ALT-Lock: Logic and Timing Ambiguity-Based IP Obfuscation Against Reverse Engineering

Jonti Talukdar, Woo-Hyun Paik, Eduardo Ortega, and Krishnendu Chakrabarty, *Fellow, IEEE*

***Abstract*— We present a logic ambiguity-based intellectual property (IP) obfuscation method that replaces traditional key gates with key-controlled functionally ambiguous logic gates, called LGA gates. We also protect timing paths by developing timing-ambiguous sequential cells called TA cells. We call this locking scheme ambiguous logic and timing logic locking (referred to as ALT-Lock). ALT-Lock ensures a two-pronged system-level security scheme where the attacker is forced to unlock not only combinational logic obfuscation but also timing obfuscation. We show that a combination of logic and timing ambiguity (TA) provides security against oracle-guided attacks. This method is superior to other traditional IP protection schemes such as combinational or sequential locking as it guarantees security against both oracle-guided and oracle-free attacks, while ensuring low power, performance, and area (PPA) overhead.**

***Index Terms*— Hardware security, intellectual property (IP) security, standard cells, supply chain security, timing obfuscation.**

I. Introduction

THE globalization of the integrated circuit (IC) supply chain has led to outsourcing of IC fabrication, packaging, assembly, and testing operations to third-party entities. Guaranteeing the trustworthiness of all third-party vendors part of the global supply chain is extremely challenging, resulting in an untrusted IC supply chain. This globalized IC manufacturing flow poses significant risk to the security of intellectual property (IP). Exposing IP to untrusted third parties in the supply chain may result in IP theft, counterfeiting, IC overbuilding, or the insertion of hardware Trojans in the design. A variety of countermeasures such as split manufacturing, IC metering, and IP obfuscation have been proposed to tackle these threats [1].

Logic obfuscation, also called logic locking, protects the IP by inserting additional "locking logic" into the design. Unlocking is possible only upon the application of the correct key. The two major techniques considered for logic locking are combinational and sequential locking. Combinational locking techniques require the insertion of key gates into the combinational cone of logic within the design [2]. Sequential locking techniques insert additional obfuscation states and state transitions to prevent an adversary from ascertaining the correct order of functional states within the IP [2].

Logic locking protects against IP theft through IC reverse engineering, IP counterfeiting, and overbuilding as the design remains functionally inoperable unless the appropriate unlocking key is applied. However, different forms of attacks, aimed at recovering functional key used for logic obfuscation, have been developed. For example, SAT is an oracle-guided attack that can break most combinational logic-locking solutions [3], [4], [5]. In addition, other variants of SAT, such as KC2, have been created to successfully attack sequential locking [5]. Thus, there exists a need for stronger locking schemes that provide security against not only oracle-guided attacks such as SAT-based attacks but also a rapidly expanding arsenal of oracle-free attacks [6], [7], [8].

In this work, we present a logic and timing ambiguity (TA)-based IP obfuscation technique that obfuscates both the combinational logic paths and timing paths in the design through the use of novel combinational and sequential locking cells. We refer to this technique as ambiguous logic and timing locking (referred to as ALT-Lock). Through ALT-Lock: 1) logic ambiguity is achieved by the insertion of functionally ambiguous gates, i.e., logic gates (termed as LGA gates) whose functionality is determined based on the applied key and 2) TA is achieved by the insertion sequential elements whose timing properties change upon the application of the appropriate key. The use of both logic and TA provides twofold security such that not only the functionality of the combinational logic but also the cycle-based timing behavior of the circuit is obfuscated. Unlike the existing IP obfuscation techniques that rely on the use of XOR-gates, look-up-tables (LUTs), or multiplexers (MUXes) for locking, we present functionally ambiguous logic gates (LGA gates) that contain embedded key transistors (MOSFETs) to allow the logic gates to change their functionality based on applied key [9], [10], [11]. The use of LGA gates allows us to not only obfuscate the design functionally but also control the output corruptibility of the design. We show that combining both the LGA gates with timing ambiguous sequential cells leads to significant protection against both oracle-guided and oracle-free attacks.

The major contributions of this article are as follows.

1) We describe three different types of LGA gates [NAND/NOR (NDNR), NAND/XOR (NDXR), and NAND/XNOR (NDXNR)], which can achieve gate-level

Manuscript received 30 October 2023; revised 19 March 2024, 7 May 2024, and 29 May 2024; accepted 31 May 2024. Date of publication 17 June 2024; date of current version 26 July 2024. This work was supported in part by the Semiconductor Research Corporation (SRC) Center for Heterogeneous Integration of Micro Electronic Systems (CHIMES), one of the seven centers in Joint University Microelectronics Program (JUMP) 2.0; in part by SRC Program sponsored by Defense Advanced Research Projects Agency (DARPA) under Grant Task 3136.005; and in part by the National Science Foundation under Grant CNS-2011561. *(Corresponding author: Jonti Talukdar.)*

Jonti Talukdar and Woo-Hyun Paik are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: jonti.talukdar@duke.edu).

Eduardo Ortega and Krishnendu Chakrabarty are with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2024.3411033.

Digital Object Identifier 10.1109/TVLSI.2024.3411033

logic obfuscation. These gates provide the necessary logic ambiguity for ALT-Lock.
2) We present a timing-ambiguous sequential cell (TA cell), which can change its functionality between a buffer (delay element), latch, and a flip-flop, based on the applied key. This is used to achieve timing obfuscation.
3) We present a methodology for the insertion of the logic and timing ambiguous obfuscation cells for ALT-Lock, along with a dynamic obfuscation of logic paths using TA cells that are driven by reconfigurable blocks (RBs).
4) We evaluate the effectiveness of ALT-Lock against state-of-the-art oracle-guided and oracle-free attacks.
5) We present an assessment of the power, performance, and area (PPA) overhead associated with ALT-Lock.

The rest of this article is organized as follows. Section II reviews prior work along with the necessary background and threat model. Section III presents the architecture of the ALT-Lock obfuscation scheme along with the insertion methodology. Section IV presents the security analysis of ALT-Lock. Section V presents a discussion of the associated overhead. Finally, Section VI concludes the article.

II. Background and Motivation

A. Related Prior Work

Combinational logic locking using XOR/XNOR gates, LUTs, or MUXes remains vulnerable to attacks based on key sensitization and test-data access [2], [12]. Improved logic-locking methods that exploit structural properties of the netlist to prevent sensitization of key bits have been designed to protect against such attacks [13]. By controlling output corruptibility and preventing key-bit sensitization, these logic-locking methods can defend against previously stated attacks [13]. However, the attack described in [3] highlighted a new vulnerability of logic locking solutions. The oracle-guided Boolean Satisfiability (SAT) attack prunes the key search space by identifying distinguishing input patterns (DIPs). By identifying these patterns, the SAT attack is able to reverse engineer (RE) secret key information from logic locking solutions [3]. New SAT-resilient logic locking was created to defend against the SAT attack [4] using point functions to restrict the number of keys that can be eliminated in each iteration of the SAT attack [4], [9]. However, while providing defense against the SAT attack, these logic locking solutions suffer from poor output corruption and are vulnerable to removal and approximate attacks [14], [15].

Sequential locking techniques modify the design's original finite state machine (FSM), expanding the functional state space through obfuscation states and state transitions [16], [17]. Only through a specific input sequence can these sequential logic locking circuits be unlocked. In addition, the obfuscation state space can also include a set of authentication states that can be traversed to generate a unique output. This unique output from authentication states can be used to watermark the design and detect tampering. Attacks that reverse-engineer the underlying states from sequential logic locking solutions using state transition graphs (STGs) have been developed [18], [19]. Once the STG is known, the locked FSM can be deobfuscated by recovering the appropriate key sequence [18], [19]. It is important to note that these attacks rely on an exhaustive search of the state space to identify the STG. To defend against such attacks, [17] created a method to add false state transitions to augment FSMs. However, newer SAT attacks have been developed to attack these sequential logic locking schemes [5]. Thus, there still lies an inherent vulnerability with current logic locking techniques (both sequential and combinational) [5].

SAT attack requires complete controllability and observability of the design [3]. Scan test access can be used to apply the SAT attack on individual combinational blocks of a sequential system. Defenses have been proposed to thwart such attacks [20], [21]. These defenses lock scan chains and obfuscate scan data to ensure the SAT attack requirements are not met. Unlike sequential locking techniques, scan obfuscation does not obfuscate the state of the system. Instead, it prevents scan readout by either corrupting scan chain data or blocking scan access altogether [20], [22], [23]. Attacks such as [7] can bypass scan data corruption by modeling the scan obfuscation logic as part of the overall combinational logic. However, these attacks are not effective against defenses that restrict scan access altogether [21]. Therefore, oracle-free bounded model-checking (BMC)-based sequential-SAT attacks have been proposed against such defenses [24].

B. Need for Ambiguity in Both Logic and Timing

The different categories of IP obfuscation methods described in prior work (combinational locking, sequential locking, and test locking) have been tailored to be resilient against specific categories of attacks, which broadly include oracle-guided and oracle-free attacks. However, the inability of these methods to obfuscate both the timing properties of the circuit along with the functional logic prevents them from achieving resilience against both simultaneously. Although it is possible to combine all these methods to create an ensemble locking scheme that is resilient against both the categories of attacks, such a method is likely to suffer from a large area overhead. This is because these methods require the insertion of specialized blocks, such as point-function implementations, along with scan obfuscation circuitry to prevent oracle access.

Instead of relying on different system-level approaches, we address this problem by designing new types of locking cells that can achieve both timing and logic obfuscation. Unlike the use of traditional XOR-based key gates that flip the signal phase (from 0/1) based on the key value, LGA gates flip the functionality of the gate itself. Depending on the type of LGA gate, functionality can be flipped between NDNR, NDXR, and NDXNR respectively. Thus, depending on the type of LGA gate and its location, the designer can control the amount of output corruption induced. Similarly, obfuscating the timing path by designing a sequential locking cell that can shift its property between a flip-flop, latch, and buffer, the key value can control whether the signal is latched (either level or edge-triggered) or buffered through a given path. We make minimal modifications to the existing CMOS-based implementations to achieve low overhead while achieving both logic and timing obfuscation.

C. Threat Model and Assumptions

The threat model specifies the untrusted entities in the IC supply chain and highlights their resources and constraints. The typical threat model used in logic-locking literature assumes that both the foundry and the end-user are untrusted [25]. Thus, it is assumed that the attacker has access to the following resources [12].

1) *Locked Netlist:* The end-user can RE the locked netlist of the IP using advanced imaging, delayering, depackaging techniques [26]. Similarly, the foundry can reconstruct the netlist of the locked IP by processing the GDSII files.
2) *Activated Chip:* The untrusted entity can procure an unlocked and fully functional chip from the open market.

The attacker can perform topological analysis of the RE'd netlist (locked) and use it for functional simulations, equivalence checking, etc. The attacker can also use the activated chip as an oracle, using it to query functionally correct input/output combinations. Both these resources allow the attacker to launch oracle-guided attacks.

As discussed in [2] and [12], invasive probing attacks that partially recover the key directly from activated chips do not fall under the scope of attacks on logic-locking. This is because these attacks do not exploit the underlying weakness of the locking solution itself but rather exploit the vulnerabilities associated with the fabrication technology used to manufacture the chip. Moreover, these attacks have not been demonstrated beyond the 28-nm technology node thus far due to challenges associated with probing for smaller process nodes [27].

III. ALT-Lock: Architecture and Locking Methodology

In this section, we describe the circuit-level implementation of the proposed logic and TA cells. We also present the methodology used for inserting these cells.

A. Achieving LGA

The XOR-gates have been widely used as key gates due to the ambiguity introduced in signal propagating through the gate. Consider an XOR-based key gate $G_{\text{XOR},k}(i)$, where k is the locking key bit and i is the input bit. Depending on k, the output of the XOR key gate, $O_{\text{XOR},k}$, is either i or $\bar{i}$. Thus, using XOR-gates as key gates introduces input signal ambiguity, i.e., for $k = 0$, the output signal remains unchanged, whereas for $k = 1$, the output signal is flipped. Thus, the output of the circuit is given by: $O_{\text{XOR},k} = i \oplus k = k(\bar{i}) + \bar{k}(i)$. We define logic gate ambiguity (LGA) as the ability to control the *functional logic type* of the combinational gate based on the applied key. Unlike an XOR-based key gate, where the key bit, k, only determines the phase of the input signal transmitted to the output (i.e., phase ambiguity in i with either $O = i$ or $O = \bar{i}$), LGA introduces Boolean functional ambiguity in the output of the obfuscated gate.

Let $G_{\text{LGA},k}$ denote an LGA gate controlled by key, k, which performs the transformation $G_{\text{LGA},k}{:}I \rightarrow O$, where $I = \{0,1\}^M$ and $O = \{0,1\}^N$ represent an M-bit input and N-bit output in the Boolean domain, respectively. In this work, we design LGA gates with two inputs and a single output, and therefore, $M = 2$ and $N = 1$. Unlike XOR-based key gates that work on obfuscating the phase of a single bit (i.e., only for $M = 1$), $G_{\text{LGA},k}$ obfuscates the output as a function of the key and all the gate's m input bits, $O_{\text{LGA},k} = G_{\text{LGA},k}(i_1, i_2, \ldots, i_M)$, where i_M is the Mth input bit of the input vector, I. For example, if a logic gate switches functionality between NAND and NOR operations, then the value of the applied key will determine whether $O_{\text{LGA}} = k(\overline{i_1 \cdot i_2}) + \bar{k}(\overline{i_1 + i_2})$, where i_1 and i_2 are the two inputs to the circuit and O_{LGA} is the obfuscated output bit.

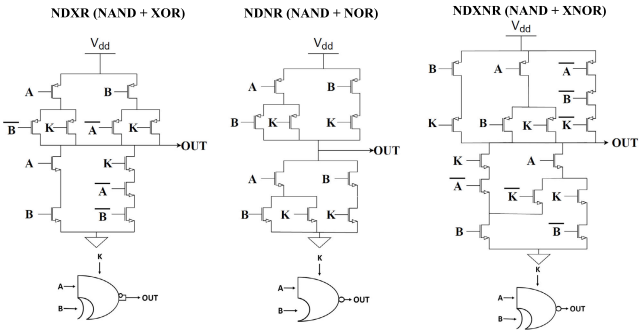
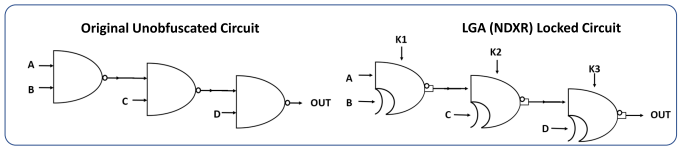


Fig. 1. CMOS implementation of the two-input NDXR, NDNR, and NDXNR LGA gates.

We implement three different types of mixed functionality two-input LGA gates, namely, NDXR, NDNR, and NDXNR. Depending on the value of the applied key, the LGA gates function as follows.

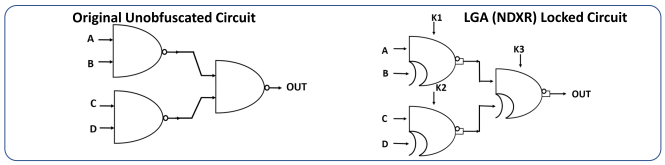
1) NDXR gates can change their functionality between NAND ($k = 0$) and XOR ($k = 1$). Thus, $O_{\text{NDXR}} = k(i_1 \oplus i_2) + \bar{k}(\overline{i_1 \cdot i_2})$.
2) NDNR gates can change their functionality between NAND ($k = 0$) and NOR ($k = 1$). Thus, $O_{\text{NDNR}} = k(\overline{i_1 + i_2}) + \bar{k}(\overline{i_1 \cdot i_2})$.
3) NDXNR gates can change their functionality between NAND ($k = 0$) and XOR ($k = 1$). Thus, $O_{\text{NDXNR}} = k(\overline{i_1 \oplus i_2}) + \bar{k}(\overline{i_1 \cdot i_2})$.

Fig. 1 illustrates the CMOS implementation of the three different types of LGA gates. LGA cells can be used to both structurally and functionally obfuscate combinational logic gates. They can be used to replace NAND, NOR, XOR, or XNOR logic gates, all of which are prevalent in modern designs. Using a combination of LGA gates, we can achieve different levels of functional obfuscation, i.e., output corruption. This is illustrated through examples shown in Fig. 2. In Fig. 2(a), a small combinational circuit consisting of three series cascaded two-input NAND gates is obfuscated using the NDXR variant of LGA gates. Note that different key combinations produce different values of output corruption, highlighted by the last column that indicates the percentage match between the output of the unobfuscated and the LGA-locked circuit. The key $(0, 0, 1)$ produces maximum output corruption. This is because $K = (0, 0, 1)$ leads to the logic gate driving the primary output (PO) of the circuit being functionally obfuscated into an XOR gate, leading to maximum output corruption. Similarly, Fig. 2(b) shows the output



K3K1K2/ ABCD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Match
0 (ORG)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	0	-
1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	1	19%
2	1	1	1	0	1	1	1	0	1	1	1	0	1	0	1	1	56%
3	1	0	0	1	1	0	0	1	1	0	0	1	0	1	1	0	69%
4	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	1	81%
5	0	1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	31%
6	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	0	81%
7	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1	44%

(a)



K3K1K2 /ABCD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Match
0 (ORG)	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	-
1	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0	1	6%
2	0	1	1	0	0	1	1	0	0	1	1	0	1	1	1	1	44%
3	1	0	0	1	1	0	0	1	1	0	0	1	0	1	1	0	69%
4	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	44%
5	1	1	1	0	0	0	0	1	0	0	0	1	1	1	1	0	69%
6	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	31%
7	1	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	44%

(b)

Fig. 2. Example illustrating NDXR-type LGA-locking of combinational logic consisting of three two-input NAND gates (a) cascaded in series and (b) cascaded in a tree topology. The tables highlight the corrupted output (in red) of both the locked designs due to the activation of LGA gates across all combinations of inputs and key values.

corruption for an NDXR-type LGA-locked circuit consisting of three NAND gates connected in a tree-based topology. Note that the same key, $K = (0, 0, 1)$, produces maximum output corruption. The designer can combine different types of LGA cells and place them appropriately in the design to achieve the desired levels of output corruption. This process of security-aware LGA placement and insertion is described in more detail in Section III-C.

B. Achieving TA

Recent work has shown that output corruption and oracle-guided attacks resilience against SAT-based attacks can be two competing objectives [28]. If a locking scheme produces large amount of output corruption, it becomes easier for the oracle-guided attack to find equivalence key sets of large sizes that can be eliminated from the key search space. Similarly, point-function-based schemes that produce very less amount of output corruption fare much better against such oracle-guided attacks due to the difficulty in finding equivalence key classes of significant sizes, forcing the number of iterations required by the SAT-solver to approach brute-force enumeration of the key search space. However, by expanding the functional obfuscation space from purely combinational to TA, the designer is no longer constrained along the tradeoff between oracle-guided attack resilience and output corruption [29]. Introducing TA in the design through key-controlled sequential cells ensures that not only the combinational functionality but also the timing behavior of the reverse-engineered netlist is obfuscated. This makes it impossible for the attacker to establish structural equivalence (through timing behavior) between the oracle and the reverse-engineered netlist, making oracle-guided attacks impossible. This is described more formally in Section IV.

We design a TA sequential cell to change its timing properties between a flip-flop, a latch, and a delay element based on the supplied value of the key bits. The basic element of a TA cell consists of a key and clock-controlled inverter, as shown in the box in Fig. 3. Note that when $k = 0$, the clock branches of the inverter are bypassed transforming it into a regular combinational inverter. However, when $k = 1$, the inverter

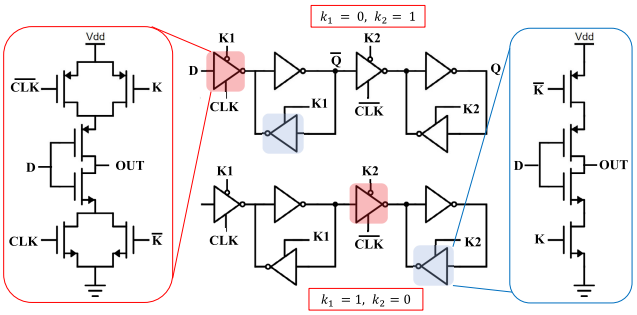


Fig. 3. Single TA cell consisting of two stages of clock and key-controlled inverters with feedback.

becomes clock-controlled. These inverters can be connected in a variety of ways to achieve the functionality of a latch, a flip-flop, or a delay element. The TA cell is modeled as a memory element with two key inputs, (k_1, k_2) such that the output O_{TA} changes as follows.

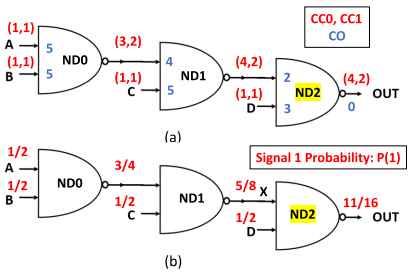
1) $(k_1 = 0, k_2 = 0)$*:* This key configuration programs the TA cell to work as a delay element consisting of a chain of four inverters.
2) $(k_1 = 1, k_2 = 1)$: This key configuration programs the TA cell to work as a flip-flop, as both the feedback paths of both the stages work.
3) $(k_1 = \overline{k2})$*:* This key configuration (where k_1 is the complement of k_2, i.e., $(k_1, k_2) = (0, 1)$ or $(1, 0)$) programs the TA cell to work as a latch, as one of the clock-controlled feedback path is active while the other is inactive.

Fig. 3 illustrates the building block along with two configurations of the TA cell.

C. ALT-Lock: Security-Driven Insertion Methodology

In this section, we provide general guidelines on the selection and placement of LGA gates and TA cells in a security-aware manner.

1) Logic Ambiguity: Choosing LGA Type and Placement: Depending on the type of gates present in the combinational logic, the designer can choose to replace NAND, NOR, XOR, or XNOR gates with their LGA equivalent. Each of these gate types can be replaced with either of the three LGA gates



X/D	Probability (@ND2)
0/0	3/16
0/1	3/16
1/0	5/16
1/1	5/16

2-Input	NAND	NOR	XOR	XNOR
00	1	1	0	1
01	1	0	1	0
10	1	0	1	0
11	0	0	0	1
HIT	-	50%	75%	25%

(c)

Fig. 4. Computing (a) observability and (b) signal probability for different nets in the design. (c) Table showing updated signal probability for different input combinations and percentage match in functionality between LGA gate candidates.

(NDNR, NDXR, or NDXNR). Therefore, the designer has the following options when introducing logic-ambiguity-based obfuscation to the design: 1) replace an existing two input gate with an LGA variant and 2) insert a new LGA gate in the design. Note that inserting a two-input LGA gate in a single-bit signal path requires the other input of the LGA gate to be tied, making it easier for removal style attacks. This would necessitate the insertion of multiple gates to obfuscate a single functional path, leading to the addition of unnecessary delay and area overhead. As a result, in this work, we focus on the first option, i.e., replacement of existing gates with LGA gates. This brings us to the following guidelines.

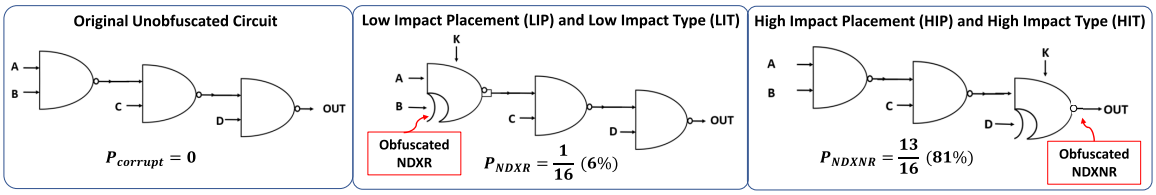
1) *LGA Placement: Which gate to pick for replacement?* The choice of gate for replacement with an LGA gate is determined by the amount of functional corruption (output corruption) desired at the PO. For example, an LGA gate placed closer to the output has more influence on perturbing that output. Thus, this decision is tied very closely to the observability of the gate. The more observable a net that is driven by an LGA gate, the greater is the likelihood of functional corruption. This is also evident from the example demonstrated in Fig. 2 where the specific key (0, 0, 1) leads to the gate closest to the PO causing functional obfuscation.
2) *LGA Impact: Which LGA gate type to use?* The choice of LGA gate is also based on the percentage of functional corruption desired at the output. Once the original gate is picked for replacement, the designer can choose the LGA gate variant that maximizes output corruption. This can be done by computing the change in signal probability post replacement and comparing it across all the potential LGA gate candidates.

a) Working example: Let us consider the example circuit from the previous section consisting of three two-input NAND gates in series (ND0, ND1, and ND2). To decide LGA placement, i.e., which gate to replace; we compute the observability of the different nets in the design. As is consistent with the example shown in Fig. 2(a), picking the gate with highest observability (ND2) increases the ability to introduce maximum output corruption. Fig. 4(a) shows the observability values for all the nets in blue. To choose the LGA gate type for replacement, we compute the input signal probabilities for the different nets in the design. If P_i is the input signal probability for the ith input net of an M input gate, then the output signal probability for the different gates is given as follows: 1) AND: $\prod_i P_i$; 2) NAND: $1-\prod_i(1-P_i)$; 3) OR: $\Sigma_i P_i - \prod_i P_i$; 4) NOR: $1-(\Sigma_i P_i - \prod_i P_i)$; 5) XOR: $\Sigma_{i,j} P_i(1-P_j)$; and 6) XNOR: $1-P(\text{XOR})$. Using these rules, we compute the signal probabilities for all the nets in the design. As we chose ND2 for replacement (based on *LGA Placement Guideline*), we compute the probability for output corruption for the other three LGA candidates. This is done by identifying all the inputs for which the output of the original gate differs from the LGA gate, i.e., the output becomes corrupted. Let $I_{\text{corrupt}} \subset I$, such that $\forall i \in I_{\text{corrupt}}: G_{\text{ND2}}(i) \neq G_{\text{LGA}}(i)$. Then, the probability of output corruption due to a given LGA candidate is given by $P_{\text{LGA}} = \Sigma_i i P_i$. Through data presented in Fig. 4, we obtain $P_{\text{NDXR}} = 0.19$, $P_{\text{NDXNR}} = 0.5$, and $P_{\text{NDXNR}} = 0.81$. Thus, picking NDXNR LGA type will lead to most output corruption. This is also evident if we observe the percentage match in the output of the LGA candidates when compared with the gate being replaced (ND2) from the last row in the bottom table [see Fig. 4(c)]. It has the lowest match percentage, indicating maximum output corruption.

Thus, the choice of which gate to replace in the design and what type of LGA gate to replace it with are both driven by the amount of functional corruption desired by the designer. Fig. 5 illustrates the impact of choosing two extreme corners of the LGA placement and LGA gate type guidelines described above. In the first case, the least observable gate (ND0) is chosen for replacement. Furthermore, ND0 is chosen to be replaced with an NDXR LGA gate type, which has the maximum functional overlap or percentage match in terms of the number of inputs producing the same output. This leads to poor output corruption (6%). In the second case, the gate with maximum observability (ND2) is chosen and replaced with the NDXNR LGA gate type which has the maximum functional mismatch. This leads to significant output corruption (81%).

b) Preventing key-based logic sensitization: Placing LGA gates in highly observable locations in the design, although increases output corruption, can also facilitate functional reverse engineering. This is because using oracle access, the attacker can set the primary inputs (PIs) to the circuit such that the LGA gate experiences all the four input combinations. The attacker can then analyze the POs for all those input combinations, thereby reverse engineering the functionality of the LGA gate. This is possible only when the LGA gate's inputs have high controllability and the outputs are propagated without any form of interference from other LGA gates. To prevent this type of reverse engineering, multiple LGA gates should be placed in the design, such that the output of neither LGA gate can be independently propagated to a PO. An LGA gate covers a PO if its output can be propagated to that PO. By ensuring that no PO is exclusively covered by just one LGA, it becomes impossible for the attacker to determine the functionality of an LGA without making assumptions about other LGAs. This is shown in Lemma 1.

Lemma 1: If multiple LGAs are in the same logical path, i.e., the fan-out cone of one LGA drives another LGA downstream, then the individual outputs of the LGAs in that path cannot be independently propagated to the same PO without exploring all possible key combinations for the interfering LGAs.



ABCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
ORG	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	0
LIT+LIP	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1	0
HIT+HIP	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	1

Fig. 5. Impact of two extreme corners of LGA gate placement and LGA gate type decisions on the output corruptibility.

Proof: Consider two LGA gates, G_{LGA_1,k_1} and G_{LGA_2,k_2}, controlled by keys k_1 and k_2, respectively. The output of these LGA gates (O_{LGA_1,k_1}, O_{LGA_2,k_2}) lies exclusively in the fan-in cone of the PO, such that the output of the obfuscated design is given by $O_{\text{PO}} = f(O_{\text{LGA}_1,k_1}, O_{\text{LGA}_2,k_2}, I)$, where f represents the logic transformation between the obfuscated output of the LGA gates and the input I. Given that f is reconstructed from the reverse-engineered netlist, the attacker's goal is to find the input vector I that can independently propagate either O_{LGA_1,k_1} or O_{LGA_2,k_2} to O_{PO}. However, since the attacker is unaware of either k_1 or k_2, they cannot propagate either O_{LGA_1,k_1} or O_{LGA_2,k_2} without making assumptions about the other key (k_1 or k_1). Thus, to propagate O_{LGA_1,k_1} to O_{PO}, the attacker must make G_{LGA_2,k_2} transparent, which cannot be done unless k_2 is known. Similarly, to propagate O_{LGA_2,k_2} to O_{PO}, the attacker is forced to make G_{LGA_2,k_2} transparent, which cannot be done unless k_1 is known. Thus, both the LGA gates in the same logic path are said to be strongly connected, i.e., to propagate the output of one gate to the PO, the attacker is forced to brute-force through all the keys of the other LGA gate, thereby forcing the attacker to brute-force through the entire key search space. □

2) TA: Choosing Timing Paths and Key Configuration: The designer has the option to replace the existing delay cells, latches, or flip-flops in the design with the TA cell or insert such cells additionally (and program them as delay elements to prevent functional impact on timing). In either of these cases, i.e., replacement of an existing element/cell or insertion of a new element, the designer has to be wary of the attacker trying to *infer* the sequential element type by analyzing the length of the different timing paths. As the functional clock frequency of the design might already be common knowledge and assumed to be known to the attacker, the attacker can simply analyze the length of all the paths and eliminate keys that lead to extremely long paths. As a result, the following criteria must be used for the insertion of (or replacement with) TA cells.

1) Any flip-flop can be replaced with a latch and vice versa. Therefore, replace either a flip-flop or a latch with a TA cell with key $(0, 1)$ or $(1, 0)$. This will directly introduce a timing violation in the design as it may propagate unwanted data transitions to the downstream logic, thereby increasing the likelihood of capturing incorrect data by sequential cells further downstream.
2) Buffers in long paths in the design can be replaced (also known as splicing) with TA cells. This will also introduce timing violation for all the keys except $(0, 0)$.
3) TA cells can be inserted in longer paths in the design with key $(0, 0)$. This makes it difficult for the attacker to reconstruct the design as the cell could function as either a buffer or a sequential element from the attacker's perspective.

Preventing Combinational Loop-Based Reconstruction: Modern digital designs seldom have combinational feedback loops. Therefore, it is possible for the attacker to identify paths exclusively with TA cells (and no other sequential elements) that form feedback loops. Attackers can eliminate the buffer functionality of TA cells in such paths as modern designs avoid combinational feedback loops. Therefore, during TA replacement or insertion, it should be ensured that the path after TA cell insertion/replacement does not form a feedback loop without the presence of other sequential elements that can break such loops.

3) Logic and TA Co-Design: Note that LGA gates and TA cells obfuscate designs by replacing the functional gates and timing paths with key-controlled cells. Unlike traditional XOR-based key gates, the design's functionality remains structurally and functionally hidden without the need for resynthesizing the netlist. Furthermore, LGA gates and TA cells can be co-inserted such that they can *co-obfuscate* logic and timing paths, i.e., paths that are functionally obfuscated can also be timing obfuscated. In the following lemma, we show that TA, when combined with logic ambiguity, can make the design functionally different from the original unobfuscated design, thereby making it impossible to establish functional equivalence between the two.

Lemma 2: TA cells on logic paths that are driven by LGA gates make the obfuscated design functionally nonequivalent to the original design unless the correct key is applied.

Proof: Consider a circuit that is obfuscated using p LGA gates, such that q internal nets terminating in sequential elements and/or POs are covered by the p LGA gates ($q \leq p$). Let us replace and/or insert q TA cells at these internal nets terminating in sequential elements and/or POs, such that the TA cells are driven by a q-bit key vector, $\overrightarrow{K_{\text{TA},q}}$. As per Lemma 1, the output at each of the q terminating nets will be a function of the input vector ($\vec{I}$) and the p-bit LGA-key vector ($\overrightarrow{K_{\text{LGA},p}}$), such that $O_q = f(\overrightarrow{K_{\text{LGA},p}}, \vec{I})$. For this intermediate functional output (O_q) to be correctly fed into the subsequent logic, the TA cells will have to be initialized with the correct key so that the output of the obfuscated circuit is given by $O_{\text{ALT}} = g(\overrightarrow{K_{\text{TA},q}}, f(\overrightarrow{K_{\text{LGA},p}}, \vec{I}))$, where g represents the subsequent timing obfuscated logic driven

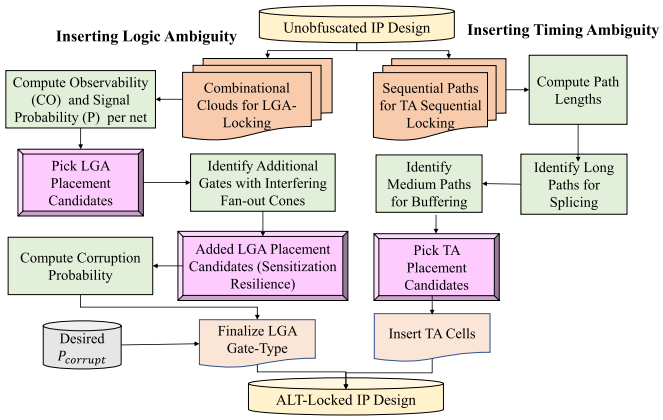


Fig. 6. Overall methodology for ALT-Lock insertion.

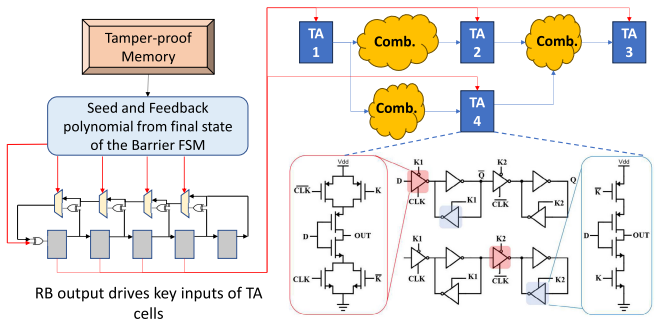


Fig. 7. Dynamic obfuscation of timing cells using an RB.

by the LGA-locked cells and PIs. Assuming that $\overrightarrow{K_{\text{LGA},p}}$ is correct, even if a single bit in $\overrightarrow{K_{\text{TA},q}}$ is incorrect, there is no guarantee that the timing behavior of the obfuscated design C_{ALT} will match the original unobfuscated design, C_{or} (due to insertion guidelines for TA cells mentioned Section III-C2). Thereby, making it impossible to prove the functional equivalence between C_{or} and C_{ALT}. □

From Lemma 2, we can infer that adding TA to paths that are already obfuscated with LGA gates significantly increases the attacker's difficulty in proving functional equivalence between the obfuscated and unobfuscated versions of the circuit. Furthermore, the best design practice is to match the number of LGA gates with the number of TA cells ($q = p$), with both obfuscating part of the same logic paths. However, it is possible to have $q \leq p$ if multiple LGA gates converge to the same net. The overall ALT-Lock insertion methodology, consisting of LGA gate insertion and TA sequential cell insertion, is illustrated in Fig. 6. Note that methods that obfuscate timing paths have been presented before [30] and [31]. Zhang et al. [30] present a method to obfuscate timing through wave pipelining. However, it requires the removal of flip-flops from the design. Furthermore, it cannot obfuscate paths with delays below the clock period. Sweeney et al. [31] do not support logic obfuscation and rely solely on obfuscating timing paths. In addition, neither [30] nor [31] presents a methodology to control output corruptibility.

4) Dynamic Obfuscation of Timing Cells: We develop a method for dynamically obfuscating the state of the IP by driving the key inputs to TA cells using an RB. An RB consists of a linear feedback shift register (LFSR) with a reconfigurable feedback path [20]. The RB can be programmed by loading in the seed and the select lines to the MUXes in the feedback path. The select lines can be set by the designer to achieve any possible LFSR feedback polynomial. The initialization seed can be used to determine the starting sequence of the LFSR. The RB is programmed through a key stored in the tamper-proof memory. The key inputs to the TA cells are driven by the output of the RB. Fig. 7 illustrates the RB-driven dynamic obfuscation of the TA cells. RBs have been previously demonstrated as effective instruments to achieve dynamic obfuscation for securing test paths [20], [21]. In this work, we use RBs to dynamically obfuscate TA cells, therefore using the same method to achieve dynamic obfuscation of timing paths in the design. As will be described later, dynamically changing the constituent timing properties of logic paths enhances the security by achieving resilience against not only combinational but also sequential deobfuscation attacks.

Dynamic obfuscation of the TA cells is achieved upon the assertion of the scan enable pin. Activation of the scan enable signal leads to the key inputs of the TA cells being connected to the output of the RB. As this RB output changes dynamically every clock cycle, this implies that at every clock cycle, the initialization keys for the TA cells change depending on the state of the RB. As a result, the configuration of the TA cell can dynamically change between a flop, a latch, and a buffer, respectively, depending on the variation in the dynamic key sequence generated by the RB. This is illustrated in Fig. 8, where Case 1 represents the static key in unlocked state. During this stage, the TA cells are all configured as flops (blue). Upon the assertion of the scan enable pin, the key inputs to the TA cells are initialized from the dynamically changing output of the RB, which changes the configuration of the logic paths belonging to the IP dynamically with time. Case 2 represents the reconfigured logic paths after the first clock period. Note that TA cells TA1 and TA4 have been reconfigured as latches (yellow). During the second clock period, i.e., Case 3, TA cells TA2 and TA3 are reconfigured as buffers (red). Case 4 represents the third clock period upon the assertion of the scan enable signal. In this case, TA1 is reconfigured as a buffer while TA3 is reconfigured as a latch.

Note that changing the state of the logic paths dynamically in every clock cycle changes the logic state of the IP. This is because changing the nature of sequential elements in a logic path induces changes in the FSM transitions, in some cases even changing the register size of the FSM (when some flops are reconfigured as buffers). As a result, the IP no longer remains functionally correct. Triggering this change in the state of the IP dynamically with time upon the assertion of the scan enable signal denies the attacker oracle access. In the next section, we will discuss how this enhances the resilience of ALT-lock against both oracle-guided and oracle-free attacks.

IV. Security Analysis of ALT-Lock

A. Brute-Force Resilience

Consider an IP that is protected through the insertion of m LGA gates and n TA cells. In such a scenario, the attacker can deobfuscate the circuit by guessing each key combination

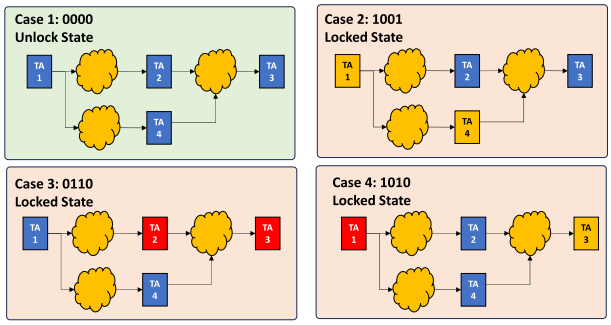


Fig. 8. Changing state of logic paths in IP due to dynamic obfuscation of TA cells. All TA cells are initially flops (blue), but change their functionality to latches (yellow) and buffers (red) depending on the dynamic key sequence of the RB.

and testing it for functional correctness. While each LGA gate consists of one unique key bit, each TA cell consists of two unique key bits. Thus, the total key size is given by $m + 2n$. As only one key among all the key values will be the correct key, the brute-force resilience, which is quantified in terms of the number of attempts the attacker must make in the worst case to unlock ALT-Lock, is given by $t_{\text{bf}} = 2^{m+2n}$. For a total key size of 128 bits, the total number of guesses needed is in the range of 10^{38}, which is prohibitively high.

B. Oracle-Guided Attacks

All the oracle-guided attacks rely on the ability of the attacker to create SAT-based models of the reverse-engineered IP. It is assumed that the attacker has access to the reverse-engineered gate-level netlist of the design excluding the value of the key. The attacker can then eliminate incorrect keys quickly by generating distinguishing input pairs, i.e., inputs that produce different outputs for two different keys, and querying them with the oracle.

Oracle-guided attacks are easily applicable for combinational logic locking as the timing properties of the circuit are assumed to be unaffected. The timing and sequence in which the Boolean inputs are applied to the circuit, and the output responses are sampled from the circuit, are assumed to be known. Therefore, an attacker can simply scan in the DIPs and scan out the responses from the oracle without having to take timing into account.

In ALT-Lock, the timing paths of the locked circuit are also obfuscated. A TA cell could be a latch, buffer, or a flop, thereby making it difficult for the attacker to ensure an equivalent match in the timing and length of the input being applied to the oracle and the locked design.

Theorem 1: Combinational SAT attack cannot identify locked keys in the presence of TA in the locked circuit.

Proof: Let C_{or} be the oracle that performs the transformation C_{or}:$I \rightarrow O$ where $I = \{0,1\}^M$ and $O = \{0,1\}^N$, representing M-bit inputs and N-bit outputs in the Boolean domain $\mathbb{B}$. In other words: $\forall \overrightarrow{X_i} \in I$:$C_{\text{or}}(\overrightarrow{X_i}) = \overrightarrow{Y_i}$, where $\overrightarrow{X_i} \in \mathbb{B}^M$, $\overrightarrow{Y_i} \in \mathbb{B}^N$. Now, consider na ALT-Locked circuit C_{ALT} with combinational LGA-key, $\overrightarrow{K_{\text{LGA}}}$, and TA key, $\overrightarrow{K_{\text{TA}}}$. Consider a combinational SAT attack $\mathcal{A}^{\mathcal{SAT}}$, which unlocks the combinationally obfuscated circuit function $C_{\text{ALT,LGA}}$ to the functionally unlocked circuit C_{or}, i.e., $\mathcal{A}^{\mathcal{SAT}}$:$C_{\text{ALT,LGA}} \rightarrow C_{\text{or}}$. This can be accomplished by extracting the correct key vector $\overrightarrow{K} = \overrightarrow{K_{\text{LGA}}}$ such that the circuit can be unlocked. This is equivalent to solving the quantified Boolean formula (QBF)

$$\text{QBF: } \exists \overrightarrow{K_{\text{LGA}}}\ \forall \overrightarrow{X_i} \in I : C_{\text{ALT,LGA}}\left(\overrightarrow{X_i}, \overrightarrow{K_{\text{LGA}}}, \overrightarrow{Y_i}\right) \wedge C_{\text{or}}\left(\overrightarrow{X_i}, \overrightarrow{Y_i}\right). \quad (1)$$

However, due to TA of the locked circuit, the attacker only has access to reverse-engineered gate-level netlist, C_{ALT}, where both $\overrightarrow{K_{\text{TA}}}$ and $\overrightarrow{K_{\text{LGA}}}$ are unknown. From Lemma 2, we infer that C_{or} is not equivalent to $C_{\text{ALT,LGA}}$ unless the right value of $\overrightarrow{K_{\text{TA}}}$ is specified. Since the timing impact of $\overrightarrow{K_{\text{TA}}}$, which is determined by the timing characteristics of the TA cells and timing requirements of the original design, cannot be specified using a Boolean expression, $\overrightarrow{K_{\text{TA}}}$ cannot be modeled in (1). As a result, without the right value of $\overrightarrow{K_{\text{TA}}}$, it is impossible to verify the correctness of (1). As a result, the key search space will not be pruned using any equivalence class of keys, therefore forcing the attacker to resort to brute-force key guessing. □

C. Oracle-Free Attacks

As demonstrated through Theorem 1, the use of TA cells makes combinational SAT attacks ineffective against ALT-Lock. However, there exist another set of oracle-free attacks that only require access to the locked design and obviate the need for scan access. Such categories of attacks rely on SAT-based sequential deobfuscation by unrolling the locked IP [24], [32]. Furthermore, other sets of deobfuscation techniques rely on training machine learning (ML) models. In this section, we demonstrate the ALT-Lock's resilience against both these category of attacks.

1) Sequential Deobfuscation Attacks: The use of RB-driven dynamic reconfiguration of TA cells in the logic paths of the IP forces the attacker to rely on an unrolling-based oracle-free technique for deobfuscation. This is because the sequential state of the IP changes every clock cycle. Oracle-free model checking-based sequential SAT attacks unroll the states of the locked sequential circuit to identify distinguishing input sequences (DISs) [24], [33]. Let $C_{os}(i, s_o)$ denote a sequential oracle with input i and state register s_o. An encrypted sequential system $c_{es}(i, s_e, k_*)$ is unlocked when there exists a key k_*, such that

$$k_* \in K_* : c_{os}(i, s_o) = c_{es}(i, s_e, k_*) \quad \forall i \in I^{\infty}. \quad (2)$$

Therefore, c_{es} is considered unlocked when (2) is satisfied for all the sequential input traces $\langle i^0, i^1, i^2, \ldots, i^u \rangle \in I^{\infty}$, where u is the max unroll count. Similar to SAT attacks, this is done by applying the input traces to a Miter circuit to eliminate incorrect keys from the search space. Sequential SAT attacks are computationally very expensive and do not scale well for large circuits [34]. A recent variant of this type of attack is KC2 [24]. It uses additional simplification techniques, which include integrating SAT with bounded model checking (BMC), using BDD to simplify circuit representation, and simplifying constraints using key sweeping. This attack has also been modeled for delay locked designs [32].

TABLE I

EVALUATING THE INCREASING COMPLEXITY OF UNROLLING-BASED SEQUENTIAL SAT ATTACK (KC2) ON ISCAS SEQUENTIAL BENCHMARKS LOCKED WITH 8- AND 16-bit RBS

8-bit RB						
Benchmark	#SAT Variables		#SAT Clauses		Total Time (s)	
	XOR-based	ALT-Lock	XOR-based	ALT-Lock	XOR-based	ALT-Lock
s27	276	12969	764	40575	0.015	6.54
s382	16844	24615	52349	76860	55	3600
s713	3162	104203	7868	276141	0.18	8596.43
s1512	36887	49566	90261	136289	9.12	TO
s3384	46922	139152	111931	381248	113.23	TO
16-bit RB						
Benchmark	#SAT Variables		#SAT Clauses		Total Time (s)	
	XOR-based	ALT-Lock	XOR-based	ALT-Lock	XOR-based	ALT-Lock
s27	NA	NA	NA	NA	NA	NA
s382	70685	45000	75391	78990	2597.44	7808.4
s713	98762	114739	8504	310917	0.59	9284.85
s1512	110344	150875	455291	541349	100.7	TO
s3384	168355	195413	226928	226928	1009.8	TO

TABLE II

EVALUATING THE INCREASE IN MAXIMUM UNROLLING DEPTH OF KC2 ATTACK FOR 8- AND 16-bit RBS

Benchmarks	8-bit Keys		16-bit Keys	
	XOR-based	ALT-Lock	XOR-based	ALT-Lock
s27	2	3	NA	NA
s382	6	11	7	43
s713	8	16	8	56
s1512	11	45	14	70
s3384	15	67	20	82

TABLE III

RESULTS FOR THE KC2 ATTACK APPLIED TO BENCHMARKS PROTECTED USING 64- AND 128-bit RBS. TO: TIME-OUT OF 120 h, MEMORY LIMIT: 60 000 Mb

Benchmark	# Flops	RB Size	Runtime (s)	# Clauses
sha3_low	618	64-bit	TO	2.1×10^4
		128-bit	TO	8.44×10^5
gcm_aes	951	64-bit	TO	6.5×10^4
		128-bit	TO	15.5×10^5
MIPS32R	2072	64-bit	TO	1.6×10^4
		128-bit	TO	8.6×10^5
AES	11191	64-bit	TO	4.3×10^5
		128-bit	TO	2.6×10^6
RocketCore	43357	64-bit	TO	9.2×10^5
		128-bit	TO	4.54×10^6

In case of ALT-Lock, the use of RBs in the key initialization path drastically increases the effort for sequential deobfuscation attacks such as KC2 and timingSAT. Since KC2 and other sequential SAT attacks rely on unrolling the circuit, the use of reconfigurable feedback increases the number of frames that need to be unrolled before a DIS can be identified. Furthermore, as the characteristics of the logic paths themselves change during each clock cycle, all the potential path configuration options must be accounted for during each instance of unrolling. Depending on the configuration of every TA cell, there exist four different logic paths that can be created during each unrolling instance (depending on whether the TA cell is unrolled into two types of latch configurations, flop configuration, or buffer configuration). As a result, if the design is unrolled for u instances, and four potential paths need to be analyzed for every unrolled instance, the total number of unrolling candidates is given by 4^u. Different model checking configurations can be used to determine the number of queries and the maximum length of the DIS required to unlock the IP. When the number of queries is bounded, a bounded model checker (BMC) is used. To exhaustively continue unrolling the IP until all the SAT constraints are satisfied, unbounded model checkers (UMCs) are used. When using UMC mode, the number of unrolling rounds is unconstrained and the solver continues unrolling the IP and adding SAT clauses to the solver until all the constraints specified in (2) are satisfied. Note that the state space for the sequential deobfuscation attack increases exponentially depending on the number of unrolled sequences. This makes sequential deobfuscation attacks intractable on ALT-Lock.

Table I shows the comparison between the number of SAT clauses and SAT variables for ISCAS sequential benchmarks locked with traditional XOR-gates and those locked with 8- and 16-bit RBs using ALT-Lock. It is evident that both the attack effort and runtime increase significantly for circuits locked with ALT-Lock. In addition, note that $s27$ is too small to be locked with a 16-bit key. All the experiments were run on an Intel Xeon-Gold Server with 60 GB of primary memory. The increase in maximum unrolling depth of the solver is also presented in Table II. Note that using 8- and 16-bit RBs can significantly increase the complexity of unrolling of the model checker, therefore increasing the attack effort.

We experimentally demonstrate the resilience of ALT-Lock against sequential deobfuscation attacks. We ran KC2 on several OpenCore benchmark circuits protected using ALT-Lock. Every benchmark was protected using a 64- and 128-bit RB, respectively. Note that every output bit of the RB drives a TA cell, and therefore, the number of TA cells used for protection is equal to the size of the RB. Table III shows the result of the KC2 attack variant applied on the different variants of the ALT-Lock architectures. A timeout of 120 h was set for every experiment. Note that the attack times out for all the benchmarks, indicating the resilience of the dynamic obfuscation architecture against sequential deobfuscation attacks. Table III also shows the number of SAT clauses that are added to the solver until time-out occurs for all the dynamically obfuscated benchmarks. The explosion in the number of SAT clauses is another indicator to the difficulty faced by the existing constraint-based sequential deobfusation attacks. Increasing the size of the RB will further increase the number of SAT clauses, therefore increasing the computational complexity at an exponential level. As a result, computational resources will have to also scale exponentially to attempt deobfuscation. As we refer to recent prior work that also shows how increasing the number of SAT clauses also increases the complexity of SAT-based deobfuscation, showing that exponential growth in the number of SAT clauses also increases deobfuscation time exponentially [35]. In [35], the SAT solver is unable to converge to a solution with less than 2×10^4 clauses even after ten days. We conclude that scaling the number and size of RBs beyond 128-bit keys will make SAT-based deobfuscation computationally intractable.

2) ML-Guided Attacks: Oracle-free attacks only use the reverse-engineered netlist of the design to deobfuscate the IP. A wide class of ML-based attacks have been recently proposed that rely on analyzing only the gate-level reverse-engineered netlist of the design to reconstruct the original circuit of the design [6], [8], [36]. These attacks have only been

demonstrated to reverse-engineer combinational XOR gate-based logic-locked netlists that rely on netlist resynthesis after key-gate insertion.

a) GNN-guided netlist reverse engineering (OMLA): OMLA uses graph neural networks (GNNs) to learn the graph-based key embeddings from a combinationally locked circuit. It relies on a large training dataset that must be generated by locking the circuit several times with the same number of combinational key gates and resynthesizing the design. Unlike traditional methods that use XOR-gates for locking the design, ALT-Lock does not require gate-level resynthesis to obfuscate the design. Furthermore, ALT-Lock does not use traditional key gates to achieve locking. It obfuscates combinational logic using custom implementations of LGA gates. Such implementations cannot be modeled by the existing ML-based reconstruction methods that rely on netlist reconstruction using XOR-based key gates. Furthermore, TA cells in ALT-Lock add an additional layer of obfuscation that cannot be modeled using the existing GNN-based netlist reconstruction methods. To reverse-engineer the complete ALT-Locked circuit, the attacker will be forced to partition the entire design into smaller subblocks of combinational logic, with boundaries demarcated by location and placements of TA cells and flip-flops. Furthermore, given that the key configuration of the TA cells is unknown, combinational reconstruction of such partitions will only lead to incomplete key recovery of the ALT-Locked design. Finally, to verify the correctness of combinationally locked gates, the attacker will first have to ensure that the timing paths are deobfuscated, which requires brute-force effort, making such an oracle-guided attacks futile.

b) ML-guided latch unraveling attack: Latch-based logic locking obfuscates certain timing critical paths in the design by inserting key-programmable latches. These latches can be phase-controlled depending on the supplied key, and then be inserted into timing critical paths in the design by duplicating selected flip-flops in those paths followed by retiming those paths [31]. Latch-based locking also relies on the insertion of additional decoy latches to obfuscate the design. An ML-guided Latch Unraveling attack [31] has been proposed; it uses Boolean analysis and ML-guided integer linear programming (ILP) to deobfuscate latch-locked circuits. The attack runs in two phases.

1) *Phase 1:* A sequential graph of the locked design is built. Random Forest classifier is then used to identify randomly inserted logic decoys and simplify the circuit.
2) *Phase 2:* The simplified circuit is fed into an ML-classifier, which outputs the probability of a latch being a delay decoy. These probabilities are used as constraint coefficients for an ILP optimizer whose objective is to assign each sequential cell into its appropriate type (primary/secondary/delay latch).

The key assumption made in this attack is that the sequential graph of a latch-locked circuit must be *two-colorable*, i.e., alternating between primary and secondary latches. Since latch-based locking inserts decoy latches in the design, it leads to the creation of additional false paths in the design, which also alters the colorability of the graph. The ML-guided ILP optimizer can then identify such latches and recolor the sequential graph of the locked circuit, deobfuscating it.

In contrast, ALT-Lock does not insert additional decoy cells, thereby avoiding the creation of false paths. The existing cells such as buffers and flip-flops are replaced with TA cells without the addition of secondary bypass paths that could be identified as potential false paths. The timing arcs within the original path are preserved. Furthermore, we show below that the sequential colorability of a circuit remains preserved in ALT-Lock.

Definition 1: Let $\mathcal{G}_{\text{orig}}(V, E)$ be the sequential graph of a circuit, where V denotes the set of all sequential nodes in the circuit, and E denotes the set of combinational paths (denoted by an edge, $e \in E$) connecting the nodes in V.

Definition 2: Let $\mathcal{G}_{\text{ALT}}(V_{\text{ALT}}, E_{\text{ALT}})$ be the sequential graph of original circuit, locked by replacing existing sequential cells with TA cells and inserting and/or replacing combinational cells with LGA cells.

Lemma 3: $\mathcal{G}_{\text{orig}}(V, E)$ and $\mathcal{G}_{\text{ALT}}(V_{\text{ALT}}, E_{\text{ALT}})$ *are isomorphic.*

Proof: LGA cells are combinational. Therefore, insertion and/or replacement of LGA cells in $\mathcal{G}_{\text{orig}}(V, E)$ does not alter either V or E. Similarly, replacing an existing sequential cell $v \in V$ with a TA cell v_{TA} does not create any additional edges in E. As a result, $V \equiv V_{\text{ALT}}$ and $E \equiv E_{\text{ALT}}$, therefore making $\mathcal{G}_{\text{orig}}$ and $\mathcal{G}_{\text{ALT}}$ isomorphic. □

Theorem 2: TA cell replacement and LGA cell insertion and/or replacement does not alter the sequential colorability of the primary–secondary latches in the unlocked circuit.

Proof: From Lemma 3, we observe that insertion and/or replacement of LGA cells does not alter the isomorphism between $\mathcal{G}_{\text{orig}}$ and $\mathcal{G}_{\text{ALT}}$. Similarly, the replacement of sequential cells with TA cells preserves this isomorphism. Given that $\mathcal{G}_{\text{orig}}$ is two-colorable, and $\mathcal{G}_{\text{ALT}}$ is isomorphic to $\mathcal{G}_{\text{orig}}$; therefore, $\mathcal{G}_{\text{ALT}}$ also remains two-colorable. □

We therefore conclude that ML-guided ILP optimization used in latch unraveling attacks cannot be directly applied against ALT-Lock, making ALT-Lock resilient against such attacks. Furthermore, the insertion of new TA cells in the combinational path of the circuit does not generate false paths, thereby making the identification of logic decoys in Phase 1 of the attack inapplicable. Finally, the insertion of TA cells only in flop-to-flop paths of the circuit ensures that the colorability properties of the original unlocked circuit can be preserved.

3) Scan Deobfuscation Attacks: A recent category of attacks rely on assuming that the scan chains are obfuscated/locked. These attacks (ScanSAT [37], DynUnlock [7]) aim at extracting the seed that is used to dynamically obfuscate scan chains. However, such attacks assume that the attacker is aware of the feedback polynomial that is used to obfuscate the logic paths. In case of ALT-Lock, scan access is restricted and the IP enters a dynamically obfuscated state such that the nature of the logic paths themselves changes with every clock cycle. This along with the use of LFSRs with reconfigurable feedback makes it impossible to identify the correct feedback polynomial required during the first stage of the attack. As a result, ALT-Lock is resilient against attacks.

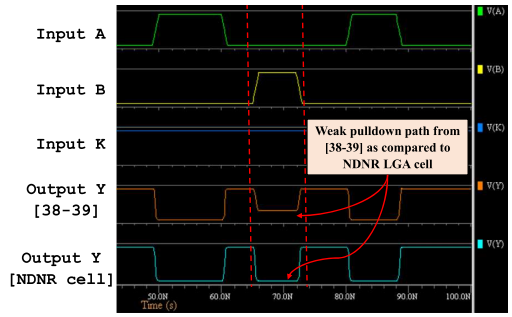


Fig. 9. Output waveforms for the same set of inputs applied to the stacked NDNR cell from [38] and [39] in orange and the NDNR LGA cell from ALT-Lock in turquoise.

D. Comparison With Similar Security Techniques

Methods that use gate-level polymorphism and TA to obfuscate IP have been proposed.

1) Gate-Level Polymorphism: In [40] and [41], the authors develop a technique to design polymorphic gates that can be reconfigured using the value of the supply voltage. Depending on V_{dd}, the gate functionality can be reconfigured between NAND and NOR logic. However the key drawback of such polymorphic logic gate designs is the need for multiple voltage levels for functional operation. This makes it necessary to incorporate additional circuits that can generate multiple levels of power supply voltage in addition to voltage selection circuit for all the key gates. As the LGA gates used in ALT-Lock are completely digital and technology-independent, these standard cells do not suffer from voltage requirement issues as other methods. In [42], the authors have developed a hybrid method to achieve gate-level polymorphism by integrating CMOS gates with memristors. This method suffers from the challenges associated with memristor configuration during initialization. Furthermore, this method suffers from poor write endurance [43]. Finally, the use of memristors does not allow this method to be scalable to purely digital CMOS designs.

In [38] and [39], an NDNR stack-based locking cell is proposed that is similar in structure to the NDNR LGA gate. However, there exist important differences in the placement and connectivity of the key transistors used for reconfiguring the NDNR functionality of the LGA gate. The placement of key gates on both the branches of the nMOS stack leads to a stronger pulldown path for the NDNR LGA cell when compared with prior work in [38] and [39] leading to better performance. Fig. 9 shows comparison of the output for both the NDNR locking cell from [38] and [39] and the NDNR LGA cell from ALT-Lock. Note that for the case where the input signal B transitions from a low value to a high value while signal A is tied to a low value, the NDNR cell provides a strong pulldown path for the output signal's (Y) transition from a high value to a low value when compared with the locking cell from [38] and [39]. Moreover, unlike ALT-Lock, these methods do not use timing obfuscation in combination with functional logic obfuscation, which provides resilience against a wide array of oracle-guided and oracle-free attacks with a very low footprint controlled by the designer.

2) TA Through Timing Camouflage: The goal here is to achieve TA in using wave pipelining [30]. Two data waves are propagated on a logic path at the same time. However, due to the constraints imposed by signal propagation delay across logic paths, this method is only applicable to paths whose signal propagation delay (t_d) is in the range of $T \leq t_d \leq 2T$, where T is the clock period. Another drawback of this method is the need to duplicate combinational logic gates in the paths being locked with wave pipelining. Furthermore, this method remains susceptible to process variations, aging, and small delay defects, which can alter the signal propagation delay across different logic paths over time. This can lead to unanticipated locking of the circuit if the wave pipelining constraints are violated due to aging and timing degradation of logic paths. In contrast, the TA cells in ALT-Lock can be implemented for at-speed synchronous logic designs without path delay constraints while making the method robust against process variations.

3) Latch-Based Locking: As discussed above, latch-based locking uses key-programmable latches to obfuscate IP [31]. Additional decoy latches act as both logic decoys and delay decoys. This method leads to the creation of false paths, which can then be exploited by the attacker to reconstruct the locking keys, as shown above in the ML-guided latch unraveling attack. As highlighted in the previous subsection, ALT-Lock offers several advantages over traditional latch-based locking, without incurring the addition of false paths and preserving the properties of sequential graphs. In addition, dynamic obfuscation using RBs ensures that ALT-Lock remains resilient against unrolling attacks. Furthermore, we show in Section V, that TA cells are more area-efficient than the key-programmable latches.

4) Delay Locking: In [44], the authors use tunable delay key gates (TDKs) to obfuscate the timing profile of combinational paths. Applying the correct key to the TDK gate, which comprises a conventional key gate (XOR/XNOR gate) paired with a tunable delay buffer, satisfies the timing constraints of the obfuscated combinational path. Delay locking has been shown to be vulnerable against SAT-based attacks such as TimingSAT [32]. The timing information of every gate in the design is thus embedded in SAT formulation through a characterization step carried out by the foundry. The correct key can then be reconstructed by querying the oracle and generated DIPs like the traditional SAT attack. It should be noted that while delay locking remains vulnerable to TimingSAT attack, ALT-Lock offers natural resilience against such attacks due to the following reasons.

1) Timing profile of a path locked by a TA cell cannot be characterized unless an assumption is made about its key.
2) Embedding the timing profile of delay obfuscated paths in delay locking assumes that the gate-level functionality of the circuit is known before-hand. This is not true for paths locked using TA cells, which could be functionally different depending on the supplied key.
3) ALT-Lock is resilient against oracle-guided attacks including variants of the SAT-attack (see Theorem 1).
4) Dynamic reconfiguration of TA cells in ALT-Lock makes unrolling attacks such computationally intractable for large enough keys (see Section IV-C).

TABLE IV
EVALUATING THE IMPACT OF NONSCANNED TA CELLS ON FAULT COVERAGE AND THE EFFECTIVENESS OF SEQUENTIAL ATPG IN RECOVERING THE LOSS IN FAULT COVERAGE

IP	Baseline Fault Coverage (%)	Initial Pattern Count	Reduced Fault Coverage (%)			Coverage Recovered using Sequential ATPG (%)			New Pattern Count			Percentage of Original coverage Recovered (%)		
			32	64	128	32	64	128	32	64	128	32	64	128
GPS	93.81	356	92.17	86.74	81.22	93.28	92.33	91.11	357	395	468	99.43503	98.42234	97.121842
FIR	95.67	160	90.52	83.6	-	92.51	91.85	-	203	240	-	96.69698	96.00711	-
IIR	87.98	193	82.46	77.23	75.69	87.62	82.56	79.84	250	268	302	99.59082	93.83951	90.7478972
SHA256	99.54	316	99.07	97.825	95.75	99.32	98.95	98.77	402	454	490	99.77898	99.40727	99.2264416

E. Implications on Testability

1) Manufacturing Testing: Scan-based manufacturing test can be done in an untrusted setting on the deactivated IP (without loading the tamper-proof memory). Structural testing of the LGA cell will be no different than any other standard cell. Furthermore, the TA cell has key-controlled feedback paths that could be tested in all four configurations without the use of additional scan-based hardware overhead. System-level testing can be performed by loading dummy key values for both LGA gates and TA cells. As TA cells may not be scan-enabled (to save area overhead), there will be some loss in fault coverage due to the replacement of certain scan-enabled flip-flops in the design with TA cells. Using Mentor Tessent, we experimentally evaluate the percentage loss in fault coverage due to the replacement of a certain number of scan cells in the design with TA cells across a range of CEP benchmark circuits (see Table IV). We replace 32, 64, and 128 scan cells in the design with TA cells and observe the corresponding loss in fault coverage. We then show that using sequential ATPG can successfully recover on average over 95% of the original fault coverage, thereby showing that TA cell insertion can be done with minimal impact on fault coverage. It should also be noted that the experiments are conservative in assuming that all the TA cell insertions are scan cell replacements, which is unlikely to be the case in practice. Note that for the FIR IP, making 128 replacements is not possible due to the size of the scan chains.

2) IP Activation: Due to the globalization of the fabrication process, it is reasonable to assume that the IP will be fabricated in an untrusted environment. As a result, the fabricated IP will be shipped to the design house for activation, i.e., for loading the tamper-proof memory. Since activation of the IP is done in a trusted setting, the design house can also perform additional testing using precomputed test vectors to ensure that the chip has not been tampered with. The integrity of the dynamic obfuscation scheme can be ascertained by deliberately applying incorrect keys and performing scan operation. Additional testing can also be done to detect Trojans [45].

3) In-Field Testing: Since in-field test is also a form of structural test, it can be done on a deactivated/dysfunctional IP. ALT-Lock does not require modification of scan cells to suit the needs for testing either LGA gates or TA cells. During in-field test, the test patterns and the test response signatures can be evaluated and stored based on the dynamic sequence of keys generated by the RB. Since these signatures are computed for the compacted test responses, they will not reveal internals of the IP or facilitate any form of partial oracle-guided attack.

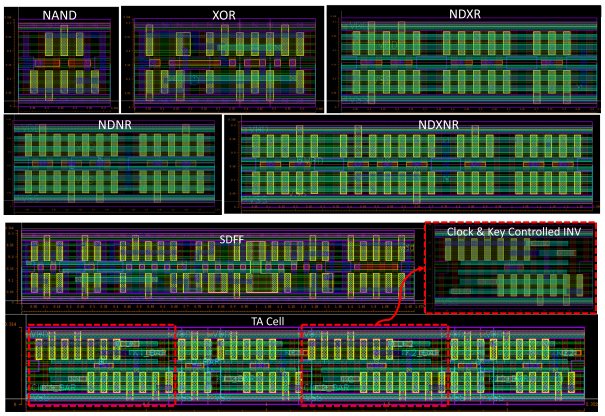


Fig. 10. LGA gates (NDNR, NDXR, and NDXNR) and TA cell implemented in the ASAP 7-nm technology. Two-input NAND, two-input XOR, and SDFF cells from ASAP7 PDK for reference.

TABLE V
EVALUATING THE PPA IMPACT OF THE LGA GATES AND TA CELL IMPLEMENTED USING ASAP7 PDK

Standard Cell	X Dim. (um)	Y Dim. (um)	Area (μm^2)	Delay (ns)	Power (μW)
XOR	0.684	0.314	0.214776	1.16	2.12
NDNR	0.757	0.314	0.237698	2.17	6.87
NDXR	0.918	0.314	0.288252	2.98	4.39
NDXNR	1.296	0.314	0.406944	1.82	6.85
SDFF	1.673	0.314	0.525322	4.09	17.45
TA Cell	2.32	0.314	0.72848	6.09	30.32

V. OVERHEAD ANALYSIS OF ALT-LOCK

In this section, we evaluate the overhead associated with the ALT-Lock scheme. The layout for all the LGA cells (including NDNR, NDXR, and NDXNR along with the TA cell) is carried out using Cadence Virtuoso and the ASAP7 7-nm PDK. Library characterization including area, power, and delay numbers are then obtained from the postlayout files and through SPICE simulations. For delay evaluation of standard cells, a fan-out of four inverters (FO4 load) is used as the parasitic load.

The two-input XOR gate from the ASAP7 PDK is used as the control for evaluation of the area, power, and delay overheads for the LGA cells. Since two-input XOR gates are widely used in traditional logic locking, it is a good baseline to perform a fair comparison. Similarly, the scan-based D flip-flop (SDFF) from the ASAP7 PDK is used as the baseline for overhead evaluation of the TA cell. Fig. 10 shows the standard cell layouts of all the LGA gates and the TA cell implemented using the ASAP7 PDK. Two-input NAND, XOR, and the SDFF cell are also shown from the same PDK for comparison.

TABLE VI

COMPARATIVE PERCENTAGE OVERHEAD RESULTS FOR THE PPA OF (a) TA SEQUENTIAL CELL WITH LATCH-BASED LOCKING [31] AND (b) ALT-LOCK SCHEME WITH DIFFERENT KEY SIZES USED FOR LOCKING CEP BENCHMARK IPS

Locking Cell	% Overhead		
	Area	Delay	Power
TA Cell	38.0	9.20	42.45
Prior Work [31]	86.0	11.0	64.32

(a)

Benchmark IP	ALT-Lock Key Size in bits (K_{NDXNR}, K_{TA})	Total Key Size in bits ($K_{NDXNR} + K_{TA}$)	% Overhead		
			Area	Delay	Power
FIR IP	(64, 64)	128	0.51	0.15	0.40
	(128, 128)	256	0.73	0.25	0.66
	(512, 512)	1024	1.25	0.43	0.85
IIR IP	(64, 64)	128	0.69	0.10	0.33
	(128, 128)	256	0.82	0.23	0.66
	(512, 512)	1024	2.02	0.53	1.95
GPS IP	(64, 64)	128	0.71	0.12	0.89
	(128, 128)	256	1.31	0.42	1.15
	(512, 512)	1024	2.18	1.15	2.45

(b)

Table V shows the area, delay, and power consumption of the LGA gates and the TA cell. When compared with a traditional XOR-based key gate, the NDNR LGA gate has a 10.67% higher area. Similarly, the NDXR LGA gate has an area overhead of 34.21%, and the NDXNR LGA gate has an 89.47% area overhead. As LGA gates replace regular logic gates in place, they do not incur the delay associated with an additional XOR-gates that must be inserted in the combinational paths as additional key gates. Table VI(a) presents the overhead of the TA sequential cell and prior work on latch-based logic locking compared with a standard flip-flop [31]. Note that TA cell is quite small compared with the existing latch-based sequential locking schemes [31], thereby making it more attractive for timing obfuscation. Table VI(b) shows the PPA overhead for the ALT-Lock scheme, consisting of the largest LGA gate (NDXNR) paired with the TA cell, used to lock various IPs of the common evaluation platform (CEP) benchmark suite. CEP is a popular benchmark suite used for evaluation of hardware overheads for security solutions [20], [21]. When compared with other standard cells in the design, it is important to note that although the LGA gates are bigger than regular standard cells, their delay overhead is not significant when we consider combinational logic paths containing tens of standard cells. Furthermore, the total key size when used to lock practical circuits (containing tens of thousands of standard cells) is unlikely to exceed 500 [23], which means that the overall impact of the ALT-Lock scheme will be minimal. The designer can decide which type of LGA gate to insert based on the PPA budget available for the given design. When replacing a NAND gate, NDXR will provide a corruption rate of 25% with an area overhead of about 35%. Similarly, using NDNR will achieve an output corruption rate of 50% with only 10% area overhead. NDXNR will achieve the highest corruption at 75% while also using 89%. The cost of replacement can therefore be used to guide the insertion process. Although the individual LGA overhead may seem significant when compared with traditional standard cells, the overhead at the IP level is very small as can be seen from Table VI(b).

VI. CONCLUSION

We have demonstrated a logic and TA-based IP obfuscation method, ALT-Lock. We have presented a methodology for insertion of logic ambiguity (LGA)-based gates and TA-based sequential cells. We have demonstrated security against both oracle-guided and oracle-free attacks. In addition, we have shown the PPA of our overhead.

ACKNOWLEDGMENT

The authors acknowledge the contribution of Akshay Vyas in standard-cell characterization of LGA and TA cells.

REFERENCES

[1] 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.

[2] B. Tan et al., "Benchmarking at the frontier of hardware security: Lessons from logic locking," 2020, *arXiv:2006.06806*.

[3] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Int. Symp. Hardw. Orient. Secur. Trust (HOST)*, 2015, pp. 137–143.

[4] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.

[5] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "On the security of sequential logic locking against oracle-guided attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 42, no. 11, pp. 3628–3641, 2023.

[6] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1602–1606, Mar. 2022.

[7] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking scan chains obfuscated using dynamic keys," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 270–273.

[8] P. Chakraborty, J. Cruz, and S. Bhunia, "SURF: Joint structural functional attack on logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 181–190.

[9] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1601–1618.

[10] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in IC piracy with test-aware logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 961–971, Jun. 2015.

[11] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *Proc. IEEE Comput. Soc. Annu. Symp.*, Jul. 2018, pp. 405–410.

[12] A. Chakraborty et al., "Keynote: A disquisition on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 1952–1972, Oct. 2020.

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

[14] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr. 2020.

[15] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust*, May 2017, pp. 95–100.

[16] 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.

[17] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "SCRAMBLE: The state, connectivity and routing augmentation model for building logic encryption," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 153–159.

[18] M. Fyrbiak et al., "On the difficulty of FSM-based hardware obfuscation," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 293–330, Aug. 2018.

[19] A. Patooghy, E. Aerabi, H. Rezaei, M. Mark, M. Fazeli, and M. A. Kinsy, "Mystic: Mystifying IP cores using an always-ON FSM obfuscation method," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 626–631.

[20] J. Talukdar, S. Chen, A. Das, S. Aftabjahani, P. Song, and K. Chakrabarty, "A BIST-based dynamic obfuscation scheme for resilience against removal and Oracle-guided attacks," in *Proc. IEEE Intl. Test Conf.*, Oct. 2021, pp. 170–179.

[21] J. Talukdar, A. Chaudhuri, and K. Chakrabarty, "TaintLock: Preventing IP theft through lightweight dynamic scan encryption using taint bits," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2022, pp. 1–6.

[22] R. Karmakar and S. Chattopadhyay, "On securing scan obfuscation strategies against ScanSAT attack," in *Proc. 21st Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2020, pp. 213–218.

[23] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting all logic locking attacks: Dishonest Oracle with truly random logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 9, pp. 1740–1753, Sep. 2021.

[24] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 534–539.

[25] J. Talukdar, A. Chaudhuri, J. Kim, S. K. Limt, and K. Chakrabarty, "Securing heterogeneous 2.5D ICs against IP theft through dynamic interposer obfuscation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Apr. 2023, pp. 1–2.

[26] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. 48th ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, Jun. 2011, pp. 333–338.

[27] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, "The key is left under the mat: On the inappropriate security assumption of logic locking schemes," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 262–272.

[28] Y. Zhong and U. Guin, "Complexity analysis of the SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 42, no. 10, pp. 3143–3156, 2022.

[29] D. Duvalsaint and R. D. S. Blanton, "Characterizing corruptibility of logic locks using ATPG," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2021, pp. 213–222.

[30] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 91–96.

[31] J. Sweeney, V. M. Zackriya, S. Pagliarini, and L. Pileggi, "Latch-based logic locking," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, Dec. 2020, pp. 132–141.

[32] M. Li, K. Shamsi, Y. Jin, and D. Z. Pan, "TimingSAT: Decamouflaging timing-based logic obfuscation," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[33] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 33–40.

[34] T. Meade, Z. Zhao, S. Zhang, D. Pan, and Y. Jin, "Revisit sequential logic obfuscation: Attacks and defenses," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[35] S. M. Rahman et al., "Security assessment of dynamically obfuscated scan chain against oracle-guided attacks," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–27, 2021.

[36] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 56–61.

[37] L. Alrahis et al., "ScanSAT: Unlocking static and dynamic scan obfuscation," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 4, pp. 1867–1882, Oct. 2021.

[38] K. Juretus and I. Savidis, "Reduced overhead gate level logic encryption," in *Proc. Int. Great Lakes Symp. VLSI (GLSVLSI)*, May 2016, pp. 15–20.

[39] I. Savidis and K. Juretus, "Reduced overhead gate level logic encryption," U.S. Patent 11 282 414, Mar. 22, 2022.

[40] R. Ruzicka, L. Sekanina, and R. Prokop, "Physical demonstration of polymorphic self-checking circuits," in *Proc. 14th IEEE Int. On-Line Test. Symp.*, Jul. 2008, pp. 31–36.

[41] R. Ruzicka and V. Simek, "NAND/NOR gate polymorphism in low temperature environment," in *Proc. IEEE 15th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2012, pp. 34–37.

[42] A. Rezaei, J. Gu, and H. Zhou, "Hybrid memristor-CMOS obfuscation against untrusted foundries," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 535–540.

[43] M. Lanza et al., "Standards for the characterization of endurance in resistive switching devices," *ACS Nano*, vol. 15, no. 11, pp. 17214–17231, Nov. 2021.

[44] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction," in *Proc. 54th ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[45] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojan detection," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2009, pp. 396–410.

Jonti Talukdar received the B.Tech. degree in ECE from Nirma University, Ahmedabad, India, in 2018, and the M.S. and Ph.D. degrees in ECE from Duke University, Durham, NC, USA, in 2020 and 2024, respectively. His Ph.D. Thesis was on IP security for 2.5-D/3-D HI systems.

His research interests include intersection of hardware security, test, and applied machine learning for silicon security, health, and lifecycle management.

Woo-Hyun Paik received the bachelor's and master's degrees in chip design from Korea University, Seoul, South Korea, in 2012 and 2014, respectively.

He was a Visiting Researcher with Duke University, Durham, NC, USA, and expanded his area to hardware security. He is a Memory Chip Designer. His research interests include 3-D memory architecture, hardware security, and design automation.

Eduardo Ortega received the B.A./B.S. degree in integrated engineering from the University of San Diego, San Diego, CA, USA, in 2020. He is currently working toward the Ph.D. degree at the Ira A. Fulton Schools of Engineering, School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ, USA.

He is a Fulton Fellow of the Ira A. Fulton Schools of Engineering, School of Electrical, Computer, and Energy Engineering, Arizona State University.

Krishnendu Chakrabarty (Fellow, IEEE) received the B.Tech. degree from Indian Institute of Technology at Kharagpur, Kharagpur, India, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 1992 and 1995, respectively.

He is currently a Fulton Professor of Microelectronics with the School of Electrical, Computer and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA. He is also the Director of the ASU Center on Semiconductor Microelectronics and the CTO of the Department of Defense Microelectronics Commons Southwest Advanced Prototyping (SWAP) Hub. His current research interests include design-for-test of 2.5-D/3-D ICs and heterogeneous integration, hardware security, AI accelerators, microfluidic biochips, and AI for healthcare.

Dr. Chakrabarty is a fellow of ACM and AAAS and a Golden Core Member of the IEEE Computer Society.