Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction

Yang Xie
ECE Department
University of Maryland, College Park
Maryland 20742, USA
yangxie@umd.edu

Ankur Srivastava
ECE Department
University of Maryland, College Park
Maryland 20742, USA
ankurs@umd.edu

ABSTRACT

Logic locking is a technique that has been proposed to thwart IC counterfeiting and overproduction by untrusted foundry. Recently, the security of logic locking is threatened by a new attack called SAT attack, which can effectively decipher the correct key of most logic locking techniques. In this paper, we propose a new technique called delay locking to enhance the security of existing logic locking techniques. For delay locking, the key into a locked circuit not only determines its functionality, but also its timing profile. A functionality-correct but timing-incorrect key will result in timing violations and thus making the circuit malfunction.

1. INTRODUCTION

To access advanced semiconductor technology at a low cost, IC design companies are increasingly outsourcing their designs to offshore foundries. This trend of fabrication outsourcing creates a new concern on the security of outsourced designs. Without close monitoring and direct control, an untrusted foundry might pirate or counterfeit the design and fabricate extra unauthorized chips to gain profits [9]. These security threats (also known as supply chain attacks) pose a significant economic risk to most IC design companies.

Logic locking is a technique that has been proposed to counter the supply chain attacks by untrusted foundries. An overview of a locked circuit is shown in Fig. 1. During design time, a circuit is locked by inserting additional key-controlled logic gates (key-gates) into its combinational block. The key-inputs are connected to an on-chip memory and the locked circuit preserves the correct functionality only when a correct key is loaded into the on-chip memory after fabrication. This correct key is kept secret and is not accessible to the untrusted foundry. Besides, to prevent an adversary from probing internal signals of a running chip, a tamper-proof chip protection shall also be implemented. Recent years have seen various logic locking techniques based on different key-gate types and key-gate insertion algorithms [1, 3, 6, 7, 8, 9]. In general, the objective of these logic locking techniques is to increase the output corruptibility given an incorrect key, and to prevent effective key-learning attacks.

The security of logic locking is based on the assumption that the correct key is not accessible to the untrusted

DAC '17, June 18-22, 2017, Austin, TX, USA
© 2017 ACM. ISBN 978-1-4503-4927-7/17/06...$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062226

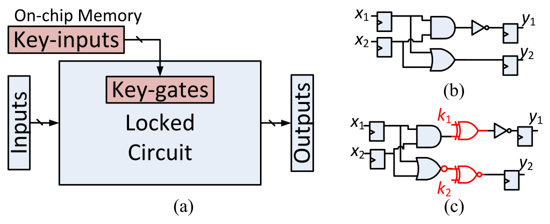


Figure 1: Logic locking techniques: (a) Overview; (b) An original netlist; (c) XOR/XNOR based logic locking

foundry. Otherwise, the foundry can use it to activate overproduced chips and sell them illegally. Various attacks have been proposed to learn the correct key. Rajendran *et al.* [7] proposed a fault-analysis attack which intends to automatically find a set of input patterns that can sensitize the correct key values to primary outputs (POs). Plaza *et al.* [6] proposed a randomized local key-searching algorithm to search the key that can satisfy a subset of correct input/output patterns. Recently, Subramanyan *et al.* [10] proposed a satisfiability-checking (SAT) based attack that can effectively break most logic locking techniques proposed in [1, 3, 7, 8, 9] within a few hours even for a reasonably large key-size. These emerging attacks undermine the security offered by existing logic locking techniques.

In this paper, we propose a new technique called delay locking to enhance the security of existing logic locking techniques against the emerging attacks. *The basic idea of our work is to make the circuit delay dependent on the key values. When enhanced with the delay locking, a key can not only control a locked circuit's functionality but also its timing profile.* Therefore, a correct key not only recovers the original combinational functionality, but also the correct timing profile that can satisfy pre-defined timing constraints. The combination of logic locking and delay locking is referred to as *delay+logic locking (DLL)*. With DLL, all previous attacks including the SAT attack are thwarted because they only focus on deciphering a "functionality-correct" key with respect to the combinational logics. However, this deciphered key might not be a "timing-correct" key. A "functionality-correct but timing-incorrect" key will result in timing violations and thus producing incorrect outputs, so the key cannot be used to unlock the overproduced chips. We summarize the contribution of this work as follows.

- A delay+logic locking (DLL) technique is propose to enhance the security of existing logic locking techniques to prevent IC counterfeiting and overproduction.

- A new type of key-gate called tunable delay key-gate (TDK) is introduced, which has two types of keys: functional-key and delay-key. The functional-key controls the TDK's functionality while the delay-key determines its gate delay.

- An overall DLL design flow is proposed, which allocates the new TDK gates and designs the timing constraints for simultaneous functional and delay obfuscation.

- Our proposed approach is fundamentally immune to previous attacks such as the SAT attack because these attacks only focus on deciphering the correct functional-key. Finding the correct delay-key can be formulated to be an instance of mixed-integer-linear-programming (MILP). However, necessary constraint relaxations to satisfy the linear formulation make it fail to find the correct delay-key (as discussed in Sec. 4.3). Hence our approach of simultaneous functional and delay obfuscation results in substantial security enhancements.

2. ATTACKS ON LOGIC LOCKING

2.1 Attack Model

Following the attack model assumed in [6, 7, 10], this paper assumes that the attacker is an untrusted foundry whose objective is to obtain the correct key of a locked circuit and use it to unlock overproduced chips. The malicious foundry has access to the following two components:

1. A locked gate-level netlist, which can be obtained by reverse-engineering the layout file of the locked circuit provided by the designer.

2. An activated functional chip, which can be obtained from an open market or an IC testing facility. This chip can be used to observe a set of correct input/output pairs as a black box.

Fig. 2 shows an attack flow against logic locking. During fabrication time, the foundry firstly reverse-engineer the layout file of the locked circuit to obtain a locked gate-level netlist (component 1) using state-of-the-art reverse-engineering technique. At the same time, he utilizes the already built masks to overproduce extra copies of locked chips, which await to be unlocked when the correct key is learned. After fabrication, the attacker obtains an activated chip (component 2) from the open market and intends to learn the correct key based on the above two components. After obtaining the correct key, the attacker can unlock the overproduced locked chips and sell them into the market.

2.2 Existing Attack Algorithms

Various attack algorithms have been proposed to learn the correct key. Rajendran *et al.* [7] proposed a fault-analysis attack which tried to automatically find a set of input patterns that can sensitize the correct key values to POs. Plaza *et al.* [6] proposed a randomized local key-searching algorithm with restart to search the correct key that can satisfy a subset of correct input/output responses. Corresponding countermeasures to these attacks have also been proposed in their work [6, 7].

Recently, Subramanyan *et al.* [10] proposed a new attack called SAT attack. The insight of SAT attack is to identify a small number of input/output pairs (called distinguishing input/output pairs) that can be used to prune out incorrect keys. The process of finding such input/output pairs is iteratively formalized as a sequence of SAT formulas that can be solved by state-of-the-art SAT solvers. It has been shown that most logic locking techniques proposed in [1, 3, 7, 8, 9] can be deciphered within a few hours even for a reasonably large key-size. Countermeasures have been proposed to mitigate the SAT attack [12, 13, 15]. Basically, they proposed

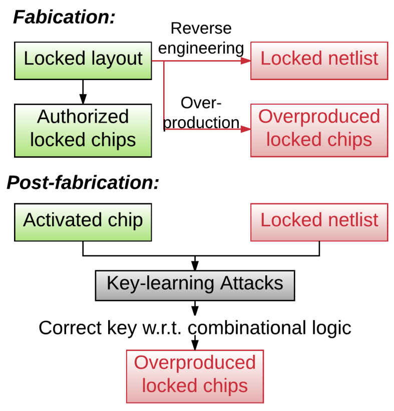


Figure 2: An attack flow against logic locking

to insert additional SAT-attack resistant logic blocks such as the Anti-SAT block [12], the SARLock [13], or an AES block with a fixed AES key [15] into the locked circuit to increase the SAT attack iterations and execution time. Although effective, one limitation of all above countermeasures is that these SAT attack resistant logic blocks have a special and separable structure. They may be removed or nullified by an attacker if they are identified. Then, the SAT attack can be launched to unlock the circuit without these SAT-resistant logic blocks. In [14], a new attack called signal probability skew (SPS) attack was proposed to identify the location of Anti-SAT block in a locked circuit.

3. DELAY+LOGIC LOCKING (DLL)

To enhance the security of existing logic locking techniques, we propose a new technique called delay locking that can thwart the previously proposed attacks. The basic idea of delay locking is to make the circuit's delay dependent on the key value. When logic locking is enhanced with delay locking, the key into a locked circuit not only determines its functionality but also its timing profile. A correct key value should recover the original combinational functionality as well as the correct timing profile that can satisfy the pre-defined timing constraints. On the other hand, a key is said incorrect if it fails to recover a) the original functionality or b) the correct timing profile. Since all previous attack algorithms on logic locking (described in Sec. 2.2) only focus on retrieving the correct combinational functionality, they are not guaranteed to recover the timing-correct key. A "functionality-correct but timing-incorrect" key will result in timing violations and thus making the circuit malfunction.

In the remaining of this section we discuss how the delay locking is implemented and introduce the design objectives, design techniques, and the overall design flow of the DLL design.

3.1 Tunable Delay Key-gates

To make the delay of a key-gate dependent on its key value, we propose a tunable delay key-gate (TDK). Fig. 3 illustrates the structure of the TDK, which combines a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB) [11]. TDB is a widely used solution for post-silicon adjustment of gate/circuit delays. One typical application of TDBs is to correct timing violations that are induced by process, temperature and other type of variances. Various implementations of TDBs have been proposed in previous literatures. One low-power TDB design is proposed by Tsai *et al.* [11], which is based on two inverters with a set of NMOS-based capacitive loads in between, as shown in

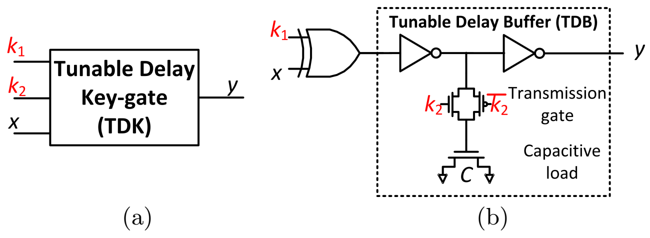


Figure 3: Tunable delay key-gate (TDK): (a) Overview; (b) Implementation

Table 1: Functionality and delay of the TDK

Key ($k_1 k_2$)	Functionality	Delay
00	$y = x$	d_0
01	$y = x$	d_1
10	$y = \bar{x}$	d_0
11	$y = \bar{x}$	d_1

Fig. 3(b). Each capacitive load is controlled by a transmission gate. When the transmission gate is activated by its control signal, the corresponding capacitive load is added into the path between the pair of inverters, thus obtaining tunable delays.

As shown in Fig. 3, each TDK has two key-inputs, one feeding into the XOR/XNOR gate (referred to as the *functional-key*) and the other feeding into the control signal of the TDB (referred to as the *delay-key*). The impact of the keys on the functionality and delay of the TDK is shown in Table 1. The functional-key k_1 determines whether the TDK behaves as a buffer or an inverter while the delay-key k_2 determines whether the TDK gate delay is d_0 or d_1. The TDK delay ratio

$$r = d_1/d_0 \tag{1}$$

can be set at design time by tuning the capacitive load. This delay ratio has a great impact on the circuit delay distribution across different key values as well as the timing violation sensitivity, which will be discussed in Sec. 3.3.2.

In the remaining of the paper, we refer a circuit that's locked with the TDKs to be a *delay-logic-locked circuit (DLL circuit)*.

3.2 Timing Constraints of DLL Circuit

Now we describe how allocation of TDKs impacts the overall timing constraints of the design. Given a sequential design, we can represent it as a directed graph $G = (V, E)$, where V is a set of flip-flops (FFs) and E is a set of edges representing the combinational logic paths between the FFs. Let us assume that T_i and T_j are the clock arrival time at FFs i and j. T_i and T_j may not be the same due to clock skews (the spatial variation in arrival time of different FFs). Let D_{ij}^{long} be the longest path delay between FFs i and j, *i.e.,* the maximum delay among all combinational logic paths between two FFs i and j. Let T_{set}^{j} be the setup time for FF j and T_{clk} be the clock period. To meet the longest path timing constraint, the circuit needs to satisfy:

$$D_{ij}^{long} + T_{set}^{j} \le T_{clk} + T_j - T_i, \forall i, j \tag{2}$$

This longest path timing constraint indicates that the clock period should be large enough for the data to propagate through the combinational logic paths and to be set up at the destination FF before the next trigging edge of the clock arrives.

Besides, the circuit should also satisfy the shortest path timing constraint (hold time constraint) between two FFs. Let D_{ij}^{short} be the shortest path delay between FFs i and j and T_{hold}^{j} be the hold time for FF j. The hold time of the destination FF j must be shorter than the shortest path delay through the combinational logic network, considering the clock skew phenomenon:

$$D_{ij}^{short} \ge T_{hold}^{j} + T_j - T_i, \forall i, j \tag{3}$$

Based on Eq.(2) and Eq.(3), we can represent the timing constraints for a DLL circuit as follows. Let $D_{ij}^{short}(\vec{K})$ and $D_{ij}^{long}(\vec{K})$ be the shortest/longest path delay between FFs i and j of a DLL circuit with a key $\vec{K}$. The timing constraints for the DLL circuit can be represented as:

$$\begin{aligned} D_{ij}^{short}(\vec{K}) &\ge T_{hold}^{j} + T_j - T_i \equiv LB_{ij} \\ D_{ij}^{long}(\vec{K}) &\le T_{clk} + T_j - T_i - T_{set}^{j} \equiv UB_{ij} \end{aligned} \tag{4}$$

Eq. (4) enforces that the combinational path delays (for a correct key $\vec{K} = \vec{K}_C$) between two FFs should satisfy both the shortest and the longest timing constraints. Allocation of keys to the fastest or slowest corner may not be the correct timing solution because a key that increases delay may violate the upper bound (UB) and a key that decreases delay may violate the lower bound (LB). This makes the delay-key determination problem very hard from an attacker's standpoint.

3.3 DLL Design Flow

In this section, we introduce the design objectives, design techniques, and the overall design flow of the DLL design. In general, the design problem is formulated as follows: we want to achieve simultaneous functionality and delay obfuscation by allocating the TDK gates and designing the timing constraints as shown in Eq. (4).

3.3.1 Design Objective

The objective of DLL design consists of two aspects:

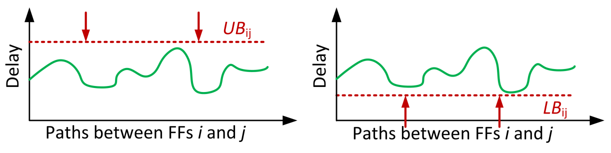
1) Functionality obfuscation: the functionality of a circuit shall be obfuscated in order to conceal the correct functionality and implementation when it passes through the untrusted foundry and other potentially untrusted phases of the supply chain. This requires the TDKs to be located in *functionality-critical* spots that can impact the circuit's functionality.

2) Delay obfuscation: the timing profile of the locked circuit should be obfuscated in order to defend the functionality-oriented attacks as discussed in Sec. 2.2. For an incorrect delay-key, at least one path will violate either the longest or the shortest path timing constraint as described in Eq. (4). This requires the paths which comprise the TDKs to be *timing-critical.*

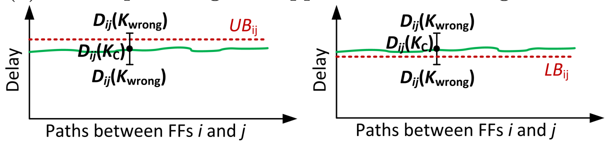
3.3.2 Design Techniques

The first design objective (functionality obfuscation) can be achieved by previously proposed key-gate insertion algorithms [7, 8, 9], which insert the key-gates to internal wires of a netlist to obfuscate the original functionality. However, such locations might not belong to the timing-critical paths. Therefore, an incorrect key might not result in sufficient timing violations to cause error. To achieve better delay obfuscation, we need to make the paths which comprise the TDKs become timing-critical. We propose the following three design techniques to achieve this.

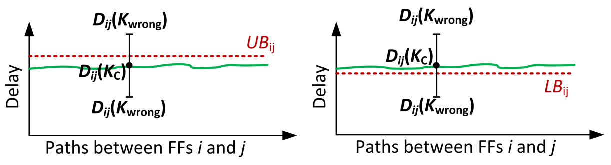
1) Timing bounds design. The timing bounds UB_{ij} or LB_{ij} shall be sufficiently tightened to the longest or shortest path delay, respectively. In other words, either UB_{ij} is set to be larger than but sufficiently close to $D_{ij}^{long}(\vec{K}_C)$, or LB_{ij} is set to be less than but sufficiently close to $D_{ij}^{short}(\vec{K}_C)$, as illustrated in Fig. 4 (a). When the bounds are sufficiently tight, it can ensure that the paths with delay values that are



(a) Technique 1: tighten upper or lower timing bounds



(b) Technique 2: path delay balancing



(c) Technique 3: increase TDK gate delay ratio r

Figure 4: Illustrative examples of three design techniques for delay locking

very close to LB_{ij} or UB_{ij} can violate the timing bounds if the delay-keys are incorrect. As shown in Eq. (4), LB_{ij} and UB_{ij} can be tuned by changing the clock arrival time T_i and T_j, which can be controlled by the clock tree design [4]. Clock tree design which exploits the tuning of T_i and T_j is also called useful-skew based optimization. In this case, we exploit useful skews for security purpose.

2) Path delay balancing. The path delays shall be balanced such that every path delay is close to the longest path delay of the whole combinational block, denoted as $D^{balanced}$. Imbalanced path delays mean that TDKs on the non-critical paths will not result in desired timing sensitivity. On the contrary, making all paths almost equally critical would make the timing profiles sensitive to all TDKs. Hence, path delay balancing make it easier for the circuit to violate the timing constraints when the delay-key is incorrect. Path delay balancing is a widely used technique for eliminating glitches. There exists many approaches for path delay balancing. In this work, we exploit gate sizing and buffer insertion [5] to achieve the path delay balancing.

3) Increase TDK delay ratio r. The TDK delay ratio r shall be large enough to ensure a desired timing violation magnitude, *i.e.,* the difference between an incorrect delay and the violated timing bound. A larger r indicates that when a key-bit flips, the delays of the paths that comprise this TDK will increase or decrease with a larger magnitude. Therefore, these paths would have more severe timing violations, as shown in Fig. 4(c). A larger timing violation magnitude is preferred because it indicates a higher probability of fault occurrences.

The above mentioned three techniques will be used to achieve a high timing violation sensitivity to incorrect keys leading to maximum chances of delay locking to be effective. To quantify the timing violation level of a DLL circuit with an incorrect key, we define a security metric call timing violation ratio (TVR). For FFs i and j, the TVR of a key $\vec{K}$ can be calculated as:

$$TVR_{ij}(\vec{K}) = \frac{\max\{0, D_{ij}^{long}(\vec{K}) - UB_{ij}, LB_{ij} - D_{ij}^{short}(\vec{K})\}}{D^{balanced}} \tag{5}$$

A larger TVR indicates more timing violation and a higher probability of fault occurrences. On the other hand, for a correct key, there is not timing violation and TVR is 0. Based on Eq.(5), we can define the TVR for the whole circuit

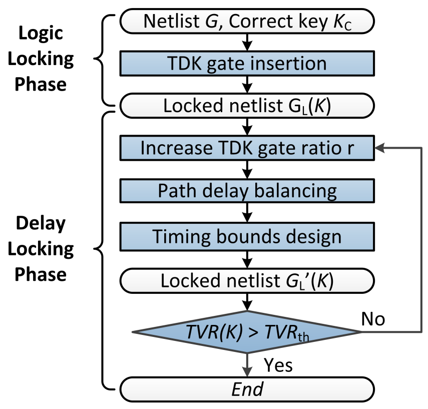


Figure 5: DLL design flow

as follows:

$$TVR(\vec{K}) = \max_{\text{FFs } i,j} \{TVR_{ij}(\vec{K})\} \tag{6}$$

This metric captures the maximum timing violations (if any) among all pairs of FFs.

3.3.3 Design Flow

The overall delay locking design flow is shown in Fig. 5. It consists of two design phases: logic locking and delay locking.

Logic Locking Phase: Given a netlist G and a (randomly generated) correct key $\vec{K}_C$, we first integrate the TDK gates into the combinational block of the original netlist and produce a locked netlist $G_L(\vec{K})$. The locations for the TDK gates can be determined using previously proposed key-gate insertion algorithms, such as random insertion [9], fault-analysis based insertion [8] and interference-analysis based insertion [7], which is not the focus of this work.

Delay Locking Phase: In the delay locking phase, we apply three design techniques as discussed in Sec. 3.3.2 to improve the timing violation sensitivity. The TDK gate delay ratio r is gradually increased until the $TVR(\vec{K})$ (defined in Eq.(6)) for a random key is larger than a pre-defined security threshold TVR_{th}. After the delay locking phase, we obtain the final DLL design $G'_L(\vec{K})$.

4. SECURITY ANALYSIS OF DLL

In this section, we evaluate the security of DLL technique against different attacks.

4.1 TDK Removal Attack

The attacker might attempt to nullify or bypass the TDK gates (either the XOR/XNOR gate, the TDB, or both) in the DLL circuit by replacing it with a normal wire. However, such attempt cannot recover the original functionality and delay if he does not know the correct functional-key and delay-key. On one hand, an TDK can function as a buffer or inverter depending on the functional-key. Replacing it with a wire might flip the correct functionality. On the other hand, nullifying the TDK gates (or just the TDB) is simply equivalent to decreasing the path delays, which might violate the timing lower bound ensured by the shortest path timing constraint.

4.2 Functionality Oriented Attacks

As discussed in Sec. 2.2, previous attacks [6, 7, 10] on logic locking such as the SAT attack algorithm are all functionality oriented, which means that an attacker only focuses on finding a functionality-correct key w.r.t. the combinational logics of the circuit. However, when the delay locking technique is utilized, the key obtained by these attacks cannot unlock the overproduced locked chip because they are not guaranteed to obtain a timing-correct key. A circuit based on a functionality-correct (but not timing-correct) key will violate the pre-defined timing constraints and thus producing incorrect outputs, which will be shown in the experiment section (Sec. 5). As a result, the functionality oriented attacks fail to unlock a chip that is enhanced with the delay locking technique.

4.3 MILP Based Delay-Key Attack

Assuming an attacker can obtain a correct functional-key using previously proposed attacks, in order to unlock the circuit, he has to find a correct delay-key that can satisfy all pre-defined timing constraints. Here we assume a stronger attack model, where the attacker knows the timing LB and UB of all paths and he intends to formulate a mixed integer linear programing (MILP) to solve the delay-key. A straightforward formulation can set UB and LB as constraints for each path delay and find a delay-key that satisfies these constraints. However, this direct formulation is impractical because the number of possible signal paths can be exponential in the total number of gates [2]. This issue can be handled by the classic technique which divides the constraints on path delay into constraints on a gate's arrival time [2]. Let a_i denotes the arrival time of the output of a gate or a primary input i. Let g_i denotes the delay of a gate i. If it's a TDK gate, then g_i is dependent on the delay-key value denoted as x_i. We can formulate the above problem as:

$$\begin{aligned}
&\text{find } \vec{x}, \vec{a} \\
&\text{s.t. } LB_j \le a_j \le UB_j, && \forall \text{PO } j \\
&\quad a_j + g_i \le a_i, && \forall \text{gate } i, \forall j \in inputs(i) \\
&\quad g_i \le a_i, && \forall \text{PI } i \\
&\quad g_i = x_i \times d_1 + (1 - x_i) \times d_0, && \forall \text{gate } i \in TDK \\
&\quad x_i = 0 \text{ or } 1, && \forall \text{gate } i \in TDK
\end{aligned} \tag{7}$$

But the above formulation might not recover the correct arrival time and delay-key because the timing constraint for a gate $a_i = \max\{a_j + g_i\}, \forall j \in inputs(i)$ is relaxed to be $a_i \ge a_j + g_i$ in order to form an MILP formulation. This relaxation will make the arrival time a_i inaccurate and result in an incorrect delay-key. As will be shown in Sec. 5, the resulting key values from above MILP formulation will violate the pre-defined timing constraints. The attacker can attempt to iteratively run the MILP attack and add new constraints on the key values to prune out the incorrect keys discovered in previous iterations. However, since each iteration can only prune out one incorrect key, when key-size is large, the execution time to find a correct delay-key will be exponential in the key-size, as will be shown in Sec. 5.

5. EXPERIMENTS AND RESULTS

5.1 Experiment Setup

We validate the effectiveness of the delay locking technique using 8 sequential benchmarks from ISCAS89. The benchmark information is shown in Table 2. Each benchmark is synthesized using Cadence RTL compiler with SAED 90nm digital standard cell library. Timing information of the standard cell library is extracted for timing analysis.

For the TDK, when the delay-key $k_2 = 0$ the gate delay d_0 is the XOR/XNOR gate delay plus a buffer delay. When $k_2 = 1$, its gate delay is set to be $d_1 = r \times d_0$, where r is the delay ratio. The TDKs are implemented and simulated using Cadence Virtuoso and its gate area and delay are obtained for overhead evaluation. In the logic locking phase,

Table 2: Benchmark information and MILP based delay-key attack results.

Circuit	#Gates	#FFs	#Key gates	MILP Attack	
				Correctness ($\vec{K}_{guess}$)	TVR ($\vec{K}_{guess}$)
s1488	336	6	34	44.12%	28.13%
s5378	748	179	75	58.67%	39.25%
s9234	1014	211	101	59.41%	34.60%
s13207	1924	638	192	63.02%	33.60%
s15850	1952	534	195	63.08%	27.64%
s35932	4763	1728	476	68.07%	26.18%
s38417	5066	1636	507	60.95%	31.87%
s38584	6857	1426	686	58.60%	36.39%

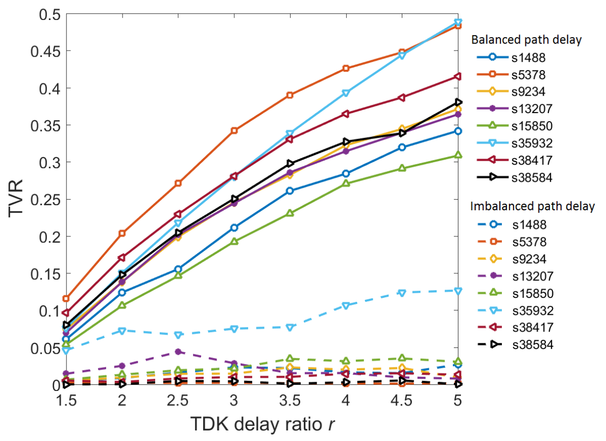


Figure 6: The impact of path delay balancing and TDK delay ratio r on the TVR for 8 ISCAS89 benchmarks.

we randomly generate a correct key and adopt the random key-gate insertion algorithm [9] to insert key-gates into the combinational block of the benchmark. The number of key-gates equals 10% the number of original gates, as shown in Table 2. In the delay locking phase, we apply three design techniques (Sec. 3.3.2) to improve delay obfuscation.

To evaluate the level of timing violation given an incorrect delay-key, we compute the TVR (defined in Eq. (6)) for the DLL circuit under the case that the functional-key values are all correct but the delay-key values are randomly generated. Noted that in the experiment we set the timing bounds to the balanced path delay for the easy of TVR computation and comparison. The TVR is averaged over 1000 random key trails.

5.2 Impact of Path Delay Balancing and TDK Delay Ratio

In this experiment, we validate the effectiveness of our proposed design techniques (Sec. 3.3.2) in improving the level of timing violations. We assume technique 1 is always applied and evaluate the impact of other two techniques: path delay balancing and increasing TDK delay ratio r . Fig. 6 shows the TVR values of different delay ratios r when the path delay balancing is applied (bold lines) and when it's not applied (dash lines). As seen, without path delay balancing (dash lines), the TVR values of most benchmarks are below 5%, and increasing the delay ratio r cannot effectively achieve a higher TVR value. This is because that when path delays are imbalanced, most TDKs are not in the timing-critical paths so an incorrect delay-key cannot cause severe timing violations. When path delay balancing is applied (bold lines), we can see a remarkable improvement in TVR for all choices of r. Besides, as r increases, the TVR value increases from about 8% to 39%. The value of r can be designed to achieve a desired TVR threshold.

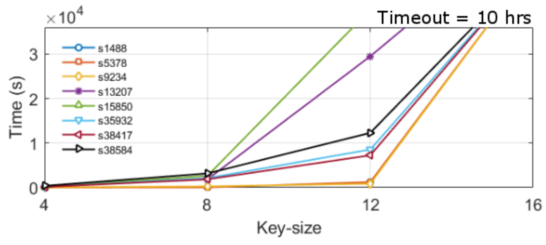


Figure 7: Iterative MILP attack results (Timeout is 10 hours)

5.3 MILP Based Delay-Key Attack Result

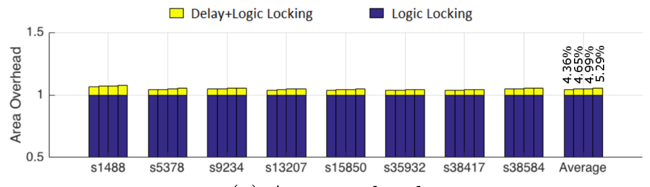
As discussed in Sec. 4.3, a MILP formulation might be applied by an attacker to find the delay-key. However, such formulation requires necessary constraint relaxation to satisfy the linear formulation and thus fail to retrieve the original delay and the correct delay-key. To validate this analysis, we implemented the MILP formulation based attack. For each DLL circuit, r is set to 3 and path delay balancing is applied because they can result in a relatively large TVR value as shown in Fig. 6. The correctness (# bits that are the same as the correct key) and the corresponding TVR values of the MILP solution $\vec{K}_{guess}$ are computed and shown in Table 2. As seen, the resulting delay-keys are incorrect for all benchmarks and they will violate the timing constraints with an average TVR of 32.21%. We also implement an iterative MILP attack which iteratively performs the MILP attack and add new constraints on the key values to prune out the incorrect keys discovered in previous iterations. The attack results on 8 benchmarks for key-size ranging from 4 to 16 is shown in Fig. 7. As seen, since each iteration can only prune out one incorrect key, as the key-size increases, the execution time to find a correct delay-key increases exponentially. When key-size is 16, the attack timeouts (>10 hrs) for all benchmarks.

5.4 Overhead Evaluation

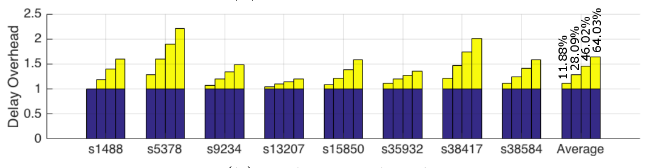
Fig. 8 shows the area and delay overhead of TDK based delay+logic locking for 8 benchmarks when compared to the conventional XOR/XNOR based logic locking. For each benchmark, we vary the TDK delay ratio r and report its impact on area and delay. Four bar plots of each benchmark correspond to $r = 2, 3, 4, 5$. As seen, when r increases from 2 to 5, the average area overhead increases from 4.36% to 5.29%. The area overhead mainly comes from the TDBs. A larger r requires a larger capacitive load so it results in a slightly higher area overhead. The impact of r is mainly reflected in the delay overhead. As seen, with r increases, the averaged delay overhead increases from 11.88% to 64.03%. This is because that a larger r will lead to a larger key-gate delay (if the correct delay-key is 1) and increase the delay for the overall circuit. Combined with the result in Fig. 6, we can see that our approach is capable of generating a tradeoff between the TVR and the performance overheads. By tuning the TDK delay ratio r, we can achieve a desired TVR with acceptable overheads.

6. CONCLUSION

In this paper, we propose a new technique called delay locking to enhance the security of existing logic locking techniques. A tunable delay key-gate is proposed and used to obfuscate both functionality and timing profile of an IC design. An overall delay+logic locking design flow is proposed to increase the timing violation sensitivity to incorrect key values. The security of proposed delay locking technique is evaluated with previous attacks and a new attack based on MILP. Both analytical and experimental results show that such formulation fails to find the correct delay-key.



(a) Area overhead



(b) Delay overhead

Figure 8: Area and delay overhead for the DLL technique. Four bar plots of each benchmark correspond to TDK delay ratios $r = 2, 3, 4, 5$

7. ACKNOWLEDGMENTS

This work was supported by NSF under Grant No. 1223233 and AFOSR under Grant FA9550-14-1-0351.

8. REFERENCES

[1] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 2010.

[2] C.-P. Chen, C. C. Chu, and D. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):1014–1025, 1999.

[3] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 49–54. IEEE, 2014.

[4] V. Khandelwal and A. Srivastava. Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008.

[5] S. Kim, J. Kim, and S.-Y. Hwang. New path balancing algorithm for glitch power reduction. *IEE Proceedings-Circuits, Devices and Systems*, 148(3):151–156, 2001.

[6] S. M. Plaza and I. L. Markov. Solving the third-shift problem in IC piracy with test-aware logic locking. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(6):961–971, 2015.

[7] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference*, pages 83–89. ACM, 2012.

[8] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault analysis-based logic encryption. *Computers, IEEE Transactions on*, 64(2):410–424, 2015.

[9] J. A. Roy, F. Koushanfar, and I. L. Markov. Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 1069–1074. ACM, 2008.

[10] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 137–143. IEEE, 2015.

[11] J.-L. Tsai, D. Baik, C. C.-P. Chen, and K. K. Saluja. A yield improvement methodology using pre-and post-silicon statistical clock scheduling. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 611–618. IEEE Computer Society, 2004.

[12] Y. Xie and A. Srivastava. Mitigating sat attack on logic locking. *Cryptographic Hardware and Embedded Systems (CHES)*, 2016.

[13] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu. Sarlock: Sat attack resistant logic locking. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pages 236–241. IEEE, 2016.

[14] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran. Security analysis of anti-sat. Cryptology ePrint Archive, Report 2016/896, 2016. http://eprint.iacr.org/2016/896.

[15] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri. On improving the security of logic locking. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2015.