Mystic: Mystifying IP Cores Using an Always-ON FSM Obfuscation Method

Ehsan Aerabi, Ahmad Patooghy, Hamidreza Rezaei, Miguel Mark, Mahdi Fazeli, and Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering, Boston University

***Abstract*—The separation of manufacturing and design processes in the integrated circuit industry to tackle the ever increasing circuit complexity and time to market issues has brought with it some major security challenges. Chief among them is IP piracy by untrusted parties. Hardware obfuscation which locks the functionality and modifies the structure of an IP core to protect it from malicious modifications or piracy has been proposed as a solution. In this paper, we develop an efficient hardware obfuscation method, called Mystic (Mystifying IP Cores), to protect IP cores from reverse engineering, IP overproduction, and IP piracy. The key idea behind Mystic is to add additional state transitions to the original/functional FSM (Finite State Machine) that are taken only when incorrect keys are applied to the circuit. Using the proposed Mystic obfuscation approach, the underlying functionality of the IP core is locked and normal FSM transitions are only available to authorized chip users. The synthesis results of ITC99 circuit benchmarks for ASIC 45nm technology reveal that the Mystic protection method imposes on average 5.14% area overhead, 5.21% delay overhead, and 8.06% power consumption overheads while it exponentially lowers the probability that an unauthorized user will gain access to or derive the chip functionality.**

I. Introduction

Growing complexity and critical time-to-market have played key roles in the current semiconductor design and manufacturing supply chain landscape. For example, many fab-less companies have emerged to take advantage of low-cost overseas foundries for IC production. Companies, such as ARM Holdings, develop and sell their soft intellectual properties (IP) to other chip manufacturers for the hard implementations. There is a large international market for these pre-built, verified and ready-to-use soft IP designs. Under this globalization trend, *IP piracy* has become an increasing concern which has drawn a great deal of research and investment from both academia and industry [1]. For instance, an untrusted party can reverse-engineer and steal an IP core and then claim ownership, resell or over-produce it [2].

Logic Masking is a set of IP piracy prevention methods which obfuscate the circuit's main functionality to prevent unauthorized access to the chip's functionality. The circuit cannot properly operate until the owner activates it by means of an activation key. Logic masking is generally achieved by inserting *Key Gates e.g XOR/XNOR/MUX/AND/OR* into the original combinational circuit, each of which is driven by a bit of the activation key. These key gates mask the circuit's functionality in a way that only a unique correct key can neutralize their effects. After chip fabrication, the secret key is programmed usually in a secure internal EEPROM memory and the masked IP is unlocked by the owner.

In order to maximize the mismatch points between the masked circuit and the original circuit when comparing them by formal methods, Chakraborty *et. al* [3] proposed a method based on fan-in and fan-out cones in the circuit. Rajendran *et. al* [4] presented another logic masking method in which key gates have more effect on each other. This hinders attacker's effort to reveal the key by feeding the circuit with specific inputs and propagating uncorrelated key bits to the outputs. Using the concept of *Fault Propagation*, [5] tries to perform a proper key gate insertion. The aim is to have nearly 50% of the wrong output bits when a wrong key-vector is applied. By choosing appropriate nets to insert *XOR/XNOR* gates, one can achieve 50% correctness in the Hamming Distance (*HD*) among outputs for the valid key and invalid key. Using the *MUX* primitive as key-gates instead of *XOR/XNOR* has been proposed in [6] and [5]. When supplied key-inputs are correct, the MUXs pass the correct input, otherwise they pass a wrong value coming from other parts of the circuit. Overall, the goal in [5] is to achieve a high *HD* between the correct and wrong keys.

Sequential circuit masking methods try to obfuscate the finite state machine (FSM) of the circuit at the system level perspective. Under this set of approaches, the original state transitions are modified in a way that only a unique sequence of keys can drive it through its correct transitions. Otherwise, the system is lost in out-of-order or fake states.

Chakraborty *et. al* proposed an FSM-based method called HARPOON [7], which adds a finite state machine to the IP core netlist. The FSM outputs are connected to the internal nodes of the circuit via some *XOR* gates. Therefore, the circuit cannot properly work until the output of the added FSM becomes logical zero. Zhang *et. al* presented an IP protection and FPGA licensing scheme which combined FSM masking with PUFs (Physical Unclonable Function) [8]. Recently, Sumathi *et. al* [9] published an FSM-based IP protection to improve the HARPOON approach.

The contribution of this paper is two fold.

- *A new hardware attack specification:* although substantial research has been done on sequential logic masking, we show that most of the previous sequential logic masking techniques are vulnerable to *FSM Separation Attack*.
- *A new obfuscation method:* to address this weakness, we propose an FSM-based logic masking technique at the RTL level. The proposed method can effectively protect against *FSM Separation Attacks* as well as the recently presented SAT attacks [10], [11].

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00119

This paper is organized as follows. In Section II, we provide a brief overview of the previously proposed obfuscation methods. Section III introduces our proposed attack method. In Section IV, we explain our proposed FSM encoding method and show how it overcomes the mentioned weakness using an illustrative case. Section V contains an explanation of the experimental system setup and results. Finally, Section VI concludes the paper.

II. Backgrounds

Logic masking protection mechanisms can be divided into "sequential logic" and "combinational logic" protections. For sequential logic circuits, the protection method is applied to the state transition graph of the circuit by adding extra states with the aim of masking or authenticating [9], [3], [12], [13]. Almost all of the previously proposed sequential encodings are based on the concept illustrated in Figure 1. As shown in the figure, a set of obfuscated states is added to the FSM of the original design. The circuit starts from an initial state in the obfuscated states set. In this initial state, the circuit is locked and its produced outputs are intentionally wrong. To successfully traverse the obfuscated states and enter the original states of the circuit, one must apply to the input(s) the correct sequence of i_0 to i_n for $(n+1)$ consecutive clock cycles. Exiting the obfuscated states will lead to the first state within the normal FSM, T_0. Therefore, the circuit will now correctly respond to inputs since outputs here are functions of inputs and the original FSM states. If k is the number of primary inputs which are applied to the obfuscated states, then an attacker needs to potentially perform $2^k \times 2^n$ searches to unlock the circuit.

For the combinational logic circuits, some extra gates (XOR/XNOR or multiplexer) are inserted into the original combinational circuit. Each obfuscating gate has an input that is derived from the secret key, so that the correct combination of the key bits would neutralize the masking effect of these gates. Consequently, an incorrect input key will lead to incorrect circuit functionality. XOR/XNOR gates could be inserted randomly in the circuit as expressed in [14], but there is no guarantee of the circuit malfunctioning if the wrong keys are used as input. Some researchers [3] have tried to improve the robustness of these obfuscation methods by combining combinational and sequential techniques. To achieve this goal, the outputs of obfuscated states e.g., S_0 to S_{n-1} are connected to *Modification Cells* which are extra logic inserted into the circuit's combinational part (See Figure 2). While the circuit is in the obfuscated states, the outputs of S_0 to S_{n-1} enable the Modification Cells and disables normal circuit operation. The Modification Cell combines the original net of the circuit (p) with a high fan-in signal borrowed from another part of the circuit to add more obfuscation.

The output of the Modification Cell is often expressed as an $output = p \cdot \bar{f} + \bar{p} \cdot g \cdot f$ where f is the obfuscation enabling signal, $f = 1$ when the circuit is in the obfuscated states and otherwise $f = 0$. p is the original net and g is the high fan-in net. Since f is a function of S_0 to S_{n-1}, it evaluates to zero

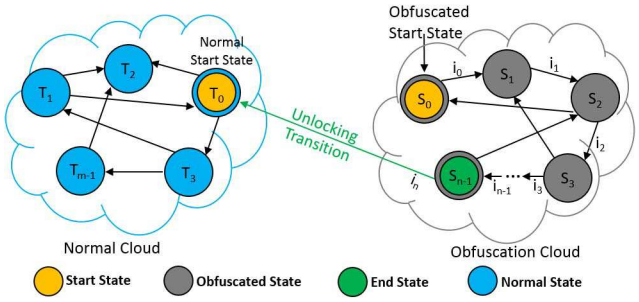


Fig. 1. General block diagram of FSM encoding methods.

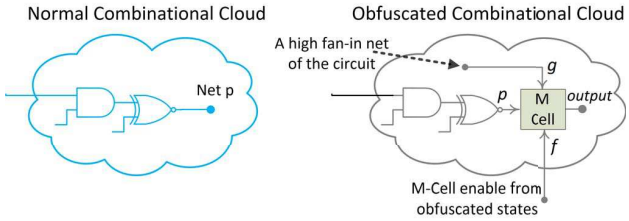


Fig. 2. Combinational logic obfuscation using modification cells.

whenever the obfuscated FSM goes through the *Unlocking Transitions* and reaches the normal FSM.

III. FSM Separation Attack

In this section we describe the *FSM Separation Attack* which can exponentially reduce the search space for attackers to unlock an obfuscated circuit. This builds on the work presented in [15] with key clarifications to the attack steps and concrete implementations of the attack on real circuits.

Suppose a circuit which is jointly protected using FSM and combinational obfuscation methods as described in the previous section. We know that the circuit has some memory elements storing its obfuscated S_0 to S_{n-1}, and original T_0 to T_{m-1} states along with some combinational parts. The resilience of the circuit relies on the fact that the attacker cannot distinguish between the added state elements S_0 to S_{n-1} and the original ones T_0 to T_{m-1}. If an attacker can manage to find the added states, they would be able to traverse all 2^n value space of S_0 to S_{n-1} to find out which one unlocks the circuit; then they can set it to obtain the normal operation of the circuit illegally. The FSM separation attack has three stages:

- In stage one, the circuit HDL code is used by an attacker to divide the combinational and sequential parts of the circuit. Note that this is possible since FSM memory elements can be easily distinguished from combinational part of a given IP core. However, since the attacker does not know how many state elements were added to the original circuit, the attack moves to stage two.
- For stage two, the attacker has to assume all possible values for n from 1 to L where L is the total number of state elements of the circuit, $L = m + n$. The attacker needs to figure out which subset of L states are the added S_i's. Assuming $\hat{n}$ as a hypothesis for n, there are $\binom{L}{\hat{n}}$ possible combinations for S_0 to $S_{\hat{n}-1}$.

- In the final stage, for each hypothesis, the attacker needs to (i) eliminate flip-flops which are assumed to be the (added) obfuscation ones, (ii) put zero as their outputs and (iii) see if their elimination unlocks the circuit.

It is an iterative attack where the second and third stages will loop until the circuit is unlocked and produces a valid response. The total number of trials that the attacker needs to unlock the circuit can be computed by Equation (1).

$$\Psi = \sum_{\hat{n}=1}^{L} \binom{L}{\hat{n}} = 2^L \tag{1}$$

It is worth noting that this number differs from the search space presented in [3] i.e., $(2^{m\times k})$ where m is the number of a circuit's obfuscated states and k is the number of a circuit's primary inputs. 2^L could still be a large number of states with a large number of flip flops, but $\hat{n}$ is generally a very small number due to the overhead associated with the added FSM. Hence, the iteration loop will terminate much sooner than exhausting the entire 2^L space. We will show this fact in an experimental study later in the section. A key aspect of the FSM separation attack is that its search space does not depend on the number of circuit inputs k. In fact, this attack reduces the attack complexity by separating the combinational and sequential parts of the circuit.

To examine the impacts of this complexity reduction, let us consider the following synthetic [3] and real examples. Assuming $k = n = 16$, the computation space is $2^{256} \approx 10^{77}$ which is infeasible to search. For real commercial IP cores, we consider NEO430 and ao68000, two CISC (Complex Instruction-Set Computers) open core CPUs. These IPs are relatively big circuits e.g., 3500 VHDL lines for ao68000. They are well within the range of real world circuits.

The NEO430 and ao68000 have 860 and 724 memory elements i.e., flip-flops, respectively. However, the largest logic block in these CISC IPs is a 5-bit state variable and therefore less than $2^5 = 32$ states. Obviously a large portion of memory elements in each IP is used to store processed data and a very little portion is used as state holders. This means that when a designer chooses a bigger number for n to make it more difficult for the attacker to traverse the computational space, the hardware overhead should be considered with respect to the number of memory elements which are doing state holding. For example [3] has reported 18.44 and 15.88 percent overheads for only six added state elements. These sample circuits confirm that in the real world, (1) parameter m is not very large and (2) overheads of using a large number of obfuscating states i.e., parameter n can be very high. To investigate the effects of FSM separation attacks against existing protection schemes on real circuits, we studied the feasibility of these attacks on ISCAS'89 circuits. In our evaluations, we used the largest circuits namely (a) s38417 circuit with 28 primary inputs and 1635 D-type flip-flops and (b) s38584 circuit with 38 inputs and 1425 D-type flip-flops. The circuits are synthesized targeting the Spartan-6 FPGA board using the Xilinx ISE Design Suite operating at 100 MHz.

TABLE I
SUCCESSFUL ATTACK TIME ESTIMATION FOR ISCAS CIRCUITS OBFUSCATED BY TRADITIONAL METHODS.

		Number of Added State Elements									
Circuit	#FF	1	2	3	4	5	6	7	8	9	10
S298	14	3 µs	33 µs	203 µs	968 µs	3 ms	13 ms	42 ms	122 ms	326 ms	817 ms
S344	15	4 µs	38 µs	242 µs	1 ms	5 ms	18 ms	61 ms	183 ms	510 ms	1 s
S349	15	4 µs	38 µs	242 µs	1 ms	5 ms	18 ms	61 ms	183 ms	510 ms	1 s
S526	22	5 µs	74 µs	649 µs	4 ms	24 ms	118 ms	508 ms	1 s	7 s	23 s
S641	19	5 µs	57 µs	442 µs	2 ms	13 ms	57 ms	222 ms	777 ms	2 s	7 s
S713	19	5 µs	57 µs	442 µs	2 ms	13 ms	57 ms	222 ms	777 ms	2 s	7 s
S838	32	8 µs	148 µs	1 ms	16 ms	125 ms	815 ms	4 s	23 s	1 M	7 M
S1196	18	4 µs	52 µs	384 µs	2 ms	10 ms	44 ms	164 ms	555 ms	1 s	5 s
S1238	18	4 µs	52 µs	384 µs	2 ms	10 ms	44 ms	164 ms	555 ms	1 s	5 s
S1423	74	18 µs	731 µs	19 ms	375 ms	6 s	1 M	15 M	2 H	1 D	9 D
S1488	6	1 µs	8 µs	29 µs	82 µs	197 µs	428 µs	857 µs	1 ms	2 ms	4 ms
S5378	179	45 µs	4 ms	251 ms	11 s	7 M	3 H	4 D	96 D	6 Y	105 Y
S9234	211	53 µs	5 ms	408 ms	22 s	15 M	9 H	12 D	346 D	23 Y	517 Y

For the sake of fairness, we used the same simulation setup as used in [3]. We added two extra d-type flip-flops and inserted four XOR gates into high fan-in nets in s38417 and s38584 circuits. Based on normal calculations, an *exhaustive* attack will test $2^{38\times4} = 2^{152}$ inputs to unlock the s38584 circuit, and $2^{28\times4} = 2^{112}$ inputs for the s38417 circuit. These circuits were assumed unbreakable for a polynomial time attack scheme. However, we showed that if an FSM separation attack is occurs, the circuit degenerates into lower orders.

To attack the s38417 circuit, we assume 1 to 1635 of flip-flops as FSM masking ones and check our hypothesis. This attack is accomplished in a short time, since only two of the 1635 flip-flops are intended to do FSM encoding i.e., the circuit has just 4 obfuscating states. The search took $1,340,703$ clock cycles in our simulation environment. This means that it took about 335 milliseconds to attack the obfuscated s38417 circuit. We performed the same FSM separation attack on the obfuscated circuit of s38584 and were able to break the circuit in only 254 milliseconds. It should be noted that the FSM state elements in ISCAS'89 circuits are not distinguishable from the rest of the memory elements, otherwise the attack could be significantly faster.

Table I estimates the FSM separation attack duration time for some of the other ISCAS circuits when different numbers of flip-flops are used in the obfuscating FSM. In this table we have examined up to 10 added obfuscation flip-flops and calculated the required time for a successful FSM separation attack. For those circuits with a large number of flip-flops (e.g S9234 and S5378) the attack time is in order of hundreds of years (which is still assumed a feasible attack on distributed and parallel systems). Nevertheless, we can conclude that regardless of the circuit size, using a separated obfuscating FSM to lock the chip cannot protect the chip especially when the number of memory elements in the obfuscating FSM is not very high (see S1488 results with 6 flip-flops in Table I). On the other hand, adding a large number of obfuscating states implies an unacceptable overhead on the protected circuit.

In this paper, we propose a method where the robustness of obfuscation does not depend on the number of added states. Since it checks for the correct key before every FSM transition, it can be used for any desired level of obfuscation.

IV. The Proposed *Mystic* Method

In order to reduce the chance of a successful attack, we believe that the added obfuscating FSM should not be completely separated from the original FSM. We propose a masking technique that is active during the whole operation of the circuit in its lifetime. The technique combines original and obfuscated states and significantly reduces the probability of an attack successfully unlocking the target circuit. As shown in Figure 3, the circuit starts working from an original state and works correctly. However, in each state in the circuit's FSM, a set of the key bits should be correct in order for the FSM to go through the correct transition and operate consistently. On a wrong key, the FSM goes to a wrong state which will perturb the entire computation for the rest of the circuit operation.

The key underpinning of the proposed technique is the concept that a designer may add more state transitions and/or additional states to mask the original FSM. Going back to Figure 1, it should be highlighted that the proposed method actually adds a *Locking Transition* to the model in opposite direction of the *Unlocking Transition*. To compensate for a small state space, a designer may add extra states as a means of increasing the transition candidates for the masking process.

We propose an iterative masking algorithm to systematically add obfuscation transitions or states to a given circuit. On each iteration, the algorithm chooses a state to which an extra obfuscating transition is going to be added. To choose an appropriate state, we propose a simple but effective *Security Metric* that can be assigned to each state *s* as Equation (2)

$$SM_s = \frac{OutEdge_s}{KeyCount_s + 1} \tag{2}$$

where $OutEdge_s$ is the total number of transitions started from the state *s* and $KeyCount_s$ is the number obfuscating transitions which have been previously added to state *s* during the masking process. One general observation is that the states with high *OutEdge* tend to be more critical in the operation of the FSM. Therefore, an intuitive and judicious way for selecting the order of states to guard when all the circuit states cannot be guarded is to use the *OutEdge* degree. In addition, having the *KeyCount* on the denominator of the metric gradually decreases the importance of previously masked states. *Locking transitions* are added to the original circuit's FSM in a manner that allows the circuits to check their activation keys at runtime. Whenever the input key is not valid, the circuit follows one of the added *Locking transitions* and goes to an intentionally wrong state.

Algorithm 1 presents the masking procedure. It receives an input key $\mathcal{K}$ and an FSM graph $\mathcal{F}$. First, in lines 2 and 3, it prepares two lists for storing $KeyCount$s and Security Metrics associated with each state. Then in lines 5 to 8, it calculates initial security metrics for all states. The main masking iterations start from line 10. For each key bit $\mathcal{K}[j]$,

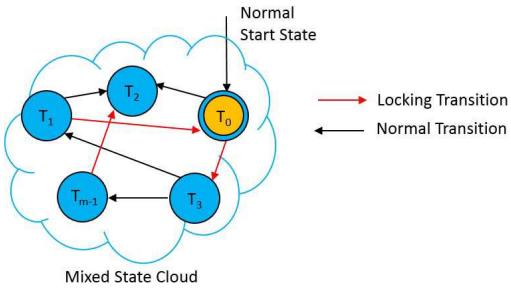


Fig. 3. Our proposed FSM encoding methods.

Algorithm 1: The Proposed Masking Algorithm

Input: Key $\mathcal{K}$
Input: FSM graph $\mathcal{F}$ with $S_\mathcal{F}$ states and $E_\mathcal{F}$ edges
Result: Obfuscated FSM graph $\mathcal{F}$

1 *// Initialization*
2 Define zero-initialized list of integers $KeyCount$ with size of $|S_\mathcal{F}|$;
3 Define zero-initialized list of floats $SecMetric$ with size of $|S_\mathcal{F}|$;
4 *// Calculating Security Metrics*
5 **for** $i \leftarrow 1$ **to** $|S_\mathcal{F}|$ **do**
6 $\quad OutEdge \leftarrow$ number of edges in $E_\mathcal{F}$ starting from $S_\mathcal{F}[i]$;
7 $\quad SecMetric[i] \leftarrow \frac{OutEdge}{KeyCount[i]+1}$;
8 **end for**
9 *// Masking* $\mathcal{F}$
10 **for** $j \leftarrow 1$ **to** $|\mathcal{K}|$ **do**
11 $\quad m \leftarrow$ index of the largest value in $SecMetric$;
12 $\quad$ Add to $E_\mathcal{F}$ an edge from $S_\mathcal{F}[m]$ to a random state in $S_\mathcal{F}$ with $\bar{\mathcal{K}}[j]$ activator;
13 $\quad$ *//Update Security Metric for* $S_\mathcal{F}[m]$
14 $\quad KeyCount[m] \leftarrow KeyCount[m] + 1$;
15 $\quad SecMetric[m] \leftarrow \frac{OutEdge}{KeyCount[m]+1}$;
16 **end for**
17 Return $\mathcal{F}$;

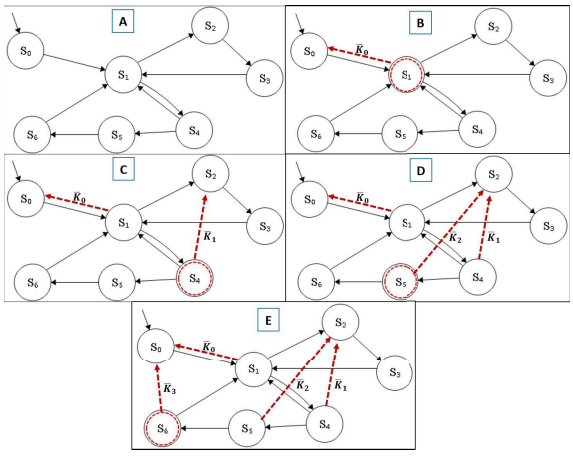


Fig. 4. An example of the proposed masking algorithm.

the algorithm finds the state $S_{\mathcal{F}}[m]$ with the highest security metric value and then adds a transition $E_{key=\bar{\mathcal{K}}[j]}$ starting from $S_{\mathcal{F}}[m]$ which ends at a random (not already connected) state. The transition $E_{key=\bar{\mathcal{K}}[j]}$ shifts the FSM to a wrong state whenever the attacker's key bit is not equal to the correct value $\mathcal{K}[j]$.

To illustrate the operation of the proposed *Mystic* algorithm, we applied it to the FSM graph shown in Figure 4-A. As shown in this figure, the highest Security Metric belongs to states S_1 and $S4$ due to their higher $outEdge$ degree which is equal to 2. Security Metric for the other five states is 1. Therefore the first bits of the key are negated and inserted as obfuscating transition of state S_1 in Figure 4-B and state S_4 in Figure 4-C. By updating the security metrics, we have all states with security metrics of 1. The algorithm chooses a random state on each iteration (S_5 and S_6 on Figure 4-D and 4-E) and adds an obfuscating transition to them. This process continues until all bits of the activation key are used. Note that it is possible for a transition to have multiple key bits or even a function of key bits as an activating function on obfuscating transitions. To answer the question of wether we have multiple key bits activator or not, we have to compare $|\mathcal{K}|$ with $|S_{\mathcal{F}}|$. If $|\mathcal{K}| \geq |S_{\mathcal{F}}|$, we have some states with multiple key bits activators, but when $|\mathcal{K}| < |S_{\mathcal{F}}|$ we can manage to have no such state. Since allowing multiple key bits as activator i) increases the hardware overhead of our proposed method and ii) loses the termination condition of our proposed masking algorithm, in this example we bound the number of added obfuscating transitions to 4.

In terms of estimating the obfuscation level of *Mystic*, it is important to note that each state is obfuscated using a subset of the masking key. All of the key bits in that subset should be correct for the FSM to perform a single transition correctly. Similarly, for the following transition, another subset of the key bits should be correct. From the attacker's perspective, for each state, a subset of the key bits needs to be guessed correctly. For key hypothesis checking, the attacker needs to test all possible values for all combinations of key subsets with different lengths. The compute complexity of this verification operation is:

$$\phi = \sum_{k=1}^{|\mathcal{K}|} \binom{|\mathcal{K}|}{k} 2^k \tag{3}$$

It is worth noting that the key size $|\mathcal{K}|$ in (3) is much bigger than the number of states in the circuit, L. In fact, probability of correctly passing a state by an unauthorized attacker is $\frac{1}{\phi}$.

Generally, the attacker's only reference to evaluate the correctness of a key guess is the output of the circuit i.e., a correct output. Under the *Mystic* obfuscation method, the attacker does not have any reference output, especially when it comes to large IP cores like CPUs and cryptographic cores. This constitutes another important security feature of the *Mystic* technique to further reduce the chance of a successful attack. Because these large IPs oftentimes do not produce meaningful intermediate outputs, the attack must pass several transitions correctly to produce a meaningful output. Without an intermediate output, an attacker would need to guess all the key bits used in several states. Therefore the probability that an unauthorized attacker generates a meaningful output is reduced to $\frac{1}{\phi^v}$ where v is the average number of cycles needed to produce the next meaningful output of the obfuscated circuit.

TABLE II
OVERHEADS OF THREE SAMPLE OBFUSCATED IP CORES USING *Mystic* SYNTHESIZED FOR A XILINX VIRTEX-7 FAMILY FPGA.

	AES IP Core			CR16 IP Core			RISC IP Core		
Key Length	Slice Reg.	Slice LUTs	Slices	Slice Reg.	Slice LUTs	Slices	Slice Reg.	Slice LUTs	Slices
0	824	2873	1073	4211	1920	1330	4473	1825	1490
1	824	2873	1075	4211	1920	1329	4473	1828	1492
4	824	2876	1079	4211	1922	1335	4474	1841	1516
8	826	2894	1089	4213	1931	1359	4491	1899	1593
16	831	2991	1094	4221	2172	1396	4549	2014	1689
32	847	3224	1102	4296	2354	1416	4678	2288	1803
64	859	3489	1131	4384	2567	1489	4792	2413	1898
96	863	3755	1253	4432	2931	1564	4881	2698	1972
128	872	4093	1390	4503	3456	1679	5153	2984	2299
AVG	841	3229	1142	4298	2352	1433	4662	2198	1750

V. *Mystic* HARDWARE OVERHEADS

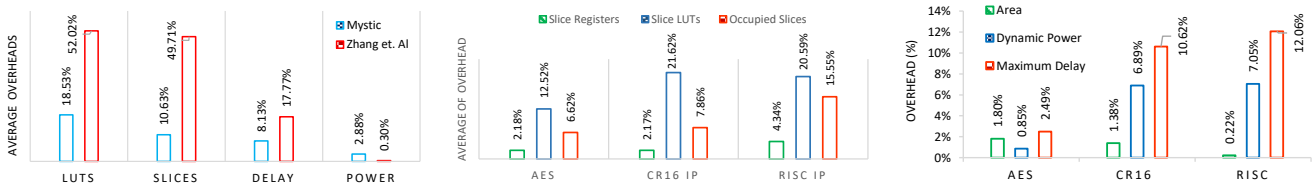
To evaluate hardware overheads of the *Mystic* technique, we have developed a CAD tool in Python. The tool takes the RTL description of a circuit, extracts the state machine and then obfuscates it using the key provided by the user. Finally, the original FSM inside the circuit is replaced with the obfuscated FSM for heightened security. We performed our experiments on three benchmarks circuits of 1) AES cryptographic core, 2) A RISC processor and 3) CR16 microprocessor. Results of the synthesis for both FPGA and ASIC implementations are compared with those of the recent approach proposed in [8].

A. FPGA Implementation Results

The synthesis results of the three benchmarks on the Xilinx Virtex-7 xc7vx330t FPGA board are shown in Table II. The first row in this table shows the hardware utilization for the three normal benchmarks with no obfuscation. We performed the obfuscation under 1, 4, 5, 16, 32, 64, 96, and 128 widths of key. Based on the results, overhead growth is not very sharp i.e., we have the highest area overhead of 14% for the RISC IP Core benchmark when the obfuscation key of 128 bits is used. The main reason of such a relatively low overhead is that the FSM part is not normally a large portion of the whole digital circuit. It can be seen that the key length growth mostly affects the number of used LUTs in the FPGA implementation. Since the proposed masking algorithm does random selections in some steps, we repeated the obfuscation process of benchmarks with 128-bit key for 10 times. Figure 5b shows the average overheads when a 128-bit obfuscation is done on benchmarks. We have also compared the *Mystic* method with a recent work presented in [8] in terms of area, delay, and power. Figure 5a illustrates the overheads comparison between *Mystic* and the PUF-FSM based method.

B. ASIC Implementation Results

In the second set of experiments, we added the ITC99 circuit benchmarks to the three mentioned cores. We synthesized



(a) FPGA resource overheads for *Mystic* and [8]. (b) *Mystic* overheads per obfuscated IP cores. (c) *Mystic* ASIC implementation overheads.

Fig. 5. FPGA Virtex-7 and ASIC 45nm technology implementations resource utilization results.

TABLE III
OVERHEAD RESULTS FOR ITC99 CIRCUIT BENCHMARKS SYNTHESIZED FOR AN ASIC 45NM TECHNOLOGY.

Circuit	Design Overheads (%)		
	Area	Delay	Power
b01	3.14	2.55	5.63
b02	3.05	2.41	5.88
b03	2.11	1.89	4.12
b04	2.14	1.95	4.05
b05	2.89	2.23	5.41
b06	3.12	2.65	5.88
b07	3.16	2.89	5.69
b08	2.88	2.11	5.01
b09	2.96	2.32	4.87
b10	3.99	2.96	6.56
b11	3.55	2.88	6.74
b12	4.55	3.98	7.86
b13	4.14	3.87	7.42
b14	6.08	8.97	10.56
b15	4.84	5.44	8.67
b17	4.96	5.23	8.91
b18	8.06	8.88	11.65
b19	9.34	9.76	11.98
b20	9.55	9.95	12.58
b21	9.61	10.58	12.74
b22	10.88	12.65	14.23
b30	8.54	8.53	10.89
Average	**5.14**	**5.21**	**8.06**

all 26 circuits using the Synopsis Design Compiler tool for 45nm technology and overheads are logged. The area, dynamic power and maximum delay overheads of the three IP cores with respect to their non-obfuscated versions are shown in Figure 5c. The AES circuit has the lowest overheads and RISC processor has the highest. This is due to more complex combinational part of the AES circuit in comparison with its simple sequential logic. The *Mystic* method does not add any additional state to the FSM. Instead it adds extra combinational logic to produce the transition guard considering the key inputs. Table III shows the overheads for an obfuscated ITC99 circuit benchmark by *Mystic* for the ASIC 45nm technology.

VI. CONCLUSIONS

In this paper, we showed that the previously proposed FSM obfuscation methods can be easily broken with a simple *FSM separation attack*. This attack is able to break secure-through-obfuscation circuits with a very low time complexity. We also presented an always-on obfuscating method which acts as a security watchdog at runtime and throughout the lifetime of the chip. When an attacker applies the first wrong key, this action activates the security watchdog and the chip goes to an intentionally wrong state resulting in incorrect functionality. Since the proposed method has a runtime defense mechanism and covers the lifetime of the chip, it is also robust against SAT attacks. The synthesis results and the comparative study with previous obfuscation methods show that the proposed method provides stronger circuit obfuscation guarantees with better efficiency in terms of area, delay, and power consumption.

VII. ACKNOWLEDGMENTS

This research is partially supported by the NSF grant (No. CNS- 1745808).

REFERENCES

[1] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.

[2] "Innovation Is at Risk as Semiconductor Equipment and Materials Industry Loses up to $4 Billion Annually Due to IP Infringement," http://www.marketwired.com, [Online; accessed 12-July-2017].

[3] R. S. 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*, vol. 28, no. 10, pp. 1493–1502, 2009.

[4] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 83–89.

[5] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2015.

[6] A. Nejat, D. Hely, and V. Beroulle, "Facilitating side channel analysis by obfuscation for hardware trojan detection," in *2015 10th International Design Test Symposium (IDT)*, 2015, pp. 129–134.

[7] R. S. Chakraborty and S. B., "Rtl hardware ip protection using key-based control and data flow obfuscation," in *Proceedings International Conference on VLSI Design*, ser. VLSID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 405–410.

[8] J. Zhang, Y. Lin, Y. Lyu, and G. Qu, "A puf-fsm binding scheme for fpga ip protection and pay-per-device licensing," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 6, pp. 1137–1150, 2015.

[9] G. Sumathi, L. Srivani, D. T. Murthy, A. Kumar, K. Madhusoodanan, and S. A. V. S. Murty, "Structural modification based netlist obfuscation technique for plds," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2016, pp. 1418–1423.

[10] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental sat-based reverse engineering of camouflaged logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[11] M. E. Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes," in *NDSS*, 2015.

[12] T. Meade, S. Zhang, and Y. Jin, "Ip protection through gate-level netlist security enhancement," *Integration, the VLSI Journal*, vol. 58, no. Supplement C, pp. 563 – 570, 2017.

[13] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. ACM, 2013, p. 8.

[14] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[15] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.