A Glitch Key-Gate for Logic Locking

De-Xuan Ji, Hsiao-Yu Chiang, Chia-Chun Lin, Chia-Cheng Wu, Yung-Chih Chen, and Chun-Yao Wang

2019 32nd IEEE International System-on-Chip Conference (SOCC) 978-1-7281-3483-3/20/$31.00 ©2020 IEEE 10.1109/SOCC46988.2019.1570547988

Abstract—**Logic locking is a technique used for intellectual property protection. An effective attacking method based on satisfiability (SAT) algorithm, known as SAT attack, was proposed to decrypt an encrypted design successfully. To strengthen logic locking, this paper proposes a glitch-based logic locking method designed for sequential circuits. The proposed new schemes of key-gates can generate glitches, and use rising and falling transitions as key-inputs for the comprehensive logic locking. Experimental results show that the proposed glitch key-gate (GK) has high capability to be embedded in a set of IWLS2005 Benchmarks [22]. The cell area overhead in the designs encrypted with GKs are 10.68%, 12.22%, and 26.11% on average for encryptions with 8, 16, and 32 key-inputs, respectively, and the overhead can be reduced substantially when the GKs are combined with other logic locking methods.**

I. Introduction

Semiconductor industry is built on a large-scale supply chain associated with many different companies distributed worldwide due to globalization of integrated circuit (IC) design business. Design companies often pass their designs to the foundries for fabrications. If there exists an untrusted foundry, the designs could be threatened by intellectual property (IP) piracy and other improper usages. Thus, many different hardware security techniques have been proposed [8].

Among a variety of hardware security techniques, logic locking is an effective technique for hardware IP protection. The basic concept of logic locking is to hide the functionalities of circuits by inserting a number of key-gates that are controlled by a key-vector. A classic XOR/XNOR-based logic locking technique [9] is shown in Fig. 1. Fig. 1(a) is the original circuit and Figs. 1(b) and 1(c) are both the encrypted circuits by inserting two key-gates. In Fig. 1(b), two key-gates are inserted at the outputs of the circuit, and they work as buffers to keep the outputs as the same as the original ones. In addition, the XOR/XNOR based key-gates can be designed as inverters to encrypt the circuit as shown in Fig. 1(c). As a result, it is challenging for attackers to identify a buffer or an inverter that each XOR/XNOR-based key-gate works as, and the difficulty of decryption grows exponentially with the number of key-gates. Based on this concept of inserting key-gates and using a key-vector as a license, there were several logic locking techniques proposed in [1] [7] [9] [12] [13] [14] [17] recently.

However, these logic locking methods for the combinational block of designs faced a great challenge recently from *SAT attack* proposed in [11]. SAT attack is an attacking method based on Boolean satisfiability (SAT) algorithms. The attacking model assumes that attackers have two essentials in hand: (1) An encrypted netlist with key-gate insertion; (2) A functionally correct chip with the correct key-vector assigned to it. SAT attack constructs a miter-like circuit with two copies of the encrypted netlist to identify the differences between their primary outputs (POs). Note that both netlists share the same primary inputs (PIs) but their key-inputs are independent. Then SAT solvers will be called to compute their distinguishing input patterns (DIPs) iteratively. DIP is an input pattern that can make two different key-vectors of encrypted netlist generate different outputs. When a DIP is found by the SAT solver, the functionally correct chip will be evaluated for the correct output of the DIP. Then the SAT solver uses the correct relation between the DIP and its corresponding POs to make the SAT solver filter out incorrect key-vectors. The attack will be successful when there is no DIP found by the SAT solver, which means all the incorrect key-vectors are filtered out.

Due to the effective decryption from SAT attack, some SAT attack-resistant methods were proposed [13] [14]. The objectives of these methods were to drag out the time required in the process of SAT attack. SARLock [14] used the concept of Point-Function [10] to achieve the effect that only one incorrect key-vector will be filtered out by a single DIP. Besides, Anti-SAT [13] took advantage of the difference between the on-set and off-set sizes of a function to build the security structure. Then the structure weakened the corruptibility to POs by incorrect key-vectors for mitigating the DIP search in SAT attack. Thus, the required efforts from SAT solvers in these methods grow exponentially with the increase of key-input number.

However, these SAT attack-resistant methods have vulnerabilities to *removal attack* [15] [16], which is a method removing or bypassing security structures to restore the original functions. Hence, these SAT attack-resistant methods have to be integrated with other obfuscation methods to hide the security structures. On the other hand, these methods caused little differences between the POs of encrypted circuit assigned with incorrect key-vector and the POs of original circuit. Hence, they have to rely on other encryption techniques to increase the corruptibility of the incorrect key-vectors. Unfortunately, an attacking method [10] exploited the dependence on other encryption techniques to crack these SAT attack-resistant methods.

As logic locking on combinational block has encountered aforementioned dilemma, researchers turn to develop logic locking techniques for sequential part [12]. SAT-based attacking methods usually cannot well deal with transitional data, like delay or timing. Hence, a timing-based logic locking method with a Tunable Delay Key-gate (TDK) scheme was proposed in [12]. As shown in Figs. 2(a) and 2(b), a TDK consists of a functional key-gate, i.e., an XOR gate, and a Tunable Delay Buffer (TDB), and they are separately controlled by a functional-key k_1 and a delay-key k_2. When k_2 is incorrect, the delay of TDB will either be added into the timing path for violating the setup time constraints as shown in Fig. 2(c), or be taken away from the timing path for violating the hold time constraint as shown in Fig. 2(d). However, this scheme could also suffer from the removal attack. This is because although the removal of the TDB and the delay-key k_2 may violate the timing constraints, the netlist after this removal can be re-synthesized to fix the timing violations, then SAT attack can be applied further to decrypt it. As a result, the decrypted designs can be operated normally with little performance degradation.

In this paper, to deal with the vulnerabilities to SAT attack and

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 106-2221-E-007-111-MY3, MOST-106-2314-B-007-005, MOST 106-2622-8-007-015-TA, MOST 107-2622-8-007-014-TA, MOST 107-2221-E-155-046, and MOST 108-2221-E-155-047.

D.-X. Ji, H.-Y. Chiang, C.-C. Lin, C.-C. Wu, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C.

Y.-C. Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan 32003, R.O.C.

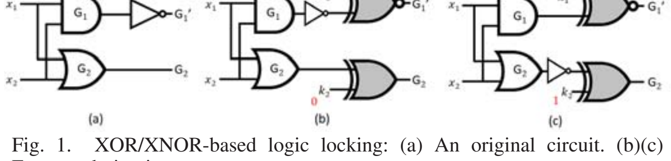


Fig. 1. XOR/XNOR-based logic locking: (a) An original circuit. (b)(c) Encrypted circuits.

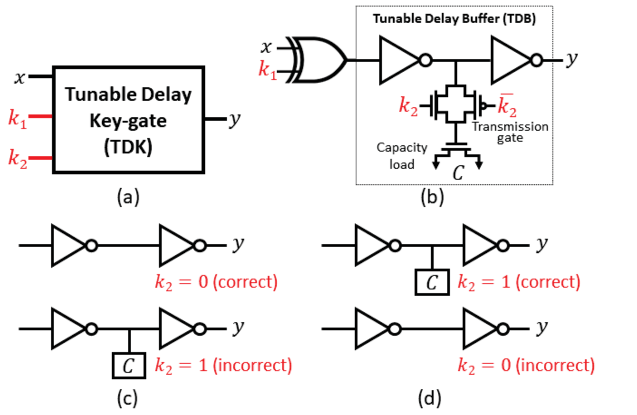
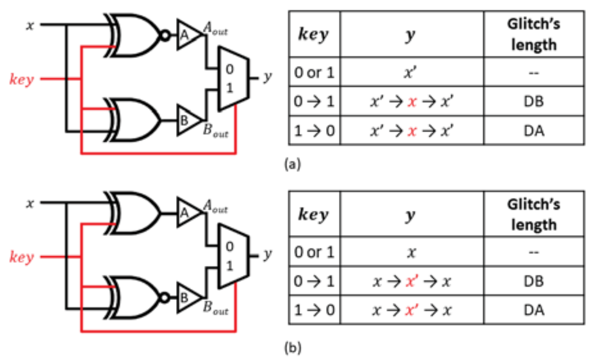


Fig. 2. A Tunable Delay Key-gate [12]: (a) Overview. (b) Implementation. (c) $k_2 = 0$ is correct. (d) $k_2 = 1$ is correct.



key	*y*	Glitch's length
0 or 1	x'	--
0→1	$x' \to x \to x'$	DB
1→0	$x' \to x \to x'$	DA

key	*y*	Glitch's length
0 or 1	x	--
0→1	$x \to x' \to x$	DB
1→0	$x \to x' \to x$	DA

Fig. 3. (a) The proposed GK. (b) The proposed GK with an opposite allocation of XOR and XNOR gates.

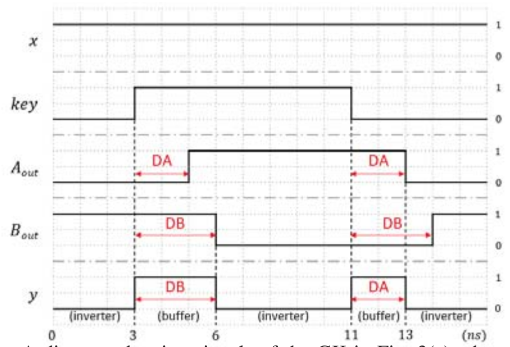


Fig. 4. A diagram showing signals of the GK in Fig. 3(a) when x is 1.

removal attack, we propose new schemes of Glitch Key-gate (GK) for logic locking in sequential circuits. Different from the previous SAT attack-resistant methods, which mitigate the efficiency of SAT attack, GK encryption tends to invalidate SAT attack from basis. GK encryption mainly focuses on generating glitches and using timing sensitivity of the glitches to encrypt designs. Since glitches are the phenomenon occurring within signal transitions, SAT-based attack will have troubles dealing with them. It is not only because SAT solver cannot handle transition signals, but also because the momentary value on the level of a glitch is invisible from the viewpoint of logic implication that SAT algorithm relies upon. Besides, the mentioned glitches generated by GKs can be designed to work as inverters or buffers at different timings to transmit data under different key-inputs upon the requests from designers. This feature ensures the corruptibility of the encryption with GKs.

The main contributions of this work are threefold:

- We propose new schemes of Glitch Key-gate (GK).
- The proposed GKs introduce the characteristic of glitch into logic locking to defend SAT attack and removal attack, and extend the key-inputs of key-gates to transitional signals.
- A thorough design flow incorporating existing commercial EDA tools for inserting GKs into designs is demonstrated.

II. Glitch Key-gate

In this section, we introduce the proposed Glitch Key-gates (GKs). A GK has two inputs, one is the signal to be encrypted and the other is the key-input. The key-input can be assigned as a constant 0, 1, a rising or a falling transition. When the key-input is a constant, the output of GK is *glitchless*. However, when the key-input is a rising or a falling transition, the GK generates a glitch at the output. The details about how our GKs work will be introduced in this section.

A. Working Mechanism of GK

The basic structures of GKs are shown in Fig. 3, where x is the input signal to be encrypted, key is the key input, y is the output, and A, B are delay elements with delay values DA and DB, respectively.

Let us explain the behavior of the proposed GKs. In Fig. 3(a), when the key-input is 0, the MUX will select the upper XNOR gate, and the XNOR gate works as an inverter. Similarly, when the key-input is 1, the MUX will select the lower XOR gate, and the XOR gate works as an inverter as well. Hence, we summarize that when the key-input is either assigned a constant 0 or 1, the output y is x'.

However, a glitch occurs at the output y when the key-input key is a rising or a falling transition due to the delay elements A and B with different delay values DA and DB. Fig. 4 is a timing diagram showing the internal signals of the GK in Fig. 3(a) under the input x = 1, DA = 2ns, and DB = 3ns. To simply the description about the behavior of the GK, here we first ignore gate delays, which will be discussed in Sec. IV. In Fig. 4, during the period of (0, 3) ns, the output $y = x' = 1' = 0$ since the GK works as an inverter as mentioned in the last paragraph. When a rising transition occurs at 3ns on the key-input key, A$_{out}$ and B$_{out}$ need 2ns and 3ns to update the values, respectively. However, the MUX immediately selects the B$_{out}$ and reports the old value of B$_{out}$. After the DB delay (3ns), the output of MUX obtains the updated value such that a glitch, from 3ns to 6ns with the length of DB, is generated. Similarly, the falling transition at 11ns also triggers a glitch with the length of DA.

Now we can see that the GK in Fig. 3(a) is able to work as an *inverter* when the key-input is a constant, and it can also generate a glitch to work as a *buffer* for a certain delay period when the key-input is a transitional signal. Besides, if we exchange the locations of the XNOR and XOR gates in the GK as shown in Fig. 3(b), the behavior of the GK will be completely opposite. That is, the GK will work as a *buffer* when the key-input is a constant; the generated glitch will work as an *inverter* when the key-input is transitional signal.

In summary, using the proposed GKs, designers can have a key-gate working as an inverter or a buffer under the correct key-input assigned at precise timing.

B. Key Generator for GK

As we discussed above, some behavior of GKs needs transitions to trigger. Hence, in this subsection, we discuss the implementation of generating these transitions. Since the proposed GKs are designed to encrypt sequential circuits, the behavior of a GK should be same in each clock cycle. If the predetermined behavior of a GK needs a transitional signal to trigger, a transitional signal generated and assigned to the key-input of the GK in every clock cycle is necessary.

We build a key generator (KEYGEN) for providing a transitional signal in each clock cycle. However, we need to shift the transitional signals with an adjustable delay for controlling the timing of triggering the transitional signals to key-input of GK. For the adjustable delay, we implement a simplified version of Adjustable Delay Buffer

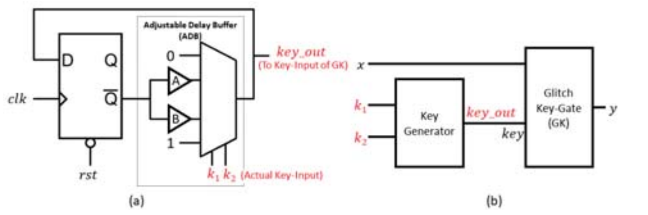


Fig. 5. (a) The structure of a KEYGEN. (b) The overall connection of KEYGEN with GK.

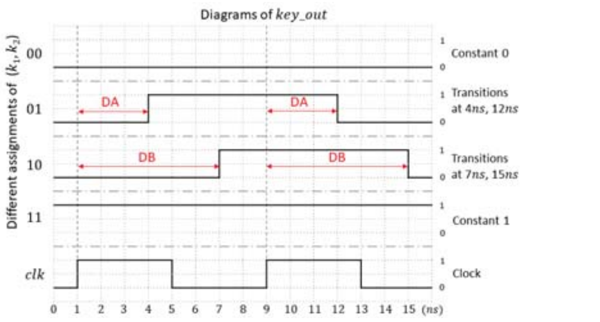


Fig. 6. A diagram showing the signal on key_out of KEYGEN.

(ADB. The simplified ADB is a MUX with different delays inserted at its inputs, then we can select different delays by different assignments to selection pins of the MUX.

Consisting of a D-type Flip-flop (FF) and an ADB, the structure of KEYGEN is illustrated in Fig. 5(a). The key_out of KEYGEN is connected to the key-input of the GK as shown in Fig. 5(b). A, B are delay elements in ADB with delay values DA and DB, respectively, and ($k1$, $k2$) selects one of four inputs of the MUX to key_out. Fig. 6 shows a timing diagram of the signal key_out of KEYGEN with DA=3*ns*, DB=6*ns*. The function of each assignment of ($k1$, $k2$) is transmitting constant 0, shifting a transitional signal with delay of DA, shifting a transitional signal with delay of DB, and transmitting constant 1 from the top to the bottom of Fig. 6.

III. Behavior of Glitches

When the length of a glitch is adjustable by designers, a glitch is not a waste anymore. As we discussed in Sec. II-A, the length of glitch generated by a GK is predetermined. Designers can exploit glitches to transmit correct data as conventional key-gates do at precise timing. Fig. 7 shows four scenarios that clock and output y of the GK work well without violating the setup time and the hold time constraints of a FF. The glitch in Figs. 7(a), 7(b), and 7(c) are the same glitch but triggered at different timings. When the transmitted data to the FF is encrypted on the level of the glitch, the starting point of the glitch should be arrival prior to the setup time of the FF, and the duration of the glitch should be long enough to meet the hold time constraint as shown in Fig. 7(a). Figs. 7(b) and 7(c) show the scenarios that the transmitted data to the FF is designed to be *not* on the level of the glitch, and indicate that the complete glitch should not interfere with the output. From Figs. 7(a), 7(b), and 7(c), we can know that with different timings to trigger, the glitch will cause different results. Fig. 7(d) shows a glitchless scenario that the GK is used to transmit data when the key input is a constant 0 or 1. In addition to these four scenarios, the glitch generated by the GK will violate the timing constraints.

IV. Timing Constraints and Design Flow

In this section, we introduce the insertion of GKs into designs from the viewpoint of implementation. In Sec. IV-A, we explain the timing constraints, and introduce different timings to assign transitions on the key-inputs for different behavior of GKs. In Sec. IV-B, we propose a design flow for achieving the proposed encryption.

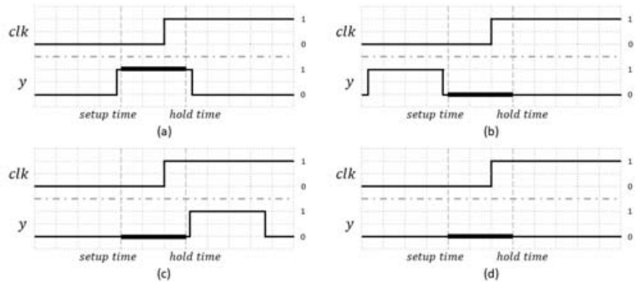


Fig. 7. Scenarios that values can be transmitted without timing violations.

A. Timing Constraints

First, let us review the timing constraints about the path delays from FF i to FF j in [12].

$$\begin{aligned} LB_{ij} &\equiv T^j_{hold} + T_j - T_i \\ UB_{ij} &\equiv T_{clk} + T_j - T_i - T^j_{set} \end{aligned} \tag{1}$$

The lower bound of the path delay LB_{ij} and the upper bound of the path delay UB_{ij} from FF i to FF j are summarized in Eq. (1), where T_i and T_j are the clock arrival times at FFs i and j, T_{clk} is the clock period, and T^j_{set} and T^j_{hold} are the setup time and the hold time of FF j. T_i and T_j may be different due to clock skew between FFs i and j. For example, assume that LB_{ij} = 5*ns*, UB_{ij} = 10*ns*, and original valid path delay is 7*ns*. If the path delay does not exceed 10*ns* after inserting a GK into the propagation path between FFs i and j, this insertion is valid. In this section, we will extend Eq. (1) to analyze the timing issue of the proposed GKs.

Fig. 8 is a simplified structure of the proposed GKs, where $PathA$ and $PathB$ represent the paths consisting of the XOR/XNOR gate and the inserted delay elements A and B, respectively.

The length of generated glitch (L_{glitch}) is determined by the summation of *PathA* or *PathB* delay (D_{Path}) and MUX delay (D_{MUX}), as shown in Eq. (2). This is because the transition on key needs to go through the *PathA* or *PathB* and the MUX to update the transition as we discussed in Sec. II-A.

$$L_{glitch} = D_{Path} + D_{MUX} \tag{2}$$

If we want to transmit data on the level of the glitch to FF j, L_{glitch} has to be designed as being greater than or equal to $(T^j_{set}+T^j_{hold})$ to satisfy the timing constraints of the FF j. However, after the arrival of the original signal, only a limited timing period we can use for inserting a GK without adjusting original clock cycle period. We will then discuss the constraints for the insertion, and it can be used to judge if a location is qualified for inserting the GK.

$T_{arrival}$ is the time needed by the original signal to arrive at the input x of GK. To obtain the correct value on the level of the glitch, we have to ensure that the data has arrived at the input of the MUX in GK before the transition occurs at the key-input. For example, in Fig. 4, when a rising transition on the key occurs at 3*ns*, the MUX selects B_{out} as its output y. However, y works as a buffer during the period of (3, 6) *ns* only when the value has already arrived at B_{out} before the rising transition on the key at 3ns. Hence, GK needs $D_{ready} = D_{PathB}$ to make the value from x become the value of the glitch and ready at B_{out} before the rising transition. Based on the same reason, the GK needs $D_{ready} = D_{PathA}$ to make the value from x become the value of the glitch and ready at A_{out} before the

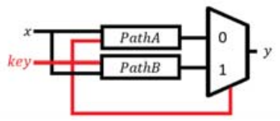


Fig. 8. A simplified structure of GK.

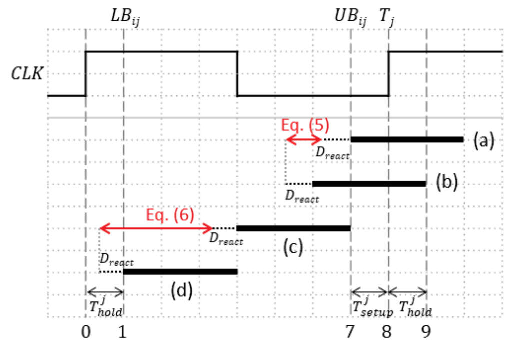


Fig. 9. Levels of four glitches at the boundaries of ranges for assigning transition on the key-input.

falling transition. Then $T_{arrival}+D_{ready}$ represents the delay needed for the success of glitch generation.

Next, we discuss D_{react} in Eq. (3). When we assign a transition to the key-input of GK, there is a latency (D_{react}) to generate the corresponding glitch at the output of GK. Since the key-input is also connected to the selection pin of the MUX, $D_{react} = D_{MUX}$.

$$LB_{ij} \leq T_{arrival} + D_{ready} + D_{react} \leq UB_{ij} \tag{3}$$

If we want to transmit the data on the level of the glitch as Fig. 7(a), we have to trigger glitch whose starting point should occur in the range between LB_{ij} and UB_{ij} for complying with the setup time constraints. Hence, Eq. (3) indicates that the summation of $T_{arrival}+D_{ready}$ (delay for generating the glitch), and D_{react} (delay to trigger the glitch) must be within the range between LB_{ij} and UB_{ij}.

$$LB_{ij} \leq T_{arrival} + max(D_{Path}) + D_{MUX} \leq UB_{ij} \tag{4}$$

However, if we want to transmit the data *not* on the level of glitch as shown in Figs. 7(b) and (c), we do not need to comply with the constraint described in Eq. (3). We only have to ensure that the insertion of GK will not violate the original signal. As a result, Eq. (4) indicates that the summation of $T_{arrival}$, $max(D_{Path})$, and D_{MUX}, where $max(D_{Path})$ is the largest one between delays of *PathA* and *PathB*, has to be within the range between LB_{ij} and UB_{ij}.

Now we can use Eq. (2) to design a glitch with a proper length, and use Eq. (3) and Eq. (4) to determine the position for inserting the GK. The last issue we need to address is the timing for triggering the glitch to determine the behavior of the GK.

In Fig. 9, assume that the clock cycle time is 8*ns*, the setup time and the hold time of FF j are both 1*ns*, and the clock arrival time at FF j (T_j) is 8*ns*. Hence, $UB_{ij} = 8-1 = 7ns$, $LB_{ij} = 1ns$ without clock skew. In Fig. 9, we only show levels of four glitches with length of 3*ns* at the boundaries of valid range for assigning transition on key-input. (a) and (b) are the glitches used to transmit data on the level of glitch. For glitch (a), the transition should be assigned before $UB_{ij} - D_{react}$ to satisfy the setup time constraint. For glitch (b), the transition should be assigned after $T_j + T^j_{hold} - L_{glitch} - D_{react}$ to satisfy the hold time constraint. Hence, we summarize the range of triggering in the first part of Eq. (5). The second part of Eq. (5) ensures that the timing assigning the transition on the key-input should be after $T_{arrival} + D_{ready}$ because we need this delay to generate the glitch as we discussed for Eq. (3).

$$\begin{aligned} &T_j + T^j_{hold} - L_{glitch} - D_{react} < T_{trigger} < UB_{ij} - D_{react} \\ &\text{and } T_{arrival} + D_{ready} \leq T_{trigger} \end{aligned} \tag{5}$$

(c) and (d) are the glitches used to transmit data *not* on the level of glitch. These glitches will not interfere the setup time and the hold time constraints of FF j. For glitch (c), the transition should be assigned before $UB_{ij} - L_{glitch} - D_{react}$ to ensure no interfering with the setup time constraint. For glitch (d), the transition should be assigned after $LB_{ij} - D_{react}$ to ensure no interfering with the hold time constraint. Hence, we summarize the range in Eq. (6).

$$LB_{ij} - D_{react} < T_{trigger} < UB_{ij} - L_{glitch} - D_{react} \tag{6}$$

Finally, if the transition on the key-input is assigned at the timing against Eq. (5) and Eq. (6), it will cause a timing violation. Having information about timing ranges, we can determine the behavior of the GK by assigning transitions on key-inputs in different timings.

B. Design Flow

We propose a design flow that cooperates with existing EDA tools for achieving the proposed encryption. First, the original design described in Verilog is synthesized by Design Compiler [18]. Then the netlist is translated to a layout by a Placement and Routing (P&R) tool - IC Compiler [19]. We also conduct timing analysis to obtain the slack of each FF with PrimeTime [20]. Having this timing information, we can determine feasible FF locations for inserting GKs under the same clock period of the original circuit.

After selecting positions for the encryption, we specify the desired behavior and structure of each GK, and insert GKs based on encryption strategies. Then we insert the delay elements and the corresponding KEYGEN of each GK into the netlist as shown in Figs. 3 and 5 by altering the netlist. The way we used for inserting delay elements is setting *design constraints* on the path that is for adding delays. That is, we re-synthesize the modified netlist, which is with GKs and the corresponding KEYGENs insertion, using design constraints. Design Compiler [18] maps delay elements from the library for satisfying the constraints. After the re-synthesis, we pass the design constraints to IC Compiler such that a layout satisfying the constraints is obtained by adjusting the mapping of cells.

To examine if the insertions of GKs and their corresponding KEYGENs work correctly, we further conduct timing analysis on the FFs that receive the output values of GKs after the P&R stage. If a GK and its corresponding KEYGEN are designed to generate glitches working as Figs. 7(a) and 7(c), EDA tools will report that the FF at the output of the GK is violated. This is because EDA tool sums up the delays we deliberately inserted, which exceeds the setup time of the FF. In fact, this delay is intentionally inserted for generating glitches. The delay will *not* cause timing violation in the actual circumstances on condition that we have ensured the starting point and the ending point of the glitch do not violate the timing constraints of the FF. To distinguish the ”true” timing violation and ”false” timing violation, we can further check the arrival time of each pin on the reported violated path. If there exists a true timing violation after checking, the GK and its generator will be removed. Then the design flow goes back to the feasible location selection stage and repeat the procedure until there is no true timing violation.

There may have some concerns about the change of critical paths due to the re-synthesis and the re-P&R. However, we can actively avoid choosing FFs on the critical paths as the feasible FF locations for inserting GKs. Furthermore, we adopt the same clock period for the synthesis and P&R of encrypted circuits, such that no delay overhead occurred on the critical paths.

V. Security Analysis of GK

A. SAT Attack

As we have discussed in Sec. 1, SAT attack is based on finding the DIP, which is an input pattern that can generate different outputs under assigning different key-vectors to two copies of the encrypted netlist. However, the output of our GK will be the same value under the key-input assigned with a constant 1 and 0. That means no possible DIPs occurring from our GK.

On the other hand, a glitch generated by our GK is a phenomenon occurred within a signal transition. However, SAT solvers only work on the stable logic value rather than logic transitions. Furthermore, the value transmitted on the level of the glitch is invisible from the viewpoint of logic implication. As a result, SAT attack cannot decrypt designs with GKs encryption.

B. Enhanced SAT Attack

In [3], the authors successfully used SAT solvers to generate patterns for detecting violations in rising and falling transitions. They proposed the Timed Characteristic Function (TCF), which is a function considering the timing features of the circuit, and used it to construct a SAT problem with the timing constraints. To test the *delay defects*, two patterns are needed to generate the transition for testing. Hence, they transformed the circuit into TCF for handling timing issue and into conjunctive normal form (CNF) for handling functional issue. Then they used TCF and CNF to compose a SAT problem, and used SAT solvers to generate the required patterns.

In our work, however, instead of being affected with delays, the output of the GK can be designed as a glitch deliberately, which is a phenomenon occurred within a signal transition. Both CNF and TCF cannot model the behavior of glitches because the value transmitted on the level of the glitch does not exist from the viewpoint of the functionality. In other words, we can never derive the value transmitted on the glitch through the CNF and TCF. As a result, the work in [3] can handle the delay issue of circuits, but not the glitch issue our work builds on.

C. Removal Attack

Removal attack is a method that removes or bypasses the security structures and restoring the original function of the circuit. It can effectively crack the *SAT attack-resistant* methods [13] [14]. The security structures of these SAT attack-resistant methods usually result in a phenomenon that the circuit may still have correct outputs even applying incorrect key-vectors. Such feature indicates that certain signals in the security structures have a high probability of being 0 or 1. With these skews in signal probabilities, attackers can locate the security structures or even derive their correct outputs [15] [16].

On the contrary, removal attack does not quite effectively decrypt conventional key-gates. This is because even these key-gates are located, attacked still has to guess whether a buffer or an inverter that each key-gate acts as. When the number of the key-gates grows, the number of attacking guesses will grow exponentially. Although our GK aims at defending against SAT attack, its strength of working as a buffer or an inverter can also defend the removal attack.

D. Enhanced Removal Attack

To demonstrate the strength of the proposed GKs, we also consider a new attacking method that combines SAT attack with removal attack. The scenario of this new attacking method is as follows:

1) Attackers locate the security structure in the encrypted circuit.
2) Each security structure is replaced by an XOR gate with a key-input, or is replaced by a MUX having multiple encryption behavior from the MUX's inputs and selected by key-inputs.
3) SAT attack is applied to crack the encryption.

This attacking method is effective to decrypt circuits when the security structures are located. To defend against this new attacking method, we can combine the withholding technique [5] [6] with our GK. As a result, we are able to prevent this new attacking method from modeling encryption behavior of GKs and from applying SAT attack to decrypt the encrypted circuits further.

The withholding is an encryption method storing the truth table of a subcircuit into a lookup table (LUT), and the LUT is not accessible externally. With the technique, the subcircuit is operated normally but its netlist is invisible to attackers. As the number of input increases, the size of LUT increases exponentially, and the possible combinations of the encrypted subcircuit even increase drastically. Fig. 10 is an example of our GK associated with the withholding technique by reusing an AND gate from the encrypted path in the original circuit, which is at the input x of the GK. We can encrypt the GK with more gates into LUT to elevate the security level of encryption. Hence, this new attacking method gets into trouble dealing with the huge number of the possible encryption behaviors.

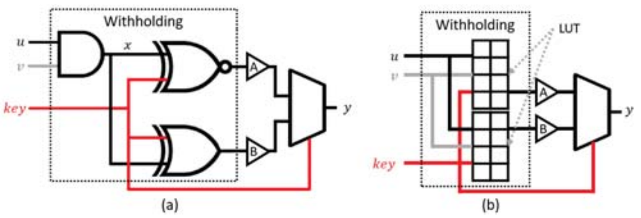


Fig. 10. (a) An AND gate encrypted by our GK. (b) An example of our GK associated with the withholding technique by reusing an AND gate.

TABLE I
THE NUMBER OF AVAILABLE FFS FOR ENCRYPTION.

Bench.	$\lvert$**Cell**$\rvert$	$\lvert$**FF**$\rvert$	$\lvert$**Ava. FF**$\rvert$	**Cov. (%)**	$\lvert$**Ava. FF**$\rvert$ **[4]**
s1238	341	18	16	88.89	4
s5378	775	163	104	63.80	89
s9324	613	145	74	51.03	59
s13207	901	330	185	56.06	36
s15850	447	134	58	43.28	51
s38417	5397	1564	1037	66.30	920
s38584	5304	1168	924	79.11	105
Avg.				64.07	

VI. EXPERIMENTAL RESULTS

To validate the effectiveness of the proposed GKs, we encrypt some sequential benchmarks from IWLS2005 Benchmark [22]. Each benchmark was first synthesized and optimized by Design Compiler (DC) with TSMC 0.13μm (CL013G) Process 1.2-Volt SAGE-X^{TM} Standard Cell Library, then is analyzed by PrimeTime for timing report. At last, we ran P&R with IC Compiler (ICC).

In our experiments, GKs inserted in the benchmarks are all designed to transmit values on the levels of glitches with length of $1ns$. This is because this scenario needs the strictest requirement. We kept the same clock period applied on the circuit before and after encryptions to show no timing overhead of the proposed GK. The ADB in each KEYGEN for GK was designed as Fig. 5 for generating transitions at different timings depending on its key-inputs.

Table I shows the experimental results about the number of available FFs for encryption in the benchmarks. Columns 1 to 3 are benchmark information after synthesis and optimization. Columns 4 and 5 show the number and the ratio of FFs (Cov.) that can be encrypted with GKs. According to Table I, 64.07% of FFs can be encrypted in optimized designs on average, indicating high applicability of the proposed encryption method. However, the area overhead might be too large when all of these available FFs are encrypted. The last column shows the number of FFs selected by an algorithm [4] from the available FFs. The algorithm aims at searching for a group of FFs fanouting to the same set of POs. Then we can select FFs to be encrypted among this group of FFs, which are able to defend against the Scan-based Attack [4] with a higher probability.

Hence, we selected some FFs among this set of available FFs and reported the overhead as shown in Table II. We inserted 4, 8, and 16 GKs separately on each benchmark if applicable, and each GK with its KEYGEN designed as Fig. 5 provided two key-inputs.

According to Table II, we can see that the overhead of cell number and cell area grows with the increase of the number of inserted GKs. This overhead is not small and is not proportional to the number of logic gates that each GK uses due to the following reasons:

1) We set design constraints on the paths to automatically insert delays into GK and its KEYGEN by DC and ICC.
2) The inserted delay elements, e.g., inverters or buffers are all from the cell library to composite a unique delay it needs.
3) The number of these delay elements is often larger than that of logic gates we used for GK and KEYGEN.

Hence, the delay elements for generating a unique delay value is far from being optimal currently. When the customized delay elements for GKs are available, the area overhead will be significantly reduced. This issue is out of the scope of current paper and will be our future work for a solid encryption practice.

We also ran SAT attack on these encrypted designs to evaluate the defense ability of our GK. Before SAT attack decrypts sequential circuits, it will first extract the combinational part from the original sequential circuit. Hence, we transformed the encrypted sequential circuits into combinational by treating the inputs and outputs of FFs as pseudo primary outputs and inputs, respectively. We removed the KEYGEN of each GK and treated its key-input as the key-input of the design. Then we apply SAT attack on the transformed circuits. Not surprisingly, the attack stopped at the first iteration of searching the DIP and reported *unsatisfiable*. The results demonstrate that our GK can prevent SAT attack from finding out the DIP. Without DIPs, SAT attack will be invalid.

We have explained that our GK has strong defense against the SAT attack. In general, the strength of encryption is highly positively correlational to the number of key-gates and key-inputs. Additionally, a GK also can act as an inverter or a buffer just like conventional key-gate does, and the behaviors provide a stronger corruptibility to POs than other SAT resistant methods. However, our GK may has a weakness when there are built-in self-test (BIST) structures such as scan-chain in the circuit. This is because GKs are used to encrypt the input of FFs, and scan-chain can be designed to test the paths between FFs in sequential circuits. When the test is applicable to the FFs on both sides of the path, the GK that works solely to encrypt the input of FF at the end of the path can provide only limited security.

To solve this possible problem, we can encrypt the circuit using our GK with other logic locking methods [1] [9]. That is, our GK defends against SAT attack for these additional logic locking methods, and these additional logic locking methods also help GK elevate the security level. Therefore, we conducted the experiments that inserting GKs and conventional XOR/XNOR [9] for encryption. Specifically, we insert XOR gates to the paths encrypted by GK to defend against the attack from BIST. We randomly used one half of the key-inputs to control the XOR key-gates, and the other half is for GKs. The column 3 and 4 in Table II shows the overhead comparison between the encryption with GK only, and the hybrid encryption with XOR gates and GKs. We can see that the overhead is reduced substantially by using XOR/GK encryption. From the result of this experiment and the discussion in Sec. V-D, we can conclude that our GK assists previous encryption methods to defend SAT attack. Besides, the previous encryption methods can also strengthen the security of GK.

VII. Conclusion

In this paper, we propose schemes of Glitch Key-gates, which strengthen logic locking against attacks, such as SAT attack and removal attacks. In addition to constant values, GKs also use transitions as key inputs. Experimental results show that the proposed method is able to insert a high percentage of key-gates for elevating the security. Experimental results also show that a hybrid encryption method combining GK with other logic locking methods strengthen the defense, but with a smaller area overhead. We can conclude that our GK can assist previous encryption methods to defend against SAT attack, and they can also strengthen the security of GK.

References

[1] X. Chen et al., "Low-Overhead Implementation of Logic Encryption Using Gate Replacement Techniques," *Int'l Symp. on Q. E. Design*, 2017.
[2] C.-H. Chou et al., "Skew Minimization With Low Power for Wide-Voltage-Range Multipower-Mode Designs," *IEEE Trans. on Very Large Scale Integration Systems*, 2016.
[3] S.-Y. Ho et al., "Automatic Test Pattern Generation for Delay Defects Using Timed Characteristic Functions," *Int'l Conf. on Computer Aided Design*, 2013.
[4] R. Karmaka et al., "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," arXiv:1801.04961 [cs.CR], 2018.
[5] S. Khaleghi et al., "IC piracy prevention via design withholding and entanglement," *Asia and South Pacific Design Automation Conf.*, 2015.
[6] B. Liu et al., "Embedded Reconfigurable Logic for ASIC Design Obfuscation Against Supply Chain Attacks," *Design, Automation and Test in Europe Conf. and Exhibition*, 2014.
[7] J. Rajendran et al., "Fault Analysis-Based Logic Encryption," *IEEE Trans. on Computers*, vol. 64, no. 2, pp. 410-424, 2013.
[8] M. Rostami et al., "A Primer on Hardware Security: Models, Methods, and Metrics," in *Proc. of the IEEE*, vol. 10, no. 8, pp. 1283-1295, 2014.
[9] J. A. Roy et al., "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30-38, 2010.
[10] K. Shamsi et al., "AppSAT: Approximately Deobfuscating Integrated Circuits," *Inte'l Symp. on Hardware Oriented Security Trust*, 2017.
[11] P. Subramanyan et al., "Evaluating the Security of Logic Encryption Algorithms," *Int'l Symp. on Hardware Oriented Security Trust*, 2015.
[12] Y. Xie et al., "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction," *Design Auto. Conf.*, 2017.
[13] Y. Xie et al., "Mitigating SAT Attack on Logic Locking," *Int'l Conf. on Cryptographic Hardware and Embedded Systems*, 2016.
[14] M. Yasin et al., "SARlock: SAT Attack Resistant Logic Locking," *Int'l Symp. on Hardware Oriented Security Trust*, 2016.
[15] M. Yasin et al., "Security Analysis of Anti-SAT," *Asia and South Pacific Design Automation Conf.*, 2016.
[16] M. Yasin et al., "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.
[17] J. Zhang, "A Practical Logic Obfuscation Technique for Hardware Security," *IEEE Trans. on Very Large Scale Integration Systems*, 2015.
[18] (April 20, 2018). Synopsys Design Compiler. [Online]. Available: https://www.synopsys.com
[19] (April 20, 2018). Synopsys IC Compiler. [Online]. Available: https://www.synopsys.com
[20] (April 20, 2018). Synopsys PrimeTime. [Online]. Available: https://www.synopsys.com
[21] (April 20, 2018). Synopsys Verdi. [Online]. Available: http://www.synopsys.com
[22] (April 20, 2018). IWLS2005 Benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

TABLE II
THE OVERHEAD OF OUR APPROACH AFTER INSERTING DIFFERENT NUMBERS OF GKS.

	4 GKs, 8 key-inputs		8 GKs, 16 key-inputs		16 GKs, 32 key-inputs		8 GKs + 16 XORs, 32 key-inputs	
Bench.	**|Cell| OH (%)**	**Area OH (%)**	**|Cell| OH (%)**	**Area OH (%)**	**|Cell| OH (%)**	**Area OH(%)**	**|Cell| OH (%)**	**Area OH(%)**
s1238	22.87	38.51	–	–	–	–	–	–
s5378	10.06	9.12	17.29	16.93	33.03	37.91	21.68	19.65
s9234	8.81	8.54	19.90	20.49	38.34	42.37	21.53	21.78
s13207	6.77	5.79	15.09	11.10	29.97	23.10	13.65	11.08
s15850	15.44	9.30	28.41	21.23	54.59	42.76	33.11	25.46
s38417	0.74	1.71	2.17	0.66	4.22	4.32	2.20	0.66
s38584	1.69	1.80	2.93	2.92	5.64	6.20	3.20	3.26
Avg.	9.48	10.68	14.30	12.22	27.63	26.11	15.9	13.65