Novel Dynamic State-Deflection Method for Gate-Level Design Obfuscation

Jaya Dofe, *Student Member, IEEE*, and Qiaoyan Yu, *Member, IEEE*

***Abstract*—The emerging security threats in the integrated circuit supply chain do not only challenge the chip integrity, but also raise serious concerns on hardware intellectual property (IP) piracy. Hardware design obfuscation is a promising countermeasure to resist reverse engineering attacks and IP piracy. The majority of existing hardware obfuscation methods modify the original finite state machine (FSM) by adding additional state transitions and utilizing a key sequence to lock the transition from the nonfunctional states to the functional reset state. Those methods are effective to prevent attackers from entering the normal functional mode but they lack resilience if the FSM is already in the normal mode. This paper proposes to protect all the states with a low-cost state-deflection-based obfuscation method, which dynamically deflects state transitions from the original transition path to a black hole cluster if a wrong key is applied. Unlike other works that use static transitions between legal states to black hole states at the design time, this method utilizes a state rotation function (Rotatefunc) and selective register flipping function (Mapfunc) to dynamically control the state deflection paths. Hence, the difficulty of reverse engineering and thwarting register overwrite attacks is increased. Simulations performed on ISCAS'89 benchmark circuits show that the proposed method significantly reduces the difference of the net toggle activities between the correct and wrong key scenarios, and achieves up to 56% higher code coverage than the most efficient obfuscation method. Thanks to the dynamic deflection feature, on average, this method generates about 100 more unique state register patterns than other methods with moderate power increase. Moreover, the proposed method achieves the Hamming distance of primary outputs and state registers close to 50%.**

***Index Terms*—Design obfuscation, gate-level netlist, hardware security, intellectual property (IP) piracy, reverse engineering, state deflection.**

I. Introduction

DUE TO the threats from the globalized semiconductor supply chain, the security of integrated circuits (ICs) has emerged as a serious concern [1]–[10]. Among the well-recognized security threats, the growing hardware intellectual property (IP) piracy and reverse engineering challenge the traditional chip design and fabrication [2]–[4], [11], [12].

Manuscript received October 7, 2016; revised February 7, 2017 and March 22, 2017; accepted March 29, 2017. Date of publication April 25, 2017; date of current version January 19, 2018. This paper was recommended by Associate Editor Q. Xu. *(Corresponding author: Qiaoyan Yu.)*

The authors are with the Department of Electrical and Computer Engineering, University of New Hampshire, Durham, NH 03824 USA (e-mail: qiaoyan.yu@unh.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2697960

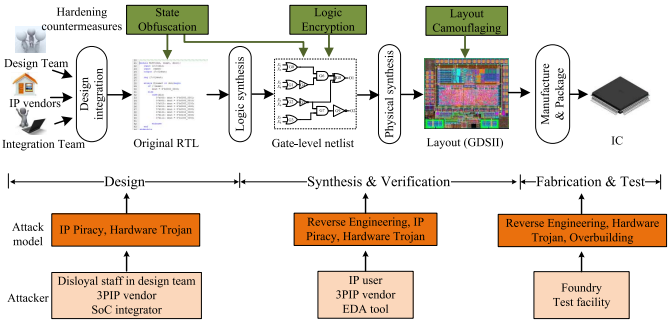


Fig. 1. Hardware hardening techniques in the IC design flow.

IP piracy attacks can take place at various stages in the IC supply chain. As shown in Fig. 1, the potential IP piracy attackers could be disloyal members in a design team, third-party IP vendor, and systems-on-chip (SoC) integrator at design, synthesis, and verification stages. In the fabrication stage, an untrusted foundry may overbuild IP cores and sell them under a different brand name to make profit. To resist IP piracy, some countermeasures like hardware IP metering [3], [4], authentication-based IP protection [13]–[16], and source-code encryption rely on the rigorous chip activation process. Alternatively, other IP piracy countermeasures change the name of internal nets in original netlists or obfuscate the original register-transfer level (RTL) design at certain degree, so that it is not practical for an attacker to retrieve the original chip design [17]. As demonstrated by the countermeasures in [14]–[16] and [18]–[23], design obfuscation countermeasures are promising for IP vendors to strengthen their hard and soft IPs against reverse engineering and IP piracy attacks.

The majority of existing methods obfuscate RTL designs [16], [18], [21]. Countermeasure designers typically need knowledge of states in the original design and the signals triggering state transitions. Unfortunately, it is not trivial to collect the above mentioned knowledge if the design under protection is in a format of a gate-level netlist, which is a common format to deliver a firm hardware IP/macro. In this paper, we propose a state-deflection obfuscation method for the gate-level netlist without using the prior knowledge of the used and unused states and transition trigger signals.

The IPs under consideration in existing works are mainly hard IP core/chip (i.e., only primary inputs and outputs are available to the attackers), and the testing entity is assumed to be trusted. If the key sequence is sufficient long, the obfuscated finite state machine (FSM) is theoretically not breakable

by brute-force attacks. The assumption does not hold for soft or firm IP scenarios. Attackers may use advanced electronic design automation (EDA) tools (e.g., code coverage analysis in NCverilog and Modelsim) to probe in the RTL or gate-level netlist and obtain the targeted details, such as FSM register content, and the correlation between FSM states and different wrong keys. In logic encryption methods for combinational circuits [24], [25], output Hamming distance (HD) is adopted as a metric to quantize the ambiguousness introduced by a wrong key. It is imperative to evaluate the attack resilience of obfuscated sequential circuits with a similar security metric.

Outline of this paper is as follows. Existing obfuscation methods and the main contributions of this paper are summarized in Section II. Security metric and attack model used in this paper are presented in Section III. Our novel obfuscation method and the corresponding implementation flow are proposed in Section IV. Thorough comparison of the proposed method with existing relevant approaches is provided in Section V. This paper is concluded in Section VI.

II. Related Work and Our Contributions

A. Related Work

An anti-piracy design flow for SoC hardware obfuscation is presented in [15]. The main principle of this method is to add a key-controlled obfuscation mode before FSM enters the normal mode. In the obfuscation mode, two keys, respectively, control the obfuscation and authentication FSMs. Theoretically, an adversary cannot reach the normal mode without a correct key sequence. This method lacks of protection on the normal mode. The essence of this method is to prevent an attacker from finding the true reset state for the normal mode. This obfuscation concept is extended to RTL obfuscation in [18]. The RTL hardware description is converted into control and data flow graphs and then dummy states are added to obfuscate the design with a enabling key sequence. Their follow-up work [26] proposes to increase the number of unreachable states for the obfuscation mode to thwart hardware Trojans triggered by rare events in the circuit. The authors believe that the expanded state space will make it more difficult for attackers to identify a true rare event and successfully insert a hardware Trojan.

The work [21] suggests using obfuscation to thwart hardware tampering. Different from using an external key sequence in [15], this paper locks the design with an internally generated *codeword* that are based on a special order of state transitions chosen as the secret. Although this method may be helpful to address the issue of key leaking in [15], [18], and [26], the degree of circuit obfuscation is significantly reduced. The reason is, a wrong key only causes the FSM temporally deviate from the correct transition path after one or few state transitions, the normal state transition will resume. Another limitation of this method is, if the internal code-world generation module is compromised, the obfuscation on the entry mode will be nullified.

A remote activation scheme [14] is proposed to resist IP piracy. The continuousness of state transitions is granted by a key uniquely generated by the random unique block on the chip. This method duplicates one of the existing states three times, rather than creating new states as in [15], [18], [21], and [26], to stop the transition from the state under protection to the next valid state when a wrong key is applied to the FSM. The limitation of this method is, an attacker could identify the stalled state and then enforce the state registers change to other values. Thus, the key protection fails.

The piracy-aware IC design flow in [3] proposes to enrich the RTL description with an on-chip true random number generator and public-key cryptography (RSA). Koushanfar further extended their work in [4]. Besides the idea of an obfuscation mode similar to the work in [18], Koushanfar [4] also add a channel from the normal state transition to the black hole states, which prohibits the state recovery to the normal operation. The path from normal states to black hole ones provides protection for the normal operation mode.

Methods for IP piracy are mainly aiming for hard IP core, rather than soft or firm IP cores. The latter ones may be attacked with the assist of advanced EDA tools, especially testing and reverse engineering tools, such as TetraMAX [26], formality [15], and electronic system level (ESL) [11]. The analysis in existing work assumes that an attacker needs to exhaustively search for the secret key sequence to reach the functional initial states. However, if the IP core is delivered as a soft IP (or even a firm IP), an attacker could overwrite FSM registers and skip the stalled state or a repeated state-transition loop without using the correct key. The correlation between FSM register content and wrong key sequences may leak some clues to speed up the processing of reverse engineering and IP piracy attacks.

B. Our Contributions

The preliminary version of this paper is available in [27]. In this paper, we extend our early work by comparing our method with existing obfuscation approaches using two more security metrics and providing a way to leverage hardware cost and security vulnerability. Overall, the proposed method is superior to existing other works as follows.

1) We propose a low-cost state-deflection method to protect every state in the normal mode. This is different than the existing methods that rely on the obfuscation mode to prevent the attacker from entering the normal operation mode and do not further protect any normal states. Unlike the previous work, our wrong-key induced state deflection is not predetermined at the design time. We propose to dynamically and selectively flip the state registers based on the unique wrong key applied.
2) Our obfuscation is designed for general gate-level netlists, rather than RTL designs. Furthermore, our method eliminates the need of the prior knowledge of the used and unused states and the trigger signals for state transitions. This makes it feasible to decouple the obfuscation step from the original function implementation.

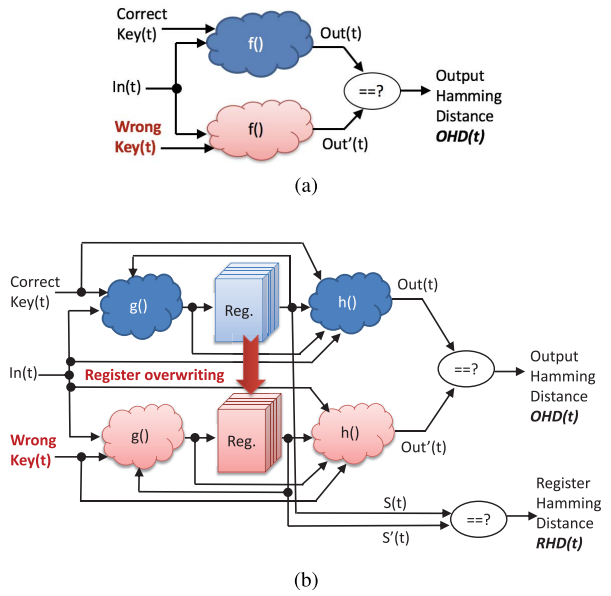


Fig. 2. Measuring HD in (a) combinational and (b) sequential circuits to retrieve the obfuscation key.

3) This paper is the first effort that performs a comprehensive comparison among different obfuscation methods in the context of soft/firm IP core obfuscation. We utilized EDA tools to generate test patterns, measure code coverage, analyze net toggle activity, and evaluate area, delay, and power overhead.
4) We propose to use register HD (RHD) to assess the correlation between wrong key sequences and FSM state registers in sequential circuits. Furthermore, we propose the self-correlation of FSM state registers as a metric to ensure that the register patterns belonging to the black hole states will not be repeated in a reasonably long period of time.

III. Security Metric and Attack Model

A. Security Metric

HD between two binary vectors of equal length measures the number of positions at which the vectors have different values. The work [24], [25] adopted HD as a security metric to assess the difficulty for an attacker to retrieve the key from encrypted (i.e., locked) combinational circuits. The output HD (OHD $\in$ [0, 1]) for combinational circuits can be obtained from the setup shown in Fig. 2(a). As can be seen, HD depends on the input and the applied key at the instance t. The corresponding mathematical expressions are represented in (1)–(3). A design resulting in a lower HD indicates that the applied key has a higher correlation with the output. Thus, an attacker can exploit the output to retrieve the correct key.

$$\mathrm{Out}(t) = f(\mathrm{In}(t), \text{Correct Key}(t)) \tag{1}$$

$$\mathrm{Out}'(t) = f(\mathrm{In}(t), \text{Wrong Key}(t)) \tag{2}$$

$$\mathrm{OHD}(t) = \frac{\mathrm{Out}(t) \wedge \mathrm{Out}'(t)}{\text{Total Number of Outputs}} \tag{3}$$

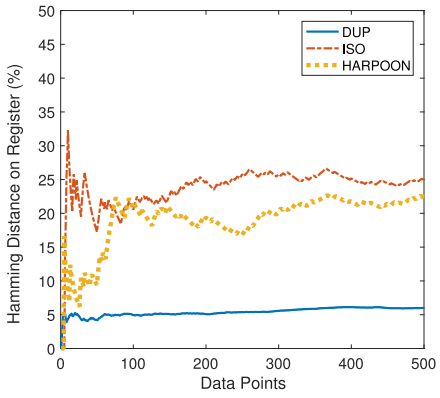


Fig. 3. RHD of s5378 obfuscated with existing methods. Note, the ideal HD is 50%.

If we follow the same idea as used in combinational circuits to obtain the OHD from sequential circuits, the correlation between the key and the output is weaker than that in combinational circuits. As shown in Fig. 2(b), the registers *Reg.* block is not on the path of the direct communication between the key and the output. Therefore, even if we duplicate the circuit, OHD does not provide a clear clue for attackers to reverse engineer the function and the key. As expressed in (4), the output of a sequential circuit has one more dependent factor than that of a combinational circuit, the state register $S(t)$. The state register $S(t)$ is a function of inputs, keys and the previous state register $S(t-T_{\mathrm{clk}})$, as described in (5). T_{clk} stands for the clock period.

$$\mathrm{Out}(t) = h(\mathrm{In}(t), \mathrm{Key}(t), S(t)) \tag{4}$$

$$S(t) = g(\mathrm{In}(t), \mathrm{Key}(t), S(t - T_{\mathrm{clk}})) \tag{5}$$

In addition to compute OHD in (3), we also propose to examine the RHD (RHD $\in$ [0, 1]) due to the following reasons.

1) If an attacker can probe in the registers in the circuit with the correct key, reverse engineering methods for combinational circuits are also applicable for recovering the $g()$ function in sequential circuits.
2) If an attacker finds a way to copy the registers content from the circuit copy with the correct key to the circuit using a wrong key, he/she could retrieve the $h()$ function without knowing the $g()$ function.

We define RHD in (6), where the number of state bits stands for the length of $S(t)$.

$$\mathrm{RHD}(t) = \frac{S(t) \wedge S'(t)}{\text{Total Number of State Bits}} \tag{6}$$

We conducted experiments on ISCAS benchmark circuit s5378 to examine the RHD of different methods. Test benches for different methods were generated by TetraMAX. All registers in s5378 were probed for RHD calculation in 500 testing cycles. As shown in Fig. 3, DUP [14] yields an RHD of approximately 5%. The RHDs for ISO [26] and HARPOON [15] are close to 25%, which is well below the ideal HD, 50%. Thus, the existing obfuscation methods are vulnerable to reverse engineering attacks if the attacker can manipulate the register contents.

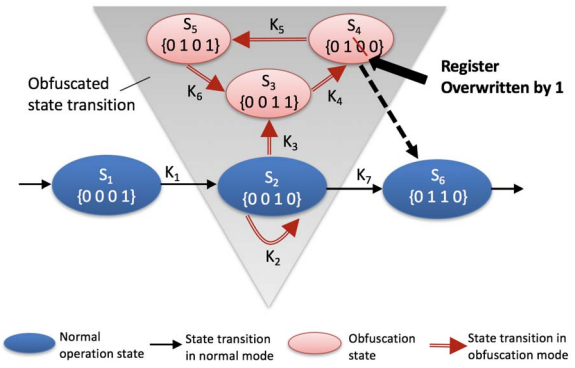


Fig. 4. Simplified example of state obfuscation.

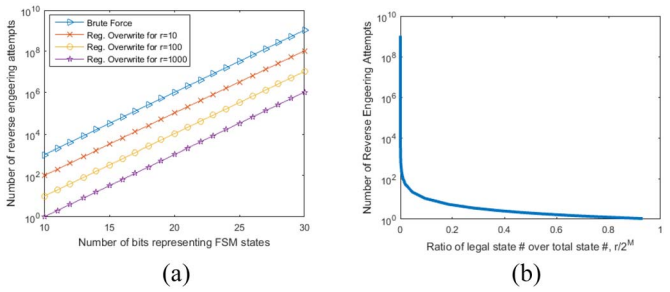


Fig. 5. Attack attempt reduction by register overwrite attack to recover the key sequence. (a) Attempt reduction versus number of bits used in the FSM state. (b) Attempt reduction versus the ratio of legal states over the total states in FSM.

B. New Attack Model

State obfuscation provides a method to lock the normal state transitions. Fig. 4 summarizes the common feature of existing state obfuscation methods. A key sequence $K_1, \ldots, K_7$ guides the state transitions. An authorized user will have the correct key sequence $K_1, \ldots, K_7$ to obtain the state transition path $S_1 \rightarrow S_2 \rightarrow S_6$. Without the correct key sequence, the circuit will run into the obfuscation mode, which is highlighted by the shaded triangle in Fig. 4. The state machine will sustain in an inner loop $S_3 \rightarrow S_4 \rightarrow S_5$ or the current state S_2 forever. However, if an attacker could manipulate the register content by overwriting some registers, instead of seeking the correct key sequence, S_4 will change to S_6 and the normal state transition will resume.

We envision two possible ways that may implement the register overwrite attack. If the attacker has a copy of an activated IP, he/she is able to observe the normal state transition from the initial power-up state. Then, the attacker can compare the state transition of the inactivated IP with that of the activated IP and determine which state vector should be used to overwrite the stalled state in the inactivated IP. A wired logic can be used to externally change the state register content without the need of modifying the obfuscated netlist. For advanced attackers, they can alter the sequential elements in the IP netlist by adding XOR logic and an arbitrary obfuscation key vector. Even if the attacker does not have a copy of an activated IP, he/she can also try to blindly and manually change the state registers content with two techniques mentioned above.

Advanced EDA tools like Cadence Incisive Comprehensive Coverage (ICCR) and ESL [11] are capable of providing code analysis to facilitate the recovery of high-level circuit designs. Hence, the assumption that attackers have to exhaustively search for the secret key sequence to pirate IPs does not hold true. It is possible for attackers to overwrite the register content with externally added signals. We assume that the number of FSM states after obfuscation is 2^M, the number of states used in the original FSM without obfuscation is r, and the rest of the states belong to obfuscation states or black hole states. If an attacker performs a register overwrite attack that modifies one register's value, the FSM state might turn into a normal state with a probability $P_{\text{BH}\rightarrow\text{Normal}}$ expressed in (7). Thus, statistically speaking, the average number of brute-force attempts that an attacker needs to bypass the obfuscation mode is equal to (8), instead of 2 to the power of the key size

$$P_{\text{BH}\rightarrow\text{Normal}} = \frac{r}{2^M - 1 - r} \tag{7}$$

$$\#\text{Attempts} = \frac{1}{P_{\text{BH}\rightarrow\text{Normal}}}. \tag{8}$$

In Fig. 5, we plot the maximum number of attempts needed to leave the sustained FSM state. We assume that an attacker can overwrite the state register by flipping one flip-flop logic value in each attempt and check whether the FSM operates in one of the normal operation states. As can be seen from Fig. 5(a), the register overwrite technique can reduce the number of attempts by three orders of magnitude over using a brute-force method, if the number of normal states in the FSM is 1000. Certainly, if the number of normal operational states is significantly less than the number of obfuscation and authentication states, the effort of reverse engineering will dramatically increase. As shown in Fig. 5(b), when the ratio of the number of normal states over the total number of states in the obfuscated FSM is close to 0, the number of reverse engineering attempts for register overwrite attack is approaching to that of needed in the brute-force method. However, more states in the FSM normal mode help to reduce more attack attempts. This is reasonable, as the defender pays more hardware overhead for a better obfuscation.

IV. Proposed Dynamic State-Deflection Method

A. Overview of Proposed Method

1) Exploiting Black Hole Cluster to Protect Normal Mode: Obfuscation on the true power-up state of an FSM has been demonstrated as a promising countermeasure to thwart IP piracy. The general principle for existing obfuscation is depicted in Fig. 4. A secret key sequence has to be known in order to pass the authentication/obfuscation mode. We assume that the secret key is not accessible except the IP provider. Once the FSM enters the normal operation mode, no more protection is available. This leaves a security vulnerability for attackers to exploit, especially in the context of soft or firm IP piracy scenarios.

To address this security vulnerability, we propose a novel dynamic state-deflection method to protect the states in the normal operation mode. As shown in Fig. 6(a), each normal state ST_x transition in the normal mode needs to pass the key

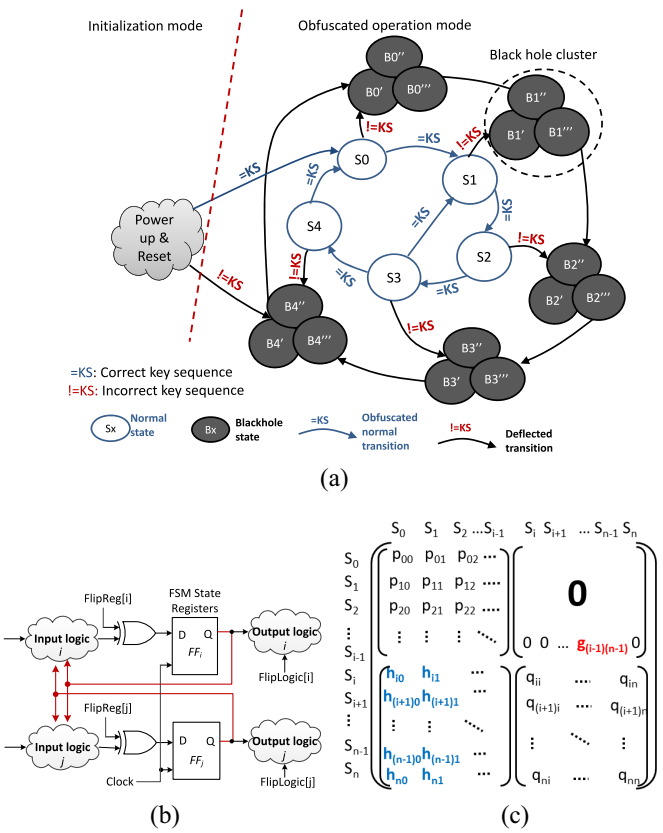


Fig. 6. Proposed dynamic state-deflection method for netlist obfuscation. (a) State transition graph, (b) circuit example of our method, and (c) proposed state transition paths represented with a Markov chain matrix.

authentication. If a wrong key (i.e., ! = KS) is applied to the FSM, a normal state will be deflected to its black hole cluster B_x. The FSM never returns to the normal state once it enters the black hole cluster.

Each black hole cluster is composed of multiple states (only three shown for brevity), rather than a single one. This is because we map each black hole state to a unique wrong key (to save the hardware cost, multiple wrong keys can share one black hole state). As we cannot predict which wrong key will be applied by the attacker, we propose a *Mapfunc* function to dynamically assign one black hole state to one wrong key case. The state within the black hole cluster does not remain stable; instead, it constantly switches to other black hole states. More details of black hole state creation and dynamic transition are provided in Sections IV-B and IV-C, respectively.

Considering that a gate-level netlist does not provide a clear picture of used and unused states, we use a state flip vector *FlipReg* to selectively invert the state bits and a logic flip vector *FlipLogic* to modify the primary outputs using the setup shown in Fig. 6(b). Thanks to the inherent feedback loops in the sequential circuit, the FSM state bits will naturally switch over time without using predetermined transition specifications.

2) Mathematical Model Comparison: The modification on the state transition graph can be described with a Markov chain transition matrix, which indicates the probability for changing

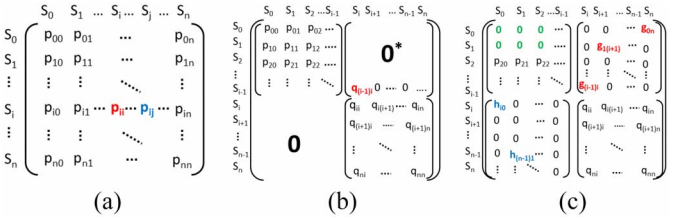


Fig. 7. Transition Markov chain matrices. (a) DUP [14], (b) HARPOON [15], (c) Codeword [21].

one state to another one. Let us use $S_0, \ldots, S_{i-1}$ to represent the states for authentication, obfuscation, or black hole cluster and use $S_i, \ldots, S_n$ to represent the normal operation states. Fig. 6(c) expresses the Markov chain matrix of our obfuscation method, in which the nonzero $g_{i(n-1)}$ stands for the single transition path from the obfuscation mode to the normal operation mode. The nonzero H matrix (composed of h_{ij}) in the big matrix means that our method creates the channel for each normal state to enter one of the black hole states.

Markov chain transition matrices for other methods under comparison, DUP [14], HARPOON [15], and Codeword [21], are shown in Fig. 7(a)–(c), respectively. In the DUP method [14], the state under protection (for instance, S_i) is duplicated several times and that state is locked by a key. Only when the correct locking key is applied to the state machine, the state machine will leave the state S_i. The application of a wrong key will lead the state machine remain in the same state S_i. In essence, this obfuscation method increases the self-transition probability for the state S_i (i.e., p_{ii}), and thus the probabilities for S_i transitioning to other state p_{ij} will decrease accordingly.

The two zero submatrices for the HARPOON [15] Markov chain transition matrix shown in Fig. 7(b) obstruct the transition between obfuscation states and the true functional states except the reset state in the functional mode. As we can see, $q_{(i-1)i}$ is the single nonzero probability. The P (element p_{ij}) and Q (element q_{ij}) matrices are strictly confined in its own submatrix. If the attacker overwrites the original state, the key obfuscation step can be bypassed. The Markov chain matrix shown in Fig. 7(c) is the Codeword-based obfuscation method [21], in which the obfuscated state is not a black hole state and can return to a normal functional state from the obfuscated state.

Through the matrix comparison we can see, our method set a strict rule on prohibiting the state transition from black hole states to normal states. Moreover, our method leaves more channels for normal states transiting to black hole states than existing methods. Thus, it is easier for the FSM to enter in one of the black hole cluster and remain there forever, even if register overwrite attack may happen. Our method raises the bar for reverse engineer and IP piracy attackers to compromise the design.

B. Black Hole State Creation for Gate-Level Obfuscation

In RTL obfuscation methods, the design team (i.e., the obfuscation mechanism provider) who adds dummy states into

the obfuscation mode needs to clearly know: 1) what states have been used in the normal operation mode and 2) the exact signals that drive the state transitions. Thus, the obfuscation mechanism provider is assumed to have sufficient knowledge of the original design. The goal of our method is to eliminate this assumption. We propose to obfuscate the gate-level netlist without the prior knowledge of the states in use and the signals triggering state transitions. We believe that the gate-level obfuscation is more attractive and flexible than the RTL obfuscation, as we can decouple the design obfuscation from the original design.

In our method, we assume that the obfuscation provider is able to learn the number of registers that represent the state binary bits by reading the gate-level netlist. This is reasonable as one can differentiate flip-flop cells from combinational logic gates on the netlist. We form the black hole states shown in Fig. 6(a) as follows. First, each state (i.e., $\{S3, S2, S1, S0, B3, B2, B1, B0\}$) in the new FSM is composed of all the original state bits $\{S3, S2, S1, S0\}$ and newly added flip-flops $\{B3, B2, B1, B0\}$. When the correct key sequence is applied, the newly added flip-flops $\{B3, B2, B1, B0\}$ are set to zero and thus the new states remain consistent with the FSM state before our state extension. This arrangement guarantees that the newly added flip-flops do not change the original function if the key is correct.

If the state of $\{B3, B2, B1, B0\}$ is not all zero due to a wrong key application, then the new state $\{S3, S2, S1, S0, B3, B2, B1, B0\}$ belongs to one of the black hole states in the cluster. The correct key can be hard wired in the netlist or saved in a trusted memory. The former one is less secure (but more hardware efficient) than the latter one at certain degree. Since the attacker can only view the gate-level netlist, the correct key and key comparison logic have emerged with other logic and thus it is not easy to be reverse engineered. Note, more flip-flops added for new states increase the difficulty of hardware tampering and meanwhile incur more hardware cost. The number of newly added flip-flops is a user's choice. In real design, only the obfuscation provider can differentiate *Bx* bit from *Sx* bit and the FSM state may not be ordered as $\{S3, S2, S1, S0, B3, B2, B1, B0\}$.

C. Dynamic State-Deflection in Black Hole Cluster

We further harden the FSM by creating dynamic transitions in black hole clusters. Different than the existing work [15], [18], [26], our state transition among the black hole states is not static and predefined. As highlighted in the obfuscation flow chart shown in Fig. 8, the dynamicity in the proposed state deflection is achieved by utilizing the secret *RotateFunc* and *Mapfunc* algorithms.

A *Mapfunc* function is used to deflect the original state bits from the correct state transition. The *Mapfunc* algorithm takes the newly added state bits and the wrong key sequence as an input to generate the *FlipReg* vector, which will selectively flip some bits in the FSM state. Each wrong key sequence will lead to a unique *FlipReg* vector. Fig. 9 describes an example in which a case statement defines the *Mapfunc* algorithm. Although the algorithm itself is statically

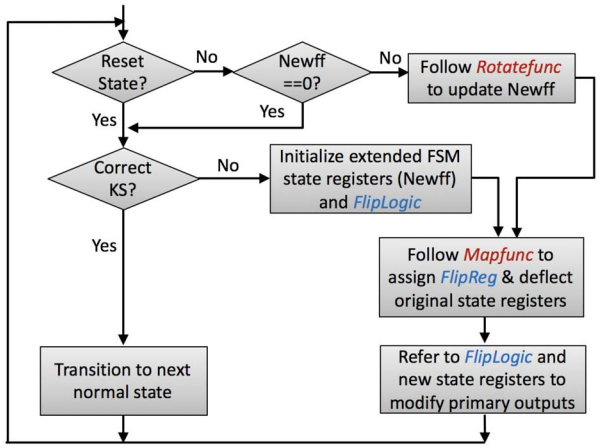


Fig. 8. Flow chart for the proposed dynamic state deflection.

```
**********************
*****MapFunc Example*****
**********************
Case({Newff, KS}):
{CORRECTKEY, zero-Newff}:
        FlipReg = 0;
{KS1, Newff1}:
        FlipReg =FlipReg1;
{KS2, Newff2}:
        FlipReg = FlipReg2;
{KS3, Newff3}:
        FlipReg = FlipReg3;
        ......
{KSn, Newffn}:
        FlipReg = FlipRegn;
default:
        FlipReg = FlipRegdefault;
```

Fig. 9. Pseudo code for a Mapfunc function.

defined, the *FlipReg* vector is assigned dynamically, depending on the KS and NewFF at runtime. We assume that an attacker may try different obfuscation keys to bypass the obfuscation mode. Because a different KS_j will lead to a different *FlipReg* vector, every brute-force attempt will mislead the attacker to draw a new conclusion, thus increasing the attackers time on reverse engineering attack. Moreover, the case condition further uses $Newff_j$, which is not statically assigned at the design time. Instead, the $Newff_j$ varies over the time due to the rotation function *RotateFunc*. The uncertainty of KS_j and $Newff_j$ makes our method achieve obfuscation dynamicity. Defenders can decide the number of KS cases for the *Mapfunc* function. The more KS cases, the stronger obfuscation strength. However, to save the cost, multiple wrong keys can be mapped to a default value. Default vector $FlipReg_{\text{default}}$ value should be chosen carefully to obtain the ideal RHD and state register autocorrelation (the details are discussed in Section V).

A detailed example of dynamic state deflection is shown in Fig. 10. If a wrong key sequence ($! = KS$) is used, the new state bits are preset to a nonzero vector (e.g., 4'b0001) and the original state bits remain whatever they are defined in the original design. After cycle 0, the newly added state bits are modified by a *RotateFunc* algorithm, which changes the vector $\{B3, B2, B1, B0\}$ to a new nonzero vector. In this example shown in Fig. 10, the *RotateFunc* is a circular

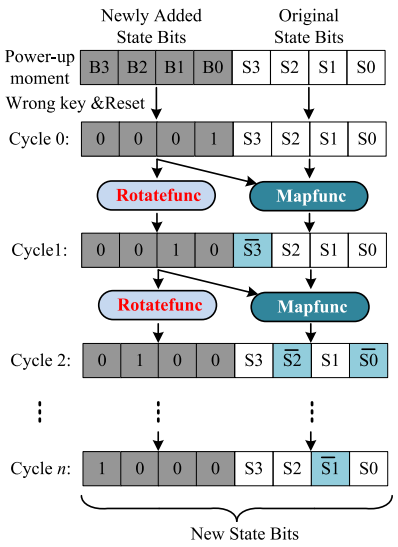


Fig. 10. Example of FSM state deflection induced by a wrong key sequence.

left-shift function, but it could be any algorithm defined by the obfuscation provider. When $\{B3, B2, B1, B0\}$ is 4'b0001, the $S3$ original state bit is flipped by a *FlipReg* of 4'b1000. As the entire FSM state registers are deflected to a new value, the FSM will generate a new *FlipReg* vector of 4'b1101 after cycle 1. Thus, the $S2$ and $S0$ bits in the original state are then flipped to its complementary value, and $S3$ is inverted back at the end of cycle 2. From this example we can see, our method inverts FSM state bits pseudo-randomly and selectively, rather than setting the original state bits to a fixed vector. As a result, the black hole state is not determined at the design time.

In parallel to switching the black hole state bits, our method applies the *FlipLogic* vector to alter the primary outputs. The process of *FlipLogic* vector generation in sequential circuits is similar to logic encryption methods for combinational circuits, except that our *FlipLogic* is *not* deterministic. The *FlipLogic* depends on the key sequence. Even if an attacker tries different key sequences to correlate the primary output and the locking key to retrieve the logic function, the observed correlation will not match to what is designed.

D. Proposed Design Flow

Our obfuscation design flow is summarized in Fig. 11. Six steps are needed in total.

- Step 1: Specify the newly added state bits, key bit width, and *FlipReg* width for *Mapfunc*.
- Step 2: Add new signals to the port list; step 3, compare the key sequence and insert the *Mapfunc* algorithm.
- Step 4: Follow the *RotateFunc* algorithm to switch the content of newly added state flip-flops.
- Step 5: Flip the original state bit if the corresponding *FlipReg* bit is true.
- Step 6: Flip the output bit selectively (depending on FlipLogic) if the current state belongs to one of the black hole states.

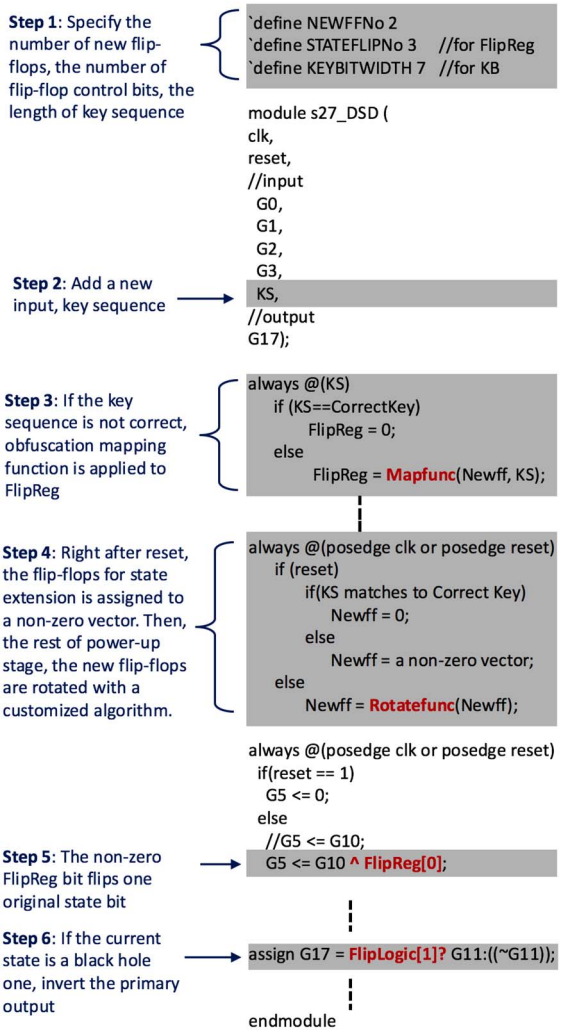


Fig. 11. Pseudo code for proposed obfuscation method.

Note, to make it readable, we use pseudo RTL description to demonstrate the obfuscation process. After logic synthesis, the obfuscation logic will not be different from regular logic circuit.

V. Experimental Results

A. Experimental Setup

To assess the capability against reverse engineering attacks, we applied three representable obfuscation approaches—DUP [14], ISO [26], and HARPOON [15]—and our method to the generic gate-level netlist of ISCAS'89 benchmark circuits [28]. The obfuscated Verilog source code files were synthesized in Synopsys Design Vision using a TSMC 65 nm CMOS technology library. The test benches for the synthesized netlists were generated by Synopsy's TetraMAX. TetraMAX is an automatic test pattern generation tool that generates high quality random test patterns for different types of designs. Moreover, TetraMAX delivers high test coverage on a wide range of design styles. Verilog simulations were conducted in Cadence NClaunch and the code coverage analyses were performed with the Cadence ICCR tool. To be fair, the length of

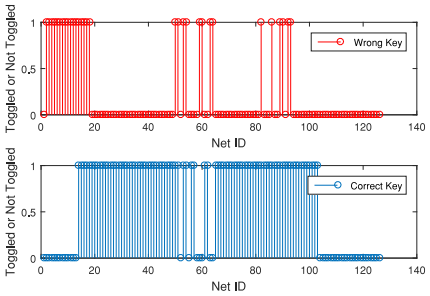


Fig. 12. DUP net toggle in s298.

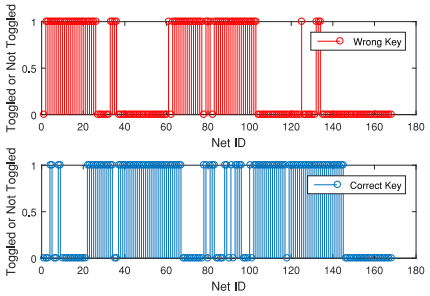


Fig. 13. ISO net toggle in s298.

the obfuscation key applied in each method is the same for a given benchmark circuits.

B. Hardening Capability Against Circuit Switching Activity Analysis

The goal of state obfuscation is to inhibit reverse engineers from retrieving the circuitry used for the true operation functions. Existing works have not extensively discussed the hardening efficiency of their methods in the scenario that an attacker could exploit the EDA code analysis tool (rather than completely use brute force attempts) to identify the original circuit. In this paper, we fill in this gap and study the net toggle activities and code coverage of the circuits hardened with different obfuscation methods.

1) Complementary Net Toggle Activities: State obfuscation with a key sequence blocks the gateway to reach the true state transition graph. In the existing methods, the use of a wrong key will lead the FSM stuck at either one obfuscation state or one inner loop in specified obfuscation mode. Thus, the majority of the true circuit under protection will not switch. If the attacker compares the net toggle activities with different keys, he/she can sort out the un-toggled nets and recognize that those nets probably used for the true operational circuit. To avoid leaking this type of information, we propose the dynamic state-deflection obfuscation method to reduce the complementary degree of the net toggle activities between the scenarios of the correct key and the wrong keys. Net toggle activity is the metric to assess whether a specific net has switched from 0(1) to 1(0) during the simulation. The Cadence ICCR tool provides the statistic results for net toggle activities.

Fig. 12 shows the net toggle activities for the s298 circuit protected with the DUP obfuscation method. In the scenarios of wrong keys, multiple key sequences are automatically generated by TetraMAX. As can be seen, if wrong key sequences are applied, the majority nets in the DUP netlist do not toggle. In contrast, those un-toggled nets indeed switch if the correct key sequence is used. Thus, the net toggle activities for the case with/without the correct key sequence are nearly complementary. Due to the transitions in the obfuscation mode, the number of toggled nets from the ISO and HARPOON netlist is more than that of the DUP netlist when wrong keys are applied. However, from Figs. 13 and 14, we still can observe the similar complementary toggle activities.

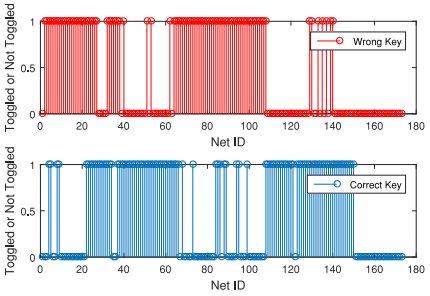


Fig. 14. HARPOON net toggle in s298.

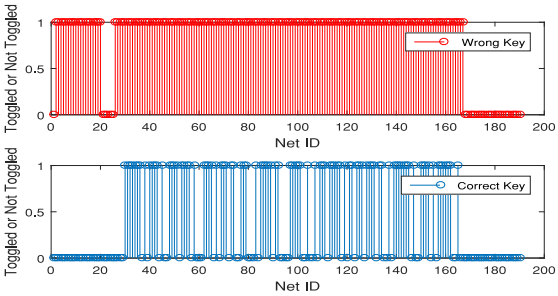


Fig. 15. Proposed method net toggle in s298.

The proposed obfuscation method examines the key sequence at every state transition and deflects the state to a black hole state. In our method, the FSM does not remain at a small inner loop. Instead, our FSM further changes over time within the black hole cluster. The FSM state in the black hole cluster will continue to reinforce the state registers and outputs to switch. Consequently, our method obtains the nearly same net toggle activities for the case of with/without the correct key sequence, as shown in Fig. 15. Note, as the test bench generated by TetraMAX cannot reach 100% test coverage, some logic in the design module are not toggled for both wrong and correct key cases.

We define the complementary degree of the net toggle activities for the correct and wrong keys as the expression shown in Eq. (9).

$$P_{\text{complement}} = \frac{N_{\text{different toggled nets}}}{N_{\text{total nets in netlist}}} \tag{9}$$

A lower complementary degree means that less information will be leaked through code analysis in EDA tools. As shown in Fig. 16, the proposed method yields the lowest complementary percentage than all the other methods. Experiments performed on the ISCAS benchmark circuits show that our

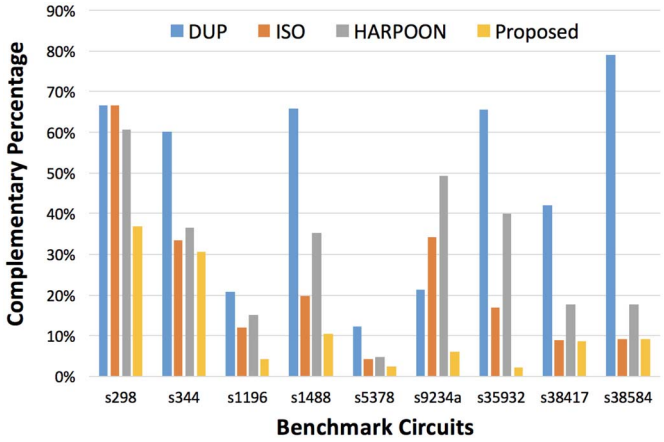


Fig. 16. Complementary percentages of the net toggling in different benchmark circuits.

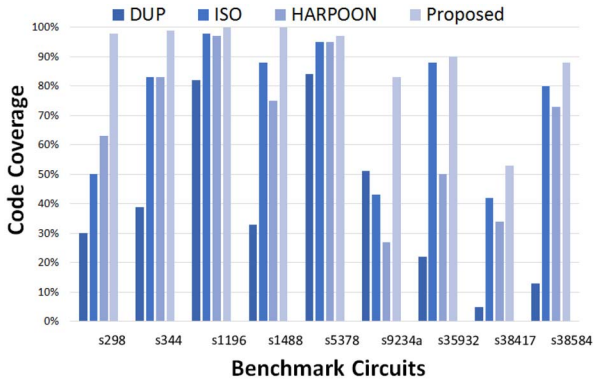


Fig. 17. Code coverages for the wrong key scenarios.

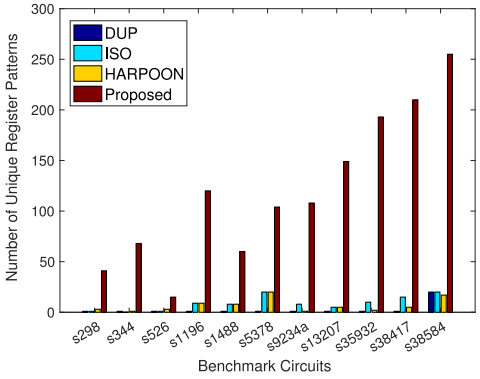


Fig. 18. Number of unique register patterns in different benchmark circuits.

method can reduce the complementary percentage by up to 77%, 85%, and 90% than that of DUP, ISO, and HARPOON, respectively.

2) Code Coverage: We use code coverage as another metric to compare the hardness achieved by different obfuscation methods. If the obfuscated netlist with wrong key sequences obtains a higher code coverage, this equivalently indicates that this method will yield less net toggle complementary percentage. Our method tightly blends the dummy states with the original states, so that it is difficult for attackers to identify the dummy states through code coverage analysis. For simulation of a gate-level netlist, only toggle coverage can be measured. Branch and FSM coverages are not eligible in the code coverage analysis tool. In the method [15], a wrong key sequence will lead the FSM to eventually enter in an isolation mode; thus, the FSM will freeze and the code coverage will be limited. Same reason applies to the method [14], where the FSM will stuck in the same state. The use of black hole states will lock the chip operation. But, if the attacker has the soft IP or firm IP running in a code analysis tool, the un-toggled statements will reveal the information relevant to black hole states.

Our method keeps the state transition going all the time. In contrast, other methods, either stuck at wrong states, or enter an internal small obfuscation states. Hence, as shown in Fig. 17, the proposed method always achieves the highest code coverage among the methods under comparison. Our method improves the code coverage by up to 56% over the most efficient obfuscation methods if a wrong key sequence is applied.

C. State Register Correlation

1) Number of FSM Register Patterns in Obfuscation Mode: In the previous works [15], [18], [26], the state registers either remain same or switch within a loop of different patterns if a wrong key sequence is applied. Both of the scenarios will facilitate attackers to identify the states in the obfuscation mode. Then, he/she can add new logic to overwrite the state registers and possibly skip the livelock of the obfuscation state transition. In contrast, our method utilizes *Mapfunc* function to "pseudo-randomly" change the state register values. An attacker may know some *Mapfunc* function is applied in the obfuscation mode, but they do not have knowledge of the exact *Mapfunc* in use. Hence, it is difficult for the attacker to effectively manipulate the FSM registers.

To validate the impact of secret *Mapfunc* function, we performed experiments to gather the state register values for different wrong keys. As shown in Fig. 18, our method is the one that generates the largest number of unique state patterns over the other methods. For a small circuit like s298, the proposed method produces 41, 40, and 38 more unique state patterns than DUP, ISO, and HARPOON, respectively. For a large circuit like s35932, the number of unique state patterns increases to 107 (versus DUP), 100 (versus ISO), and 98 (versus HARPOON). This observation confirms that our method not only obfuscate the state transition, but also can resist the register pattern analysis and register overwrite attack.

2) Repeatedness of FSM Register Patterns: If the FSM register contents are repeated in a predictable way in the scenarios of wrong key sequences, this information may give attackers a clue to correlate the key sequence with the register pattern. In the ideal case, the FSM register content should randomly change without a clear pattern even in the obfuscation mode. Unfortunately, due to the constraint on hardware cost, once the FSM enters the obfuscation mode, the FSM register pattern may be repeated after a while. We use the autocorrelation coefficient as a metric to evaluate the repeatedness

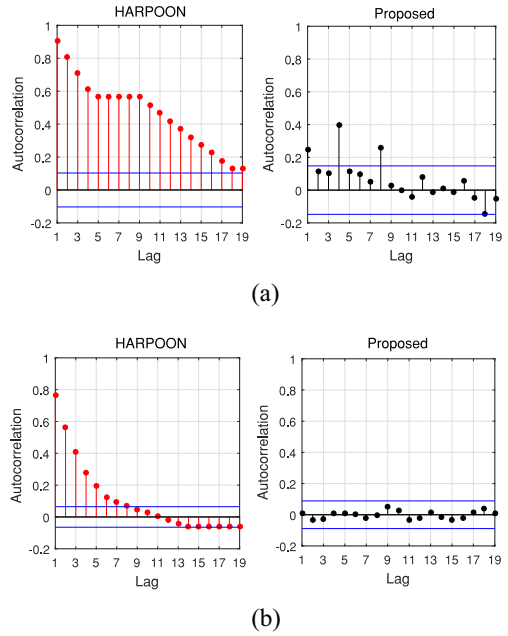


Fig. 19. Autocorrelation coefficient of the FSM state register. (a) s298 and (b) s5378. Lag represents the length of the repeated pattern under examination. The blue line in each subplot is the 95% confidence interval.

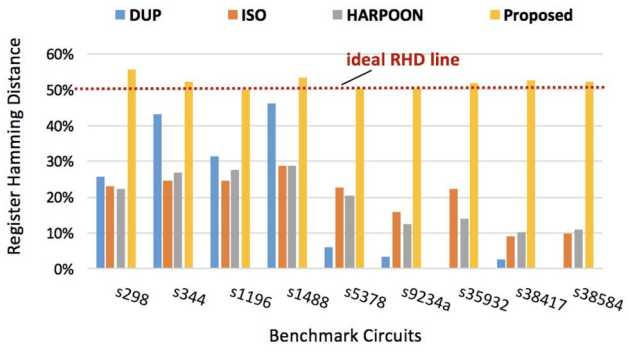


Fig. 20. RHD in different benchmark circuits. Note, the ideal HD is shown by red dotted line.

of FSM register pattern in the case of FSM using wrong key sequences.

Fig. 19 shows the autocorrelation coefficients of HARPOON and our method in s298 and s5378 benchmark circuits. As it can be seen, the autocorrelation of our method is nearly within the boundaries; while the HARPOON method yields a autocorrelation that noticeably exceeds the boundaries. This means, if a wrong key sequence is applied to the netlist obfuscated by HARPOON, attackers may exploit the repeated register patterns to bypass the obfuscation mode without using a correct key.

3) Register Hamming Distance: We use TetraMAX to generate the test bench for each obfuscated circuit. The process of RHD computation is same as the setup depicted in Fig. 2(b). All RHD of different methods were collected from the simulation for average RHD comparison. As shown in Fig. 20, our method is able to accomplish an RHD of approximately 50% (i.e., the best RHD to resist reverse engineering attack [24]) in all benchmark circuits under consideration. In contrast, the

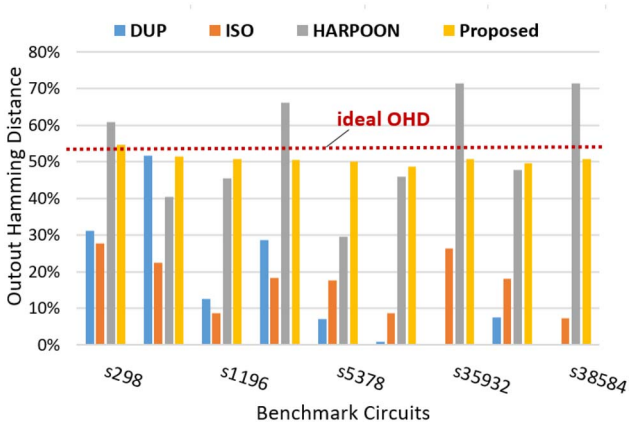


Fig. 21. OHD in different benchmark circuits (ideal HD is shown by red dotted line).

TABLE I
AVERAGE STANDARD DEVIATION OF MEASURED RHD AND OHD IN FIGS. 20 AND 21

	DUP	ISO	HARPOON	Proposed
Avg. Std. RHD	0.1034	0.1977	0.1904	0.1788
Avg. Std. OHD	0.0776	0.1815	0.3576	0.0870

RHD of other methods cannot reach 50%. As the size of the benchmark circuit increases, the RHD of other methods can be less than 10%. As explained in Section IV-C, our method keeps on transitioning into black hole cluster. While in DUP, ISO, and HARPOON obfuscation methods, the FSM will not transition and stuck into a specific state or loop which in turn prohibits the diverse transition activities.

D. Circuit Output Correlation

In this section, we use OHD as a metric to assess the circuit output correlation with the secret key. The detailed OHD calculation is discussed in Section III-A. As shown in Fig. 21, the proposed method achieves the desirable HD of almost 50% in all benchmark circuits, superior to DUP, ISO, and HARPOON. In some scenarios, e.g., s9234a and s38417, the HARPOON obfuscation method can approximately obtain 50% HD. However, the average standard deviation as shown in Table I for HARPOON is quite high compared with the proposed method. Comparatively, high standard deviation (std.) means the HD for individual test cases varies in a wide range. The OHD of DUP and ISO obfuscation methods is low due to the stuck states in obfuscation mode.

E. Hardware Overhead

1) Area: The silicon area cost of the benchmark circuits with different obfuscation methods is compared in Table II. The original circuit is the one without any protection mechanism. In DUP, we randomly chose one state as the stuck state for any wrong key. For ISO, we implemented an isolation FSM of ten additional states before the true functional mode. In HARPOON, we added an authentication FSM of six

TABLE II
HARDWARE AREA COST IN A 65 nm TECHNOLOGY (UNIT: μm^2)

Circuit	Original	DUP	ISO	HARPOON	Proposed
s298	223.6	283.3	375.5	403.92	345.9
s344	250.2	316.1	405.0	445.32	386.6
s1196	654.1	737.3	817.6	873.72	825.1
s1488	627.5	657.7	754.6	839.16	779.4
s5378	2580.1	2599.6	3112.9	3298.68	2892.6
s9234a	2158.9	3080.2	2716.6	2840.76	2395.8
s35932	23564.9	28824.1	28887.1	29173.3	23961.6
s38584	18523.1	22149.7	22244.4	22995.4	21835.1
s38417	22008.2	26767.8	26829.4	26961.5	22336.6
AAI	0%	20%	37%	40%	23%

AAI: average area increase over the original design

TABLE III
CRITICAL DELAY IN A 65 nm TECHNOLOGY (UNIT: NS)

Circuit	Original	DUP	ISO	HARPOON	Proposed
s298	0.40	0.43	0.41	0.42	0.50
s344	0.41	0.42	0.40	0.71	0.67
s1196	0.55	0.57	0.56	0.54	0.62
s1488	0.73	0.69	0.69	0.65	0.66
s5378	0.55	0.55	0.65	0.55	0.57
s9234a	0.80	0.77	0.81	0.85	0.80
s35932	0.43	0.43	0.44	0.64	0.59
s38417	1.15	1.15	1.11	1.10	1.22
s38584	0.87	0.87	0.88	0.88	1.16
ADI	0%	0.5%	2%	13%	19%

ADI: average delay increase over the original design

states and an obfuscation FSM of ten states before the normal mode. The same key size of 12-bit was applied to all the obfuscation methods. Our method does not need to examine the arrival of every authentication/obfuscation state like DUP, ISO, and HARPOON do. Instead, our method has a clear indication from the extended FSM register to differentiate a normal state from a black hole state. Moreover, our method utilizes dynamic state deflection with simple flipping logic, rather than modifying the state to a predetermined new state. Thus, our method reduces the area by 14% and 17% over ISO and HARPOON, respectively. Compared to DUP, our method consumes more area but provides better IP hardening performance as discussed in the previous sections. The design overhead in our method is caused by: 1) the addition of combinational logic for the implementation of the function *Mapfunc* and 2) few sequential elements for the implementation of the function *RotateFunc*.

2) Delay: We evaluated the critical delay of different benchmark circuits in Table III. Our method induces an average delay increase by 19% over the original circuits. Compared to HARPOON, the proposed obfuscation causes 6% more average delay overhead. Note that, the worst-case delay for each obfuscated circuit occurs when a wrong key is applied to the circuit. The circuit delay for the correct key scenario is less that what is reported in Table III.

3) Power: According to the critical delay reported in Table III, we set up the clock period as 2 ns for all the benchmark circuits. We used Synopsys Design Vision to report the total power. The power consumption of different methods applied in ISCAS benchmark circuit is compared in Table IV. Due to the key sequence checking and the FSM state deflection at every state transition cycle, our method, on average, consumes 7% and 5% more power than ISO and HARPOON, respectively. Again, the power of DUP is 14% less than our method, but losing obfuscation strength.

TABLE IV
POWER CONSUMPTION IN A 65 nm TECHNOLOGY (UNIT: mW)

Circuit	Original	DUP	ISO	HARPOON	Proposed
s298	0.0659	0.0576	0.0802	0.0824	0.092349
s344	0.0742	0.0652	0.0882	0.0903	0.0929
s1196	0.1073	0.1021	0.1252	0.1287	0.1162
s1488	0.0509	0.0359	0.0581	0.0630	0.0577
s5378	0.8853	0.8547	0.7346	0.7442	0.8834
s9234a	0.6380	0.8553	0.6620	0.6264	0.6718
s35932	8.0220	7.9409	7.2642	7.2956	7.9780
s38584	6.7421	6.6117	6.3806	6.3883	6.9262
s38417	5.3477	5.3479	4.9328	4.9430	5.8367
API	0%	-3%*	4%	6%	11%

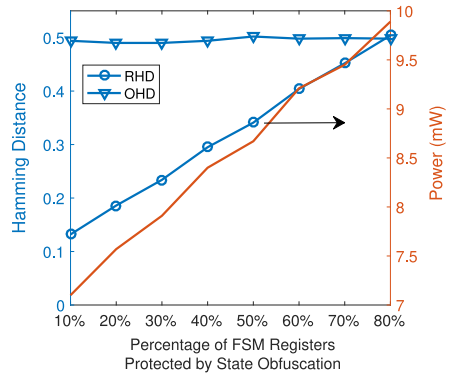
API: average power increase over the original design
*: the negative average power overhead is because of the inactivities of the circuit with a wrong key. If a large key size is selected for the experiment, the power overhead will increase.

F. Design Space Exploration for Proposed Method

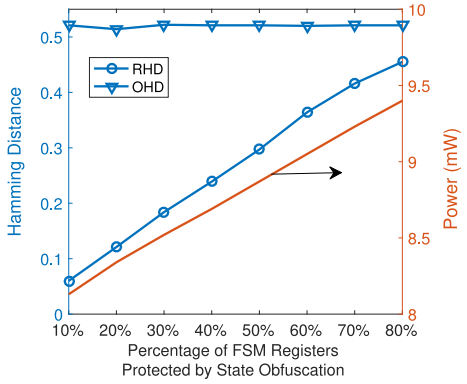
1) Impact of Mapfunc Parameter on Netlist Hardening: In the *Mapfunc* function shown in Fig. 9, the *FlipReg* vector controls which FSM registers to be inverted. If we choose a larger size for the *FlipReg* vector, more register bits will be flipped when the FSM is in one of the black hole states. Thus, a larger number of different register contents and a longer period for the repeated register patterns will be achieved. However, the use of a larger *FlipReg* vector will lead to more hardware overhead. As each circuit has different number of FSM registers, we varied the percentage of FSM registers protected (via increasing the width of the *FlipReg* vector) by the proposed obfuscation method to examine OHD and RHD, as well as power consumption for two large benchmark circuits s35932 and s35932. As shown in Fig. 22(a) and (b), in general, more FSM registers obfuscated will make the RHD closer to the ideal 50% HD. Accordingly, more power is consumed as a larger *FlipReg* vector leads more FSM register switching. This conclusion is confirmed by both circuits in this experiment. In s38417 and s35932, if one prefers to achieve an RHD close to 50%, then 80% and 90%, respectively, FSM registers should be protected by our method. Note, the mentioned power overhead is due to the wrong key application, which happened to be true in our case study. In our method if a correct key is applied to the obfuscated circuit, the power consumption is expected to be less. The interesting observation from Fig. 22(a) and (b) is that the OHD is not affected significantly by the percentage of FSM registers under protection. This is because the *RotateFunc* in our method flips the primary outputs based on the *FlipLogic* vector, independent to the *FlipReg* vector. Hence, if the attacker has only access to the primary inputs

TABLE V
DESIGN OVERHEAD FOR DIFFERENT KEY SIZES USED IN PROPOSED METHOD. AREA UNIT: μm^2, POWER UNIT: mW, AND DELAY UNIT: NS

	Key size=16			Key size=32			Key size=64		
Benchmark	Area	Power	Delay	Area	Power	Delay	Area	Power	Delay
s298	360.00	0.0912	0.51	380.16	0.0915	0.52	444.24	0.0928	0.50
s344	417.24	0.1034	0.64	435.96	0.1023	0.64	488.88	0.1048	0.65
s1196	808.56	0.1161	0.62	849.24	0.1164	0.62	905.40	0.1168	0.62
s1488	778.68	0.0574	0.68	815.76	0.0575	0.66	875.16	0.0582	0.66
s5378	2888.63	0.8850	0.57	2934.72	0.8836	0.57	2996.99	0.8830	0.57
s9234a	2375.64	0.6710	0.82	2394.72	0.6711	0.80	2449.44	0.6709	0.80
s35932	23828.76	7.9658	0.59	23843.16	7.9364	0.59	23904.36	7.9658	0.59
s38417	21971.88	6.9213	1.22	21997.80	6.9275	1.22	22056.84	6.9095	1.22
s38584	21225.24	5.8175	1.16	21246.12	5.8587	1.16	21304.08	5.8390	1.16



(a)



(b)

Fig. 22. Power and HDs tradeoff in (a) s38417 and (b) s35932.

and outputs, the proposed method works well with minimum overhead.

2) Impact of Key Size on Hardware and Performance Overhead: To achieve a higher level of security assurance of the system, a larger key width is preferred [29]. We repeated our previous experiments for Tables II-IV by varying the key width. Table V shows the hardware area, power, and critical delay overhead for resynthesized benchmark circuits. On an average, the total area for key size of 32 and 64 is 2.5% and 8% more than that for key size of 16. The key size induced changes to power and delay by less than 1%. Hence, we believe that the impact of the key size on the hardware area, critical delay, and power is not significant for large circuits.

VI. CONCLUSION

Since more and more designs are outsourced to overseas foundries, hardware security and trust emerges as a serious concern. As hardware IP reusing is prevalent in embedded and many-core systems, the presence of reverse engineering attacks is now not only limited in malicious foundries but also in the IP integration stage, where advanced EDA tools may leak detailed insights of high-level design to facilitate reverse engineering attacks. Key-based obfuscation is one of the promising countermeasures to thwart the hardware attacks like reverse engineering and IP piracy. The existing RTL obfuscation methods require the prior knowledge of all the used and unused states and lack any protection on normal operation states. To address the aforementioned limitation, we propose a low-cost dynamic state-deflection method for gate-level netlist to protects all the normal states. FSM state registers and primary outputs are selectively altered in a way customized for different wrong key sequence at runtime. Simulations performed on various benchmark circuits prove that the proposed method successfully eliminates the complementary net toggle activities for the correct key and wrong keys, thus reducing the success rate for the attack exploiting code coverage analysis. Our method achieves 56% higher code coverage than other existing methods in the scenario of a wrong key sequence applied to the obfuscated design. The number of register and output patterns for the wrong key cases in our method is significantly larger than that in other methods. As the state and output flipping vectors are generated at runtime, rather than being statically assigned at design time, our method obtains the HD of outputs and state registers close to 50%, which is the ideal HD to resist reverse engineering attacks.

The limitation of this method is hardware cost. Our method increases the average area, the critical delay, and the total power consumption by 23%, 19%, and 11% over the original benchmark circuits, respectively. However, compared to the existing obfuscation methods, our overhead on power is below 10%. Our case study shows that we can leverage the hardware overhead with the size of key sequence and the parameters in state deflection functions. In this paper, the register and output flipping vectors are pseudo-randomly selected to achieve the best desirable HD. In future, we will focus on its optimization strategies to minimize the area and performance overheads while maintaining the security metrics unaffected. Moreover, we will implement the proposed method on system level design like crypto-system to validate the efficiency.

REFERENCES

[1] U. Guin *et al.*, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

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

[3] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Munich, Germany, Mar. 2008, pp. 1069–1074.

[4] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 51–63, Feb. 2012.

[5] (2012). *Intel's 22-nm Trigate Transistors Exposed*. [Online]. Available: http://www.chipworks.com/blog/technologyblog/2012/04/23/intels-22-nm-tri-gate-transistors-exposed

[6] J. Rajendran, O. Sinanoglu, and R. Karri, "VLSI testing based security metric for IC camouflaging," in *Proc. Int. Test Conf. (ITC)*, Anaheim, CA, USA, Sep. 2013, pp. 1–4.

[7] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2014, pp. 270–271.

[8] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. Comput. Commun. Security (CCS)*, Berlin, Germany, 2013, pp. 709–720.

[9] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[10] ExtremeTech. (Sep. 2012). *iPhone 5 A6 SoC Reverse Engineered, Reveals Rare Hand-Made Custom*. [Online]. Available: https://www.extremetech.com/computing/136749-iphone-5-a6-soc-reverse-engineered-reveals-rare-hand-made-custom-cpu-and-a-tri-core-gpu

[11] J. Rajendran, A. Ali, O. Sinanoglu, and R. Karri, "Belling the CAD: Toward security-centric electronic system design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1756–1769, Nov. 2015.

[12] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *Proc. Int. Workshop Cryptographic Hardw. Embedded Syst. (CHES)*, Lausanne, Switzerland, 2009, pp. 363–381.

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

[14] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2007, pp. 674–677.

[15] R. S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[16] R. S. Chakraborty and S. Bhunia, "RTL hardware IP protection using key-based control and data flow obfuscation," in *Proc. Int. Conf. VLSI Design*, Bengaluru, India, Jan. 2010, pp. 405–410.

[17] (2010). *Semantic Designs*. [Online]. Available: http://www.semdesigns.com

[18] R. S. Chakraborty and S. Bhunia, "Security through obscurity: An approach for protecting register transfer level hardware IP," in *Proc. Hardw.-Orient. Security Trust (HOST)*, Jul. 2009, pp. 96–99.

[19] B. Liu and B. Wang, "Reconfiguration-based VLSI design for security," *IEEE J. Emerg. Sel. Topic Circuits Syst.*, vol. 5, no. 1, pp. 98–108, Mar. 2015.

[20] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Hardw.-Orient. Security Trust (HOST)*, Washington, DC, USA, May 2015, pp. 137–143.

[21] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in *Proc. Cyber Security Inf. Intell. Res. Workshop (CSIIRW)*, Oak Ridge, TN, USA, 2013, Art. no. 8.

[22] J. Zhang, "A practical logic obfuscation technique for hardware security," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 3, pp. 1193–1197, Mar. 2016.

[23] S. M. Awan, S. Rashid, M. Gao, and G. Qu, "Security through obscurity: Integrated circuit obfuscation using don't care conditions," in *Proc. Int. Conf. Control Autom. Inf. Sci. (ICCAIS)*, Ansan, South Korea, Oct. 2016, pp. 64–69.

[24] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[25] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Cryptograph. Hardw. Embedded Syst. (CHES)*, vol. 9813. Santa Barbara, CA, USA, 2016, pp. 127–146.

[26] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan attacks using key-based design obfuscation," *J. Electron. Test.*, vol. 27, no. 6, pp. 767–785, 2011.

[27] J. Dofe, Y. Zhang, and Q. Yu, "DSD: A dynamic state-deflection method for gate-level netlist obfuscation," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Pittsburgh, PA, USA, Jul. 2016, pp. 565–570.

[28] *ISCAS Benchmark Circuits*. Accessed on Jan. 4, 2016. [Online]. Available: http://www.pld.ttu.ee/maksim/benchmarks/iscas89/verilog/

[29] A. Moh'd, Y. Jararweh, and L. Tawalbeh, "AES-512: 512-bit advanced encryption standard algorithm design and evaluation," in *Proc. Int. Conf. Inf. Assurance Security (IAS)*, Malacca City, Malaysia, Dec. 2011, pp. 292–297.

Jaya Dofe (S'14) is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of New Hampshire, Durham, NH, USA.

Her current research interests include hardware security which includes design obfuscation, side channel analysis of encryption algorithms, fault attack analysis, and emerging technologies with emphasis on hardware security and trust.

Dr. Dofe was a recipient of the Best Poster Award in IEEE ISVLSI 2016, and the College of Engineering and Physical Sciences Graduate Fellowship in 2015 from the University of New Hampshire.

Qiaoyan Yu (S'03–M'11) received the B.S. degree from the Xidian University of China, Xian, China in 2002, the M.S. degree from the Zhejiang University of China, Hangzhou, China, in 2005, and the Ph.D. degree in electrical and computer engineering from the University of Rochester, Rochester, NY, USA, in 2011.

She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of New Hampshire, Durham, NH, USA. Her current research interests include hardware security and trust, cyber-physical system, error control for networks-on-chip, fault-tolerance for many-core systems, and emerging nanoelectronics.

Dr. Yu was a recipient of the NSF CAREER Award in 2017. She has served on the technical program committees of HOST, Asian HOST, DFT, ASP-DAC, GLSVLSI, ISVLSI, and ISCAS and the editorial boards of *Integration, the VLSI Journal*, *Microelectronics Journal*, and the *Journal of Circuits, Systems, and Computers*.