward till we find the highest level of corruption on the datapath. So, for all the results demonstrated in this Section, we select one lockedFSM that meets Lemma 1, has a high corruption on the datapath, and is deeper in the FSM.

Fig. 11 demonstrates the resiliency of ReTrustFSM against oracle-guided BMC attack on all benchmarks of Table 2. We utilized `Cadence JasperGold` as the model-checking engine for BMC attack [41]. A timeout margin of seven (7) days is considered for all the attacks (marked by the horizontal dotted red lines in Fig. 11). We varied the key size based on Table 2 and selected candidate state based on Lemma 1, 1, and 1. We varied the count value from 1 to 64 until a timeout was encountered. All the results in Fig. 11 were averaged over three different trials.

TABLE 3. Effect of candidate state on BMC attack complexity.

Candidate State	Satisfies Lemma	Number of Iterations	Number of Unrolling	Time (s)
S_4^N	1, 3	21	30	1199
S_2^N	1, 2	36	28	2760
S_3^N	1, 2, 3	37	29	3769
S_2^N, S_3^N, S_4^N	1, 2, 3	34	28	4241

In the case of a BMC attack, breaking the ReTrustFSM means that the BMC attack can generate a set of $\{dis\}$es that reveals the explicit secrecy. But the implicit secrecy is still unknown to the user because the value of input patterns required for correct traversal of encFSM is not revealed in the $\{dis\}$es that BMC finds. It is unknown to the adversary that for a $dis_i = \{i_1, i_2, \ldots, i_n\}$, what part of dis_i as $\{i_c, i_{c+1}, i_{c+2}\}$ is for the clock cycles in which the FSM traversing $S_0^{obf} \rightarrow S_1^{lfsr} \rightarrow S_2^{cnt} \rightarrow S_0^{obf}$. Hence, to fully break ReTrustFSM, the BMC attack needs to be followed by a process similar to the structural+functional attack so that the attacker can construct the STG of the ReTrustFSM obfuscated circuit. In this case, the adversary can track $\{dis\}$es on the extracted STG to find the part dedicated for the traversal of encFSM as the implicit secrecy of the ReTrustFSM. As mentioned in IV-C, the complexity of structural+functional is very high.

Assuming that the adversary can perform the structural attack that is required for the clock cycles corresponding to the implicit secrecy, Fig. 11 shows the JasperGold-based BMC attack still can not effectively crack the ReTrustFSM. Although for smaller circuits with small count values (e.g., $\leq$32), the BMC attack broke the obfuscation, we observed that the attack suffered from scalability issues for larger count values in even small circuits and circuits containing more flip-flops and states. For example, the attack failed to break `b05` locked with 16-bit seed and the count value of 72, facing a sudden increase in the required number of unrolling. One can increase the key size by using

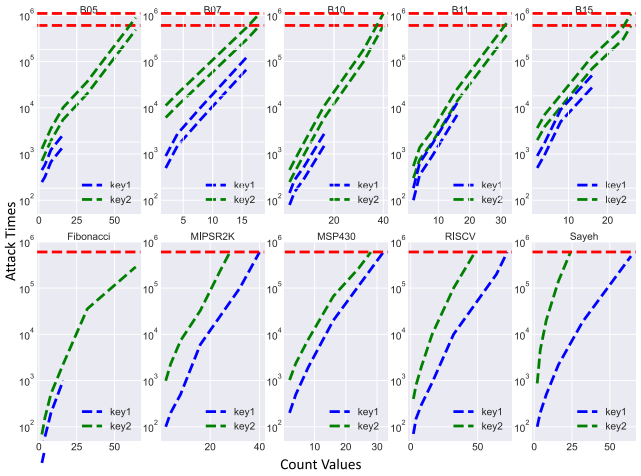


FIGURE 11. BMC attack time (in log scale) for Varying Count Values. Timeout is 7 days and all the attack results are averaged over three different trials.

TABLE 4. Effect of output frequency on BMC attack complexity.

Circuit	Low Output Freq.			High Output Freq.		
	Iterations	Unrolling	Time	Iterations	Unrolling	Time
Fib	33	29	2842	44	29	4241
B10	30	28	1093	33	28	1751
B11	100	27	20739	111	27	27071

longer LFSR/counter circuitry, exponentially increasing the resiliency against pruning-based (brute-force) attacks.

C. IMPACT OF OUTPUT FREQUENCY ON BMC ATTACK

To consider REQ_4, we briefly touched on the circuit's behavior on the BMC attack complexity. In the case of SAT/BMC-guided I/O query-based attacks, as explained in Section II-C, the attack method generates a DIP from the miter circuit (SATC in Fig. 1) and adds the learned I/O pair as a constraint. The larger this constraint gets, the more difficult it becomes for the satisfiability algorithm to converge. If the output gets updated more frequently (without impacting the original functionality), the constraint grows faster, requiring more time/effort for de-obfuscation. This exciting feature is experimentally validated in Table 4. Here we modified (increased) the output frequency of three such benchmarks that require a valid/done/ready signal-based handshaking to notify the user regarding a legitimate output. We performed BMC attack [41] on the ReTrustFSM obfuscated design of original benchmarks and modified (more frequent output) benchmark and reported the attack time, number of iterations, and required unrolling depth in Table 4. While the number of needed unrolling remained consistent, the number of iterations and attack time increased in designs with more frequent outputs for all benchmarks.

D. ML-BASED ATTACK RESULTS

We also performed ML-based SWEEP attack [52] on all benchmarks of ReTrustFSM. However, the oracle-less ML-based attack failed to extract any secret from the ReTrustFSM obfuscated circuit. As discussed in Section IV-G, the obfuscation circuitry is inserted into the original design at the RT-level. The oracle-less ML attack works on a re-synthesized netlist. The obfuscation circuitry gets blended with the original design during the transformation, mapping, and optimization from RTL to gate-level design. The keyless obfuscation component in ReTrustFSM is based on a specific input sequence and is not dependent on any particular gate type. In addition, the LFSR and counter circuitry relies on no specific gate type essential for learning by ML-based attacks [52], [60].

E. OVERHEAD ANALYSIS

Table 5 displays the implementation overhead of ReTrustFSM in terms of area, power, performance, and testability. For the original design, we report the actual area (um^2), power (mW), critical path delay (ns), and test coverage (%). For the ReTrustFSM obfuscated designs, we report the overheads for two different key sizes. As discussed previously, ReTrustFSM obfuscation can be deployed in any SoC scenario with minimal impact in terms of PPA overhead. For many cases in Table 5, we observe minor improvement in terms of delay due to the non-determinism of optimization algorithms. For smaller benchmarks (from Fib to b11 where the number of gates is $\leq$1000), we observed a large area and power overhead due to the gate count incurred by LFSR and counter circuitry. However, as ReTrustFSM does not scale with the design size, the area, and power overhead dropped $<$5% for larger benchmarks. ReTrustFSM inserts additional sequential circuitry in the original design. Therefore, regardless of the seed size, count value, and circuit size, ReTrustFSM has almost no impact on test coverage compared to the original design, as demonstrated in Table 5. Please note that the delay overhead is the timing paths' critical delay overhead. Adding extra states does not necessarily affect the timing delay negatively. However, in terms of throughput, adding extra states (obfuscation states) may affect the performance. This is because one extra state (S_0^{obf}) will be always part of the FSM even after activation. It is worth mentioning that it is possible for the design team to select the candidate in a way that the extra state is out of the controlling of the execution paths. In this case, the extra state will not affect stream of data, and the throughput will be less affected. In general, for all circuits of Table 2, we observe no more than 1% throughput degradation as the obfuscated states added in non-critical (data-oriented) paths.

F. CORRUPTIBILITY ANALYSIS

In Table 6, we present the *output corruption*, the security metric that indicates the final confusion created by an obfuscation method, along with the *data-path corruption*. For each

TABLE 5. PPA (Post-layout) overhead and test coverage of the ReTrustFSM obfuscated circuit.

Circuit	Original				Overhead							
					Area(%)		Delay(%)		Power(%)		Test(%)	
	Area (um^2)	Delay (ns)	Power (mW)	Test (%)	key1	key2	key1	key2	key1	key2	key1	key2
Fib	820	8.59	0.07	99.30	39.36	48.76	0.2	-0.46	35.22	45.23	99.51	99.65
b05	2148	7.79	0.15	99.52	13.91	25.65	-0.8	-0.51	12.21	23.45	99.47	99.34
b07	977	8.62	0.07	99.78	18.46	41.12	-0.4	-0.23	16.32	39.67	99.21	99.57
b10	438	8.92	0.03	99.60	48.22	57.87	3.2	0.55	45.76	55.32	99.32	99.23
b11	1262	8.48	0.01	98.29	16.96	35.23	-0.2	-0.83	13.97	32.18	98.79	98.11
b15	15321	5.89	1.01	99.28	2.24	4.89	-1.8	0	1.51	4.56	98.99	98.92
Sayeh	5451	2.35	0.35	99.13	1.71	3.62	2.1	0	1.45	3.57	98.95	98.87
MSP430	20099	4.44	1.39	98.80	2.07	3.51	0.1	-0.90	1.89	3.23	98.44	98.35
MIPSR2K	22376	1.17	2.01	98.56	0.75	2.95	0.2	-1.73	0.78	2.89	98.43	98.23
RISCV	44881	2.02	3.78	97.53	0.89	1.92	0.07	0.2	0.68	2.01	97.57	97.56

benchmark of Table 2, we applied 2^n number of input patterns (i), ten different incorrect keys (k), captured the generated output from the obfuscated circuit ($c_{obf}(i,k)$), and compared with the one generated from the original circuit ($c_{org}(i)$). Table 6 reports two output corruption metrics, *error rate* and *HD rate* defined in Equ. 5 and 6. According to Equ. 5 and 6, *error rate* captures the ratio of incorrect outputs to the number of generated outputs, and *HD rate* rate denotes the number of incorrect output bits.

$$\forall i, \exists k \neq k_c : ER = \frac{No.\ of\ \{c_{obf}(i,k) \neq c_{org}(i)\}}{Total\ No.\ of\ c_{org}(i)} \quad (5)$$

$$HD\ Rate = \frac{HD(c_{obf}(i,k), c_{org}(i))}{\|c_{org}(i)\|} \quad (6)$$

Table 6 reports the average of both output corruption metrics for ten different incorrect keys. It is noticeable that for smaller circuits (Fib to b11), the *error rate* is 100% which drops to around 85% for larger ones (e.g., MIPSR2K) due to their increased size. We also noticed a similar trend for *HD rate*. Table 6 also reports the percentage of data-path circuitry corrupted by ReTrustFSM obfuscated control circuitry. To identify this corruption metric of ReTrustFSM, we utilize the `analyze_datapath` command in the `Synopsys Design Compiler` tool to extract the data-path circuitry from the synthesized netlist. Later we use fan-in and fan-out-based structural analysis to identify the percentage of data path corrupted by obfuscated FSM. From Table 6 is can be observed that ReTrustFSM can corrupt $>$ 95% of the data-path circuitry, except in the case of `MSP430`. This high data-path corruptibility validates the efficiency of ReTrustFSM in corrupting functionality for incorrect keys.

G. FUNCTIONAL VERIFICATION OF ReTrustFSM

To verify that the original functionality of the design is intact after obfuscating with ReTrustFSM, we performed a formal equivalency checking between the original RTL and the ReTrustFSM obfuscated RTL using `Synopsys Formality` tool which uses a static path-based method to determine if two versions of the design are functionally equivalent. We provided the original design RTL as the `reference` design and the ReTrustFSM obfuscated RTL as the `implementation` design. The correct unlocking key (seed/count pair) value was applied as a constraint to the verification flow using `set_constant` command. As the formal equivalency checking method utilize a path-based analysis, any additional flip-flops in the `implementation` design will create extra timing paths and eventually will mismatch with respect to the `reference` design. The LFSR/counter circuitry of ReTrustFSM inserts additional flip-flops in the `implementation` design. Therefore, we utilize `set_dont_verify` command to exclude the added flip-flops from analysis and perform the equivalency checking for the rest of the original circuitry between the `reference` and `implementation` design. All the ReTrustFSM obfuscated benchmarks from Table 2 passed verification, and their functionality matched with the corresponding original design.

TABLE 6. Corruptibility analysis of the ReTrustFSM (Percentage).

Circuit	Datapath Corrupt	PO Corrupt HD Rate	PO Corrupt Err Rate	Circuit	Datapath Corrupt	PO Corrupt HD Rate	PO Corrupt Err Rate
b05	100	63	100	Fib	100	79	100
b07	100	78	100	Sayeh	100	57	95
b10	95	74	100	MSP430	60	37	90
b11	100	71	100	MIPSR2K	100	39	85
b15	96	46	95	RISCV	100	31	90

H. COMPARISON WITH THE PRIOR ART

In Section II-D, we discussed the state-of-the-art FSM-based obfuscation techniques, their mechanisms, vulnerabilities, and attacks. This subsection compares the ReTrustFSM to the prior art in performance, implementation overhead, security, and attack complexity. HARPOON [30] and active metering [5] were proposed as one of the very first FSM

TABLE 7. Comparison of ReTrustFSM with the existing FSM obfuscation methods.

FSM Obfuscation	Overhead			Security
	Area	Power	Delay	
HARPOON [30]	High	High	Low	Vulnerable [28, 29, 31]
Interlocking [32]	Low	Low	High	Vulnerable [29, 31]
State Deflection[33]	Low	High	High	Vulnerable [29, 31]
Active Metering [5]	High	High	High	Vulnerable [29, 31]
JANUS [34, 35]	Low	Low	Low	Unprotected scan (not evaluated)
ReTrustFSM	Low	Low	Low	–

obfuscation methods that not only suffer from the high area and power overhead [33] but also vulnerable to several attacks [28], [29], [31]. While Interlocking [32] and state deflection [33] slightly improved the area overhead, both of these methods are still vulnerable to {structural+functional} attack [31] and functional corruptibility guided SAT attack [29]. JANUS [34], [35] inscribes the obfuscation states in the original STG. However, as discussed in Section III-D, leaving the scan chain unprotected makes the FF configuration (D/T select) vulnerable to scan-based leakage attack [50]. Additionally, as explained in Section II-E, the security of JANUS can be compromised by the invocation of I/O query-based attacks. On the contrary, ReTrustFSM pushes the obfuscation states deeper into the STG, which makes the proposed technique resilient against the {structural+functional} attack [31] and Fun-SAT attack [29]. Moreover, the efficacy of ReTrustFSM against SAT/BMC guided algorithmic attacks [28] is discussed in detail in Section IV-A and experimentally validated in Section V-B across a wide range of benchmarks. Additionally, the implementation and performance overhead of ReTrustFSM is negligible, as experimentally validated in Section V-E and Table 5.

VI. CONCLUSION

In this paper, we proposed ReTrustFSM, a novel, and comprehensive FSM obfuscation solution at the Register-Transfer (RT) level, which not only allows the designer to have more control and concentration on the semantics of the design, with expanding the nature of the threat models, it also provides robustness against a wider range of threats. ReTrustFSM is mainly a hybrid solution of highly correlating and integrating both explicit (key-based) and implicit (keyless) secrecy in a cycle-accurate fashion. Using this hybrid solution has made ReTrustFSM fully resilient against the state-of-the-art attacks on logic locking, particularly those targeting the FSM obfuscation. In ReTrustFSM, we also engaged a state encoding that is the outcome of a specific sequence of operations at both obfuscated and original mode. Additionally, we showed how the security complexity of the ReTrustFSM obfuscated circuit could be estimated based on its CNF features predictive of SAT complexity. We also verified the robustness of ReTrustFSM against structural+functional attacks on obfuscated FSMs, oracle-less ML-based attacks , and functional corruptibility guided SAT-based attacks. Our experimental results further showed the robustness of ReTrustFSM against BMC-based sequential attack while the PPA overhead is low and testability is not affected by the locking approach.

REFERENCES

[1] B. Shakya, M. Tehranipoor, S. Bhunia, and D. Forte, "Introduction to hardware obfuscation: Motivation, methods and evaluation," in *Hardware Protection Through Obfuscation*. Cham, Switzerland: Springer, 2017, pp. 3–32.

[2] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Cham, Switzerland: Springer, 2011.

[3] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[4] A. B. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design IP protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252, Oct. 2001.

[5] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proc. USENIX Secur. Symp.*, 2007, pp. 291–306.

[6] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 709–720.

[7] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing scan design using lock and key technique," in *Proc. 20th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 51–62.

[8] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Advances in logic locking: Past, present, and prospects," *Cryptol. ePrint Arch.*, vol. 2022, pp. 1–10, Mar. 2022.

[9] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[10] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1069–1074.

[11] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 83–89.

[12] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr. 2020.

[13] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2514–2527, 2020.

[14] Z. Han, M. Yasin, and J. Rajendran, "Does logic locking work with *EDA* tools?" in *30th USENIX Secur. Symp. (USENIX Secur. 21)*, 2021, pp. 1055–1072.

[15] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 236–241.

[16] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. IACR Conf. Cryptograph. Hardw. Embedded Syst.*, 2016, pp. 127–146.

[17] 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. Secur.*, Oct. 2017, pp. 1601–1618.

[18] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4439–4452, Dec. 2020.

[19] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[20] D. Sisejkovic, F. Merchant, L. M. Reimann, and R. Leupers, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1716–1729, Jun. 2022.

[21] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, p. 89.

[22] J. Sweeney, M. J. H. Heule, and L. Pileggi, "Modeling techniques for logic locking," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[23] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "InterLock: An intercorrelated logic and routing locking," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[24] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 10, pp. 97–122, Nov. 2018.

[25] N. G. Jayasankaran, A. S. Borbon, A. Abuellil, E. Sanchez-Sinencio, J. Hu, and J. Rajendran, "Breaking analog locking techniques via satisfiability modulo theories," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–10.

[26] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–9.

[27] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "Timing-Camouflage: Improving circuit security against counterfeiting by unconventional timing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 91–96.

[28] S. Roshanisefat, H. M. Kamali, K. Z. Azar, S. M. P. Dinakarrao, N. Karimi, H. Homayoun, and A. Sasan, "DFSSD: Deep faults and shallow state duality, a provably strong obfuscation solution for circuits with restricted access to scan chain," in *Proc. IEEE 38th VLSI Test Symp. (VTS)*, Apr. 2020, pp. 1–6.

[29] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "FunSAT: Functional corruptibility-guided SAT-based attack on sequential logic encryption," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2021, pp. 1–11.

[30] R. 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.

[31] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the difficulty of FSM-based hardware obfuscation," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 15, pp. 293–330, Aug. 2018.

[32] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in *Proc. 8th Annu. Cyber Secur. Inf. Intell. Res. Workshop*, Jan. 2013, pp. 1–8.

[33] 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.

[34] L. Li, S. Ni, and A. Orailoglu, "JANUS: Boosting logic obfuscation scope through reconfigurable FSM synthesis," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2021, pp. 1–11.

[35] L. Li and A. Orailoglu, "JANUS-HD: Exploiting FSM sequentiality and synthesis flexibility in logic obfuscation to thwart SAT attack while offering strong corruption," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1–6.

[36] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 405–410.

[37] F. Koushanfar, "Active hardware metering by finite state machine obfuscation," in *Hardware Protection Through Obfuscation*, 2017, pp. 161–187.

[38] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[39] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "SCRAMBLE: The state, connectivity and routing augmentation model for building logic encryption," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 153–159.

[40] M. S. Rahman, R. Guo, H. M. Kamali, F. Rahman, F. Farahmandi, M. Abdel-Moneum, and M. Tehranipoor, "O'clock: Lock the clock via clock-gating for SoC IP protection," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 775–780.

[41] S. Roshanisefat, H. M. Kamali, H. Homayoun, and A. Sasan, "RANE: An open-source formal de-obfuscation attack for reverse engineering of logic encrypted circuits," in *Proc. Great Lakes Symp. VLSI*, Jun. 2021, pp. 221–228.

[42] K. Z. Azar, H. M. Kamali, F. Farahmandi, and M. Tehranipoor, "Warm up before circuit de-obfuscation? An exploration through bounded-model-checkers," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Jun. 2022, pp. 1–4.

[43] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "From cryptography to logic locking: A survey on the architecture evolution of secure scan chains," *IEEE Access*, vol. 9, pp. 73133–73151, 2021.

[44] Y. Shi, C. W. Ting, B.-H. Gwee, and Y. Ren, "A highly efficient method for extracting FSMs from flattened gate-level netlist," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 2610–2613.

[45] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jun. 1972.

[46] S. Takamaeda-Yamazaki, "Pyverilog: A Python-based hardware design processing toolkit for verilog HDL," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2015, pp. 451–460.

[47] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 5s, pp. 1–20, Dec. 2014.

[48] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[49] M. Hoffmann and C. Paar, "Stealthy opaque predicates in hardware - obfuscating constant expressions at negligible overhead," 2019, *arXiv:1910.00949*.

[50] J. D. Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Are advanced DfT structures sufficient for preventing scan-attacks?" in *Proc. IEEE 30th VLSI Test Symp. (VTS)*, Apr. 2012, pp. 246–251.

[51] S. M. Rahman, A. Nahiyan, F. Rahman, S. Fazzari, K. Plaks, F. Farahmandi, D. Forte, and M. Tehranipoor, "Security assessment of dynamically obfuscated scan chain against oracle-guided attacks," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–27, 2021.

[52] A. Alaql, D. Forte, and S. Bhunia, "Sweep to the secret: A constant propagation attack on logic locking," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2019, pp. 1–6.

[53] C. Coarfa, D. Demopoulos, A. Aguirre, D. Subramanian, and M. Vardi, "Random 3-SAT: The plot thickens," in *Proc. Conf. Princ. Pract. Constraint Program.*, 2000, pp. 143–159.

[54] E. Nudelman, K. Leyton-Brown, H. Hoos, A. Devkar, and Y. Shoham, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Proc. Int. Conf. Princ. Pract. Constraint Program.*, 2004, pp. 438–452.

[55] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[56] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The NUXMV symbolic model checker," in *Proc. Int. Conf. Comput. Aided Verification*, 2014, pp. 334–342.

[57] P. Beerel, M. Georgiou, B. Hamlin, A. Malozemoff, and P. Nuzzo, "Towards a formal treatment of logic locking," *IACR Trans. CHES*, vol. 2, pp. 92–114, Jan. 2022.

[58] E. Masserova, D. Garg, K. Mai, L. Pileggi, V. Goyal, and B. Parno, "Logic locking-connecting theory and practice," *Cryptol. ePrint Arch.*, vol. 2022, pp. 1–10, Jan. 2022.

[59] N. B. Atti, G. M. Diaz–Toca, and H. Lombardi, "The Berlekamp–Massey algorithm revisited," *Applicable Algebra Eng., Commun. Comput.*, vol. 17, no. 1, pp. 75–82, Apr. 2006.

[60] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 56–61.

[61] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test Comput.*, vol. 17, no. 3, pp. 44–53, Jul./Sep. 2000.

M. SAZADUR RAHMAN received the B.Sc. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology and the M.Sc. and Ph.D. degrees from University of Florida, under the supervision of Prof. Mark Tehranipoor. He was a design engineer in different fabless semiconductor companies for four years in industrial scale 28nm and 14nm custom ICs. He has published one book and several peer-reviewed publications in premier ACM/IEEE journals and conferences, including the Design Automation Conference (DAC), Design Automation and Test in Europe (DATE), IEEE International Test Conference (ITC), IEEE Hardware Oriented Security and Trust (HOST), Elsevier Integration, and ACM Transactions on Design Automation of Electronic Systems (TODAES). He has multiple internship experiences at Intel Corporation, where he performed FIPS 140-3 security certification and developed an automated threat model review tool for different adversary models. His research interests include IP protection and authentication, logic locking, security estimation, and CAD for security.

RUI GUO received the M.S. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, under supervision of Prof. Farimah Farahmandi. His current research interests include hardware security and trust, logic locking, and VLSI CAD.

HADI M. KAMALI received the B.S. degree from K. N. T. University, in 2011, the M.S. degree from the Sharif University of Technology, in 2013, and the Ph.D. degree from George Mason University, in 2021, all from the Department of Electrical and Computer Engineering. He is a Research Assistant Professor with the Department of Electrical and Computer Engineering, University of Florida. His research interests include hardware security, with a particular focus on exploiting IP protection techniques, design-for-trust for VLSI circuits, and CAD frameworks for security (design-for-security), in which he has numerous publications in top journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, and Design Automation Conference (DAC), with awards including nominations/recipients of the Best Paper Award in ISVLSI'20, ICCAD'20, ICCAD'21, IEEE DCAS 2020, and HOST 2022.

FAHIM RAHMAN received the B.S. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Bangladesh, the M.S. degree in electrical and computer engineering from the University of Connecticut, USA, in 2015, and the Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA, in 2018. He is currently a Research Assistant Professor with the Department of Electrical and Computer Engineering, University of Florida. His research is sponsored by SRC, AFOSR, AFRL, DARPA, Cisco, TI, and NIST. His current research interests include hardware and cybersecurity and trust, including electronic supply-chain security, CAD for security and automatic assessment, and hardware-assisted cybersecurity. He is a member of ACM.

FARIMAH FARAHMANDI (Member, IEEE) received the B.S. and M.S. degrees from the Department of Electrical and Computer Engineering, University of Tehran, Iran, in 2010 and 2013, respectively, and the Ph.D. degree from the Department of Computer and Information Science and Engineering, University of Florida, in 2018. She is an Assistant Professor with the Department of Electrical and Computer Engineering, University of Florida. Her research has been sponsored by SRC, AFRL, DARPA, and Cisco. Her research interests include design automation of System-on-Chips and energy-efficient systems, formal verification, hardware security validation, and post-silicon validation and debug. Her research has resulted in two books, seven book chapters, and several publications in premier ACM/IEEE journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, Design Automation Conference (DAC), and Design Automation and Test in Europe (DATE). She is a member of ACM. Her research has been recognized by several awards, including IEEE System Validation and Debug Technology Committee Student Research Award, the Gartner Group Info-Tech Scholarship, a nomination for the Best Paper Award in ASPDAC 2017, and the DAC Richard Newton Young Student Fellowship. She is currently serving as the Founding Director for the Florida Institute for Cybersecurity Research (FICS). She has served for many technical program committees as well as organizing committees of premier ACM and IEEE conferences.

MARK TEHRANIPOOR (Fellow, IEEE) is currently the Intel Charles E. Young Preeminence Endowed Chair Professor of cybersecurity with the University of Florida, where he is currently serving as the Chair for the Department of Electrical and Computer Engineering (ECE). His current research interests include hardware security and trust, supply chain security, the IoT security, VLSI design, and test and reliability. He is a fellow of ACM, a Golden Core Member of IEEE CS, and a member of ACM SIGDA. He was a recipient of a dozen of the Best Paper Awards and nominations, as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 AFOSR MURI Award. He received the 2020 University of Florida Innovation of the year as well as the Teacher/Scholar of the Year Awards. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) and IEEE International Conference on Physical Assurance and Inspection of Electronics (PAINE). He serves on the program committee of more than a dozen leading conferences and workshops. He has also served as the Program and General Chair for a number of IEEE and ACM sponsored conferences and workshops (HOST, ITC, DFT, D3T, DBT, NATW, and more). He served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTERS, *JETTA*, *JOLPE*, *TODAES*, *IEEE Design & Test Magazine*, and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. He is currently serving as a founding Editor-in-Cheif for *Journal on Hardware and Systems Security* (HaSS).