Does logic locking work with EDA tools?

Zhaokun Han
Texas A&M University
hzhk0618@tamu.edu

Muhammad Yasin
Texas A&M University
myasin@tamu.edu

Jeyavijayan (JV) Rajendran
Texas A&M University
jeyavijayan@tamu.edu

Abstract

Logic locking is a promising solution against emerging hardware security threats, which entails protecting a Boolean circuit using a "keying" mechanism. The latest and hitherto unbroken logic-locking techniques are based on the "corrupt-and-correct (CAC)" principle, offering provable security against input-output query attacks. However, it remains unclear whether these techniques are susceptible to structural attacks. This paper exploits the properties of integrated circuit (IC) design tools, also termed electronic design automation (EDA) tools, to undermine the security of the CAC techniques. Our proposed attack can break all the CAC techniques, including the unbroken CAC_{rem} technique that 40+ hackers taking part in a competition for more than three months could not break. Our attack can break circuits processed with any EDA tools, which is alarming because, until now, none of the EDA tools can render a secure locking solution: logic locking cannot make use of the existing EDA tools. We also provide a security property to ensure resilience against structural attacks. The commonly-used circuits can satisfy this property but only in a few cases where they cannot even defeat brute-force; thus, questions arise on the use of these circuits as benchmarks to evaluate logic locking and other security techniques.

1 Introduction

1.1 Security Concerns in the IC Supply Chain

Integrated circuits (ICs) are used in virtually all modern electronic systems. IC design and fabrication involves several stages that are highly automated using electronic design automation (EDA) tools. Traditionally, a company would perform these steps in-house. However, the complexity of ICs has grown enormously, necessitating the use of highly-specialized foundries that cost beyond $10 billion [1]. Many companies such as Apple operate fabless and outsource IC fabrication and other services to offshore vendors. Apple procures intellectual property (IP) cores from IP vendors including Arm, delegates IC fabrication to TSMC or Samsung, and deputes product assembly/test services to Foxconn [2].

In a globalized supply chain, the untrusted entities may obtain a design netlist[1], a chip layout, or a manufactured IC. This may lead to threats such as IP piracy, counterfeiting, reverse engineering (REing), overbuilding, and insertion of hardware Trojans [3]. IP piracy entails malicious entities illegally using IPs. A foundry can manufacture additional ICs to sell at lower profit margins. End-users can conduct piracy by REing an IC to extract the design netlist or other design/technology secrets. REing of an IC involves peeling off the package of an IC, etching the IC layer-by-layer, imaging each layer, and stitching the images together to extract the design netlist. IP piracy issues alone incur annual losses up to $4 billion for the semiconductor industry [4].

1.2 Countermeasures against Piracy

The countermeasures developed to foil hardware security threats include watermarking [5], metering [6],logic locking [7–11], split manufacturing [12], and camouflaging [13]. Logic locking introduces additional protection logic into a design that "locks" the design's functionality with a secret key. The design is unlocked by loading the secret key to the on-chip tamper-proof memory. The design produces correct outputs only upon loading the correct key.

In contrast to most other countermeasures, logic locking can protect against both the untrusted foundries and end-users. Moreover, logic-locking techniques can be enacted earlier in the design flow, without any modifications of most design/fabrication processes. Consequently, they are currently being developed in the academic context, as well as in the semiconductor industry. Logic locking has been incorporated into the Trust Chain framework of Mentor Graphics [14, 15]. Defense Advanced Research Projects Agency (DARPA) has included logic locking as a defense technique in its latest multi-million dollar Automatic Implementation of Secure Silicon (AISS) program. The program aims at building scalable

[1] A netlist is a Boolean circuit in the form of logic gates and wires.

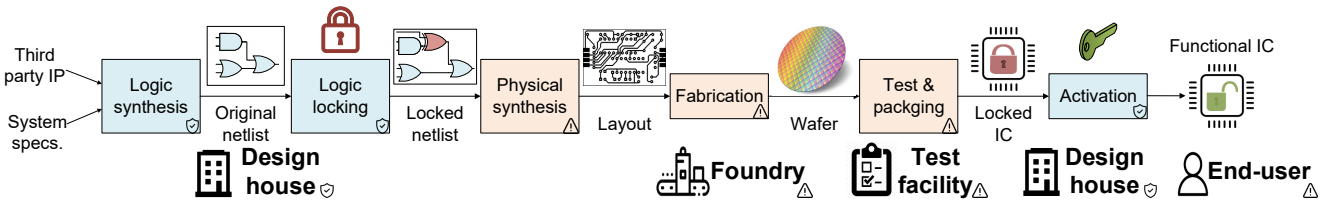

Figure 1: The IC design flow incorporating logic locking. The orange (blue) regions denote untrusted (trusted) entities.

hardware defenses against IC supply chain attacks through collaboration between universities and leading semiconductor and defense companies [16].

1.3 Applications of Logic Locking

1. **Thwarting piracy and overbuilding.** Logic locking, as initially intended, thwarts piracy and overbuilding [7, 17, 18]. Pirated designs overproduced by a malicious foundry are useless without the secret key held by the designer.

2. **Anti-reverse engineering.** To prevent REing, commercial foundries (e.g., TSMC) and companies (e.g., Syphermedia [19] and Mentor Graphics [20]) produce camouflaged designs. The gates in camouflaged designs look alike but implement different Boolean functions [13]. Upon REing, an attacker cannot infer the functionality of these gates [21]. Camouflaging and logic locking are transformable, i.e., the attacks and defenses developed for logic locking can apply to camouflaging and vice versa [22].

3. **Upgradable processors**. A decade ago, Intel introduced the notion of "upgradable processors" for their Sandy Bridge processors [23]. Once the customer pays an additional amount, Intel *unlocks* certain features. This technique can be deployed with fuses or software. However, these methods are not secure; e.g., an attacker could burn the fuses to unlock those features without paying Intel. Researchers have shown that logic locking can securely enable differentiation features such as processor performance settings [24], GPU cache configurations [25], and instruction sets of hardware accelerators [26].

4. **Parametric locking.** Logic-locking techniques can hide both the functionality and the parametric behavior of the IC [27]. In delay locking, the key determines the output of a circuit and its timing profile [28]. An incorrect key may lead to timing violations, forcing attackers to operate the circuit at a frequency lower than the desired one.

5. **Protecting analog circuits.** Analog and mixed-signal IPs (with digital and analog elements) are the most counterfeited semiconductor product [29]. Recently, logic locking has been used to protect AMS circuits such as band-pass filters, operational transconductance amplifiers, and voltage regulators [30, 31]. The digital part of the circuit is locked, and only the correct key fine-tunes the tunable analog components, e.g., resistors and capacitors, to meet the specifications.

All these applications hinge upon the assumption that **the underlying logic-synthesis algorithm is secure.** By exposing the vulnerabilities of the state-of-the-art "secure" logic-locking techniques that rely on the conventional EDA tools, this paper invalidates the stated assumption.

1.4 State-of-the-art Logic Locking

Logic locking was first introduced in 2008 [7]. The earlier techniques focused on protecting designs so that an incorrect key ensures an incorrect output while incurring minimal power, performance, and area (PPA) overhead. However, [8] showed how an attacker could retrieve the key by having access to: 1) a locked netlist obtained from a malicious foundry or by REing a chip, and 2) a functional chip (aka oracle) obtained from the market which acts as an oracle. By using Boolean satisfiability (SAT) and satisfiability modulo theories (SMT) solvers, attackers can improve their efficacy [9, 11, 41, 42]. We refer to these attacks as *input-output query attacks (I/O attacks)*, as they analyze the locked netlist and repeatedly query the oracle to find the correct key.

To thwart the I/O attacks, researchers used point functions as the protection logic [38,39], since point-based functions are cryptographically obfuscatable and has become attractive for logic locking [43]. One set of techniques that use point functions are the *corrupt and correct (CAC)* techniques [10,40,44]. They "strip" the point function(s) from the target Boolean function [10, 40]. The circuit-to-be-locked is first "corrupted" and then is "corrected" only on applying the correct key, as shown in Fig. 2. An attacker lacking the correct key will obtain only a "corrupted" design. Multiple CAC techniques have been invented since 2016[2], offering 1) trade-offs among the amount of corruption and overhead, and 2) mathematically-proven security against I/O attacks [46]. The security reasoning is that the probability of finding a "hidden" point function in a large Boolean space just by querying the oracle is exponentially small in the input size of the function. Consequently, an attacker has to query the oracle exponential times in the key size.

[2]CAC techniques have also been referred to as stripped functionality logic locking (SFLL) in the literature [10, 27, 40, 45].

Table 1: State-of-the-art logic locking attacks and defenses. × denotes a successful attack. ✓ denotes a successful defense.

Defense	I/O attacks			Structural attacks			
	Sensitization [8]	SAT, SMT [9, 11]	AppSAT, 2-DIP [32, 33]	SPS, ATR [34, 35]	FALL [36]	SAIL [37]	**SPI (proposed)**
XOR-based (random, strong, fault-based, LUT-based [7, 8, 18])	×	×	×	✓	✓	×	×
Point-function (AND-tree, SARLock, Anti-SAT [35, 38, 39])	✓	✓	✓	×	✓	✓	×
CAC_{hd}, CAC_{flex} [10]	✓	✓	✓	✓	×	✓	×
CAC_{rem} [40]	✓	✓	✓	✓	✓	✓	×

1.5 Scope and Contributions

All CAC techniques rely on the assumption that the underlying logic-synthesis tools used by the semiconductor industry can effectively "hide" this point function in the target design. Unfortunately, this is not the case. Traditionally, EDA tools focus only on the PPA metrics and not on security. Thus, their optimizations expose the hidden point function, which can be removed by *structural attacks* [35, 36, 47]. An attacker in the foundry or an end-user can RE an IC, analyze the structure of the netlist, and find the hidden point function to break logic-locking techniques. All the CAC techniques except CAC_{rem} are vulnerable to structural attacks, as listed in Table 1.

Over the past decade, there have been many logic-locking attacks and defenses— basically a kind of "cat and mouse" game — without any concrete notion of security against structural attacks [35, 36, 47–50]. Every technique is considered secure until someone develops a heuristic that can break it. This paper raises and addresses the following related questions: (i) What is the fundamental theory of structural attacks? (ii) What makes logic-synthesis tools render designs insecure? (iii) How can we fix the EDA tools to help logic locking?

To this end, we first develop a unified attack called a *sparse prime implicant (SPI) attack*, which considers logic-synthesis principles and breaks thus far unbroken techniques. We also identify a security property to ensure the resilience against structural attacks. The most important result of this paper is to show that none of the commonly-used benchmarks can simultaneously satisfy this property and achieve a reasonable key size, calling into question the use of these circuits as benchmarks to evaluate logic-locking techniques.

A competition on logic locking. Recently, New York University held a competition on logic locking [51]. CAC_{rem} circuits were fielded for participants to develop new attacks and break them. 18 teams with 40+ hackers from 15 universities from across the world competed over three months. Most teams have had prior publications on logic-locking techniques and thus can be considered to have a reasonable level of expertise. (Un)Fortunately, over the span of three months, none of the teams were able to break CAC_{rem} even with a key size of 80. No one has reported breaking these circuits since they were released about a year ago. The proposed SPI attack can break this unbroken technique within seconds, and the competition organizers verified our attack's effectiveness on the competition circuits. Additionally, they also provided our research team with three harder circuits with larger key sizes (up to 195); our attack can break these circuits as well. Therefore, the contributions of this paper are as follows:

1. We demonstrate that the state-of-the-art logic-locking technique is vulnerable to structural attacks. Our proposed SPI attack circumvents CAC_{rem} and other variants of the CAC techniques within seconds (see Section 3).

2. We demonstrate the effectiveness of the SPI attack on circuits processed by various EDA tools targeting at both application-specific integrated circuit (ASIC) and field-programming gate arrays (FPGA) implementation[3]. We deploy five industrial tools, Cadence Genus [53], Synopsys Design Compiler [54], Synopsys Synplify [55], Xilinx Vivado [56], Mentor Graphics Precision RTL [57], and one academic tool ABC [58], to synthesize the circuits (see Section 4).

3. We develop a security property for ensuring security against the SPI attack and other structural attacks. The property, referred to as Dist2, is based on the notion of "distant" prime implicants (see Section 5).

4. We examine several industrial and academic circuits to determine how well they satisfy the Dist2 property. The benchmarks include several circuits from the `ITC'99` benchmark suite, controllers of an `ARM Cortex-M3` microprocessor [59], and a `GPS` module as part of Common Evaluation Platform for evaluating hardware security solutions [60]. An alarming finding in this paper is that all commonly-used circuits fail to satisfy the Dist2 property with a reasonable key size, thus making them unsuitable for any logic-locking technique.

The code of the SPI attack is available at `https://seth.engr.tamu.edu/software-releases`.

[3] Logic locking has also been used to protect FPGA bitstreams [52].

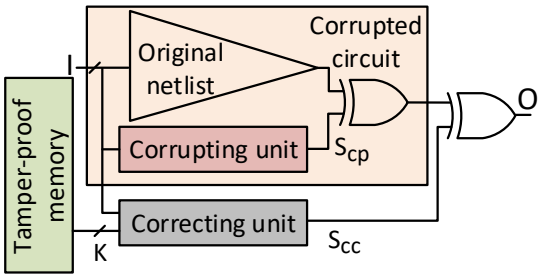

Figure 2: A CAC circuit comprises a corrupted circuit and a correcting unit. The correction happens only for the correct key [10, 27, 40, 45].

2 Background and Related Work

2.1 Threat Model

In this paper, we follow the standard locking threat model adopted in the literature [8–11].

The attacker. The design house and the IC design tools are trustworthy. The untrusted entities are the foundry, the test facility, and the end-user, as highlighted in Fig. 1. The attacker can obtain a locked netlist either by overbuilding or REing. The attacker's objective is to identify the secret key from the locked netlist and activate the design.

Capabilities of an attacker. An attacker has access to a 1) REed netlist and 2) a functional IC, i.e., an IC with the correct key. A foundry can extract the required netlist from the layout files, whereas an end-user can obtain it via REing the IC. The attacker buys the functional IC from the market. The IC acts as an oracle: the attacker can apply input patterns and observe the correct outputs[4].

2.2 Logic Locking Attacks and Defenses

We now explain the relationship between logic-locking attacks and defenses. In particular, we describe the CAC techniques in detail since they offer provable-resilience against most existing attacks, such as SAT [9, 41], sensitization [8], approximate [32, 33], and SMT [11] attacks. For recent and detailed surveys on logic locking, please refer to [46, 49, 61, 62].

XOR-based locking. The earliest logic-locking technique inserts XOR/XNOR key gates in a circuit. The objective is to ensure that incorrect keys produce an incorrect output [7]. The sensitization attack breaks this defense by computing input patterns that propagate the values of keys to outputs [8].

I/O attacks. The sensitization attack spawned a series of subsequent I/O attacks that rely on the output of a black-box oracle to filter out unlikely key candidates. The SAT attack uses a SAT solver to weed out incorrect keys [9]. The SMT attack uses an SMT solver for the same purpose [11].

[4] An input value is also referred to as an input pattern in the literature.

Point-function-based locking[5]. When a point function, such as a comparator whose one input is the key and the functional input, is XORed with the original circuit, the output is inverted/corrupted only for one input pattern for any incorrect key. Thus, point function-based techniques, such as SARLock [38] and Anti-SAT [39, 63], thwart any I/O attack because the probability to find the input patterns (i.e., the key) that corrupt the outputs is exponentially small in the input size. Approximate I/O attacks, such as AppSAT [32] and 2-DIP [33], focus on retrieving the best set of keys for a given time limit.

Structural attacks. A drawback of point-function techniques is that the point-function can be easily identified and removed by white-box structural attacks. For example, the signal probability skew (SPS) attack uses signal probabilities to locate and remove large AND gates and/or comparators in Anti-SAT and SARLock [34]. The AND-tree removal (ATR) attack achieves the same by analyzing the gates in the locked netlist [35]. The bypass attack restores original functionality by adding a bypass circuitry around the locked circuit [48].

CAC locking defends against both I/O and structural attacks [10, 40, 45]. Fig. 2 shows the architecture of CAC techniques. The original circuit is initially corrupted by XORing it with S_{cp}, the output of the *corrupting unit*, which can be a point function. The output of the *correcting unit*, S_{cc}, restores the correct output on applying the correct key. The input patterns for which the circuit output is corrupted are known as *protected input patterns (PIPs)*. The corrupted circuit is created by XORing the original circuit with a *corrupting unit* that hard-codes the PIP(s). When a single PIP needs to be protected, the correcting unit is realized as a comparator with the key as one input and the functional inputs as the other; the secret key is the same as the PIP in this case. Only when the correct key is applied at the key inputs of the correcting unit, the output is corrected. For an incorrect key, the correcting unit does not correct the corrupted output for precisely one input pattern. Thus, a high resilience against the SAT attack can be achieved similar to point-function-based locking.

The CAC techniques differ in the construction of the corrupted circuit. These techniques allow trade-offs among the security level against different attacks, the number of PIPs, and the PPA overhead. For example, the higher the number of PIPs, the higher will be the error-rate observed at circuit output. CAC_{flex} allows the designer to specify the set of PIPs. The correcting circuit is implemented as a lookup table. Each PIP can be considered as a secret key for the circuit. The locked circuit produces correct output only when all the keys (PIPs) are loaded into the lookup table. In CAC_{hd}, the set of PIPs includes all input patterns that have a certain Hamming distance h from the secret key key_c, i.e., $\forall p \in PIPs, HD(key_c, p) = h$.

[5] In point-function-based locking, the point-function(s) are simply XORed with the original circuit. It is different from point-function obfuscation where several cryptographic primitives such as random permutations are required [43].

By choosing appropriate h, designers change the number of PIPs and thus trade-off the resilience against SAT, approximate, and structural attacks. CAC_{rem} relies on EDA tools to generate potential PIPs for a given circuit [40, 45].

Structural attacks on CAC locking. CAC_{flex} and CAC_{hd} synthesize the corrupted circuit using logic-synthesis tools, which may retain the corrupting unit unmerged during synthesis. A recent attack by Yang et al. [50] breaks CAC_{hd} circuits by locating and removing the corrupting unit. A relatively more sophisticated FALL attack focuses on recovering the key for CAC_{hd} circuits from the properties of the Hamming distance unit [36]. These attacks may work against CAC_{flex} for a small number of PIPs. However, none of the existing structural attacks can break CAC_{rem}: CAC_{rem} does not utilize a corrupting unit, structural attacks that attempt to locate and remove the corrupting unit are thwarted [40].

2.3 Security Definitions

Grounded in the above discussion, we define the security properties for logic locking techniques. We use $\mathcal{C}_{orig}$, $\mathcal{C}_{lock}$, and $\mathcal{C}_{cp}$ to denote the original circuit (implementing the Boolean function f), the locked circuit, and the corrupted circuit, respectively. We assume that $\mathcal{C}_{orig}$ has n inputs and only one output[6]. The number of key inputs in $\mathcal{C}_{lock}$ is the same as the number of primary inputs, i.e., the key size $k = n$. Thus, C_{cp} has k inputs and one output. The input space is $I = \{0,1\}^k$, and the keyspace is $K = \{0,1\}^k$. key_c is the correct key.

Definition 1. Correctness. When supplied with the correct key, the locked circuit must produce correct output for all input patterns, i.e.,

$$\begin{cases} C_{lock}(i, key) = C_{orig}(i) & \forall i \in I, \quad \text{if } key = key_c \\ C_{lock}(i, key) \neq C_{orig}(i) & \exists i \in I, \quad \text{if } key \neq key_c. \end{cases}$$

Definition 2. Security against I/O attacks. Following the definition in [10], a logic locking technique $\mathcal{L}$ is α-secure against a probabilistic polynomial-time adversary $\mathcal{A}_{\mathbb{IO}}$, if upon making a polynomial number of queries $q(\alpha)$ to the oracle, the probability of retrieving a PIP, and thus obtaining key_c is no greater than $\frac{q(\alpha)}{2^\alpha}$.

Definition 3. Security against structural attacks. A logic locking technique $\mathcal{L}$ is β-secure[7] against an adversary $\mathcal{A}_S$, who has access to a locked circuit or its PIT and conducts white-box structural analysis if the probability of an attacker to identify a PIP is no greater than $\frac{1}{\beta}$.

[6] This can be generalized to circuits with multiple outputs since a multi-output circuit can be divided into multiple single-output circuits [64].

[7] This notion of security is the same as "k-secure" [65]. Since the logic locking community uses k to refer to key size, we use the term β-secure to indicate "k-secure."

2.4 A Primer on Logic Synthesis

Various EDA tools streamline the design and production of billions of ICs sold annually, with many ICs containing billions of transistors. Logic synthesis translates a high-level design description (typically in a hardware description language) into an optimal low-level representation (e.g., a gate-level netlist). Commercial tools (e.g., Synopsys Design Compiler [54], Cadence Genus [53], etc.) as well as open-source logic-synthesis tools (e.g., ABC [58]) help designers generate netlists in a timely fashion while optimizing the PPA costs.

A simple and widely known format, referred to as the sum of products (SOP), uses only two levels of gates. The AND gates implement product terms, which are ORed (summed). Let us consider a Boolean function f with n inputs and one output. Then, a minterm m_i, $0 \leq i < 2^n$, is a product of exactly n variables; each variable is complemented if the value assigned to it is 0 and uncomplemented if it is 1. We can say that the function f:$\{0,1\}^n \rightarrow \{0,1\}$ is a mapping from 2^n minterms to an output value of 1 or 0. Minterms mapping to 1 form the ON-set F^{ON}; minterms that map to 0 constitute the OFF-set, F^{OFF}. A set of minterms may be represented compactly as an implicant. In an implicant, each variable $x \in \{0,1,\text{-}\}$, where "-" is a don't care; a value of 1 or 0 represents a specified bit. For example, $ab\bar{c}d$ is a minterm, whereas, ab-- is an implicant with two don't care bits. This implicant can also be denoted as ab. It represents four minterms: $abc\overline{d}$, $ab\bar{c}d$, $abc\overline{d}$, and $abcd$.

The key to reducing cost (e.g., area, number of gates, etc.) is to eliminate redundancy. Accordingly, *prime implicants (PIs)*, i.e., implicants that cannot be covered by (in other words, cannot be a subset of) a more general implicant, are central to logic synthesis. A set/table of PIs that contains all minterms of F^{ON} makes a *cover or a prime implicant table (PIT)*. The lowest cost is incurred by a minimum cover/PIT, i.e., a cover that is not a proper superset of any other cover of f. We elaborate on the principles of logic synthesis via K-maps used for minimizing Boolean functions [64].

Example. Fig. 3(a) shows the K-map for the Boolean function $f_1 = \overline{a}\overline{b}cd + \overline{a}bcd + a\overline{b}cd + abcd + \overline{a}\overline{b}d$. Each cell in the K-map is a minterm. A prime implicant is the largest square/rectangle group of adjacent cells in the powers of two. From the K-map, the minimum SOP expression for f_1 is $cd + \overline{a}\overline{b}d$, which has only two PIs.

Distance-1 merging. The distance between two PIs A and B, $D(A,B)$, is the number of bits where A and B conflict. For example, $D(abc\text{-}, \text{-}\overline{b}cd) = 1$, and $D(abc\overline{d}, \text{-}\overline{b}cd) = 2$. K-maps are arranged such that the distance between adjacent cells is one. This arrangement satisfies the distance-1 merging rule, which states that two implicants can only be merged if their distance is one [66]. In Fig. 3(b), the minterm $\overline{abcd}$ stands isolated from other PIs of f_2 as its distance is at least two from those PIs. Our SPI attack exploits this rule.

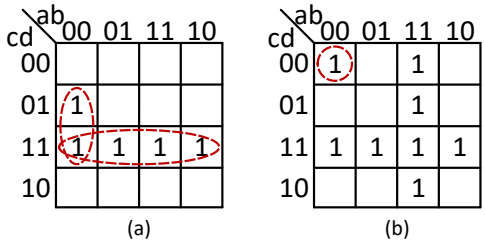

Figure 3: Logic synthesis using K-maps. a) The Boolean function f_1 is represented using only two PIs: cd and $\overline{abd}$. b) f_2 is represented using three PIs: $abcd$ and $\overline{abcd}$.

3 Sparse Prime Implicant (SPI) Attack

We now introduce introduce new structural vulnerabilities of the CAC techniques. Building on these vulnerabilities, we develop the SPI attack that can circumvent all the CAC techniques, irrespective of the EDA tool used for logic synthesis. While traditional structural attacks target gate-level netlists, the SPI attack analyzes the PITs to recover secrets.

3.1 Vulnerabilities of Logic Locking

The CAC techniques construct the corrupted circuit $\mathcal{C}_{cp}$ by adding/removing selected minterm(s) to/from the original circuit $\mathcal{C}_{orig}$. Logic-synthesis tools then synthesize the resulting corrupted circuit. We demonstrate how the optimization conducted for minimizing the PPA cost may expose the PIP. The added/removed PIP may or may not merge with the PIs in the original PIT, as dictated by the distance-1 merging rule. We consider four cases for different combinations of addition/removal and merge/unmerge of PIPs. We use F_{orig}^{ON}, F_{orig}^{OFF}, F_{cp}^{ON}, and F_{cp}^{OFF} to denote the ON-set and OFF-set of $\mathcal{C}_{orig}$ and $\mathcal{C}_{cp}$, respectively. m denotes the PIP.

Table 2: Notations of attacker's search space.

Notation	Explanation
S_{u1}	Number of PIs in F_{cp}^{ON} when $m \notin F_{orig}^{ON}$ ($m \in F_{orig}^{OFF}$) and m does not merge with other PIs in F_{cp}^{ON}
S_{m1}	Number of minterms in F_{cp}^{ON} when $m \notin F_{orig}^{ON}$ ($m \in F_{orig}^{OFF}$) and m merges with other PIs in F_{cp}^{ON}
S_{u2}	Number of PIs in F_{cp}^{OFF} when $m \notin F_{orig}^{OFF}$ ($m \in F_{orig}^{ON}$) and m does not merge with other PIs in F_{cp}^{OFF}
S_{m2}	Number of minterms in F_{cp}^{OFF} when $m \notin F_{orig}^{OFF}$ ($m \in F_{orig}^{ON}$) and m merges with other PIs in F_{cp}^{OFF}

Case 1. Adding a PIP. Suppose F_{cp}^{ON} is constructed by selecting an arbitrary PIP from F_{orig}^{OFF} and adding it to F_{orig}^{ON}. Thus, $F_{cp}^{ON} = F_{orig}^{ON} \cup \{m\}$.

Case 1(a). PIP does not merge. When the distance of m from all the PIs of F_{orig}^{ON} is greater than one, a logic-synthesis tool cannot merge m with any of the PIs of F_{orig}^{ON}. Thus, m appears in the PIT of F_{cp}^{ON} as a PI without any don't care bits.

Example. Fig. 4(a) shows the PIT of F_{orig}^{ON} with two PIs. The PIT of F_{cp}^{ON}, shown in Fig. 4(b), is constructed by removing the PIP 0000 from F_{orig}^{OFF} and adding it to F_{orig}^{ON}. Any logic-synthesis algorithm will retain 0000 as a PI in F_{cp}^{ON} since $D(m, PI_1) \geq 2$ and $D(m, PI_2) \geq 2$. In other words, m is excluded from the distance-1 merging operations.

The isolated m may be recovered directly from the PIT of F_{cp}^{ON}. Let $\mathcal{S}_{u1}$ denotes the search space for the attacker. Assuming there are $|\mathcal{S}_{u1}|$ PIs in F_{cp}^{ON}, the probability of success for an attacker is $\frac{1}{|\mathcal{S}_{u1}|}$, showing that the CAC techniques protecting arbitrary PIPs are only $|\mathcal{S}_{u1}|$-secure against structural attacks in the worst case.

Case 1(b). PIP merges. If the distance of m from any PI of F_{orig}^{ON} is exactly one, a logic-synthesis tool can merge m with a PI. Adding minterm(s) to a PIT allows grouping the minterms in new ways, potentially leading to the creation of new PIs.

Example. Fig. 4(f) shows that adding the PIP 0101 to F_{orig}^{ON} generates a new PI, -1-1. This PI has two don't care bits compared to zero in the PIP. An attacker cannot recover a merged PIP directly from the PIT. The search space $\mathcal{S}_{m1}$ for the attacker is the set of all minterms contained in the ON-set of the corrupted circuit i.e., $|\mathcal{S}_{m1}| = |\{m_i | \forall m_i \in F_{cp}^{ON}\}|$. For most Boolean functions, $|\mathcal{S}_{m1}| >> |\mathcal{S}_{u1}|$.

Case 2. Removing a PIP. A designer can also build F_{cp}^{ON} by removing a minterm m from F_{orig}^{ON}. Using De Morgan's law, this removal is equivalent to adding m into F_{orig}^{OFF}, i.e.,

$$F_{cp}^{ON} = \overline{F_{cp}^{OFF}} = \overline{F_{orig}^{OFF} \cup \{m\}}. \quad (1)$$

Case 2(a). PIP does not merge. When m does not merge with any PI in F_{orig}^{OFF}, it appears as a standalone PI in the PIT of F_{cp}^{OFF}. One can recover m directly from the PIT of F_{cp}^{OFF}. Similar to Case 1(a), the search space is $\mathcal{S}_{u2}$, and $|\mathcal{S}_{u2}|$ is the number of PIs in F_{cp}^{OFF}. m does not merge with F_{orig}^{OFF} as its distance from all PIs of F_{orig}^{OFF} is greater than one. This condition implies that the distance of m must be one from at least one of the PIs of F_{orig}^{ON}.

Example. Fig. 4(g) shows that upon removing the PIP 1111, the two original PIs split into four new PIs which get implemented as the corrupted circuit. The original PIs have two don't care bits each, whereas the new PIs have only one don't care bit. Thus, removing minterm(s) from a PIT may lead to the generation of additional PIs. Consequently, the PIT of F_{orig}^{ON} contains information about the PIP, which may be exploited by attackers. This example demonstrates that removing a PIP from PIT can introduce more PIs into a PIT as compared to adding a PIP.

Case 2(b). PIP merges. If m merges with any of the PIs of F_{orig}^{OFF}, it does not appear as an isolated PI in F_{cp}^{OFF}. The search space S_{m2} is the set of all minterms in F_{cp}^{OFF}, i.e., $|\mathcal{S}_{m2}| =$

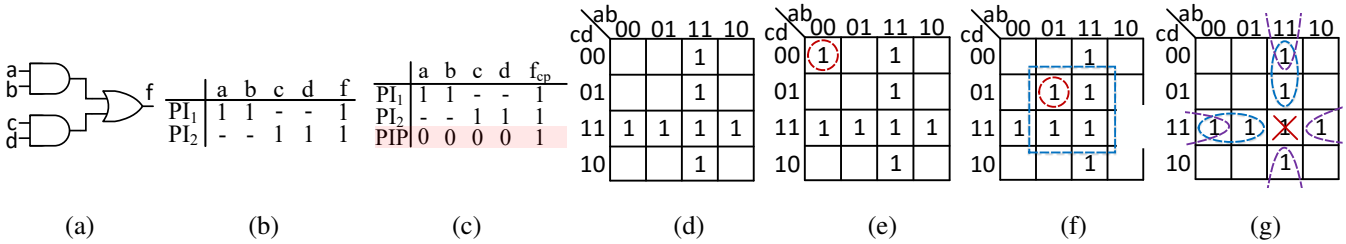

Figure 4: Logic optimization and logic locking: (a) The original circuit, (b) its PIT, (c) PIT of the corrupted circuit constructed by adding the PIP 0000, (d) K-map of the original circuit, (e) K-map of the corrupted circuit with the PIP 0000 as an isolated PI, (f) K-map of the corrupted circuit with the PIP 0101 leading to the creation of new PIs, and (g) K-map of with the removed PIP 1111 splitting the two original PIs.

$|\{m_i' | \forall m_i' \in F_{cp}^{OFF}\}|$. An attacker may know how the corrupted circuit is constructed. Access to this information may impact the size of the search space (see Appendix B).

Key takeaways. According to the relationship between logic synthesis and the creation of corrupted circuits, we observe: (i) If a PIP does not merge, it appears as a fully specified PI and can be recovered directly from the PIT. (ii) If a PIP merges, it introduces new PIs, which may reveal the PIP.

This subsection pointed out a vulnerability associated with the synthesis of locked circuits. The next subsection describes how our attack exploits this vulnerability.

3.2 Exploiting the PIT

We now explain how logic-synthesis principles can be exploited to extract the PIP from a PIT of the corrupted circuit. The case where a PIP does not merge with the PIs and appears as an isolated PI is easy to exploit. However, when a PIP merges with the existing PIs, new PIs are introduced into the PIT. Here, we explain the properties of the new PIs that can be used to determine the PIP. We focus on the removal of a PIP from a PIT since it introduces more pronounced changes; the addition of a PIP to the ON-set is equivalent to removing it from the OFF-set (using De Morgan's law) and vice versa.

Inferring PIP from split PIs. As mentioned in Case 2(a), removing a PIP may split a *merged PI*, representing a larger set of minterms, into multiple PIs sans the PIP; we denote the resultant PIs as *split* PIs. Consider removing the PIP 000100110 from the original PIT shown in Fig. 5(a). Upon removing this PIP, the first PI splits into six PIs as shown in Fig. 5(b). The other two PIs are unaltered. If the same PIP is added back to the corrupted circuit PIT, the split PIs can merge to form the merged PI. With the PIP removed, the split PIs cannot merge. The distance of any split PI and the PIP is one as the PIP is the glue to merging the split PIs. The distance between any two split PIs is zero since all split PIs are derived from the same merged PI by setting a unique don't care bit to the complement of the corresponding bit in the PIP. If the merged PI has s don't care bits, each split PI will have $(s-1)$ don't care bits. For example, PI_1 in Fig. 5(b) is generated by the setting the bit j to 1 since the rightmost bit in the PIP is 0.

Consider the simplest case where the merged PI is the universal set $\mathbb{U}$, i.e., all the bits are don't cares. Let M be the number of split PIs, and PI_S^j denote the j^{th} split PI. The PIP is the difference between $\mathbb{U}$ and the union of split PIs, i.e.,

$$PIP = \mathbb{U} \setminus \bigcup_{j=1}^{M} PI_S^j = \overline{\bigcup_{j=1}^{M} PI_S^j} = \bigcap_{j=1}^{M} \overline{PI_S^j}. \quad (2)$$

Thus, the PIP can be computed by intersecting the complements of the split PIs. In reality, a merged PI will not span the entire $\mathbb{U}$ and will contain certain specified bits. For these bits, all the split PIs are in consensus. Thus, the specified bits in the merged PI will be replicated in the split PIs. For example, in Fig. 5(a), the inputs d, h, and i appear as 1 in PI_1. In the shaded region of Fig. 5(b), the values of these three inputs are mostly 1 while the rest are mostly don't cares.

Determining split PIs. The split PIs may only be a small subset of all PIs in the PIT of a corrupted circuit. We can determine the split PIs from a PIT by finding a subset of PIs that have the same number of don't care bits and the inter-PI distance is zero. From observing thousands of PITs of benchmark circuits, we find that the split PIs tend to have a large percentage ($\geq 50\%$) of don't care bits. We refer to PIs with $\geq 50\%$ don't care bits as *sparse PIs*. Given a PIT, we identify the set of SPIs by grouping PIs based on the number of don't care bits. For example, PIs 1-6 in Fig. 5(a) are both

	a	b	c	d	e	g	h	i	j	f_{orig}
PI_1	-	-	-	1	-	-	1	1	-	1
PI_2	0	0	0	0	0	0	1	0	0	1
PI_3	1	1	1	0	1	1	1	0	1	1

(a)

	a	b	c	d	e	g	h	i	j	f_{fsc}
PI_1	-	-	-	1	-	-	1	1	1	1
PI_2	-	-	-	1	-	1	1	1	-	1
PI_3	-	-	-	1	1	-	1	1	-	1
PI_4	-	-	1	1	-	-	1	1	-	1
PI_5	-	1	-	1	-	-	1	1	-	1
PI_6	1	-	-	1	-	-	1	1	-	1
PI_7	0	0	0	0	0	0	1	0	0	1
PI_8	1	1	1	0	1	1	1	0	1	1

(b)

Figure 5: The PITs of the (a) original circuit and (b) corrupted circuit. The PIP is 000100110. PI_1–PI_6 are SPIs.

split PIs and SPIs. However, not all SPIs are split PIs. We eliminate false positives by checking the inter-PI distance.

Given a set of split PIs, we can identify the specified bits by determining the bits for which all the split PIs are in consensus. Since the exact computation of consensus between PIs is computationally expensive, we approximate it with majority voting, yielding the specified bits. In Fig. 5(b), majority voting on the values of the inputs d, g, and h yields their correct value, 1, as listed in Fig. 5(a). These heuristics are implemented in the *find_and_parse_split_PIs* step of Algorithm 1. The value of the PIP is determined using Eq. (2).

3.3 SPI Attack Algorithm

The SPI attack can recover the PIP by analyzing the PIT of the corrupted circuit. The first step of the algorithm is extracting the corrupted circuit. This step can be performed using component-level REing tools [67, 68]. Note that the CAC techniques also assume that the attacker can extract the corrupted circuit. From the corrupted circuit, we extract the PIT using logic-synthesis tools, such as ABC [58, 64, 69]. As shown in Alg. 1, the SPI attack has two stages, which are based on the observations in Section 3.1.

Stage 1 attempts to recover a PIP directly from the PIT of the corrupted circuit. Recall that an unmerged PIP tends to exhibit itself as a PI with all the bits specified. The SPI attack can search for the PIP in both F_{cp}^{ON} and F_{cp}^{OFF}. The correctness of the extracted PIP is verified by querying the oracle with the PIP. The output of the corrupted circuit will not match that of an oracle for a true PIP. The attack proceeds to Stage 2 if the true PIP is not recovered.

Stage 2 finds the set of SPIs by grouping the PIs based on the number of don't care bits. It eliminates the false positives for split PIs by checking if the inter-PI distance is zero. The heuristics from Section 3.2 help determine the most probable value for each bit of the PIP. In case CAC techniques are combined with XOR-based locking, the SPI attack can be used in conjunction with the SAT attack to recover key bits for XOR-based locking [9]. Section 4 experimentally validates the effectiveness of the SPI attack and the heuristics.

Attack on multi-output circuits. Till now, we explained our algorithm using a single-output circuit. The SPI attack can break multi-output circuits by reducing them to multiple single-output circuits, aka *logic cones*. This is a common technique used in logic synthesis and logic locking [64, 69–71].

3.4 Improving the Scalability

The SPI attack extracts the PIT from the corrupted circuit, which is an NP-Hard problem [64]. Over the last four decades, the logic-synthesis community has developed efficient heuristics for computing PITs for common circuits. We use the open-source tool ABC [58] to compute the PITs. Using ABC we can compute the required PITs for all but the three harder competition circuits within 48 hours. For the large circuits, ABC either does not finish PIT computation or terminates early due to insufficient memory.

To overcome this limitation, we rely on a key insight on arbitrarily selecting PIPs that allow us to run the SPI attack at the sub-circuit level. We observe that if a PIP is isolated in the PIT of the complete circuit, it tends to be isolated in the PITs of the sub-circuits. When the distance of the PIP from the other PIs in the PIT of the complete circuit is larger than two, there is a good chance that the partial PIPs (PIPs at the circuit level) can have a distance of two from other PIs, especially if the sub-circuits are for the nodes close to the output of the complete circuit.

This insight allows us to follow a divide-and-conquer approach and compute PITs only for the sub-circuits without computing the PIT for the complete circuit, which lowers computational effort by several orders of magnitude. We (i) divide a circuit into several sub-circuits using the depth-first search to find gates in the fan-in of a node, (ii) extract the PITs of sub-circuits, and (iii) launch the SPI attack on sub-circuits to recover parts of the PIP. When only a subset of PIP bits recovered from the sub-circuits or there are multiple candidates for the PIP, brute-force may be used to determine remaining bits or prune the incorrect candidates; alternatively, the SAT attack may also be used [9, 72]. Due to space limitations, we present the complete algorithm in Appendix A. Following this new approach, we can break any of the harder competition circuits within 10 seconds; these circuits have key sizes up to 195 compared to the largest key size of 80 for the rest of

Algorithm 1: SPI attack

Input: Locked netlist $\mathcal{C}_{lock}$ and Oracle O
Output: Correct key K_c

1 $\boldsymbol{LC}_{lock} \leftarrow$ extract_logic_cones($\mathcal{C}_{lock}$)
2 **for** $lc_{lock} \in \boldsymbol{LC}_{lock}$ **do**
3 $lc_{cp} \leftarrow$ extract_corrupted_circuit(lc_{lock})
4 $PIT \leftarrow$ extract_PIT(lc_{cp})
5 //——————**Stage 1**——————
6 $\boldsymbol{P_1} \leftarrow$ get_fully_specified_PI(PIT)
7 $\boldsymbol{PIPs}_{verified} \leftarrow$ verify($\boldsymbol{P_1}, lc_{cp}, O$)
8 **if** ($\boldsymbol{PIP}_{verified} \neq \varnothing$) **then**
9 **return** $\boldsymbol{PIP}_{verified}$
10 **end**
11 //——————**Stage 2**——————
12 $\boldsymbol{P_2} \leftarrow$ find_and_parse_split_PIs(PIT)
13 $\boldsymbol{PIPs}_{verified} =$ verify($\boldsymbol{P_2}, lc_{cp}, O$)
14 **if** ($\boldsymbol{PIPs}_{verified} \neq \varnothing$) **then**
15 **return** $\boldsymbol{PIPs}_{verified}$
16 **end**
17 **goto** line 2 // Process next logic cone
18 **end**
19 **return** $\varnothing$

the circuits. While the PITs for the complete circuit cannot be computed in 48 hours, those for the sub-circuits can be computed within 10 seconds (see Section 4.2).

3.5 Broader Applicability

Breaking all the CAC techniques. The SPI attack operates on a PIT, which is specific to a Boolean function and is agnostic to the netlist structure. Consequently, the SPI attack is independent of the underlying locking technique or the way the corrupted circuit is generated. Thus, our attack can break all the CAC techniques, as they only differ on how the corrupted circuit is generated (see Section 4.3). Also, our attack has a tremendous advantage over existing attacks, which are tailored for specific defense techniques. For instance, SPS attack identifies AND trees and uses it to break Anti-SAT; [39] thwarts this by altering the netlist. The FALL attack is specific to CAC_{hd} and cannot circumvent CAC_{rem} because it exploits the heuristics of the former, and the latter violates that heuristic. However, our SPI attack is agnostic to the circuit structure, making it widely applicable.

Effectiveness across EDA tools. The implementation-agnostic nature of our attack gives us two distinct advantages. First, our attack is independent of the designer's tools and their objectives, such as minimizing PPA costs, etc. Thus, our attack is independent of the use-case scenario of the IC. Second, our attack does not depend on the implementation of the function as an ASIC or using FPGA. In Section 4.4, we demonstrate that the SPI attack can break all the CAC techniques synthesized using various commercial and academic logic-synthesis tools, and a combination thereof.

4 SPI Attack Results

4.1 Experimental Setup

Platform and EDA tools. We perform our attack experiments on a 32-core Intel Xeon processor at 2.6 GHz with 512 GB RAM. We used the ABC logic-synthesis tool to extract PITs [58] and Synopsys Design Compiler as the logic-synthesis tool [54] unless otherwise specified. Our experiments use the NanGate FreePDK45 Open Cell Library for ASIC implementations [73] and Xilinx Spartan-3 FPGA for the FPGA implementation [74]. The proposed approach is applicable to other technology libraries and FPGA platforms.

Benchmark circuits. We show the effectiveness of the SPI attack primarily against CAC_{rem} since this technique remains unbroken in the competition [40]. We also run our experiments on another unbroken technique, CAC_{flex}. The organizers of the logic-locking competition provided six CAC_{rem} circuits and their oracles. This competition uses the `ITC'99` benchmark suite [75]. As shown in Table 3, the key size is 16 for four circuits[8] and 80 for two circuits. Upon reporting our attack results to the organizers, they provided with three more circuits locked with larger key sizes, i.e., 102, 95, and 195 for the `b17L`[9]. We also use the controller circuits of the ARM `Cortex-M3` processor [59] and the `GPS` module with 213K gates from the Common Evaluation Platform for evaluating hardware security schemes [60].

To demonstrate that the SPI attack can break all the CAC techniques, we generated locked circuits for TTLock, CAC_{hd}, CAC_{flex} [10, 27]. We locked only the cone with the largest available input size. Only one PIP is used to lock each circuit, as this is the case for competition circuits.

4.2 Breaking CAC_{rem}

Success rate. Table 4 presents the attack results on the competition circuits [51]. It shows that the SPI attack can break all the circuits. We attribute this success to the exploitable changes made to the PITs by existing logic-synthesis algorithms upon adding/removing PIPs arbitrarily.

Execution time. Since we deploy the divide-and-conquer approach mentioned in Section 3.4 to break the larger competition circuits, we discuss their results separately in the next paragraph. This paragraph discusses results for all but the competition-large circuits. Table 4 shows that the SPI attack takes less than a second to break any circuit. The execution time remains small since the number of PIs in the PITs of the competition circuits is relatively small. The largest PIT is for the circuit `b15` with only 171 PIs. Another reason for the smaller execution time is that the SPI attack at first targets smaller logic cones and can terminate successfully as soon as all the bits of the PIP are determined; the larger and computationally intensive cones need not be processed. The execution time for `b15` is the highest since few of its processed logic cones have at least 105 or higher inputs.

Competition-large circuits. Since ABC could not extract PITs for the competition-large circuits within the time limit, we break the circuits using the divide-and-conquer approach. On attacking only the sub-circuits, the SPI attack exits within 10 seconds. For all large circuits, we identify at least one sub-circuit containing all the primary inputs that feed the correcting circuit. By excluding the parts of the circuits not involved in logic locking, the PIT computation becomes faster. For each large circuit, the divide-and-conquer SPI attack determines a single candidate PIP, the true PIP. For the ease of discussion, consider that the PIT for each large circuit is extracted and the SPI attack completes in exactly 48 hours. With this conservative assumption, the divide-and-conquer SPI attack runs $\geq$17000X faster than the basic SPI attack. The precise speed-up depends on the circuit being processed.

[8] While we understand the limitations of using small key sizes to evaluate attacks, we still include them in the results as they are part of the competition.

[9] The original `b17` circuit is the same in both instances. The small and large versions of `b17` are locked with 80 bits and 102 bits, respectively.

Table 3: The statistics of the benchmark circuits. The parameters of the competition circuits are reported as provided by the organizers. For ARM Cortex-M3 and GPS circuits, we lock the logic cone with the largest key size.

Circuit	Competition-small						Competition-large			ARM Cortex-M3				CEP
	b10	b11	b12	b13	b15	b17	b17L	b20	b22	ARMc1	ARMc2	ARMc3	ARMc4	GPS
# inputs	28	38	126	63	485	1452	1452	522	767	34	509	213	232	9707
# outputs	17	31	119	53	449	1445	1445	512	757	125	63	66	43	9731
# gates	172	726	944	289	11577	37479	37479	19682	29162	1362	2188	657	491	213125
# protected cones	9	19	21	16	166	42	1	1	1	1	1	1	1	1
Key size	16	16	16	16	80	80	102	95	195	33	19	69	26	63

Table 4: Success rate and execution time (s) of different attacks on CAC_{rem} circuits from the logic locking competition [51]. "TO" denotes a timeout of 48 hours.

Attack \ Circuit	Attack success									Execution time (s)								
	Competition-small						Competition-large			Competition-small						Competition-large		
	b10	b11	b12	b13	b15	b17	b17L	b20	b22	b10	b11	b12	b13	b15	b17	b17L	b20	b22
SAT [9]	✓	✓	✓	✓	✓	×	×	×	×	2.4	6.4×10^4	0.2	1.1	5.8	TO	TO	TO	TO
AppSAT [32]	×	×	×	✓	✓	×	×	×	×	6.5	2.8	3.9	0.5	1.7	73	9.0	10.0	18.2
ATR [35]	×	×	×	×	×	×	×	×	×	0	0	0.1	0	0.4	1.7	0.1	0.2	0.3
SPS [47]	×	×	×	×	×	×	×	×	×	0.1	0.2	0.3	0.1	3.3	21	0.8	1.3	0.8
FALL [36]	×	×	×	×	×	×	×	×	×	0.2	0.6	0.8	0.3	50	0	3.4	3.1	5.4
SPI	✓	✓	✓	✓	✓	✓	✓	✓	✓	0.4	0.4	0.4	0.2	0.5	0.3	8.3	6.8	8.6

Table 5: Execution time (s) of the SPI attack on circuits protected using different CAC techniques. The success rate of the SPI attack is 100% for all the circuits and thus, not presented.

Technique \ Circuit	ITC'99						ARM Cortex-M3				CEP
	b10	b11	b12	b13	b15	b17	ARMc1	ARMc2	ARMc3	ARMc4	GPS
TTLock [27]	0.4	0.5	1.2	0.8	17.1	0.6	0.3	0.2	0.2	0.4	0.3
CAC_{hd} [10]	0.5	0.7	1.2	0.7	22.5	0.7	0.5	0.5	0.5	0.7	0.5
CAC_{flex} [10]	0.3	0.6	0.7	0.3	23.4	0.7	0.3	0.2	0.4	0.2	0.2

Comparison with existing attacks. Table 4 also presents a comparison of the SPI attack with three structural attacks and two I/O attacks. SAT attack breaks all the circuits locked with 16-bit keys, as the search space is only 2^{16}. However, it cannot break the `b17` circuit and the three hard circuits in 48 hours. The `b15` circuit is an exception since the SAT attack breaks it within six seconds because 90% of the key bits for `b15` are 0s, the default starting point of the SAT solver. The AppSAT attack finishes within a few seconds for all circuits albeit without recovering the PIP. Only for two 16-bit circuits and the `b15` circuit, AppSAT can extract the correct key. Thus, none of the existing structural attacks can break even a single CAC_{rem} circuit. The SPS and ATR attacks fail to identify point-functions in CAC_{rem} circuits. The FALL attack cannot locate the correcting unit. In comparison, the SPI attack breaks all the circuits within a few seconds.

4.3 Breaking all CAC techniques

Table 5 reports the effectiveness of the SPI attack against different CAC techniques: TTLock, CAC_{hd}, and CAC_{flex} [10, 27]. The success rate of the attack is 100%. The execution time only varies slightly across different CAC techniques. The reason is that the SPI attack does not target the netlist but the Boolean function to find the PIPs, and the CAC techniques only alter the netlist.

4.4 Evaluation against Different EDA Tools

Table 6 reports the results of the SPI attack on CAC_{rem} circuits synthesized using six logic-synthesis tools. We experiment with five industrial tools, Cadence Genus, Synopsys Design Compiler, Synopsys Synplify, Xilinx Vivado, and Mentor Graphics Precision RTL, and one academic tool, ABC [53–58]. The SPI attack breaks all the circuits, irrespective of the logic-synthesis tool used to generate the corrupted circuit. Each tool uses a different set of optimization heuristics. A designer may re-synthesize a circuit with multiple tools to achieve minimum PPA overhead. We replicate this scenario by cascading the logic-synthesis tools, i.e., passing the output of one tool to the next for further optimization. Even this sophisticated synthesis setup fails to impact the success rate of the SPI attack.

Table 6: Execution time (s) of the SPI attack on CAC_{flex} circuits synthesized using different EDA tools. The success rate of the SPI attack is 100% for all the circuits and thus, not presented. The industrial tools are anonymized to avoid disclosure conflicts.

Circuit / EDA tool	Tool category	ITC'99						ARM Cortex-M3				CEP
		b10	b11	b12	b13	b15	b17	ARMc1	ARMc2	ARMc3	ARMc4	GPS
EDA tool ①	Industrial	0.3	0.6	0.7	0.3	23.4	0.7	0.3	0.2	0.4	0.2	0.2
EDA tool ②	Industrial	0.4	0.5	1.2	0.8	17.1	0.6	0.3	0.2	0.2	0.4	0.3
EDA tool ③	Industrial	0.5	0.7	1.2	0.7	22.5	0.7	0.5	0.5	0.5	0.7	0.5
EDA tool ④ (ABC [58])	Academic	0.2	0.5	0.5	0.3	14.3	1.1	0.7	0.6	0.5	0.6	0.2
EDA tool ⑤	Industrial	0.3	0.7	0.4	0.4	18.6	0.8	0.4	0.4	0.8	0.4	0.2
EDA tool ⑥	Industrial	0.2	0.5	0.5	0.4	10.0	0.7	0.2	0.9	0.4	0.8	0.3
① › ② › ③ › ④ › ⑤ › ⑥	Mix	0.3	0.8	0.7	0.6	15.1	0.9	0.5	0.5	0.5	0.5	0.3

5 Dist2: A Security Property

The results in the previous section demonstrate the vulnerability of CAC techniques to structural attacks. The techniques remain specifically vulnerable to the SPI attack since they select PIPs on an arbitrary basis. This hints that PIPs must be selected more carefully by taking into account the distance with the PIs of the original circuit. Recall that an addition (deletion) of a PIP changes the PIT only if the added (deleted) PIP has a distance less than 2 from any other PI. To ensure that the PIT does not leak information about the PIP, only those PIs should be selected that are "sufficiently distanced," i.e., at least distance two away from PIs of the original circuit. We refer to this condition for the resilience against structural attacks as *Dist2 property* and to such PIPs as D2PIPs, with D2 denoting a minimum distance of two. Note that this can be generalized to a distance of at least d. Without satisfying the Dist property, CAC circuits remain vulnerable to the SPI attack and similar structural attacks. A D2PIP satisfies

$$\begin{cases} \text{if } D2PIP \in F_{orig}^{ON}, & \forall PI_i \in F_{cp}^{ON} \quad D(D2PIP, PI_i) \geq d \\ \text{if } D2PIP \in F_{orig}^{OFF}, & \forall PI_i' \in F_{cp}^{OFF} \; D(D2PIP, PI_i') \geq d. \end{cases}$$

These conditions guarantee that (i) a D2PIP cannot be merged with any PI in F_{cp}^{ON} if the corrupted circuit is created by removing a PIP from F_{orig}^{ON}, and (ii) a D2PIP cannot be merged with any PI in F_{cp}^{OFF} if the corrupted circuit is created by adding a PIP to F_{orig}^{ON}. Thus, the feature of D2PIP is, on choosing a D2PIP as the PIP, the rest PIs in F_{orig}^{ON} or F_{orig}^{OFF} do not change, thereby the constructed F_{cp}^{ON} or F_{cp}^{OFF} will not to leak the secret by letting the PIP isolated separately.

Dist2 property simply introduces a new constraint on the PIPs that can be protected. The property can be easily used in conjunction with CAC techniques to attain resilience against both I/O and structural attacks. Once a set of D2PIPs has been found, they can be protected using any of the CAC techniques. We emphasize that if Dist2 property is not taken into account, a locked circuit remains vulnerable to SPI attack and anticipated structural attacks of the same nature.

5.1 Attack Resilience under Dist2

Structural attacks. Locking using only D2PIPs ensures that the PIP will not alter the structure of the PIT and thus not reveal any information about the PIP, through neither isolated PIPs nor split PIs. When locking using arbitrary PIPs, the search space for an attacker is all 2^k candidate PIPs. However, D2PIPs form only a small subset of all input patterns, and future attacks can attempt to enumerate all D2PIPs.The search space $\mathcal{S}$ for an attacker is the set of all possible PIs that have a distance greater than one from all the PIs of the corrupted circuit. The size of the search space quantifies the resilience against structural and is highly circuit specific. Theorem 1 presents the resilience against structural attacks for locking two-level SOP circuits using only D2PIPs. Recall that the SOP format uses two levels of gates, AND gates (implementing product) followed by an OR gate (implementing sum).

Theorem 1: *Satisfying Dist2 property is sufficient to achieve* β*-security against structural attacks on Boolean circuits represented in SOP form, where* $\beta = |\mathcal{S}|$.

Proof: The proof is presented in Appendix C. □

The theorem provides a fundamental condition for resilience against structural attacks but only for two-level SOP circuits. This result can be extended to multi-level circuits (or any circuit format) by taking into account the principles of multi-level logic synthesis (or a specific circuit format). Suppose a corrupted circuit PIT is constructed by removing a D2PIP from the original circuit. When the PIT is synthesized into a gate-level netlist, a logic-synthesis algorithm should not introduce a standalone sub-circuit for the excluded D2PIP into the netlist since that will increase the PPA cost. Even upon synthesis with current EDA tools, the chances for a D2PIP to appear standalone in a netlist will be small since the tools aim at the least cost implementation. However, a challenge in establishing security guarantees is that the precise EDA algorithm used to synthesize the circuit in the given format must be considered so that the desired security properties are maintained at each step of the algorithm.

I/O attacks. D2PIPs may contain both specified bits and don't care bits. Only the specified bits can be treated as key bits for the circuit function remains the same regardless of the value assigned to the don't care bits. A caveat of protecting

Table 7: Feasibility of satisfying Dist2 property by common benchmark circuits. A protectable logic cone has at least one D2PIP.

Circuit	ITC'99								ARM Cortex-M3				CEP
	b10	b11	b12	b13	b15	b17	b20	b22	ARMc1	ARMc2	ARMc3	ARMc4	GPS
# Logic cones	17	31	119	53	449	1445	512	757	125	63	66	43	9.7×10^3
Logic cones examined (%)	100	100	100	100	81.7	81.1	40.2	38.9	100	90.5	89.4	93	99
Protectable logic cones (%)	0	16.1	7.6	13.2	0.4	0.4	1.6	1.1	31.2	1.6	3	2.3	51.4
Max # D2PIPs (in one cone)	0	3	32	2	1	1	2	8	3	1	2	2	64
Max # PIs (in one cone)	24	256	132	105	5.6×10^4	8.2×10^4	8.2×10^4	8.4×10^4	363	12	6.5×10^3	14	1.1×10^3
Cumulative # D2PIPs	0	11	222	11	2	6	14	21	66	1	4	2	7.6×10^4
Max key size	N/A	15	26	11	7	7	29	29	20	6	2	2	63
Max execution time (s)	0.1	76.8	102.7	60.9	1.3×10^4	2.2×10^3	4.5	4.9	56	3.4	3.4	2.9	85

only D2PIPs is that the maximum key size is dictated by the number of specified bits in the D2PIPs. If PI_1 is selected as a D2PIP in Fig. 5(a), the key size is only three; upon selecting PI_2, the key size is nine. Existing synthesis algorithms aim at maximizing the number of don't care bits in PIs to reduce the implementation cost. Thus, the circuits processed using existing EDA tools will achieve relatively smaller key sizes.

5.2 Can Benchmark Circuits Satisfy Dist2?

Checking whether a circuit satisfies the Dist2 property requires the computation of a PIT, which is an NP-Hard problem. As already mentioned in Section 3.4 PITs may not be computed for certain circuits. For all but two circuits (`b20` and `b22`), we can extract the PITs for more at least 80% of the logic cones in standard benchmark circuits. The two circuits have unusually high percentages of logic cones with 215 or more inputs; thus, ABC cannot extract PITs for them in 48 hours. Since for all the circuits except `b20` and `b22`, we can extract the PITs for more than 90% of the logic cones, on average, we claim that the data in Table 7 represents the general characteristics of standard benchmark circuits. Even for circuits `b20` and `b22`, the computed PITs exhibit the same trend for D2PIPs as that for the remaining circuits.

Satisfying Dist2 property. A critical finding of our study is that only a small fraction of the logic cones are protectable using D2PIPs. We say a logic cone is protectable if its PIT has at least one D2PIP. On average, only 1.3, 9.5%, and 51.4% of logic cones satisfy the Dist2 property in the `ITC'99` circuits, `ARM Cortex-M3` controllers, and the `GPS` circuit, respectively. The `GPS` circuit and `ARMc1` controllers are notable exceptions with 51.5% and 31.2% protectable cones. More importantly, the number of D2PIPs found in any circuit is extremely small. The `GPS` circuit has the largest number of D2PIPs in any logic cone, which is only 64. All other circuits have smaller numbers of D2PIPs, mostly ≤ 10. This dearth of D2PIPs deprives the designers of any choice in protecting PIs based on their application or use-case. Furthermore, since only a few D2PIPs exist, the error-rate at the outputs will remain low. One may ask if it is possible for the majority of D2PIPs to be concentrated in the un-examined cones for which the PITs could not be computed in 48 hours. This is unlikely as D2PIPs that can only be a small subset of the input space.

Table 7 also reports the maximum achievable key size, the main indicator of resilience against I/O attacks. The maximum key size attained is 29, 20, and 63 for the `ITC'99`, `Cortex-M3`, and `GPS` circuits, respectively. Key sizes around 30 can be easily brute-forced using today's desktop computers, implying that most of the benchmark circuits cannot even defend against brute-force. A security level of 2^{63} is significantly smaller compared to the NIST recommended security level of 2^{112} [76]. Thus, our most important finding is that the commonly-used circuits are unsuitable for any logic-locking technique as they fail to achieve a sufficiently large key size when satisfying the Dist2 property. To secure the IC supply chain against piracy and to securely outsource IC fabrication, designers can currently rely on schemes such as verifiable computation, homomorphic encryption, and multi-party computation that are computationally expensive and may incur prohibitive PPA overhead [77, 78].

6 Discussion

6.1 Potential Countermeasures

As we have seen in Section 5.2, the number of D2PIPs in conventional designs is mostly a few tens. This small number indicates that logic locking cannot protect the vast majority of the circuits. The reason for a small number of D2PIPs is the EDA tools and their optimization objectives. Before logic-synthesis tools convert high-level functions into their corresponding Boolean circuits, the high-level variables need to be Boolean encoded. For instance, the encoding process translates instructions in the high-level format, such as ADD, SUB, MUL, etc., to their corresponding opcode. The resultant Boolean circuit implements the mapping between inputs and outputs. Usually, such encoding schemes aim at packing the maximum number of codewords in the least number of bits [69] while simultaneously reducing the PPA of the resultant circuit [79]. Unfortunately, as we observed, these optimization objectives lead to a PIT with many PIs closely arranged, thereby reducing the number of D2PIPs. A recent analysis of properties of Boolean functions also alludes to the difficulty of locking certain circuits [80]. By demonstrating that the current EDA algorithms and encoding schemes are

not fully compatible with existing logic-locking techniques, our paper calls for a revamp of existing industrial EDA tools to take supply chain security into account. In the future, we intend to develop a security-centric encoding scheme to ensure that the PIs maintain a minimum inter-PI distance. This distance constraint will increase the number of specified bits in the PIT, leading to high PPA costs. Thus, we need to find an optimal trade-off between security and PPA overhead.

6.2 Scalability

The SPI attack requires generating the PIT of a Boolean function and computing the distance between all PIs. The exact optimization of a PIT is an NP-Hard problem [81], which raises concerns scalability of the attack. However, as described in Section 3.4 and demonstrated experimentally in Section 4.2, the attack can be easily scaled to larger circuits using a divide-and-conquer approach that eliminates the need for computing PIT of the complete circuit.

6.3 Other Locking Techniques

Meerkat considers resilience against structural attacks [82]. It introduces key gates such that that netlist structure remains identical regardless of the key value, hampering structural analysis attacks on netlists. Meerkat, however, requires the transformation of a netlist into a reduced-ordered binary decision diagram and cannot scale well for large circuits [82]. Meerkat, however, does not account for I/O attacks.

Cyclic locking techniques attempt to defeat I/O attacks by introducing cycles in a combinational circuit [83–85]. The underlying assumption is that the SAT/SMT solvers will loop forever in the cycles. However, our attack is still applicable to cyclic locking: a circuit protected by cyclic locking could be converted to an acyclic circuit by unrolling it. Successful attacks also exist against cyclic locking [84].

Sequential locking introduces additional states into finite state machines (FSM) so that a design enters the desired functional states only upon applying the correct key [6, 17, 86–88]. Even FSM synthesis techniques follow the distance-1 merging rule, similar to that for combinational circuits [70,89]. Naturally, one can extend our SPI attack to sequential locking by targeting the sequential synthesis steps.

Scan locking. All oracle-based attacks rely on scan chains to query the oracle. A few defense techniques thwart I/O attack by locking the scan chains [90] or making the scanned-out responses independent of the key value [91]. However, logic locking can be broken even without scan access [87, 92].

7 Conclusion

We have examined how vulnerable logic-locking techniques become when processed through industry-standard EDA tools. Contrary to the existing attacks, our SPI attack does not target a specific netlist format. Instead, it analyzes the Boolean functions, which renders our proposed attack and security property widely applicable. Our SPI attack takes only a few seconds to break any locked circuit irrespective of the CAC technique. The attack has a 100% success rate even when the circuits are optimized using various logic-synthesis tools, highlighting the inadequacy of the EDA tools in defending against white-box attacks. Our findings, in satisfying the Dist2 property, highlight the infeasibility of locking circuits with reasonable security.

Ramifications. The SPI attack calls for reevaluating the security of logic-locking techniques. The encoding schemes underlying current logic-synthesis tools and the subsequent optimizations in their current form fail to make a logic-locking technique secure. We have also highlighted the scarcity of D2PIPs in common benchmark circuits. Thus, we urge logic-locking researchers to consider the following: (i) Is the gate-level netlist the right abstraction level to apply logic locking? Thus, are ISCAS and ITC circuits appropriate benchmarks to evaluate logic locking? (ii) Can we blindly trust EDA tools for logic locking and other hardware security problems? We urge the need for the community to develop EDA tools and locking techniques that are cognizant of each other.

Acknowledgement

We thank Dr. Abhrajit Sengupta, Nimisha Limaye, Prof. Ozgur Sinanoglu from NYU-AD, and other CSAW'19 organizers for their help in providing locked circuits and verifying the keys. We thank other members of the TAMU SETH lab for their help in collecting data. We also thank Prof. Krishna Narayanan from TAMU for valuable discussions. And, we thank anonymous reviewers for their comments. The work was supported in part by the National Science Foundation (NSF CNS-1749175) and the Defense Advanced Research Projects Agency grants HR0011-20-9-0043 and FA8650-18-1-7827. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government.

References

[1] D. Takahashi, "Globalfoundries: Next-generation chip factories will cost at least $10 billion." https://rb.gy/pjllsf, 2017. Last accessed on 10/12/2020.

[2] J. Purcher, "Apple Supply Chain News: TSMC & Foxconn Plan new chip plants." https://rb.gy/ot1hfv, 2017. Last accessed on 10/12/2020.

[3] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[4] SEMI, "Innovation is at Risk Losses of up to $4 Billion Annually due to IP Infringement." https://rb.gy/ajtlnw, 2008. Last accessed on 10/04/2020.

[5] A. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking Techniques for Intellectual Property Protection," *IEEE/ACM Design Automation Conference*, pp. 776–781, 1998.

[6] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," *USENIX Security Symposium*, pp. 291–306, 2007.

[7] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *IEEE/ACM Design, Automation & Test in Europe*, pp. 1069–1074, 2008.

[8] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *IEEE/ACM Design Automation Conference*, pp. 83–89, 2012.

[9] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 137–143, 2015.

[10] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory To Practice," *ACM SIGSAC Conference on Computer & Communications Security*, pp. 1601–1618, 2017.

[11] K. Azar, H. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 97–122, 2018.

[12] R. Jarvis and M. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," *US Patent no. 7,195,931*, 2007.

[13] J. Baukus, L. Chow, R. Cocchi, and B. Wang, "Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing," *US Patent no. 20120139582*, 2012.

[14] J. P. Skudlarek, T. Katsioulas, and M. Chen, "A Platform Solution for Secure Supply-Chain and Chip Life-Cycle Management," *IEEE Computer*, vol. 49, no. 8, pp. 28–34, 2016.

[15] S. Leef, "In Pursuit of Secure Silicon." https://rb.gy/ngjzfd, 2017. Last accessed on 09/28/20.

[16] D. P. Affairs, "DARPA Selects Teams to Increase Security of Semiconductor Supply Chain." https://www.darpa.mil/news-events/2020-05-27, 2020. Last accessed on 10/05/2020.

[17] R. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[18] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[19] Rambus, "Circuit camouflage technology." https://rb.gy/eowsah, 2017. Last accessed on 05/04/2020.

[20] Siemens, "TrustChain Security Platform." https://rb.gy/kcehg1, 2017. Last accessed on 09/28/2020.

[21] R. Torrance and D. James, "The State-of-the-Art in Semiconductor Reverse Engineering," *IEEE/ACM Design Automation Conference*, pp. 333–338, 2011.

[22] M. Yasin and O. Sinanoglu, "Transforming Between Logic Locking and IC Camouflaging," *IEEE International Design & Test Symposium*, pp. 1–4, 2015.

[23] K. Vättö, "Intel to Offer CPU Upgrades via Software for Selected Models." https://bit.ly/2Mnbn2j, 2011. Last accessed on 08/01/18.

[24] M. Zaman, A. Sengupta, D. Liu, O. Sinanoglu, Y. Makris, and J. Rajendran, "Towards Provably-Secure Performance Locking," *IEEE/ACM Design, Automation & Test in Europe*, pp. 1592–1597, 2018.

[25] A. Chakraborty, Y. Xie, and A. Srivastava, "GPU Obfuscation: Attack and Defense Strategies," *IEEE/ACM Design Automation Conference*, pp. 122:1–122:6, 2018.

[26] A. Chakraborty and A. Srivastava, "Hardware-Software Co-Design Based Obfuscation of Hardware Accelerators," *IEEE Computer Society Annual Symposium on VLSI*, pp. 547–552, 2019.

[27] M. Yasin, A. Sengupta, B. Schafer, Y. Makris, O. Sinanoglu, and J. Rajendran, "What to Lock?: Functional and Parametric Locking," *ACM Great Lakes Symposium on VLSI*, pp. 351–356, 2017.

[28] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking Against IC Counterfeiting and Overproduction," *IEEE/ACM Design Automation Conference*, pp. 1–6, 2017.

[29] J. Cassell, "Reports of Counterfeit Parts Quadruple Since 2009, Challenging US Defense Industry and National Security." https://bit.ly/2KWVkJh, 2012. Last accessed on 05/04/2020.

[30] V. V. Rao and I. Savidis, "Protecting Analog Circuits with Parameter Biasing Obfuscation," *IEEE Latin American Test Symposium*, pp. 1–6, 2017.

[31] N. G. Jayasankaran, A. S. Borbon, E. Sanchez-Sinencio, J. Hu, and J. Rajendran, "Towards Provably-Secure Analog and Mixed-Signal Locking Against Overproduction," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2018.

[32] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately Deobfuscating Integrated Circuits," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 95–100, 2017.

[33] Y. Shen and H. Zhou, "Double DIP: Re-Evaluating Security of Logic Encryption Algorithms," *ACM Great Lakes Symposium on VLSI*, pp. 179–184, 2017.

[34] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," *IEEE Asia and South Pacific Design Automation Conference*, pp. 342–347, 2016.

[35] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Pan, "Provably Secure Camouflaging Strategy for IC Protection," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 28:1–28:8, 2016.

[36] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," *IEEE/ACM Design, Automation & Test in Europe*, pp. 936–939, 2019.

[37] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," *IEEE Asian Hardware Oriented Security and Trust Symposium*, pp. 56–61, 2018.

[38] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 236–241, 2016.

[39] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 127–146, 2016.

[40] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly Stripping Functionality for Logic Locking: A Fault-based Perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[41] M. Massad, S. Garg, and M. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," *Network and Distributed System Security Symposium*, 2015.

[42] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1647–1659, 2017.

[43] H. Wee, "On Obfuscating Point Functions," *ACM Symposium on Theory of Computing*, pp. 523–532, 2005.

[44] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[45] A. Sengupta, M. Nabeel, M. Yasin, and O. Sinanoglu, "ATPG-Based Cost-Effective, Secure Logic Locking," *IEEE VLSI Test Symposium*, pp. 1–6, 2018.

[46] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, and M. Zuzak, "Keynote: A Disquisition on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[47] M. Yasin and B. Mazumdar and O. Sinanoglu and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[48] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 189–210, 2017.

[49] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann, 2018.

[50] F. Yang, M. Tang, and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) - Unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, 2019.

[51] NYU CSAW, "Logic Locking Conquest 2019." https://rb.gy/amdgdf, 2019. Last accessed on 09/28/2020.

[52] B. Olney and R. Karam, "Tunable FPGA Bitstream Obfuscation with Boolean Satisfiability Attack Countermeasure," *ACM Transactions on Design Automation of Electronic Systems*, vol. 25, no. 2, pp. 1–22, 2020.

[53] Cadence, "Genus Synthesis Solution." https://rb.gy/gqbpgd. Last accessed on 09/28/2020.

[54] Synopsys, "Design Compiler NXT." https://rb.gy/wgeq1m. Last accessed on 05/04/2020.

[55] Synopsys, "Synplify Pro." https://rb.gy/qscv0c. Last accessed on 09/28/2020.

[56] Xilinx, "Vivado." https://rb.gy/yrdbsa. Last accessed on 05/04/2020.

[57] Mentor Graphics, "Precision RTL." https://rb.gy/03zuos. Last accessed on 09/28/2020.

[58] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," *International Conference on Computer Aided Verification*, pp. 24–40, 2010.

[59] J. Yiu, *The Definitive Guide to the ARM Cortex-M3*. Newnes, 2009.

[60] B. Chetwynd, K. Bush, and K. Ingols, "CEP v2.0 Security Evaluation Targets." https://rb.gy/ssfjbk, 2019. Last accessed on 05/04/2020.

[61] Y. Kasarabada, D. Luria, and R. Vemuri, "Trust in IoT Devices: A Logic Encryption Perspective," *IFIP International Internet of Things Conference*, pp. 123–141, 2019.

[62] S. Keshavarz, C. Yu, S. Ghandali, X. Xu, and D. Holcomb, "Survey on Applications of Formal Methods in Reverse Engineering and Intellectual Property Protection," *Journal of Hardware and Systems Security*, vol. 2, no. 3, pp. 214–224, 2018.

[63] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2019.

[64] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[65] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," *Advances in Cryptology*, pp. 471–486, 1993.

[66] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *IEEE/ACM Design Automation Conference*, pp. 321–326, 1994.

[67] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse Engineering Digital Circuits Using Structural and Functional Analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2013.

[68] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "WordRev: Finding Word-level Structures in a Sea of Bit-Level Gates," *IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 67–74, 2013.

[69] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Springer Science & Business Media, 2006.

[70] P. Ashar, S. Devadas, and R. Newton, *Sequential Logic Synthesis*. Springer Science & Business Media, 2012.

[71] A. Sengupta, B. Mazumdar, M. Yasin, and O. Sinanoglu, "Logic Locking with Provable Security Against Power Analysis Attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[72] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," *Network and Distributed System Security Symposium*, pp. 1–14, 2015.

[73] C. Torng, "FreePDK45 and the Nangate Open Cell Library." https://rb.gy/azbenb, 2020. Last accessed on 02/12/2021.

[74] Xilinx, "Spartan-3 FPGA Family." https://rb.gy/pupygs, 2020. Last accessed on 02/12/2021.

[75] S. Davidson, "Characteristics of the ITC'99 Benchmark Circuits," *IEEE International Test Synthesis Workshop*, 1999.

[76] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," *NIST Special Publication*, vol. 800, p. 131A, 2011.

[77] G. Ateniese, A. Kiayias, B. Magri, Y. Tselekounis, and D. Venturi, "Secure Outsourcing of Cryptographic Circuits Manufacturing," *ACM International Conference on Provable Security*, pp. 75–93, 2018.

[78] C. Konstantinou, A. Keliris, and M. Maniatakos, "Privacy-Preserving Functional IP Verification Utilizing Fully Homomorphic Encryption," *IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition*, pp. 333–338, 2015.

[79] S. Devadas and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 13–27, 1991.

[80] J. Sweeney, M. Heule, and L. T. Pileggi, "Sensitivity Analysis of Locked Circuits," *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pp. 483–497, 2020.

[81] M. R. Garey and D. S. Johnson, *ACM Computers and Intractability*, vol. 174. Freeman San Francisco, 1979.

[82] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism," *arXiv preprint arXiv:1703.10187*, 2017.

[83] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic Obfuscation for Creating SAT-Unresolvable Circuits," *ACM Great Lakes Symposium on VLSI*, pp. 173–178, 2017.

[84] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-Based Attack on Cyclic Logic Encryptions," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 49–56, 2017.

[85] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic Locking and Memristor-Based Obfuscation Against CycSAT and Inside Foundry Attacks," *IEEE/ACM Design, Automation & Test in Europe*, pp. 85–90, 2018.

[86] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the Difficulty of FSM-based Hardware Obfuscation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 293–330, 2018.

[87] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation," *IEEE/ACM Design, Automation & Test in Europe*, pp. 534–539, 2019.

[88] S. Koteshwara, C. H. Kim, and K. K. Parhi, "Key-Based Dynamic Functional Obfuscation of Integrated Circuits Using Sequentially Triggered Mode-Based Design," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 79–93, 2018.

[89] Z. Kohavi and N. K. Jha., *Switching and Finite Automata Theory*. Cambridge University Press, 2009.

[90] R. Karmakar, H. Kumar, and S. Chattopadhyay, "Efficient Key-gate Placement And Dynamic Scan Obfuscation Towards Robust Logic Encryption," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[91] N. Limaye, A. Sengupta, M. Nabeel, and O. Sinanoglu, "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan Chain Access," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2019.

[92] M. El Massad, S. Garg, and M. Tripunitara, "Reverse Engineering Camouflaged Sequential Circuits Without Scan Access," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 33–40, 2017.

[93] J. Chen, D. Hermelin, and M. Sorge, "On Computing Centroids According to the p-Norms of Hamming Distance Vectors," *arXiv preprint arXiv:1807.06469*, 2018.

[94] D. Q. Naiman and H. P. Wynn, "Inclusion-Exclusion-Bonferroni Identities and Inequalities for Discrete Tube-Like Problems via Euler Characteristics," *IMSTAT Annals of Statistics*, vol. 20, no. 1, pp. 43–76, 1992.

[95] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to Differential Power Analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[96] S. Tajik, H. Lohrke, J.-P. Seifert, and C. Boit, "On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs," in *ACM SIGSAC Conference on Computer and Communications Security*, pp. 1661–1674, 2017.

[97] M. Rahman, S. Tajik, M. Rahman, M. Tehranipoor, and N. Asadizanjani, "The Key is Left Under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," tech. rep., Cryptology ePrint Archive, Report 2019/719, 2019, https://eprint.iacr.org/2019/719, 2019.

[98] S. Engels, M. Hoffmann, and C. Paar, "The End of Logic Locking? A Critical View on the Security of Logic Locking," *IACR Cryptology ePrint Archive*, vol. 2019, p. 796, 2019.

A SPI attack using divide and conquer

Algorithm 2: SPI attack with divide & conquer

Input: Locked cone $\mathcal{C}_{lock}$, Oracle O, Timeout T
Output: Protected input pattern candidates PIPs: PIP_{list}

$\mathcal{C}_{cp} \leftarrow$ generate_corrupted_circuit($\mathcal{C}_{lock}$)
$\boldsymbol{Nodes}_{sort} \leftarrow$ sort_by_protecting_inputs($\mathcal{C}_{cp}$)
$\boldsymbol{PIPs}_{cand} \leftarrow \varnothing$
for $node \in \boldsymbol{Nodes}_{sort}$ **do**
  $\mathcal{C}_{node} \leftarrow$ generate_subcircuit($\mathcal{C}_{cp}, node$)
  $PIT_{node} \leftarrow$ extract_PIT(C_{node}, T)
  $\boldsymbol{P_1} \leftarrow$ get_fully_specified_PI(PIT)
  $\boldsymbol{PIPs}_{verified} \leftarrow$ verify($\boldsymbol{PIPs}_{verified}, \mathcal{C}_{cp}, O$)
  if ($\boldsymbol{PIPs}_{verified} \neq \varnothing$) **then**
    return $\boldsymbol{PIPs}_{verified}$
  end
  $\boldsymbol{P_2} \leftarrow$ find_and_parse_split_PIs(PIT)
  $\boldsymbol{PIPs}_{verified} \leftarrow$ verify($\boldsymbol{P_2}, \mathcal{C}_{cp}, O$)
  if ($\boldsymbol{PIPs}_{verify} \neq \varnothing$) **then**
    return $\boldsymbol{PIPs}_{verified}$
  end
  $\boldsymbol{PIPs}_{cand} \leftarrow \boldsymbol{PIPs}_{cand} \bigcup \boldsymbol{P_1} \bigcup \boldsymbol{P_2}$
end
$\boldsymbol{PIPs}_{good} \leftarrow$ PIPs_with_most_specified_bits($\boldsymbol{PIPs}_{cand}$)
for $pip \in \boldsymbol{PIPs}_{good}$ **do**
  $pip_{full} \leftarrow$ brute_force($pip, \mathcal{C}_{lock}, O$)
  $\boldsymbol{PIP}_{list} \leftarrow \boldsymbol{PIPs}_{list} \bigcup \{pip_{full}\}$
end
return verify($\boldsymbol{PIP}_{list}, \mathcal{C}_{cp}, O$)

Alg. 2 describes the scalable version of the SPI attack that employs a divide-and-conquer strategy. The inputs to the algorithm are a locked cone $\mathcal{C}_{lock}$, the oracle O, and a timeout parameter T. First, the attack extracts the corrupted circuit $\mathcal{C}_{cp}$ by removing the correcting unit. The nodes in $\mathcal{C}_{cp}$ are sorted in descending order by the number of protecting inputs. Recall that the correcting unit has k primary and k key inputs, we refer to the primary inputs feeding the correcting unit as protecting inputs. Only the protecting inputs are included in the PIP and thus are of relevance for the attack.

For each sorted node, a sub-circuit is constructed and an attempt is made to extract its PIT. If the PIT can be extracted within the time limit T, the PIT is parsed for candidate PIPs using the two methods already used in Alg. 1, i.e., 1) fully specified PIPs and 2) split PIs. If the attacker is fortunate enough, the PIT may be computed for a sub-circuit with all protecting inputs and the recovered PIP may be fully specified (no don't care bit). In such a case, the PIP can be verified using the oracle. The attack completes successfully if a true PIP has been recovered.

In certain cases, the PIT may be extracted only for sub-circuits that have only a subset of protecting inputs. In such scenarios, our attack targets the candidate PIPs with the largest number of specified bits. Brute-force or even attacks such as SAT/SMT attack may be used to determine the values of the remaining bits in the PIP.

B Success Rate of Structural Attacks

Continuing the discussion in Section 3.1, we consider two scenarios for attackers.

Well-informed attacker. Let us first assume that the attacker knows whether F_{cp}^{ON} is constructed by adding a PIP or removing a PIP from F_{orig}^{ON}. Later, we also consider the case where the attack does not know this information. Considering the cases where the PIP is merged, i.e., Case 1(a) or 2(a), $Pr(\mathcal{A}_{\mathcal{S}})$, i.e., the probability of success for a structural attack to recover the PIP from the PIT is

$$Pr(\mathcal{A}_{\mathcal{S}}) \leq max\left\{\frac{1}{|\mathcal{S}_{u1}|}, \frac{1}{|\mathcal{S}_{u2}|}\right\}.$$

When the PIP is merged, i.e., Case 1(b) and 2(b), the attacker has to account for all minterms in either F_{cp}^{ON} or F_{cp}^{OFF}. Then,

$$Pr(\mathcal{A}_{\mathcal{S}}) \leq max\left\{\frac{1}{|\mathcal{S}_{m1}|}, \frac{1}{|\mathcal{S}_{m2}|}\right\}.$$

Uninformed attacker. The CAC techniques may construct the $\mathcal{C}_{cp}$ by XORing $\mathcal{C}_{orig}$ with a PIP, i.e., the PIP may be either added or removed. Thus, the attacker has to account for both addition and removal cases. When the PIP remains isolated, the success rate for the attacker is $\frac{1}{|\mathcal{S}_{u1}|+|\mathcal{S}_{u2}|}$, implying that choosing unmergeable PIPs is at most $(|\mathcal{S}_{u1}|+|\mathcal{S}_{u2}|)$-secure against structural attacks. When the PIP is merged, the search space for attacker is the set of all minterm contained in either F_{cp}^{ON} or F_{cp}^{OFF}, i.e., $Pr(\mathcal{A}_{\mathcal{S}}) = \frac{1}{|\mathcal{S}_{m1}|+|\mathcal{S}_{m2}|} = \frac{1}{2^k}$.

C Proof of Theorem 1

Theorem 1: *Satisfying Dist2 property is sufficient to achieve* β*-security against structural attacks on Boolean circuits represented in SOP form, where* $\beta = |\mathcal{S}|$.

Proof: We assume that the attacker has access to 1) the corrupted circuit implemented in SOP format and 2) the PIT of the corrupted circuit. Locking using a D2PIP ensures that the PIT does not directly reveal the PIPs to the attackers. The search space $\mathcal{S}$ for an attacker is the set of potential D2PIP candidates an attacker has to account for. Since the attacker does not know whether a D2PIP has been added or removed, he/she must consider both cases.

Case 1. Removing a D2PIP. Removing a D2PIP from F_{orig}^{ON} yields,

$$F_{cp}^{ON} = F_{orig}^{ON} \setminus \{D2PIP\},\ \forall PI_i \in F_{cp}^{ON},\ D(D2PIP, PI_i) \geq d.$$

For an attacker, the search space $\mathcal{S}_1$ is the set of all potential PIPs that are at least distance two (or more generally distance d) away from the PIs of F_{cp}^{ON}, i.e.,

$$\mathcal{S}_1 = \{pi \,|\, D(pi, PI_i) \geq d,\ \forall PI_i \in F_{cp}^{ON}\} = \bigcap_{i=1}^{P} \{pi \,|\, D(pi, PI_i) \geq d\}, \tag{3}$$

where, P is the number of PIs in F_{cp}^{ON} and $pi \in \{0,1\}^k$.

Case 2: Adding a D2PIP. Adding a D2PIP into F_{orig}^{ON} can be considered as removing a D2PIP from F_{orig}^{OFF}. Hence,

$$F_{cp}^{OFF} = F_{orig}^{OFF} \setminus \{D2PIP\},\ \forall PI_i' \in F_{cp}^{OFF},\ D(D2PIP, PI_i') \geq d.$$

Therefore, the search space $\mathcal{S}_2$ for the attacker is,

$$\mathcal{S}_2 = \{pi \,|\, D(pi, PI_i') \geq d,\ \forall PI_i' \in F_{cp}^{OFF}\} = \bigcap_{i=1}^{P'} \{pi \,|\, D(pi, PI_i') \geq d\}, \tag{4}$$

where, P' is the number of PIs in F_{cp}^{OFF} and $pi \in \{0,1\}^k$.

Since an attacker does not know whether the corrupted circuit is constructed by adding or removing a D2PIP, the overall search space

$\mathcal{S}$ for the attacker is $\mathcal{S} = \mathcal{S}_1 \bigcup \mathcal{S}_2$. Note that $\mathcal{S}_1$ and $\mathcal{S}_2$ are disjoint. A PI that belongs to $\mathcal{S}_1$ must belong to F_{cp}^{OFF}, and a PI that belongs to $\mathcal{S}_2$ must belong to F_{cp}^{ON}. F_{cp}^{OFF} and F_{cp}^{ON} are disjoint. Hence, the size of the overall search space for the attacker is $|\mathcal{S}| = |\mathcal{S}_1| + |\mathcal{S}_2|$. The probability of success for an attacker is $\frac{1}{|\mathcal{S}|}$. Thus, from Definition 3, locking only D2PIPs in two-level SOP circuits achieves a security level of $\beta = |\mathcal{S}| = |\mathcal{S}_1| + |\mathcal{S}_2|$ against structural attacks. □

D Computing Upper and Lower Bounds for the Size of the Search Space

For a given circuit, the exact computation of $|\mathcal{S}_1|$ and $|\mathcal{S}_2|$ is an NP-Hard problem [93]. However, we can determine the upper and lower bounds using the inclusion-exclusion inequality [94].

Bounds for Case 1. This analysis is based on the case where a PIP is removed from F_{orig}^{ON}, i.e., $F_{cp}^{ON} = F_{orig}^{ON} \setminus \{D2PIP\}$, and $\forall PI_i \in F_{cp}^{ON},\ D(D2PIP, PI_i) \geq d$. The search space $\mathcal{S}_1$ for the attacker is the set of possible PIs that are distance $\geq$ d from the PIP.

$$\mathcal{S}_1 = \bigcap_{i=1}^{P} \{pi | D(pi, PI_i) \geq d\} = \overline{\bigcup_{i=1}^{P} \{pi | D(pi, PI_i) < d\}}, \quad (5)$$

where, P is the number of PIs in the F_{cp}^{ON}, and pi is a candidate PI. Therefore, the size of the search space is

$$|\mathcal{S}_1| = 2^k - \left| \bigcup_{i=1}^{P} \{pi | D(pi, PI_i) < d\} \right|. \quad (6)$$

Further, we could estimate the upper and lower bounds of $|\mathcal{S}_1|$ by using the inequality of inclusion-exclusion principle [94]

$$C_1 - C_2 \leq \left| \bigcup_{i=1}^{P} \{pi | D(pi, PI_i) < d\} \right| \leq C_1, \quad (7)$$

where, C_1 and C_2 are

$$\begin{cases} C_1 = \sum_{i=1}^{P} |\{pi | D(pi, PI_i) < d\}| \\ C_2 = \sum_{1 \leq i < j \leq P} |\{pi | D(pi, PI_i) < d, D(pi, PI_j) < d\}|. \end{cases} \quad (8)$$

Calculating C_1. Assume that there are w_i don't care bits i^{th} PI in PI_i, which is the PIT of F_{cp}^{ON}, where $i \in \{1, 2, ..., P\}$. It follows that

$$C_1 = \sum_{i=1}^{P} |\{pi | D(pi, PI_i) < d\}| = \sum_{i=1}^{P} \sum_{t=0}^{d-1} 2^{w_i} \times \binom{k - w_i}{t}. \quad (9)$$

Calculating C_2. Calculating C_1 does not require inter-PI distance, which is not the case for C_2. Assume that for two PIs PI_i and PI_j in F_{cp}^{ON}, $1 \leq i < j \leq P$. w_{ij} is the number of don't care bits in PI_i and PI_j, s_{ij} is the number of specified bits that have the same value in PI_i and PI_j, f_{ij} is the number of specified bits that are mismatched in PI_i and PI_j, h_{ij}^i is the number of bits that are specified in PI_i but don't care PI_j, and h_{ij}^j is the number of bits that are don't care in PI_i but specified in PI_j. Therefore,

$$C_2 = \sum_{1 \leq i < j \leq P} \left[\sum_{t_1=0}^{d-1} \sum_{t_2=0}^{d-1} 2^{w_{ij}} \times \sum_{g=0}^{min(t_1,t_2)} \binom{s_{ij}}{g} \times \sum_{a_1=0}^{t_1-g} \sum_{a_2=0}^{t_2-g} \binom{h_{ij}^i}{a_1} \binom{h_{ij}^j}{a_2} \binom{f_{ij}}{t_1 - g - a_1, t_2 - g - a_2} \right]. \quad (10)$$

Therefore, the lower and upper bounds of $|\mathcal{S}_1|$ are

$$\begin{cases} LB = 2^k - C_1 \\ UB = 2^k - C_1 + C_2, \end{cases} \quad (11)$$

where, C_1 and C_2 are specified in Eq. (9) and Eq. (10), respectively.

Bounds for Case 2. In Case 2, the process of adding the D2PIP into F_{orig}^{ON}, could be considered as removing the D2PIP from F_{orig}^{OFF}, i.e., $F_{cp}^{OFF} = F_{orig}^{OFF} \setminus \{D2PIP\}$, and $\forall PI_i' \in F_{cp}^{OFF}, D(D2PIP, PI_i') \geq d$.

The calculation of the upper and lower bounds of $|\mathcal{S}_2|$ is similar to that for Case 1. However, the analysis that was applied to PIs of F_{cp}^{ON} to will now be applied to PIs of F_{cp}^{OFF}. Therefore, the upper bound UB' and the lower bound LB' of $|\mathcal{S}_2|$ are:

$$\begin{cases} LB' = 2^k - C_1' \\ UB' = 2^k - C_1' + C_2', \end{cases} \quad (12)$$

where, C_1' and C_2' are

$$\begin{cases} C_1' = \sum_{i=1}^{P'} \sum_{t=0}^{d-1} 2^{w_i'} \times \binom{k - w_i'}{t} \\ C_2' = \sum_{1 \leq i < j \leq P'} \left[\sum_{t_1=0}^{d-1} \sum_{t_2=0}^{d-1} 2^{w_{ij}'} \times \sum_{g=0}^{min(t_1,t_2)} \binom{s_{ij}'}{g} \times \sum_{a_1=0}^{t_1-g} \sum_{a_2=0}^{t_2-g} \binom{h_{ij}'^i}{a_1} \binom{h_{ij}'^j}{a_2} \binom{f_{ij}'}{t_1 - g - a_1, t_2 - g - a_2} \right]. \end{cases} \quad (13)$$

Assume PI_i' is the i'^{th} PI in F_{CP}^{OFF}'s PIT, where $i \in \{1, 2, \ldots, P'\}$. The definitions of the parameters are slightly different from those in Case 1: w_i' is the number of don't care bits in PI_i', w_{ij}' is the number of don't care bits in PI_i' and PI_j', s_{ij}' is the number of specified bits that have the same value in PI_i' and PI_j', f_{ij}' is the number of specified bits that are mismatched in PI_i' and PI_j', $h_{ij}'^i$ is the number of bits that are specified in PI_i' but don't care PI_j', and $h_{ij}'^j$ is the number of bits that are don't care in PI_i' but specified in PI_j'.

E Discussion: Physical Aspects of Security

Even cryptographic techniques with proven guarantees are subjected to physical and side-channel attacks [95, 96]. Thus, logic locking is also susceptible to physical attacks [97, 98] Such attacks can be thwarted by 1) storing the key in a tamper-proof memory and 2) adopting defenses developed for side-channel attacks [71].