SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Building Logic Encryption

Hadi Mardani Kamali*, Kimia Zamiri Azar*, Houman Homayoun†, Avesta Sasan*
*Department of ECE, George Mason University, E-mail: {hmardani, kzamiria, asasan}@gmu.edu
†Department of ECE, University of California, Davis, E-mail: hhomayoun@ucdavis.edu

***Abstract*—In this paper, we introduce SCRAMBLE, as a novel logic locking solution for sequential circuits while the access to the scan chain is restricted. The SCRAMBLE could be used to lock an FSM by hiding its state transition graph (STG) among a large number of key-controlled false transitions. Also, it could be used to lock sequential circuits (sequential datapath) by hiding the timing paths' connectivity among a large number of key-controlled false connections. Besides, the structure of SCRAMBLE allows us to engage this scheme as a new scan chain locking solution by hiding the correct scan chain sequence among a large number of the key-controlled false sequences. We demonstrate that the proposed scheme resists against both (1) the 2-stage attacks on FSM, and (2) SAT attacks integrated with unrolling as well as bounded-model-checking. We have discussed two variants of SCRAMBLE: (I) Connectivity SCRAMBLE (SCRAMBLE-C), and (b) Logic SCRAMBLE (SCRAMBLE-L). The SCRAMBLE-C relies on the SAT-hard and key-controlled modules that are built using near non-blocking logarithmic switching networks. The SCRAMBLE-L uses input multiplexing techniques to hide a part of the FSM in a memory. In the result section, we describe the effectiveness of each variant against state-of-the-art attacks.**

***Keywords*-Reverse Engineering, Logic Locking, Sequential Locking, FSM Locking, Scan Chain Locking**

I. Introduction

Due to the ever-increasing cost of IC manufacturing, many design houses have been forced to become fabless. Outsourcing the manufacturing stages, including fabrication/testing/packaging, to potentially untrusted entities raises multiple forms of security threats such as IC overproduction, Trojan insertion, reverse engineering (RE), intellectual property (IP) theft, and counterfeiting [1], [2].

Among many active and passive *design-for-trust* (DfT) techniques, logic locking [3], [4], as a proactive solution, hides the functionality of the netlist by inserting additional programmable gates (key gates), whose programming values (key values) are stored in tamper-proof memories. However, the strength of logic locking solutions was seriously challenged in recent years by the Boolean satisfiability (SAT) attack [5], [6].

Although the SAT attack (and many of its derivatives) only works on combinational circuits [7], having access to the scan chain allows an adversary to apply the SAT attack on each combinational logic part of sequential circuits separately. Accordingly, the adversary can target sequential circuits using the SAT attack. Hence, few recent studies investigated the possibility of restricting the scan chain using scan chain locking/blocking [9]–[11]. Also, considering that the access to the scan chain is restricted/locked, several studies investigated the possibility of applying the logic locking to the whole sequential circuits [14], [15], particularly FSMs [14]–[19]. However, further research revealed that new attacks could be formulated for these locking solutions even while access to the scan chain is restricted/locked.

In case of FSM locking [16]–[18], [21], a new attack, without oracle access, denoted as ***2-stage attacks on FSM*** (2-stage) was formulated [14], [19]. Also, in case of sequential (datapath) or scan chain locking [9]–[11], [14], [15], a new breed of SAT-based attacks, referred as ***unrolling-based SAT attack*** (UB-SAT) as well as SAT attacks integrated with bounded-model-checking (BMC) was formulated [22]–[24], challenging the validity of these solutions.

To defend against UB-SAT or BMC, and to break 2-stage attacks on FSMs, we introduce a new state, connectivity and routing augmentation model for building logic encryption (SCRAMBLE). SCRAMBLE is designed to add and maximize (a) the *false transitions* within STG when FSM is targeted for locking, (b) the *false connections* between datapath flip flops (FFs) when sequential datapath locking is targeted, (c) the *false sequences'* possibilities in scan FFs (SFFs) when the scan chain is targeted. SCRAMBLE, with 2 variants, can resist both 2-stage attacks on FSM as well as UB-SAT or BMC attacks on sequential datapath or scan chain locking.

II. Background and Related Work

A. FSM, Sequential Datapath, and Scan Chain Locking

In FSM locking [16]–[18], [21], few extra sets (modes) of states are added to the original state transition graph (STG), such as locking/authentication mode states or black holes. The traversal sequence of locking/authentication modes is the locking/authentication key, and a correct traversal that reaches the initial state of the original STG unlocks the FSM. Also, the output generated by the correct traversal of authentication states serves as a watermark. In addition to these groups, a set of studies focuses on locking the STG without adding any extra state. However, the complexity and overhead (area) of this approach is higher compared to other schemes [14].

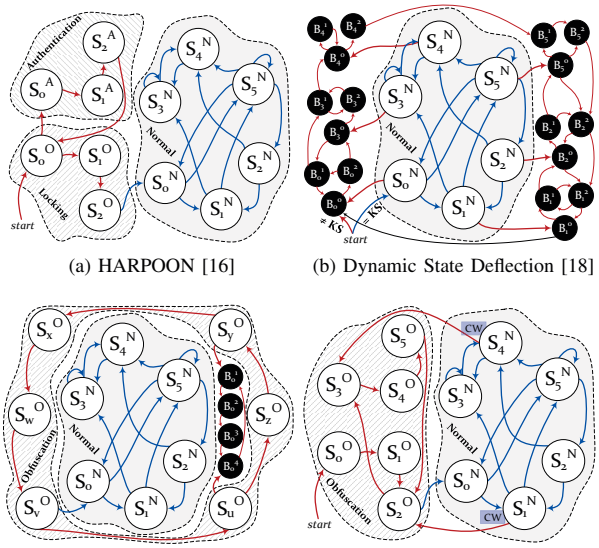
2159-3477/20/$31.00 ©2020 IEEE
DOI 10.1109/ISVLSI49217.2020.00037

In sequential datapath locking or scan chain locking, XOR- or MUX-based locking has been added into timing paths or the scan chain. For instance, in scan chain locking solutions, the scan chain has been blocked or locked to prevent the scan chain loading and readout (shift/load) for protecting the logic against the SAT attacks. For example, the *Encrypt Flip-Flop* solution [9] adds key-programmable MUXes on the outputs of SFFs enabling the negation of the SFFs when the scan chain locking key is incorrect.

B. Attacks on FSM, Sequential, and Scan Chain Locking

To assess the strength of FSM locking solutions, many studies evaluated the possibility of deploying 2-stage attacks, as an *oracle-less* attack, on locked FSMs [14], [19]. The 2-stage attacks on FSM are composed of: (1) ***(stage 1): topological analysis*** (described in line 2-13 of Algorithm 1), which is a detection algorithm to find FFs that are responsible for storing the state values (separating them from datapath FFs), and ***(stage 2): functional analysis*** (described in line 14-21 of Algorithm 1) that finds the STG based on the list of FFs found in (stage 1). In such an attack, the topological analysis, which is derived from [25], identifies FFs whose input contains a combinational feedback path from their output. Then, it reduces the set of possible state FFs by (a) grouping the FFs controlled by the same set of signals, and (b) finding strongly connected components (SCC) using Tarjan's algorithm [14], [19], [26], [27]. In the functional analysis stage (stage 2), the attacker attempts to extract/re-draw the STG. This is done by first attempting to find the initial state, and then identifying the reachable states by creating a reduced binary decision diagram (BDD) or using a SAT solver. After re-drawing STG by using a 2-stage attack, in most FSM locking solutions, the adversary can readily distinguish the original part of the FSM from either extra added states or extra state transitions, leading to extracting the original FSM. Fig. 1 illustrates two examples of FSM locking. As shown, the original part of the FSM is easily distinguishable from extra locked states in these solutions.

In UB-SAT or BMC [22]–[24] on the other hand, as a much stronger attack that is applicable to all FSM locking, sequential datapath locking, and scan chain locking, the adversary does not need to have access to the scan chain. In these attacks, the adversary unrolls the reverse-engineered netlist n times and then creates a double circuit similar to the SAT attack. Then, the adversary uses a SAT solver to find a sequence of n inputs and two key values such that the output of the meter (double) circuit detects a mismatch. Such an input sequence is denoted as a *discriminating input sequence* (DIS). The attacker increases the unrolling depth (n) until a termination condition is met. The overall flow of this breed of attacks is captured in Algorithm 2. By using this structure, the adversary can target and attack any sequential logic locking solution even while the access to the scan chain



(a) HARPOON [16] (b) Dynamic State Deflection [18]

(c) Hardware Active Metering [17] (d) Interlocking Obfuscation [21]

Figure 1. FSM Obfuscation Solutions.

is restricted. Also, as an enhanced version of this group of attacks, KC2 [22], accelerates the original UB-SAT [23] by using some additional simplification steps. Some of the added features include (a) integrating BMC with SAT, (b) model conversion to BDD to simplify the circuit representation, and (c) constraint simplification using key-sweeping. Similarly, the ScanSAT [24] is another unrolling-based SAT attack that only focuses on scan chain locking solutions, which is applicable to both statically and dynamically scan chain locking solutions.

Algorithm 1 2-stage on FSM Locking [14], [19]

```
1:  function FSM_EXTRACT(Circuit C_L)
2:      SFF ← [];                                          ▷ State Flip Flops
3:      RS ← classify(FFs);                  ▷ Classifying FFs into Register Sets
4:      for each set ∈ RS do
5:          set ← set - notSCC(set);          ▷ Keeps Strongly Connected
    Components
6:          if is_splittable(set) then
7:              RS ← {RS - set} ∪ split(set);
8:      CLFP ← find_feedback_circuits(C_L, Reg_Sets);
9:      for each set ∈ RS do
10:         set ← set - notInfDep(set);                ▷ Keeps Intersected
    Influence/Dependence
11:         set ← set - InputIndependt(set);       ▷ Check Control Metrics
12:         update(CLFP);
13:         SFF ← SFF ∪ set;
14:     S_0 ← initial_state(state_regs); SQ ← [];              ▷ State Queue
15:     SQ ← SQ ∪ S_0; STF ← [];                ▷ State Transition Table
16:     while SQ ≠ [] do
17:         state ← SQ.dequeue();
18:         for each DIP do                              ▷ DIP found by SAT
19:             if eval(state_regs, DIP, state) ∉ SQ then
20:                 SQ.enqueue(nx_state);
21:                 STF ← STF ∪ {state, DIP, nx_state, PO}
        return SQ, S_0, STF;                ▷ States, Initial, Transition Func.
```

Algorithm 2 UB-SAT on Sequential/Scan Locking [22]–[24]

```
1:  function UB_SAT(Circuit C)
2:      b ← initial_boundary;
3:      Terminated ← False;
4:      MC_circuit ← C(X, K1, Y1) ∧ C(X, K2, Y2) ∧ (Y1 ≠ Y2);
5:      while !Terminated do
6:          while (X_DIS, K1, K2) ← BMC(MC_circuit, b) = T do
7:              Y_f ← C_BlackBox(X_DI);
8:              MC_circuit =∧ C(X_DIS, K1, Y_f) ∧ C(X_DIS, K2, Y_f);
9:          if not BMC(MC_circuit, b) then                    ▷ UC
10:             Terminated = True;
11:         else if not BMC(MC_comb_circuit, b) then          ▷ CE
12:             Terminated = True;
13:         else if UMC(MC_circuit, b) then                   ▷ UMC
14:             Terminated = True;
15:         else
16:             b ← b + boundary_step;
17:     KeyGenCircuit = DIVC ∧ (K1 = K2)
18:     Key ← BMC(emphKeyGenCircuit, b)
```

III. Proposed Scheme: SCRAMBLE

In SCRAMBLE, we engage the term *augmentation* to refer to the process illustrated in Fig. 2. *Augmentation* in SCRAMBLE adds false state transitions in case of FSM locking, or adds false FF-to-FF timing paths in case of sequential datapath locking, or adds false scan chain sequence in case of scan chain locking. SCRAMBLE is proposed in two variants: (1) The first variant is *connectivity* SCRAMBLE (SCRAMBLE-C) that hides the connectivity to the targeted FFs using logarithmic switching network. (2) The second variant is *logic* SCRAMBLE (SCRAMBLE-L) that hides the logic by implementing part(s) of the logic within memory. The SCRAMBLE-C could be used for locking either FSMs, sequential datapath, or scan chains, to protect the locked design against all UB-SAT and BMC attacks, such as KC2 or ScanSAT. SCRAMBLE-L, on the other hand, is mostly applicable to FSMs to provide resilience against 2-stage attacks [14], [19].

A. SCRAMBLE-C

The overall structure of SCRAMBLE-C has been illustrated in Fig. 3. In SCRAMBLE-C, the connectivity between the targeted FFs and their *fan-in-cone*s (FiCs) (combinational logic cones) is locked. Hence, before connecting the output of corresponding FiCs to the FFs, a *configurable routing and logic block* (CRLB) has been inserted to control the connections. For instance, in Fig. 3, a CRLB with size 8 has been inserted before a combination of FSM FFs and datapath FFs. The CRLB, which must be inserted between the targeted FFs and their corresponding FiCs, is a key-programmable switching network that is built using self-routing logarithmic ($log_2(N)$) networks [28], [29]. The $log_2(N)$ networks, compared to the existing and well-known switching networks, such as mesh, crossbar, etc., incur less area overhead. Also, we demonstrate that due to its cascading structure, $log_2(N)$ networks could improve the robustness against the SAT attack.

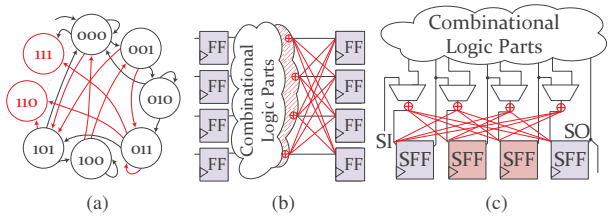


Figure 2. *Augmentation* for (a) FSM Locking, (b) Sequential Datapath Locking, and (c) Scan Chain Locking. (Black Lines: Original. Red lines: False by SCRAMBLE.)

As shown in Fig. 3, the CRLB is built using key-programmable 2x2 switch-boxes (sw_{ij}). Based on its key, a sw saves or changes the order of inputs while connecting them to output pins. Also, the connection between the layers of sws is fixed. This inter-layer connection determines the topology of the $log_2(N)$ network. For instance, the architecture of a sample CRLB (*shuffle* topology) with size 8 has been demonstrated in Fig. 3. Also, as shown in this example, to add the capability of logic programmability, we add one extra key-controlled (XOR) inversion layer, as the final layer of CRLB, to twist routing locking with logic locking. The addition of the inversion layer allows the CRLB to not only permute the inputs, but it also negates them based on the keys of the inversion layer.

In SCRAMBLE-C, the CRLB must be inserted before the targeted FFs. When FSM locking or sequential datapath locking is targeted, during either the physical design or after DFT synthesis step, the CRLB is placed on wires that connect the outputs of FiCs to the *data-in (DI) pin* of targeted FSM FFs or datapath FFs. When scan chain locking is targeted, after DFT synthesis, the CRLB is placed in *scan network* before the *scan-in (SI) pin* of the targeted SFFs.

Although engaging self-routing $log_2(N)$ networks provides a low-overhead routing locking solution, we have to address a few issues: (1) The size of the $log_2(N)$ circuit grows exponentially as the input size grows. (2) The nature of $log_2(N)$ networks is blocking, and many of the input permutations cannot be routed. Hence, the number of false transitions/connections/sequences in FSM/datapath/scan would be very limited. Hence, the wires (N) as the inputs of CRLB must be small enough to make the network overhead reasonable; and large enough to make it resistant against SAT-driven attacks, i.e. UB-SAT or BMC. It raises two questions: (1) which N FFs must be selected? and (2) How we can minimize N?

*1) **Selection of** N **FFs**:* The selection of FFs (to insert CRLB before them) in SCRAMBLE-C could significantly impact its locking strength, particularly in FSM locking. For example, let us consider the engaging of SCRAMBLE-C for an FSM presented in Fig. 4(b), which is generated using Binary encoding of 4 FFs. In this example, if we select *two* least significant bits (LSB) FFs to insert a CRLB with size 2 before them (Circuit of Fig. 4(a)), the locked FSM is demonstrated in Fig. 4(c). Fig. 4(a) shows how the false

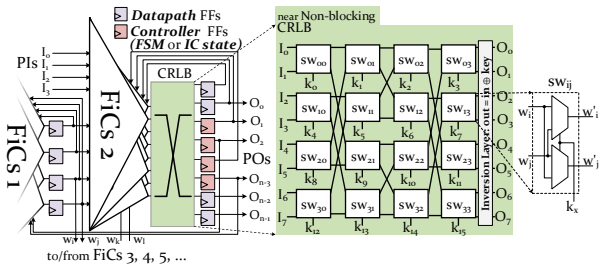


Figure 3. *Augmentation* via Inserting CRLB (*shuffle*-based) in SCRAMBLE-C.

transitions in Fig. 4(c) have been generated for some of the original transitions. As shown in 4(c), when 2 LSBs are selected, a limited number of false transitions are added, and only one extra (unreachable) state is visited. However, if we select *two* most significant bits (MSB) FFs, it will visit all extra states and generates a large number of false transitions demonstrated in Fig. 4(d). Since 2-stage attacks are applicable to FSM locking, maximizing false transitions as well as extra states makes SCRAMBLE-C more robust against this attack. Accordingly, being aware of the encoding style of FSM will impact its locking strength. For instance, in Binary encoding, a synthesis tool usually encodes the states from low to high (0 to 2^{N-1}). Hence, using the N FFs representing the MSB of state value results in the inclusion of the largest number of previously unreachable states and false transitions in the locked FSM. Also, Fig. 4(e) shows locked FSM when *three* LSB FFs are selected. It shows that even increasing the size of CRLB by one adds numerous false state transitions into the locked FSM. Note that unlike FSM locking, the selection of N FFs has no impact on locking strength when sequential datapath locking or scan chain locking is targeted.

*2) **Reducing N by making CRLBs near non-blocking**:* The implementation of blocking $log_2(N)$ network revealed that even a 256-input CRLB could be broken by SAT attack in less than an hour. Hence, to address the blocking nature of CRLB and to resist against UB-SAT or BMC attacks (with a small CRLB), we expand the $log_2(N)$ network towards non-blocking via adding extra (cascaded) stages. The expanded $log_2(N)$ network with strictly non-blocking structure is generalized under the notation $LOG_2(N, M, P)$, where N denotes the number of inputs, M is the number of extra (cascaded) stages, and P indicates that there are $P-1$ additional copies (of whole network) *vertically cascaded* [28]. However, For a network with an arbitrary N, the minimum value of M and P to make the network strictly non-blocking are extremely large. For instance, with $N = 64$ the choice of M and P should be 3 and 6 respectively, resulting in 5x area overhead compared to a blocking $log_2(N)$ [28]. Hence, strictly non-blocking incurs almost prohibited area overhead.

To move close enough towards non-blocking nature without incurring large area overhead, we used the *"near non-blocking"* structure [28]. In near non-blocking, not all but *almost all* permutations are feasible, while it could be implemented using a $LOG_2(N, log_2(N) - 2, 1)$, meaning it has only $M = log_2(N) - 2$ extra stages and no additional copy ($P = 1$). The CRLB depicted in Fig. 3 is an example of a near non-blocking CRLB for 8 inputs. Our implementation shows that a 32-input near non-blocking network ($LOG_2(32, 3, 1)$) is far stronger against SAT attack compared to a 256-input blocking network $log_2(256)$, while it is 8x smaller.

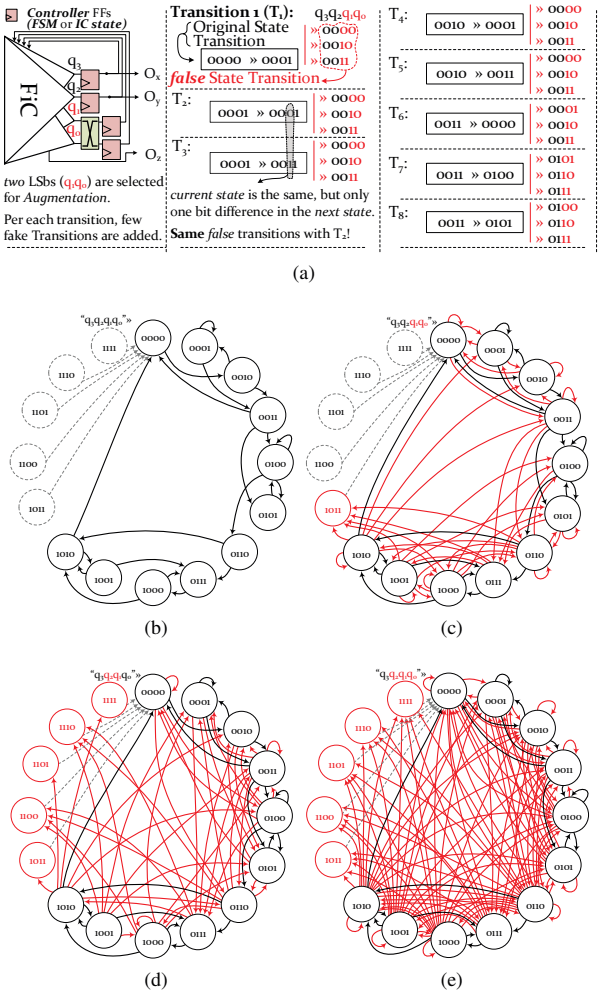


Figure 4. Using SCRAMBLE-C for FSM Locking (a) FSM circuit and transitions generation, (b) the original FSM, (c) 2 LSBs are inputs to SCRAMBLE-C, (d) 2 MSBs are inputs to SCRAMBLE-C, (e) 3 LSBs are inputs to SCRAMBLE-C.

B. SCRAMBLE-L

In SCRAMBLE-L, which is proposed for FSM locking against 2-stage attacks, the logic before the targeted FFs is locked using in-memory computation. As shown in Fig. 5, a small part of the combinational logic in the FiCs of the targeted FFs is replaced with a one-cycle read memory, such as SRAM. As an example, FiC_2 and FiC_4 are replaced

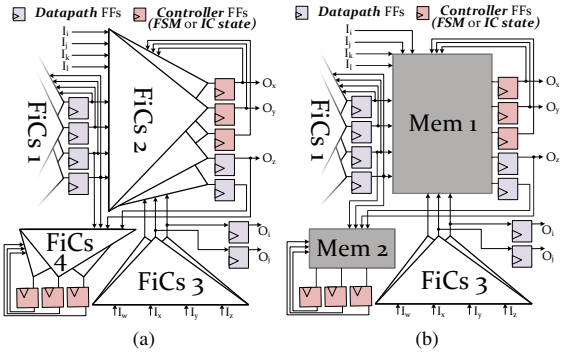


Figure 5. Sequential Circuits Locking using SCRAMBLE-L.

with equivalent memories. The content of the memories must provide the same output compared to that of FiC_2 and FiC_4 while the triggering input is the same. Hence, the truth table corresponded to those FiCs must be generated and stored in the memories. The memories would be initialized during boot-up from a tamper-proof NVM which serves as the key storage. To avoid additional delay incurred by memories, the selection of FFs must be done with respect to their available timing slack.

SCRAMBLE-L makes 2-stage attacks almost impractical. Considering that the adversary has no access to the contents of memories after reverse engineering, there is no equivalent logic for the memories, and the BDD- or SAT-based *functional analysis* (stage 2) cannot be accomplished on the locked circuit. Also, similar to Fig. 5(b), if the designer selects a combination of datapath FFs and FSM FFs, the adversary cannot distinguish between them when deploying *topological analysis* (stage 1) of the 2-stage attack, resulting in the inclusion of an extremely large number of non-FSM FFs in the candidate FSM FFs. Hence, none of the existing 2-stage attacks can be applied to SCRAMBLE-L.

The big challenge with the SCRAMBLE-L is the size of the memory for implementing the selected FiCs. However, since SCRAMBLE-L is proposed for FSM locking, this problem could be easily addressed by engaging the FSM input multiplexing (FSMIM) techniques [30]. In this technique, considering that the next state and the outputs of the FSM are a function of a subset of the inputs (not all), a set of multiplexers has been used to select only those signals that affect the next state and the output. Hence, the designer is able to minimize the number of inputs to the memories (as address width), resulting in a significant decrease in the size of memory. The main difference between traditional FSM implementation, memory-based FSM, and FSMIM has shown in Fig. 6.

In FSMIM, multiplexers could be controlled using two different strategies: (1) using the value of the current state, (2) using code-words stored in the memory. The first option is more efficient in terms of memory size reduction; however, the second method has better efficiency in reducing

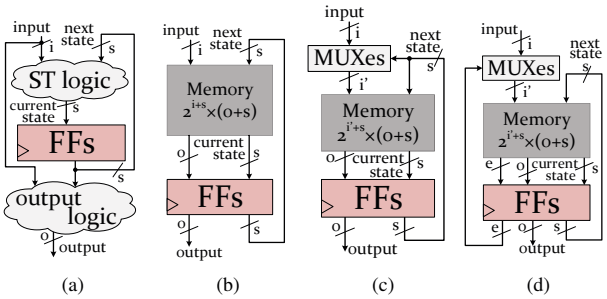


Figure 6. FSM Implementation (a) Traditional, (b) Memory-based, (c) Memory-based with Input Multiplexing using Current State, (d) Memory-based with Input Multiplexing using Code-word stored in Memory.

Table I
SIMPLIFICATION RATIO OF INPUT MULTIPLEXING (FSMIM).

FSM	Required Memory Size and Additional MUXes						
	Traditional	Input MUXing			Input MUXing + State Reduce		
	$Size_{Kb}$	$Size_{Kb}$	MUX	Reduction	$Size_{Kb}$	MUX	Reduction
s510	435,500	5.5	14, 5	∼99.9%	2.5	14, 7	∼99.9%
s820	195,000	255	5, 4, 3, 2	∼99.9%	38	7, 6, 4, 4, 2, 2. 2	∼99.9%
s832	200,000	262.5	5, 4, 3, 2	∼99.9%	69	5, 4, 4, 4, 2	∼99.9%
s1488	408,000	110,500	2, 2	73%	16,000	4, 4, 2, 2, 2	92.5%
s1494	408,000	110,500	2, 2	73%	16,000	4, 4, 2, 2, 2	92.5%

the multiplexers complexity. Hence, the first option has been used in SCRAMBLE-L to minimize the area overhead of the memories. Our evaluation in Table I illustrates the effectiveness of FSMIM when applied to the ISCAS-89 benchmarks, resulting in memory size reduction above 90%.

IV. DISCUSSION

Table II shows the effectiveness of each variant of SCRAMBLE against 2-stage and UB-SAT or BMC attacks. Although the main aim of SCRAMBLE-C is to protect the design against UB-SAT and BMC, it also breaks 2-stage attacks. Similar to SCRAMBLE-L, if we use a combination of both datapath FFs and FSM FFs as input to SCRAMBLE-C (Similar to Fig. 3), topological analysis (stage 1) of 2-stage attack cannot detect the correct set of FSM FFs. Therefore, the functional analysis (stage 2) has to generate the STG using an incorrect set of FFs (extremely larger set), resulting in a significant increase in the attack time with respect to the number of additional (datapath) FFs included in the set. Also, the extracted STG is constructed using a combination of datapath FFs and FSM FFs, which leads to an incorrect STG, and the adversary is not able to extract the original FSM from the extracted STG.

Although SCRAMBLE-L protects the design against 2-stage attacks by hiding the logic within memory, the adversary can generate the equivalent logic of the memory (X input (address width) and Y outputs (word size)) by replacing it with Y of X-input LUTs, which is a fully configurable logic, and then using SAT attack. However,

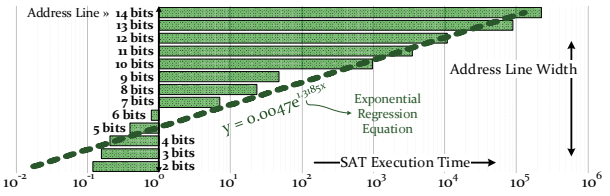


Figure 7. The Impact of Address Width of the Memory on the Execution Time of its LUT-based Equivalent Model.

increasing the input size of the LUTs exponentially increases the run time of BDD-based or SAT-based attacks. Fig. 7 shows that by increasing the address width (from 2 bits to 14 bits), when we replace the memory with the same size LUTs, SAT execution time increases exponentially [31]. In addition, due to the unrolling structure of UB-SAT or BMC, these LUTs must be replicated per each iteration (per each unrolling), which makes them almost unresolvable by SAT-driven attacks. We demonstrate that UB-SAT or BMC cannot reveal the correct functionality of a design even while SCRAMBLE-L has been used with only one 256 words (address width is 8) memory.

V. Experimental Results

We evaluated the strength of SCRAMBLE on two sets of benchmark circuits: (1) sequential ISCAS-89 benchmark circuits and (2) few well-known small-scale ASICs to large-scale microprocessors. We have deployed a 2-stage attack according to Algorithm 1 to assess the strength of SCRAMBLE in FSM locking. For sequential datapath locking, we deployed an integrated BMC with SAT [23] accelerated using stages described in KC2 [22]. Finally, to assess the effectiveness of scan chain locking, we implemented the ScanSAT as described in [24]. All attacks are carried on a Dell PowerEdgeR620 with Intel Xeon E5-2670 2.50GHz and 64GB of RAM.

Table III captures the execution time of scanSAT [24] (for scan locking) and accelerated BMC [22], [23] (for sequential datapath locking) while SCRAMBLE-C is used on ISCAS-89 benchmarks. The maximum runtime of attack is set to 10^6 seconds, and attack will time-out (✗ in tables) if attack time exceeds the limit. Note that in some cases, the number of required FFs is limited. For instance, in *s1196*, with 18 FFs, the maximum possible size of CRLB is only 16. As illustrated, by utilizing the CRLB with size 16, for almost all benchmark circuits, both attacks cannot retrieve the keys. Also, Table III reports the power, performance, and area (PPA) overhead of SCRAMBLE-C with a CRLB of size 16. While the CRLB size is fixed, the area overhead is constant and the percentage area overhead reduces when the size of the benchmark circuits increases. As shown, for even mid-size ISCAS-89 benchmark circuits, the area overhead is less than 10%.

Table II
The Effectiveness of Variants of SCRAMBLE in case of FSM/Sequential/Scan Obfuscation.

Variants	SCRAMBLE-C		SCRAMBLE-L	
Attacks	2-stage	UB-SAT *or* BMC	2-stage	UB-SAT *or* BMC
FSM	✓	✓	✓	✓
Sequential Datapath	N/A	✓	N/A	✓*
Scan-chain	N/A	✓	N/A	✓*

∗: Requires large augmentation model incurring area overhead.

To assess the robustness of SCRAMBLE for FSM locking, both SCRAMBLE-C and SCRAMBLE-L have been used on the second group of circuits. Also, the locked circuits have been evaluated using both BMC and 2-stage attacks. As illustrated in Table IV, BMC can break SCRAMBLE-C while the CRLB size is up to 16. However, for none of the circuits, BMC cannot retrieve the correct key while the CRLB size is 32. Also, in case of BMC, only utilizing a memory with 256 words (address width = 8) is enough to make the locked circuit resilient against BMC.

Unlike BMC, which can break SCRAMBLE for small-size CRLBs and memories, 2-stage attacks are far weaker. As shown in Fig. 4, since the number of false paths is extremely larger, after re-drawing the FSM using 2-stage, there is no chance for the adversary to extract the original part of the FSMs. Hence, as shown in Table IV, 2-stage attacks completely fail against SCRAMBLE.

Since the minimum size of CRLB in SCRAMBLE-C (memory in SCRAMBLE-L), which provides a resilient FSM locking against BMC, is 32 (256 words), we reported the PPA overhead of these sizes for second groups of the circuits in Table V. As shown, even for mid-size *32b RSA* circuit, the overhead is less than 5%. Also, the impact of increasing the size of CRLB or memory on the PPA overhead for different sizes has been illustrated in Table VI. As shown, in both SCRAMBLE-C and SCRAMBLE-L, increasing either the size or address width, approximately doubles the overhead. However, compared to ISCAS-89 benchmark circuits, such as *s15850* or *s38584*, the incurred overhead is reasonable.

VI. Conclusion

In this paper, we introduce SCRAMBLE, as a comprehensive obfuscation solution for protecting FSMs, se-

Table III
The attack time for breaking SCRAMBLE-C used for scan chain locking and sequential datapath locking of ISCAS-89 benchmarks.

				Attack Time (second)						Datapath Locking		
				scanSAT			BMC			PPA overhead of		
				CRLB Size			CRLB Size			16-input CRLB		
Circuit	#FF	#Gate	In/Out	8	16	32	8	16	32	Power	Delay	Area
s1196	18	529	14/14	2029	✗	**N/A**	1109	✗	**N/A**	26.3%	36.5%	24.1%
s1423	74	657	17/5	3441	✗	✗	438.6	9356	✗	25.8%	28.1%	23%
s5378	179	2779	35/49	6406	✗	✗	6921	✗	✗	8.9%	18.5%	7.1%
s9234	211	5597	36/39	1801	✗	✗	1548	✗	✗	5.1%	14.8%	3.9%
s15850	534	9772	77/150	5882	✗	✗	7097	✗	✗	3.1%	12.9%	2.4%
s35932	1728	16065	35/320	8604	✗	✗	7110	✗	✗	1.1%	6.5%	0.9%
s38584	1426	19253	38/304	4072	✗	✗	6287	✗	✗	1.2%	5.7%	0.9%

✗ : *Timeout* = 10^6 *Seconds*

Table IV
THE BMC AND 2-STAGE ATTACK TIME FOR BREAKING SCRAMBLE-C AND SCRAMBLE-L USED FOR FSM LOCKING.

			Attack Time (second)											
			BMC						2-stage					
			SCRAMBLE-C CRLB Size			SCRAMBLE-L Mem Addr			SCRAMBLE-C CRLB Size			SCRAMBLE-L Mem Addr		
Circuit	#FF	#Gate	8	16	32	7	8	9	8	16	32	7	8	9
RS232	168	59	2.7	2029	✗	35.7	✗	✗	✗	✗	✗	✗	✗	✗
32b RSA	555	2139	1.4	3441	✗	45.8	✗	✗	✗	✗	✗	✗	✗	✗
MC8051	578	6590	47.7	6406	✗	50.1	✗	✗	✗	✗	✗	✗	✗	✗
SPARC	120K	233K	938	✗	✗	288.2	✗	✗	✗	✗	✗	✗	✗	✗

✗ *: Timeout* = 10^6 *Seconds*

Table V
THE PPA OVERHEAD FOR BUILDING A LOCKED FSM RESISTANT TO BMC ATTACK USING SCRAMBLE-C AND SCRAMBLE-L

	SCRAMBLE-C (Resilient with CRLB Size = 32)				**SCRAMBLE-L** (Resilient with SRAM Size = $2^8 \times 8$)			
Circuit	RS232	32b RSA	MC8051	SPARC	RS232	32b RSA	MC8051	SPARC
Area (%)	38.5%	4.5%	1.2%	0.05%	173%	17.8%	5.1%	0.1%
Power (%)	44.8%	5.6%	1.7%	0.1%	224%	26.8%	7.2%	0.3%
Delay (%)	48.4%	10.8%	11.4%	9.7%	22.7%	5.5%	6.8%	3.9%

quential circuits, and scan chains against IP piracy and reverse engineering. The proposed solution, SCRAMBLE, resist against both (1) the 2-stage attacks on FSM, and (2) unrolling-based SAT attacks while sequential or scan obfuscation is targeted. We have discussed two variants of SCRAMBLE: (a) SCRAMBLE-C, and (b) SCRAMBLE-L. The SCRAMBLE-C relies on the SAT-hard and key-controlled modules that are constructed using the near non-blocking logarithmic switching network. The SCRAMBLE-L uses input multiplexing techniques to hide a part of the FSM in a memory. In our result section, we illustrated that attack time could be made unreasonably long using any of these techniques.

VII. ACKNOWLEDGEMENT

This research is supported by the Defense Advanced Research Projects Agency (DARPA-AFRL, #FA8650-18-1-7819) and National Science Foundation(NSF, #1718434).

REFERENCES

[1] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proc. of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
[2] M. M. Tehranipoor *et al.*, "Invasion of the Hardware Snatchers," *IEEE Spectrum*, vol. 54, no. 5, pp. 36–41, 2017.
[3] J. A. Roy *et al.*, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[4] J. Rajendran *et al.*, "Security Analysis of Logic Obfuscation," in *Design Automation conference (DAC)*, 2012, pp. 83–89.
[5] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.
[6] M. El Massad *et al.*, "IC Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *Network and Distributed System Security Symposium (NDSS)*, 2015, pp. 1–14.
[7] K. Z. Azar *et al.*, "Threats on Logic Locking: A Decade Later," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 471–476.
[8] K. Z. Azar *et al.*, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Trans. on Cryptographic Hardware and Embedded Systems (TCHES)*, pp. 97–122, 2019.

Table VI
THE PPA OVERHEAD OF SCRAMBLE-C AND SCRAMBLE-L WHEN USING CRLB/SRAM OF DIFFERENT SIZES.

	SCRAMBLE-C (CRLB Input Size)				**SCRAMBLE-L** (SRAM Size)			Sample ISCAS-89 Benchmarks	
Overhead	8	16	32	64	$(2^7 \times 8)$	$(2^8 \times 8)$	$(2^9 \times 8)$	s15850	s38584
Area (um^2)	58.1	136.7	330.8	620.4	305.8	612.1	1119	6262	21458
Power (uW)	4.5	6.9	14.5	31.9	31.4	80.6	118.9	325.7	1031
Delay (ns)	0.33	0.37	0.48	0.56	0.17	0.18	0.19	1.24	2.68

[9] R. Karmakar *et al.*, "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," *arXiv preprint arXiv:1801.04961*, 2018.
[10] U. Guin *et al.*, "Robust Design-for-Security Architecture for Enabling Trust in IC Manufacturing and Test," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 26, no. 5, pp. 818–830, 2018.
[11] X. Wang *et al.*, "Secure Scan and Test using Obfuscation throughout Supply Chain," *IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1867–1880, 2018.
[12] S. Roshanisefat *et al.*, "DFSSD: Deep Faults and Shallow State Duality, A Provably Strong Obfuscation Solution for Circuits with Restricted Access to Scan Chain," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.
[13] H. M. Kamali *et al.*, "On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, pp. 1–6.
[14] T. Meade *et al.*, "Revisit sequential logic obfuscation: Attacks and defenses," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.
[15] L. Li *et al.*, "Structural Transformation for Best-possible Obfuscation of Sequential Circuits," in *IEEE Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2013, pp. 55–60.
[16] R. Chakraborty *et al.*, "HARPOON: An Obfuscation-based SoC Design Methodology for Hardware Protection," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.
[17] F. Koushanfar, "Active Hardware Metering by Finite State Machine Obfuscation," in *Hardware Protection through Obfuscation*, 2017, pp. 161–187.
[18] J. Dofe and Q. Yu, "Novel Dynamic State-Deflection Method for Gate-Level Design Obfuscation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 273–285, 2018.
[19] M. Fyrbiak *et al.*, "On the Difficulty of FSM-based Hardware Obfuscation," *IACR Trans. on Crypto Hardware and Embedded Systems (TCHES)*, pp. 293–330, 2018.
[20] K. Z. Azar *et al.*, "COMA: Communication and Obfuscation Management Architecture," in *Int'l Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019, pp. 181–195.
[21] A. R. Desai *et al.*, "Interlocking Obfuscation for Anti-Tamper Hardware," in *Proc. of the Cyber Security and Information Research Workshop*, 2013, pp. 1–8.
[22] K. Shamsi *et al.*, "KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation," pp. 1–6, 2019.
[23] M. El Massad *et al.*, "Reverse Engineering Camouflaged Sequential Circuits without Scan Access," in *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2017, pp. 33–40.
[24] L. Alrahis *et al.*, "ScanSAT: unlocking obfuscated scan chains," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 352–357.
[25] Y. Shi *et al.*, "A Highly Efficient Method for Extracting FSMs from Flattened Gate-level Netlist," in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, 2010, pp. 2610–2613.
[26] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.
[27] T. Meade *et al.*, "Netlist Reverse Engineering for High-Level Functionality Reconstruction," in *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2016, pp. 655–660.
[28] D.-J. Shyy *et al.*, "Log/sub 2/(N, m, p) Strictly Nonblocking Networks," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1502–1510, 1991.
[29] H. M. Kamali *et al.*, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Design Automation Conference (DAC)*, 2019, pp. 1–6.
[30] I. G. Vargas *et al.*, "ROM-based Finite State Machine Implementation in Low Cost FPGAs," in *IEEE Int'l Symp. on Industrial Electronics*, 2007, pp. 2342–2347.
[31] H. M. Kamali *et al.*, "LUT-lock: A Novel LUT-based Logic Obfuscation for FPGA-bitstream and ASIC-Hardware Protection," in *IEEE Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 405–410.