Practical Implementation of Robust State-Space Obfuscation for Hardware IP Protection

Md Moshiur Rahman, *Member, IEEE*, and Swarup Bhunia, *Senior Member, IEEE*

***Abstract*— Hardware obfuscation is a design transformation technique that transforms a design to protect its confidentiality against untrusted parties. In particular, it aims at protecting proprietary hardware intellectual property (IP) blocks against reverse engineering (RE), piracy, and extraction of design secrets. A wide array of existing works on hardware obfuscation have demonstrated its security promises and theoretical robustness against diverse attacks. However, these techniques lack in: 1) scalability to large commercial-scale designs; 2) ease of integration with existing electronic design automation (EDA) tool flow; 3) ability to protect against emergent attack modes, such as structural analysis-based attacks; and 4) ability to efficiently trade off security with design overhead. The latter requires an effective metric to quantify robustness against RE attacks. In this article, we introduce a practical state-space obfuscation algorithm and associated automation tool, `ProtectIP`, that address the above shortcomings. The algorithmic steps have polynomial complexity and are scalable for large designs. We have developed a complete EDA tool flow that integrates `ProtectIP`. We show that exponential resistance can be achieved against all known RE attacks while incurring modest area overhead (24% on average) with negligible impact (maximum 5% overhead) on critical-path delay. We quantify the level of achieved security using metrics and show that an intelligent attacker with partial knowledge of the obfuscation process has a minimal success rate (a probability of 0.33) in RE attacks.**

***Index Terms*— Finite-state machine (FSM), hardware intellectual property (IP), hardware obfuscation, ProtectIP, state-space obfuscation.**

I. Introduction

SYSTEM-ON-CHIP (SoC) design using reusable hardware intellectual property (IP) is common in the semiconductor industry. Integration of these tested, verified, and reusable IP modules to design a complex system incurs significantly lower cost and time-to-market [1]. Hence, SoC integrators prefer using these proprietary IPs to build their systems. As part of this process, IP vendors send their IPs to the SoC design houses, fabs, and testers, and each of these untrusted entities may need white-box access (gate-level netlist, layout, and so on) to the IPs. Hardware obfuscation is a comprehensive and effective solution to protect IPs from being affected by reverse engineering (RE), cloning, Trojan insertion, degradation in performance of the IP, and loss of revenue for the IP vendors. Hardware obfuscation is a methodology that transforms the IP's embedded finite-state machine (FSM) so that the functionality remains obfuscated by default. Upon provisioning specific keys, correct functional behavior can be reproduced from the design, meaning that the design can be deobfuscated and used for its intended purpose. Although several obfuscation schemes have been proposed in the literature under two categories: combinational logic locking [2], [3], [4] and sequential logic obfuscation [5], [6], [7], [8], [9], unfortunately, obfuscated hardware IPs contain undetected and severe security flaws inherent to the current state-of-the-art obfuscation algorithms and fail to prevent the IP infringement issues we have mentioned earlier. Existing methodologies are vulnerable to several structural [8], [10], [11] and functional attacks [12], [13], [14] due to their limitations in achieving a high level of RE attack complexity.

In this article, we present a scalable state-space obfuscation methodology that is amenable to automation and integration into the commercial electronic design automation (EDA) tool flow. It incorporates an exponential increase in state-space reachability of an IP by inserting nonfunctional states in the FSM of the IP. We implement the methodology into a computer-aided design (CAD) tool, referred to as `ProtectIP`, and interface it with existing EDA tools. In a gate-level netlist, an FSM is a register or group of flip-flops that controls the operation of the digital logic by traversing through several states. `ProtectIP` methodology incorporates FSM transformation to increase the reachable state space of the FSM, as shown in Fig. 1. In commercial IPs, the state space of the FSM is significantly smaller, and the ratio of the state space being used to the state space available, or in other terms, the *reachability*, is relatively low [15]. An adversarial entity can take advantage of this smaller state-space size and easily reverse engineer the FSM to get access to the control logic of the IP with minimal effort [8]. Hence, `ProtectIP` deliberately inserts an exponentially large number of nonfunctional states using extra flip-flops to increase the complexity of RE state space. The normal mode of operation can be retrieved by applying the unlocking key sequence.

The major contributions of this article are listed as follows.

1) It identifies the security and scalability issues in existing obfuscation techniques and introduces scalable state-space obfuscation methodology that is amenable to automation. The proposed methodology increases the reachable state space of the IP without requiring

Manuscript received 25 October 2022; revised 15 February 2023, 5 May 2023, and 6 July 2023; accepted 27 July 2023. *(Corresponding author: Md Moshiur Rahman.)*

The authors are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: rahman.m@ufl.edu; swarup@ece.ufl.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2023.3307027.

Digital Object Identifier 10.1109/TVLSI.2023.3307027

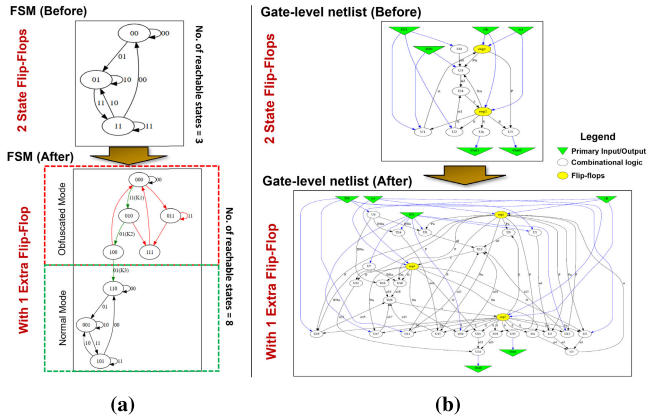


Fig. 1. Illustration of state-space obfuscation. (a) State transition graph of a 2-bit FSM before and after obfuscation. (b) Gate-level netlist representation of pre- and post-OFSM.

the enumeration of the state space. It provides robust protection against state-space enumeration-based RE as well as structural analysis attacks.

2) It implements the methodology into a CAD tool and integrates it into commercial EDA flow. Using the integrated flow, `ProtectIP` is evaluated with respect to design overhead and security against several known attacks that apply to sequential obfuscation for a suite of large-scale IP blocks.
3) It proposes three security metrics to quantify the level of improved RE complexity to attack state-space obfuscation under different threat models. It evaluates the security of the transformed designs using existing tools and demonstrates robustness against potent structural analysis attacks.

The rest of this article is organized as follows. We present the security vulnerabilities in existing FSM obfuscation techniques in Section II with an introduction to existing attacks. Section III provides the major challenges of state-space enumeration and lays out the scope of contribution. Section IV justifies classifying different flip-flops' structures in a gate-level IP. Section V illustrates state-space expansion methodology, whereas Section VI explains the structural transformation. Initialization and unlocking of an obfuscated design are presented in Section VII. We introduce our CAD tool, `ProtectIP`, in Section VIII. Section IX provides the security evaluation of the proposed methodology. We present the results and security benefits of obfuscating several large open-source IPs in Section X. We conclude in Section XI.

II. Background

Hardware obfuscation is a well-established concept in the field of hardware IP protection. In this section, we briefly present the basic ideas of existing attacks and defenses in the sequential logic obfuscation research area. We also give pointers to the limitations of existing defense techniques in preventing IP RE attacks.

A. Existing Attacks on Obfuscation

1) Functional Attacks: Boolean satisfiability (SAT) attack [12] is the most prominent attack against the majority of logic locking algorithms. A SAT solver iteratively prunes out the key search space by removing the incorrect key combinations for unlocking a combinational logic-locked design and often can generate the exact unlocking key combination. Key condition crunching (KC2) [16] is another variant of SAT attack that applies to sequential design where the design is first unrolled to generate the purely combinational logic representation of the sequential design, and then, SAT is applied.

2) Structural Attacks: DANA [17] and REFSM [8] are the two most applicable structural attacks in the context of FSM obfuscation. Both attacks share the same basic principle of clustering algorithm to identify the register groups in a given sequential design by looking at the register-to-register connectivity. However, REFSM has additional capabilities to reconstruct the functional behavior of the FSM from the identified register or group of flip-flops.

B. Existing Defenses and Shortcomings

1) HARPOON: It inserts an FSM in addition to the existing FSM that controls key gates in the combinational logic of the gate-level netlist of an IP [5]. An IP-specific input sequence needs to be applied to bypass the nonfunctional entrance FSM and reach the existing or original FSM. HARPOON has its shortcomings that potentially reduce the RE complexity as listed in the following.

1) HARPOON does not explicitly modify the state space of the existing FSM and hence may not provide adequate protection against structural analysis-based RE attacks [8], [18].
2) It does not include quantifiable RE complexity metrics.
3) It does not address the scalability issue in analyzing the state space of the existing FSM.

2) Boosted FSM: Boosted FSM (BFSM) [6] uses a random unit block (RUB) as part of the IP, which generates a random unreachable initial state and forces the flip-flops to power up from this state. Although multiple state progression paths are created from the random initial state, one (or multiple) control input sequence(s) leads to the functional initial state of the FSM, and the control input sequence becomes the activation key sequence. However, due to the absence of structural changes to the FSM and no logical co-dependency between the existing FSM and RUB, BFSM schemes are vulnerable to structural attacks [18]. In addition, the threat model of BFSM assumes zero access to the design flip-flops, which is a very conservative and obsolete assumption.

3) Encrypt Flip-Flop: A simple multiplexer-based output inversion technique [19] uses the fundamental structure of flip-flops where each flip-flop comes with two complimentary output ports (Q and $\overline{Q}$). The 2-to-1 multiplexing of a flip-flop's outputs and a key input controlling the branch of the multiplexer allows control over the operation of that flip-flop. The wrong value at the key input inverts the input to the flip-flop fan-out logic, making the design work in obfuscated mode. Only the correct key bit to the key input chooses the functional output of the flip-flop. However, multiplexing does not alter the state transition; it only gets inverted. Moreover,

the original FSM's structure remains unchanged, making the scheme vulnerable to many structural and functional attacks, including SAT attack [12].

4) Dynamic State Deflection: Dynamic state deflection [9] relies on inverting the input to the flip-flops by inserting additional gates. Similar to the encrypt flip-flop methodology, combinational logic gates are placed at the input of the state flip-flops, and external key input ports control those gates. A wrong key input takes the FSM to a black hole state, and there is no path to get out from the loop of black hole states. Therefore, the correct key needs to be applied in each transition to keep the FSM in the functional state space. As the methodology implies, static key input ports create an attack surface for SAT attack and probing. Furthermore, decorrelated black hole states are another point of interest for attackers to launch structural attacks and isolation of added states to reverse engineer the FSM [8].

5) Cellular Automata-Guided FSM Obfuscation: Cellular automata (CA)-guided FSM obfuscation [20] is another approach where state transitions are protected using a secret key. The state encoding is guided by CA along with key dependency. However, a recent oracle-based attack ORACALL [21] shows that CA-guided FSM obfuscation is vulnerable to SAT attack.

6) Hidden State Transition: Synthesis of hidden state transitions (HSTs) [22] is a recent work on FSM obfuscation where a set of key-dependent state transitions are introduced in a sequential design that helps prevent SAT attack. An unrolling step helps guide the impact of modifying the state transition logic and estimating the size of the key. Since HST is designed with the objective of preventing SAT attack, it can prevent classes of functional attacks leaving room for structural attacks. In addition, unrolling comes with the disadvantage of poor scalability with the increase in the size of the designs.

7) JANUS: A set of FSM transformation-based logic obfuscation is proposed under the common name: JANUS [23], [24]. In JANUS, the state transition diagram of an FSM is partitioned into smaller groups, and based on the origin and destination of state transitions, the configurable flip-flops either pass or flip the next state configuration. Although JANUS illustrates high SAT resilience, it fails to demonstrate and quantify the structural attack resistance. In addition, JANUS has a dependency on reachable state analysis, which is exponentially complex for larger FSMs and there is the possibility of nonconvergence due to the deficiencies in required functional behaviors for a few types of FSMs. This is a drawback of applying it for any and all types of FSMs. In addition, specific types of flip-flop usage create structural signatures that aid easy RE using DANA and other similar attacks.

III. Desired Features

To provide comprehensive resilience against RE attacks on an IP, it is imperative to perform both structural and functional transformation; thus, isolation of obfuscation logic and RE can be prevented. However, the existing techniques deviate from following the fundamentals of secure obfuscation and lack the structural deformity between added logic and existing logic. Therefore, there is a need to develop a scalable methodology for FSM transformation-based state-space obfuscation with a strong security guarantee, which has the following distinguishable features when compared to other techniques.

1) It increases the reachable state space exponentially in order to prevent state-space enumeration, analysis, and reconstruction of FSM functionality using DANA/REFSM.
2) Unlike REFSM/HST/JANUS, the obfuscation methodology can be applied without any requirement of extracting the functional behavior or state transition diagram.
3) Unlike HST/JANUS, transformation is not FSM behavior agnostic and can be applied with no requirement of validating the obfuscation conditions. Hence, it can be applied to all types and sizes of FSMs.
4) The FSM transformation withstands functional query-based SAT attack even with access to an unaltered scan chain, which JANUS and HST approaches fail to provide.
5) Both functional and structural attack resilience can be achieved. Therefore, it can provide comprehensive protection against all attack modalities, unlike transformation algorithms that target specific attacks (e.g., SAT attack resilience in JANUS).

In the following, we disclose the fundamental area of our focus while bringing in the novelties listed above.

A. Increasing the Reachability of FSM

An FSM, the most abundant control component in a hardware IP, dictates the IP's operation at any given time and is driven by control inputs (either internal or external). Logic synthesis process maps an FSM to a group of sequential cells, e.g., D-type flip-flops, and some combinational cells, e.g., AND/OR/XOR gates. The combinational cells constitute the state transition logic, and the flip-flops hold the state values. The number of flip-flops (n) in the FSM depends on the type of encoding scheme, such as binary, gray, and one-hot, and has the room to create 2^n number of states, maximum. However, the required number of states depends on IP's operation and is unrelated to the encoding scheme. Legacy and modern IP designers often use an encoding scheme that is better for the IP performance or other budgeting constraints, which may lead to some (usually a significant) portion of the available state space being unused. Untrusted entities can take advantage of this unused state space to reverse engineer and get insights into the control FSM. To hinder that effort, we enrich the existing state space of the design by modifying the FSM state transition function, which increases the reachable state space of the FSM exponentially, and RE attack becomes computationally infeasible.

B. Scalability of State-Space Enumeration

To transform the state machine and expand the state space, it is required to know the transfer function or state transition diagram of the existing FSM. Such transition function identification starts with identifying the FSM from the IP. Although identification of small FSMs from an IP is not so difficult [25], with the increasing complexity of commercial

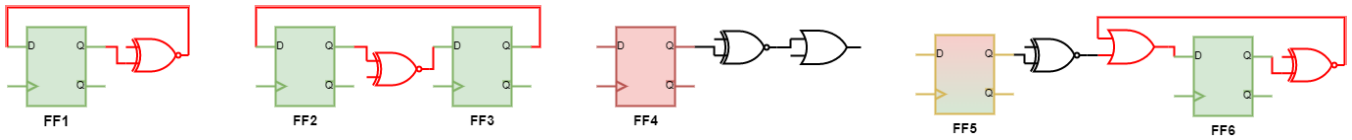


Fig. 2. Structures of different classes of flip-flops. FF1, FF2, FF3, and FF6 are state flip-flops as they have feedback paths (red logic cone) from their outputs to inputs. FF4 is a data-path flip-flop as it only feeds forward (black logic cone). FF5 is a pseudo-state flip-flop as it is feeding forward (as a data-path flip-flop) to FF6 (a state flip-flop).

IPs, it is exponentially harder and computationally expensive to extract and enumerate the entire state space before injecting additional states and transforming the FSM. Therefore, the implementation of state-space obfuscation is not a direct mapping of the theories and algorithms of FSM obfuscation, which comprises increasing the state-space reachability by making unreachable states (if any) reachable or increasing FSM width and reencoding. To avoid the computational complexity of the state space, scalable FSM obfuscation needs to be applied that serves the purpose of obfuscation without requiring the enumeration. Obfuscation can be performed to address the scalability issue by adding extra state elements (flip-flops) in the gate-level netlist. However, the insertion of new flip-flops needs to be corelated with the existing states to create functional and structural dependencies between the functional and nonfunctional state spaces to mitigate the security vulnerabilities in existing obfuscation techniques. Therefore, achieving both functional and structural integrity of obfuscation is extremely challenging.

IV. Addressing Scalability

State-space obfuscation considers modifying the underlying FSM of a given hardware IP's gate-level netlist. Although FSMs get mapped to flip-flops as state elements, several other behavioral logic blocks or operations may also require flip-flops to characterize the physical implementation. Hence, it is crucial to classify the flip-flops based on their contribution to different logic blocks as a preprocessing step of our proposed obfuscation. According to our algorithm, the flip-flops in a gate-level netlist can be divided into three classes: 1) state; 2) data path; and 3) pseudo-state flip-flops. Fig. 2 shows the topological differences between each class.

A. State Flip-Flops (FSM Flip-Flops)

The behavioral operation of an FSM is to generate the next state based on the current state and input stimuli. As a result, gate-level netlist of an FSM consists of feedback path(s) from the output of the flip-flops (current state) to their inputs (next state). The feedback path is a block of combinational logic gates, which may contain a set of control inputs (input stimuli), as shown in Fig. 2. Such topological configuration of a flip-flop signifies the notion of state. Hence, we define it as state flip-flop. We can avoid state-space enumeration by considering each state flip-flop as a separate FSM component and obfuscating their input function to improve the scalability.[1]

Special Case: A flip-flop can have a feedback path, irrespective of it being part of explicit FSM logic. For example,

[1]In the rest of this article, state flip-flop (^{S}FF) is used as an alias of FSM flip-flop.

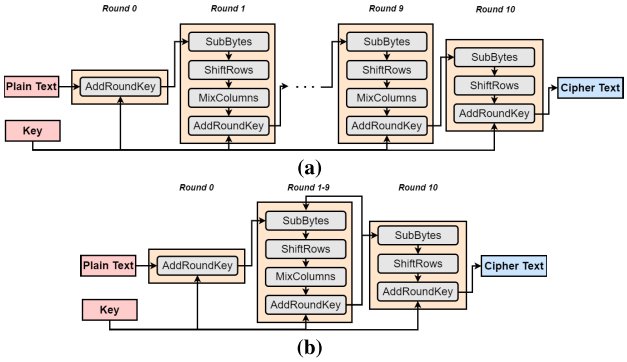


Fig. 3. Two different implementations of AES-128. (a) Fully unrolled by duplicating the data path for each round results in fewer state flip-flops. (b) Optimized by sharing the logic among identical rounds leads to more number of state flip-flops.

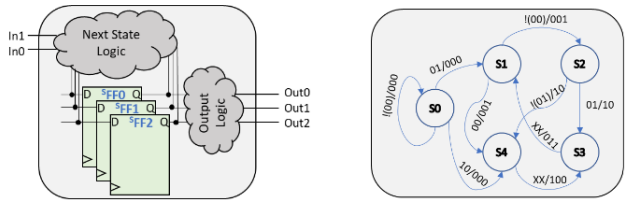
in the AES module [26], output from one round is being used as an input to the next round where each round consists of pure data-path operation, as shown in Fig. 3(a). However, the implementation can be optimized by reusing the data path of one round in the subsequent identical rounds, as shown in Fig. 3(b). Logic sharing results in all the flip-flops in the data path of one round to form feedback path(s). Hence, the flip-flops in the data path are classified as ^{S}FF flip-flops according to our ^{S}FF identification algorithm, although they are not part of FSM. In this way, the proposed state-space obfuscation algorithm is not conservative in defining and extracting the explicit FSM from a given gate-level netlist.

B. Data-Path Flip-Flops

Almost all commercial hardware IPs contain pipelined logic to improve the performance or leverage the architectural resources in an SoC. Pipelined logic is almost always clock-enabled, which results in flip-flops. Unlike state flip-flops, these pipeline flip-flops do not contain any feedback path. These feedforward-only flip-flops are referred to as data-path flip-flops (^{D}FF). If traversing the fan-out cone of a flip-flop in every possible path never renders a feedback path, the flip-flop is marked as ^{D}FF. In short, any flip-flop in a netlist that is not a ^{S}FF can be labeled as ^{D}FF.

C. Pseudo-State Flip-Flops

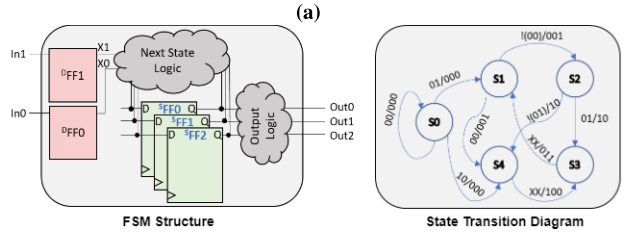
A subset of all ^{D}FFs, found in the fan-in cones of ^{S}FFs, is called pseudo-state flip-flops (^{P}FF). ^{P}FFs indirectly impact the state transition of the FSM. To elaborate on this, let us assume two instances of a sample FSM shown in Fig. 5 where, in one instance, the FSM has no ^{D}FF in the fan-in cone. Let us consider the FSM has three ^{S}FFs: $^{S}\text{FF}_0$, $^{S}\text{FF}_1$, and $^{S}\text{FF}_2$; two-bit control input In = {In1, In0}; and three-bit control output



Random Input (*In*)	State Transition Input (*In*)	Current State (S_c)	Next State (S_n)	Output (*Out*)
10	10	000 (*S0*)	100 (*S4*)	000
01	01	100 (*S4*)	011 (*S3*)	100
01	01	011 (*S3*)	001 (*S1*)	011

State traversal sequence : S0 > S4 > S3 > S1 > ... ...
Output sequence : 000 > 100 > 011 > ... ...

(a)



Random Input (*In*)	State Transition Input (*X*)	Current State (S_c)	Next State (S_n)	Output (*Out*)
10	00	000 (*S0*)	000 (*S0*)	000
01	10	000 (*S0*)	011 (*S4*)	000
01	01	100 (*S4*)	011 (*S3*)	100

State traversal sequence : S0 > S0 > S4 > S3 > S1 > ... ...
Output sequence : 000 > 000 > 100 > ... ...

(b)

Fig. 4. Impact of data-path flip-flops on state transition of an FSM. (a) State traversal and output sequence of the FSM with no data-path flip-flops in fan-in cone for a random input sequence. (b) When inputs are pipe-lined via addition of data-path flip-flops, the same FSM state traversal sequence changes.

Out = {Out2, Out1, Out0}. The structure and state transition diagram is shown in Fig. 4(a), which depicts that the control outputs and next states are decided for different inputs and current states combinations. Since the next state logic of the FSM contains the current state and inputs directly (without having ^{D}FF), the input stimuli are readily available as part of the next state logic in the same clock period. This results in a state traversal sequence of $S0 \rightarrow S4 \rightarrow S3 \rightarrow S1 \rightarrow \cdots$ and output sequence of $000 \rightarrow 100 \rightarrow 011 \rightarrow \cdots$.

In the other instance, each ^{S}FFś input is pipelined, as shown in Fig. 4(b). As a result, the input stimuli lag one clock cycle before contributing to the next state logic. For the same input patterns as the previous instance, the operation of the pipelined FSM is different as the state traversal order changes to $S0 \rightarrow S0 \rightarrow S4 \rightarrow S3 \rightarrow \cdots$, which impacts the generation of control outputs ($000 \rightarrow 000 \rightarrow 100 \rightarrow \cdots$).

Based on the discussion of the above two instances, it is evident that the data-path flip-flops in the fan-in cones of the state flip-flops impact the next state logic. Hence, they are not true state flip-flops. Instead, we mark them as pseudo-state flip-flops.

D. Target Set of Flip-Flops for Obfuscation

In general, a real IP contains all three types of flip-flops. Once the classification of flip-flops is completed based on the above structural properties, a subset of the flip-flops needs to be targeted for obfuscation. Since state flip-flops are the ones that contain the state values and pseudo-state flip-flops have an indirect impact on these, both state flip-flops and pseudo-state flip-flops must be chosen for state-space obfuscation.

V. Extension of Reachable State Space

State-space obfuscation methodologies include two mutually inclusive approaches to increase the reachable state space: 1) increasing the reachability of the existing ^{S}FFs by superimposing a highly reachable FSM and 2) adding new state flip-flops to the netlist to control the mode of obfuscation and existing ^{S}FFs.

A. Superimposition of Highly Reachable FSM

As discussed earlier, the reachability of the FSM in an IP may not be high enough to prevent an attacker from RE FSM. To address this vulnerability, the state-space obfuscation algorithm incorporates the superimposition of a highly reachable FSM on the existing state flip-flops in the gate-level netlist. Due to the superimposition, the same group of flip-flops can operate as two different FSMs: 1) the less reachable original FSM and 2) the highly reachable (ideally 100%) superimposed FSM. To illustrate the implementation of superimposition, let us consider a state machine (FSM1) shown in Fig. 5(a), which has two state flip-flops. This handcrafted FSM has two reachable states: 00 and 11. We take another 2-bit FSM for superimposition. In this example, we superimpose a 2-bit counter (FSM2) of full reachability (four states: 00, 01, 00, and 10), as shown in Fig. 5(b). The superimposition algorithm ties the input and output logic of FSM2 flip-flops to the FSM1 flip-flops through multiplexing at the input, as shown in Fig. 5(c). Both the original FSM and the highly reachable FSM branches are interchangeable, meaning that they can be connected to the inputs of the multiplexer in any order. Due to the superimposition on FSM1, the same group of FSM1 flip-flops, which had 50% reachability (two out of four), now becomes 100% reachable (four out of four) if the FSM2 logic branch of the multiplexer is selected. Although for such a small circuit (few state flip-flops), the state-space increase is not significant, with the increment in the number of state flip-flops in a design, the state-space explosion is exponential.

A variety of highly reachable FSMs can be used for superimposition. However, the choice of FSM needs to be guided by overhead, maximum reachability, and nondeterministic state transition function. In our implementation, we use maximum period nonlinear feedback shift registers (NLFSRs) [27] as superimposing FSM. NLFSRs exhibit 100% reachability and incur significantly lower overhead, even for a large-sized NLFSR. Moreover, the next state of NLFSR is a nonlinear function of the current state, making it nondeterministic. Hence, it is a secure choice to superimpose using NLFSR.

B. Addition of Flip-Flops Through Obfuscation FSM

As we discussed earlier, the number of state flip-flops dictates the number of states in the design where each flip-flop can contribute to growing the state space by $2X$. State-space

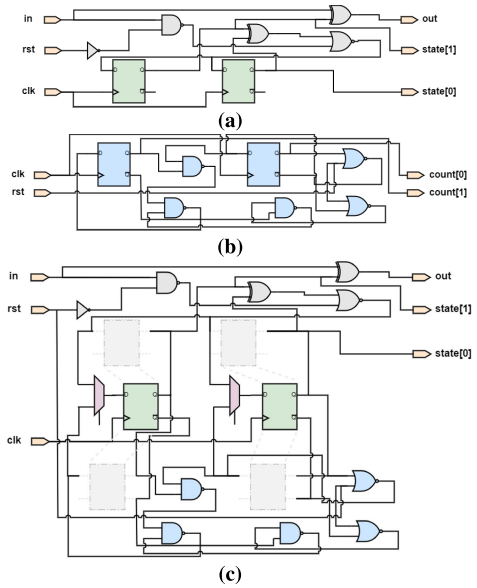


Fig. 5. Superimposition of highly reachable FSM on top of the existing FSM. (a) FSM (FSM1) with two state flip-flops. (b) 2-bit counter (FSM2). (c) Multiplexing input logic of FSM1 flip-flops to include the logic of both FSM1 and FSM2 where multiplexer control inputs are open to be driven by enable signals.

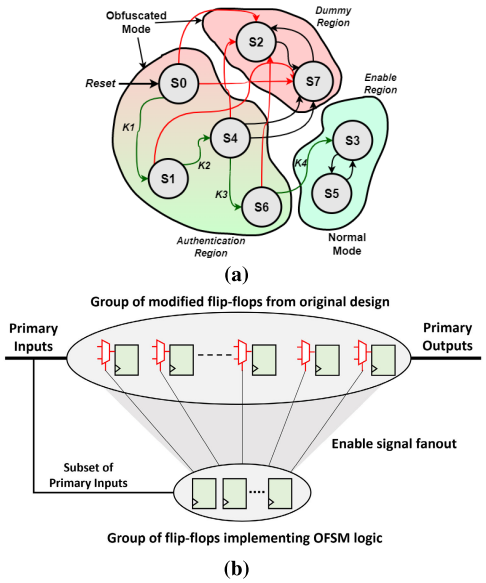


Fig. 6. OFSM insertion in a design. (a) State transition among different regions of a three flip-flop OFSM (maximum eight states). (b) OFSM is merging with existing design via enabling signals.

obfuscation utilizes this fundamental advantage of adding flip-flops to the existing pool of state flip-flops to increase the state space exponentially. Instead of enumerating the state space and increasing the FSM size by adding random flip-flops to random FSM logic, the state-space obfuscation algorithm inserts configurable FSM, which includes several flip-flops. This FSM is called obfuscation FSM (OFSM) as these control whether the IP will operate in obfuscated mode or normal mode. Fig. 6 includes the state transition diagram of an example OFSM merged with the original design.

1) Configuration: The size (number of flip-flops) of each OFSM in a given design can be any positive integer value based on the number of state flip-flops in the netlist, the expected level of RE complexity, and overhead constraints. However, each OFSM must include enough flip-flops to be able to define three separate regions of operation. The three regions are authentication region, enable region, and dummy region, each composed of multiple states. If the current state of the OFSM belongs to authentication region or dummy region, the design operates in an obfuscated mode, whereas state values belonging to enable region ensure the normal mode.

2) Key-Based Traversal: Unlike static keys in the added key inputs in combinational logic locking, OFSM state traversal is controlled by a sequence of multibit input patterns or key sequence. The key sequence is applied via existing primary inputs of the design, starting from the power-up or reset state of the OFSM. In this way, OFSM acts as a key-based authentication FSM. As shown in Fig. 6(a), the power-up state is $S0$ and a sequence of keys, $K1 \rightarrow K2 \rightarrow K3 \rightarrow K4$, each of multiple bits, can be applied to traverse through the authentication region states ($S0 \rightarrow S1 \rightarrow S4 \rightarrow S6 \rightarrow$) and reach the first enable region state ($S3$). Until the OFSM is reset, it remains in enable region states ($S3$ and $S5$). However, before reaching the enable region, if a wrong key is applied, the OFSM enters into the dummy region and keeps looping through the dummy region states until the OFSM is reset.

3) Merging With Original FSM: The OFSM generates a set of enable signals, $E = \{0, 1\}^N$, to configure the multiplexed input branches of superimposed state flip-flops (described in Section V-A). Correct enable signal values are generated only in enable region which select the original FSM logic branch of the state flip-flop input multiplexer, and the normal mode of operation is guaranteed. Incorrect enable signals are generated in both authentication region and dummy region which choose a random combination of the superimposed and original FSM logic branches. To avoid static enable signals in the obfuscated mode, the enable signals keep switching to random values in each state transition, which may enable random logic. However, such random intentional unlocking of part of the design defeats the switching activity-based enable signal identification. Only the enable region ensures that all enable signals generate correct values to enable the entire design and switch to the normal mode. Existing functional inputs to the IP are chosen as key inputs to apply the key and make state transitions. OFSM shares the same block-level clock and reset ports as the original design, thereby avoiding adding extra inputs to facilitate the obfuscation. The block-level diagram shown in Fig. 6(b) illustrates the merging of an OFSM with an existing but superimposed state flip-flop array.

VI. Transformation of Combinational Logic

One of the main objectives of hardware obfuscation is to hide the design structure from the attackers to prevent unauthorized use of the IP. Structural attacks can locate critical parts of the data path of a design and perform local analysis to extract secret information [11]. Hence, a robust transformation of combinational logic needs to be achieved in parallel with the FSM transformation to mitigate structural property-driven vulnerabilities. In a control-path-heavy design, only FSM transformation may not introduce enough output corruptibility.

In such cases, combinational logic transformation helps to prevent black-box usage of the IP by corrupting the data path. However, the proposed obfuscation technique explicitly transforms the sequential logic boundary of the IP, and it does not guarantee the increase in the RE complexity of a data-path-only design. Therefore, applying the obfuscation algorithm in a data-path-only design is not encouraged.

A. Modification Cell Insertion

State-space obfuscation allows an IP to be transformed structurally by inserting modification cells in the data path or combinational logic. Fundamental logic gates or complex logic blocks consisting of multiple logic gates can be used as modification cells. This approach is similar to combinational logic locking. Therefore, modification cell insertion is the step that can leverage the combinational logic locking algorithms, e.g., Anti-SAT block [28], SAT-hard functions [4], and SFLL [3], and combine the complementary techniques for comprehensive coverage over all known attacks whether targeted to combinational or sequential logic locking. However, the major difference here is that, in state-space obfuscation, the key input to the modification cell is driven by enable signals generated from added FSM. In contrast, key inputs are driven by observable external key inputs in combinational locking. Our proposed modification cell insertion step has more security benefits when compared to the combinational logic locking algorithms. Applying SFLL-like combinational logic locking to obfuscate sequential logic has its own security vulnerabilities. Static key application during/for normal mode of operation allows probing attack to extract the unlocking key values [29]. In addition, static keys are more susceptible to SAT analysis [16].

In contrast to the above disadvantages, the proposed obfuscation algorithm implements sequential dependency of key application for unlocking or transitioning to the normal mode of operation, which has an exponentially larger impact on SAT attack resilience by forcing SAT-solver toward a larger problem (larger combinational logic cones after unrolling). The added FSM, which drives the modification cells, is called data-path enable FSM (DFSM). The structure and state transition function of a DFSM are identical to an OFSM, but the only distinguishing factor is that the DFSM drives the data path. A block-level diagram of DFSM merging with existing combinational logic is shown in Fig. 7. Due to the addition of modification cells, the existing data path becomes control-path-dependent, achieving significant structural transformation during the synthesis and optimization process. Furthermore, a new set of flip-flops are added, further enhancing the state-space explosion.

B. Node Selection for Modification Cell Insertion

To achieve maximum structural transformation through obfuscation, we need to choose a set of optimal nodes for inserting modification cells. The level of structural transformation can be quantified by doing simple logic equivalence checking (LEC) of the preobfuscated and postobfuscated designs. LEC reports the percentage of node failures between

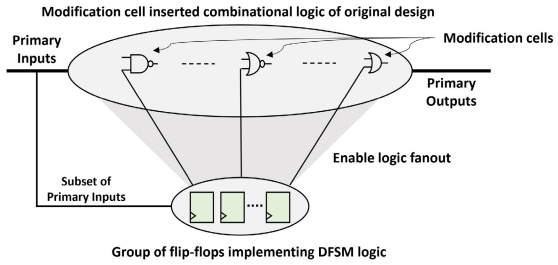


Fig. 7. Merging DFSM with modification cell inserted combinational logic of the design.

two instances where flip-flop pairs are used as cut-points to define the region of equivalence check [30]. To increase the number of failing equivalent nodes for any obfuscated design in formal verification, we need to choose nodes with larger fan-in and fan-out logic cones. The nodes having high fan-in logic indicate a higher logic depth, whereas high fan-out logic indicates that the node modification is most likely to impact a large portion of the design. Let us assume that the number of nodes (gates) in the fan-in and fan-out cones of a node M in the design is FI_M and FO_M, respectively. We use the following equation to assign a rank to the node and choose the higher ranked nodes to insert modification cells:

$$\mathrm{rank}_M = \frac{\mathrm{FI}_M + \mathrm{FO}_M}{2}. \tag{1}$$

VII. Unlocking and Initialization

As discussed in Sections V and VI, we add a large number of nonfunctional states that correspond to the obfuscated mode of operation. To bring the IP to normal mode, an IP-specific key sequence needs to be applied through appropriate primary inputs of the obfuscated design. During the key application period, in the process of unlocking the IP, the IP operates in the obfuscated mode for multiple (= length of the key sequence) clock cycles and the original state flip-flops have a highly reachable FSM being superimposed. Therefore, existing state flip-flops reach random nonfunctional states. Hence, it is essential to initialize those to the original IP's initial state upon provisioning the key sequence. This initialization preserves the functional equivalence between the original IP and the unlocked obfuscated IP.

A. Identification of Initial State

To initialize the original state flip-flops to functional initial/reset/power-up state, it is required to know the state or output of each state and pseudo-state flip-flop in the preobfuscated design. Since our methodology does not rely on extracting the state space through state enumeration, the nonobfuscated design is simulated under reset for multiple clock cycles.

B. Bringing the FSM to Correct Initial State

Since OFSM (and DFSM) shares the same reset port that resets the original design, we cannot use the reset port to initialize the state and pseudo-state flip-flops after key sequence provisioning. Moreover, resetting after the key application also

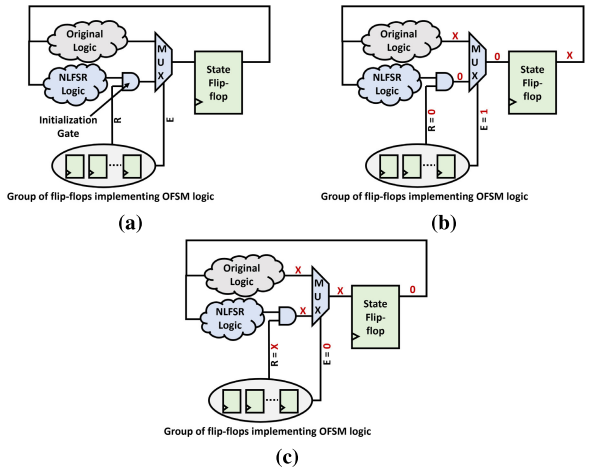


Fig. 8. Initialization of a state flip-flop. (a) Configuration of the state flip-flop requires to be initialized to 0 when initialization signal (R) and correct enable signal (E) values are 0 and 0, respectively. (b) Configuration of the flip-flop at preenable state ($(t-1)$th clock cycle where t is the length of the key sequence). (c) Initialization of the flip-flop at first enable state [(t)th clock cycle] where 0 appears at the output of the flip-flop, and the original logic branch is chosen by asserting 0 (enable) at the select input of the MUX.

resets the OFSM and brings the design back to the obfuscated mode. Therefore, we need to use the data path of the flip-flops to initialize them. The OFSM forces the superimposed state and pseudo-state flip-flops to pass a constant (reset state value captured through simulation) to the output of each flip-flop through the data input of the same flip-flop. To allow such constant propagation, in addition to the enable signals generated from the OFSM, a set of new (internal) initialization signals, R is generated against each state of the OFSM ($R = \{0, 1\}^N$). Although the initialization signals can be derived from the enable signals, having separate signals helps in the decorrelation of these two types and allows more randomness in switching activity. The initialization signals contain random values in authentication region and dummy region and the reset values required to initialize each flip-flop right before entering the enable region.

The initialization signals get connected to the obfuscation logic branch (NLFSR) of the multiplexers at the input of the state flip-flops through initialization gates. The purpose of the initialization gate is to nullify the impact of random state traversal and force a specific value through the multiplexer during initialization. The type of initialization gate is decided by the reset value required for one particular state flip-flop and the value of the initialization signal generated from OFSM during the initialization stage. For example, let us consider the flip-flop shown in Fig. 8(a) that needs to be initialized to 0. Suppose that the initialization signal value is 0. In that case, the type of initialization gate needs to be an AND gate to ensure that the logic input to the obfuscation branch of the multiplexer becomes 0 as the other input to the AND gate is driven by random (NLFSR) logic. Different gates can be used for various combinations of the flip-flop's initialization signal value and required initial state value.

The initialization gate can force the particular value to the input of the flip-flop (output of the multiplexer), provided that the multiplexer select signal chooses the obfuscation logic branch (NLFSR branch). To sync every possible combination of the state of the OFSM, the value of the initialization signal, the select signal value of multiplexer, and the reset value of the flip-flop, the OFSM needs to have one initialization state or preenable state before entering the enable region. In the preenable state, the enable signals generated from OFSM choose the NLFSR logic branch, which has a constant value (reset value of that particular flip-flop). Such configuration permits a constant to be pushed to the input of the state flip-flop, as shown in Fig. 8(b). Once the OFSM switches from the preenable state to the enable state (one clock cycle apart), the value at the input of the flip-flop passes through the state flip-flop (due to clock port excitation) and appears at the output of the same flip-flop. To this end, the output of the flip-flop holds the initial state value, and the input of the flip-flop has the original design logic branch by configuring the multiplexer with the correct enable signal at the select input, as shown in Fig. 8(c).

C. Supporting Multiple OFSMs

Commercial IPs are typically large IPs with a high volume of flip-flops, which may result in hundreds to thousands of state flip-flops. To prevent high logic fan-out from one OFSM and avoid the vulnerability of structural analysis, multiple OFSMs are inserted to drive the array of state flip-flops. The state flip-flop array is divided into multiple chunks, and each chunk can be driven by one OFSM. The number of flip-flops in each chunk depends on the number of flip-flops in the OFSM and maximum fan-out constraints. All OFSMs can share the same set of inputs to read in the key sequence. Unless key application inputs are mutually exclusive, multiple OFSMs cannot be unlocked in parallel. On the other hand, serial unlocking of each OFSM initializes and unlocks part of the design (a subset of the state flip-flop array being driven by that OFSM) at different clock cycles and results in erroneous initialization of the IP. Therefore, correct key sequence application does not guarantee unlocking in multi-OFSM scenarios since proper initialization is part of the unlocking process. Therefore, all OFSMs must initialize all the state flip-flops simultaneously and make the simultaneous transition to enable region, as shown in Fig. 9.

To adhere to the simultaneous switching to enable region, each of the three OFSMs must be brought to their preenable state by applying the correct key sequence and retaining the state until all the other OFSMs reach their corresponding preenable states. This ensures that the entire design remains in the preenable state for at least one clock cycle and initialization takes place for the whole IP. Afterward, all OFSMs can be triggered to switch to the enable states simultaneously by applying one common input pattern (as part of the key sequence). The number of states in different regions of the OFSMs shown in Fig. 9 can be increased to enhance the level of security of the obfuscated design. Similarly, multiple DFSMs may be needed based on the number of modification cells inserted. However, the multi-DFSM scenario does not require preenable state configuration since there is no notion of initialization in data-path (combinational) logic.

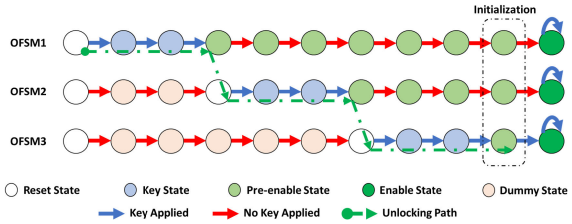


Fig. 9. State traversal of multiple OFSMs to ensure proper initialization. All OFSMs need to stay in their corresponding preenable states for at least one cycle before simultaneous switching to enable states.

VIII. CAD Flow: ProtectIP Tool

State-space obfuscation incorporates both control- and data-path obfuscation and is a complex yet comprehensive solution compared to the existing logic locking techniques. As discussed in Sections IV–VII, it is required to analyze different design features and algorithmically modify the behavior. Hence, we developed a fully automated CAD tool, ProtectIP, which can be used by a user[2] to obfuscate their IP. ProtectIP can take a Verilog gate-level netlist with flip-flops and generate the obfuscated version without any user intervention following the flow shown in Fig. 10. The algorithmic pseudocode is presented in Algorithm 1. The worst case algorithmic complexity of ProtectIP is polynomial. We have integrated our tool within the commercial toolchain of EDA vendor Synopsys, although ProtectIP can be easily integrated with any EDA flow. To the best of our knowledge, this is the first time an EDA-embedded CAD flow is presented to perform obfuscation. However, our flow is technology independent as it provides the IP owner to choose their standard cell library (as shown in Fig. 10. We have integrated EDA tools at the beginning to automate the consumption of the design information into our internal data structure since the basis of the obfuscation algorithm lies within the consumed data. If the required information is being fed by the IP owner manually, these initial steps are not required. Therefore, we leave minimal attack surface for an attacker to launch structural attacks. Our DANA analysis (described later) also supports the fact that the proposed flow does not have structural signatures to look for. Each step of the flow is briefly described in the following, along with the required inputs and outputs.

A. Generating Graph and Characterizing the Flip-Flops

ProtectIP uses graph-based algorithms for internal operation. Although ProtectIP supports RTL IP as input, the tool's front end generates the graph from the gate-level netlist.[3] Therefore, one round of bare-minimum synthesis is performed without any constraints. Afterward, using the features described in Section IV, ProtectIP identifies the classes of flip-flops present in the graph. In rare events, the design may not have any state flip-flop, and ProtectIP may end up with no target flip-flop. In such cases, ProtectIP randomly chooses a subset of the data-path flip-flops to continue the obfuscation process. To summarize, ProtectIP can handle designs with any number of state, pseudo-state, and data-path flip-flops.

[2]An example user is the IP owner who wants to protect their IP.

[3]In the rest of the section, the gate-level netlist is represented by the graph.

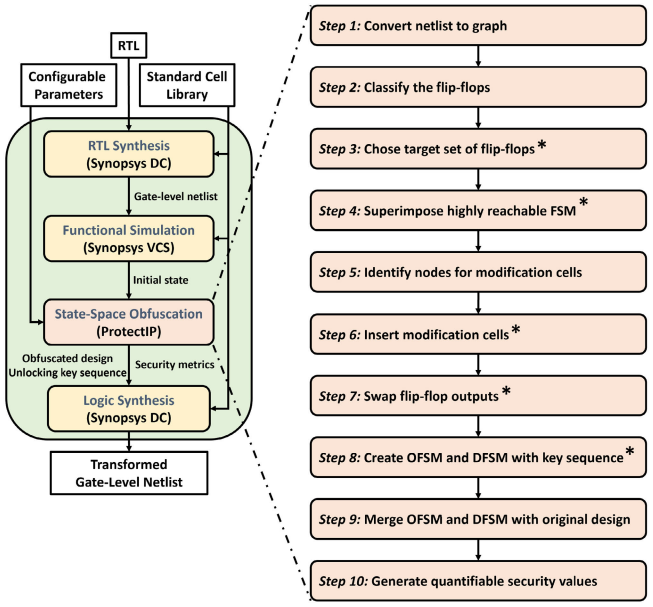


Fig. 10. Major steps of the proposed state-space obfuscation algorithm (ProtectIP) as a CAD tool, integrated into commercial (Synopsys) EDA flow. The step with an asterisk (*) implies that the step is user-configurable via specific parameters.

B. Superimposing FSM

ProtectIP uses the list of target flip-flops to superimpose highly reachable FSM. Gate-level design of a presynthesized NLFSR is converted to a graph, and only the combinational logic of this graph is superimposed through the multiplexer, as discussed in Section V-A. NLFSR and original FSM logic can be connected to random input branches (select0 or select1) of each multiplexer to avoid structural signatures.

C. Inserting Modification Cells

Strategic nodes of the graph are chosen based on their ranks to insert modification cells. Target nodes are further analyzed to identify the nonsuitable nodes directly connected to the primary input, primary output, or target flip-flop's input. Modification cell insertion in primary input–output leads to identifying the obfuscation logic (strongly connected). In contrast, the flip-flop input functions are already obfuscated using a superimposition scheme, and inserting modification cells is redundant. Hence, ProtectIP selectively figures out a set of suitable nodes for modification cell insertion and inserts m number of modification cells where $m = x\%$ of total combinational logic cells in the design ($0 \leq x \leq 100$), and x can be specified by the user based on the overhead constraints.

D. Creating OFSMs and DFSMs

The most critical asset of obfuscated design is the added OFSM (and DFSM, if any). ProtectIP uses a scalable, secure, and efficient algorithm to design very large OFSMs with configurable size and reachability.

Algorithm 1 Pseudocode of State-Space Obfuscation

Input : Gate-level netlist(original design), Gate-level netlist(NLFSR), Constraints
Output : Obfuscated gate-level netlist

function MAIN($Netlist_{org}, Netlist_{nlfsr}, Constraints$)
  $G_{org} = $ CREATEGRAPHFROMVERILOG($Netlist_{org}$)
  $ListStateFF = $ IDSTATEFLOPS(G_{org})
  $ListPstateFF = $ IDPSTATEFLOPS($G_{org}, ListStateFF$)
  $ListFF = ListStateFF + ListPstateFF$
  GENERATEOFSMDFSM($Constraints$)
  $G_{nlfsr} = $ CREATEGRAPHFROMVERILOG($Netlist_{nlfsr}$)
  $G_{hira} = $ MERGENLFSR($G_{org}, G_{nlfsr}, ListFF$)
  $G_{obf1} = $ MERGEOFSM(G_{hira})
  $G_{obf2} = $ INSERTMODCELL($G_{obf1}, Constraints$)
  $G_{obf} = $ MERGEDFSM(G_{obf2})
  $Netlist_{merged} = $ CONVERTGRAPH2VERILOG(G_{obf})
  $Netlist_{obf} = $ DOSYNTHESIS($Netlist_{merged}$)
  return $Netlist_{obf}$
end function

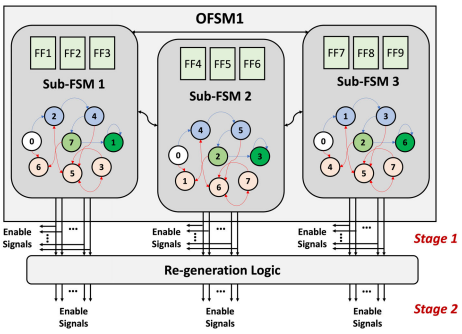


Fig. 11. Increasing driving strength of added FSMs by reusing enable logic to generate more enable signals. (a) Multiple enable signal vectors generated from two OFSMs. (b) Same number of enable signals can be derived from one OFSM by using additional combinational logic.

1) Size and Reachability: ProtectIP can design an FSM with any number of flip-flops. Therefore, the user can configure the minimum and maximum size of OFSM (number of flip-flops). Moreover, for efficient run time of the design and overhead limitation, the reachability of each added FSM also needs to be configured to avoid enumerating all possible states of a very large FSM (e.g., 32 flip-flops can have 2^{32} states). Therefore, the user can specify the number of states enumerated for each added OFSM.

2) Configuration: ProtectIP breaks down the specified size of each OFSM to multiple subgroups to efficiently generate the enable signals (and initialization signals). Let us use the FSM in Fig. 11 with nine flip-flops as an example. First, a random grouping of flip-flops is done that divides nine flip-flops into several subgroups, e.g., three subgroups of three, three, and three flip-flops. Each subgroup is considered a separate sub-FSM. In this way, ProtectIP avoids enumerating 2^9 states that would result in significant area overhead. The state space of these sub-FSMs is divided into three regions (authentication region, dummy region, and enable region) to generate enable signals. Combining a specific region from each sub-FSM, one holistic region is formed for the full OFSM. ProtectIP considers the maximum allowed fan-out (constraint imposed by the user) to decide on the number of signals generated from each sub-FSM and to drive a maximum number of flip-flops or modification cells. In this way, an optimized number of OFSMs is generated based on the number of state flip-flops. In a similar way, the number of modification cells and fan-out guides the number of DFSMs.

3) Fan-Out and Overhead Reduction: Due to many flip-flops in commercial IPs, many OFSMs/DFSMs may need to be added to avoid high fan-out cones of enable signals, which otherwise can potentially create an attack surface. On the other hand, if a large number of OFSMs/DFSMs are added, obfuscated design may incur significant design overhead. To deal with these issues, ProtectIP increases the driving strength of the enable signals to drive more flip-flops maintaining maximum fan-out constraint. The concept is to reuse the enable signals to generate a new set. A logic block comprising random Boolean functions can be placed in the fan-out cones of these enable signals to generate a second stage of enable signals, which can drive additional flip-flops or modification cells. Fig. 11 shows multistage enable signal generation of one single OFSM that effectively reduces the number of flip-flops, with a minimal increase in the combinational logic, which is way lower than adding a full OFSM. By making a tradeoff between the level of security and incurred overhead, the number of stages of enable signals from one FSM can be decided.

4) Key Inputs and Sequence: Once the added FSMs are configured, the length of the key sequence (or the number of states in authentication region), Q is known to ProtectIP. ProtectIP decides the number of average key bits per transition, P', using the following equation where keySize is an obfuscation constraint imposed by the user:

$$P' = \text{keySize}/Q. \tag{2}$$

According to (2), if the request is 512-bit key and a sequence length of 200, an average of 3-bit (P') per key value is required. ProtectIP uses a range, $(P' - p)$ to $(P' + p)$, to randomly assign the width of each key (P) per FSM transition. Random P primary inputs are chosen as key inputs, and a random P-bit binary number is used as the key. Due to the randomness, the key's final size may not match the requested key size. However, p can be specified to 0 to suppress randomness and get an exact match with the requested key size.

E. Outputs

The OFSMs and DFSMs are merged with the modified original design and synthesized to generate the gate-level netlist. In addition to the netlist, ProtectIP generates the key sequence file, area, power, and delay overhead values, and multiple security metrics, described in the following.

IX. Security Analysis and Metrics

The robustness of any obfuscation technique must undergo an evaluation phase to ensure that the proposed methodology can resist all known or potential attacks. Quantifiable metrics make it easier to understand the level of complexity against RE attacks on IP. To quantify the security and robustness of our proposed obfuscation algorithm, we present two metrics that prove the security or resistance of an IP against brute-force RE attacks. We also explain complexity against two more sophisticated structural attacks well-suited to state-space obfuscation along with the complexity of unroll-and-SAT.

A. Brute-Force RE Attack Complexity

A naive attacker with zero to little prior intelligence of the obfuscated design is forced to reverse engineer the design using brute force. Based on the level of access to the obfuscated design, we can devise two types of brute-force attacks and their corresponding complexity to succeed in attempting RE and recovering the IP functionality.

1) Black-Box Attack: In this brute-force attack, we assume that an attacker has access to two instances of fabricated integrated circuits (ICs) where one of the instances is unlocked (golden reference or oracle) and the other is obfuscated. We also assume that the attacker's ability is limited to accessing the I/O interface only and not seeing the underlying logic gates and wires. Therefore, the attacker must apply random input patterns to unlock the obfuscated design.

Complexity Metric: If the obfuscated design has I primary inputs (excluding clock and reset), the length of the key sequence is Q, and the width of ith key is P_i $(1 \leq i \leq Q)$, black-box attack complexity has

$$\text{BBC} = \prod_{i=1}^{Q} {}^{I}C_{P_i} \times 2^{P_i}. \tag{3}$$

BBC quantifies the number of trials an attacker has to go through to find the entire correct key sequence required to unlock the locked chip. The combinatorial component enumerates the number of trials to figure out the correct key (primary) input port combinations, whereas the exponential component enumerates the number of possible key combinations. In each trial, the attacker must verify the equivalence of the outputs from both instances of the ICs. BBC increases exponentially with increment in overall key size (P_i and/or Q).

2) White-Box Attack: In this attack, the attacker can access an obfuscated gate-level netlist or layout and a functioning IC (oracle). We assume that the attacker can see the underlying logic gates and wires since the design under attack is a gate-level netlist or layout. The attacker is also aware of the structural differences between a state flip-flop and a data-path flip-flop, which helps the attacker to focus only on the state flip-flops. Furthermore, we assume that the attacker knows the number of added flip-flops for obfuscation (this is a powerful assumption that favors the attacker).

Complexity Metric: If the number of state flip-flops before obfuscation is n and the number of added flip-flops is r, white-box attack complexity has

$$\text{WBC} = {}^{n+r}C_r \times 2^{r+I}. \tag{4}$$

WBC quantifies the number of trials an attacker needs to go through to find the right set of the added flip-flops and force the set of flip-flops to one of the enable states (or one combined enable state for all added FSMs) that enables the normal mode of operation. The combinatorial represents the number of possible sets of added flip-flops that increases overwhelmingly with a change in distributions of n and r. In contrast, the exponent represents the complexity of combining the one preenable state and the input key to make the transition to enable region that increases exponentially with an increase in the number of added flip-flops. Since we are favoring the attacker by assuming access to the value of n and r, (4) calculates the minimum number of trials, which does not consider the factor of finding the value of n or r.

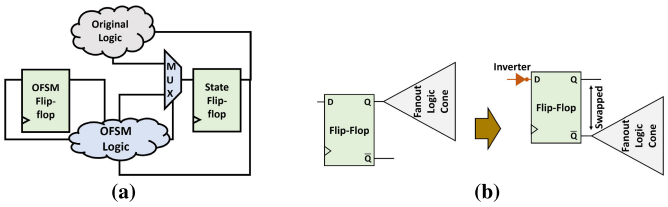


Fig. 12. Modifying existing state flip-flop to increase security benefits. (a) OFSM design can include the existing flip-flop. (b) Swapping flip-flop output logic cones to change the initial state from 0 to 1 without altering the functionality to prevent initialization attack.

B. Resistance Against DANA Attack

DANA [17] is the latest structural attack that can reduce the white-box attack complexity by analyzing the flop-to-flop connectivity and isolating the added flip-flops. DANA bypasses the combinatorial part of WBC by structurally analyzing the flop-to-flop connection and clustering the added flip-flops. Such clusters can be further analyzed functionally to reconstruct the OFSMs/DFSMs [8]. To deceive the DANA attack, `ProtectIP` intentionally increases the number of false positives in the DANA clustering algorithm by using the existing flip-flops to design the OFSMs/DFSMs. This results in making existing flip-flops look structurally similar to the added FSM flip-flops, which results in either a huge cluster size that includes both original and added state flip-flops that requires a large number of states to be enumerated or fairly tiny clusters (two or two flip-flops per cluster) that does not give away any information about the structure of the FSMs. Fig. 12(a) shows the structure of the existing state flip-flop once it is integrated with OFSM/DFSM.

Complexity Metric: We derive a metric called Impurity that quantifies the inverse of "purity" metric described in [17] where Impurity = [0, 1]. Higher Impurity (≈1) is an indicator of defeating DANA, whereas an Impurity value close to 0 is expected for a successful clustering attack. We use the following formula to calculate Impurity:

$$\text{Impurity} = 1 - \frac{K_d}{K_o} \tag{5}$$

where K_d is the number of clusters reported by DANA that includes exact register grouping and K_o is the number of FSMs (OFSM and DFSM) added through obfuscation.

C. Resisting Initialization Attack

In line with the prior discussion in Section VII, it is fairly easy to guess or identify the initial state of a design. Usually, flip-flop types in a gate-level design are dictated by the reset or initial state of each flip-flop (variable in RTL). For example, if a variable uses asynchronous reset logic to be reset to 0, it will mostly be synthesized using an asynchronous clear flip-flop. Hence, by analyzing the netlist and looking into the types of flip-flops, it is easy for an adversary to figure out the initial state and force the flip-flops to have those

TABLE I
STATISTICS OF PREOBFUSCATED AND POSTOBFUSCATED CEP IPS (EXTRACTED AND OBFUSCATED BY PROTECTIP UNDER THE SAME OBFUSCATION CONSTRAINTS)

Benchmarks	No. of Inputs	No. of Outputs	No. of Gates	No. of Existing Flip-flops				No. of Added Flip-Flops	Length of Key Sequence	Size of key (bits)	Overhead* (%)		RE Complexity Metrics		
				^{S}FF	^{P}FF	^{D}FF	Total				*Area*	*Timing*	BBC	WBC	Impurity
aes-192	323	129	321038	325	1	9056	**9382**	121	140	283	23.39	-0.5	10^{85}	10^{149}	0.94
des3	236	65	3937	198	1	0	**199**	8	4	126	33.84	3.49	10^{37}	10^{17}	0.50
dft	67	65	216588	34779	3917	65	**38761**	1274	1279	2566	31.55	-1.28	10^{772}	10^{2833}	0.96
fir	34	32	2701	32	384	32	**448**	8	4	155	10.0	0.0	10^{46}	10^{19}	1.00
gps	9	270	325397	664	10	9056	**9730**	202	209	443	5.92	-8.53	10^{133}	10^{265}	1.00
idft	67	65	215575	34779	3885	65	**38729**	1274	1279	2566	29.69	5.14	10^{772}	10^{2833}	0.50
iir	34	32	4238	256	384	32	**672**	8	4	155	19.92	4.42	10^{46}	10^{19}	0.50
md5	516	130	9004	267	1	1	**269**	10	12	273	22.69	-6.99	10^{82}	10^{22}	0.50
sha-256	516	258	11570	1040	0	0	**1040**	33	31	139	37.07	-0.15	10^{41}	10^{73}	0.33

* Under 5% delay overhead constraint.

values to the output of the flip-flop (using scan chain). This is called initialization attack. This attack can bypass state-space obfuscation logic (or the key traversal) to reach the initial state. Therefore, ProtectIP incorporates randomizing the initial state by swapping the output nets between two output ports and adding an inverter to the input port of random flip-flops. Swapping allows flipping the initial states of those particular flip-flops. The implementation is shown in Fig. 12(b), which swaps the initial state of a flip-flop from 0 to 1. The advantage of swapping the output of the flip-flop is the exponentially increased complexity of initializing a design to its initial state, while the scan chain is accessible, bypassing the obfuscation logic (assuming that the attacker can isolate the original state flip-flops). Suppose that a preobfuscated design has n number of state flip-flops. In that case, the complexity to correctly initialize or force all the flip-flops to start from their initial state will be 2^n after swapping. This is an additional security feature that increases RE further.

D. Analysis Against SAT Attack

SAT cannot be directly applied to the sequential design, locked either using combinational or sequential logic locking. The obfuscated designs need to be unrolled first to create the combinational logic representation of the design. This unroll-and-SAT is not scalable for larger designs even with small FSMs with less reachability [31]. In contrast, the proposed technique increases the FSM size (structurally in a number of flip-flops and functionally in state space) exponentially, which makes unroll-and-SAT even more difficult.

X. RESULTS AND DISCUSSION

In this section, we discuss the implementation of state-space obfuscation algorithm using ProtectIP on open-source large SoC IPs. We also analyze and quantify the RE complexity of the obfuscated designs using security metrics and show that ProtectIP algorithm is secure and robust against all known FSM RE attacks.

A. Benchmarks and Features

Most of the earlier locking and attacks have been performed on ISCAS [32], [33] and other smaller open-source benchmarks. To demonstrate the implementation of state-space obfuscation, we ran ProtectIP tool on common evaluation platform (CEP) [34] IPs. These multimillion transistor logic blocks are representatives of commercial IPs due to their sizes and application domains (cryptography, digital signal processing, and so on) and also help exhibit the scalability of ProtectIP. We ran the tool to extract the features of the designs that include the estimation of different types of flip-flops in the synthesized gate-level netlists. Our entire flow includes using LEDA 250-nm standard cell library and Design Compiler (DC) tool for logic synthesis and Verilog Compiler and Simulator (VCS) for functional simulation, all from Synopsys. As presented in Table I, ProtectIP is very scalable and can handle designs with almost half a million standard cells.

B. Obfuscated Benchmark Preparation

Based on the extracted features, a user can impose fine-grained obfuscation constraints that meet the target needs from a cost and security point of view. However, we used the same obfuscation constraints for all designs to make the obfuscation process transparent and the effect of constraints on different types and sizes of designs. The statistics of the obfuscated CEP IPs have been presented in Table I, where we used 128-bit as the requested key size and 4-to-6-bit FSMs. We chose 1% of the existing nodes for modification cell insertion, and maximum fan-out was constrained at 8. The obfuscated benchmarks are available in the public domain [35].

By analyzing the results, we can see that key size and sequence length are directly proportional to the number of flip-flops in a given design, as the number of added FSMs (flip-flops) is dependent on the number of state flip-flops. In addition, with an increase in the key size, BBC increases, whereas WBC is dependent on the number of added flip-flops. Note that configuration parameters are meant to be guides for the obfuscation process. ProtectIP by default does not use any of the parameters as absolute requirements and has the freedom to randomize the obfuscation logic (e.g., the width of the key in each transition and length of the unlocking sequence). Therefore, the final key size may not meet the requested key size. However, a user can enforce meeting the constraints by overriding the freedom of randomization.

C. Evaluation of Security Against RE Attacks

As part of the security evaluation, ProtectIP generates its security metrics. Table I includes BBC and WBC, reported by ProtectIP, and one additional metric Impurity derived

TABLE II

COMPARISON OF PROTECTIP WITH EXISTING FSM OBFUSCATION TECHNIQUES IN TERMS OF ATTACK RESISTANCE. ✓ IMPLIES RESISTANT TO ATTACK, WHEREAS ✗ DENOTES VULNERABLE TO ATTACK

Sequential Obfuscation Algorithm	*SAT Resistant*	*KC2 Resistant*	*REFSM Resistant*	*DANA Resistant*	*Withstands Scan-Based Attacks with No Additional Protection*
HARPOON	✓	✓	✗	✗	✗
BFSM	✓	✓	✗	✗	✓
Encrypt FF	✗	✗	✗	✗	✗
Dynamic state deflection	✗	✗	✗	✗	✓
CA-guided	✗	✗	✓	✓	✓
HST	✓	✓	✗	✗	✗
ProtectIP	✓	✓	✓	✓	✓

from DANA [17] attack. We have provided a comparative analysis between existing FSM obfuscation techniques in terms of attack resilience in Table II.

1) BBC: The reported BBC metric values in Table I are not direct representations of the corresponding (3) discussed earlier. ProtectIP uses a generalized form of BBC with assumptions of the same width of the key and the same set of inputs as key inputs for each key value of the entire sequence. Hence, (3) can be simplified to $\text{BBC} = 2^{\text{key\_size}} = 10^{\text{key\_size}/3.322}$. Therefore, BBC increases exponentially with an increase in the key size, which is evident from the BBC values of the obfuscated IPs.

2) WBC: Similarly, WBC numbers do not reflect (4). To avoid large combinatorial calculation, ProtectIP uses the Kamenetsky formula [36] to approximate the complexity values. As evident from the data in Table I, WBC values are guided by the number of added flip-flops and are computationally infeasible for bigger designs with a large number of flip-flops.

3) DANA Impurity: We also ran the DANA tool, available in public domain [37], in steered mode on our minimally obfuscated designs to demonstrate the worst case for a defender but the best case for the attacker and analyze the success rate in RE. As DANA reports the clusters of flip-flops where each cluster represents a multibit register (either data path or control path), we extracted the clusters using the tool. We checked the percentage of the clusters that belong to the added OFSMs/DFSMs. We have used (5) to calculate the Impurity value based on our analysis. The Impurity values from Table I imply that a fairly smaller portion of the minimally obfuscated designs could be recovered using the clustering approach of DANA. Although DANA potentially reported several clusters, the clusters either contained a mix-up of added and existing flip-flops or included scattered clusters of one single OFSM (or DFSM). The lowest value we got is 0.33, demonstrating a low attack success rate and signifies the RE complexity of implementing bare-minimum obfuscation. However, the Impurity can be further increased for the low-impurity designs by reusing the existing flip-flops to design the OFSMs. It is worth mentioning that successful clustering can only lower the WBC. The reconstruction of FSMs from each cluster still has exponential complexity that must be solved to recover a fully functional design. To summarize, higher Impurity implies higher difficulty and is good for the defender (IP owner) and bad for an attacker. In essence of that, the lowest Impurity (0.33 among all the listed benchmarks) is the best case for an attacker, which is the preprocessing step to reverse engineer the state space. Without proper clustering, full RE is infeasible due to the exponential size of the state space.

XI. CONCLUSION

In this article, we have presented a practical automated state-space obfuscation methodology that addresses the scalability issue and adds exponential levels of complexity in performing successful RE attacks on hardware IP. We have also presented a CAD tool, ProtectIP, that implements this methodology and integrates it into commercial tool flow. Although the fundamental approach of increasing state-space enumeration and reachability enhancement faces the issue of scalability, ProtectIP addresses these challenges with innovative low-complexity algorithmic solutions. We have proposed effective metrics that can be used to quantify the level of security for an obfuscated design against RE attacks. We have also performed extensive security evaluation using external tools to show the efficacy of the algorithm in protecting hardware IPs against known attacks on sequential logic obfuscation for a suite of large-scale design benchmarks. Future directions of this work will include an extension to higher level (such as register-transfer level) design descriptions and integration of state-space obfuscation with complementary techniques, such as logic locking and hardware redaction.

REFERENCES

[1] E. Castillo, U. Meyer-Baese, A. Garcia, L. Parrilla, and A. Lloris, "IPP@HDL: Efficient intellectual property protection scheme for IP cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 578–591, May 2007.

[2] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *Proc. IEEE 20th Int. On-Line Test. Symp. (IOLTS)*, Jul. 2014, pp. 49–54.

[3] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with Hamming distance-based restore unit (SFLL-hd)—Unlocked," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 10, pp. 2778–2786, Oct. 2019.

[4] A. Alaql and S. Bhunia, "SARO: Scalable attack-resistant logic locking," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3724–3739, 2021.

[5] 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.

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

[7] L. Li and H. Zhou, "Structural transformation for best-possible obfuscation of sequential circuits," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 55–60.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] D. Duvalsaint, Z. Liu, A. Ravikumar, and R. D. Blanton, "Characterization of locked sequential circuits via ATPG," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Sep. 2019, pp. 97–102.

[15] H. Yotsuyanagi and K. Kinoshita, "Undetectable fault removal of sequential circuits based on unreachable states," in *Proc. 16th IEEE VLSI Test Symp.*, Apr. 1998, pp. 176–181.

[16] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 534–539.

[17] N. Albartus, M. Hoffmann, S. Temme, L. Azriel, and C. Paar, "DANA universal dataflow analysis for gate-level netlist reverse engineering," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 309–336, Aug. 2020.

[18] M. Fyrbiak et al., "On the difficulty of FSM-based hardware obfuscation," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 293–330, Aug. 2018.

[19] R. Karmakar, S. Chattopadhyay, and R. Kapur, "A scan obfuscation guided design-for-security approach for sequential circuits," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 3, pp. 546–550, Mar. 2020.

[20] R. Karmakar, S. S. Jana, and S. Chattopadhyay, "A cellular automata guided two level obfuscation of finite-state-machine for IP protection," *Integration*, vol. 74, pp. 93–106, Sep. 2020.

[21] A. Saha, H. Banerjee, R. S. Chakraborty, and D. Mukhopadhyay, "ORACALL: An oracle-based attack on cellular automata guided logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2445–2454, Dec. 2021.

[22] K. Juretus and I. Savidis, "Synthesis of hidden state transitions for sequential logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 1, pp. 11–23, Jan. 2021.

[23] 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. 292–303.

[24] 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. 1323–1328.

[25] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, "Gate-level netlist reverse engineering for hardware security: Control logic register identification," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1334–1337.

[26] N.-F. Standard, "Announcing the advanced encryption standard (AES)," *Federal Inf. Process. Standards Publication*, vol. 197, p. 3, Nov. 2001.

[27] E. Dubrova, "A list of maximum period NLFSRs," Cryptol. ePrint Arch., Paper 2012/166, 2012. [Online]. Available: https://eprint.iacr.org/2012/166

[28] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.

[29] S. Engels, M. Hoffmann, and C. Paar, "The end of logic locking? A critical view on the security of logic locking," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 796, Jan. 2019.

[30] A. Kuehlmann and C. A. van Eijk, "Combinational and sequential equivalence checking," in *Logic synthesis and Verification*. Berlin, Germany: Springer, 2002, pp. 343–372.

[31] H. Pearce, R. Karri, and B. Tan, "High-level approaches to hardware security: A tutorial," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 3, pp. 1–40, 2023.

[32] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmarkcircuits and a targeted translator in FORTRAN," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Special Session ATPGand Fault Simulation*, Jun. 1985.

[33] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1989, pp. 1929–1934.

[34] MI Technology. *Common Evaluation Platform*. Accessed: Mar. 2019. [Online]. Available: https://github.com/mit-ll/CEP

[35] *Obfuscated Benchmarks: Sequential Obfuscation*. Accessed: Aug. 2020. [Online]. Available: https://cadforassurance.org/benchmarks/sequential-obfuscation-benchmark/

[36] Dmitry Kamenetsky. *Kamenetsky Formula*. Accessed: Dec. 2019. [Online]. Available: https://mathoverflow.net/questions/19170/how-good-is-kamenetskys-formula-for-the-number-of-digits-in-n-factorial

[37] *DANA—Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering*. Accessed: Jan. 2021. [Online]. Available: https://cadforassurance.org/tools/reverse-engineering-and- visualization/dana/

Md Moshiur Rahman (Member, IEEE) graduated in electrical and electronic engineering from the Ahsanullah University of Science and Technology (AUST), Dhaka, Bangladesh, in 2016. He is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering (ECE), University of Florida (UF), Gainesville, FL, USA, under the supervision of Prof. Swarup Bhunia.

He is also working as a Graduate Research Assistant at the Department of Electrical and Computer Engineering, UF. His research interests include hardware intellectual property (IP) protection, countermeasures against reverse engineering, Trojan resilient IP design, computer-aided design, and electronic design automation.

Swarup Bhunia (Senior Member, IEEE) received the B.E. degree (Hons.) from Jadavpur University, Kolkata, India, in 1995, the M.Tech. degree from IIT Kharagpur, Kharagpur, India, in 1997, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

He was appointed as the T. and A. Schroeder Associate Professor of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland, OH, USA. He is currently a Professor and the Semmoto Endowed Chair with the University of Florida, Gainesville, FL, USA. He has more than ten years of research and development experience with more than 200 publications in peer-reviewed journals and premier conferences. His research interests include hardware security and trust, adaptive nanocomputing, and novel test methodologies.

Dr. Bhunia received the SRC Technical Excellence Award as a Team Member in 2005, the Semiconductor Research Corporation Inventor Recognition Award in 2009, the National Science Foundation Career Development Award in 2011, the IBM Faculty Award in 2013, and several best paper awards/nominations. He has served as a Guest Editor for IEEE DESIGN & TEST OF COMPUTERS in 2010 and 2013 and IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS in 2014. He has been serving as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, and *ACM Journal on Emerging Technologies in Computing Systems*.