Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TCAD.2024.3387873

STATION: State Encoding-based Attack-Resilient Sequential Obfuscation

Zhaokun Han, *Graduate Student Member, IEEE,* Aneesh Dixit, *Graduate Student Member, IEEE,*
Satwik Patnaik, and Jeyavijayan (JV) Rajendran, *Senior Member, IEEE*

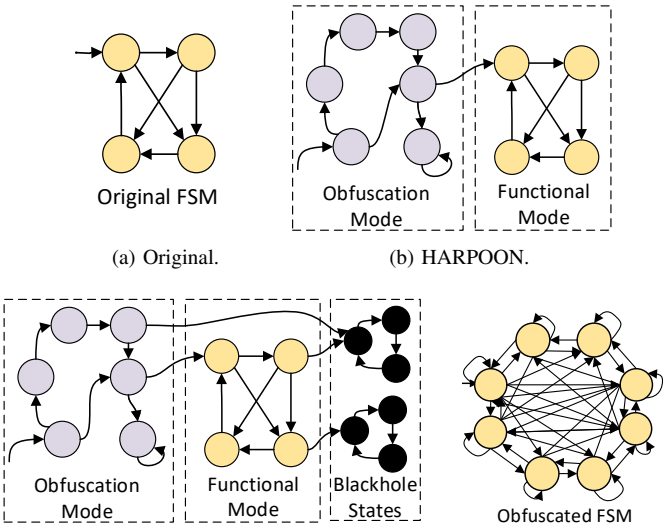
***Abstract*—The unauthorized duplication of design intellectual property (IP) and illegal overproduction of integrated circuits (ICs) are hardware security threats plaguing the security of the globalized IC supply chain. Researchers have developed various countermeasures such as logic locking, layout camouflaging, and split manufacturing to overcome the security threat of IP piracy and unauthorized overproduction. Logic locking is a holistic solution among all countermeasures since it safeguards the design IP against untrusted entities, such as untrusted foundries, test facilities, or end-users throughout the globalized IC supply chain. There are well-known logic locking techniques for combinational circuits with well-established security properties; however, their sequential counterparts remain vulnerable. Since most practical designs are inherently sequential, it is essential to develop secure obfuscation techniques to protect sequential designs. This paper proposes a sequential obfuscation technique, *STATION*, building on the principles of finite state machine encoding schemes. *STATION* is resilient against various attacks on sequential obfuscation—input-output (I/O) query attacks and structural attacks, including the ones targeting sequential obfuscation—which have broken all state-of-the-art sequential obfuscation techniques. *STATION* achieves good resilience and desired security against various I/O and structural attacks, which we ascertain by launching 9 different attacks on all tested circuits. Moreover, *STATION* ensures tolerable overheads in power, performance, and area, such as 8.75%, 1.22%, and 5.63% on the largest tested circuit, containing 102 inputs, 7 outputs, 6.1×10^4 gates, 7 flip flops, 100 states, and 3.0×10^3 transitions.**

***Index Terms*—IP protection, Logic locking, Sequential obfuscation, Finite state machine, FSM encoding**

I. Introduction

A. Logic Locking for Intellectual Property (IP) Protection

THE globalized integrated circuit (IC) supply chain is a critical facet of the modern semiconductor industry, and this complex supply chain has led to multiple entities operating in various geographic regions. Currently, design houses account for $143 billion in added value in a supply chain with a value of $444.5 billion [1]. Alarmingly, the increasing presence of third-party entities also provides pathways for malicious entities to pirate the design intellectual property (IP) or engage in illegal IC overproduction. Multiple countermeasures, such as IC camouflaging [2], [3], split manufacturing [4], and logic locking [5], have been proposed to restrict illegal activities while retaining the benefits of the globalized IC supply chain. Among these countermeasures, logic locking promises the most comprehensive and practical solution against threats emerging from untrusted foundries, test facilities, and end-users [5]–[7]. Thereby, logic locking has been set for industry adoption, examples of which include the Automatic Implementation of Secure Silicon (AISS) program supported by Synopsys and Defense Advanced Research Projects Agency (DARPA) [8] and Structured Array Hardware for Automatically Realized Applications (SAHARA) program supported by Intel and DARPA [9].



(a) Original. (b) HARPOON. (c) DSD. (d) *STATION*.

Fig. 1. FSMs of (a) original design, (b) HARPOON protected design, (c) DSD protected design, and (d) *STATION* protected design (**Our work**).

One can broadly classify the logic locking techniques into combinational and sequential locking (a.k.a. sequential obfuscation). Combinational locking techniques insert extra logic driven by additional input ports, referred to as key inputs. A tamper-proof memory stores the correct key and drives the key inputs. Only the correct key recovers the original functionality; applying any arbitrary incorrect key to the locked design leads to corrupted/incorrect functionality. Similarly, sequential obfuscation modifies the finite state machine (FSM) of the design by hiding specific states or state transitions [6], [10]–[12]. Here, a key can also be an input sequence to the FSM.

Zhaokun Han, Aneesh Dixit, and Jeyavijayan (JV) Rajendran are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (email: hzhk0618@tamu.edu; aneeshdixit@tamu.edu; jv.rajendran@tamu.edu).

Satwik Patnaik was with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, USA. He is now with the the Department of Electrical and Computer Engineering, University of Delaware, Newark, USA (email: satwik@udel.edu).

B. Limitations of Combinational Locking

Before explaining the various defenses and attacks on sequential obfuscation techniques, we state the limitations of combinational locking techniques and motivate the need for better sequential obfuscation techniques. The seminal SAT-based attack [13] broke most combinational locking techniques, leading researchers to develop SAT-resilient techniques (also called post-SAT techniques). Among these SAT-resilient techniques, some aim to increase the number of iterations required by the SAT attack, which are known as point function-based techniques, utilizing a point function[1] to make the SAT attack eliminate only a few incorrect keys per SAT iteration, forcing the performance of the SAT attack close to brute-force attacks [7], [14]. Other post-SAT techniques aim to increase the run time of each SAT iteration [15], [16], which are also called SAT-hard techniques. However, the point function-based techniques are vulnerable to structural attacks [3], [17]–[19], and the SAT-hard techniques are broken by neural network-guided attacks [20]. Another major drawback of the point function-based techniques is their output error rate, which is exponentially small with respect to the key-size [21], [22]. On the other hand, sequential obfuscation techniques offer a higher output error rate than point function-based techniques [23], [24]. Additionally, since most practical designs are inherently sequential, it is essential to develop secure obfuscation techniques.

C. Prior Works on Sequential Obfuscation: Defenses

We now provide an overview of prior defense techniques. For a detailed summary of all the techniques, we refer the reader to Chakraborty *et al.* [34]. *Active Metering* [10] augments the original state register with additional flip-flops (FFs), thereby increasing the number of states. An input sequence is used as the key. The design only enters the correct initial state after applying the correct key sequence. After that, the design traverses the FSM correctly. *HARPOON* [6] augments an FSM with additional states and has two modes of operation: an obfuscation mode and a functional mode. The FSM functions correctly only when states are correctly traversed through the obfuscation mode and successfully enter the initial state of the functional mode. *Interlocking* [11] introduces multiple transitions between the obfuscated and functional modes. *Dynamic State Deflection (DSD)* [12] ensures that the state transitions are deflected from the original transition to a blackhole state when applying an incorrect key. Blackhole states are states with no outward transitions in the FSM. Thus, upon entering a blackhole state, the design cannot go into functional mode. *State, connectivity, and routing augmentation model for building logic encryption* (*SCRAMBLE*) [30] obfuscates FFs or logic with a shuffling/re-permutation block to protect the input signals of FFs. However, SCRAMBLE assumes that the attacker cannot access the scan chain; if the scan chain is open to the attacker, the combinational logic of a SCRAMBLE-protected design is vulnerable to attacks (e.g., SAT attack [13]). Another obfuscation technique, *synthesis of Hidden State Transitions (HSTs)* [31], introduces "clock glitch" to increase the search space in the obfuscated FSM for the attacker by changing the frequency of the clock signal [18]. However, the operation of "clock glitch" on the clock signal [31] is equivalent to modifying the combinational logic, and it is unclear whether the technique achieves resilience against the SAT attack on the combinational logic [13]; moreover, one research work, *JANUS* [33], states that HST contains potential vulnerabilities against structural attacks on FSM [28], [29]. As a more robust version of HST [31], coupling capacitance based HST [32] inserts clock glitching at the physical design level, masking the location of the HST triggers. Therefore, this operation hinders reverse engineering from the protected physical design to its gate-level netlist. As a result, coupling capacitance based HST protects the design against the BMC and KC2 attacks. Unlike previous approaches, JANUS [33] protects the sequential design while offering a high error rate by configuring the functionality of the FFs using a dynamic key. The key decides whether the FF is a D-type FF, whose output is the same as the input signal, or a T-type FF, whose output is kept/toggled compared to the previous state when the input signal is 0/1. In this way, JANUS hides the transition from the current state to the next state. However, there is no experimental result showing the resilience of JANUS against various attacks, such as input-output (I/O) query attacks [13], [25], [26] and structural attacks [3], [17]–[19], [28], [29]. *TriLock* [24] uses a point function and state re-encoding to protect the design against multiple attacks [27]–[29] while keeping a high error rate. However, structural vulnerabilities still exist due to the usage of point functions, as an attacker can analyze the structure of the combinational part of the obfuscated sequential design and further recover the design [3], [17]–[19].

D. Prior Works on Obfuscation: Attacks

Multiple attacks have been developed to overcome sequential obfuscation. Some attacks target breaking the combinational part of the sequential design, while sequential attacks analyze the FSM to remove sequential obfuscation protections. Besides, broadly, these attacks can be classified into: (i) input-output (I/O) query attack, where an attacker applies inputs to the functional chip and observes its output to search for the key based on the analysis of the locked netlist, and (ii) structural attacks, where an attacker analyzes the structure of the locked netlist or the FSM to remove the protection or extract the correct key.

Attacks on combinational locking techniques. We now explain the different attacks targeting the combinational part of the sequential netlist, thereby removing the protection provided by sequential obfuscation techniques.

For I/O attacks, *Boolean Satisfiability (SAT) attack* [13] targets combinational locking techniques. The SAT attack builds a miter circuit and detects distinguishing input patterns (DIPs), which are input patterns to eliminate incorrect keys. The SAT attack effectively breaks all the locking techniques prior to the SAT attack (a.k.a pre-SAT locking techniques) [5], [35]–[37]. The success of the SAT attack led to the development of defenses after the SAT attack (a.k.a post-SAT

[1]A point function is a single-output Boolean function that has a single or a few input pattern(s) resulting in output as 1.

TABLE I
STATE-OF-THE-ART SEQUENTIAL OBFUSCATION TECHNIQUES AND ATTACKS. ATTACKS ARE CATEGORIZED AS INPUT-OUTPUT (I/O) AND STRUCTURAL ATTACKS. ✓/✗ DENOTES A SUCCESS/FAILURE OF THE DEFENSE TECHNIQUE, N/A IS NOT APPLICABLE, AND ∗ SHOWS NO ANALYSIS

Attack / Defense	I/O attack			Structural attack				
	SAT [13]	**BMC, KC2 [25], [26]**	**Fun-SAT [27]**	**SPS [17]**	**ATR [3]**	**FALL [18]**	**SPI [19]**	**SCC [28], [29]**
Active Metering [10]	N/A	∗	✗	N/A	N/A	N/A	N/A	✗
HARPOON [6]	✗	✗	✗	N/A	N/A	N/A	N/A	✗
Interlocking [11]	N/A	∗	✗	N/A	N/A	N/A	N/A	✗
Dynamic State Deflection [12]	N/A	∗	✗	N/A	N/A	N/A	N/A	✗
SCRAMBLE [30]	✗	✓	∗	N/A	N/A	N/A	N/A	✓
HST [31], [32]	∗	✓	∗	N/A	N/A	N/A	N/A	∗
JANUS [33]	∗	∗	∗	∗	∗	∗	∗	∗
TriLock [24]	✓	✓	✓	∗	∗	∗	∗	✓
STATION	✓	✓	✓	✓	✓	✓	✓	✓

techniques) [3], [7], [14], [38]. These techniques use specific functions, such as point functions, that force the SAT attack to use an exponential number of DIPs in key-size. However, many of these techniques are vulnerable to structural attacks described below.

For structural attacks, *Signal Probability Skew (SPS)* [17] detects and bypasses/removes the point function logic, thereby recovering the design with its original functionality. *AND-Tree Removal (ATR) attack* [3] searches for multi-input AND tree components and removes that logic to get a functional design. *Functional Analysis attacks on Logic Locking (FALL)* [18] does structural analysis to locate the gates in the circuit leading the subcircuits whose Boolean functions are unate[2] and collect a list of possible keys. *Sparse Prime Implicant (SPI) attack* [19] analyzes the prime implicants (PIs) on the PI table (PIT) of the locked circuit. The correct key is extracted from the PIT if the circuit is protected by the point function-based techniques (*e.g.,* SAT attack [13]), no matter what is the structure of the locked netlist is.

Attacks on sequential obfuscation techniques can also be categorized into I/O attacks and structural attacks.

For I/O attacks, such as *Key Condition Crunching (KC2) attack* [26], it reduces the attack complexity by using incremental bounded-model checking and simplifying key conditions. *Fun-SAT attack* [27] estimates the minimum number of unrollings and accelerates the search for the correct key. It uses the notion of functional corruptibility combined with model checking to verify a candidate key. A monotonically increasing functional corruptibility on every unrolling implies the elimination of more incorrect key sequences.

For structural attacks, such as *Strongly Connected Component (SCC) attack* [28], [29], it analyzes the topology of the obfuscated netlist/FSMs, maps the individual FFs into their respective FSMs, and removes the FFs inserted by the obfuscation technique. The sequential obfuscation techniques [6], [10]–[12] discussed in Sec. I-C are vulnerable to the SCC attack [28]. The vulnerability exists because these sequential obfuscation techniques insert additional states into the FSM, usually clustered into one or multiple groups/clusters and distinguished from the states of the original FSM. These different clusters are connected by one or multiple directed transition(s), as shown in Fig. 1(b)–(c). The SCC attack easily detects and exploits these directed transitions between different clusters. It then bypasses these directed transitions to remove the clusters except the one belonging to the original FSM, thereby recovering the original FSM. The foundation of these techniques is based on hiding the directed transitions from obfuscation mode to functional mode. If these transitions are detected, then the designs are broken.

Table I summarizes some of the state-of-the-art sequential obfuscation techniques that are empirically vulnerable to structural attacks or I/O attacks. Some sequential obfuscation techniques, such as HARPOON, are key-free in implementation, where the key sequence is provided through the primary inputs. This type of key-free obfuscated design has an equivalent design with key-input ports; thus, the attacker can apply combinational attacks, such as the SAT attack on the obfuscated design. Meanwhile, advanced sequential obfuscation techniques, including JANUS [23], [33], TriLock [24], SCRAMBLE [30], and synthesis of HSTs [31], have certain vulnerabilities, as mentioned in Sec. I-C. Hence, in this work, we address the following question: *Can we develop a holistic sequential obfuscation technique that can provide a comprehensive defense against both I/O and structural attacks?*

E. This Work

Our approach protects FSMs such that it groups all the states into one cluster to defend against the SCC attack [28], [29] (see Fig. 1(d)). To this end, we develop *DisJoint*, an FSM encoding scheme for sequential obfuscation. We then apply a combinational locking technique to the state-transition block of this encoded FSM; we call this approach *STATION*. Our contributions are:

- Our defense technique, *STATION*, is the first defense technique to combine fundamental concepts from combinational locking with FSM encoding schemes from sequential circuits to develop a sequential obfuscation technique.
- We discuss how our encoding technique, *DisJoint*, can easily integrate with modern logic synthesis tools and overcome the vulnerabilities created by sequential synthesis tools.
- We introduce the concept of protected state transitions to the research community. We expect that the notion of protected state transitions shall prove helpful in quantifying structural resilience.

[2]An n-input 1-output Boolean function $f(x_1, x_2, \cdots, x_n)$ is unate means that, for each $x_i \in \{x_1, x_2, \cdots, x_n\}$, it is always true with either $f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \leq f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$ or $f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \geq f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$, $\forall (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) \in \mathbb{B}^{n-1}$ ($\mathbb{B} = \{0, 1\}$) [18].

- Our encoding scheme aids the designer by providing greater flexibility. We demonstrate this flexibility by allowing the designer to flexibly choose and protect state transitions.
- *STATION* has the same time complexity as the synthesis step, converting from the original FSM to the locked netlist. Thus, integrating *STATION* into the IC design flow only incurs a negligible time effort, especially for large-scale designs.
- We showcase the strength of *STATION* by demonstrating the resilience against SAT [13], BMC [25], KC2 [26], Fun-SAT [27], SPS [17], ATR [3], FALL [18], SPI [19], and SCC [29] attacks on both academic standard benchmark circuits and large-scale synthetic circuits, including the state-of-the-art I/O and structural attacks.
- Our technique limits the overheads for large-scale designs. The average power, performance, and area (PPA) overheads are 7.00%, 0.15%, and 8.69%, respectively.

Paper Organization. The remainder of the paper is organized as follows. We provide a brief background in Sec. II. Sec. III describes the methodology of *STATION*. We present and analyze various results in Sec. IV. Finally, discussions and concluding remarks are provided in Sec. V and Sec. VI.

II. Background and Preliminaries

In this section, we explain the underlying combinational locking technique used in *STATION*, stripped-functionality logic locking (SFLL), and the FSM encoding principles.

A. Stripped-Functionality Logic Locking (SFLL)

We now explain the underlying combinational locking technique that is among the building blocks of *STATION*. SFLL is a family of logic locking techniques based on the principle of corrupt-and-correct (or strip-and-restore) [7]. An SFLL-protected circuit consists of a functionality-stripped circuit (FSC) and a restore unit, as shown in Fig. 2. The output of the FSC differs from the original circuit for specific input patterns known as protected input patterns (PIPs); for any input pattern that is not a PIP, the outputs of the FSC and the original circuit are identical. The restore unit corrects/restores the output of the FSC only when applied with the correct key. SFLL is provably-secure against I/O attacks, especially the SAT attack [13], [35], [39]. When the protected design contains p PIPs and a key-size of k, the resilience against I/O attacks is $\alpha = k - log_2 p$ [7]. When there are a few PIPs, the achieved α-security is close to k. With fewer PIPs, the SFLL-protected design has higher resilience against I/O attacks. Specifically, when $p = 1$, the design contains the highest security level with a certain key-size. At the same time, SFLL is empirically secure against most removal/structural attacks, such as SPS and ATR attacks [3], [17]. The attacker can remove SFLL blocks, such as the restore unit, to extract a recovered design using removal attacks [3], [17]. However, the functionality difference between the original and recovered designs is hard to detect without the original netlist. Further, the attacker cannot ensure and verify whether the recovery is a success or a failure, as discussed in Shamsi *et al.* [39]. Some structural attacks can extract the secret key from the FSC of SFLL, such as FALL and SPI attacks [18], [19]. FALL [18] breaks SFLL-HD using the unateness property. Additionally, the SPI attack breaks all SFLL techniques, including SFLL-fault [40] that keeps resilient against all prior structural attacks [41] (e.g., SPS [17], ATR [3], and FALL attacks [18]), by analyzing the PIT of the locked circuit's FSC to extract the PIPs [19].

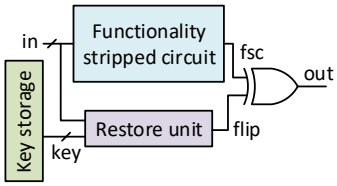


Fig. 2. General structure of a circuit protected by stripped functionality logic locking [7].

B. FSM Encoding Techniques

FSM is a representation of the functionality of a sequential design. Given the current state and the input, the FSM transitions to the next state and returns the corresponding output. Usually, these states are represented in a high-level language. A synthesis tool, such as *Synopsys Design Compiler* [42], *Cadence Genus* [43], and *Mentor Graphics Precision RTL* [44], converts the high-level description of the FSM into the gate-level netlist of the hardware. During the synthesis process, FSM encoding assigns a numeric value to each state. Some well-known encoding schemes include binary, gray, and one-hot encoding [45]. Multiple encoding schemes have been developed to achieve various design objectives. For example, gray encoding minimizes power consumption by reducing the switching activity while transitioning between states [45]. While encoding schemes for conventional circuit design objectives (*e.g.,* design overheads) are well known, there has not been significant interest in developing encoding schemes for modern circuit design objectives for the sake of security.

III. Proposed Techniques

In this section, we define the threat model and the challenges faced by sequential obfuscation techniques. We then present how *DisJoint* encoding and *STATION* are able to overcome the research challenges and withstand the complex threats environment. Finally, we discuss the complexity of *STATION* to display its practicality.

A. Threat Model

In this subsection, we state the attacker's objective, locations, and resources. We also state the resources available to the defender. Our threat model is consistent with existing sequential obfuscation techniques [6], [10]–[12], [33].

Attacker's Objective. The attacker aims to retrieve the unlocked design. To this end, the adversary attempts to gain the correct key or remove the logic blocks added by the obfuscation scheme.

Attacker's Location. The attacker could be an untrusted foundry, test facility, end user, or the collusion of multiple untrusted entities.

Attacker's Resources. Considering the attacker's location, the attacker can potentially access one or multiple resources, as listed below.

- The attacker can access the netlist of the locked design.
- The attacker can access a functional chip (a.k.a oracle), such as a chip legally purchased from the market.
- The adversary can know the employed defense technique. For example, for a *STATION*-obfuscated design, the attacker can know that its protection defense is *STATION* using the *DisJoint* encoding scheme.

Defender's Resources. The defender is located within the design house and knows the FSM's state-transition table. This is a valid assumption since the design house possesses the design specifications. The defender utilizes this information to protect the design.

B. Research Challenges

At first glance, it appears that one can use an existing secure combinational logic locking technique (*e.g.,* SFLL [7] using D2PIPs [19]) to defend against either I/O or structural attacks. Unfortunately, this task is not trivial to solve and requires our proposed scheme, *STATION*, to overcome the following challenges:

1) Existing combinational locking techniques operate at the gate level, whereas sequential obfuscation techniques operate at the FSM level. Thus, one needs to develop a methodology that spans both abstraction levels.
2) Security-agnostic logic synthesis tools simplify the state-transition table of the FSM by state minimization. However, this process creates structural vulnerabilities. Analogous logic minimization techniques have been known to create structural vulnerabilities in combinational logic locking [17], [19], [46]. To overcome such vulnerabilities, one needs to account for the effect of logic synthesis tools and develop techniques that can be easily integrated into the synthesis process.
3) The proposed technique should incur a reasonable overhead on practical circuits while guaranteeing security.
4) The proposed obfuscation technique needs to consider the complexity of implementation, such as how long it could take on average and how stable the protection is.

In the remaining sections, we elaborate on how *STATION* overcomes these challenges.

C. DisJoint Encoding and Analysis

Our first objective, *DisJoint* encoding, is to guarantee the presence of D2PIPs [19] in the state-transition logic and provide security against structural attacks. D2PIP is a PIP that has a distance of at least 2 to any ON-set minterm of the FSC. In our proposed obfuscation on FSM, if the Hamming distance between the encoding (binary values) of two states is at least 2, we call the corresponding state transition a protected state transition (PST). By definition, the logic of the chosen PST(s) can be considered analogous to D2PIP(s) of an SFLL-protected design. Unfortunately, PSTs are rare in practical circuits, making it difficult to ensure sufficient resilience. To increase the number of PSTs in the circuit, we modify each state's value using an FSM encoding scheme.

To this end, our proposed *DisJoint* encoding scheme encodes the states such that the distance between any two states in our encoding is at least 2. This satisfies the property of D2PIPs (*Dist2* property) [19], ensures the presence of PSTs in our scheme, and allows the designer to protect the transition(s) of their choice.

Algorithm 1 lists the proposed *DisJoint* encoding. It uses a SAT solver to search for the solution of *DisJoint* encoding. The input to the SAT solver is the original state-transition table and the number of state bits (numerically the same as the number of FFs) allowed by the designer. For every transition in the state-transition table, we assign an extra constraint: the distance between the resulting solution should be at least 2. If the solution of *DisJoint* encoding exists, the SAT solver returns it; otherwise, the designer has to increase the number of state bits (numerically the same as the number of FFs).

For example, Table II shows the same FSM in Fig. 4 with different encoding schemes. The 2nd column (corresponding to the original encoding) does not contain any PST, indicating the vulnerability to defend against structural attacks. On the other hand, we can observe that *DisJoint* encoding ensures each state has a distance of at least 2 from any other state, which means that, after *DisJoint* encoding, each state transition is a PST, and we can use the PST(s) to guarantee the resilience against structural attacks.

Tackling logic synthesis tools. As we have seen in the case of combinational logic locking, logic synthesis can undermine the security offered by the logic locking technique by leaving structural "traces" about the PIP or the correct key [18], [19]. We now explain why *DisJoint* encoding and *STATION* are able to hide such traces in obfuscated sequential designs.

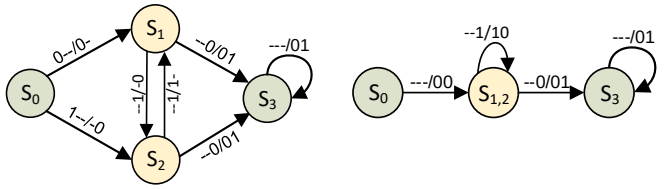
If two states are redundant (i.e., the states go to the same next state and return the same output for the same input pattern) [45], logic synthesis tools merge these two states into one state. For example, in Fig. 3(a), state s_1 and state s_2 are merged into $s_{1,2}$. Such merging can reduce the number of reachable states explored by the attacker (*i.e.,* the size of the clique in the obfuscated FSM is reduced).

STATION does not suffer from the above problem. SFLL corrupts the protected output to either 0 or 1. If the defender chooses to protect all pseudo outputs (*i.e.,* the inputs of the FFs), there exists the input pattern and the key reaching an arbitrary next state. Thereby, any of the 2^N states can be visited, where N is the number of FFs. This ensures that each state is unique because the next state can be any state based on the key: the FSM goes to the correct state for the correct key and all the other incorrect states for incorrect keys. Since the states are unique, synthesis tools will not merge them. This way, the FSM extracted from the *STATION*-obfuscated netlist is a clique, as shown in Fig. 1(d).

D. Overhead Analysis of DisJoint Encoding

In the above example (Table II), each state requires at least 3 bits while applying the *DisJoint* encoding scheme compared to the 2 bits in the original encoding. Correspondingly, the

Accepted Manuscript - Version of Record at: https://doi.org/10.1109/TCAD.2024.3387873



(a) (b)

Fig. 3. s_1 and s_2 in (a) are merged to $s_{1,2}$ in (b) during synthesis.

Algorithm 1: *DisJoint* encoding algorithm.

Input: State-transition table T, size of state bits x
Output: *DisJoint* encoding solution $DisJoint\_encoding$

```
CNF_0 ← ∅                                   //Constraint initialization
for transition_i ∈ T do
    s_cur, s_next ← Transition_parser(transition_i)
    CNF^cnst ← Distance_at_least_2(s_cur, s_next, x)
    CNF_0 ← CNF_0 ∧ CNF^cnst
end
DisJoint_encoding ← SAT_solver(CNF_0)
if DisJoint_encoding == UNSAT then
    return None        //Cannot be achieved using x bits
end
return DisJoint_encoding
```

TABLE II
ORIGINAL AND *DisJoint* ENCODING SCHEMES FOR THE FSM IN FIG. 4. THE REMAINING STATES ARE HIDDEN FOR SIMPLICITY

State	Original encoding	*DisJoint* encoding
s_0	00	000
s_1	01	011
s_2	10	101
s_3	11	110

DisJoint encoded netlist requires at least 3 FFs, while the original encoding requires only 2 FFs. Thus, it is necessary to understand the minimum number of FFs required by *DisJoint* encoding, as this minimum bound will help designers reduce the overhead without sacrificing security.

Theorem 1. *If an original sequential design contains m states and n FFs, where $2^{n-1} < m \leq 2^n$, then DisJoint encoding can be achieved with $n+1$ FFs.*

Proof. If the circuit has m states and n FFs, where $2^{n-1} < m \leq 2^n$, then there is a non-repeating sequence of n-bit binary values $V = [v_1, v_2, \ldots, v_m]$. Since each element in V is unique (non-repeating), for $\forall i \neq j \in \{1, 2, \ldots, m\}$, the distance between v_i and v_j is no less than 1, *i.e.,* $Dist(v_i, v_j) \geq 1$.

Let us construct another sequence of m binary values with $(n+1)$ bits for each, *i.e.,* $V' = [v'_1, v'_2, \ldots, v'_m]$. For each $v'_i \in V'$ $(i \in \{1, 2, \ldots, m\})$, v'_i satisfies $v'_i = v_i \cdot b_i$ (concatenation operation), and $b_i \in \mathbb{B}$. Notably, b_i satisfies

$$b_i = \begin{cases} 0, & \text{if } v_i \text{ contains even number of 1's} \\ 1, & \text{if } v_i \text{ contains odd number of 1's.} \end{cases}$$

Consequently, each $v'_i \in V'$ contains even number of 1's.

For $\forall i \neq j \in \{1, 2, \ldots, m\}$, if supposing the numbers of 1's in v_i and v_j share the same parity (both are even or both are odd), then $b_i = b_j$. Since $Dist(v_i, v_j)$ must be an even number and $Dist(v_i, v_j) \geq 1$, we can know that $Dist(v_i, v_j) \neq 1$ and $Dist(v_i, v_j) \geq 2$. Furthermore,

$$Dist(v'_i, v'_j) = Dist(v_i, v_j) \geq 2. \tag{1}$$

Otherwise, if the numbers of 1's in v_i and v_j have different parities, then $b_i \neq b_j$. Thus, $Dist(b_i, b_j) = 1$ and $Dist(v_i, v_j) \geq 1$. Since $v'_i = v_i \cdot b_i$,

$$Dist(v'_i, v'_j) = Dist(v_i, v_j) + Dist(b_i, b_j) \geq 2. \tag{2}$$

Therefore, from Eq. (1) and Eq. (2), there exists an encoding scheme with $n+1$ bits, such that the distance between any two states is no less than 2. Hence, *DisJoint* encoding can be achieved with $n+1$ FFs. □

Notably, the proof provides a solution of *DisJoint* encoding, which is a parity encoding. In other words, for an encoding scheme, all states' binary values containing even/odd numbers of 1s (parity encoding) is a solution of *DisJoint* encoding. However, it does not mean that any *DisJoint* encoding must be a parity encoding. Instead, there is a chance that a non-parity encoding can be a *DisJoint* encoding. Algorithm 1 gives a method to find a *DisJoint* encoding, and its result can be a non-parity solution. Since we know that a parity solution is always a solution of *DisJoint* encoding, launching the parity encoding to do *DisJoint* encoding can reduce time complexity, especially when the time effort is critical for the user to protect the design.

Theorem 2. *If the original sequential design contains m states and n FFs, where $2^{n-1} < m \leq 2^n$, then DisJoint encoding cannot be achieved with n FFs.*

Proof. From the proof in Theorem 1, we know that there exists an n-bit encoding scheme for 2^{n-1} states, such that the distance between any two states is no less than 2. Thus, all 2^{n-1} states take half of the entire space $\mathbb{B}^n$. Assume, on the contrary, all 2^{n-1} states do not share the same parity: some states have even numbers of 1's, and other states have odd numbers of 1's. Then, since all states take exactly half of the entire space, there must exist a pair of binary values (v_i, v_j), such that v_i and v_j are adjacent (*i.e.,* $Dist(v_i, v_j) = 1$). Consequently, it contradicts the assumption that the encoding scheme can ensure that each state keeps a distance of at least 2 to all others. Therefore, for an n-bit encoding scheme on 2^{n-1} states satisfying each state has at least distance 2 from all others, all states must share the same parity. Thus, *DisJoint* encoding cannot be achieved with only n FFs. □

Considering both two theorems: (i) Theorem 2 provides a strict lower bound, and the available FFs' amount must be greater than n, and (ii) Theorem 1 ensures the existence of the *DisJoint* encoding scheme when there are no less than $n+1$ FFs. Therefore, the minimum number of extra FFs of *DisJoint* encoding is exactly 1. Moreover, this lower bound helps the designer/defender decide the proper number of FFs to achieve lower overhead while ensuring security.

E. Protecting State-Transition Logic

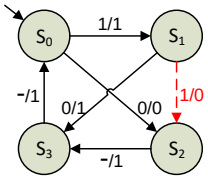
Our objective is to protect the state transition(s) in the original FSM—for example, $s_1 \rightarrow s_2$ in Fig. 4. Protecting the state transition in the FSM is equivalent to locking the state-transition logic in the design. For instance, protecting $s_1 \rightarrow s_2$ in Fig. 5(a) is equivalent to protecting the 3rd row in the state-transition table in Fig. 4(b). To this end, as stated previously, one can use a secure combinational locking technique, such as SFLL with D2PIPs, to protect the design. We consider the state-transition logic in Fig. 4(a) as the target combinational logic for SFLL. A tuple of the input pattern and the current state of the state-transition logic acts as the PIP of SFLL, and it affects the output or the next state. The key-size of the SFLL determines the security level, which the designer can fine-tune to achieve a tradeoff between security vs. overhead. In our case, the key-size is the sum of the number of primary inputs and the number of FFs that drive the state-transition logic.

While SFLL naturally protects against I/O attacks and certain structural attacks [7], it fails to defend against the FALL attack [18] and the SPI attack [19]. To prevent any type of structural attack, one has to ensure that the Hamming distance between the PIP(s) and the FSC of the SFLL should be no less than 2 [19]. Han *et al.* [19] refers to this secure property as *Dist2* property. Not all tuples of the input pattern and the next state can satisfy this property. Hence, we present the *DisJoint* encoding scheme that encodes the states to satisfy this property.

F. STATION's Flow and Time Complexity Analysis

The generation of the protected netlist requires several stages. From the original FSM, it needs to go through the following stages.

Performing *DisJoint* encoding, creating the FSC of the encoded FSM, and generating the gate-level FSC. From the proof of Theorem 1, we can conclude that a parity encoding is a solution of *DisJoint* encoding. Notably, the time complexity to launch *DisJoint* encoding can be reduced to $O(m)$, where m is the number of states in the state-transition table of the original design. Besides, the complexity of updating the encoding (binary value) of each state in the state-transition table is $O(t)$, where t is the number of transitions in the FSM. Once we have such an encoded FSM, the defender can construct the FSM of the FSC by hiding the chosen PSTs from the state-transition table of the encoded FSM. The time complexity of converting the encoded FSM to the FSM of the FSC is $O(t)$. To convert the state-transition table of the FSC from *KISS2* format to behavioral *Verilog* format, the converter tool needs to operate on each transition separately. Thus, the complexity of the conversion is $O(t)$. Once this conversion is finished to get the behavioral *Verilog* design, there is a synthesis step to generate the gate-level *Verilog* using a commercial synthesis tool. The synthesis step takes the majority of the time. Thus, the time complexity of getting the FSC netlist (starting from the original design's state-transition table) is approximately equal to the complexity of a synthesis process.

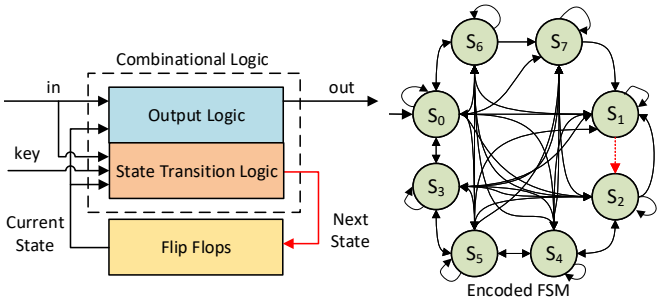


(a)

in	$state_{cur}$	$state_{next}$	out
1	s_0	s_1	1
0	s_0	s_2	0
1	s_1	s_2	0
0	s_1	s_3	1
–	s_2	s_3	1
–	s_3	s_1	1

(b)

Fig. 4. Original FSM and corresponding state-transition table. The defender protects the dashed/shaded transition in red.



(a) (b)

Fig. 5. Proposed sequential obfuscation technique and the corresponding state-encoded FSM.

Appending the restore unit to the FSC and generating the netlist of the *STATION*-obfuscated design. Once we get the netlist of the FSC (gate-level *Verilog* from the last step), we can add the logic of the restore unit to construct the logic of *STATION*-protected design according to Fig. 2. In this stage, most of the time effort is spent in running re-synthesis to get the entire locked netlist.

Therefore, except for two (re-)synthesis steps, the time complexity is $O(m + 3t)$, which is linear in the number of states or transitions in the original FSM. As for the re-synthesis step, many modern commercial tools can efficiently achieve it, such as *Synopsys Design Compiler* [42], *Cadence Genus* [43], and *Mentor Graphics Precision RTL* [44].

IV. Experimental Evaluation

In this section, we first discuss the experimental setup, followed by results on execution time. Following this analysis, we empirically demonstrate the security of our technique and overhead analysis in terms of power, performance, and area (PPA).

A. Experimental Setup

We perform the experiments on a 32-core Intel Xeon processor at 2.6 GHz with 512 GB RAM.

We utilize the *NEOS* tool [47] to launch the BMC attack [25] and the KC2 attack [26]. And we use the *Kiss2vl* tool [48] to convert the *KISS2* format to *Verilog* files. Since our proposed technique performs *DisJoint* encoding on the FSMs, we utilize the benchmarks where the FSMs have been explicitly defined in terms of state-transition table, such as designs in the *KISS2* format. Hence, we use benchmarks from

TABLE III
THE SCALE OF EACH TESTED CIRCUIT, INCLUDING THE NUMBERS OF INPUTS, OUTPUTS, INSTANCES, STATES, TRANSITIONS, AND PSTS

Circuit		# Inputs	# Outputs	# Instances		# States	# Transitions	# PSTs	
				# Gates	# FFs			Original	*DisJoint*
Benchmark	s27	6	1	70	3	7	41	0	41
	bbara	6	2	75	4	10	60	0	60
	beec	5	4	82	3	7	29	0	29
	dk14	5	5	105	3	7	57	0	57
	dk17	4	3	56	3	8	32	0	32
	dk27	3	2	26	3	7	15	0	15
	dk512	3	3	68	4	15	31	0	31
	dnfile	4	1	165	5	24	97	0	97
	origin	3	1	4	1	6	13	0	13
	s298	5	6	1.4×10^3	8	218	1.1×10^3	0	1.1×10^3
	shiftreg	3	1	26	3	8	17	0	17
Synthetic	fsm^1_{syn}	32	7	1.7×10^4	7	80	2.0×10^3	0	2.0×10^3
	fsm^2_{syn}	32	7	2.5×10^4	7	80	3.2×10^3	0	3.2×10^3
	fsm^3_{syn}	52	7	2.4×10^4	7	80	1.9×10^3	0	1.9×10^3
	fsm^4_{syn}	32	7	3.3×10^4	7	80	4.4×10^3	0	4.4×10^3
	fsm^5_{syn}	52	7	3.6×10^4	7	100	3.0×10^3	0	3.0×10^3
	fsm^6_{syn}	62	7	4.1×10^4	7	100	3.1×10^3	0	3.1×10^3
	fsm^7_{syn}	102	7	4.0×10^4	7	80	1.9×10^3	0	1.9×10^3
	fsm^8_{syn}	102	7	6.1×10^4	7	100	3.0×10^3	0	3.0×10^3

TABLE IV
ATTACK RESULTS OF VARIOUS ATTACKS ON DIFFERENT CIRCUITS. A ✓/✗ REPRESENTS THE SUCCESS/FAILURE OF THE ATTACK ON THE CIRCUIT. IN THE 5th, THE 6th, AND THE 7th COLUMNS (SAT, BMC, AND KC2 ATTACKS), NUMERIC VALUES REPRESENT THE REQUIRED NUMBERS OF SAT ITERATIONS AND BMC/KC2 DISCRIMINATING INPUT SEQUENCES (DISES) TO EXTRACT THE CORRECT KEY, A "TIME-OUT" REPRESENTS THAT THE SAT/KC2 ATTACK TOOL CANNOT FIND THE KEY WITHIN 72 HOURS, AND A "OUT-OF-MEMORY" REPRESENTS THE BMC ATTACK CANNOT FIND THE KEY WITH A MEMORY LIMIT OF 10 GB SET IN *NEOS*

Circuit		# States	Key-size	I/O attack				Structural attack				
				SAT [13]	BMC [25]	KC2 [26]	Fun-SAT [27]	SPS [17]	ATR [3]	FALL [18]	SPI [19]	SCC [29]
Benchmark	s27	7	8	98	82	91	✗	✗	✗	✗	✗	✗
	origin	6	5	7	6	6	✗	✗	✗	✗	✗	✗
	bbara	10	9	318	66	94	✗	✗	✗	✗	✗	✗
	dk27	7	5	30	6	8	✗	✗	✗	✗	✗	✗
	beec	7	7	8	36	38	✗	✗	✗	✗	✗	✗
	shiftreg	8	5	2	2	2	✗	✗	✗	✗	✗	✗
	dk512	15	6	27	3	8	✗	✗	✗	✗	✓	✗
	dk17	8	6	10	26	32	✗	✗	✗	✗	✓	✗
	dk14	7	7	126	36	40	✗	✗	✗	✗	✗	✗
	dnfile	24	8	9	43	45	✗	✗	✗	✗	✗	✗
	s298	218	12	2.9×10^3	1.4×10^3	1.4×10^3	✗	✗	✗	✗	✗	✗
Synthetic	fsm^1_{syn}	7	38	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^2_{syn}	7	38	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^3_{syn}	7	58	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^4_{syn}	7	38	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^5_{syn}	7	58	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^6_{syn}	7	68	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^7_{syn}	7	108	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗
	fsm^8_{syn}	7	108	time-out	out-of-memory	time-out	✗	✗	✗	✗	✗	✗

`MCNC-91` suite where the *KISS2* format is available. The chosen set of FSMs in *KISS2* is widely used in academic research works, such as Alkabani *et al.* [10], Koushanfar *et al.* [49], and Li *et al.* [23], [33]. However, the `MCNC-91` suite lacks large-scale designs. Moreover, current FSM extraction tools cannot handle large-scale designs for converting netlists into FSMs.[3] To demonstrate *STATION*'s scalability (performance on large-scale designs), we generated large-scale random FSMs to mimic the real-world FSMs. We use user-defined specifications for inputs, outputs, states, and transitions to generate a random synthetic FSM following Algorithm 2. The process described in lines 6-17 details the generation of each transition ($tran$). To prevent multiple transitions sharing the same trigger condition (where two different transitions start from the same input pattern and current state but lead to different output patterns or next states), we implement a verification step in the algorithm. This step, shown in line 10, checks whether the current transition ($tran$) can potentially conflict with the set storing all previously collected transitions ($\mathcal{G}.Trans$). In case of a conflict (different transitions with identical trigger conditions), the new transition, $tran$, undergoes continuous updates until there is no conflict.

[3]The existing FSM extraction tools cannot extract FSMs for large benchmarks, such as the *NEOS* [47] and *SIS* [50] tools. We have reported and confirmed this issue with the *NEOS* developers. We encountered a similar issue while testing the *SIS* tool [50].

B. Execution Time of STATION

Fig. 6 describes the implementation flow of *STATION* containing multiple steps categorized into two stages: generating

Algorithm 2: Random FSM generation.

Input: # inputs n_input, # outputs n_ouput,
states n_state, # transitions n_tran
Output: Random FSM $\mathcal{G}$

```
G ← init_FSM()
n_ff ← minimum_num_FFs(n_state)
G.States ← state_random_encoding(n_state, n_ff)
G.Trans ← ∅                          //Transitions inititalization
for i ∈ {1, 2, ···, n_tran} do
    while True do
        tran ← init_transition()
        tran.in ← random_input_pattern(n_input)
        tran.state_cur ← random_state(G.States)
        conflict ← exists_tran_start(G.Trans, tran)
        if ¬conflict then
            break //Avoid the same tran trigger condition
        end
    end
    tran.state_next ← random_state(G.States)
    tran.out ← random_output_pattern(n_output)
    G.Trans ← G.Trans ⋃ {tran}
end
return G
```

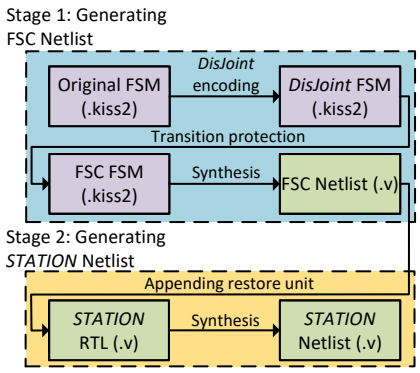


Fig. 6. Two stages of implementing *STATION*. Each box represents the status of the design at the step, along with the file format used.

the netlist of FSC and generating the netlist of *STATION* protected circuit. Given the FSM of the original design (in the *KISS2* file format) as the input, *STATION* first does *DisJoint* encoding on the initial FSM. Once we get the *DisJoint* encoded FSM, *STATION* obfuscates with the chosen PST to produce the FSM of FSC. After obtaining the FSM of FSC, *STATION* utilizes *Design Compiler*, does synthesis on the FSM, and generates the gate-level netlist of the FSC. Thus, in the first stage, we get the netlist of the FSC. Later, to obtain the entire *STATION* protected design, we append the restore unit to get the entire *STATION* protected design according to Fig. 2. We realize this step by adding the RTL description of the restore unit into the Verilog file of the FSC netlist. Lastly, to get the netlist of the *STATION* protected design, we launch another synthesis process utilizing *Design Compiler* and obtain the netlist of the *STATION*-obfuscated design from the netlist of the FSC.

To show the reasonable execution time for *STATION* within a reasonable time limit, the remainder of this subsection

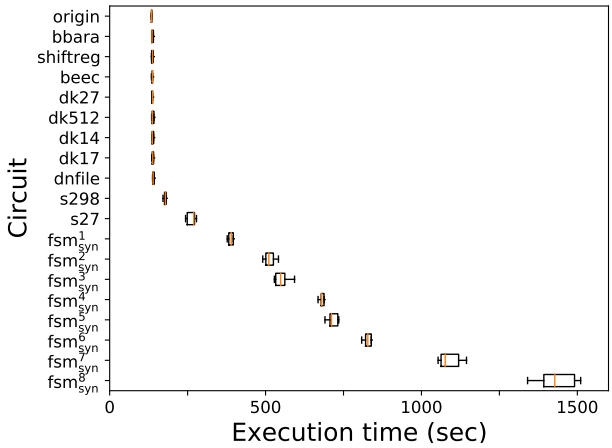


Fig. 7. The distribution of the total execution time after repeating *STATION* on each tested circuit for 10 times.

analyzes the experimental execution time from different perspectives, including the stability and average execution time in different stages of *STATION* by repeating the *STATION* obfuscation process on each design 10 times.

Execution Time. Fig. 7 shows the total execution time after repeatedly performing *STATION* obfuscation on each design 10 times. Each box in Fig. 7 tells the minimum, lower quartile, median, upper quartile, and maximum values of the *STATION* execution time on each circuit. The difference/fluctuation in the execution time is within a narrow range compared to the mean/average execution time. For example, on the synthetic design $\text{fsm}^8_{\text{syn}}$, a relatively large-scale design among all tested circuits, the ratio of standard deviation over mean value is 3.91%. Thus, we consider that the total execution time of *STATION* is stable.

Average Execution Time in Each Stage. According to the theoretical time complexity analysis in Sec. III-F, we know that the generation of each *STATION*-obfuscated circuit can be separated into two stages: the generation of FSC's netlist and the generation of the *STATION*-locked circuit's netlist. In the stage of generating the netlist of the FSC, *STATION* parses the original state-transition table, decides the PST, constructs the FSM of the FSC (in the form of state-transition table), and does synthesis to get the netlist of the FSC based on the state-transition table of the FSC. Once *STATION* generates the netlist of the FSC, *STATION* continually constructs the locked design by XORing the restore unit with the FSC and goes through the re-synthesis process. We can see that, in each stage, there's a synthesis process. In practice, this synthesis process, utilizing the commercial tool *Synopsys Design Compiler*, takes the majority of execution time compared to the effort in other operations, such as parsing the state-transition table, deciding the PST, appending the restore unit, etc. Fig. 8 presents the average execution time in each stage on every tested circuit after repeatedly generating each *STATION*-obfuscated design 10 times.

We can observe from Fig. 8 that, in each tested circuit, the average execution time in two stages (generation of the FSC and generation of the locked design) is almost equal. This is because the input files sent to *Synopsys Design Compiler* (Verilog file before synthesis) have similar sizes. Thus, the

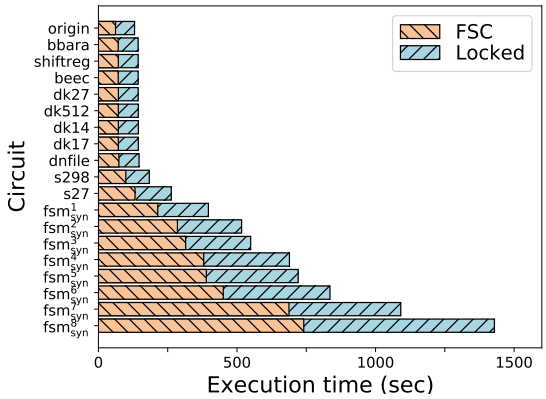


Fig. 8. The average execution time of *STATION* in different stages on each circuit.

complexities in these two stages are almost the same. Therefore, to accelerate the *STATION*'s process, one can append the restore unit without re-synthesis (canceling the re-synthesis step in the second stage). Eliminating the re-synthesis step in the second stage may increase the power, performance, and area (PPA) cost since there's no optimization step to combine the FSC and the restore unit. At the same time, the designer can benefit from the perspective of the execution time since the cancellation can reduce the execution time almost to its half compared to keeping the re-synthesis step in the second stage. Meanwhile, the cancellation of the re-synthesis step in the second stage cannot compromise security even with the exposure of the restore unit. This is because the restore unit stores no secret but purely compares both the input pattern and the key input pattern.

C. Security Analysis of STATION

To show the security of *STATION*, this subsection presents an analysis of the number of PSTs in conventional designs before and after using our encoding scheme, along with the resilience against many I/O and structural attacks.

Number of PSTs. Table III and Table IV provide the number of PSTs and the maximum achievable key-size on the *DisJoint* encoded circuits. Our analysis shows that the original circuits do not have any PSTs. On the contrary, after applying the proposed *DisJoint* encoding, all transitions are PSTs so that a defender can flexibly choose one or multiple transition(s) they want to protect. These results demonstrate the need for the *DisJoint* encoding for sequential obfuscation. In our implementation, we choose to protect a single PST to achieve the highest resilience against I/O attacks, according to the security analysis in Sec. II-A.

Resilience against structural attacks. An attacker can attempt to launch structural attacks like SPS, ATR, FALL, SPI, and SCC [3], [17]–[19], [29] to circumvent the security of the protected state-transition logic. To this end, we run these attacks using the source code developed by the respective researchers. For the SCC attack, we used the DANA tool [29]. SPS, ATR, and FALL attacks target combinational designs and are not applicable to sequential designs. Despite that, we ran the attack on the state-transition logic to demonstrate the strength of *STATION*. As shown in Table IV, none of the structural attacks breaks *STATION*.

Resilience against I/O attacks. Though *STATION* is a sequential obfuscation technique, we evaluate its security against both sequential I/O attacks (*e.g.,* BMC [25] and KC2 [26]) and combinational I/O attacks (*e.g.,* SAT [13]). While the former targets sequential designs, we applied the latter also because the state-transition logic is combinational. We first execute the SAT attack [13] on the protected designs. The 5th column of Table IV shows the SAT attack results on all tested circuits. When the SAT attack tool can find a key within 72 hours, the table lists the required number of iterations by the SAT attack tool, as the SAT attack result on each tested benchmark circuit. If the SAT attack tool cannot find a key within 72 hours, it is considered a "time-out," as the SAT attack results on each synthetic FSM. If we suppose that the required number of SAT iterations satisfies the uniform random distribution in the range of $\{1, 2, 3, \cdots, 2^k\}$ (k is the key-size), then the distribution of $\frac{\text{\# iterations}}{2^{k-1}}$ approximately satisfies the uniform random distribution on the range of $(0, 2]$. The approximation happens since there is a transformation from the discrete space to a continuous space. Further, the expected mean and variance are $\mathbb{E}(\mu) = 1$ and $\mathbb{E}(\sigma^2) = \frac{1}{3}$. Based on the collected number of SAT iterations from the SAT results on all tested benchmark circuits, we can get that the mean and variance of the distribution of $\frac{\text{\# iterations}}{2^{k-1}}$ are $\mu = 0.86$ and $\sigma^2 = 0.49$. We have $\mu \neq \mathbb{E}(\mu)$ and $\sigma^2 \neq \mathbb{E}(\sigma^2)$ for three reasons: (i) there is an approximated mapping for the data points ($\frac{\text{\# iterations}}{2^{k-1}}$ from the discrete space $\{\frac{1}{2^{k-1}}, \frac{2}{2^{k-1}}, \cdots, \frac{2^k}{2^{k-1}}\}$ to the continuous space $(0, 2]$), (ii) it is unstable for the number of SAT iterations when the key-size k is a small value, and (iii) since there are only 11 data points (the numbers of SAT iterations on 11 benchmark circuits), it is more likely to have unstable values of μ and σ^2 due to limited data points. We consider that *STATION* is secure against the SAT attack since, on relatively larger key-size cases, the circuits protected by *STATION* show high resilience from experimental results, including circuits requiring much more SAT iterations on larger benchmark circuits (*e.g.,* s298) and all synthetic FSMs ("time-out" on these synthetic FSMs). We evaluate the resilience against the BMC [25] and KC2 [26] attacks. In Table IV, the "BMC/KC2" column shows the number of discriminating input sequences (DISes) required by the BMC/KC2 attack. Our observations indicate that: (i) for benchmark circuits, the number of DISes grows exponentially with the key-size, and (ii) for synthetic FSMs with large key-sizes, the BMC attack cannot find the key within 72 hours (i.e., "time-out"), and the KC2 attack cannot find the key with a memory limit of 10 GB set in *NEOS* (i.e., "out-of-memory"). Based on these findings, we conclude that *STATION* is secure against the BMC and KC2 attacks. For instance, considering the tested benchmark s27, the *STATION*-obfuscated s27 has a key-size of 8. Theoretically, the expectation value of the number of iterations or DISes is $\mathbb{E}(\textit{\# iterations}) = \mathbb{E}(\textit{\# DISes}) = (\sum_{i=1}^{2^k} i)/2^k \approx 2^{k-1} = 2^{8-1} = 128$. In our experimental analysis (refer to Table IV), we can observe that the actual numbers of SAT iterations, BMC DISes, and KC2 DISes are

TABLE V
ATTACK RESULTS OF VARIOUS ATTACKS ON SMALL-SCALE SYNTHETIC FSMS

Circuit		# States	Key-size	I/O attack				Structural attack				
				SAT [13]	BMC [25]	KC2 [26]	Fun-SAT [27]	SPS [17]	ATR [3]	FALL [18]	SPI [19]	SCC [29]
Synthetic	$\text{fsm}_{\text{syn}}^{\text{s27}}$	7	8	112	15	16	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{origin}}$	6	5	7	10	12	✗	✗	✗	✓	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{bbara}}$	10	9	153	77	58	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{dk27}}$	7	5	22	6	5	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{beec}}$	7	7	50	46	56	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{shiftreg}}$	8	5	17	10	9	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{dk512}}$	15	6	4	22	22	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{dk17}}$	8	6	26	12	11	✗	✗	✗	✓	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{dk14}}$	7	7	122	18	13	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{dnfile}}$	24	8	210	100	98	✗	✗	✗	✗	✗	✗
	$\text{fsm}_{\text{syn}}^{\text{s298}}$	218	12	3.4×10^3	1.6×10^3	1.5×10^3	✗	✗	✗	✗	✗	✗

98, 82, and 91, respectively,[4] which are close to the expected value ($\mathbb{E} = 128$ for s27). Based on these results, we conclude that the *STATION*-obfuscated design is resilient against the SAT, BMC, and KC2 attacks.

Finally, we tested *STATION* against the Fun-SAT attack [27]. Fun-SAT targets to break the circuit protected by HARPOON [6] or Interlocking [11]. However, some assumptions made by Fun-SAT to break other protected designs are not applicable to *STATION*-obfuscated designs. A *STATION*-obfuscated circuit does not contain an obfuscation mode or require a corresponding input key sequence of length t_k. Thus, *STATION* requires adjustment to work with Fun-SAT. Since the *STATION*-obfuscated circuit does not have an obfuscation mode, we consider $t_k = 0$ for all tested circuits. Finally, a circuit protected by HARPOON [6] or Interlocking [11] does not have explicit key ports like *STATION* but instead has a key sequence that is fed from input ports. Thus, we consider the key ports of the *STATION*-obfuscated circuit as input ports while running Fun-SAT. Fun-SAT cannot prune out keys effectively on *STATION*-obfuscated designs, as functional corruptibility does not always increase monotonically. This is due to a combination of factors, such as the low output corruptibility in the *STATION*-obfuscated design and the lack of distinct functional and obfuscation modes in *STATION*. In most cases, Fun-SAT terminates early after encountering a sequence of iterations where the functional corruptibility does not increase monotonically. The early termination even occurs when user-defined parameters are set to avoid an early termination. Thus, Fun-SAT is unable to break *STATION*.

To summarize, we observe that our proposed sequential obfuscation technique can defend against the state-of-the-art I/O and structural attacks mentioned in Table IV.

Does the Design Suite Matter for the Security of *STATION*? To show that *STATION* works on both small-scale and large-scale designs, we chose MCNC benchmarks and large synthetic FSMs. The attack results in Table IV highlight that the SAT attack breaks small-scale MCNC benchmarks while returning "time-out" on large-scale synthetic FSMs. To show that *STATION* provides similar resilience on same-scale designs irrespective of the suites where designs belong, we first create additional small-scale synthetic FSMs with the exact sizes of all 11 tested benchmarks in the MCNC suite (the same numbers of inputs, outputs, states, and transitions). For example, in the MCNC suite, bbara has 6 inputs, 2 outputs, 10 states, and 60 transitions; when generating the corresponding synthetic FSM, this FSM has the same number of inputs, outputs, states, and transitions as bbara; we name this synthetic FSM $\text{fsm}_{\text{syn}}^{\text{bbara}}$. Next, we protect small-scale synthetic FSMs with *STATION* and conduct all attacks on these small-scale synthetic FSMs. Table V depicts the attack results on small-scale synthetic FSMs. The SAT, BMC, and KC2 attacks break all small-scale synthetic FSMs by returning correct keys, and this case is the same as the SAT, BMC, and KC2 attack results on MCNC benchmarks. The FALL attack breaks two designs ($\text{fsm}_{\text{syn}}^{\text{origin}}$ and $\text{fsm}_{\text{syn}}^{\text{dk17}}$) with small key-sizes (5 and 6), which is reasonable since the FALL attack returns multiple candidate keys. Thus, we see similar attack performance on MCNC benchmarks and small-scale synthetic FSMs. Thus, we conclude that *STATION* provides a similar level of resilience to designs irrespective of the suite of designs. Therefore, if there is a large-scale industrial FSM in practice to protect with, *STATION* can provide resilience against SAT, BMC, KC2, and FALL attacks.

D. Overhead of STATION

Table VI showcases the PPA overheads between the original design and the *STATION*-obfuscated design. On benchmark circuits with smaller sizes, the average overheads on power, performance, and area are 122.19%, 53.05%, and 113.12%. The overheads are high on these benchmark designs because these benchmarks are inherently very small (the maximum number of gates is only $1,425$ on s298). At the same time, on the larger synthetic FSMs, the average PPA overheads are 7.00%, 0.15%, and 8.69%. This is in par with existing sequential obfuscation techniques, *e.g.,* the PPA overheads of HARPOON-obfuscated designs are 13.31%, -1.73%, and 8.92% [6]. Additionally, these controllers are $\leq 1\%$ in modern processors [10], further reducing the overhead in large-scale designs.

To show how the number of PSTs can affect the overheads, we conducted an experiment on the largest tested FSM, $\text{fsm}_{\text{syn}}^{8}$, by changing the number of PSTs from 1 to 10, as shown

[4]Using the *NEOS* tool suite, we run both BMC and KC2 attacks and collect different numbers of DISes for BMC and KC2 attacks. The difference in the numbers of DISes is because the *NEOS* tool generates random seeds at the beginning of the BMC/KC2 attack; consequently, after re-running the attack, the number of DISes changes.

TABLE VI
THE PPA OVERHEADS RESULTS BETWEEN THE ORIGINAL DESIGN AND THE *STATION*-OBFUSCATED DESIGN

Circuit		Overhead (%)		
		Power	**Performance**	**Area**
Benchmark	s27	220.16	18.75	61.39
	bbara	69.24	431.82	85.31
	beec	76.27	1.6	69.86
	dk14	83.17	5.79	87.47
	dk17	72.23	6.6	101.86
	dk27	101.55	−73.04	117.97
	dk512	106.97	37.04	97.05
	dnfile	79.69	0.0	114.21
	origin	203.98	69.23	227.59
	s298	234.36	77.03	145.48
	shiftreg	96.46	8.77	136.15
	Average	122.19	53.05	113.12
Synthetic	fsm^1_{syn}	10.86	−1.84	11.64
	fsm^2_{syn}	10.44	0.87	11.0
	fsm^3_{syn}	10.37	−5.04	9.27
	fsm^4_{syn}	9.31	2.55	9.68
	fsm^5_{syn}	8.07	1.67	9.05
	fsm^6_{syn}	7.36	0.41	6.88
	fsm^7_{syn}	7.10	1.33	6.41
	fsm^8_{syn}	8.75	1.22	5.63
	Average	7.00	0.15	8.69

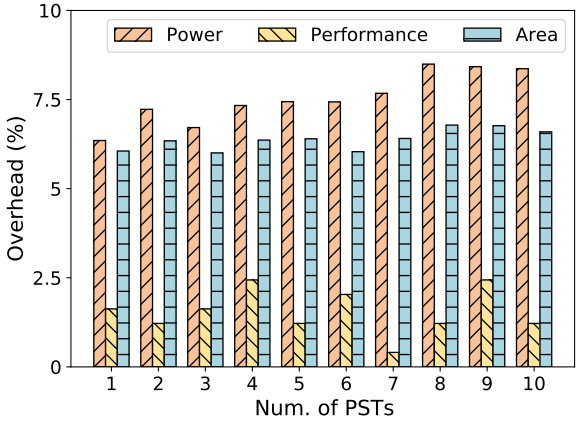


Fig. 9. Trade-off between the number of PSTs and power, performance, and area (PPA) overheads on fsm^8_{syn}.

in Fig. 9. We observe that, on the largest tested FSM, the overhead of *STATION* implementation is within a reasonable range. When increasing the number of PSTs, there is a small increase in power. Comparing the power overheads when protecting the design with 1 PST and 10 PSTs, the difference is less than 0.54%. The performance (delay) overhead varies, maintaining within a negligible range (less than 2.5%). On fsm^8_{syn}, there is no significant variation in the area overhead, and the standard deviation is within 0.27%.

E. Vulnerabilities of Previous Obfuscations Techniques

Table VII shows the resilience of SCRAMBLE, HST, or JANUS-obfuscated designs against SAT and BMC attacks. As shown in the table, SCRAMBLE is vulnerable to BMC attack. In the case of HST, except for s298 with the key-size of 217, the BMC attack can break the tested benchmark circuit with a few DISes. The BMC attack cannot break s298 for HST and JANUS as finding each DIS costs a long execution time due to the large-scale unrolled design (the number of unrollings is 217 on s298). In the case of JANUS and HST, the BMC attack can find the key except for s298 but can break all the circuits, thereby rendering them non-secure. We attribute this outlier to the nature of s298 rather than security of HST and JANUS as evidenced in other circuits. However, *STATION* is secure against these attacks, as listed in Table IV.

V. DISCUSSION AND LIMITATIONS

Is *STATION* vulnerable to scan-based attacks? If the attacker has scan-chain access, they can scan out the next-state values, eventually extracting the logic of a transition in the obfuscated FSM. However, due to *DisJoint* encoding and *STATION*, there are exponential numbers of states and transitions introduced into the FSM. Furthermore, one can also use *STATION* in conjunction with techniques that lock scan chains [51], thereby preventing an attacker from scanning out the values.

Is *Disjoint* encoding the same as parity encoding? *DisJoint* encoding is not the same as parity encoding. If there are 2^n states, then each solution of *DisJoint* encoding with $n+1$ bits is a parity encoding. However, if there are more than $n+2$ bits, it is not parity encoding, but it is still a *DisJoint* encoding. To the best of our knowledge, *DisJoint* is the first encoding scheme for sequential obfuscation.

Is *STATION* dependent on SFLL? *STATION* is independent of SFLL. *STATION* relies on a secure combinational locking technique to protect the state-transition logic. As long as this locking technique is secure against all attacks, *STATION* remains secure. So far, SFLL is secure against all these attacks (when PIPs are chosen to ensure structural security, such as D2PIPs [19]). Hence, we used SFLL. However, one can use other logic locking techniques for *STATION*.

VI. CONCLUSION

Various sequential obfuscation techniques have been put forward in the research community; however, many of these have been circumvented by attackers. Also, multiple defense techniques are proposed to defend against a specific set of attacks. However, *STATION* can simultaneously defend against multiple I/O and structural attacks by leveraging *DisJoint* encoding and existing secure combinational logic locking techniques. Our experimental analysis over different circuits demonstrates that *STATION* achieves the desired security against nine attacks, including four I/O attacks and five structural attacks while incurring tolerable power and area overheads with a minimal performance penalty. One can achieve a holistic obfuscation solution by using *STATION* for FSMs/controllers and combinational logic locking techniques for datapath units.

ACKNOWLEDGEMENT

We thank Prof. Kaveh Shamsi from University of Texas at Dallas for helping provide the *NEOS* tool suite during reviewing process [47]. We thank the members of the TAMU SETH lab for their help in improving the paper. Moreover, we

TABLE VII
THE SAT ATTACK RESULTS ON SCRAMBLE-OBFUSCATED DESIGNS, AND THE BMC ATTACK RESULTS ON HST-OBFUSCATED DESIGNS AND JANUS-OBFUSCATED DESIGNS. "N/A" DENOTES THAT SCRAMBLE IS NOT ABLE TO PROTECT THE DESIGN ORIGIN; "TIME-OUT" DENOTES THAT THE BMC ATTACK DOES NOT FIND THE CORRECT KEY WITHIN 72 HOURS

Benchmark	# States	SCRAMBLE		HST		JANUS	
		Key-size	# Iterations	Key-size	# DISes	Key-size	# DISes
s27	7	3	1	6	5	18	11
bbara	10	8	4	9	15	36	23
beec	7	3	2	6	4	18	15
dk14	7	3	1	6	2	18	7
dk17	8	3	2	7	1	21	9
dk27	7	3	2	6	1	18	8
dk512	15	8	2	14	1	56	13
dnfile	24	8	4	23	1	115	528
origin	6	N/A	N/A	5	3	5	4
s298	218	224	5	217	time-out	1,736	time-out
shiftreg	8	3	2	7	2	21	12

thank the reviewers for providing valuable comments during the reviewing process. The work was supported in part by the National Science Foundation (NSF CNS–1822848) and the DARPA grants (HR0011-20-9-0043 and M2102069) from the Automatic Implementation of Secure Silicon (AISS) program [8] and the Structured Array Hardware for Automatically Realized Applications (SAHARA) program [9]. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government.

REFERENCES

[1] S. Khan, A. Mann, and D. Peterson, "The Semiconductor Supply Chain: Assessing National Competitiveness," *Center for Security and Emerging Technology*, 2021.

[2] J. Baukus, L. Chow, R. Cocchi, and B. Wang, "Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing," *US Patent no. 20120139582*, 2012.

[3] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably Secure Camouflaging Strategy for IC Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 8, pp. 1399–1412, 2017.

[4] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," *USENIX Security Symposium*, pp. 495–510, 2013.

[5] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," *IEEE/ACM Design, Automation and Test in Europe (DATE)*, pp. 1069–1074, 2008.

[6] R. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 10, pp. 1493–1502, 2009.

[7] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory To Practice," *ACM Conference on Computer and Communications Security (CCS)*, pp. 1601–1618, 2017.

[8] Defense Advanced Research Projects Agency (DARPA), "Automatic Implementation of Secure Silicon," https://www.darpa.mil/news-events/2020-05-27, 2020, Last accessed on 04/02/2024.

[9] Intel, "Structured Array Hardware for Automatically Realized Applications," https://blogs.intel.com/psg/intel-and-darpa-announce-sahara-program-to-develop-secure-reliable-domestic-source-of-structured-asics/, 2021, Last accessed on 04/02/2024.

[10] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," *USENIX Security Symposium*, pp. 291–306, 2007.

[11] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking Obfuscation for Anti-Tamper Hardware," *ACM Cyber Security and Information Intelligence Research (CSIIRW)*, pp. 1–4, 2013.

[12] J. Dofe, Y. Zhang, and Q. Yu, "DSD: a Dynamic State-Deflection Method for Gate-Level Netlist Obfuscation," *IEEE Computer Society Annual Symposium on VLSI (ISLVLSI)*, pp. 565–570, 2016.

[13] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," *IEEE Hardware Oriented Security and Trust (HOST)*, pp. 137–143, 2015.

[14] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 2, pp. 199–207, 2019.

[15] 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," *IEEE/ACM Design Automation Conference (DAC)*, pp. 1–6, 2019.

[16] ——, "InterLock: An Intercorrelated Logic and Routing Locking," *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[17] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 342–347, 2017.

[18] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 15, pp. 2514–2527, 2020.

[19] Z. Han, M. Yasin, and J. Rajendran, "Does logic locking work with EDA tools?" *USENIX Security Symposium*, pp. 1055–1072, 2021.

[20] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[21] M. Yasin, J. Rajendran, and O. Sinanoglu, "A Brief History of Logic Locking," in *Trustworthy Hardware Design: Combinational Logic Locking Techniques*. Springer, 2020, pp. 17–31.

[22] Z. Han, M. Yasin, and J. Rajendran, "Multi-Objective Strategies for Stripped-Functionality Logic Locking," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[23] L. Li and A. Orailoglu, "JANUS-HD: Exploiting FSM Sequentiality and Synthesis Flexibility in Logic Obfuscation to Thwart SAT Attack while Offering Strong Corruption," *IEEE Design, Automation & Test in Europe (DATE)*, pp. 1323–1328, 2022.

[24] Y. Zhang, Y. Hu, P. Nuzzo, and P. A. Beerel, "TriLock: IC Protection with Tunable Corruptibility and Resilience to SAT and Removal Attacks," *IEEE Design, Automation & Test in Europe (DATE)*, pp. 1329–1334, 2022.

[25] M. El Massad, S. Garg, and M. Tripunitara, "Reverse Engineering Camouflaged Sequential Circuits Without Scan Access," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 33–40, 2017.

[26] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation," *IEEE/ACM Design, Automation & Test in Europe (DATE)*, pp. 534–539, 2019.

[27] 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," *IEEE Hardware Oriented Security and Trust (HOST)*, 2021.

[28] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the Difficulty of FSM-based Hardware Obfuscation,"

IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), pp. 293–330, 2018.

[29] N. Albartus, M. Hoffmann, S. Temme, L. Azriel, and C. Paar, "DANA Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, pp. 309–336, 2020.

[30] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Muilding Logic Encryption," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 153–159, 2020.

[31] K. Juretus and I. Savidis, "Synthesis of Hidden State Transitions for Sequential Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 1, pp. 11–23, 2020.

[32] P. Shrestha and I. Savidis, "Synthesis of Coupling Capacitance Based Hidden State Transitions for Sequential Logic Locking," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1734–1738, 2022.

[33] L. Li, S. Ni, and A. Orailoglu, "JANUS: Boosting Logic Obfuscation Scope Through Reconfigurable FSM Synthesis," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 292–303, 2021.

[34] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, and M. Zuzak, "Keynote: A Disquisition on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 1952–1972, 2020.

[35] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Transactions on Computers (TC)*, vol. 64, no. 2, pp. 410–424, 2015.

[36] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[37] N. Karousos, K. Pexaras, I. G. Karybali, and E. Kalligeros, "Weighted Logic Locking: A New Approach for IC Piracy Protection," *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 221–226, 2017.

[38] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the Approximation Resiliency of Logic Locking and IC Camouflaging Schemes," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 14, no. 2, pp. 347–359, 2018.

[39] K. Shamsi and Y. Jin, "In Praise of Exact-Functional-Secrecy in Circuit Locking," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 16, pp. 5225–5238, 2021.

[40] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly Stripping Functionality for Logic Locking: A Fault-based Perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020.

[41] B. Tan, R. Karri, N. Limaye, A. Sengupta, O. Sinanoglu, M. M. Rahman, S. Bhunia, D. Duvalsaint, R. D. Blanton, A. Rezaei, Y. Shen, H. Zhou, L. Li, A. Orailoglu, Z. Han, A. Benedetti, L. Brignone, M. Yasin, J. Rajendran *et al.*, "Benchmarking at the Frontier of Hardware Security: Lessons from Logic Locking," *arXiv preprint arXiv:2006.06806*, 2020.

[42] Synopsys, "Design Compiler NXT," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, Last accessed on 04/02/2024.

[43] Cadence, "Genus Synthesis Solution," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, Last accessed on 04/02/2024.

[44] Mentor Graphics, "Precision RTL," https://eda.sw.siemens.com/en-US/ic/precision/rtl/, Last accessed on 04/02/2024.

[45] G. DeMicheli, P. Antognetti, and A. Sangiovanni-Vincentelli, *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*. Springer Science & Business Media, 1987, vol. 136.

[46] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing (TETC)*, vol. 8, no. 2, pp. 517–532, 2017.

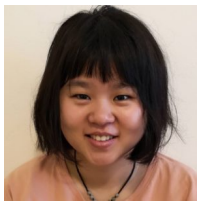
[47] K. Shamsi and Y. Jin, "NEOS," https://bitbucket.org/kavehshm/neos/src/master/, Last accessed on 04/02/2024.

[48] C. Pruteanu, "Kiss to Verilog FSM Converter," http://codrin.freeshell.org, Last accessed on 09/18/2021.

[49] F. Koushanfar and Y. Alkabani, "Provably Secure Obfuscation of Diverse Watermarks for Sequential Circuits," *IEEE Hardware Oriented Security and Trust (HOST)*, pp. 42–47, 2010.

[50] M. Iervasi and T. Villa, "SIS: UNIVR Logic Synthesis Software," https://jackhack96.github.io/logic-synthesis/sis.html, Last accessed on 04/02/2024.

[51] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting All Logic Locking Attacks: Dishonest Oracle With Truly Random Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 9, pp. 1740–1753, 2020.



Zhaokun Han received her Bachelor of Science degree in Physics from the University of Science and Technology of China in 2018. She joined the Secure and Trustworthy Hardware lab as a doctoral student in Fall 2018. Now, her interests in research include logic locking, hardware security, logic synthesis, and the security of embedded FPGAs.

Aneesh Dixit received his Bachelor of Technology in Electronics and Communication Engineering from the National Institute of Technology, Karnataka, India, in 2020. He has been a part of the SETH lab at Texas A&M since Fall 2021. His interests include logic locking, hardware security, and CAD for security.

Satwik Patnaik is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Delaware. Previously, he was a postdoctoral researcher at Texas A&M University. He obtained his Ph.D. degree in Electrical Engineering from the Tandon School of Engineering, New York University, in September 2020.

His research encompasses strengthening the security of the semiconductor supply chain and developing security-focused computer-aided design tools to realize secure and trustworthy hardware and systems. In addition, his research leverages the 3D paradigm for computer security, exploits security properties of emerging post-CMOS devices, and applies machine learning and reinforcement learning techniques for enhancing and evaluating attacks and defenses for different hardware security problems.

Jeyavijayan (JV) Rajendran is an Assistant Professor in the Department of Electrical and Computer Engineering at the Texas A&M University. Previously, he was an Assistant Professor at UT Dallas between 2015 and 2017. He obtained his Ph.D. degree from New York University in August 2015. His research interests include hardware security and computer security. His research has won the NSF CAREER Award in 2017, the ACM SIGDA Outstanding Young Faculty Award in 2019, the ACM SIGDA Outstanding Ph.D. Dissertation Award in 2017, and the Alexander Hessel Award for the Best Ph.D. Dissertation in the Electrical and Computer Engineering Department at NYU in 2016, along with several best student paper awards.