# A TECHNICAL REPORT ON KEY MANAGEMENT AND VULNERABILITY ASSESSMENT OF LOGIC OBFUSCATION

Kimia Zamiri Azar

Department of Electrical and Computer Engineering (ECE)
George Mason University, Fairfax, VA

Fall 2019

# Table of Contents

Page

List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 0

1 Introduction to Hardware Obfuscation . . . . . . . . . . . . . . . . . . . . 1

2 Threats on Logic Locking: A Decade Later . . . . . . . . . . . . . . . . . . 7

2.1 Stage 1: Test-Based Attacks . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.1 Brute Force Attack . . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.2 Sensitization Attack . . . . . . . . . . . . . . . . . . . . . . . 8

2.1.3 Random-based Hill-Climbing Attack . . . . . . . . . . . . . . . . 9

2.2 Stage 2: SAT Attack . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.3 Stage 3: Post-SAT Attacks . . . . . . . . . . . . . . . . . . . . . . . . 11

2.3.1 Removal Attack . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.3.2 Signal Probability Skew (SPS) Attack . . . . . . . . . . . . . . . 12

2.3.3 Bypass Attack . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3.4 AppSAT Attack . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.5 Double-DIP Attack . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.6 Bit-Flipping Attack . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.7 AppSAT Guided Removal Attack . . . . . . . . . . . . . . . . . . 15

2.3.8 Sensitization Guided SAT Attack . . . . . . . . . . . . . . . . . 16

2.3.9 Functional Analysis Attack . . . . . . . . . . . . . . . . . . . . 17

2.3.10 CycSAT Attack . . . . . . . . . . . . . . . . . . . . . . . . . . 18

2.3.11 Behavioral SAT (BeSAT) Attack . . . . . . . . . . . . . . . . . 20

2.4 Stage 4: SMT Attack . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

2.5 Discussion & Opportunities . . . . . . . . . . . . . . . . . . . . . . . . 23

3 SMT Attack: Next Generation Attack on Obfuscated Circuits . . . . . . . . . . 26

3.1 Boolean Logic Obfuscation . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.2 Behavioral logical obfuscation . . . . . . . . . . . . . . . . . . . . . . . 28

3.3 Attack Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.4 Limitation of SAT Attack . . . . . . . . . . . . . . . . . . . . . . . . . 30

3.5 SMT Solver . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.5.1 SMT Usage and Capabilities . . . . . . . . . . . . . . . . . . . . . . . 31
3.6 SMT Attack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.6.1 Attack Mode 1: SMT reduced to SAT Attack . . . . . . . . . . . . . 36
3.6.2 Attack Mode 2: Eager SMT Attack . . . . . . . . . . . . . . . . . . . 38
3.6.3 Attack Mode 3: Lazy SMT Attack . . . . . . . . . . . . . . . . . . . . 44
3.6.4 Attack Mode 4: Accelerated Lazy SMT Attack (AccSMT) . . . . . . 46
3.7 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
3.7.1 Evaluation of SMT reduced to SAT Attack . . . . . . . . . . . . . . 52
3.7.2 Evaluation of Eager SMT Attack . . . . . . . . . . . . . . . . . . . . . 53
3.7.3 Evaluation of Lazy SMT Attack . . . . . . . . . . . . . . . . . . . . . 55
3.7.4 Evaluation of Lazy AccSMT Attack . . . . . . . . . . . . . . . . . . . 57
4 COMA: Communication and Obfuscation Management Architecture . . . . . . . 61
4.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
4.2 Proposed COMA Architecture . . . . . . . . . . . . . . . . . . . . . . . . 64
4.2.1 2.5D-COMA: Protecting 2.5D package integrated system solutions . 64
4.2.2 R-COMA: Protecting IoT devices . . . . . . . . . . . . . . . . . . . . 69
4.3 Implementation Detail of COMA . . . . . . . . . . . . . . . . . . . . . . 70
4.3.1 Configurable Switching Network (CSN) . . . . . . . . . . . . . . . . 71
4.3.2 Authenticated Encryption with Associated Data . . . . . . . . . . . 72
4.3.3 Random Number Generator (RNG) . . . . . . . . . . . . . . . . . . . 74
4.3.4 PUF and Secure PUF Readout . . . . . . . . . . . . . . . . . . . . . . 74
4.4 COMA Resistance against various Attacks . . . . . . . . . . . . . . . . . 76
4.4.1 Assumed Attacker Capabilities . . . . . . . . . . . . . . . . . . . . . 76
4.4.2 Reverse Engineering . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
4.4.3 Algebraic Attacks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
4.4.4 Counterfeiting and Overproduction . . . . . . . . . . . . . . . . . . . 79
4.4.5 Removal attacks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.5 COMA Implementation Results . . . . . . . . . . . . . . . . . . . . . . . 80
4.5.1 COMA Area Overhead . . . . . . . . . . . . . . . . . . . . . . . . . . 80
4.5.2 COMA Performance . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
4.5.3 COMA performance in LCC mode . . . . . . . . . . . . . . . . . . . 83
4.6 Comparing COMA with Prior Work . . . . . . . . . . . . . . . . . . . . . 87
Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

# List of Tables

Table Page

2.1 Classification of KGs in Sensitization Attack. . . . . . . . . . . . . . . . . . . 9

2.2 Comparison of proposed attacks/defenses. . . . . . . . . . . . . . . . . . . . . . 24

3.1 ISCAS-85 Benchmarks and their Characteristics. . . . . . . . . . . . . . . . . 52

3.2 Execution Time of SAT vs SMT (Attack Mode 1). . . . . . . . . . . . . . . 53

3.3 Execution Time of SMT Attack in the Eager Mode (Attack Mode 2). . . . . 54

3.4 Execution Time of SMT Attack in the Lazy Mode (Attack Mode 3). . . . . 56

3.5 Comparing the AccSMT (Attack Mode 4) with the Original SAT Attack. . 56

3.6 Execution Time and the Number of Iterations of AccSMT (Attack Mode 4). 59

4.1 Main features of the two proposed COMA variants. . . . . . . . . . . . . . 80

4.2 Resource Utilization of the COMA Architecture. . . . . . . . . . . . . . . . 81

4.3 Resource Utilization for ASIC implementation of COMA. . . . . . . . . . . 82

4.4 Optimized results of COMA Architecture. . . . . . . . . . . . . . . . . . . . . 82

4.5 SAT Execution Time on Blocking CSN and a Close to Non-blocking CSN . 84

4.6 COMA vs. FORTIS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

4.7 Area Overhead of COMA vs. FORTIS. . . . . . . . . . . . . . . . . . . . . . 86

# List of Figures

Figure Page

2.1 Categorization of attacks against logic locking schemes. . . . . . . . . . . . 7

2.2 SAT Attack Iterative Flow. . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.3 Flipping Structure of SARLock and Anti-SAT. . . . . . . . . . . . . . . . . 12

2.4 SFLL-HD while $h = 0$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.1 10-Year History of Logic Obfuscation. . . . . . . . . . . . . . . . . . . . . . 27

3.2 ASIC Design Flow Integrated with Obfuscation/Activation. . . . . . . . . . 30

3.3 Overall Structure of Tunable Delay Key-Gate (TDK). . . . . . . . . . . . . 31

3.4 Overall Architecture of SMT Attack for Circuit Deobfuscation. . . . . . . . 35

3.5 Translation Table to Key Programmable Gates (KPG). . . . . . . . . . . . 35

3.6 From Obfuscated Circuit to SAT Circuit. . . . . . . . . . . . . . . . . . . . 37

3.7 SMT Execution Flow. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

3.8 Conversion Flow in SMT using Graph Theory Solver. . . . . . . . . . . . . 39

3.9 Various Delay Components of a Timing Path. . . . . . . . . . . . . . . . . 40

3.10 A Hybrid Obfuscation Scheme. . . . . . . . . . . . . . . . . . . . . . . . . . 49

3.11 Comparing the Performance of SMT-attack with that of original SAT-attack. 53

3.12 Set of potentially valid keys reduces in each iteration of SMT or SAT attack. 58

3.13 Key Reduction Rate of the Original SAT Attack and the AccSMT Attack. 58

4.1 FORTIS: Overall Architecture. . . . . . . . . . . . . . . . . . . . . . . . . . 63

4.2 Proposed COMAs for (left) 2.5D and (right) IoT-based/remote devices. . . 65

4.3 Modes of Encrypted Communication in COMA: (a) DCC, (b) LCC. . . . . 68

4.4 2.5D-COMA Architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . 69

4.5 Logarithmic Network (a) Omega-based Blocking, (b) near Non-blocking. . 71

4.6 The t-test Results for Pr and UnPr version of AES-GCM and ACORN. . . 77

4.7 Total Execution Time for AES-GCM + AES-CTR and ACORN + Trivium. 83

4.8 Energy Breakdown in COMA. . . . . . . . . . . . . . . . . . . . . . . . . . 85

4.9 The Power Consumption at LCC mode: (a) While $P < U$, (b) While $P > U$. 87

# Chapter 1: Introduction to Hardware Obfuscation

The cost of building a new semiconductor fab was estimated to be $5.0 billion in 2015, with large recurring maintenance costs [1][2], and sharply increases as technology migrates to smaller nodes. To reduce the fabrication cost, most of the manufacturing and fabrication is pushed offshore [1]. However, many of the offshore fabrication facilities are considered to be untrusted. Manufacturing in untrusted foundries has raised concern over potential attacks in the manufacturing supply chain, with an intimate knowledge of the fabrication process, the ability to modify and expand the design prior to production, and an unavoidable access to the fabricated chips during testing. Accordingly, fabrication in untrusted foundries has introduced multiple forms of security threats from supply chain including that of overproduction, Trojan insertion, Reverse Engineering (RE), Intellectual Property (IP) theft, and counterfeiting [2].

To counter these threats, various hardware *design-for-trust* techniques have been proposed, including *watermarking*, *IC metering*, *split manufacturing*, *IC camouflaging*, and *logic locking* [3–7]. The watermarking and IC metering techniques are passive protection models that could be used to detect overproduction or illegal copies, however, they cannot prevent IP theft or overproduction. The *Camouflaging* techniques use logic gates (or other physical structures such as dummy vias) with high structural similarity, that are indistinguishable from one another to protect against reverse engineering. However, camouflaging is only effective against post-manufacturing attempt(s) of reverse engineering, while it provides no limitations against a foundry's attempt at reverse engineering, as a foundry has access to all masking layers and is not trapped by structural ambiguity for being able to logically extract a netlist. The obfuscation (*logic locking*) [7] on the other hand, introduce limited programmability by inserting key programmable gates to hide or lock the functionality. By using obfuscation, the target chip produces the correct output only when the key inputs are correct. The purpose of obfuscation is to protect against RE at an untrusted foundry. By using obfuscation, even by having all mask information, the attacker cannot generate the correct functionality of a circuit without the correct key values, and such key values are not shared with the manufacturer.

logic locking techniques, however did not end the threat against IP piracy (or other related concerns), as these solutions that were proposed over the last decade were broken using various carefully crafted attacks. A decade of research in this area, has resulted in a wide range of defense and attack solutions. Shortly after the introduction of first published obfuscation schemes, a new and powerful attack based on Boolean Satisfiability (SAT) was formulated and revealed [9, 10]. In this attack model, the attacker has access to a reverse engineered but obfuscated netlist, and a functional and unlocked chip. Using this attack model, the formulated Boolean Satisfiability Attack (SAT Attack) can effectively break all previously proposed logic encryption techniques, including random insertion (RLL), fault-analysis (FLL), interference-based logic locking (SLL), and logic barriers [7, 11–17]. The SAT solver iteratively eliminates sets of incorrect keys and finds the correct key within a small time, and unlike Brute force attack that requires attack time exponential with respect to the number of inputs, its execution time grows almost polynomially. Existing SAT attack, which can be modeled with query-by-disagreement (QBD) or uncertainty-sampling, minimizes the number of queries (inputs) required to learn (deobfuscate) the target function (obfuscated logic). Also, SAT attack terminates when no more disagreeing inputs can be found, at which time the attack guarantees to find the correct key. However, to defend against powerful SAT attacks, different obfuscation schemes have been proposed, such as SARLock and Anti-SAT [15,18,19]. However, further research illustrated that some of these locking schemes are vulnerable to other types of attacks such as Signal Probability Skew (SPS) and removal attacks [20].

In addition, introducing approximate-based attacks, such as AppSAT [21] or Double-DIP [22] worsens the problem. Unlike the existing SAT attack, which needs exact learning model, approximate-based attacks can be modeled using approximate learning problems, such as the probably-approximately-correct (PAC) setting [23]. Based on the PAC model, an attack $A$, with a probability of $\lambda$, will provide an $\epsilon$-approximation (approximately correct) of the target function (obfuscated logic). Note that, an $\epsilon$-approximation of the target function is a function with only $\%\epsilon$ ($\epsilon \in O(\frac{1}{2^n})$) disagreement with correctly unlocked circuit. Accordingly, the approximate SAT attacks can find an approximate key which produces a very small error ($\%\epsilon$) in the behavior of the unlocked circuit in comparison with a correctly unlocked circuit. The approximate attacks are shown to effectively find an approximate key for SAT-resilient defenses including SARLock [18], and Anti-SAT [19]. Furthermore, Bypass attack [24] is also proposed for creating an auxiliary circuit that recovers the flipped output(s) while approximate key is applied. Then it adds a bypass circuit to correct the wrong output(s) when input pattern(s) cause incorrect output(s). Consequently, it is able to eliminate even small error in the behavior of the unlocked circuit by approximate key, and behave completely the same compared to correctly unlocked chip. In Section 2 we review many of the obfuscation solutions and attack mechanisms in more details, summarize and compare the effectiveness of obfuscation solutions against these attacks, and describe the strength and weaknesses of various obfuscation and attack solutions [25].

In general, the SAT attack benefits from the Directed Acyclic Graph (DAG) based nature of input netlist and the ability of SAT-attack to logically model the obfuscation into a satisfiability problem. To counter the SAT attack, recently some design obfuscation schemes have been proposed to violate these assumptions. For instance, in the approach adopted in [26], the DAG nature of netlist is altered by introducing cycles into the netlist for the purpose of trapping a SAT attack. Another example is the approach adopted in [27], where the obfuscation, in addition to logical properties of the netlist, targets the setup and hold properties (timing properties) of the circuit as a locking mechanism. Considering that setup and hold time are not logical properties, they cannot be translated into CNF statements for formulating a SAT attack. However, in section 3 by proposing the Satisfiability Modulo Theory (SMT)-based attack, we illustrate that even using these non-logical properties for obfuscation, does not increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security.

In addition, to remain hidden, apart from being resistant to different attack models on obfuscated circuits, the IC should also resist passive, active, or destructive attacks that could be deployed to read the key values. Hence, neither the activation of such devices nor the storage of key values in them should expose or leak the key information. Activation of an obfuscated IC requires storing the activation key in a secure and tamper-proof memory. [28, 29]. However, there exist a group of applications that could use an alternative key storage. This alternative solution is to store the key outside the IC, where the IC is activated every time it is needed. This option requires constant connectivity to the key management source and a secure communication for key exchange to prevent any leakage of the key. This solution allows an IC designer to store the chip unlock key outside of an untrusted chip. So, no secure and tamper-proof memory is needed. Since the key is stored outside the untrusted chip, a constant connectivity to an obfuscation key-management solution is an indispensable requirement for this category of devices. This requirement could be easily met for two prevalent groups of architectures: (1) 2.5D package-stack devices where a single trusted chip is used for key management and activation of multiple obfuscated ICs manufactured in untrusted foundries, and (2) IoT devices with constant connectivity to the cloud/internet.

In 2.5D package-integrated ICs, similar to DARPA SPADE architecture [30], a chip which is fabricated in a trusted foundry, but in a larger technology node, is packaged with an untrusted chip fabricated in an untrusted foundry in a smaller technology node. The resulting solution benefits from the best features of both technologies: The untrusted chip may be used as an accelerator, providing the resulting hybrid solution with the much-needed scalability (higher speed and lower power), while the trusted chip provides the means of trust and security. The untrusted chip is isolated from the outside world and any exchange of information to/from untrusted chip passes through the trusted chip.

The second group of devices in this category are IoT devices, where constant connectivity is their characterizing features. In these solutions the obfuscation key could be stored in the cloud, and activation of an IoT device could be done remotely. This model allows custom, monitored, and service oriented activation (Activation As A Service). An additional advantage is removing the possibility of extracting an unlock key from a non-volatile memory that otherwise would have to be used for storing the obfuscation unlock key. Examples of which are IoT devices used for providing various services, military drones activated for a specific mission, video decryption services for paid pay-per-view transactions, etc., where a device has to operate in an unsafe environment and is at risk of capture and reverse engineering. In these applications, the IC fabricated in an untrusted foundry is activated either every time it is powered up, or for certain time intervals. The key vanishes after the service is performed, or when the device is powered down. The activation of such devices is performed using a remote key management service (in the cloud or at a trusted base-station), and the key exchange to/from these devices should be secured.

In both 2.5D system solutions and IoT devices, the need for implementation of a tamper-proof memory, for storage of IC activation key, in an untrusted process is removed. Some reasons why implementing a secure memory in an untrusted foundry may be undesired, or practically unfeasible include:

*Availability:* The targeted foundry may not offer the required process for implementing a secure memory with the desired features. An example could be the requirement for storing sensitive information in magnetic tunnel junction (MTJ) memories to prevent probing and attacks that could be deployed against flash-based NVMs. Fabricating such ICs requires a hybrid process that supports both CMOS and MTJ devices, which may be unsupported by the targeted foundry.

*Verified Security:* The secure memory may be available in the targeted technology, however not be fully tested and verified in terms of its resistance against different attacks.

*Cost:* Implementing secure memory requires additional fabrication layers and processing steps, increasing the cost of manufacturing. Increasing the silicon area is a far cheaper solution than increasing the number of fabrication layers.

*Reusability:* In 2.5D system solutions, a trusted chip could be shared by multiple untrusted chips, manufactured in different foundries. Moving the secure memory to the trusted chip removes the need for implementing the secure memory in all utilized processes. The trusted chip could be designed once with utmost security for protection and integrity of data. This also reduce the cost of manufacturing untrusted chips by removing the need for additional processing steps for implementing secure memory.

*Ease of Design:* Implementing secure memory requires pushing the design through non-standard physical design flow to implement the tamper-proof layers in silicon and package. In addition, the non-volatile nature of tamper-proof memory requires read and write at elevated voltages, increasing the burden on the power-delivery network design. Reuse of a trusted chip with a tamper proof memory that could manage activation of other obfuscated ICs, relaxes the design requirement of ICs to standard physical design and fabrication process.

In section 4, we propose the COMA key-management and communication architecture for secure activation of obfuscated circuits that are manufactured in untrusted foundries and meet the constant connectivity requirement, namely ICs that belong to a) 2.5 package-integrated and b) IoT solutions. We describe two variants of our proposed solutions: The first variant of COMA is used for secure activation of IPs within 2.5D package-integrated devices (similar to DARPA SPADE). The second variant of COMA is used for secure activation of connected IoT devices. The proposed COMA allows us to (1) push the obfuscation key and obfuscation unlock mechanism off of an untrusted chip, (2) make the key a moving target by changing it for each unlock attempt, (3) uniquely identify each IC, (4) remove the need to implementing a secure memory in an untrusted foundry, and (5) utilize two novel mechanisms for ultra-secure or ultra-fast encrypted communication [31].

# Chapter 2: Threats on Logic Locking: A Decade Later

As illustrated in Fig. 2.1, the defense and attack solutions related to hardware obfuscation, based on functionality, capability, effectiveness and time-line are categorized into four categories: (1) *Test-Inspired Attacks* that were mostly inspired from test concepts and attempted to discover the obfuscation key value based on the location of KGs, described in Section 2.1. (2) SAT Attack, formulation and revelation of which significantly affected the direction and presumed assumptions of the hardware obfuscation research community, explained in Section 2.2. (3) Post-SAT Attacks where the focus of hardware security researchers changed to the design of an attack against obfuscation solutions that resist the SAT attack, explained in 2.3. And (4) SMT Attack as a universal attack platform capable of instantiating many theory solvers to act as pre- post- or co- processors to the SAT solver, described in Section 2.4.

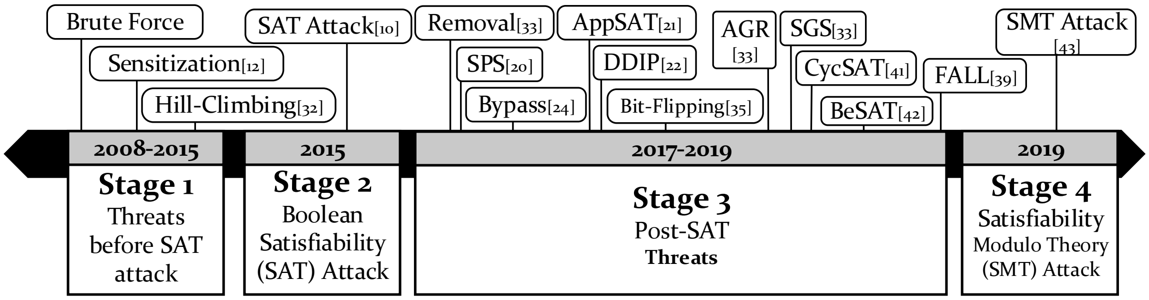


Figure 2.1: Categorization of attacks against logic locking schemes.

## 2.1 Stage 1: Test-Based Attacks

### 2.1.1 Brute Force Attack

The *brute force* attack is the most intuitive attack against obfuscated circuits. This attack exhaustively search for the correct key by testing all key and input values. For instance, assuming that adversary has access to the reverse-engineered netlist, and considering that the circuit has four $PI$s ( $i_{0..4}$) and two $KI$s ($k_{0..1}$), an exhaustive search may result in applying of $2^{2+4} = 64$ test patterns (in the worst case) and checking the output against an activated (functionally correct) chip to verify correctness. Based on the number of primary inputs ($|PI|$) and the number of key bits ($|KI|$), the number of possible test patterns is ($2^{|PI|+|KI|}$). Hence, the search space for a brute force attack is extremely large, making the attack even for small circuits and small number of keys unfeasible in a reasonable amount of time. For example, a small circuit with 20 input pins, which is obfuscated with 80 key gates poses $2^{100}$ possible test pattern. An attacker can reduce the number of test patterns using functional test or random test, in which the exponential impact of $|PI|$s will be eliminated, and only $2^{|KI|} \times (func\_test\_patterns)$ is required for brute force attack. But even with this change, the attack time is exponential with respect to the number of key gates. EPIC [7] used a random KG insertion policy referred to as random logic locking (RLL). Using RLL, EPIC reasoned that by replacing a small percentage of gates (or insertion of KGs), the obfuscation can resist brute force attacks.

### 2.1.2 Sensitization Attack

After introducing EPIC [7], Rajendran *et al.* [12] proposed a *sensitization* attack, which determines individual key values, in a time linear to the $|KI|$, by applying patterns that sensitize key values to the output. As its name implies, sensitization of an internal wire (key bit) **L** to an output **O** means that the value of **L** can be propagated to **O** and thus any change on **L** is observable on **O**. After determining an input pattern that propagates the value of the key-bit to the output, the attacker applies the input pattern to a functional

Table 2.1: Classification of KGs in Sensitization Attack.

| **Term** | **Description** | **Strategy used by attacker** |
|---|---|---|
| **Runs of KGs** | Back-to-Back KGs | Replacing by a Single KG |
| **Isolated KGs** | No Path between KGs | Finding Unique Pattern per KG (Golden Pattern (GP)) |
| **Dominating KGs** | $k1$ is on Every Path between $k0$ and $PO$s | Muting $k0$,<br>Sensitizing $k1$ |
| **Concurrently Mutable Convergent KGs** | Convergent at a Third Gate<br>Both can be Propagated to $PO$s | Muting $k0/k1$,<br>Sensitizing $k1/k0$ |
| **Sequentially Mutable Convergent KGs** | Convergent at a Third Gate<br>One can be Propagated to $PO$s | Determining $k1$ by GP,<br>Update the Netlist, Target $k0$ |
| **Non-Mutable Convergent KGs** | Convergent at a Third Gate<br>None can be Propagated to $PO$s | Brute Force Attack |

IC (An IC activated and programmed with the correct key that could be obtained from market). The correct key value will be propagated to an output by applying this pattern to the functional IC. The attacker observe and record this output as the value of the sensitized key-bit. The propagation of a key-bit to the output is heavily depending on the location of the KGs, hence, they classify KGs based on their location and discuss corresponding attack strategies for each case. The summary of strategies and techniques used in the sensitization attack is reflected in Table 2.1. To prevent sensitization attack they proposed SLL, in which the KGs are inserted in locations with maximum mutual interference. In SLL the attacker cannot sensitize the key-bit values to a primary output. Similar to SLL, several prior-art methods in the literature, including fault-analysis (FLL), LUT-based locking, etc. [11–15], tried to maximize the complexity of obfuscation using different KGs replacement strategies.

### 2.1.3 Random-based Hill-Climbing Attack

Plaza *et al.* [32] developed a new algorithmic attack that uses test patterns and observe responses. Unlike sensitization attack [12], their proposed approach does not require netlist access. They propose a randomized local key-searching algorithm to search the key that can satisfy a subset of correct input/output patterns. The algorithm proposed in [32] is iterative in nature. At first, it selects random value for key bits and then at each iteration, the key bits, which are selected randomly, are toggled one by one. The target is to minimize the frequency of differences between the observed and expected responses. Hence, a random key candidate is gradually improved based on observed test responses. If no solution is found in one iteration, the algorithm resets the key to a new random key value. However, the complexity of this attack quickly increases with increasing number of KGs.

## 2.2 Stage 2: SAT Attack

In 2015, Subramanyan *et al.* [10] proposed a new and powerful attack using Boolean satisfiability (SAT) solver, called SAT attack, that effectively and quickly broke all previously proposed logic locking techniques. As an "*oracle-guided*" attack, SAT attack requires a reverse-engineered but locked netlist ($C_L$), and a functionally activated chip ($C_O$). A circuit view of steps taken in a SAT attack is shown in Fig. 2.2. For this attack, the attacker first replicate the obfuscated circuit and builds a double circuit which is used for finding an input (X$_d$[i]) that for two different key values generates two different outputs. Such input is referred to as Discriminating Input Pattern(DIP). Each X$_d$[i] is used to create a DI validation circuit (DIVC). The validation circuit, as shown in Fig. 2.2 assures that for a previously found X$_d$[i], two different keys generate the same output value. Each iteration of the SAT attack finds a new (X$_d$[i]), and add a new DI validation circuit. The DIVCs are ANDed together to form a Set of Correct Key Validation Circuit (SCKVC). In each iteration, the SAT solver try to find a new X$_d$[i] and two key values that satisfy the double circuit (KDC) and the Validation Circuit (SCKVC). The key values and the X$_d$[i], as illustrated in Alg. 1, is found by a SAT query. This means the new key generate two different values for the new X$_d$[i], but generate the same value for all previously found X$_d$s for both key values. This process continues until the SAT solver cannot find a new X$_d$[i]. At this point any key that generates the correct output for the set of found X$_d$s is the correct key.

For all prior obfuscation schemes, even those resistant to sensitization attack, the SAT attack was able to rule out a significant number of key values at each iterations (by finding each DIP). Hence, In order to thwart SAT attack, the first attempted approach was to weaken the strength of the DIPs to reduce its pruning power. SARLock [18] and Anti-SAT

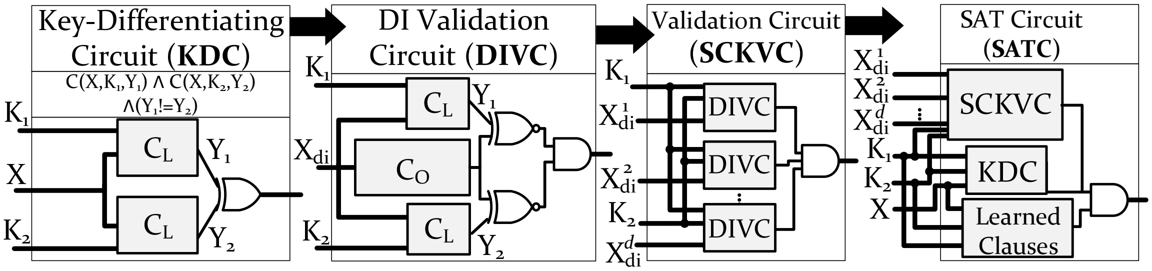


Figure 2.2: SAT Attack Iterative Flow.

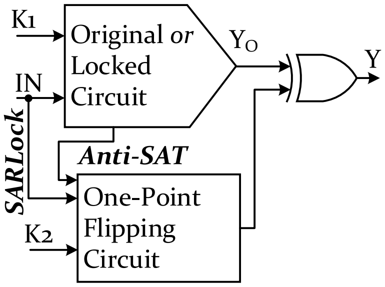
**Algorithm 1** SAT-based Attack Algorithm [10]

```
function SAT_Attack(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2)$;
    while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$; $i \leftarrow i+1$;
    $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

[19] were the first prior-art methods that accomplished this. Both SARLock and Anti-SAT engaged one-point flipping function, demonstrated in Fig. 2.3. Using this obfuscation scheme, each DIP is able to rule out only one incorrect key. Hence, the SAT attack requires to apply all $2^{|KI|}$ to retrieve the correct functionality. However, this method results in obfuscation circuits that for all but one output work as the original circuit, and the output corruption upon application of a wrong key is quite low.

## 2.3 Stage 3: Post-SAT Attacks

As discussed, the proposed SAT-resilient solutions suffered from low output corruption. This however could have been addressed by combining a SAT-hard solution with a traditional obfuscation solution, such as RLL or SLL, that exhibits high level of output corruption. Although SAT-resilient logic locking schemes provided a defense against SAT attack, researchers found new vulnerabilities associated with this class of obfuscation techniques resulting in the development of many new attacks on the presumed SAT-hard logic locking schemes described in this section.



| $Y_O$ | IN | k=0 | k=1 | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | 0 | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| ✓ | 1 | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | 2 | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| ✓ | 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| ✓ | 6 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| ✓ | 7 | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |

Figure 2.3: Flipping Structure of SARLock and Anti-SAT.

### 2.3.1 Removal Attack

As shown in Fig 2.3, in bare implementation of one-point flipping circuit, the locking circuitry is completely decoupled from the original circuit. A removal attack identifies and removes/bypasses the locking circuitry to retrieve the original circuit and to remove dependence on key values [33]. The removal attack, presented in [33], was used to detect and remove SARLock [18]. In presence of removal attack, researchers investigated SAT-hard solutions that are hard to detect (preventing removal by pure structural analysis), an example of which was Anti-SAT [19].

### 2.3.2 Signal Probability Skew (SPS) Attack

The Signal Probability Skew (SPS) attack [20] leverages the structural traces in Anti-SAT block to identify and isolate the Anti-SAT block [19]. *Signal probability skew* (SPS) of a signal $x$ is defined as $s = P_r[x = 1] - 0.5$, where $P_r[x = 1]$ indicates the probability that signal $x$ is 1. The range of $s$ is $[-0.5, 0.5]$. If the SPS of signal $x$ is closer to zero, an attacker have lower chance of guessing the signal value by random. For a 2-input gate, the signal probability skew is the difference between the signal probability of its input wires. The flipping-circuit in the Anti-SAT is constructed using two complementary circuits, $g$ and $\overline{g}$, in which the number of input vectors that make the function $g$ equal to 1 ($p$) is either close to 1 or $2^n - 1$. These two complementary circuits converge at an AND gate $G$. Considering this structure, the *absolute difference of the signal probability skew* (ADS) of the inputs of gate $G$ is quite large, noting that the SAT resilience is ensured by this skewed $p$. Algorithm

2 shows the SPS attack, which identifies the Anti-SAT block's output by computing signal probabilities and searching for the skew(s) of arriving signals to a gate in a given netlist.

**Algorithm 2** SPS Attack Algorithm [20]

```
function SPS_Attack(Circuit $C_L$)
    $ADS_{arr} \leftarrow \{\}$;
    for each $gate \in C_L$ do
        $ADS_{arr}(\text{gate}_i) \leftarrow$ Compute_ADS($C_L$, $\text{gate}_i$);
    $G \leftarrow$ Find_Maximum($ADS_{arr}$);
    $Y \leftarrow$ Find_value_from_skew($G$);            ▷ Correct value of Anti_SAT output
    $C_{Lock} \leftarrow$ remove_TFI($C_L$, $G$, $Y$);        ▷ Transitive FanIn of the gate $G$
    return $C_{Lock}$                        ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

### 2.3.3 Bypass Attack

Although SARLock and Anti-SAT break the SAT attack, their respective output corruptibility is very low if they are not mixed with traditional logic locking, such as SLL. Observing and relying on the very low level of output corruption in such SAT-hard solutions, the bypass attack [24] was introduced. The bypass attack instantiates two copies of the obfuscated netlist using two randomly selected keys, and build a miter circuit that evaluates to 1 only when the output of two circuits is different. The miter circuit is then fed to a SAT solver looking for such inputs. The SAT returns with minimum of two inputs for which the outputs are different. These input patterns are tested using an activated IC (golden IC) validating the correct output. Then a bypass circuit is constructed using a comparator that is stitched to the primary output of the netlist which is unlocked using the selected random key, to retrieve the correct functionality if that input pattern is applied. The Bypass attack works well when the SAT-hard solution is not mixed with traditional logic locking mechanism since its overhead increases very quickly as output corruption of logic locking increases. This observation motivated researchers to look at possibilities of approximate attacks to retrieve the key values associated to non SAT-hard obfuscation solutions that are mixed with SAT-hard solutions.

### 2.3.4 AppSAT Attack

So far, defences solution to mitigate the SAT attack, are based on the assumption that the attacker needs an exact attack on logic locking. However, Shamsi *et al.* [21] proposed a new attack (AppSAT), which relax this constraint. AppSAT shown in Algorithm 3, is an approximate attack on logic locking based on the SAT attack and random testing. The authors use *probably-approximate-correct* (PAC) model for formulating approximate learning problem. The exact SAT attack continues to find DIPs until no more DIPs can be found. However, the AppSAT will be terminated in any early step in which the error falls below the certain limit. If this condition happens, the key value will be considered as an approximate key with specified error rate; otherwise, the random sampling that resulted in a disagreement will be added to a SAT formula as a new constraint. In AppSAT, heuristic methods for estimating the error is used for large functions, to avoid any computation complexity.

### 2.3.5 Double-DIP Attack

Double-DIP [22] is another approximate attack, shown in Algorithm 4. Double-DIP is an extension of SAT attack in which during each iteration, the discriminating input should eliminate at least two wrong keys. To illustrate its effectiveness, researchers used double-DIP to target SARLock+SSL, representing a compound of SAT-hard and high output corruption obfuscation. When the double-DIP attack terminates, the key of the traditional logic locking (SSL) is guaranteed to be correct. As a result, the compound logic locking will be reduced to a single SAT attack resilient technique, that could then be attacked using bypass attack.

### 2.3.6 Bit-Flipping Attack

The Bit-flipping attack [35] is yet another attack against compound logic locking schemes in which a SAT-hard solution such as SARLock is combined with a traditional logic locking to guarantee both high error rates and resilience to the SAT-based attack. In Bit-flipping

**Algorithm 3** AppSAT Attack Algorithm [21]

```
function AppSAT_Attack(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2)$;
    while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$; $i \leftarrow i+1$;
        every $n$ rounds do
        for each ($x \in$ Random Patterns) do
            if $C_L(X, K_1, Y) \neq C_O(X)$ then
                $FailedPatterns \leftarrow FailedPatterns + 1$;
                $F_{i+1} \leftarrow F_{i+1} \wedge (C_L(X, K_1, Y) = C_O(X))$; $i \leftarrow i+1$;
        if error ¡ ErrorThreshold then
            return $K_1$ as an approximate key
    $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

**Algorithm 4** Double-DIP Attack Algorithm [22]

```
function DoubleDIP_Attack(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_1, Y_1) \wedge C_L(X, K_2, Y_2) \wedge C_L(X, K_3, Y_1) \wedge C_L(X, K_4, Y_2)$ ;
    while $SAT(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2)) \wedge (K_1 \neq K_3)) \wedge (K_2 \neq K_4))$;
        $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \bigwedge_{j=1}^{4} C_L(X_d[i], K_j, Y_d[i])$; $i \leftarrow i+1$;
    $K^* \leftarrow$ sat_assignment$_{K_1}(F_i)$;
```

attack, the keys are first separated into two groups ($k_1$ and $k_2$) by counting DIPs for two keys with hamming distance equal to one. The attack is motivated from the observation that in traditional logic locking, wrong key causes substantial wrong input-output pattern. However, the error rate of bit-flipping function is usually very small. As shown in Algorithm 5, after separation of keys, this attack fixes SAT-resilient keys, $k_2$, as a random number, and uses a SAT solver to find the correct key values for $k_1$. After finding $k_1$, the bypass attack is applied to retrieve the original circuit.

### 2.3.7 AppSAT Guided Removal Attack

AppSAT Guided Removal (AGR) attack targets compound logic locking, particularly Anti-SAT + traditional logic locking [33]. This attack integrates AppSAT with a simple structural analysis of the locked netlist (a post-processing steps). Unlike AppSAT, the AGR attack recovers the correct key. In this attack, first the AppSAT is used to find the key of the

**Algorithm 5** Bit-flipping Attack Algorithm [35]

```
function BitFlipping_Attack(Circuit $C_L$, Circuit $C_O$)
    for each $j$ ¡ Fixed-iteration do
        $K_A \leftarrow$ a random key;
        for each bit b $\in K_A$ do
            $K_B \leftarrow K_A$ while bit b flipped;
            $i \leftarrow 0$; $F_0 \leftarrow C_L(X, K_A, Y_A) \wedge C_L(X, K_B, Y_B)$;
            while $SAT(F_i \wedge (Y_A \neq Y_B))$ do
                $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_A \neq Y_B))$;
                $F_{i+1} \leftarrow F_i \wedge (X \neq X_d[i])$; $i \leftarrow i+1$;
                if i ¿ Threshold then
                    b is in $K_1$,
                    break;
        $j \leftarrow j + 1$;
    $K_2 \leftarrow$ all key bits / $K_1$;        ▷ Seperation is Done. Then, fix $K_2$ as a random number.
    $K_1 \leftarrow$ SAT_ATTACK $(C_L, C_O)$;                         ▷ Find Traditional Keys using SAT.
    $C_L^* \leftarrow$ update_netlist($C_L$ — $K_1$)
    return (BYPASS_ATTACK($C_L^*$);
```

traditional obfuscation scheme (used as a part of compound lock). Then, AGR targets the remaining key bits belong to the SAT-resilient logic locking, such as Anti-SAT block, through a simple structural analysis. As shown in Algorithm 6, in the post-processing steps, AGR finds the gate ($G$) at which most of the Anti-SAT key bits converge. AGR finds $G$ by tracing the transitive fanout of the Anti-SAT key inputs, where all the Anti-SAT key bits converge. The ratio of key bits converging at each of the inputs of the gate $G$, are close to 0.5, which is shown as the *selected property* in line 7 of Algorithm 6. AGR identifies the candidates for gate $G$ by checking this property for all gates in the circuit, and then sort these candidate based on the number of key inputs that converge at a gate and pick the gate $G$ from all candidates, which has the most number of key inputs converge to that gate. Then the attacker re-synthesize the design with the constant value for the output of $G$ gate and retrieving the correct functionality.

### 2.3.8 Sensitization Guided SAT Attack

While the one-point flipping circuit in Anti-SAT and SARLock are completely decoupled from the original netlist, Li *et al.* [36] proposed the AND-tree Insertion (ATI), as a SAT-resilient logic locking, which embeds AND trees inside the original netlist. It not only

**Algorithm 6** AGR Attack Algorithm [33]

```
function AGR_ATTACK(Circuit $C_L$, Circuit $C_O$)
    $\#Cand \leftarrow$ num_gates($C_L$)
    while ($\#Cand$ ¿ 1 and !$Timeout$) do
        AppSAT_Attack();                                            ▷ 4 times
        $Candidates \leftarrow \{\}$;
        for each $gate \in C_L$ do
            if $gate_i$ has the selected property then
                $Candidates \leftarrow Candidates + 1$;
    $G \leftarrow$ Find_Max_key_count($Candidates$);
    $C_{Lock} \leftarrow$ remove_TFI($C_L$, $G$);           ▷ remove Transitive FanIn of the gate $G$
    return $C_{Lock}$;                              ▷ $C_{Lock}$: $C_L$ after removing Anti_SAT block
```

makes all aforementioned attack less effective, it also decreases the implementation overhead. Additionally, the input of AND-tree are camouflaged by inserting INV/BUF camouflaged gates, which can be replaced with the XOR/XNOR gates in order to lock the AND-tree. However, this defense was broken by a new attack that was coined as Sensitization Guided SAT (SGS) attack [33]. The SGS attack is carried out in two stages: (1) *sensitization* that exploits bias in input patterns which allows an attacker to apply only a subset of DIPs, i.e., those that bring unique values to the AND-tree inputs. (2) *SAT attack* using the patterns discovered in the first stage.

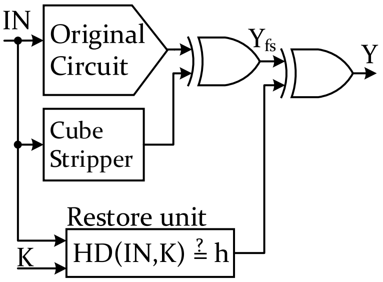
### 2.3.9 Functional Analysis Attack

Aiming to provide a defense that resists all previously formulated attacks led to the introduction of Stripped-Functionality Logic Locking (SFLL) [37]. In SFLL the original circuit is modified for at-least one input pattern (cube) using a *cube stripping unit*, demonstrated in Fig. 2.4. As shown, $Y_{fs}$ is the output of the stripped circuit, in which the output corresponding to at-least one input pattern is flipped. The restore unit not only generates the flip signal for one input pattern per each wrong key, it also restores the stripped output, (e.g. $IN = 4$ in Fig. 2.4) to recover the correct functionality on $Y$. Note that applying removal attack on restore unit recovers $Y_{fs}$, which is not the correct functionality. In addition, SFLL-HD is able to protect $\binom{k}{h}$ input patterns that are of Hamming Distance (HD) $h$ from the $k$-bit secret key, and accordingly uses Hamming Distance checker as a restore unit (e.g. $h = 0$ in Fig. 2.4 is also called TTLock [38]).

Although SFLL was resilient against all previously formulated attacks, it was exploited using a newly formulated attack, called Functional Analysis on Logic Locking (FALL) attack [39]. In this attack model, the adversary is assumed to be a malicious foundry that knows the locking algorithm and its parameters, e.g. $h$ in SFLL-HD. A FALL attack is carried out in three main stages and relies on structural and functional analyses to determine potential key values of a locked circuit. First, FALL attack tries to find all nodes which are the results of comparing an input value with a key input. It is done by a comparator identification. Such nodes ($nodes_{RU}$), which contains these particular comparators, are very likely to be part of the functionality restoration unit. The set of all inputs that appear in these comparators, should be in the fan-in cone of the cube stripping unit. Then, it finds a set of all gates whose fan-in-cone is identical to the members of $nodes_{RU}$. This set of gates must contain the output of the cube stripping unit. Second, the attacker applies functional analysis on the candidate nodes suggested by and collected from the first stage to identify suspected key values. Broadly speaking, the attacker uses functional properties of the cube stripping function used in SFLL, to determines the values of the keys. Based on the author's view, this function has three specific properties. So, they have proposed three attacks algorithms on SFLL, which exploit unateness and Hamming distance properties of the cube stripping functions. The input of these algorithm is circuit node $c$, that computed from the first stage, and the algorithm checks if $c$ behaves as a Hamming distance calculator in the cube stripping unit of SFLL-HD. If the attack is successful, the return value is the protected cube. Third, they have proposed a SAT-based key confirmation algorithm using a list of suspected key values and I/O oracle access, that verifies whether one of the suspected key values computed from the second stage, is correct.

### 2.3.10 CycSAT Attack

Considering the strength of all previously formulated attacks, some of the researchers started seeking solutions that fundamentally violated the assumptions of these attacks with respect



| $Y_{fs}$ | IN | k=0 | k=1 | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | 0 | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| ✓ | 1 | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | 2 | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| ✗ | 4 | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| ✓ | 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| ✓ | 6 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| ✓ | 7 | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |

Figure 2.4: SFLL-HD while $h = 0$.

to the nature of locked circuits. One of such attempts was the introduction of cyclic logic locking [40][26], was first proposed in [40]. In this obfuscation technique each deliberately established cycle is designed to have more than one way to open. The requirement for having more than one way to open each cycle assures that even if the original netlist has no cycle by itself, the cycles remains irreducible by means of structural analysis. The cyclic obfuscation resulted in an obfuscation with high level of output corruption, while it was able to break the SAT attack either by 1) trapping it in an infinite loop, or 2) forcing it to exit with a wrong key depending on weather the introduced cycles make the circuit stateful or oscillating.

The promise of secure cyclic obfuscation was shortly after broken by CycSAT attack [41]. In CycSAT, the key combinations that result in formation of cycles are found in a pre-processing step. These conditions are then translated into problem augmenting CNF formulas, denoted as cycle avoidance clauses, satisfaction of which guarantee no cycle in the netlist. The cycle avoidance clauses are then added to the original SAT circuit CNF and the SAT attack is executed. The validity of this attack, however, was challenged in [26], as researchers illustrated that the pre-processing time for CycSAT attack is linearly dependent on the number of cycles in the netlist. Hence, by building an exponential relation between the number of feedback, and the number of cycles in the design, the pre-processing step of CycSAT will face exponential runtime.

### 2.3.11 Behavioral SAT (BeSAT) Attack

Inability to analyze all cycles in the prepossessing step of CycSAT results in missing cycles in the pre-processing step of CycSAT, leading to building a statefull or oscillating circuit, trapping the SAT stage of the CycSAT attack. BeSAT [42] remedies this shortcoming by augmenting the CycSAT attack with a run-time behavioral analysis. As shown in Algorithm 7, by performing behavioral analysis at each SAT iteration, BeSAT detects repeated DIPs when the SAT is trapped in an infinite loop. Also, when SAT cannot find any new DIP, a ternary-based SAT is used to verify the returned key as a correct one, preventing the SAT from exiting with an invalid key.

**Algorithm 7** BeSAT Attack on Cyclic Locked Circuits [42]

```
function BeSAT_Attack(Circuit $C_L$, Circuit $C_O$)
    $W = (w_0, w_1, ...w_m) \leftarrow$ FindFeedback($C_L$);
    for each ($w_i \in W$) do
        $F(w_i, w_i') \leftarrow$ no_ structural_ path($w_i$);
    $i \leftarrow 0$; $NC(K) = \wedge_{i=0}^{m} F(w_i, w_i')$
    $C_L^*(X, K, Y) \leftarrow C_L(X, K, Y) \wedge NC(K)$; $F_0 \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
    while $SAT(F_i \wedge (Y_1 \neq Y_2))$ do
        $X_d[i] \leftarrow$ sat_assignment $(F_i \wedge (Y_1 \neq Y_2))$; $Y_d[i] \leftarrow C_O(X_d[i])$;
        $F_{i+1} \leftarrow F_i \wedge C_L(X_d[i], K_1, Y_d[i]) \wedge C_L(X_d[i], K_2, Y_d[i])$;
        if ($X_d[i]$ in DIP) and ($C_L(X_d[i], K_1) \neq Y_d[i]$)) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_1) \wedge (K_2 \neq \hat{K}_1)$;
        else if ($X_d[i]$ in DIP) and ($C_L(X_d[i], K_2) \neq Y_d[i]$) then
            $F_{i+1} \leftarrow F_{i+1} \wedge (K_1 \neq \hat{K}_2) \wedge (K_2 \neq \hat{K}_2)$;
        $i \leftarrow i+1$;
    while $SAT_{K_1}(F_i)$ do                                   ▷ Correct Key: $\hat{K}_c$
        if Ternary_SAT($F_i$, $K_c$) then
            $F_i \leftarrow F_i \wedge (K_1 \neq \hat{K}_c)$
        else
            $K^* \leftarrow \hat{K}_c$; break;
```

## 2.4 Stage 4: SMT Attack

As discussed previously, many of the attacks proposed at post-SAT attack stage were formulated by adding a pre-processing step to the original SAT attack, and/or extending the SAT attack to co-process and check additional features in each iteration. In other terms, to break many of the post-SAT era obfuscation techniques, attackers relied on compound attacks by combining SAT solvers by pre-processors (e.g. in CycSAT) and co-processors (e.g. in BeSAT) to extend its modeling reach. Motivated by this trend, the need for having pre- co- and post- processors along with a SAT solver in an attack was realized and addressed in [43] and a new and extremely powerful attack, coined as Satisfiability Module Theory (SMT) attack was introduced. The strength of SMT attack, as the superset of SAT attack, comes from its ability to combine SAT and Theory solvers. The SMT attack could be invoked with any number and combination of theory solvers, and a SAT solver, which allow the attacker to express constraints that are difficult or even impossible to express using CNF, including timing, delay, power, arithmetic, graph and many other first-order theories in general. To showcase the modeling capability of SMT attack, the authors used the SMT attack 1) to break a new breed of obfuscation that relied on locking the delay information in netlist (by generating setup and hold violations), 2) to formulate an accelerated attack (to reduce the attack time) with means of approximate exit (if trapped with SAT hard solutions).

**Algorithm 8** SMT Attack on DLL (Lazy Approach) [43]

```
function SMTLazy_Attack(Circuit $C_L$, Circuit $C_O$)
    $C^*_L \leftarrow$ toBOOLEAN($C_L$);                          ▷ Replace TDK with Buffer
    $i \leftarrow 0$; $F \leftarrow C^*_L(X, K_1, Y_1) \wedge C^*_L(X, K_2, Y_2)$;
    $G^*_L \leftarrow$ toGRAPH($C_L$);                  ▷ Wires = Edges, Gates = Vertices
    $F_T \leftarrow$ GenTCE($G^*_L$)                             ▷ Theory Learned Clauses
    $F_{SMT} \leftarrow F \wedge F_T$;                                      ▷ SMT Clauses
    while $SMT(F_{SMT})$ do                                ▷ $X_d$[i], $K_1$, $K_2$, CC
        $Y_d$[i] $\leftarrow C_O(X_d$[i]); $F \leftarrow F \wedge C^*_L(X_d$[i], $K_1$, $Y_d$[i]) $\wedge C^*_L(X_d$[i], $K_2$, $Y_d$[i]);
        $F_{SMT} \leftarrow F \wedge$ CC; $i \leftarrow i+1$;
    $K^* \leftarrow$ smt_assignment$_{K_1}(F_{SMT})$;

    function GenTCE(Graph $G^*_L$)
        Inputs $\leftarrow$ find_start_points($G^*_L$); Outputs $\leftarrow$ find_end_points($G^*_L$); $T_{CE}(K) \leftarrow$ [];
        for each ((Sp, Ep) $\in$ (Inputs, outputs)) do
            Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));     ▷ Hold Violation
            Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);          ▷ Setup Violation
            Range(Sp,Ep)(K) $\leftarrow$ Lower(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K);
            $T_{CE}(K) \leftarrow T_{CE}(K) \cup$ Range(Sp,Ep)(K);
        return $T_{CE}(K)$
```

In pursuit of obfuscation schemes that could not be attacked by SAT motivated attackers, some researchers tried to extend the locking mechanism to aspects of a circuit's function that cannot be translated to CNF. For example, Xie *et al.* proposed a timing obfuscation scheme, denoted as delay logic locking (DLL), in [27]. The Goal of DLL obfuscation scheme is introducing setup and hold violation if the correct key is not applied. In this case, the obfuscation attempts to change both logical and behavioral (timing) properties. A functionally-correct but timing-incorrect key will result in timing violations, leading to circuit malfunctions. Considering that timing is not translatable to CNF, the SAT solver remains oblivious to the keys used for timing obfuscation. Authors in [43], however, illustrated that the SMT attack could easily deploy a graph theory solver, provide timing constraints to the theory solver (in terms of required min and max delay to meet the hold and setup time), and use the theory solver in parallel with the internal SAT solver to break both logic and delay obfuscation. They additionally show that the theory solver could be initiated as a pre-processor (Eager SMT approach) or as a co-processor (Lazy SMT approach) to break the same problem, showcasing the strength of SMT attack. The *lazy* mode of this attack is illustrated in Algorithm 8. Although at about the same time Chakraborty proposed TimingSAT to attack the DLL [44], similar to many prior SAT-based attack, it was by deploying a pre-processor for analysis of graph timing, and generating helper clauses for the subsequent call to the SAT attack.

The ability of SMT solver to instantiate and integrate different theory solver makes it a suitable attack platform for modeling and formulating very strong attacks. As an example of the strength of SMT attack, the authors in [43] formulated and presented an accelerated SMT attack with ability of detecting the presence of SAT-hard obfuscation and switching to an accelerated approximate attack. As shown in Algorithm 9, this was done by invoking a *BitVector* theory solver to constrain the SMT solver for finding keys that result in highest output corruption first. This could be done by constraining the required HD between the output of double circuit when two different keys for the same discriminating input is being tested. The required HD starts from a large value, and every time that the

SMT solver return UNSAT, the constraint is relaxed until HD of 1 is reached. This leads to the guaranteed discovery of keys for traditional logic locking first. After $N$ tries (*Rep* in Algorithm 9) for HD of 1, the SMT attack exits, notes that there exist a SAT-hard obfuscation, which now could be addressed by the Bypass attack. More details on SMT attack will be discussed in section 3.

**Algorithm 9** Accelerated SMT Attack on Compound Locking [43]

```
function AccSMT_Attack(Circuit $C_L$, Circuit $C_O$)
    $i \leftarrow 0$; $HD_h \leftarrow sizeof(\text{output})$; $HD_l \leftarrow HD_h - 1$;
    $TimeOut \leftarrow 20$; $Rep \leftarrow 20$; $HD_R \leftarrow 1$; $R_{cnt} \leftarrow 0$;
    $C_L^* \leftarrow \text{toBOOLEAN}(C_L)$;                          ▷ Everything is Boolean.
    $F \leftarrow C_L^*(X, K_1, Y_1) \wedge C_L^*(X, K_2, Y_2)$;
    $BV_L^* \leftarrow \text{toBITVECTOR}(C_L)$;                  ▷ Define BITVECTOR on output.
    $BVs_L^*(X, K_1, K_2) \leftarrow \text{ONEs}(BV_L^*(X, K_1) \oplus BV_L^*(X, K_2))$;
    $F_T \leftarrow (BVs_L^*(X, K_1, K_2) \geqslant HD_l) \wedge (BVs_L^*(X, K_1, K_2) \leqslant HD_h)$;
    $F_{SMT} \leftarrow F \wedge F_T$;                                       ▷ SMT Clauses
    while $HD_l \geqslant 1$ do
        while $SMT(F_{SMT} — TimeOut)$ do                         ▷ $X_d[i], K_1, K_2$, CC
            $Y_d[i] \leftarrow C_O(X_d[i])$;
            $F \leftarrow F \wedge C_L^*(X_d[i], K_1, Y_d[i]) \wedge C_L^*(X_d[i], K_2, Y_d[i])$; $F_{SMT} \leftarrow F \wedge CC$;
            if $HD_l \leqslant HD_R$ then
                if $R_{cnt} == Rep$ then
                    break;
                $R_{cnt}$++;
        $HD_l$--;
    $K^* \leftarrow \text{smt\_assignment}_{K_1}(F_{SMT})$;
```

## 2.5 Discussion & Opportunities

Table 2.2 compares the effectiveness of the attacks discussed in this section against most notable obfuscation schemes. As illustrated the combination of FALL, Bypass and SMT attack can break all existing solutions, pointing us to a need for a new direction for generating non-bypassable SMT hard obfuscation solutions.

The dilemma is that SAT-hard solutions have extremely low output corruption, and are prone to Bypass, FALL, Removal and SPS attack. On the other hand, the traditional logic locking schemes have high output corruption, but could be easily broken with SAT/SMT attack. The compound logic locking solutions that combine the SAT-hard solutions for

Table 2.2: Comparison of proposed attacks/defenses.

| Defenses<br>Attacks | **RLL**<br>[7] | **FLL**<br>[13] | **SLL**<br>[12] | **Anti-SAT**<br>[19] | **SARLock**<br>[18] | **Compound**<br>[18] | **SFLL**<br>[37] | **Cyclic**<br>[40] | **SRCLock**<br>[26] | **DLL**<br>[27] |
|---|---|---|---|---|---|---|---|---|---|---|
| **Brute Force** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Sensitization[12]** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Hill-Climbing[32]** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SAT[10]** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SPS+Removal[33][20]** | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Bypass[24]** | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **AppSAT[21]** | ✓ | ✓ | ✓ | ✗ | ✗ | **P** | ✗ | ✗ | ✗ | ✗ |
| **Double-DIP[22]** | ✓ | ✓ | ✓ | ✗ | ✗ | **P** | ✗ | ✗ | ✗ | ✗ |
| **Bit-Flipping[35]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| **AGR[33]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| **FALL[39]** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| **CycSAT[41]** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| **BeSAT[42]** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| **TimingSAT[44]** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| **SMT[43]** | ✓ | ✓ | ✓ | ✗ | ✗ | **P** | ✗ | ✓ | ✓ | ✓ |

✓: Attack Success, ✗: Fail to Attack, **P**: Only removes the key to the traditional locking in Compound Defense.

resistance against SAT and SMT attack, and traditional logic locking for resistance against Bypass, FALL, Removal and SPS attack are also prone to approximate SAT and SMT attacks. What is really desired, is a SMT-hard logic locking scheme with high degree of output corruption. As a step in this direction, few very recent research papers have focused on increasing the execution time of each SAT/SMT iteration rather than the total execution time [45, 46]. The Full-Lock in [45] is argued that the strength of SAT/SMT solvers come from their *Conflict-Driven Clause Learning* (CDCL) ability, which is resulted by recursively calling *Davis-Putnam-Logemann-Loveland* (DPLL) algorithm. Hence, the Full-Lock creates an obfuscation method that results in very deep recursive call trees. They argue that the SAT/SMT attack execution time can be expresses by formula 2.1, in which $N$ denotes the number of iterations (DIPs) of the SAT/SMT attack, $T_{DPLL}(\Phi)$ is the execution time of recursive calls for DPLL algorithm on CNF $\Phi$, and $t$ is the execution time of the remaining book keeping code executed at each iteration.

$$T_{Attack} = \sum_{i=1}^{N} T(i) = \sum_{i=1}^{N} (t + T_{DPLL}(\Phi)) = \sum_{i=1}^{N} \sum_{j=1}^{M} (T_{DPLL}^{Avg}) \simeq MN \times T_{DPLL}^{Avg} \tag{2.1}$$

Authors argue that $M$ in formula 2.1 denotes the number of recursive DPLL calls. Accordingly, the execution time of SAT attack could also become unfeasible by building an exponential relation between the percentage gate inserted (area overhead) and M. The strong aspect of this alternative solution is that (1) the problems posed at each iteration of SAT/SMT attack is a SAT-hard problem, (2) the output corruption of this methods is significantly higher than obfuscating solution relying on increasing the $N$, (3) it is not susceptible to SPS, removal, bypass, approximate attack, to name a few. The hardness of SAT/SMT attack in the solution posed by Full-Lock cannot be assessed/formulated similar to that of SFLL. Moving towards this new direction for generating SAT-hard problems with high level of output corruption can be generalized more, where an obfuscation solution in this direction can engineer the number of recursive calls, pushing the number of recursive call to be an exponential function of added gates counts (area overhead).

# Chapter 3: SMT Attack: Next Generation Attack on Obfuscated Circuits

In this section, we present Satisfiability Modulo Theory (SMT)-based attack on obfuscated circuits, that expands the capabilities of previously proposed SAT attack by assigning theory solvers to monitor the behavioral and non-functional properties of the obfuscated circuit. To illustrate the capabilities of SMT attack, we use an SMT solver and invoke a graph-theory solver to break the logic and timing obfuscation scheme introduced in [27].

## 3.1 Boolean Logic Obfuscation

Logic locking and netlist obfuscation schemes introduce limited programmability into a netlist by means of inserting additional key programmable gates at design time. After fabrication, the functionality of the IC is programmed by loading the correct key-values. The key-inputs could be stored in and driven by an on-chip tamper-proof memory [28]. The purpose of inserting key-gates is protecting the IC design from untrusted foundries. Since the functionality of a design is locked with a secret key, the attacker cannot learn the functionality of the obfuscated netlist after reverse engineering. Logic locking and obfuscation schemes vary in terms of the usage of different key-gates types and key-gates insertion policies [47,48]. For combinational circuits, logic locking can be classified based on key-gates types to different categories. XOR/XNOR based logic locking [7,12], MUX based logic locking, and LUT based logic locking [14,15] are the most common mechanisms. Also, there are different algorithms for inserting the key-gates in the circuit. Some of these policies include random insertion (RLL), fault-analysis (FLL) insertion, and interference-based logic locking (SLL) algorithms, SARLock, Anti-SAT , etc. [7,12,13,18,19].

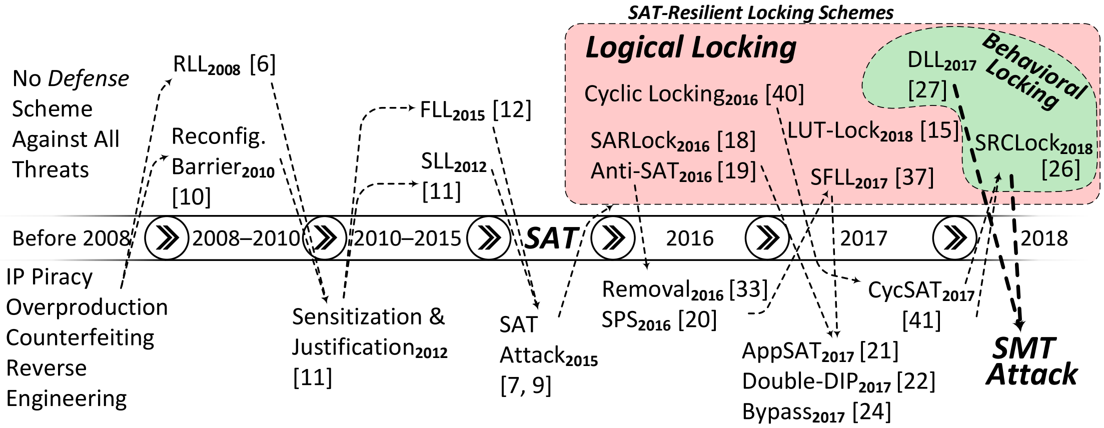


Figure 3.1: 10-Year History of Logic Obfuscation.

Fig. 3.1 captures the evolving history of obfuscation defense schemes and attack formulations since the year 2008 to the current date. After introduction of SAT attack, in 2015 in [9, 10], as illustrated in this figure, researchers proposed various mechanisms for building SAT hard obfuscation solutions. However, many of such obfuscation schemes were later broken using newer attacks like as SPS, removal, bypass, and AppSAT [20–22, 24, 33], making the current defense schemes unreliable. After 2017, a new breed of obfuscation schemes instead of building logical obfuscation schemes has been introduced, relied on breaking the SAT assumptions for building SAT hard solutions without having the vulnerabilities of the previous SAT-Hard solution. For example, Cyclic obfuscation [40] and its improved defense, the SRCLock[26], by introducing cycles into netlist break the SAT model as the netlist can no longer be represented by a Directed Acyclic Graph (DAG). Alternatively, the Delay Logic Locking (DLL) [27] extends the reach of obfuscation beyond logic and locks the circuit using its delay and timing properties, attempting to build SAT hard solutions. In this work, we introduce the SMT attack that could break such locking and obfuscation mechanisms by means of parallel invocation of SAT and theory solvers to model the non-logical and behavioral aspects of a circuit operation.

## 3.2 Behavioral logical obfuscation

As previously discussed, the logic-based obfuscation schemes that rely on extending the Boolean behavior of a circuit can be broken by at least one of the state-of-the-art attacks, including SAT, SPS, removal, bypass, and AppSAT [9, 10, 20–22, 24, 33]. Hence, recent researches have been focused on obfuscation schemes that fundamentally violate the assumptions of these attacks with respect to the nature of obfuscated circuit, or use non-logical properties of a netlist to obfuscate its behavior [26, 27, 40].

For lack of EDA tool support and limited knowledge in designing cyclic Boolean logic, most of all netlists designed and fabricated today are acyclic. One of the first attempts to break the state of the art attacks, including SAT attack, was proposed in [40] which suggested using *cycles* in combinational circuits, and illustrated that use of cycles results in either a SAT solver being trapped, or it generates incorrect key even after timely termination. This obfuscation scheme, however was shortly after broken by CycSAT attack in [41]. In the CycSAT attack, the netlist is first pre-processed based upon which a set of constraining clauses are generated. The CycSAT attack then uses these constraining clauses, in the original SAT attack, allowing the SAT solver to effectively open the cycles without being trapped, or incorrectly terminated. However, the limitation of [40] was addressed in SRCLock [26] to prevent a pre-processor from extracting all needed constraints from a cyclically locked circuit. SRCLock focuses on building an exponential relation between the number of inserted feedbacks and number of generated cycles by means of creating *super cycles*.

The second obfuscation of interest to this work is the logic and timing obfuscation scheme in [27]. In this obfuscation scheme, the delay properties of a circuit are obfuscated with the ultimate goal of introducing setup and hold violation if the correct key is not applied. In this case, the obfuscation, in addition to the logical behavior of the netlist, attempts to change its behavioral (timing) properties. Considering that timing is not translatable to CNF, the SAT solver remains oblivious to the keys used for timing obfuscation. Hence using a SAT attack to deobfuscated this circuit, result in a discovery of all keys used for logic obfuscation, but random assignment to all keys used for timing obfuscation and the circuit remains locked.

In this work, we construct an attack based on Satisfiability Modulo Theory solvers, and illustrate that the capability of this attack goes far beyond that of SAT attacks. More precisely, with specific formulation, we illustrate that SMT attack on obfuscated circuits could be significantly faster and more efficient compared to SAT attacks on Boolean logic obfuscation. Additionally, it could be used to attack behavioral logic obfuscation schemes, which is not possible by a pure SAT-based attack. To illustrate the second point, we attack and break the timing-logic obfuscation scheme in [27], based on which we generalize and illustrate how other similar SMT attacks could be formulated.

## 3.3 Attack Model

The SMT attack is an ***oracle-guided attack***. We assume that the attacker has the reverse engineered but obfuscated netlist and a functional IC (*oracle*) that is unlocked. The attacker can query the oracle with any stimuli $i$, and observe its output $o$. The purpose of the attack is to find the key inputs, that make the obfuscated netlist logically equivalent to that of the unlocked netlist.

As it can be seen in Fig. 3.2, IP owner obfuscates the Original Netlist of IP. Assuming that design integration, verification, fabrication, and packaging have been accomplished in untrusted regime, attacker is able to obtain the obfuscated (locked) netlist from (1) the IC design, or by reverse engineering the (2) synthesis/implementation (layout), (3) mask, or (4) a manufactured IC. In addition, the attacker is able to buy the correctly unlocked (activated) IC in the open market. Consequently, the attacker can apply arbitrary input to activated IC and observe its corresponding output.

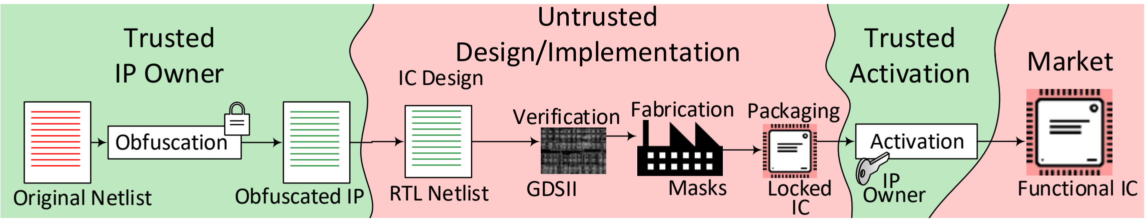
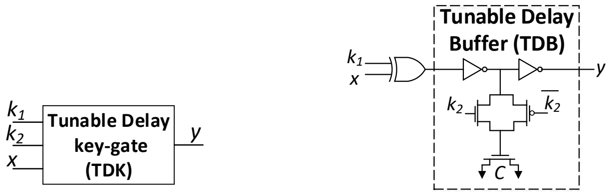


Figure 3.2: ASIC Design Flow Integrated with Obfuscation/Activation.

## 3.4 Limitation of SAT Attack

A SAT attack works perfectly fine if the logic obfuscation is of Boolean nature. This is because any Boolean logic could be easily transformed into its Conjunctive Normal Form (CNF) and be converted into a satisfiability assignment problem. But in case of Behavioral logic obfuscation, the locking mechanism is designed to control aspects of circuit operations that could not be translated to CNF as required by a SAT solver. The delay-locking (DLL) scheme proposed in [27], cyclic-based obfuscation presented in [40], and SRCLock [26] are good instances of such locking mechanism. For the purpose of locking, DLL uses a tunable delay key-gate (TDK) which is illustrated in Fig. 3.3. TDK consists of a conventional key-gate (XOR/XNOR) with a tunable delay buffer (TDB). The capacitive load of the buffer is controlled by a transmission gate, where activating the transmission gate increases the wire load capacitance of the internal wire, resulting in larger TDK propagation delay. Hence, the functionality and propagation delay of a TDK, both, depends on the value of its key-inputs.

In DLL, the TDK cells are used to control the *setup* and *hold* time violations such that only one sequence of activation keys guarantees that circuit operates with no violation. To apply the DLL, a design is first altered such that most timing paths are balanced to be sensitive with respect to small changes in the path delay, such that a small variation in delay causes setup or hold violations. This is achieved by means of carefully engineering the clock skew, cell sizing, and $V_{th}$ swapping. Then the TDK cells are inserted in the common portions of setup and hold critical paths, such that attempting to only fix setup causes hold violation, and attempting to fix hold causes setup violation with the exception of one sequence of correctly configured TDK keys that assures all timing paths meet both setup



**(a)** **(b)**

| $k_1k_2$ | f(x) | Delay |
|---|---|---|
| 00 | $y = x$ | $d_0$ |
| 01 | $y = x$ | $d_1$ |
| 10 | $y = \overline{\mathrm{x}}$ | $d_0$ |
| 11 | $y = \overline{\mathrm{x}}$ | $d_1$ |

**(c)**

Figure 3.3: Overall Structure of Tunable Delay Key-Gate (TDK).

and hold check timing constraints. Considering that the delay is not a logical behavior, the TDK cell behavior could not be completely captured by CNF, hence the delay locking is not directly attackable by a SAT attack. In [27] it was illustrated that even a mixed integer linear programming (MILP) based attack has up to 39% timing violation ratio (TVR). However, as we will illustrate in this section, by employing an SMT attack and by instantiating an integrated graph theory solver along with its SAT engine, we could find the keys to this obfuscation problem in few minutes.

## 3.5 SMT Solver

In this section, we first review the usage and capabilities of an SMT solver, and then we illustrate how the SMT solver could be used to form an SMT attack on obfuscated circuits regardless of obfuscation's reliance on logical or non-logical properties of a circuit.

### 3.5.1 SMT Usage and Capabilities

A Satisfiability Modulo Theory (SMT) is used to solve a decision problem while honoring constraints that could be expressed using first-order theories such as equality, reasoning, arithmetic, graph-based deduction, etc. Hence, it could be considered as a solver for a broad set of problems that could be categorized as Constraint Satisfaction Problems (CSP), which is a superset of Boolean Satisfiability Problems (BSP) that are solvable by SAT solvers. Additionally, the ability to express theories such as inequality (e.g. $3x + y < z$) provides a much richer Application Programming Interface (API) to the end user to define a problem compared to that of a SAT solver.

In general, there are two different approaches for solving an SMT problem. The first approach is based on translating the problem into a Boolean SAT instances denoted by *Eager* approach; In this approach the existing Boolean SAT solvers are used as is. However, the SMT solver has to work a lot harder for solving some problems that are otherwise very obvious (e.g. for checking the equivalence of two 32-bit values). However, by deploying a theory solver, this could be achieved in no time. For this reason, many SMT solvers follow another approach which referred to as the *Lazy* Approach. The Lazy approach integrates the Boolean satisfiability solvers, which are based on the Davis-Putnam-Logemann-Loveland (DPLL) in modern SAT, and theory solvers that decide the satisfiability of formulas over specific theories. Each theory solver provides two capabilities: (1) theory propagation among various theory solvers for checking possible conflicts on partial assignments, and (2) clause learning result of which is shared by the SAT solver to speed-up pruning the decision tree. Additionally, since several applications of SMT deal with formulas involving two or more theories at ones, modern SMT solvers provide the capability of combining theory solvers using Nelson-Oppen [49] or Shostak [50] method to support a more expressive language. In combining theory solvers, if two theories $\Gamma_1$ and $\Gamma_2$ are both defined axiomatically, their combination can simply be defined as the theory axiomatized by union of the axioms of the two theories, $\Gamma_1$ and $\Gamma_2$. For example, Consider $\Gamma_1$ and $\Gamma_2$ are two different theories, it is possible to define $\Gamma_1 \oplus \Gamma_2$ as a combined theory of $\Gamma_1$ and $\Gamma_2$, where $\Gamma_1 \oplus \Gamma_2$ is the set of all models that satisfy $\Gamma_1 \cup \Gamma_2$. This is adequate if the signatures of the two theories are disjoint. Otherwise, if $\Gamma_1$ and $\Gamma_2$ have symbols in common, one has to consider whether a shared function symbol is meant to stand for the same function in each theory or not. In the latter case a proper signature renaming must be applied to the theories before taking the union of their axioms. in [51] they have described general conditions for the combination of theories that may have symbols in common. The ability to combine theory solvers proves extremely useful when dealing with applications such as model checking and predicate abstraction-based model check in which we need to check the satisfiability of formulas over several data types.

Theories are defined as classes of models with the same signature. More precisely, a $\Sigma$-theory $\Gamma$ is a pair of ($\Sigma$, $A$) where $\Sigma$ is a signature and $A$ is a class of $\Sigma$-models. In general a theory solver for a theory $\Gamma$ is a procedure which takes as input a collection of $\Gamma$-literals $\mu$ and decides whether $\mu$ is $\Gamma$-satisfiable. A theory ($\Gamma$-solver) to be effectively used within an SMT solver should have the following properties [52]: *(1) Model Generation*: theory solver should be able to produce a $\Gamma$-model of the problem description $\mu$. *(2) Conflict Set Generation*: when the theory solver reaches inconsistency, it should be able to produce a subset $\eta$ of $\mu$ which has caused the inconsistency. The subset $\eta$ is referred to as *theory conflict*. *(3) Incrementality*: The $\Gamma$-solver should be able to save and keep its status across invocation calls to avoid recomputation. *(4) Backtrackability*: it is important for theory solver to has the ability to undo the step if it is needed. Equality with Uninterpreted Functions (EUF), linear real arithmetic (LRA), linear integer arithmetic (LIA), Mixed Integer and Real Arithmetic, Difference Logic, Bit Vectors, Arrays, etc. are the examples of theories commonly used in SMT.

In this work, we use an SMT solver and formulate some attacks against specific obfuscated circuits, illustrating the power of adapting various theory solvers for extending the capabilities of attack by constraining and monitoring non-logical properties of a netlist. For this purpose, and to illustrate that SMT attack is a super-set to the SAT attack, we first illustrate that the original SAT attack against obfuscated circuits could be effectively formulated using an SMT solver, resulting in similar performance. Then we illustrate how the SMT solver could be used to attack logic obfuscation problems out of the reach of pure SAT attacks, and for that purpose we break the logic and timing obfuscation in [27] which is not possible by a pure SAT attack. We illustrate that this attack could be achieved using both Eager and Lazy approach of SMT attack. Then we illustrate how the SMT attack could become significantly more efficient than a SAT attack by adopting the capabilities of theory solvers like *BitVector*, and formulate an accelerated SMT attack, that requires substantially smaller iterations and runtime compared to a SAT attack against specific obfuscation schemes. In addition, we formulate the accelerated SMT attack to be capable of approximate attacks.

## 3.6 SMT Attack

When building an SMT attack on obfuscated circuits, as illustrated in Fig. 3.4, the SMT attack could be invoked with any number and combination of theory solvers, and a SAT solver. In order to use the SMT solver to formulate an attack, few preliminary steps should be taken. The first step is to make a minor modification to an extracted netlist after reverse engineering, providing the capability of testing various behaviors of the obfuscated circuit to the SAT or SMT solver. The transformation is simply replacing the obfuscated cells with their equivalent Key Programmable Gates (KPG). A KPG performs the same function as the obfuscated cell, however, it allows building a key controlled representation of the logical behavior of the obfuscated cell for the purpose of logical-model building. Fig. 3.5 captures the KPG translation gates for each type of the gates that have previously used in recent literature for the purpose of obfuscation. For example, when attacking a camouflaged cell that could be either an $AND$ gate or an $XOR$ gate, it is replaced with its KPG which is simply a $MUX$ with each of its input tied to one of the camouflaged cell possibilities. The function performing the KPG replacement in the algorithms described in this work is ***ReplaceKPG(N***$_{obf}$***)*** that replaces all obfuscated cells in an obfuscated module with their KPGs equivalent based on translation table in Fig. 3.5.

When using an SMT solver, before invoking a theory solver, the input model or input behavior should be translated to a model $\mu$ which is understood by that theory solver. As illustrated in Fig. 3.4, the translation step may be different for each theory solver used. As an example, to break the Delay Logic Locking in [27], we use a graph theory solver and translate the obfuscated netlist to a graph model that is understood by the graph-theory solver. The required translation step ($\mu \leftarrow$ Netlist) is simply the inversion of the netlist under attack to its graph representation, where each gate is a *node* in the graph, and each

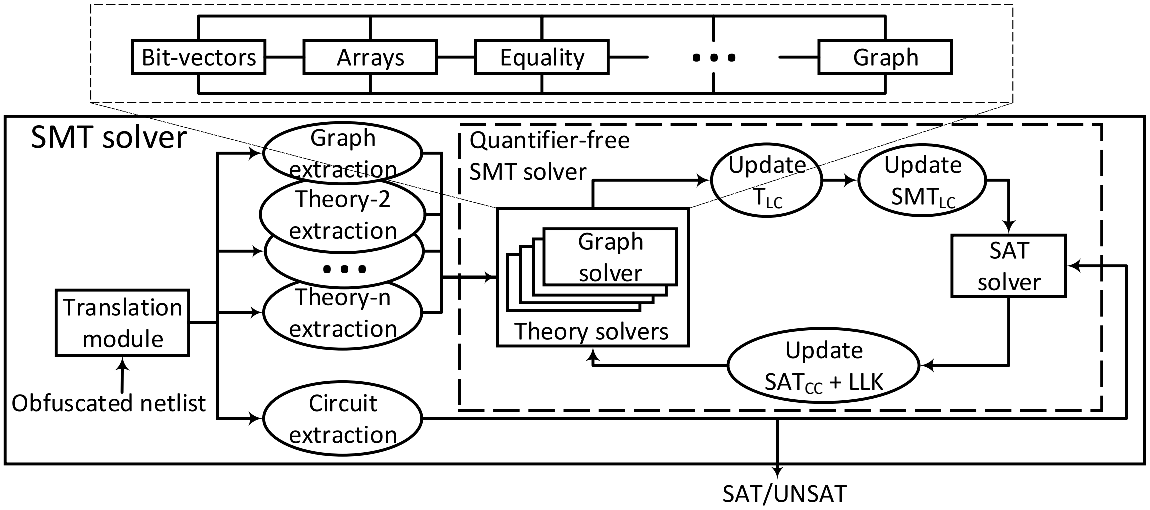


Figure 3.4: Overall Architecture of SMT Attack for Circuit Deobfuscation.

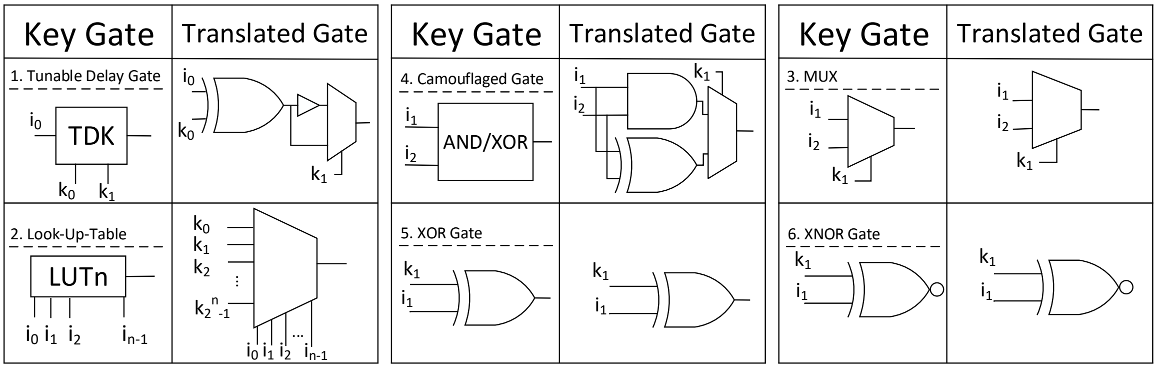


Figure 3.5: Translation Table to Key Programmable Gates (KPG).

net an *edge*. We have additionally included the functionality to compute the logical effort in our graph translation routine, that annotate each edge with the logical effort needed to drive that edge as a measure of its delay. We could alternatively use a second theory solver to capture the static timing of the netlist and exchange information with the graph theory solver for more accurate results. The final step before invoking the SMT/SAT attack is the translation of the netlist under attack into its CNF form as described in [10].

After building model $\mu$ for each Theory and SAT solver, the SMT attack is formulated based on the flow of information exchange between theory and SAT solver. In General, the formulation of the SAT portion of SMT solver is similar to that of pure SAT attack as described in [10]. However, in addition to the SAT solver, each theory solvers is then tuned by declaration of theory constraint. At this stage, invoking the SMT solver returns a satisfiable assignment and a list of learned theory and conflict clauses for theory solver and SAT solver respectively. The SMT attack is then achieved by composing the correct control flow for invocation of theory and SAT solver(s), and by managing the intermediate sequence of CNF-based information exchange. The general flow of information in an SMT formulated problem, including that of SMT attack, is illustrated in Fig. 3.4.

### 3.6.1 Attack Mode 1: SMT reduced to SAT Attack

As was mentioned previously, the SAT attack finds a functionally correct key for an obfuscated circuit by checking a small subset of all input patterns, hence removing the need for brute-force testing of all input patterns. Considering that SMT solver is a superset of SAT solver and contains a SAT solver, any attack formulated for SAT could be similarly formulated for an SMT solver.

Alg. 10 illustrates the SAT attack that could be similarly implemented in a SMT solver. The formulation of attack remains similar to that of original attack proposed in [9, 10].

**Algorithm 10** SMT Reduced to SAT Attack in [9, 10]

```
function SAT_Attack(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
    $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
    $C(X, K, Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
    $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
    $SCKVC = TRUE$;
    $SATC = KDC \wedge SCKVC$;
    $LC = TRUE$;                                         ▷ Learned Clauses
    $SMT_{LC} \leftarrow SATC$;
    while $(((X_{DI}, K_1, K_2, CC) \leftarrow SMT.Solve(SMT_{LC})) = TRUE)$ do
        $Y_f \leftarrow C_{org}(X_{DI})$;
        $DIVC = C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f)$;
        $SCKVC = SCKVC \wedge DIVC$;
        $LC = LC \wedge CC$
        $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
    $Key \leftarrow SMT.Solve(SMT_{LC})$;
```

The SAT attack in Alg. 10 follows the steps illustrated in Fig. 3.6. In this algorithm, the obfuscated gates are first replaced with key programmable gates (KPG) to create the Key Programmable Circuit (KPC). Then the CNF representation of the circuit is generated.

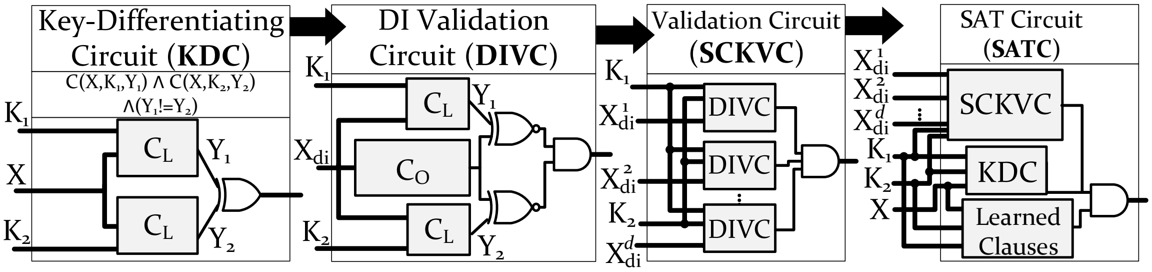
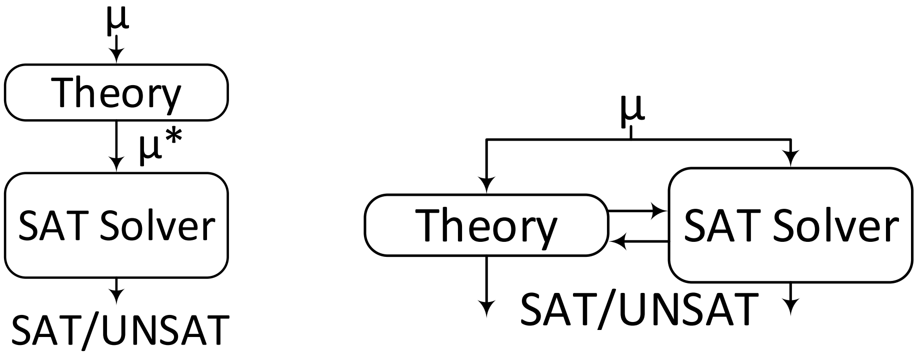

Key-Differentiating Circuit (KDC)
C(X,K1,Y1) ∧ C(X,K2,Y2) ∧(Y1!=Y2)
K1
X
K2
CL
CL
Y1
Y2
DI Validation Circuit (DIVC)
K1
Xdi
K2
CL
CO
CL
Y1
Y2
Validation Circuit (SCKVC)
K1
X1di
X2di
K2
Xddi
DIVC
DIVC
DIVC
SAT Circuit (SATC)
X1di
X2di
Xddi
K1
K2
X
SCKVC
KDC
Learned Clauses

Figure 3.6: From Obfuscated Circuit to SAT Circuit.

Two KPCs are then used to generate a Key Differentiating Circuit (KDC). The KDC receives an input and two different keys and determines whether they generate the same output or not. The KDC is then used as the first SMT satisfiability problem represented by $SMT_{LC}$ for the first invocation of SMT solver. Calling the SMT *solve* function on the posed formula then return an assignment for keys $K_1$, $K_2$, and the discriminating input $X_{DI}$ such that the formulated $SMT_{LC}$ is satisfied. In addition, the SMT solver returns a list of learned Conflict Clauses (CC). In line 10, the correct output ($Y_f$) for the discriminating input $X_{DI}$ is found. In the next step, the SMT formula needs to be updated to use the discriminating input and learned clauses to further constrain the satisfiability problem. This is done in multiple steps. In line 11, the discriminating input found in the current iteration is used to create a Discriminating Input Validation Circuit (DIVC) which is illustrated in Fig. 3.6(d). The DIVC circuits formed at each iteration are ANDed together to create a circuit that checks the correctness of a key for all previously found discriminating inputs. This circuit is referred to as Set of Correct Key Validation Circuit (SKCVC). In line 13, the currently found Conflict Clauses are added to the set of previously found Learned Clauses (LC). Note that this step is done implicitly for SMT is a stateful solver. Finally, in line 14 the SMT satisfiability problem is constrained by ANDing together the KDC, SCKVC and LC clauses. The SAT attack formulated using SMT solver continues until the SMT solver returns UNSAT. A final call to the SMT solver returns the correct key. Note that this SMT attack is a one-to-one translation of the original SAT attack in [9, 10]. In section 3.7.1, we illustrate that the formulation of SAT attack using SMT solver results in very similar performance to that of pure SAT attack. However, the SMT attack could further benefit



(a) Eager SMT

(b) Lazy SMT

Figure 3.7: SMT Execution Flow.

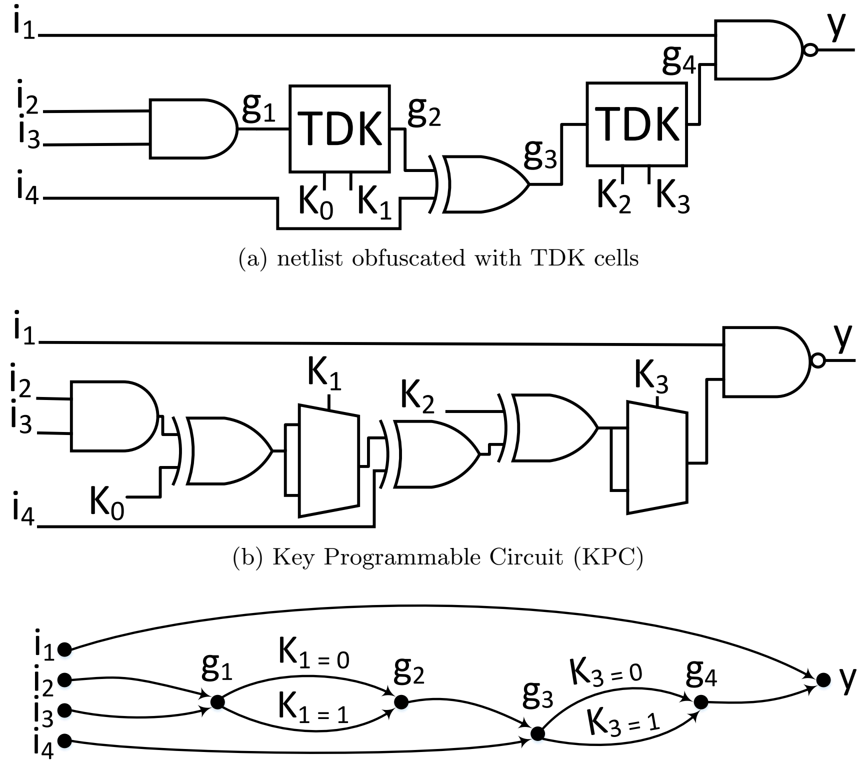
from the usage of SMT solvers to extend its capabilities to attack obfuscation schemes that could not be logically modeled.

### 3.6.2 Attack Mode 2: Eager SMT Attack

Theory solvers could be used to extend the capabilities and performance of SMT solver compared to that of a SAT solver. This, as illustrated in Fig. 3.7 could be done by (1) using the theory solver to extract all required clauses that complete the CNF description with respect to the obfuscation scheme and then to perform a SAT attack, referred to as the SMT *Eager* approach. This could be thought (2) by invoking the theory and SAT solver in parallel to simultaneously model and solve the problem, referred to as *Lazy* approach.

In this section, we illustrate how the Eager approach of SMT attack could be used to attack the obfuscation schemes that could not be broken or understood by a pure SAT attack. For this purpose, we formulated an SMT attack on the delay-locking (DLL) scheme proposed in [27]. Notice that the proposed approach could be used in formulating attacks on other obfuscation techniques that rely on non-logical properties of circuit obfuscation such as timing, power, delay, etc. by using the appropriate theory solvers.

Fig. 3.8 illustrates the translation steps for converting a DLL[27] obfuscated circuit (using translation table in Fig. 3.5) to its key programmable circuit and captures its graph representation. As illustrated in Fig. 3.8(b), $K_1$ effectively has no impact on the logical



(a) netlist obfuscated with TDK cells

(b) Key Programmable Circuit (KPC)

(c) representative graph of the netlist

Figure 3.8: Conversion Flow in SMT using Graph Theory Solver.

behavior of the circuit and only changes its delay properties. Hence, subjecting this obfuscated circuit to a SAT attack results in a random assignment to $K_1$. Therefore, by having $k$ TDK cells, which have $2k$ keys in total, a SAT solver returns one logically correct key sequence among $2^k$ different set of such logically correct keys that control the TDK cells, however, only one of such keys doesn't result in setup and hold violations. Hence, a correct attack should consider the delay and timing properties of the netlist in addition to its logical correctness.

The shortcoming of SAT attack to capture the delay and timing properties of the netlist, when attacking DLL obfuscation, is remedied in an SMT attack by means of using a graph theory solver. To illustrate this, we formulate an Eager and a Lazy SMT attack on DLL obfuscation. In the Eager approach, we use the theory solver as a mean of pre-processing the netlist by which we deduct the complete set of Valid-Path Constraint Clauses (VPCC) between all primary inputs and outputs of the obfuscated netlist. This VPCC is a CNF

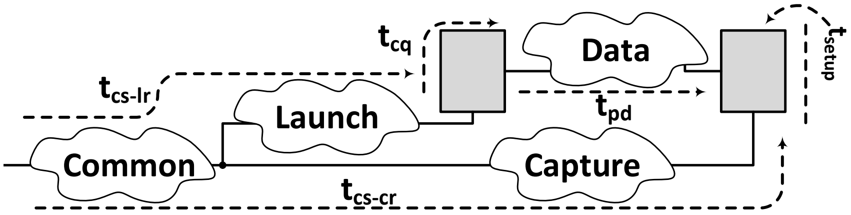


Figure 3.9: Various Delay Components of a Timing Path.

presentation of all valid assignment of the keys, such that no setup or hold violation is created. Note that among many such possibilities, only one possibility has both the correct timing and the correct logical behavior.

To build the VPCC clauses, we should compute the setup and hold constraints on every timing path. The setup and hold timing checks for a timing path is expressed using the following inequalities:

$$t_{cs-lr} + t_{clk-q} + t_p + t_{setup} + U \leq t_{cs-cr} + T_{clk} \tag{3.1}$$

$$t_{cs-lr} + t_{clk-q} + t_{cd} \geq t_{hold} + t_{cs-cr} + U \tag{3.2}$$

In this equation which uses the notation in Fig. 3.9, the $t_{cs-lr}$ is the clock source to launch register delay, $t_{cs-cr}$ is the clock source to capture register delay, $U$ is the clock jitter/uncertainty, $t_{clk-q}$ is the clock to q delay of the launch register, $t_{setup}$ is the capture-register setup time, $t_{hold}$ is the hold time requirement for the capture register, $t_p$ is the propagation delay through the longest path in the timing path, and finally the $t_{cd}$ is the propagation delay through the shortest pah in the logic. Considering that the DLL logic is only inserted on *Data* sections a of timing path (according to the tormentingly in Fig. 3.9), it can only affect the $t_p$ and $t_{cd}$. Note that it is also possible to enhance the DLL obfuscation beyond that described in [27] and use the TDK cells for building clock skew in the clock network, however, a similar attack still could be formulated. For now, let's consider that DLL, as described in [27], only affects the Data section of timing path. The equations 3.1 and 3.2 could be re-written as:

$$\mathbf{t}_p \leq T_{clk} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} - t_{setup} - U = Upper \tag{3.3}$$

$$\mathbf{t}_{cd} \geq t_{hold} + (t_{cs-cr} - t_{cs-lr}) - t_{clk-q} + U = Lower \tag{3.4}$$

Before performing any reverse engineering, we know the $T_{CLK}$ from the functional chip purchased on market. Note that a functional chip (the oracle) is needed to perform the SAT or SMT attack as explained in section 3.3. Now let's consider a netlist obtained after reverse engineering. The end-point and start-point registers for each timing path are known. Hence, by means of spice simulation, the register could be characterized and the $t_{clk-q}$, $t_{setup}$ and $t_{hold}$ are extracted. Note that there are limited type of registers used in a physical design, and at this step only a handful of registers need to be characterized. Extracting a measure for uncertainty could be also achieved by means of spice simulation.

At this point, considering that a TDK cell can change the delay of a timing path, the delay of each timing path ($D_j$) could be divided into a constant delay ($C_j$) and a variable delay ($V_j(K)$), where the variable delay is a function of the number of TDK cells in that timing path, and the key assumed for each TDK. Hence, Delay of Timing path $j$ from start point $s$ to endpoint $p$ ($D_j^{s \to p}$) that passes through $N$ TDK cells each with delay $D_{TDK}^{s \to p}(i)$, depending on the value of key $K_i$ is obtained from:

$$D_j^{s \to p} = C_j^{s \to p} + V_j^{s \to p}(k) \tag{3.5}$$

$$D_j^{s \to p} = C_j^{s \to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s \to p}(i) \tag{3.6}$$

For a given timing path, and by using the equation 3.6, we could rewrite the delay constraints in equations 3.3 and 3.4 as:

$$\forall j| \quad D_{j_{max}}^{s \to p} = C_{j_{max}}^{s \to p} + \sum_{i=1}^{N} K_i \times D_{TDK}^{s \to p}(i) \leq Upper \tag{3.7}$$

$$\forall j| \quad D^{s \to p}_{j_{min}} = C^{s \to p}_{j_{min}} + \sum_{i=1}^{N} K_i \times D^{s \to p}_{TDK}(i) \geq Lower \tag{3.8}$$

These inequalities capture the lower and upper bound delay constrain for every pair of input-output pins in a design, and collectively capture the model $\mu$ of the graph theory solver. Based on this formulation, the number of added inequalities is M × N, in which $M$ is the number of primary inputs, and $N$ is the number of primary outputs. However, one inequality bounds all timing paths between the selected input-output pin pair, removing the need to express the inequality for every timing path in the design as needed in MILP-based attack that was suggested in [27].

After writing these inequalities for each input-output pair, a call to the SMT *solve* function returns all key combinations for which all paths constraints/inequalities are satisfied. In the other word, by assuming any of the returned key combinations, the circuit will not violate its setup and hold timing checks. However, note that only one (or few) of these key values is logically correct. The correct key value then could be extracted by invoking a SAT solver, and by providing the set of key combinations (in CNF format) as a constraint to the logical circuit satisfiability problem. This process is illustrated in Alg. 11. As it can be seen in Alg. 11, function *GenTLC* is responsible for generating all inequalities. Line 7-8 of *GenTLC* function generates inequality (7) and (8) for each input (Sp) to each output (Ep).

This algorithm is similar to Alg. 10, with the additional step of using a theory solver for pre-processing the netlist in line 8, extraction of all key combination resulting in correct timing behavior in line 9, and providing these constraints to the SAT solver in the next step in line 10. Note that the *solve* function in the Eager approach is called in two places; first for generating the timing valid key combination clauses (inside GenTLC function), and then iteratively inside the SAT attack while loop.

For some obfuscation methods, the pre-processing step of Eager approach may become extremely time consuming or computationally impossible. An example of such obfuscation problem is the SRCLock [26]. The authors have shown that the obfuscation is SAT hard, since without pre-processing the cycles, the SAT solver will be trapped or produce an

**Algorithm 11** Eager SMT Attack on DLL [27]

```
function SMT_EAGER_ATT(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
    $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
    $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
    $KDC$ = C(X, $K_1$, $Y_1$) $\wedge$ C(X, $K_2$, $Y_2$) $\wedge$ ($Y_1 \neq Y_2$);
    $SCKVC$ = $TRUE$;
    $SATC$ = $KDC \wedge SCKVC$;
    $LC$ = $TRUE$;                                          ▷ Learned Clauses
    $G(X,K) \leftarrow$ Graph_Translation($N_{obf}$);
    $T_{LC} \leftarrow GenTLC(G(X,K))$;                      ▷ Theory Learned Clauses
    $SMT_{LC} \leftarrow SATC \wedge T_{LC}$;                 ▷ SMT Clauses
    while ((($X_{DI}$,$K_1$,$K_2$,CC) $\leftarrow$ $SMT.Solve(SMT_{LC})$)= $TRUE$) do
        $Y_f \leftarrow C_{org}(X_{DI})$;
        $DIVC$ = C($X_{DI}$,$K_1$,$Y_f$) $\wedge$ C($X_{DI}$,$K_2$,$Y_f$);
        $SCKVC$ = $SCKVC \wedge DIVC$;
        $LC$ = $LC \wedge CC$
        $SMT_{LC}$ = $KDC \wedge SCKVC \wedge LC$;
    $Key \leftarrow SMT.Solve(SMT_{LC})$;
```

Pre-Processing step by using a graph theory solver for SMT attack (*Eager* )

```
function GENTLC(Graph $G$)
    $Inputs \leftarrow G$.find_start_points();
    $Outputs \leftarrow G$.find_end_points();
    $T_{LC} \leftarrow []$
    for each ($Sp$ in $Inputs$) do
        for each ($Ep$ in $Outputs$) do
            Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));
            Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);
            $T_{LC} \leftarrow$ SMT.solve(Upper(Sp,Ep)(K) $\wedge$ Lower(Sp,Ep)(K) $\wedge$ $T_{LC}$);
    return $T_{LC}$
```

incorrect key. Additionally, they have suggested two mechanisms by which the number of cycles in a netlist could exponentially grow with respect to the number of inserted feedbacks. For attacking cyclic logic, as suggested by CycSAT attack [41] we need to pre-process the netlist and extract the No Cycle Conditions to prevent the SAT solver from being trapped. However, in SRCLock[26] the number of cycles grow exponentially, and therefore the runtime of pre-processing step also grows exponentially, preventing us to ever reach the SAT attack. For such problems, the Eager approach that relies on reduction of the problem to a SAT problem does not work. However, the Lazy approach of the SMT attack provides a solution.

### 3.6.3 Attack Mode 3: Lazy SMT Attack

Using the Lazy approach of SMT attack relaxes the requirement of Eager approach to complete the pre-processing step before invoking the SAT attack.

In the Lazy approach the SAT solver and theory solver(s) simultaneously check different models of a unified satisfiability problem, exchange clauses, and check each other's literal assignment. This could significantly prune the decision tree of a SAT solver search space for finding a satisfying assignment and remove the need for a complete and unbounded execution of theory solver as it only has to check the validity of constraints for SAT assigned literals.

In order to illustrate the Lazy approach of SMT attack, in this section, we formulate an SMT attack to again break the DLL [27] obfuscation. The Lazy approach of SMT attack on DLL [27] is illustrated in Alg. 12. The big difference in the Lazy and Eager approach is that after model generation for theory solver, the SMT *solve* function is not called. This is illustrated in line 9 of this algorithm, where the constraining expressions are only defined for the theory solver by making a call to $GenTCE$ function. The returned constraining expressions are then duplicated for Key $K_1$ and $K_2$. The SMT *solve* function is then called to find an assignment for a discriminating input $X_{DI}$, and two different keys $K1$ and $K2$ such that generated outputs are different at least in one bit, however both keys generate a valid timing scenario. Since the SAT model (SATC) and Theory models ($T_{CE}(K1, K2)$) share literals and are subjected to a unified set of constraints, the decision tree and search

**Algorithm 12** Overall SMT Attack (*Lazy* Approach)

```
function SMT_LAZY_ATT(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
    $KPC \leftarrow$ ReplaceKPG($N_{obf}$);
    $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
    $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
    $SCKVC = TRUE$;
    $SATC = KDC \wedge SCKVC$;
    $LC = TRUE$;                                        ▷ Learned Clauses
    $G(X,K) \leftarrow$ Graph_Translation($N_{obf}$);
    $T_{CE}(K) \leftarrow GenTCE(G(X, K))$;             ▷ Theory Constraint Expressions (Not Solved)
    $T_{CE}(K1, K2) \leftarrow T_{CE}(K1) \cup T_{CE}(K2)$ ;
    while ((($X_{DI}$,$K_1$,$K_2$,CC) $\leftarrow$ $SMT.Solve(SATC, T_{CE}(K1, K2))) = TRUE$) do
        $Y_f \leftarrow C_{org}(X_{DI})$;
        $DIVC = C(X_{DI}, K_1, Y_f) \wedge C(X_{DI}, K_2, Y_f)$;
        $SCKVC = SCKVC \wedge DIVC$;
        $LC = LC \wedge CC$
        $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
    $Key \leftarrow SMT.Solve(SMT_{LC}, T_{CE}(K))$;
```

Initialization of constraints for SMT attack (*Lazy* Approach)

```
    function GENTCE(Graph $G$)
        $Inputs \leftarrow G$.find_start_points();
        $Outputs \leftarrow G$.find_end_points();
        $T_{CE}(K) \leftarrow []$
        for each ($Sp$ in $Inputs$) do
            for each ($Ep$ in $Outputs$) do
                Upper(Sp,Ep)(K) $\leftarrow$ !(distance_leq(Sp, Ep, $t_{cd}$));
                Lower(Sp,Ep)(K) $\leftarrow$ distance_leq(Sp, Ep, $t_p$);
                $Range(Sp,Ep)(K) \leftarrow$ Lower(Sp,Ep)(K) $\wedge$ Upper(Sp,Ep)(K);
                $T_{CE}(K) \leftarrow T_{CE}(K) \cup Range(Sp,Ep)(K)$;
        return $T_{CE}(K)$
```

space for the SMT solvers is significantly reduced.

### 3.6.4 Attack Mode 4: Accelerated Lazy SMT Attack (AccSMT)

In this section, we argue that re-formulating the Lazy SMT which benefits from capabilities of *BitVector* theory solver allows us to build a more efficient attack.

Our modification to the SAT attack is inspired by the observation that higher output corruption, reduces the SAT hardness of an obfuscation scheme. A discriminating input $X_{DI}$, is an input capable of sensitizing the logic paths of the netlist under study, such that (1) some of the differences in the values of internal nodes in the result of application of two different keys $K_1$ and $K_2$ are propagated to at least one output. (2) none of the previously found DIPs (that were used in building a DIVC) were able to propagate the generated inconsistency to a primary output. This mechanism is continued until the number of sensitized paths, reaches a point where any inconsistency that from application of two different keys is propagated to the primary outputs using the constructed set of DIVC circuits. At this point, the set of previously found $X_{DI}$s form a complete set of discriminating inputs, such that if a key generates the correct output for all inputs in this set, it will generate the correct output for all other inputs.

Different DIPs have different pruning power. A DIPs strength could be assessed based on the number of inconsistencies that it could sensitize to the primary outputs conditioned that previous DIPs were incapable of doing so. Hence, depending on the pruning power of DIPs, the size of the complete set of DIPs could be different. A minimal complete set of DIPs is the smallest set of DIPs that could de-obfuscate the circuit. In our Lazy approach for SMT attack, we propose a mechanism to reduce the size of the complete set of DIPs pushing it towards the minimal set. Since in each SAT or SMT iteration one DIP is found, having a smaller number of DIPs result in smaller number of iterations.

In SAT attack, it requires only a single bit difference in the output for generation of a DIP. In SMT attack, we could make a stronger requirement for the generation of DIPs. This could be achieved by forcing the SMT solver to find DIPs with the largest possible

Hamming distance of primary outputs of the KPC circuits when for the same input, two different keys are applied. Such a DIP has a much higher pruning capability, and is able to sensitize a larger number of key-related inconsistencies to the output. The discovery of such powerful DIPs reduces the number of required DIPs that is needed to form a complete set of DIPs that could de-obfuscate the circuit. This is because when the hamming-distance is larger either the KPC circuits differ in (1) key-bit(s) that are located close to the inputs, or (2) large number of assumed key-bits (in the middle of timing paths or close to primary outputs) are different in two KPC circuits, or (3) the combination of two scenarios. In both cases, the added DIP and the resulting learned clauses eliminate the cause of obtaining such large hamming distance, resulting in the elimination of a large number of inputs as possible future DIPs while eliminating a larger set of keys as potential correct keys. Hence, when such a DIP is added to a DIVC, it poses a much stricter restriction on the requirements for finding the next DIP and reducing the attack time by almost an order of magnitude.

**Using BitVector Theory Solver:**

Assessing DIPs based on hamming distance of the primary output is easily implementable in SMT solver by using a *BitVector* theory solver. The bitVector theory solver allows us to perform integer-oriented arithmetic operations such as addition, subtraction, and multiplication. The Hamming Distance (HD) of output $Y_1$ and $Y_2$ is obtained using:

$$HD(C(X_{DI}, K_1), C(X_{DI}, K2)) = HD(Y_1, Y_2) = \sum_{i=1}^{N} Y_1(i) \oplus Y_2(i) \tag{3.9}$$

The HD is then used to write the constraining expressions that are posed on the BitVector theory solver using the formulation:

$$Th_{Lower} \leq HD(Y_1, Y_2) \leq Th_{Upper} = Size(Output) \tag{3.10}$$

The upper threshold $Th_{Upper}$ is kept constant equal to the size of output pins, but the lower threshold $Th_{Lower}$ is defined as a variable, allowing us to sweep the hamming distance constraint posed on BitVector theory solver from a maximum value of the number of output bits to a minimum value of 1. The lower bound could be reduced every time the SMT solver returns UNSAT, indicating there is no other DIP that satisfies the HD requirement poset on theory solver. The process terminates when the SMT cannot even find a DIP that causes HD of 1. Adaption of this constraint forces the SMT solver to find DIPs with higher pruning power, reducing the size of a complete set of DIPs.

**Using TimeOut:**

For an SMT or a SAT attack, the execution time is determined based on the formula, $\sum_{i=1}^{N} t(i)$, where $t(i)$ is the execution time of the $i^{th}$ iteration of an SMT attack. Hence, by just reducing the number of SAT iterations $N$, we cannot guarantee a shorter execution time, because finding a DIP with tighter constraint may pose a more difficult problem to the SMT solver and increase $t(i)$. For this purpose, we can limit the time allowance for finding a DIP in each iteration. The timeout limit $TO$ prevents the SMT solver from spending a long time for finding a DIP with large HD, when finding such DIP has become excessively difficult. By adapting the timeout feature, during an SMT attack, the HD requirement is reduced when either (1) the SMT solver returns UNSAT, indicating there exist no such input, or when (2) we encounter time-out interrupt. In this case, the HD constraints posed on BitVector theory solver is reduced by one and the SMT solver is called. Note that the time interrupt is supported by MonoSAT [54] used in this work, and many other freely available SMT solvers. Also, note that use of time interrupt pushes the final solution away from a minimal complete set of DIPs. However, our experiments illustrate that this usually results in considerably smaller execution time.

**Enabling Approximate Attacks:**

Our objective is to enable the SMT attack to be carried against a netlist similar to that of Fig. 3.10, which is obfuscated by both SAT Hard (SH) and high Corruption (HC) obfuscation schemes, to find all keys for the HC obfuscation, and to detect the trap of SH

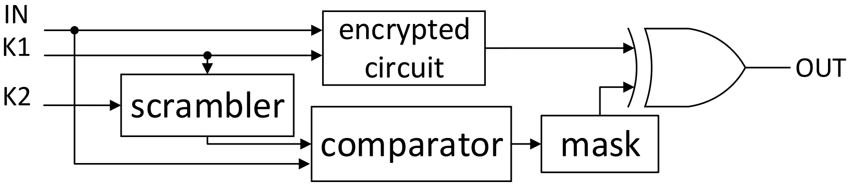


Figure 3.10: A Hybrid Obfuscation Scheme.

obfuscation and exit while generating an approximate key.

The SAT hard obfuscation mechanisms suggested in recent literature, such as *SARLock, Anti-SAT, and SFLL* [18,19,37], have a very small output corruption, and the SAT hardness is maximized when there is only a single input for a given key that results in an incorrect output. The pruning power of DIPs found in each iteration of the SAT solver for SH obfuscation solutions is very small, and each DIP eliminates a single key value. Hence, the number of SAT or SMT iterations increases exponentially with respect to the key size. This is used as a mechanism to trap the SAT solver. To increase the corruption, the SH obfuscation is combined with a HC obfuscation. The purpose of approximate attacks is to find the correct key for the HC obfuscation without being trapped by SH obfuscation.

The accelerated SMT attack could significantly improve the performance of approximate attacks. Since HC obfuscation schemes result in high output corruption, finding DIPs that lead to larger HD at the output biases the SMT attack to find the HC related obfuscation keys in the earlier iterations. The remaining problem is the design of a termination strategy for the accelerated and approximate SMT attack to detect the trap of SH obfuscation, exit and report the approximate key. For this purpose, we use a constraint on the number of allowed repetitions $R$ when HD is very small (e.g. 1). If the remaining and un-found keys are only the SH keys, the SMT keeps finding weak DIPS (HD of 1) and iterations are completed very quickly. By setting the repetition limit R to an appropriately large value, we can detect the trap and terminate the attack.

The unique feature of accelerated approximate attack is that if we remove the timeout (TO) requirement, then the approximate attack guarantees that the HD of the approximately unlocked circuit and that of the functional circuit is at most $HD_{Low}$ bits different, with $HD_{Low}$ being the hamming distance requirement in which the R repetition is taken place. This could be proven as follows: Suppose that there exists an undiscovered discriminating input and two keys that cause larger than $HD_{Low}$ bit difference ($HD_{Low}$+D) in the primary outputs. Hence, the SMT solver when constrained by bitVector theory solver expression for finding HD = $HD_{Low}$+D should return SAT. This contradicts the SMT previous execution control state where the SMT attack for that HD has returned UNSAT, otherwise the HD constraint was not reduced.

**Accelerated SMT attack formulation:**

Alg. 13 demonstrates the reformulated Lazy approach of SMT attack on obfuscated circuits. In this algorithm, the $HD_{High}$, and $HD_{Low}$ are the high and low threshold requirement for hamming distance on primary outputs, $TO$ is the timeout limit per iteration, $R$ is the repetition allowance before exiting and generating an approximate key, and $R_{HD}$ is the hamming distance after which the repetition condition is checked.

The BitVector theory solver input model is defined in lines 14 and 15, and converted to theory constraint expressions in lines 16 and 17. The $T_{CE}$ poses an upper and lower bound on the hamming weight difference of the outputs of two instances of the same circuits with the same input, but two different keys. The SMT attack sweeps the hamming distance in the first while loop, while the second while loop formulate the modulo satisfiability theory attack. The SMT solver receives the $SMT_{LC}$ model, the BitVector theory solver constraint $T_{CE}$ and the timeout allowance $TO$ and check whether there is a valid assignment for $SMT_{LC}$ conditioned that $T_{CE}$ is valid withing $TO$ time allowance. If it exists, the while loop is satisfied. Additionally, it returns the discriminating input $X_{DI}$, the two keys found $(K_1, K_2)$ and a list of learned conflict clauses $CC$. Then the $X_{DI}$, similar to the original SAT attack is used to construct additional DIVC and update the satisfiability model $SMT_{LC}$. At the end of each iteration, the algorithm checks whether the hamming distance is reduced to the limit, where the repetition condition for SH problems is checked. In this case, if the repetition count reaches the specified threshold value $R$, the SMT attack is terminated.

**Algorithm 13** Accelerated SMT Attack

```
function AccSMT_Attack(Obfuscated_Netlist $N_{obf}$, Functional_Circuit $C_{org}$)
    $HD_{High}$ = Number of output bits;                      ▷ Upper hamming distance limit;
    $HD_{Low} = HD_{High} - 1$;                               ▷ Lower hamming distance limit;
    TO = 50s;                                                 ▷ Timeout constraint;
    R = 20;                                                   ▷ Repetition limit;
    $R_{HD} = 1$;                                             ▷ Repetition condition;
    $R_{count} = 0$;                                          ▷ Repetition count variable;
    $KPC \leftarrow$ Replace_KPG($N_{obf}$);
    $C(X,K,Y) \leftarrow$ Circuit_Translation_to_CNF($KPC$);
    $KDC = C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (Y_1 \neq Y_2)$;
    $SCKVC = TRUE$;
    $SATC = KDC \wedge SCKVC$;
    $LC = TRUE$;                                              ▷ Learned Clauses
    $BV(X,K) \leftarrow$ Circuit_Output_to_BitVector($N_{obf}$);
    $BVS(X, K_1, K_2)$ = SUM_of_1s($BV(X,K_1) \oplus BV(X,K_2)$)
    $T_{CE} \leftarrow BVS(X,K_1,K_2) \geq HD_{Low}$;           ▷ Theory constraint expression;
    $T_{CE} \leftarrow T_{CE} \cup (BVS(X, K_1, K_2) \leq HD_{High})$;
    while $HD_{Low} \geq 1$ do
        while $(((X_{DI},K_1,K_2,CC) \leftarrow SMT.Solve(SMT_{LC},\ T_{CE},\ TO)) = T)$ do
            $Y_f \leftarrow C_{org}(X_{DI})$;
            $DIVC = C(X_{DI},K_1,Y_f) \wedge C(X_{DI},K_2,Y_f)$;
            $SCKVC = SCKVC \wedge DIVC$;
            $LC = LC \wedge CC$
            $SMT_{LC} = KDC \wedge SCKVC \wedge LC$;
            if $(HD_{Low} \leq HD_R)$ then
                if $(R_{count} == R)$ then
                    Break;
                $R_{count}$ ++;
        $HD_{Low}$--;
    $Key \leftarrow SMT.Solve(SMT_{LC})$;
```

Table 3.1: ISCAS-85 Benchmarks and their Characteristics.

| Circuit | c432 | c499 | c880 | c1355 | c1908 | c2670 | c3540 | c5315 | c7552 |
|---|---|---|---|---|---|---|---|---|---|
| # of Inputs | 36 | 41 | 60 | 41 | 33 | 233 | 50 | 178 | 207 |
| # of Outputs | 7 | 32 | 26 | 32 | 25 | 140 | 22 | 123 | 108 |
| # of Gates | 120 | 162 | 320 | 506 | 603 | 872 | 1179 | 1726 | 2636 |

additionally if for HD of 1, the SMT solver can no longer find a satisfying assignment, the SMT attack is terminated. A final call to SMT solver with the constructed satisfiability module theory model generates the key.

## 3.7 Experimental Results

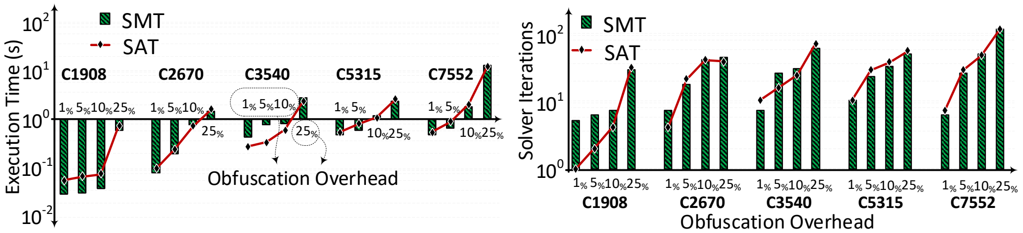
For evaluating different modes of SMT Attack, we used a farm of desktops with 4-core Intel Core-i5 CPU, running at 1.8GHz, with 8 GB RAM. The operating system on desktops was Ubuntu Server 16.04.3 LTS. For a fair comparison, and to reduce the impact of the operating system background processes, we dedicated one desktop to each SMT solver at a time. For benchmarking, we used most of ISCAS-85 benchmarks, characteristics of which is listed in Table 4.7. Since MiniSAT has been used in the SMT Solver as its built-in SAT solver, we use the default values of resource limits in MiniSAT as resource limits of the SMT attack (68 years for the CPU time limit and $\approx$ 2147 TB for the memory usage limit). As the baseline for comparing SMT attack performance against a pure SAT attack, we employed the Lingeling-based SAT attack by [10]. In addition, for each attack we ran the solvers *Five* times on SMT and SAT solvers [56] and reported the average runtime.

### 3.7.1 Evaluation of SMT reduced to SAT Attack

As explained in section 3.6.1, and explained by Alg. 10 the SMT solver could be used for a SAT attack using the same formulation as the original SAT attack as proposed in [9, 10]. In this section, we evaluate the performance of SMT attack when used in this mode. The purpose of this sections is to illustrate that attack formulate using the SMT solver is a superset of SAT attacks, and with the same formulation provides similar performance. For

Table 3.2: Execution Time of SAT vs SMT (Attack Mode 1).

| Circuit | c2670 | | | | c3540 | | | | c5315 | | | | c7552 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SAT | | SMT | | SAT | | SMT | | SAT | | SMT | | SAT | | SMT | |
| | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time |
| 1% | 3 | 0.102 | 5 | 0.474 | 10 | 0.513 | 8 | 1.31 | 9 | 0.405 | 10 | 0.441 | 11 | 0.577 | 19 | 0.806 |
| 5% | 45 | 1.514 | 57 | 3.589 | 19 | 1.502 | 25 | 1.249 | 32 | 1.354 | 24 | 2.433 | 67 | 5.271 | 42 | 4.261 |
| 10% | 312 | 14.08 | 342 | 15.752 | 36 | 1.782 | 36 | 2.973 | 59 | 3.798 | 57 | 4.881 | 97 | 15.82 | 94 | 15.67 |
| 25% | 781 | 114.5 | 692 | 108.6 | 77 | 9.796 | 65 | 8.462 | 95 | 19.63 | 107 | 22.48 | 215 | 225.6 | 228 | 270.8 |



(a) (b)

Figure 3.11: Comparing the Performance of SMT-attack with that of original SAT-attack.

this comparison, we employed two obfuscation methods: (1) random XOR/XNOR insertion (RLL) [7], and (2) obfuscation using nets with unbalanced probabilities (*IOLTS'14*) [57]. ISCAS-85 benchmarks are obfuscated using these schemes with obfuscation overhead ranging from 1% to 25%.

Table 3.2 compares the execution time of SMT attack and the SAT attack proposed in [9,10] when RLL obfuscation is deployed. As captured in this table, the execution time of the SMT attack when reduced to SAT Attack is approximately equivalent, in terms of number of iteration and execution time, with that of an original SAT attack across all benchmarks and all ranges of obfuscation overhead. Fig. 3.11 illustrates the same comparison when the *IOLTS'14* obfuscation method is deployed. As illustrated, the SMT reduced to SAT, in terms of performance, behaves similar to the SAT attack.

### 3.7.2 Evaluation of Eager SMT Attack

We used the Delay Logic Locking scheme [27] in our case study to show the extended capabilities of the SMT attack in solving obfuscation problems that cannot be modeled in a

Table 3.3: Execution Time of SMT Attack in the Eager Mode (Attack Mode 2).

| Circuit | c1908 | c2670 | c3540 | c5315 | c7552 |
|---|---|---|---|---|---|
| 1% | 0.077 + 1.663 | 0.068 + 170.0 | 0.053 + 4.054 | 1.291 + 114.6 | 0.580 + 138.6 |
| 2% | 0.016 + 1.919 | 0.221 + 175.6 | 0.200 + 5.001 | 1.535 + 144.6 | 1.808 + 185.5 |
| 3% | 0.054 + 2.161 | 0.337 + 212.7 | 1.359 + 6.328 | 3.057 + 160.4 | 2.247 + 245.9 |
| 5% | 0.075 + 2.810 | 0.495 + 248.4 | 1.553 + 8.325 | 3.891 + 256.9 | 7.812 + 353.3 |
| 10% | 0.499 + 3.812 | 38.78 + 407.1 | 1.524 + 14.35 | 16.19 + 550.3 | 33.92 + 782.7 |
| 25% | 8.951 + 21.71 | 112.4 + 972.5 | 9.459 + 92.42 | 60.30 + 1567 | 2920 + 5244 |

SMT execution time = $x + y$, $x$ : The execution time of the SAT engine of the SMT Solver,
$y$ : The execution time of the theory engine of the SMT Solver

SAT attack. The Eager approach of SMT attack is evaluated in this section, and the Lazy approach is evaluated in the following section. Additionally, to increase the obfuscation difficulty and demonstrate the strength of the SMT attack, in addition to obfuscation using DLL, we obfuscated the circuit with additional MUX and XOR gates using gate insertion policy in *IOLTS'14* [57], such that 50% of the keys are used for *DLL*, and 50% for *IOLTS'14* obfuscation. Finally, we used some of the keys for both logic and delay obfuscation to create dependencies such that the solvers could not divide and conquer the attack.

The Eager attack against DLL was formulated in Alg. 11. As the algorithm suggests, the Eager approach attacks the obfuscation in two separate phases. In the first phase, the theory solver models and constrains the problem and calls the SMT solver to extract all valid key combinations. The key combinations are converted into CNF statement, which is passed to the SAT solver. In the second phase, the SAT solver attacks the circuit satisfiability problems augmented with these additional CNF clauses on valid key combinations, and make a new round of calls to the SMT solvers. As illustrated in Fig. 3.7(a), the invocation of theory and SAT solver, and the overall SMT attack is serialized. Accordingly, in order to reflect our experimental results for evaluating of Eager approach, we separate the execution time of theory solver and that of the SAT solver.

Table 3.3 captures the results of Eager SMT attack for different ISCAS-85 benchmarks with different obfuscation overhead. The theory execution time indicates the time required by graph theory to find the all possible and valid key combinations (where only one of them is valid). Similarly, SAT execution time demonstrates the time taken by SAT solver to find a valid key, given the additional theory solver generated constraining clauses. As illustrated in this table, the SMT attack, in all cases is concluded and reported the correct key. The result of the pure SAT attack is not reported, as it always produces the wrong key for being oblivious to the DLL key values. Hence, the SMT solver in this respect extends the attack capability by means of including various theory solvers.

Note that the execution time of the SAT solver (the x value in each column of reported data in Table 3.3) depends on the (1) size of the circuit, and (2) the percentage of obfuscated cells. Hence the circuit c7552, for being larger than c1908 has a longer SAT attack time across all percentage obfuscation points. In addition, the increase in the SAT attack time is only slightly super-linear (close to polynomial) with respect to increase in the degree of obfuscation. On the other hand, the execution time of the theory solver (the y value in each column of reported data in Table 3.3) depends on (1) the number of input, (2) the number of outputs, and (3) the degree of obfuscation. Hence, a circuit with larger number of IOs has a longer execution time for its theory solver, but the execution time is bounded by O(NM), with M and N being the number of inputs and outputs respectively. This indicate that the run-time of theory solver (unlike the MILP-based attack that was suggested in [27]) does not exponentially increase with respect to number of timing paths in a netlist, as it only depends on the number of IOs and not the total number of timing paths. In addition, as illustrated, by increasing the degree of obfuscation, similar to SAT attack, the execution time of theory solver grows slowly with a close to polynomial paste.

### 3.7.3 Evaluation of Lazy SMT Attack

The Lazy approach of SMT attack, as illustrated in Fig. 3.7(b), uses the SMT solve function to simultaneously solve the theory and circuit SAT problem. In this approach, the theory model is defined but is not solved. In many applications, the Lazy approach outperforms the Eager solution. In addition, there are situations, where the Eager solution faces exponential runtime if solved separately. As an instance, SRCLock [26] focus on posing exponential runtime on pre-processor needed for detection of cycles, Hence, the Eager approach is not

Table 3.4: Execution Time of SMT Attack in the Lazy Mode (Attack Mode 3).

| Circuit | c432 | c499 | c880 | c1355 | c1908 | c2670 | c3540 | c5315 | c7552 |
|---|---|---|---|---|---|---|---|---|---|
| 1% | 0.033 | 0.177 | 0.263 | 0.567 | 0.466 | 20.44 | 0.983 | 11.53 | 13.07 |
| 2% | 0.049 | 0.262 | 0.325 | 0.676 | 0.596 | 21.86 | 3.443 | 11.76 | 17.83 |
| 3% | 0.065 | 0.329 | 0.350 | 0.877 | 0.723 | 23.39 | 2.436 | 15.27 | 19.04 |
| 5% | 0.049 | 0.340 | 0.517 | 1.085 | 1.456 | 28.87 | 2.587 | 38.87 | 45.96 |
| 10% | 0.204 | 0.503 | 1.195 | 5.622 | 3.334 | 83.06 | 6.712 | 94.80 | 319.6 |
| 25% | 0.599 | 1.481 | 2.036 | 297.2 | 95.67 | 2706 | 126.3 | 552.8 | 8045 |

Table 3.5: Comparing the AccSMT (Attack Mode 4) with the Original SAT Attack.

| Circuit | c2670 | | | | c3540 | | | | c5315 | | | | c7552 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SAT | | AccSMT | | SAT | | AccSMT | | SAT | | AccSMT | | SAT | | AccSMT | |
| | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time |
| 1% | 3 | 0.102 | 2 | 0.316 | 10 | 0.513 | 3 | 0.185 | 9 | 0.405 | 2 | 0.163 | 11 | 0.577 | 3 | 0.374 |
| 5% | 45 | 1.514 | 11 | 3.589 | 19 | 1.502 | 6 | 0.761 | 32 | 1.354 | 6 | 0.408 | 67 | 5.271 | 17 | 2.607 |
| 10% | 312 | 14.08 | 26 | 5.817 | 36 | 1.782 | 11 | 1.236 | 59 | 3.798 | 12 | 1.753 | 97 | 15.82 | 19 | 4.721 |
| 25% | 781 | 114.5 | 107 | 24.05 | 77 | 9.796 | 16 | 1.606 | 95 | 19.63 | 27 | 7.916 | 215 | 225.6 | 24 | 23.52 |

even applicable. However, the parallel invocation of the theory and SAT solver, and the resulting literal exchange, and the additional constraints posed on the solver could result in significant reduction in the time needed to explore the problem's decision tree, and removes the need to complete the pre-processing before starting the SAT attack. Hence, if the execution time of theory solver poses a runtime beyond acceptable, the problem could only be attacked by the Lazy SMT approach.

Table 3.4 shows the Lazy SMT attack execution time on ISCAS-85 benchmarks that were obfuscated using the process that was explained in the previous section (mixing 50% DLL+ 50% IOLTS). Considering the SAT and theory solver are invoked simultaneously, we have a single execution for the entire SMT problem, and unlike Eager approach we cannot separate the execution time of theory solver and the SAT solver. As illustrated, in comparison with the Eager approach, in most cases the Lazy approach finds the key obfuscation key in shorter time.

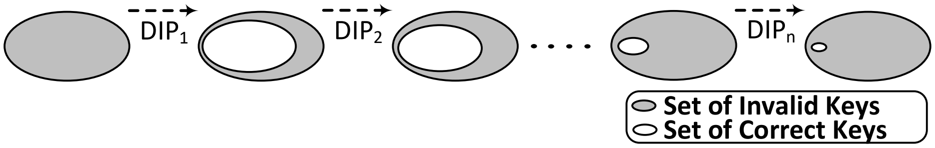
In the Lazy approach, the number of iterations decreases drastically compared to the Eager approach. However, the execution time of each iteration increases. This is because each DIP needs to satisfy both the theory constraints and the circuit SAT formulation.

However, when a DIP is found, it is a stronger DIP with higher pruning power.

By comparing the results of Eager and Lazy approach of SMT attack in Table 3.3 and Table 3.4 we observed that in majority of cases, the Lazy approach outperforms the Eager approach. However, in some cases (e.g. for Benchmark C1908 with 50% overhead), the Lazy approach may become slower than Eager approach, indicating that Lazy approach doesn't always result in the stronger attack. However, note that there exist a set of problems (such as SRCLock [26]), that the Eager approach is not even applicable, since the pre-processing alone (sole invocation of theory solver) cannot conclude in a reasonable amount of time, leaving the Lazy approach as the only solution forward.

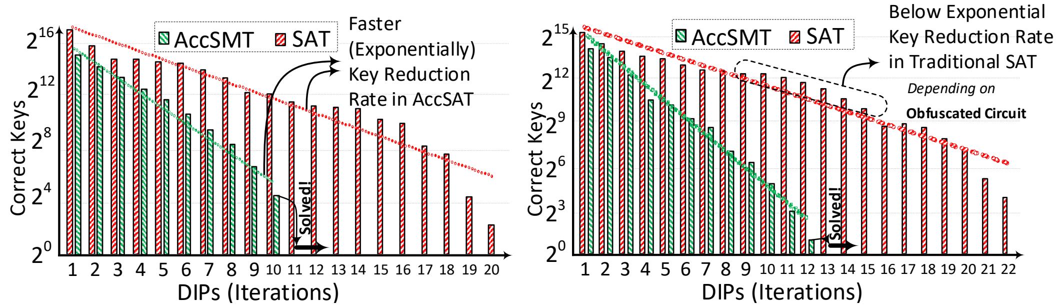
### 3.7.4 Evaluation of Lazy AccSMT Attack

**Ability to find stronger DIPs:** Before invoking the SMT or SAT attack, any key could be considered as a *potentially valid key*. The strength of a DIP comes from its ability in reducing the size of this set in each iteration. After finding each DIP, as illustrated in Fig. 3.12, the size of *potentially valid key* set reduces. When reaching a complete set of DIPs, any key left in this set is a correct key. As discussed in section 3.6.4, a stronger DIP could sensitize a larger number of inconsistencies (due to application of a discriminating input and two different keys) to the primary outputs. Hence, its natural for such a DIP to have a higher pruning power in reducing the number of *potentially valid keys*. To evaluate this claim, we profiled the number of *potentially valid key* after each iteration of SMT and SAT attack, when working on the same obfuscation problem. Fig. 3.13 illustrates the key reduction rate in three ISCAS-85 benchmarks obfuscated by RLL[7]. In all scenarios the DIPs found by AccSMT solver are stronger, as the number of remaining keys is reduced at a significantly higher rate. As illustrated, the number of iterations is also significantly reduced because the complete set of DIPs, when the pruning power of DIPs is higher, is of smaller size.



Complete Set of DIPs = {DIP$_1$, DIP$_2$, DIP$_3$, ..., DIP$_n$}

Figure 3.12: Set of potentially valid keys reduces in each iteration of SMT or SAT attack.



(a)

(b)

Figure 3.13: Key Reduction Rate of the Original SAT Attack and the AccSMT Attack.

### Stronger and shorter attack:

The stronger DIPs found by the AccSMT attack, result in significant reduction of the number of DIs needed for a complete discriminating input set. Each DI is found in one iteration, Hence, smaller number of DIs indicates a smaller number of iterations. Table 3.5 compares the execution time and the number of iterations between the SAT solver and the AccSMT solver. The ISCAS-85 benchmarks for this simulation are obfuscated using RLL [7] obfuscation scheme with the overhead of 1% to 5%. As reported in this table, across all attacks, the AccSMT attack is carried in a smaller number of iterations and requires order(s) of magnitude smaller execution time.

### Ability to carry approximate attack:

As described in section 3.6.4, the AccSMT attack is able to distinguish between SAT-hard (SH) and high-corruption (HC) obfuscation. It quickly finds the correct keys for HC obfuscation, detects the SH trap, exits, and reports the approximate key.

Table 3.6: Execution Time and the Number of Iterations of AccSMT (Attack Mode 4).

| Circuit | c1908 | | c2670 | | c3540 | | c5315 | | c7552 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | #iter | time | #iter | time | #iter | time | #iter | time | #iter | time |
| 1% | 7 | 0.512 | 16 | 3.075 | 8 | 1.304 | 3 | 0.384 | 7 | 2.905 |
| 5% | 18 | 0.701 | 25 | 11.91 | 15 | 1.681 | 11 | 1.707 | 33 | 17.56 |
| 10% | 31 | 4.085 | 51 | 26.47 | 21 | 3.779 | 35 | 7.402 | 61 | 44.07 |
| 25% | 71 | 8.605 | 105 | 76.8 | 66 | 22.91 | 56 | 16.64 | 88 | 58.32 |

To evaluate the approximate mode of the AccSMT attack, we have obfuscated the ISCAS-85 benchmarks using *SARLock + IOLTS14* as suggested in [18]. The overall structure of the obfuscated circuit is illustrated in Fig. 3.10. In this hybrid obfuscation scheme, the SARLock is the SH obfuscation, and the RLL is the HC obfuscation protocol. The invocation of the original SAT attack in [10][9] results in a timeout, due to SARLock trap. However, the AccSMT can very quickly find all the keys for HC obfuscation, detect the SH trap, and report the approximate key. Table 3.6 depicts the number of iterations and execution time of AccSMT attack for finding the approximate keys for each instance of the obfuscated circuit under attack. Note that repetition count (R=20 in our case study) is excluded from this table.

In this section, we introduce a class of Satisfiability Modulo Theory (SMT) attacks on obfuscated circuits. The SMT attack benefits from the expressive nature of theory solvers, that allow the attacker to express constraints that are difficult or even impossible to express using CNF, including timing, delay, power, arithmetic, graph and many other first-order theories. We first illustrated that a SAT attack could be easily implemented using SMT solver to prove that SMT attack is a superset of the SAT attack. Then we proposed two variants of SMT attack on obfuscated circuits using Eager and Lazy approach of SMT solver. We illustrated that using the Eager and Lazy approach, we could break the Delay Logic Locking [27] obfuscation that cannot be broken by a SAT attack, proving that SMT attack's capabilities go beyond a SAT attack. It shows that by only using non-logical properties of a netlist for obfuscation, we not provably increase the security of an obfuscated netlist, indicating the need for further study and exploration in this domain to generate obfuscation schemes with provable security. Then we proposed the Accelerated

SMT attack (AccSMT), and we illustrated that by using theory solvers (BitVector theory solver in this work), we could significantly speed-up the attack against specific obfuscated circuits, and reported significant reduction in the execution time of the AccSMT compared to SAT attack. Finally, we illustrated that with a small modification, the AccSMT could be used as an approximate attack, allowing us to find an approximate key for obfuscation schemes that combine a SAT hard obfuscation with high corruption obfuscation.

# Chapter 4: COMA: Communication and Obfuscation Management Architecture

In this work, we introduce a novel *Communication and Obfuscation Management Architecture* (COMA) to handle the storage of the obfuscation key and to secure the communication to/from untrusted yet obfuscated circuits. COMA addresses three challenges related to the obfuscated circuits: First, it removes the need for the storage of the *obfuscation unlock key* at the untrusted chip. Second, it implements a mechanism by which the key sent for unlocking an obfuscated circuit changes after each activation (even for the same device), transforming the key into a dynamically changing license. Third, it protects the communication to/from the COMA protected device and additionally introduces two novel mechanisms for the exchange of data to/from COMA protected architectures: (1) a highly secure but slow double encryption, which is used for exchange of key and sensitive data (2) a high-performance and low-energy yet leaky encryption, secured by means of frequent key renewal. We demonstrate that compared to state-of-the-art key management architectures, COMA reduces the area overhead by 14%, while allowing additional features including unique chip authentication, enabling activation as a service (for IoT devices), reducing the side channel threats on key management architecture, and providing two new means of secure communication to/from an untrusted chip.

## 4.1 Background

Active metering, Secure Split-Test, logic obfuscation, and solutions such as Ending Piracy of Integrated Circuits (EPIC) have been proposed to protect ICs from supply chain-related security threats by initializing the HW control to a locked state at power-up and hiding the design intent [3, 7, 11, 12, 58–61]. Some of these techniques support single activation, while others support active metering mechanisms. Active metering techniques [3,12,58,60] provide a mechanism for the IP owner to lock or unlock the IC remotely. In these solutions, the locking mechanism is a function of a unique ID generated for each IC, possibly and preferably by a *Physical Unclonable Function* (PUF) [28]. Only the IP owner knows the transition table and can unlock the IC. Active metering, combined with a PUF, makes the key a moving target from chip to chip. However, there exist a few issues with previous metering techniques: first, the key(s) to unlock each IC remains static. Second, these techniques unlock the chips before they are tested by the foundry. Hence, the IP owner can control how many ICs enter the supply chain, but not how many properly tested ICs exit the supply chain. Finally, these techniques do not respond well to the threat of the foundry requesting more IDs by falsifying the yield to be lower during the test process. Such shortcomings can potentially allow the foundry to ship more out-of-spec or defective ICs to the supply chain.

Many of these shortcoming were addressed in FORTIS [62] shown in Fig. 4.1. In FORTIS the registers that hold the obfuscation key are made a part of the scan chain, allowing the foundry to carry structural test by assigning test values to these registers prior to the activation of the IC. Authors of [62] argue that placing a DFT compression logic, not only reduces the test size, but also prevents the readout of the individual register values. After testing the IC, the obfuscation key is transferred and applied to unlock the circuit using two types of cryptographic modules: a public-key crypto engine, and a One Time Pad (OTP) crypto engine.

In FORTIS, the public and private keys are hardwired in the design. A TRNG is used to generate a random number (m) that is treated as a message. This message is encrypted using the private key of the chip to generate a signature sig(m). The actual message and its signature are concatenated and later used as a mean for the authentication of the chip. At the same time, the TRNG generates another random number $K_S$. This random number is used as the key for OTP, and at the same time is encrypted using the public key of the designer to generate $KD_{pub}(K_S)$. OTP uses $K_S$ for encrypting the $(m, sig(m))$, and

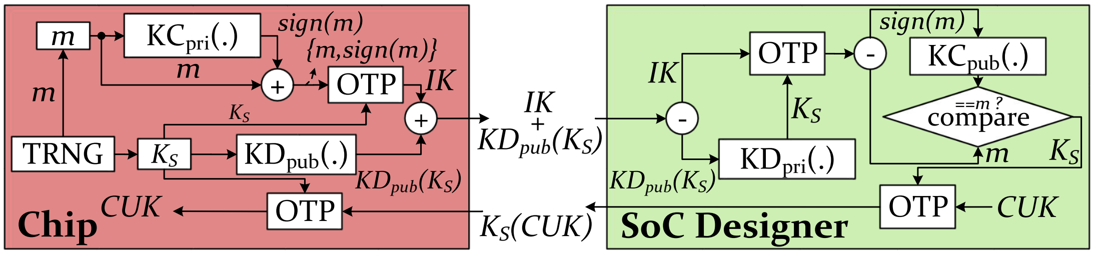


Figure 4.1: FORTIS: Overall Architecture.

the output of OTP is concatenated with the $KD_{pub}(K_S)$. The resulting string of bits is transmitted to the SoC designer. The SoC designer uses a OTP to obtain $m$ and $sig(m)$ for the purpose of authentication. She then uses the private key of the designer to recover $K_S$. Finally, $K_S$ is used by OTP to encrypt the chip unlock key (CUK). The encrypted CUK is transmitted to the chip, decrypted using OTP, and applied to the obfuscation unlock key registers to unlock the circuit.

FORTIS, however, suffers from several security issues including 1) using identical public and private keys in all manufactured chips, and thus its inability for unique device authentication, 2) being vulnerable to modeling attack in which the FORTIS structure is modeled in software for requesting the CUK from SoC designer 3) being vulnerable to side channel attacks on public-key encryption engine aimed at recovering the private key of the chip, 4) being vulnerable to fault attacks in which the value of $K_S$ is fixated, 5) requiring a secure memory for storage of the obfuscation unlock key, and 6) not addressing the mechanism for generating a unique and truly random seed to initialize PRNG. After describing our proposed solution, in section 4.6, we explain how these vulnerabilities are addressed in our proposed solution.

Our proposed solution fits the category of active metering techniques. The key is neither static nor stored in the untrusted chip. A key that is used to activate the IC at the test time cannot be reused to activate the same or a different IC in the future. Hence, the test facility is able to accomplish the test process using ATPG tools with a key which is valid for structural/functional test and it is not valid for any subsequent activation. Additionally, the communication to/from IC is secured using a side-channel protected cryptographic engine, combined with a dynamic switching and inversion structure that enhances the security of the chip against invasive and side-channel attacks. We demonstrate that COMA provides two useful means of secure communication to/from the untrusted chip, one for added security, and one for supporting a higher throughput. The proposed architecture is a comprehensive solution for the key management of the obfuscated IPs, where the challenges related to the activation of the IC and secure communication to/from the IC are addressed at the same time. However, as discussed earlier, it is not a universal solution and would fit within the context of IoT-based solutions or within 2.5D package-integrated solutions, as this solution requires constant connectivity.

## 4.2 Proposed COMA Architecture

The primary goal of the COMA is to remove the need for storing the *obfuscation key* (OK) on an untrusted chip while securing the communication flow used for activation of the obfuscated circuit in the untrusted chip. The additional benefits of the proposed architecture are the implementation of two new modes of 1) highly secure and 2) very high-speed encrypted communication.

We propose two variants of the COMA architecture: The first variant is designed for securing the activation of the obfuscated IP and communication to/from an untrusted IC in 2.5D package-integrated architectures similar to the DARPA SPADE architecture [30] (denoted by 2.5D-COMA). The second proposed architecture is designed for protecting IoT-based or remotely activated/metered devices (denoted by R-COMA). Fig. 4.2 captures the overall architecture of two variants of the proposed COMAs.

### 4.2.1 2.5D-COMA: Protecting 2.5D package integrated system solutions

The DARPA SPADE project [30] explores solutions in which an overall system is split manufactured between two different technologies, In this solution, a trusted IC which is constructed in an older yet secure technology is packaged with an IC fabricated in an

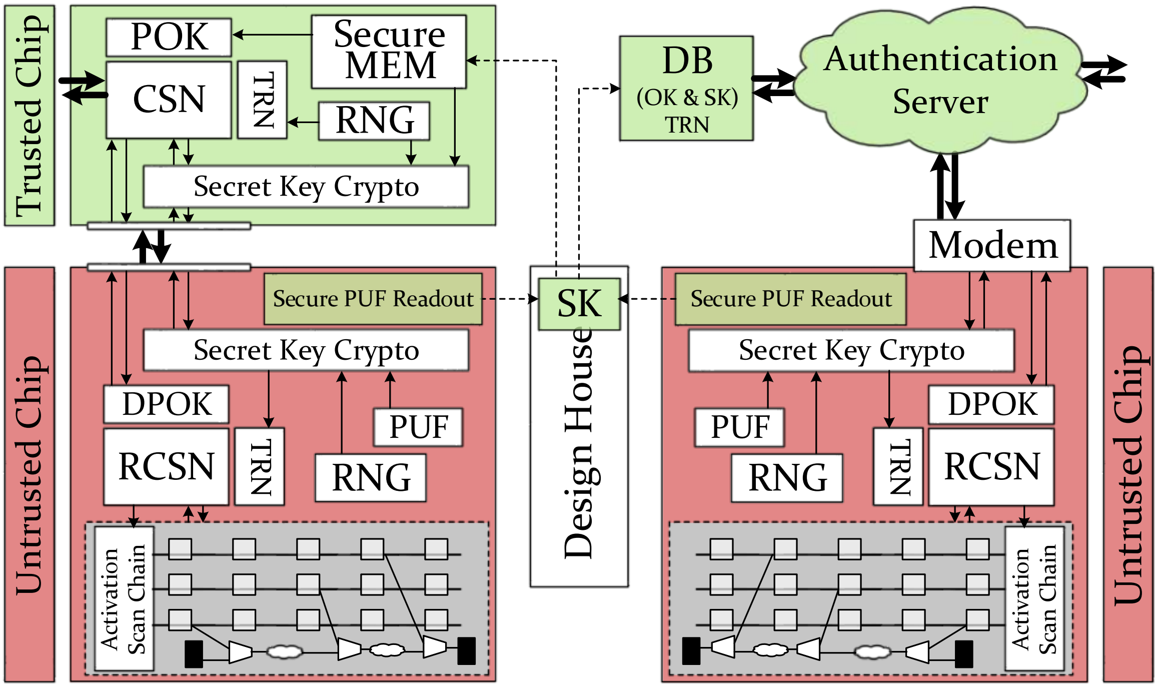


Figure 4.2: Proposed COMAs for (left) 2.5D and (right) IoT-based/remote devices.

untrusted foundry in an advanced geometry. The purpose of this solution is to provide the best of two worlds: the security of older yet trusted technology and the scalability, power, and speed of the newer yet untrusted technology. The 2.5D-COMA is designed to work with an architecture similar to the DARPA SPADE architecture. The proposed solution allows an entire or partial IP in an untrusted chip to be obfuscated, while pushing the mechanism for unlocking and secure activation of the untrusted chip out to a trusted chip. In this solution, the trusted chip encapsulates the sensitive information, verifies the integrity of the untrusted chip, performs sensitive logic monitoring, and controls the activation of the untrusted chip. Also, the key to unlock the obfuscated circuit changes per activation, details of which will be explained shortly.

As shown in Fig. 4.2, the two variants of COMA contain two main parts, the trusted side (*green*) and the untrusted side (*red*). In both variants, the architectures of untrusted chips are identical, and only the architectures of trusted sides are different. In 2.5D-COMA, only the trusted chip is equipped with a secure memory. The secure memory stores the *Obfuscation Key* (OK) and the Secret Key (SK) used for encrypted communication between the trusted and untrusted chips. The SK is generated using a PUF in the untrusted chip, thus it is unique for each untrusted chip, and the untrusted chip does not need a secure memory to store the SK. The *Configurable Switching Network* (CSN) and *Reverse* CSN (RCSN) are logarithmic routing and switching networks. They are capable of permuting the order and possibly inverting the logic levels of their primary inputs while these signals are being routed to different primary outputs. The RCSN is the exact inverse of the CSN. Hence, passing a signal through CSN-RCSN (or RCSN-CSN) will recover the original input. The switching and inversion behavior of CSN-RCSN is configured using a *True Random Number* (TRN). This TRN is generated in the trusted chip to avoid any potential weakening/manipulating of the TRNG. In addition, since the TRNG in COMA is equipped with standard-statistical-tests applied post-fabrication, such as Repetition-Count test and the Adaptive-Proportion test, as described in NIST SP 800-90B [63], any attempt at weakeningthe TRNG during regular operation (i.e. fault attack) can be detected by continuously checking the output of a source of entropy for any signs of a significant decrease in entropy, noise source failure, and hardware failure. By using TRN for the CSN-RCSN configuration, any signal passing through the CSN is randomized, and then by passing through the RCSN is recovered. Additional details are provided in section 4.3.1.

The untrusted chip unlock process in COMA is as follows: Prior to each activation, the CSN and RCSN are configured with the same TRN. Since the SK is a PUF-based key generated at the untrusted side, first the SK must be securely readout from untrusted chip. This is done by deploying public key cryptography, the details of which are described in section 4.3.4. Then, the trusted chip encrypts the TRN using the SK and sends it to the untrusted chip. To perform an activation, as shown in Fig. 4.2, the OK is read in segments, denoted as *Partial Obfuscation Key* (POK), and is passed through the CSN and encryption on the trusted side and the decryption and RCSN on the untrusted side. This process is repeated every time the obfuscated circuit in the untrusted chip is to be activated, each time using a different TRN for configuring the CSN-RCSN. Usage of a different TRN as the configuration input for the CSN-RCSN for each activation randomizes the input data to Secret key crypto engine. Hence, by using a different TRN for each activation, the obfuscation key (after passing through CSN) is transformed into a one-time license, denoted as *Dynamic Activation License* (DAL). Since the OK is read and sent in segments (from trusted chip), the DAL will be received (at untrusted chip) in segments, denoted as *Dynamic Partial Obfuscation Key (DPOK)*, shown in Fig. 4.2, and is used as an input to RCSN. Passing DPOKs through RCSN recovers the POKs, and concatenating the POKs will generate the OK. Note that the DAL is only valid until the TRN is changed. So, the DAL cannot be used to activate other chips or the same chip at a later time.

In 2.5D-COMA, the untrusted chip(s) is used as an accelerator, and for safety reasons should not be able to directly communicate to the outside world. Hence, all communication to/from the untrusted chip must go through the trusted chip. In addition, it is possible that the computation, depending on the sensitivity of processed data, is divided between the trusted and untrusted chips. Hence, there is a need for constant communication between the trusted and untrusted chips. The communication needed is sometimes for limited but highly sensitive data, and sometimes for vast amounts of less sensitive data. As illustrated in Fig. 4.3, the proposed architecture is designed to provide two hybrid means of encrypted communication : (1) Double-Cipher Communication (DCC) as ultra-secure communication, and (2) Leaky-Cipher Communication (LCC) as ultra-fast communication mechanism.

### Double-Cipher Communication (DCC)

As shown in Fig. 4.3(a), in **DCC** each message passes through both CSN-RCSN and the secret key cryptography engine, where the TRN used in CSN-RCSN is renewed every $U$ cycles. DCC provides the ultimate protection against side-channel attacks. In DCC mode, two necessary requirements for mounting a side channel attack are eliminated. The side channel attack aims to break the cryptography system by analyzing the leaked side channel information for different *input patterns*. Hence, (1) the degree of correlation between the input and the leaked side-channel information, and (2) the intensity of side-channel

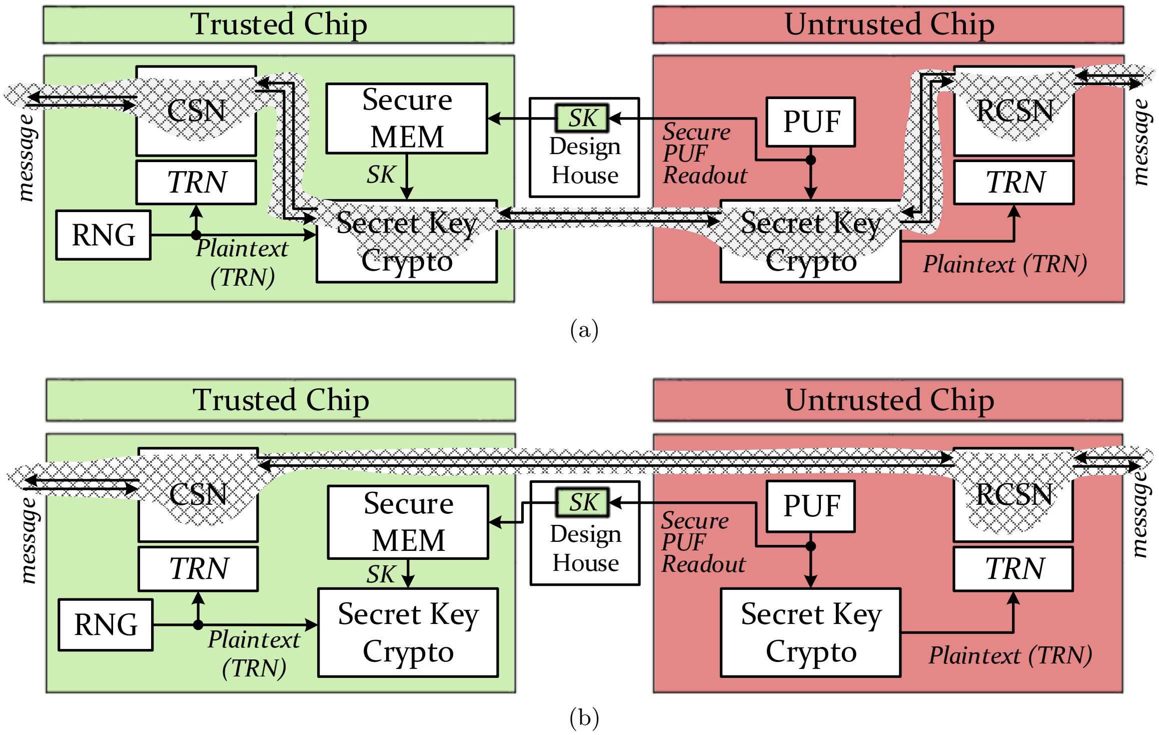


Figure 4.3: Modes of Encrypted Communication in COMA: (a) DCC, (b) LCC.

variation, are important. In COMA, the attacker cannot control the input to the secret-key cryptography. In addition, the input to the CSN is randomized using a TRN and then passed to the secret-key cryptography, removing the correlation between leaked side channel info (from secret-key cryptography) and the original input to the CSN. Additionally, the secret-key cryptography engine is side-channel protected to pass a t-test [64]. So, the intensity and variation in side-channel information is significantly reduced, making the DCC an extremely difficult attack target.

### Leaky-Cipher Communication (LCC)

**LCC** is a fast and energy efficient mode of communication between the trusted (or remote device) and the untrusted chip. As illustrated in Fig. 4.3(b), in this protocol, the CSN-RCSN pair is used for exchanging data. The secret key cryptography engine is used to transmit a TRN from one chip to the other. Since the throughput of TRNG is the bottleneck point compared to the performance of CSN-RCSN, the TRNG is used as a seed generator

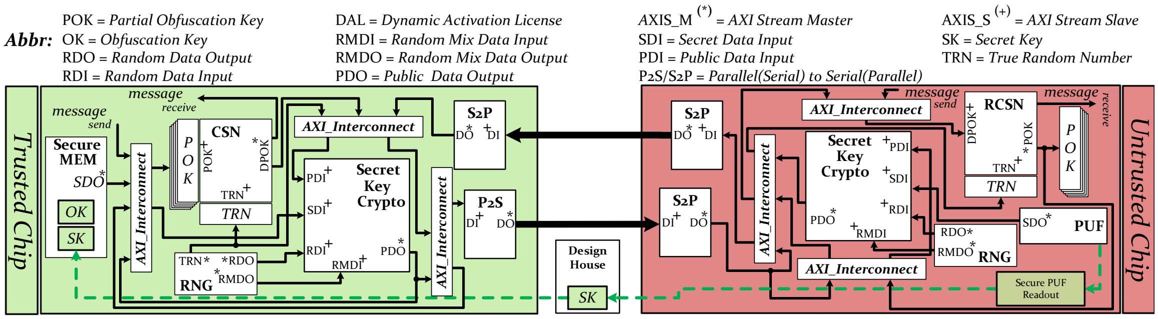


Figure 4.4: 2.5D-COMA Architecture.

to the PRNG (which offers higher performance) on both sides, Hence, in LCC mode, PRNG is used to configure the CSN-RCSN to avoid any performance degradation on transmitting data. For $U$ consecutive cycles, the PRNG is kept idle allowing the CSN to use the same PRNG output for $U$ cycles. It not only reduces the power consumption of PRNG and TRNG, it also provides faster communication in LCC mode. However, using this model of communication is prone to algebraic and SAT attacks as each communicated message leaks some information about the TRN used to configure the CSN-RCSN pair. If an attacker can control the message and observe the output of the CSN, each communicated message leaks some information about the key, reducing its security. Extracting the key from such observations is possible by various attack models, including Satisfiability attacks. Hence, an attacker with enough time and enough traces could extract the TRN and retrieve the communicated messages. Preventing such attacks poses a minimum limit to $U$ (the update frequency of the PRNG). $U$ should be small to prevent SAT and other trace-based learning or analysis attacks, but large enough to be energy efficient. In Section 4.5, we deploy a SAT attack against LCC and will further elaborate on the required TRN update frequency.

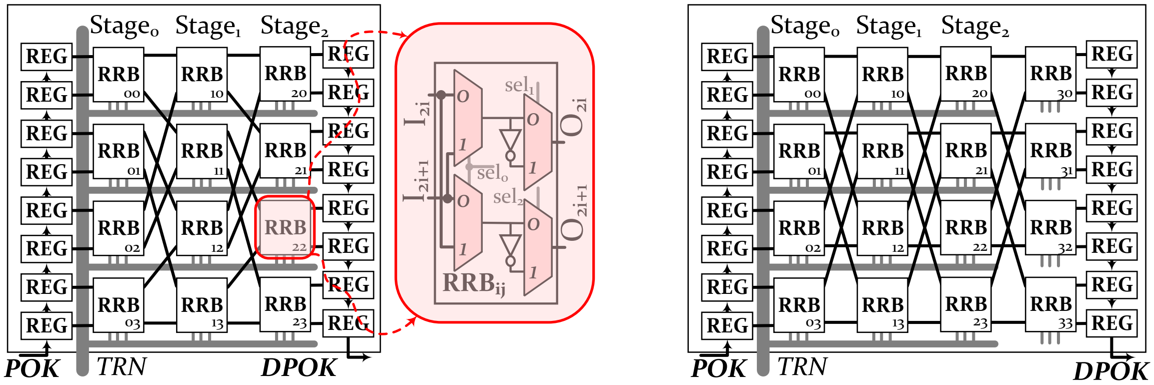
### 4.2.2 R-COMA: Protecting IoT devices

The R-COMA architecture in the untrusted chip is identical to that of 2.5D-COMA. However, the trusted chip is replaced with a remote key management service. The R-COMA provides a mechanism for an IP owner to remotely activate parts or entire functionality of the hardware. Similar to 2.5D-COMA, the DAL is different from chip to chip and from activation to activation. In R-COMA, the obfuscation unlock key is stored in a central database, while the CSN, the TRNG for configuring CSN-RCSN, and the secret key cryptography engine are implemented in software.

In *R-COMA*, an authentication server (AS) first securely receives the PUF-based SK from the untrusted chip. Then, it generates a TRN and sends it to the untrusted chip for RCSN configuration. Then, the AS starts sending the obfuscation key (OK). For the activation phase, the communication is double encrypted and authenticated using the CSN-RCSN and side-channel protected cryptography engine. Each COMA-protected device needs to be registered with the AS to receive the obfuscation key. The registration is done using the PUF-ID of the untrusted chip. Hence, the PUF is used for both authentication and generation of the secret key for communication. In R-COMA, the generation of $DAL$ is granted after PUF authentication, and is based on the generated TRN, and the stored OK, which is generated at design time. The generation of DAL is algorithmic and takes linear time.

## 4.3 Implementation Detail of COMA

Fig. 4.4 captures the overall architecture of COMA and relation and connectivity of its macros. As discussed, COMA supports both key-management and secure data communication. Based on the selected mode of communication (LCC/DCC), the message passes through {CSN → RCSN} *or* {CSN → encryption → decryption → RCSN}. RNG, which contains both TRNG and PRNG, is used in both sides. In the trusted chip, RNG is used for implementing side-channel protected cryptography engine, as well as generating the configuration of the CSN-RCSN (TRN). In the untrusted side, it is used only for implementing the side-channel protected cryptography engine. Finally, PUF is engaged in the untrusted chip for both unique IC authentication and for generation of the secret key for encryption. As shown in Fig. 4.4, all modules employ an AXI-stream interface to maximize the simplicity of the overall design, and minimize the overhead incurred by the controller of the top module in each side. The description of the behavior of each macro in COMA is provided



(a) (b)

Figure 4.5: Logarithmic Network (a) Omega-based Blocking, (b) near Non-blocking.

next.

### 4.3.1 Configurable Switching Network (CSN)

The CSN is a logarithmic routing network that could route the signals at its input pins to its output pins while permuting their order and possibly inverting their logic levels based on its configuration. Fig. 4.5(a) captures a simple implementation of an 8-by-8 CSN using *OMEGA* [65] network. The network is constructed using routing elements, denoted as Re-Routing Blocks (RRB). Each RRB is able to possibly invert and route each of the input signals to each of its outputs. The number of RRBs needed to implement this simple CSN for $N$ inputs ($N$ is a power of 2) is simply $N/2 * logN$. Each CSN should be paired with an RCSN. The RCSN, is simply constructed by flipping the input/output pins of RRB, and treating the CSN input pins as its output pins and vice versa.

The *OMEGA* network along with many other networks of such nature (Butterfly, etc.) are blocking networks [65], in which we cannot produce all permutations of input at the network's output pins. This limitation significantly reduces the ability of a CSN to randomize its input. Also, we will show that a blocking CSN can be easily broken by a SAT attack within few iterations.

Being a blocking or a non-blocking CSN depends on the number of stages in CSN. Since no two paths in an RRB are allowed to use the same link to form a connection, for a specific number of RRB columns, only a limited number of permutations is feasible. However, adding extra stages could transform a blocking CSN into a strictly non-blocking CSN. Using a strictly non-blocking CSN not only improves the randomization of propagated messages through the CSN, but also improves the resiliency of these networks against possible SAT attacks for extraction of a TRN used as the key for a CSN-RCSN cipher. A non-blocking logarithmic network could be represented using $LOG_{n,m,p}$, where $n$ is the number of inlets/outlets, $m$ is the number of extra stages, and $p$ indicates the number of copies *vertically cascaded* [66].

According to [66], to have a strictly non-blocking CSN for an arbitrary $n$, the smallest feasible values of $p$ and $m$ impose very large area/power overhead. For instance, for $n = 64$, the smallest feasible values, which make it strictly non-blocking, are $m = 3$ and $p = 6$, which means there exists more than 5× as much overhead compared to a blocking CSN with the same $n$, resulting in a significant increase in the area and delay overhead. To avoid such large overhead, we employ a *close to non-blocking CSN* described in [66] to implement the CSN-RCSN pair. This network is able to generate not all, but *almost all* permutations, while it could be implemented using a $LOG_{n,log_2(n)-2,1}$ configuration, meaning it needs $log_2(n) - 2$ extra stages and no additional copy. Fig. 4.5(b), demonstrates an example of such a close-to-non-blocking CSN with $n = 8$. In the results section, we demonstrate that using these close-to-non-blocking CSNs enhances the resiliency of a CSN against SAT attack, even in small sizes of CSNs with significantly lower power, performance and area (PPA) overhead.

### 4.3.2 Authenticated Encryption with Associated Data

The Authenticated Encryption with Associated Data (AEAD) is used in the DCC mode for communicating messages, and in the LCC mode for the initial transmission of the CSN-RCSN key (TRN). Authenticated ciphers incorporate the functionality of confidentiality, integrity, and authentication. The input of an authenticated cipher includes Message, Associated Data (AD), Public Message Number (NPUB), and a secret key. The ciphertext is generated as a function of these inputs. A Tag, which depends on all inputs, is generated after message encryption to assure the integrity and authenticity of the transaction. This tag is then verified after the decryption process. The choice of AEAD could significantly affect the area overhead of the solution, the speed of encrypted communication, and the extra power consumption. To show the performance, power, and area trade-offs, we employ two AEAD solutions: a NIST compliant solution (AES-GCM), and a promising lightweight solution (ACORN).

AES-GCM is the current National Institute of Standards and Technology (NIST) standard for authenticated encryption and decryption as defined in [67]. ACORN is one of two finalists of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), in the category of lightweight authenticated ciphers, as defined in [68]. An 8-bit side-channel protected version of AES-GCM and a 1-bit side-channel protected version of ACORN are implemented as described in [69]. Both implementations comply with lightweight version of the CAESAR HW API [70].

Our methodology for side channel resistant is threshold implementation (TI), which has wide acceptance as a provably secure Differential Power Analysis (DPA) countermeasure [71]. In TI, sensitive data is separated into shares and the computations are performed on these shares independently. TI must satisfy three properties: 1) Non-completeness: Each share must lack at least one piece of sensitive data, 2) Correctness: The final recombination of the result must be correct, and 3) Uniformity: An output distribution should match the input distribution. To ensure uniformity, we refresh TI shares after non-linear transformations using randomness. We use a hybrid 2-share/3-share approach, where all linear transformations in each cipher are protected using two shares, which are expanded to three shares only for non-linear transformations.

To verify the resistance against DPA, we employ the Test Vector Leakage Assessment methodology in [64]. We leverage a ”fixed versus random” non-specific t-test, in which we randomly interleave first fixed test vectors and then randomly-generated test vectors, leading to two sequences with the same length but different values. Using means and variances of power consumption for our fixed and random sequences, we compute a figure of merit $t$. If $|t| > 4.5$, we reason that we can distinguish between the two populations and that our design is leaking information. The protected AES-GCM design has a 5-stage pipeline and encrypts one 128-bit input block in 205 cycles. This requires 40 bits of randomness per cycle. In ACORN-1, there are ten 1-bit TI-protected AND-gate modules, which consume a total of 20 random reshare, and 10 random refresh bits per state update. In a two-cycle architecture, 15 random bits are required per clock cycle.

### 4.3.3 Random Number Generator (RNG)

An RNG unit is required on both sides to generate random bits for side channel protection of AEAD units, a random public message number (NPUB) for AEAD, and TRNs for CSN-RCSN. We adopted the ERO TRNG core described in [72], which is capable of generating only 1-bit of random data per over 20,000 clock cycles. In our TI implementations, AES-GCM needs 40 and ACORN 15 bits of random data per cycle. So, we employed a hybrid RNG unit combining the ERO TRNG with a Pseudo Random Number Generator (PRNG). TRNG output is used as a 128-bit seed to PRNG. The PRNG generates random numbers needed by other components. The reseeding is performed only once per activation.

The choice of PRNG depends on the expected performance and overhead. To support COMA, we adopted two different implementations of PRNG: (1) AES-CTR PRNG, which is based on AES, is compliant with the NIST standard SP 800-90A, and generates 12.8 bits per cycle. (2) Trivium based PRNG, which is based on the Trivium stream cipher described in [73]. The Trivium-based PRNG is significantly smaller in terms of area and much faster than AES-CTR PRNG. It can generate 64 bits of random data per cycle, however, it is not compliant with the NIST standard.

### 4.3.4 PUF and Secure PUF Readout

The response of the PUF to a challenge selected randomly by Enrollment Authority (SoC designer) is used as the secret key in AEAD. Hence, the readout of the PUF-response should be protected. The simplest solution for the safe readout of a PUF-generated key is to enable the readout by burning one set of fuses, and disabling it by burning a second set of fuses. However, this solution, especially when combined with a weak PUF, is not likely to be resistant against the untrusted foundry, which may possibly burn the first set of fuses, read out PUF key, and then repair fuses before releasing the chip. To avoid this problem, we implement a lightweight one-sided public key cryptography (encryption only) based on Elliptic-Curve Cryptography (ECC). Considering the PUF readout is a one-time event, the performance of the public-key cryptography engine is not critical.

In order to prevent any attempts at fully characterizing a PUF in the untrusted foundry, only strong PUFs, e.g. an arbiter PUF, are considered. The secure readout of the PUF key is allowed only at the device enrollment time, in the secure facility. During the secure readout, the strong PUF is fed with multiple challenges selected by the Enrollment Authority. The corresponding PUF responses are encrypted by the untrusted chip using the public key of the Enrollment Authority, that is embedded in the chip layout or stored in the one-time programmable memory. Only the Enrollment Authority has access to the decrypted responses. Afterwords, one of the previously applied challenges is randomly selected and used for the generation of the secret key. This challenge is then hardwired on the untrusted chip, and the PUF response to that challenge is recorded by the Enrollment Authority. This PUF response is then stored in the secure memory of the trusted chip in 2.5D-COMA, or in the secure cloud directory in R-COMA. This process makes each PUF key unique to a given device, and resistant against any unauthorized readout by the untrusted foundry.

Still, additional precautions must be taken to protect this scheme against an attack aimed at replacing a real PUF by a pseudo-PUF, generating randomly looking responses that can be easily calculated by an attacker. An example of such a pseudo-PUF may be a lightweight symmetric-key cipher, with a fixed key known to the untrusted party, encrypting each challenge and outputing a ciphertext as the PUF response.

Such pseudo-PUF should be treated as a Trojan and detected by Enrollment Authority using the best known anti-Trojan techniques, e.g., those based on the measurement and analysis of the power consumption during the operation of the device [74]. Additional methods may be used to differentiate the outputs of a strong PUF from encrypted data, e.g., using known correlations between the PUF responses corresponding to closely-related challenges, such as challenges differing on only one bit position, or being mutual complements of each other [75]. These kinds of PUF-health tests may be specific to a particular strong PUF type, e.g., to an arbiter PUF, and will be the subject of our future work.

## 4.4 COMA Resistance against various Attacks

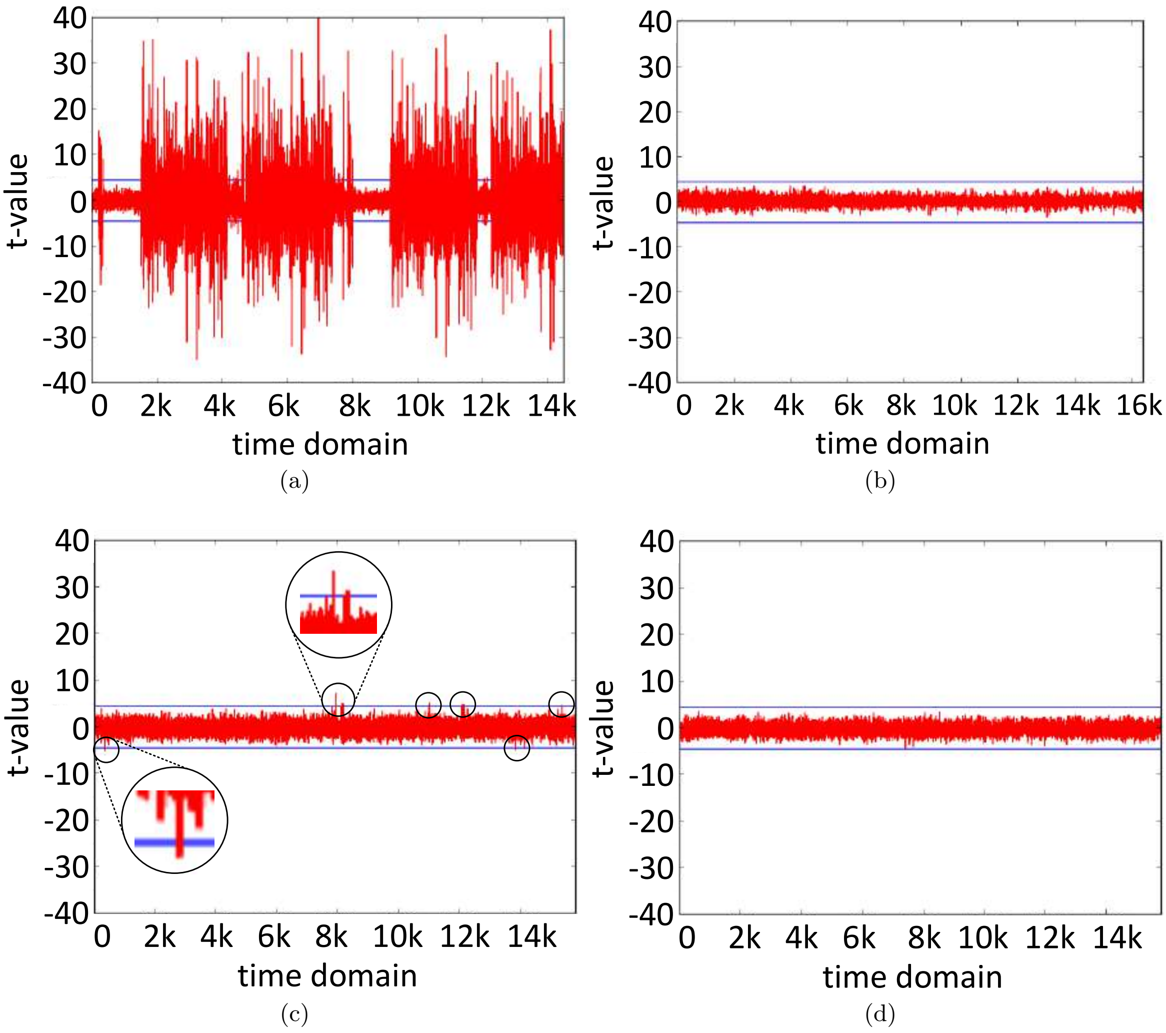
### 4.4.1 Assumed Attacker Capabilities

Different sources of vulnerability are considered in this section to demonstrate the COMA security. The attacker can be an adversary in the manufacturing supply chain, and has access to either the reverse engineered or design house-generated netlist of the COMA-protected untrusted chip. The attacker can purchase an activated COMA-protected IC from the market. The attacker can monitor the side channel information of chips at or post activation. The attacker can observe the communication between untrusted and trusted (or remote manager) chips and could also alter the communicated data. An Attack objective may be (1) extracting the obfuscation key (OK), (2) illegal activation of the obfuscated circuit without extracting the key, (3) extracting the long-term secret key (SK), (4) extracting short-term CSN keys (TRNs), (5) eavesdropping on messages exchanged between the untrusted chip and the external sources, (6) removing the COMA protection, or (7) COMA-protected IC overproduction.

Side Channel Attack (SCA)

The objective of SCA on COMA is to extract either the secret key (SK) used by AEAD or the TRN used by CSN. Extracting a SK is sufficient to break the obfuscation; extracting a TRN reveals only messages sent in the LCC mode.

DCC significantly increases the SCA difficulty, since (1) the AEAD is side-channel protected, and (2) the attacker loses access to the input of AEAD. Fig. 4.6 captures our



(a)

(b)

(c)

(d)

Figure 4.6: The t-test Results for Pr and UnPr version of AES-GCM and ACORN.

assessment of side channel resistance of AEAD using a t-test for unprotected and protected implementations of AES-GCM and ACORN [76]. As illustrated, both implementations pass the t-test, indicating increased resistance against SCA. On the other hand, the inability to control the input to AEAD comes from the COMA requirement of encryption in the DCC mode where a message first passes through the CSN. Hence, there exists no relation between the power consumption of the AEAD and the original input due to CSN randomization. CSN power consumption is also randomized as it is a function of $n$ inputs (possibly known to the attacker) and $3n \times (log_2 n - 1)$ TRN inputs unknown to the attacker, while the TRN is repeatedly updated based on the value of $U$. Note that during the physical design of COMA, the side channel information on power and voltage noise (IR drop) could be further mitigated using timing aware IR analysis [77], and voltage noise aware clock distribution techniques [78,79].

The LCC mode is prone to side-channel, algebraic, and SAT attacks aimed at extracting the TRN. However, the attack must be carried out in a limited time while the TRN of the CSN/RCSN is unchanged. As soon as the TRN is renewed, the previous side-channel traces or SAT iterations are useless. The period of TRN updates ($U$) introduces a trade-off between energy and security and can be pushed to maximum security by changing the TRN for every new input. In section 4.5.3 we investigate the time required to break the LCC using side-channel or SAT attack and accordingly define a safe range for $U$ to prevent such attacks.

### 4.4.2 Reverse Engineering

In COMA, reverse engineering (RE) to extract the secret key from layout is useless as the secret key is not hardwired in the design and is generated based on PUF. RE to extract the key from memory in an untrusted chip is no longer an option as the key is not stored in the untrusted chip. RE to extract the key from the trusted chip's memory is limited by the difficulty of tampering with secure memory in the trusted technology.

### 4.4.3 Algebraic Attacks

Algebraic attacks involve (a) expressing the cipher operations as a system of equations, (b) substituting in known data for some variables, and (c) solving for the key. AES-GCM and ACORN have been demonstrated to be resistant against all known types of algebraic attacks, including linear cryptanalysis. Therefore, in the absence of any new attacks, the DCC mode is resistant against algebraic attacks. Using CSN and RCSN for fast encryption is new and requires more analysis. CSN can be expressed as an affine function of the data input $x$, of the form $y = A \cdot x + b$, where $A$ is an $n \times n$ matrix and $b$ is an $n \times 1$ vector, with all elements dependent on the input TRN. Although recovering $A$ and $b$ is not equivalent to finding the TRN, it may enable the successful decryption of all blocks encrypted using a given TRN. We protect against this threat in two ways: (1) The number of blocks encrypted using a given TRN is set to the value smaller than $n$, which prevents generating and solving a system of linear equations with $A$ and $b$ treated as unknowns, (2) We partially modify the TRN input of CSN with each block encryption (by a simply shifting the input TRN bits), so the values of $A$ and $b$ are not the same in any two encryptions, without the need of feeding CSN with two completely different TRN values.

### 4.4.4 Counterfeiting and Overproduction

COMA can be used to prevent the resale of used ICs, usage of illegal copies, and reproduction of a design. During packaging and testing, each COMA protected IC is first tested and then is matched with a trusted chip. So, the untrusted chip can only be activated by the matched trusted chip or the registered remote manager. Building illegal copies that work without the secure chip (or remote activation) and reproduction of the design requires successful RE. Blind reproduction is useless as its activation requires a matching trusted chip or passing PUF authentication of a remote manager. By receiving one or more DALs for testing, the manufacturer cannot activate additional IPs as the DAL changes from activation to activation.

### 4.4.5 Removal attacks

Removal of the TRNG fixates the DAL and breaks the LCC mode. In DCC mode, it gives an attacker control over the input to the AEAD, increasing the chances of SCA on the cryptography engine. NIST standard SP 800-90B [63] dictates that continuous health testing must be performed on the TRNG. These tests include repetition counting to detect catastrophic failure and adaptive proportion testing to detect loss of entropy. Removal of the TRNG would be detected as this would result in insufficient entropy to satisfy the health test, assuming the test is implemented on the trusted chip. Removal of COMA architectural modules makes the chip non-functional as COMA is not a wrapper architecture, but a fused one. Complete removal of COMA requires successful RE. Removing the PUF can be made challenging by using a strong PUF, with a large number of challenge-response pairs. Replacing such a PUF with a deterministic function is challenging as such functions are likely to have a substantially different area and power, making them detectable.

Table 4.1: Main features of the two proposed COMA variants.

| Feature | COMA1 | COMA2 |
|---|---|---|
| AEAD | AES-GCM | ACORN |
| PRNG | AES-CTR | Trivium |
| BUS Width | 8 | 8 |
| Pins used for Communication | 8 | 8 |
| CSN-RCSN Size | 64 | 64 |
| Trusted Memory | 4 Kbits | 4 Kbits |
| $C_{fix}$: initialization overhead (cycles) | 10,492 | 20,452 |
| $C_{byte}$: cycles needed for encrypting each byte | 72 | 17 |
| $PRNG_{perf}$: Throughput of generating PRN | $128bit/10cycles$ | $64bit/cycle$ |

## 4.5 COMA Implementation Results

For evaluation, all designs have been implemented in VHDL and synthesized for both FPGA and ASIC. For ASIC implementation we used Synopsys generic 32nm educational libraries. For FPGA verification, we targeted a small FPGA board, Digilent Nexys-4 DDR with Xilinx Artix-7 (XC7A100T-1CSG324).

### 4.5.1 COMA Area Overhead

We implemented two variants of COMA architecture: a NIST compliant solution (denoted by COMA1) and a lightweight solution (denoted by COMA2). The AEAD and PRNG in COMA1 is based on AES-GCM and AES-CTR respectively. The COMA2 is implemented by using ACORN for AEAD and Trivium for PRNG, The details of these two variants are summarized in Table 4.1. The breakdown of area (in terms of Slices, LUTs, and FFs) for these solutions for an FPGA implementation in Xilinx Artix-7 is reported in Table 4.2.

Table 4.2: Resource Utilization of the COMA Architecture.

| Name | AES-GCM+AES-CTR | | | ACORN+Trivium | | |
|---|---|---|---|---|---|---|
| | Slice | LUT | FF | Slice | LUT | FF |
| TRUSTED | | | | | | |
| AEAD_EXT | 1,336 | 3,804 | 4,432 | 333 | 1,067 | 591 |
| RNG | 712 | 2,226 | 618 | 215 | 601 | 450 |
| CSN | 257 | 540 | 739 | 257 | 540 | 739 |
| Others | 149 | 345 | 144 | 149 | 345 | 144 |
| UNTRUSTED | | | | | | |
| AEAD_EXT | 1,336 | 3,804 | 4,432 | 333 | 1,067 | 591 |
| RNG | 738 | 2,352 | 628 | 241 | 683 | 460 |
| RCSN | 252 | 607 | 737 | 252 | 607 | 737 |
| ECC | 563 | 1569 | 1161 | 563 | 1569 | 1161 |
| PUF [80] | 177 | — | — | 177 | — | — |
| Others | 209 | 359 | 257 | 209 | 359 | 257 |

*On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.*

The breakdown of area (in terms of Cells and $um^2$), critical path, and power consumption for an ASIC implementation is reported in Table 4.3. Note that the 2.5D-COMA needs both the trusted and untrusted parts of the architecture, while the R-COMA only requires the untrusted part. Table 4.4 reports optimized area and frequency results on FPGA for top-level of trusted and untrusted sides. As illustrated, the total area of lightweight solution is around 1/3 of the NIST-compliant solution. The reported numbers in Table 4.2 include the overhead of all sub-modules including AEAD, CSN-RCSN, RNG, ECC, etc. Due to the optimization on the boundaries among the units, resource utilization in Tables 4.4 is less than the sum of row values in Table 4.2.

### 4.5.2 COMA Performance

Fig. 4.7 compares the performance of two solutions in DCC and LCC mode. As illustrated, for small data sizes, the COMA1 outperforms the COMA2 solution. However, as the size of data increases, the COMA2 outperforms the COMA1 solution. It is due to the fact that stream ciphers such as ACORN have a long initialization phase, making them inefficient for

Table 4.3: Resource Utilization for ASIC implementation of COMA.

| Name | AES-GCM+AES-CTR | | | | ACORN+Trivium | | | |
|---|---|---|---|---|---|---|---|---|
| | Cells | Area$_{um^2}$ | Tclk$_{ns}$ | Power$_{mW}$ | Cells | Area$_{um^2}$ | Tclk$_{ns}$ | Power$_{mW}$ |
| **COMA** | 25338 | 0.11 | 1.97 | 1.62 | 8681 | 0.046 | 1.18 | 0.84 |
| ▷ RNG | 5684 | 0.025 | 1.43 | 0.431 | 1267 | 0.007 | 0.27 | 0.144 |
| ▷ CSN/RCSN | 1749 | 0.008 | 0.08 | 0.11 | 1749 | 0.008 | 0.08 | 0.11 |
| ▷ AEAD | 13675 | 0.061 | 1.67 | 0.704 | 2257 | 0.013 | 0.97 | 0.251 |
| ▷ ECC | 3278 | 0.016 | 1.34 | 0.321 | 3278 | 0.016 | 1.34 | 0.321 |

*Using Synopsys generic 32nm libraries.*

Table 4.4: Optimized results of COMA Architecture.

| Name | AES-GCM+AES-CTR | | | | ACORN+Trivium | | | |
|---|---|---|---|---|---|---|---|---|
| | Slice | LUT | FF | Freq[MHz] | Slice | LUT | FF | Freq[MHz] |
| Trusted | 2,297 | 7,094 | 5,892 | 103 | 1,030 | 2,901 | 1,924 | 121 |
| Untrusted | 2,818 | 8,781 | 7,169 | 109 | 1451 | 4,182 | 3,156 | 120 |

*On Xilinx Artix-7 (XC7A100T-1CSG324) FPGA.*

small data size. In addition, our AES-GCM implementation benefits from an 8-bit datapath, but the ACORN is realized by a 1-bit serial implementation. The total latency in terms of the number of clock cycles for COMA1 and COMA2 implementations can be calculated using equation (4.1), in which the number of cycles for the initialization and finalization is fixed and is given in Table 4.1. The $C_{byte}$ is the number of cycles needed for encrypting each input message byte, which is 17 and 72 for COMA2 and COMA1, respectively. Hence, in spite of longer initialization, the COMA2 outperforms the COMA1 for message sizes larger than 128 Bytes.

$$T_{comm} = C_{fix} + Message_{size} \times C_{byte} \tag{4.1}$$

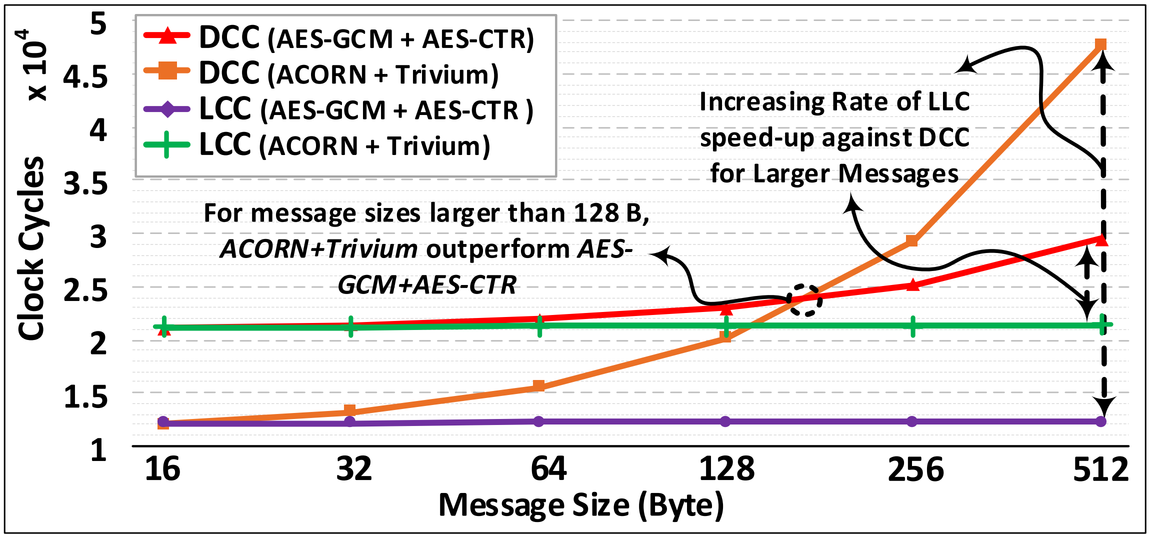


Figure 4.7: Total Execution Time for AES-GCM + AES-CTR and ACORN + Trivium.

### 4.5.3 COMA performance in LCC mode

In the LCC mode, the AEAD is used to synchronize the initial seed of the PRNG, while the CSN is used for encrypting data. The random (TRN) configuration key for the CSN-RCSN is generated by PRNG, which is updated after transferring every $U$ messages. In COMA, the PRNG has a limited buffer size, and as soon as the buffer is filled with random data, the PRNG stops producing additional bits. The consumption of TRNG output is synchronized (every $U$ messages) and the generation of random inputs is limited to the size of buffer. Hence, the PRNGs in the trusted and untrusted sides are always in sync. The number of cycles it takes to initialize the LCC mode includes the time to initialize the secret key engine ($C_{fix}$), the encryption and transfer and decryption of PRNG seed ($C_{ENC}$), and the time for the PRNG to generate enough output from a newly received TRN ($C_{PRNG}$):

$$C_{LCC-init} = C_{fix} + C_{ENC} + C_{PRNG} \tag{4.2}$$

Depending on the AEAD used for transferring the original seed, the $C_{fix}$ is obtained from Table 4.1. The seed size in our implementation is 16 Bytes, hence the $C_{ENC}$ is simply $C_{bytes} \times 16$, and the $C_{PRNG}$ is:

Table 4.5: SAT Execution Time on Blocking CSN and a Close to Non-blocking CSN .

| CSN Size | 4 | | 8 | | 16 | | 32 | | 64 | | 128 | | 256 | | 512 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mode | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk | blk | non-blk |
| SAT Iterations | 6 | 14 | 7 | 18 | 8 | 25 | 12 | 31 | 14 | TO | 24 | TO | 25 | TO | TO | TO |
| SAT Exe. Time ($s$) | 0.01 | 0.01 | 0.03 | 0.15 | 0.2 | 2.35 | 0.8 | 79.18 | 5.9 | TO | 130.5 | TO | 1136.2 | TO | TO | TO |

*TO: Timeout = $2 \times 10^6$ seconds; The SAT attack is carried on a Dell PowerEdge R620 equipped with Intel Xeon E5-2670 2.6 GHz and 64GB of RAM.*

$$C_{PRNG} = \frac{Bits_{needed}}{PRNG_{perf}} = \frac{3n \times (log_2 n - 1)}{PRNG_{perf}} \tag{4.3}$$

Finally, after initialization, and by using a CSN of size $n$ when the bus width of COMA is $BW$, the number of cycles to encrypt and transfer one byte of information is:

$$C^{LLC}_{byte} = \frac{8}{n} \times (\frac{n}{BW} + 1) \tag{4.4}$$

Using a 64-bit CSN and BW of 8 bits, the $C^{LLC}_{byte} = 9/8$. Compared to $C^{DCC}_{byte}$ for the COMA1 ($C^{DCC}_{byte}$ =72), and for the COMA2 ($C^{DCC}_{byte}$=17), the LCC mode is at least an order of magnitude faster. Fig. 4.7 compares the superior performance of LCC mode compare with DCC mode in both COMA variants.

### Frequency of TRN updates in LCC mode

The frequency of TRN update ($U$) for LCC is an important design feature. A large $U$ reduces energy as PRNG/TRNG is kept idle for $U - P$ cycles. P is the number of required cycles to refill the PRNG buffer after a TRN read. However, when the TRN is fixated for a long duration of time, the possibility of a successful side-channel, algebraic, or SAT attack on the CSN increases. The minimum number of messages required for an algebraic attack (even if such attack is possible) is $n$, which is the CSN input size. Our experiments show that a SAT attack could recover the key with an even smaller number of inputs. Knowing the number of encryptions/decryptions needed by such attacks, we can set the $U$ to a safe value smaller than the number of required messages to make it resistant against these attacks. So, the value of $U$ should be between $P \leq U \leq n$.

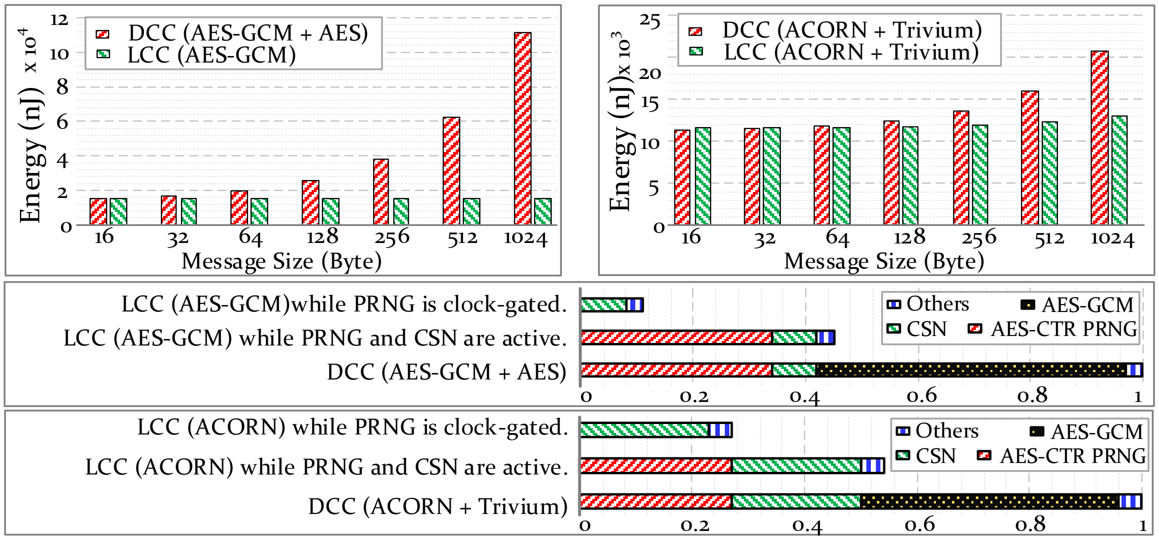


Figure 4.8: Energy Breakdown in COMA.

The SAT attack against CSN is implemented similar to [10]. In this attack the CSN gate-level netlist and an activated chip is available to the attacker, while the attacker aims to extract the CSN-RCSN configuration signals. Table 4.5 captures the results of the SAT attack against blocking and near non-blocking CSNs. As illustrated, the time to break a near non-blocking CSN is significantly larger. In each iteration SAT test one carefully selected input message. Hence, if the $U$ is kept smaller than the number of required SAT iterations, the SAT attack could not be completed.

### Energy saving in LCC mode

As illustrated in Fig. 4.9(a), in the LCC mode, the TRN is updated every $U$ cycles. $U$ is determined based on the fastest attack on CSN-RCSN pair, which is the SAT attack. After each TRN update, the PRNG takes $P$ cycles to refill its buffer. Note that $P$ cycles required for PRNG could be stacked at the beginning of $U$ cycles, or distributed over $U$ cycles depending on the size of PRNG buffer. As long as the TRN completely changes every $U$ cycle, the possibility of attack is eliminated. Hence in each $U$ cycles, for $P$ cycles the PRNG/TRNG and CSN are active, and for $U - P$ cycles, the PRNG is clock gated, and only CSN is active. In both cases, the AEAD is active only for the initial exchange of

Table 4.6: COMA vs. FORTIS.

| Scheme | Key Management | Data Comm | Private Key | SC Protected | Session Key | Activation | Need to TPM | RNG |
|---|---|---|---|---|---|---|---|---|
| FORTIS | Constant | ✗* | Embedded (known to the fab) | ✗ | Vulnerable to Fault Attack | Once | at Untrusted | PRNG |
| COMA | PUF-based Unique | ✓+ | No private key at untrusted | ✓ | Secure | per Demand | at Trusted | TRNG |

**: Not Implemented, but Naturally available using OTP. Limited Performance Due to Lightweight RSA*

*+: Available in Two Variant: DCC (Fully Secure and Limited Performance) and LCC (Leaky yet Secure and High Performance).*

Table 4.7: Area Overhead of COMA vs. FORTIS.

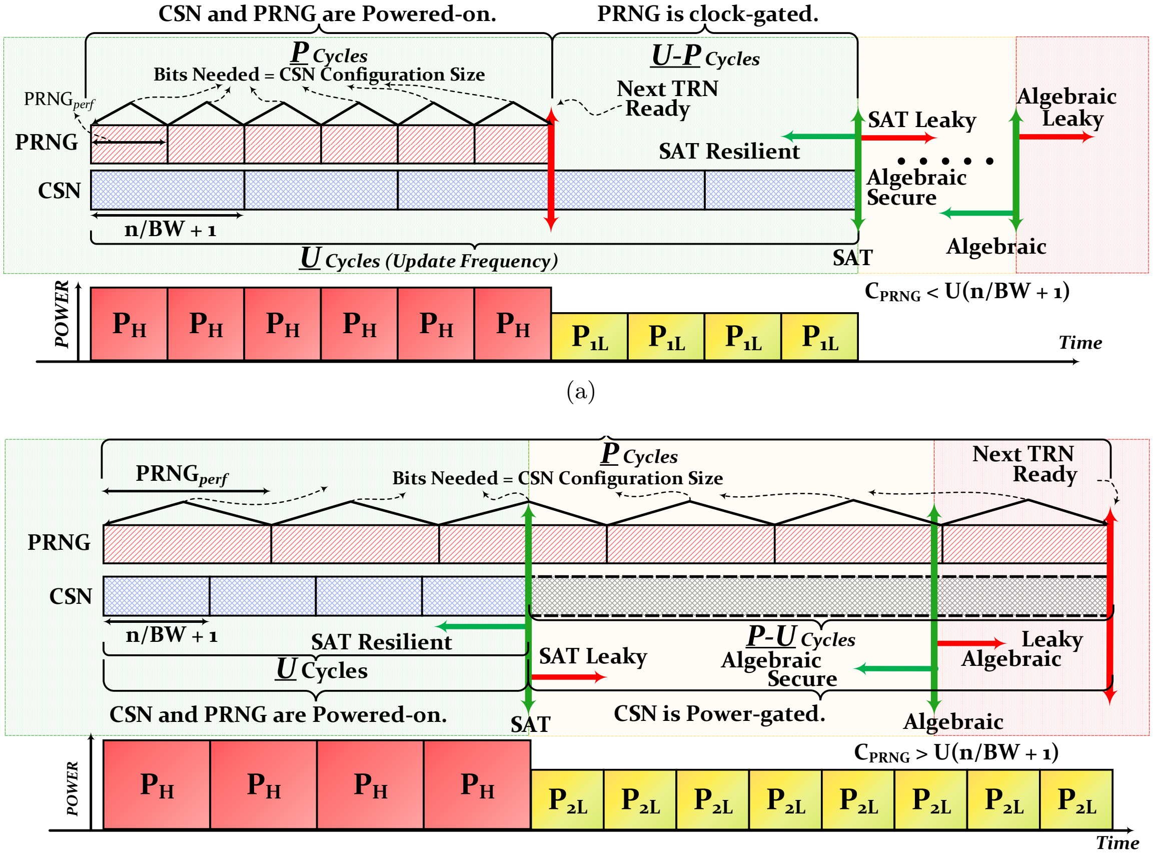
| Design | Gate Count | FORTIS/Design | COMA1/Design | COMA2/Design |
|---|---|---|---|---|
| b19 | 40,789 | 24.52% | 62.1% | 21.28% |
| VGA_LCD | 43,346 | 23.07% | 58.45% | 20.02% |
| Leon3MP | 253,050 | 3.95% | 10.01% | 3.43% |
| SPARC | 836,865 | 1.19% | 3.02% | 1.03% |
| Virtex-7 | 2M | 0.5% | 1.26% | 0.43% |

PRNG seed, allowing us to express the power consumption of the LCC mode as:

$$E_{LLC} = C_{PRNG} \times P_H + \big(U(\frac{n}{BW} + 1) - C_{PRNG}\big) \times P_L \tag{4.5}$$

Obviously, the number of required cycles to refill the PRNG buffer after TRN read ($P$) affects energy consumption and communication throughput. If $P < U$, as illustrated in Fig. 4.9(a), for $U - P$ cycles the PRNG is kept idle (power-gated). However, if $P > U$, as shown in Fig. 4.9(b), the communication should be stopped for $P - U$ cycles till the next TRN is ready and to resist SAT or algebraic attacks.

The energy consumption of LCC mode for COMA architectures constructed using NIST-compliant and lightweight solution when transmitting different size of messages is captured in Fig. 4.8. As illustrated, the LCC mode, for having to synchronize the two sides using a TRNG seed, is burdened with the initialization cost of AEAD. However, when the CSN-RCSN and PRNG are setup, the energy consumed for exchanging additional messages grow at a much lower rate compare to DCC mode (which is dominated by AEAD and PRNG power consumption as reported in table 4.3).



(a)

(b)

Figure 4.9: The Power Consumption at LCC mode: (a) While $P < U$, (b) While $P > U$.

## 4.6 Comparing COMA with Prior Work

To the best of our knowledge, FORTIS [62] is the only comprehensive key-management scheme that was previously proposed. Table 4.6 compares our proposed solution against FORTIS. COMA addresses several shortcomings of the FORTIS:

1) In FORTIS, all chips use identical keys, hence there is no mean of differentiating between chips. In COMA each chip has a unique key generated by PUF. 2) In COMA, secret key for communication and authentication is generated by PUF, when FORTIS relies on embedding the private key and public key in GDSII. So, the private key in FORTIS will be known to the fabrication posing the risk that the entire process of activation could be faked in software. In COMA, such attack is prevented as secret key is generated by PUF and is securely read out using public key cryptography. 3) In FORTIS, the usage of the private

key for chip authentication is vulnerable to SCA. In COMA, the secret-key cryptography is side channel protected, and the public-key encryption is only used once, making COMA secure against SCA. 4) In FORTIS, there is also the possibility of deploying a fault attack by fixing the value of session key Ks. In COMA, the same attack would require fixing the PUF output or replacing the PUF with a known function. This however could be tested by reading out the output of the PUF using multiple challenges and performing statistical test on the PUF response (PUF health check). 5) In FORTIS, the activation is done once, hence there is a need to store the obfuscation key in the untrusted chip. In COMA, the need to store the obfuscation key in untrusted chip is removed. In R-COMA, the activation takes place on demand, and the key is removed after power down or reset. In 2.5D-COMA, the activation key is stored in a trusted chip. 6) COMA provides two new mechanisms for communication: a) the DCC mode for added security, and b) the LCC mode for high-speed communication. 7) COMA uses a TRNG to produce the seed for PRNG, while FORTIS uses a PRNG without addressing a random source for its seed.

In terms of area overhead, FORTIS [62] provides an estimate for the incurred overhead of their solution, which is around 10K gates. As shown in Table 4.3, the numbers of cells for implementing the NIST-compliant (COMA1) implementation is 25.4K gates, while the lightweigh solution (COMA2) is implemented using 8.7K gates. Table 4.7 compares the area overhead of FORTIS against COMA1 and COMA2, when these architectures are deployed to protect a few mid- and large-size benchmarks. Using COMA2, which improves the overhead by 14% compared to FORTIS, requires between 0.43% and 21.3% of circuit area in selected benchmarks.

In this section we presented COMA, an architecture for obfuscation-key management and metered activation of an obfuscated IC that is manufactured in an untrusted foundry, while securing its communication. The proposed solution removes the need to store the key in the untrusted chip, makes the obfuscation unlock-key a moving target, allows unique identification of the protected IC, and secures the communication to/from the protected chip using two hybrid cryptographic schemes for ultra-high-speed and ultra-security. Our experimental results show that compared to the state-of-the-art key management architecture, FORTIS, COMA is able to reduce the area overhead by 14%, while addressing many of the shortcomings of the previous work.

# Bibliography

[1] A. Yeh, "Trends in the global ic design service market," *DIGITIMES research*, 2012.

[2] U. Guin, D. Forte, and M. Tehranipoor, "Anti-counterfeit techniques: from design to resign," in *Microprocessor Test and Verification (MTV), Int'l Workshop on*, 2013, pp. 89–94.

[3] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security." in *USENIX security symposium*, 2007, pp. 291–306.

[4] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proceedings of the Design Automation Conference (DAC)*, 1998, pp. 776–781.

[5] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation." in *USENIX Security Symposium*, 2013, pp. 495–510.

[6] R. P. Cocchi, L. W. Chow, J. P. Baukus, and B. J. Wang, "Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing," 2013, uS Patent 8,510,700.

[7] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[8] H. M. Kamali and S. Hessabi, "A fault tolerant parallelism approach for implementing high-throughput pipelined advanced encryption standard," *Journal of Circuits, Systems and Computers (JCSC)*, vol. 25, no. 09, p. 1650113, 2016.

[9] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes." in *NDSS*, 2015.

[10] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, 2015, pp. 137–143.

[11] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing ic piracy using reconfigurable logic barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, 2010.

[12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the Design Automation Conference (DAC)*, 2012, pp. 83–89.

[13] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on computers*, vol. 64, no. 2, pp. 410–424, 2015.

[14] T. Winograd, H. Salmani, H. Mahmoodi, K. Gaj, and H. Homayoun, "Hybrid stt-cmos designs for reverse-engineering prevention," in *Proceedings of the Design Automation Conference (DAC)*, 2016, p. 88.

[15] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 405–410.

[16] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, S. M. P. Dinakarrao, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *Proceeding of the International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[17] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, "On custom lut-based obfuscation," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI.* ACM, 2019, pp. 477–482.

[18] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, 2016, pp. 236–241.

[19] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2016, pp. 127–146.

[20] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-sat," in *Design Automation Conference (ASP-DAC), Asia and South Pacific*, 2017, pp. 342–347.

[21] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Appsat: Approximately deobfuscating integrated circuits," in *Hardware Oriented Security and Trust (HOST), IEEE Int'l Symposium on*, 2017, pp. 95–100.

[22] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 179–184.

[23] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant, "A general lower bound on the number of examples needed for learning," *Information and Computation*, vol. 82, no. 3, pp. 247–261, 1989.

[24] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks," in *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2017, pp. 189–210.

[25] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI.* ACM, 2019, pp. 471–476.

[26] S. Roshanisefat, H. Mardani Kamali, and A. Sasan, "Srclock: Sat-resistant cyclic logic locking for protecting the hardware," in *Proceedings of the Great Lakes Symposium on VLS (GLSVLSI)*, 2018, pp. 153–158.

[27] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proceedings of the Design Automation Conference (DAC)*, 2017, p. 9.

[28] P. Tuyls, G.-J. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2006, pp. 369–383.

[29] J. Guajardo, S. S. Kumar, G.-J. Schrijen , and P. Tuyls, "Physical Unclonable Functions and Public-Key Crypto for FPGA IP Protection," in *Int'l Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 189–195.

[30] D. S. Green, "Leveraging the commercial sector and providing differentiation through functional disaggregation," 2013, https://www.darpa.mil/attachments/DisaggregatetheCircuit_Slides.pdf.

[31] K. Z. Azar, F. Farahmand, H. M. Kamali, S. Roshanisefat, H. Homayoun, W. Diehl, K. Gaj, and A. Sasan, "{COMA}: Communication and obfuscation management architecture," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 181–195.

[32] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in ic piracy with test-aware logic locking," *IEEE Trans. on CAD (TCAD)*, vol. 34, no. 6, pp. 961–971, 2015.

[33] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, no. 1, pp. 1–1, 2017.

[34] H. M. Kamali, K. Z. Azar, and S. Hessabi, "Ducnoc: A high-throughput fpga-based noc simulator using dual-clock lightweight router micro-architecture," *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 208–221, 2017.

[35] Y. Shen *et al.*, "Sat-based bit-flipping attack on logic encryptions," in *DATE*, 2018, pp. 629–632.

[36] M. Li *et al.*, "Provably secure camouflaging strategy for ic protection," *IEEE Trans. on CAD (TCAD)*, 2017.

[37] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1601–1618.

[38] M. Yasin *et al.*, "What to lock?: Functional and parametric locking," in *GLSVLSI*, 2017, pp. 351–356.

[39] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *arXiv preprint arXiv:1811.12088*, 2018.

[40] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating sat-unresolvable circuits," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2017, pp. 173–178.

[41] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *Proceedings of the Int'l Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.

[42] Y. Shen *et al.*, "Besat: behavioral sat-based attack on cyclic logic encryption," in *ASP-DAC*. ACM, 2019, pp. 657–662.

[43] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR Trans. on CHES (TCHES)*, pp. 97–122, 2019.

[44] A. Chakraborty *et al.*, "Timingsat: timing profile embedded sat attack," in *ICCAD*, 2018, p. 6.

[45] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *DAC*, 2019, p. 6.

[46] K. Shamsi *et al.*, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *GLSVLSI*, 2018, pp. 147–152.

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

[48] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *IEEE Test Conference*, 2008, pp. 1–10.

[49] G. Nelson and D. C. Oppen, "Fast decision procedures based on congruence closure," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 356–364, 1980.

[50] R. E. Shostak, "Deciding combinations of theories," in *Int'l Conference on Automated Deduction*, 1982, pp. 209–222.

[51] C. Tinelli and C. Ringeissen, "Unions of non-disjoint theories and combinations of satisfiability procedures," *Theoretical Computer Science*, vol. 290, no. 1, pp. 291–353, 2003.

[52] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*, 2018, pp. 305–343.

[53] H. M. Kamali and S. Hessabi, "Adapnoc: A fast and flexible fpga-based noc simulator," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–8.

[54] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu, "Sat modulo monotonic theories." in *AAAI*, 2015, pp. 3702–3709.

[55] H. M. Kamali and A. Sasan, "Much-swift: A high-throughput multi-core hw/sw co-design k-means clustering architecture."

[56] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan, "Benchmarking the capabilities and limitations of sat solvers in defeating obfuscation schemes," in *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 275–280.

[57] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *IEEE Int'l Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 2014, pp. 49–54.

[58] R. S. 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.

[59] J. B. Wendt and M. Potkonjak, "Hardware Obfuscation using PUF-based Logic," in *IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 270–271.

[60] F. Koushanfar, "Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, 2012.

[61] G. Contreras, Md. T. Rahman, and M. Tehranipoor, "Secure Split-Test for Preventing IC Piracy by Untrusted Foundry and Assembly," in *IEEE Int'l Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013, pp. 196–203.

[62] U. Guin, Q. Shi, D. Forte, and M. M. Tehranipoor, "FORTIS: A Comprehensive Solution for Establishing Forward Trust for Protecting IPs and ICs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 4, p. 63, 2016.

[63] E. Barker and J. Kelsey, "Recommendation for the Entropy Sources used for Random Bit Generation," *Draft NIST Special Publication*, pp. 800–900, 2012.

[64] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi, "A Testing Methodology for Side-Channel Resistance Validation," in *NIST Non-Invasive Attack Testing Workshop*, vol. 7, 2011, pp. 115–136.

[65] H. Ahmadi and W. E. Denzel, "A Survey of Modern High-Performance Switching Techniques," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1091–1103, 1989.

[66] D.-J. Shyy and C.-T. Lea, "Log/sub 2/(N, m, p) Strictly Nonblocking Networks," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1502–1510, 1991.

[67] M. J. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," Tech. Rep., 2007, nIST Technical Report.

[68] H. Wu, "ACORN: A Lightweight Authenticated Cipher (v3)," *Candidate for the CAESAR Competition*, 2016, https://competitions.cr.yp.to/round3/acornv3.pdf.

[69] W. Diehl, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Face-Off between the CAESAR Lightweight Finalists: ACORN vs. Ascon," in *Int'l Conference on Field Programmable Technology (ICFPT)*, 2018.

[70] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, "CAESAR Hardware API," *Cryptology ePrint Archive, Report 2016/626*, p. 669, 2016.

[71] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations against Side-Channel Attacks and Glitches," in *Int'l Conference on Information and Communications Security*, 2006, pp. 529–545.

[72] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, "A Survey of AIS-20/31 Compliant TRNG Cores Suitable for FPGA Devices," in *Int'l Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[73] C. De Canniere and P. Bart, "Trivium Specifications," in *eSTREAM, ECRYPT Stream Cipher Project*, 2005.

[74] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *IEEE Symposium on Security and Privacy (SP)*, 2007, pp. 296–310.

[75] J. Delvaux and I. Verbauwhede, "Attacking PUF-based Pattern Matching Key Generators via Helper Data Manipulation," in *Cryptographers' Track at the RSA Conference*, 2014, pp. 106–131.

[76] W. Diehl, A. Abdulgadir, F. Farahmand, J.-P. Kaps, and K. Gaj, "Comparison of Cost of Protection against Differential Power Analysis of Selected Authenticated Ciphers," *Cryptography*, vol. 2, no. 3, p. 26, 2018.

[77] A. Vakil, H. Homayoun, and A. Sasan, "IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 152–159.

[78] L. Bhamidipati, B. Gunna, H. Homayoun, and A. Sasan, "A Power Delivery Network and Cell Placement Aware IR-Drop Mitigation Technique: Harvesting Unused Timing Slacks to Schedule Useful Skews," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017, pp. 272–277.

[79] B. Gunna, L. Bhamidipati, H. Homayoun, and A. Sasan, "Spatial and Temporal Scheduling of Clock Arrival Times for IR Hot-Spot Mitigation, Eeformulation of Peak Current Reduction," in *IEEE/ACM Int'l Symposium on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[80] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "Implementation of Double Arbiter PUF and its Performance Evaluation on FPGA," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015, pp. 6–7.