ECCLOCK: LOGIC LOCKING USING ELLIPTIC CURVE CRYPTOGRAPHY

BY

ALIYU ABUBAKAR HABIB

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

DECEMBER 2023

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **ALIYU ABUBAKAR HABIB** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies. in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

December 30th, 2023

Dr. Ali Saleh Alsuwayyan (Adviser)

Prof. Aiman Helmi El-maleh (Member)

Dr. Ashraf Sharif Hassan Mahmoud (Member)

Prof. Aiman Helmi El-maleh
Department Chairman

Prof. Suliman Saleh Al-Homidan
Dean of Graduate Studies

Date

This thesis is dedicated to my mother for her support and prayers throughout the journey of this great achievement.

ACKNOWLEDGMENTS

I would like to acknowledge and extend my gratitude to my siblings who have supported, cheered me up, and prayed for me during this thesis journey. I would also like to acknowledge and thank my supervisor Dr. Ali Al-Suwayyan for the ideas, support, and decisive inputs given to me to see that this work comes to reality. My sincere thanks go to my thesis committee members for their input and insightful suggestions, which have made this research more qualitative.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: ALIYU ABUBAKAR HABIB

TITLE OF STUDY: ECCLock: LOGIC LOCKING USING ELLIPTIC CURVE CRYPTOGRAPHY

MAJOR FIELD: COMPUTER ENGINEERING

DATE OF DEGREE: December 2023

Outsourcing integrated circuit (IC) fabrication to offshore foundries creates room for intellectual property (IP) piracy by a third party. This third party could be an untrusted foundry or malicious agent within the IC life cycle. This piracy is mostly performed to gain illegal economic benefits. This study aims to develop a new logic locking technique that utilizes Elliptic Curve Cryptography (ECC) to guard against IP piracy or make it at least uneconomical to the intended party. Dubbed ECCLock (Logic Locking using ECC), the circuit is locked by encrypting its original output, and this encrypted version is synthesized together with a decryption circuit to create the ECCLock version of the original circuit. The ECCLock circuit includes an additional key input used for the decryption of the encrypted original circuit output. For large circuits where obtaining the circuit's programmable logic array (PLA) representation is impractical, a cone

affecting most of the circuits's primary outputs is encrypted. With a minimum key size of 26 bits, ECCLock demonstrates immunity against conventional SAT attacks, which failed to recover the correct key within a 5-day (432,000 seconds). While the method incurs considerable hardware overhead, the delay and power overheads are within acceptable ranges. Our technique offers a resilient defense against the most prevalent attacks on combinational logic circuits, especially the SAT attack. These trade-offs reflect the method's emphasis on high-stakes security applications where safeguarding IP supersedes resource constraints.

ملخص الرسالة

الاسم: علي حبيب أبوبكر

عنوان الدراسة: عنوان الرسالة

التخصص: هندسة الحساب الآلي

تاريخ الدرجة العلمية: ١٤٤٥ جمادي الأولى

إن الاستعانة بمصادر خارجية لتصنيع الدوائر المتكاملة (IC) للمسابك الخارجية يخلق مجالًا لقرصنة الملكية الفكرية (IP) من قبل طرف ثالث. يمكن أن يكون هذا الطرف الثالث مسبكًا غير موثوق به أو وكيلًا ضارًا خلال دورة حياة .IC يتم تنفيذ هذه القرصنة في الغالب للحصول على فوائد اقتصادية غير قانونية. تهدف هذه الدراسة إلى تطوير تقنية قفل منطقي جديدة تستخدم تشفير المنحنى الإهليلجي (ECC) للحماية من قرصنة IP أو جعلها غير اقتصادية على الأقل للطرف المقصود. يتم قفل الدائرة، التي يطلق عليها اسم ECCLock (القفل المنطقي باستخدام ،(ECC عن طريق تشفير مخرجاتها الأصلية، ويتم تصنيع هذه النسخة المشفرة مع دائرة فك التشفير لإنشاء نسخة ECCLock من الدائرة الأصلية. تشتمل دائرة ECCLock على مدخل مفتاح إضافي يستخدم لفك تشفير مخرجات الدائرة الأصلية المشفرة. بالنسبة للدوائر الكبيرة حيث يكون الحصول على تمثيل المصفوفة المنطقية القابلة للبرمجة (PLA) للدائرة غير عملي، يتم استخدام مخروط التي تؤثر على معظم المخرجات الأولية للدوائر تكون مشفرة. مع الحد الأدنى لحجم المفتاح وهو 26 بت، يُظهر ECCLock الحصانة ضد هجمات SAT التقليدية، والتي فشلت في استنتاج المفتاح الصحيح خلال 5 أيام (432000 ثانية). على الرغم من أن هذه الطريقة تتطلب حملًا كبيرًا للأجهزة، إلا أن التأخير والطاقة يقعان ضمن النطاقات المقبولة. توفر تقنيتنا دفاعًا مرنًا ضد الهجمات الأكثر انتشارًا على الدوائر المنطقية التوافقية، وخاصة هجوم .SAT تعكس هذه المقايضات تركيز الطريقة على التطبيقات الأمنية عالية المخاطر حيث تحل حماية الملكية الفكرية محل قيود الموارد.

CHAPTER 1

INTRODUCTION

The global market of integrated circuits is expected to grow to USD 427.02 billion in 2023 (from USD 389.31 billion in 2022). In a five-year forecast study, the global integrated circuits market is projected to rise to USD 596.77 billion in 2027 at a CAGR of 8.7 percent [1]. This information signifies the importance of the market and the severe negative effects that chip piracy and illegal overproduction can cause to this market, motivating the need for an unbreakable logic locking technique, and hence, the importance of this work.

Logic locking is a technique that prevents intellectual property (IP) and data piracy, reverse engineering, and overproduction by inserting extra logic gates to the original design and additional inputs (key inputs) apart from the primary inputs that drive a tamperproof memory. These extra/protection gates alter the correct functionality of the integrated circuit (IC), only upon the application of the correct key inputs should the circuit operate correctly. The goal of logic locking is to prevent IP overproduction and/or reverse engineering. The only root of trust in the design chain is the design house; all other entities are untrusted. An easy way to lock a circuit

is to use additional locking circuitry that is either intertwined with the original logic design or augmented to it in such a way that the circuit output gets corrupted upon the application of an incorrect key. The circuit's normal/desired behavior is restored when the correct key is applied. In general, logic locking techniques can be grouped into two types: those using one key (general key) to unlock all ICs of the same design and those using a separate key (individual key) for each IC of the same design. For the general key type, the first locking technique is the ending piracy of integrated circuits (EPIC). It inserts locking gates (XOR or XNOR, and inverter) in the original circuit (oracle) in a random manner, but the effect of one locking gate could cancel the other because of the randomness, resulting in the cancellation of the security of the overall circuit. To prevent this from occurring, a strategic placement algorithm is used to place such gates in the circuit.

Although the positions of such locking circuitry are important, their type also matters. Some types of this circuitry include XOR and XNOR as in strong logic locking SLL [2], AND/OR, look-up table as in stripped functionality logic locking (SFLL) [3], and multiplexers (MUXs). Circuit level IC protection (CLIP) [4] was proposed for the individual key type. Although these locking techniques use a large number of key bit sizes, they are not secure against conventional SAT attack.

Original chip designers face several challenges, such as advancements in technology, time-to-market, and cost. It is convenient for the original chip designers to go with the shrinking scale of the integrated circuit IC by outsourcing their design to foundries with state-of-the-art facilities, since these foundries are very big, they can produce as many ICs as needed in the market and as fast as possible. This decreases the

manufacturing cost, and hence, increases the profits of the original chip designers. The original chip designers, design the IC and outsource its fabrication to a fabrication foundry. The fabricated design is then tested in a testing foundry. After testing, the working IC is packaged in a packaging foundry, then the ICs are integrated by an integration foundry, and lastly, the design house gets back the ready-to-use IC. This is referred to as the IC supply chain or IC life cycle. These foundries involved in the supply chain are distributed around the globe.

However, this advantage comes with a disadvantage. The engagement of these third parties in the IC design chain poses a threat to the integrity of each entity. Many of these entities try to steal an IC design for their interests, for example, overproducing the IC and selling it in the market or reverse-engineering the IC to insert a trojan in the design to steal information or alter the functionality of a design.

To deal with these threats of IP and data piracy, reverse engineering, and overproduction, several design-for-trust (DfTr) countermeasures have been developed in different studies. These countermeasures are categorized as passive, active, or proactive. Passive countermeasures take action after an attack has occurred. Active countermeasures are used to detect an attack at run time, while proactive countermeasures prevent the attack from occurring. Passive and active include watermarking, fingerprinting, metering, camouflaging, and split manufacturing [5]. Logic locking is considered to be the proactive countermeasure to these threats/attacks and in this era, it is the leading countermeasure among other DfTr countermeasures. However, implementing logic locking can be challenging due to the need to balance security with performance [6].

1.1 Motivation And Contribution

The study by Patnaik et. al. [7], dubbed hide-and-seek, claims to have broken 14 different logic locking techniques with 100% accuracy. Apart from developing a logic locking technique that can thwart the hide-and-seek technique, what motivated us is to see if we can develop a locking technique that is reliable in terms of security with respect to the designer and the IP owner.

The main contributions of this study are as follows:

- Proposed a logic-locking technique utilizing the concept of asymmetric cryptography. To the best of our knowledge, our proposed method is the first logic-locking technique that uses ECC to lock either completely or a portion of a logic circuit.
- The proposed technique can thwart the combinational logic attacks in the literature section 2, such as hill climbing, sensitization, and more importantly, conventional SAT attack.
- Designing ECC point adder in hardware
- Reducing hardware computation in ElGamal Decryption Algorithm. The Elliptic curve (EC) scalar point multiplication which is part of the Elgamal EC decryption algorithm [8] is performed offline so that the hardware, latency, and power overheads of this computation are eliminated.

1.2 Background

Cryptology, the method of making a ciphertext from a plaintext and vice-versa can be classified into two types: symmetric cipher and public key cryptography (PKC). The former uses only one key for both encryption and decryption, whereas the latter uses two pairs of keys, one for decryption (the private key) and the other for encryption (the public key). Kerckhoffs' principle is a fundamental principle in cryptography, it states that a cipher "must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience" [9]. The idea behind this principle is to allow cryptographers to analyze the cipher extensively and in turn, make it more secure. The algorithm in our technique employs a similar method to the public key cryptography. Two keys are generated from an ECC key generation algorithm for the encryption and decryption. The necessary and sufficient condition of a secured cipher is that the effort required to break the cipher by the best-known attack should be equivalent to that of a brute-force attack and that the key size should be large enough for any exhaustive key search algorithm/attack to find the key within a reasonable amount of time [9]. The SAT attack is a well-known and effective algorithmic attack used to decrypt logic circuits, especially combinational logic circuits. The technique proposed in this study can guard a logic circuit against this attack.

1.2.1 Elliptic Curve Cryptography

ECC is an alternative to the PKC standard modular arithmetic. It finds extensive applications in digital signatures and key agreement security modules, and other cryp-

tographic operations [10]. The price to be paid is that mathematical operations are more complex on an elliptic curve. ECC uses fewer bits to provide an equivalent level of security as the standard algorithms, as pointed out in [11] and [8]. This is the advantage gained in using ECC over its counterparts (RSA and Diffie-Hellman algorithms). It is worth noting that ECC is not a specific type of cryptosystem like RSA; it is just another way to perform the math involved in public key cryptography. It is pointed out in [10] that ECC can be used for secure communications in various applications, including IoT devices, mobile devices, and web applications. An elliptic curve (EC) is a graph that is described by a polynomial equation referred to as the Weierstrass equation which has an order of three as shown in Equation 1.1 and Equation 1.2. An elliptic curve E over a field K (where $K = F_p$ or $K = F_{2^m}$) is defined by the following equation:

$$y^2 = x^3 + ax + b \tag{1.1}$$

for prime field and,

$$y^2 + xy = x^3 + ax^2 + b \tag{1.2}$$

for binary field.

where $a, b \in K$, x, and y are the coordinates of the primitive point.

The operation on any two points in the curve evaluates to another point on the curve. The operation on any two points in the curve evaluates to another point on the curve. For example, points P1 and P2, their sum P3 is a third point that also lies

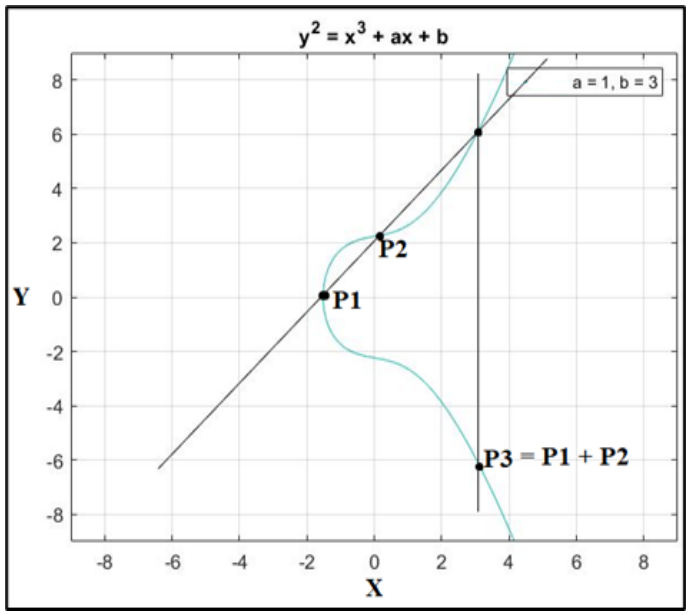

Figure 1.1: ECC geometric point addition over the curve $y^2 = x^3 + ax + b$, with a = 1 and b =3. P3 = P1 + P2.

on the curve. This sum P3, if to be found geometrically, a straight line joining P1 and P2 is drawn, it intersects another point on the curve, this point is then reflected about the x-axis to get P3 as illustrated in Figure 1.1. Operations on EC involve point addition, point scalar multiplication, point doubling, etc. At the core of ECC is point multiplication.

1.2.2 Threat Model

The standard logic-locking threat model is given below:

- Only the design house is trusted, but all other entities in the supply chain are not trusted.
- The functional chip that has the secret key and its reverse-engineered netlist are accessible to the attacker.

- The logic-locking algorithm used is known to the attacker.
- The only unknown to the attacker is the secret key.

This threat model is followed by many attacks including the well-known SAT attack [12]. In this logic locking method, we employ the ECC technique to introduce randomness in the original circuit, increasing the difficulty for an attacker to unlock it without the application of the correct key. The decryption unit is an ECC point addition that uses the right key to decrypt the encrypted output of the circuit or a cone.

CHAPTER 2

LITERATURE REVIEW

Since its emergence, the logic locking area has become a race between attacks and countermeasures. Whenever an attack is proposed, researchers attempt to develop countermeasures to that attack and vice versa. While some of these attacks and the proposed countermeasures are theoretical, others have proven to be practical. Next, a summary of the key attacks and countermeasures for combinational circuits are presented. Figure 2.1 gives the overview of the attacks and logic locking techniques discussed in this section.

Majorly, one specific countermeasure defends against one specific attack, and few countermeasures can thwart more than one attack. Our countermeasure, ECCLock, can thwart the attacks listed in this section. Combinational logic circuits face numerous attacks/threats from malicious entities seeking unauthorized access to intellectual property or sensitive data. These threats include attempts to reverse engineer the underlying netlist and instances of overproduction. These attacks can be grouped into invasive (by tampering with the circuit) and non-invasive, combinational and sequential, locked only-based vs locked and oracle-guided attacks, etc. An attack can fall

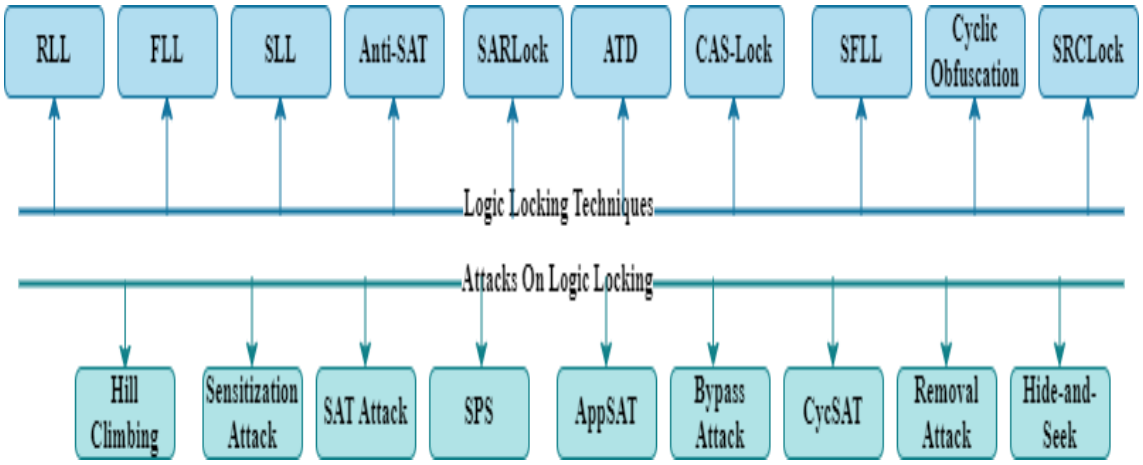

Figure 2.1: Logic Locking Techniques and Attacks

into one or more groups depending on the nature of the attack.

The journey of logic locking literally begins with random logic locking (RLL), also known as the ending piracy of integrated circuits (EPIC) [13]. It follows the standard threat model, in which only the design house is trusted in the IC supply chain, and all other entities are untrusted. RLL distorts the correct behavior of the logic circuit by randomly inserting XOR/XNOR gates into the circuit. For each candidate gate, an XOR gate is inserted at the output if the resulting key value is 0, and an XNOR gate is inserted otherwise. In order to avoid establishing a direct mapping between the key gate type and the key bit, inverters are inserted at the output of randomly selected key gates. The direct mapping means that whenever the key input fan-in to an XOR then by implication, the key value will be 0 and 1 otherwise. To break this mapping, inverters are inserted at the output of randomly selected key gates, and the corresponding key gate is changed (if it is an XOR it is changed to XNOR and vice verser). This locks the circuit, which can only be unlocked by applying the correct key. This key is programmed into a tamper-proof memory in a trusted setting. EPIC

permits remote unlocking of the fabricated IC in a foundry with the help of asymmetric key crypto. EPIC has low output corruptibility, and the random key gate effect may cancel each other when placed randomly. this leads to the development of fault-Analysis based logic locking (FLL) [14]: The FLL tends to solve this shortcoming by inserting these key gates at a location that has the highest effect on the output using the test principle of very-large-scale integration (VLSI) fault excitation, propagation, and masking. To determine the most influential gates for inserting the XOR/XNOR, the influence of a gate is given by $FaultI_G$ in equation 2.1 as stated in [14].

$$FaultI_G = (Pt_0 \times Out_0) + (Pt_1 \times Out_1) \tag{2.1}$$

Where Pt_x is the number of patterns that detect an s-a-x fault at the output of gate G, Out_x indicates the total number of output bits that are impacted by that specific s-a-x fault and $x \in (0, 1)$. Hill climbing [15] is an attack that targets RLL. The attack is non-invasive, and oracle-guided, that aims to find the key to a locked IC using a random guess. An arbitrary key input is selected at random and applied to the locked IC, the output of both the locked IC and the oracle are observed for the same primary input. The bits in the key inputs are adjusted based on the difference between the two outputs. The adjustment is made by flipping a random bit from the key input pattern [16]. This procedure of adjusting the key based on the hamming distance of the oracle and the functional IC is referred to as hill climbing. The attack succeeds when these outputs have a hamming distance of zero between them and therefore, the correct key input is found. The attack is affected by the initial random guess and

length of the key input [5]. This attack relies on random guesses, and the probability of choosing a pattern from the k bit key at random is 2^{-k}. As k increases, the probability decreases. some of the times the attack breaks RLL and sometimes it fails.

Sensitization attack [17] treats each key input as a stuck-at-fault. This stuck-at-fault value is propagated to the primary output and the resulting input patterns that propagate this value are obtained. This input pattern is then applied to the functional IC to get the correct key value. All other key values are obtained in this manner. Therefore, this attack is also oracle-guided. Remember that a sensitization attack targets the sensitization of individual key bits to primary outputs. This attack succeeded to break the security of RLL and FLL.

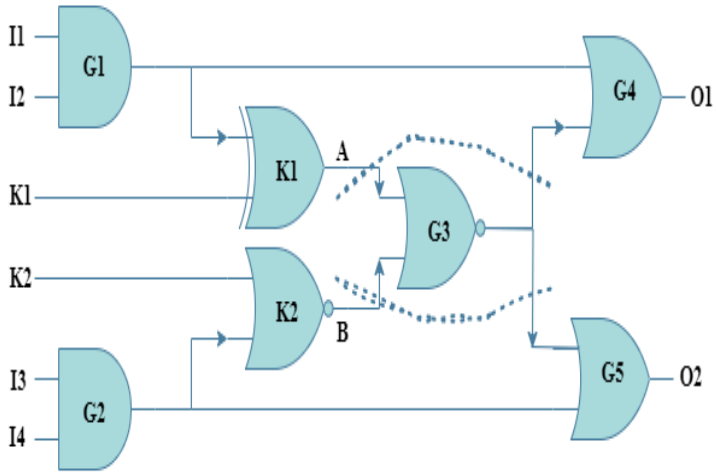

Figure 2.2: Key gate pairwise security. K1 is protecting K2, the value of the key K2/K1 cannot be sensitized to a primary output (either O1 or O2) unless the value of K1/K2 is known. I1 – I4 are primary inputs and G1 – G5 are logic gates.

The motivation for strong logic locking (SLL) [2] is to thwart sensitization attack. It creates key gate cliques that prevent the sensitization of an individual key value to

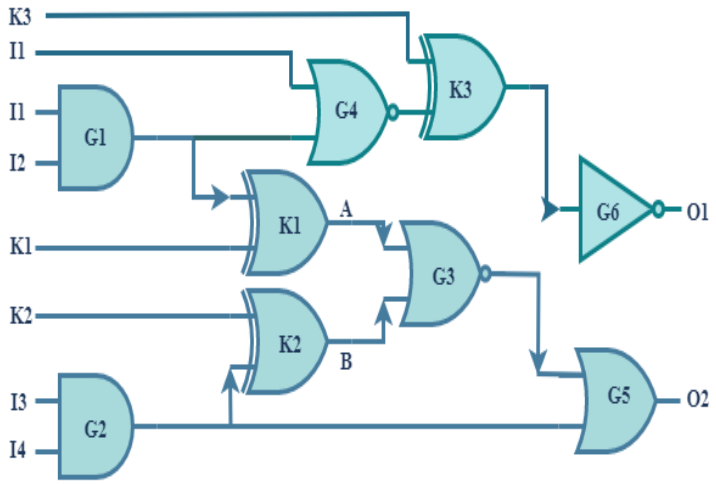

Figure 2.3: Locked circuit with 3 keys where K1 and K2 formed a clique while K3 is left unprotected.

one of the primary outputs. A clique consists of key gates that are pairwise secure with each other, and each key in the group protects its clique member. As shown in Figure 2.2, K1 and K2 are pairwise secure because the value of K2 cannot be sensitized to a primary output (either O1 or O2) unless the value of K1 is known, this subjected the attacker to an exhaustive search to find the keys.

Figure 2.3 shows an example of a key gate clique, the key inputs K1 and K2 protect each other. However, K3 is not in the clique because setting the output of G4 to 1 can sensitize it to the output O1. The algorithm is that a random key gate location is selected from the circuit, and then a heuristic is invoked to find a list of feasible gate locations that can be pairwise secure with the gates in the previously selected clique. A pairwise security check is executed between each of the feasible gates and each of the gates in the clique. Any feasible gate that meets this requirement is added to the clique until the required key size is obtained. If the clique does not have

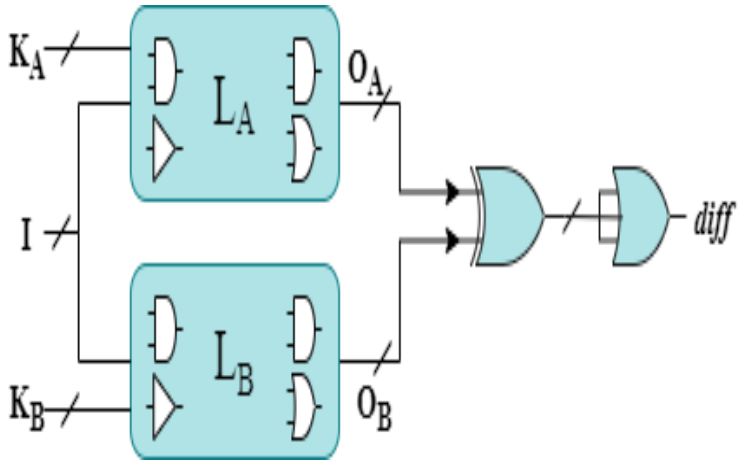

Figure 2.4: A miter circuit. L_A and L_B are the locked circuit with key K_A and K_B respectively The signal diff equals 1 only if the output (O_A and O_B) from both the circuits are differs for a given input pattern

sufficient candidate gates to meet the required key size, a new clique is formed. SLL is breakable by SAT attack [12]. The paper [2] suggested that the SLL method could be used together with one-way random functions (ORF), such as AES, to thwart SAT attack but this combination is still prone to removal attack (discussed 2).

Conventional-SAT attack [12] came up and broke RLL, FLL, and SLL. This attack is non-invasive, oracle-guided, and used mainly to attack combinational circuits. The core of this attack is the distinguishing input patterns (DIP). A DIP is an input pattern that separates possible and impossible keys from the pool of the key search space, it ensures that for any two or more different keys, the output of similar circuits differs.

A miter circuit, shown in Figure 2.4, is a circuit that is used to check whether any two logical circuits are equivalent. The miter circuit is used to find the DIPs used by the SAT attack. The circuits L_A and L_B are each a copy of the locked circuit with keys K_A and K_B respectively. The CNF of the miter circuit is passed to a SAT solver

with the constraint $diff = 1$. The solver returns I (the DIP) and the keys K_A and K_B if satisfiable. In this case, the outputs O_A and O_B are different. This DIP is applied to the functional IC and the corresponding output say O_D is recorded. The output O_D is then compared with O_A and O_B, at most one of the outputs will match O_D. The key (either K_A and K_B) that produces the correct output for this DIP is kept in the pool of feasible keys, and the other in the pool of infeasible keys. If none of the keys produces the correct output, they are both kept in the pool of infeasible keys. The correct input/output pair (I, O_D) is added as a constraint to the solver in the next iteration. This process is repeated until the solver returns unsatisfiable (UNSAT) that is no input pattern generates two different outputs and all wrong keys are eliminated. Then the SAT solver is invoked one last time to solve the CNF formula, but without the constraint on the diff signal (it is worth noting that the rest of the constraints for (I, O_D) pair remain). The resulting key (either K_A and K_B) is now the correct key. SAT attack is the most effective attack on combinational logic locking techniques and the most used to evaluate its security.

Approximate-based SAT attack (AppSAT) [18], as the name suggests, it is an approximate attack compared with the conventional SAT attack, which is an exact attack. AppSAT terminates much earlier than SAT and terminates when the output error rate falls below a certain threshold. It does not terminate only when all incorrect keys are pruned out, it terminates when an approximate key is found. This approximate key recovers the correct output for most of the input patterns. This attack makes d iterations and q queries to the oracle and then checks whether the condition is met. Experimental values suggested for the threshold, q, and d are $\frac{1}{2^k}$, 50 and 12

respectively [5], where k is the key size.

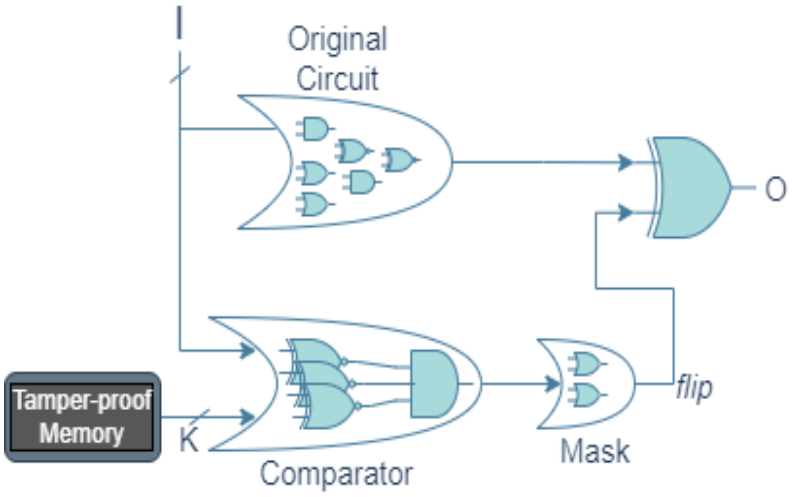

Figure 2.5: SARLock architecture. The protection circuitry, consisting of the comparator and the mask, is distinguishable from the original circuit.

The countermeasures SAT Attack Resistant Logic Locking (SARLock) [19], Anti-SAT [20], AND-tree detection (ATD), and cascaded locking (CAS-Lock) all use point function to guard against the SAT attack [12]. Any Boolean function that generates an output of 1 at a single point is said to be a point function, such as the AND gate. SARLock developed by Yasin M. et al. is one of the countermeasures that raises the bar for SAT attack. It uses a separate protection circuitry (a comparator and a mask circuit), as shown in Figure 2.5. The comparator is an AND gate coupled with XNOR gates at each of its inputs. Whenever the key input matches the primary input, the flip signal is asserted high and XORed with the primary output. The mask circuit nullifies the error generated by the comparator when the correct key is applied. The mask circuit consists of AND/NOR gates that are used to hardcode the correct key value. As the key size of SARLock increases linearly, its resistance to SAT attacks grows exponentially.

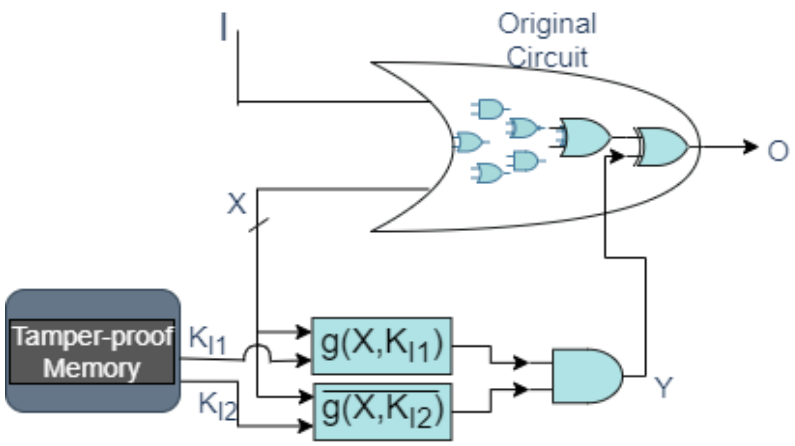

Figure 2.6: Anti-SAT architecture. g(X,K_{I1}) and $\overline{(g(X, K_{I2})}$ complementary. Y signal is asserted low when the correct key is applied thus the correct output is obtained.

Anti-SAT, unlike SARLock, avoids hardcoding the secret key into the circuit by using complementary functions (e.g., AND and NAND). The structure of this resilience countermeasure to SAT attack is depicted in Figure 2.6. K_{I1} and K_{I2} are the key inputs to g(X,K_{I1}) and $\overline{g(X, K_{I2})}$ blocks respectively. The upper and the lower blocks/functions g(X,K_{I1}) and $\overline{g(X, K_{I2})}$ outputs differ when the correct key is applied, thereby asserting the Y signal LOW. Hence, for all input patterns I applied, the correct output O is obtained. However, for any incorrect key, the output of both protection blocks becomes 1 for specific input patterns. For these patterns, the Y signal will be 1, thereby injecting an error into the circuit. The attack in [21] is able to break the security of Anti-SAT.

Signal Probability Skew (SPS) attack aims to break the security provided by Anti-SAT. The output gate of Anti-SAT protection circuitry is identified based on the principle of signal probability skew. The signal at the output of this gate is responsible for protecting the primary output, and the entire protection is removed by isolating this signal from the rest of the circuit. The SPS of a signal is a measure of the extent

to which the signal is distinguishable from a random guess, which is typically set to 0.5 [22]. One of the functions (AND) in the Anti-SAT block has a signal probability skewed towards -0.5, and 0.5 for the other function (NAND). The absolute difference between these values is close to 1. This property makes the output gate of the Anti-SAT block unique in the entire circuit. This signal is then isolated, and, consequently, the protection is nullified.

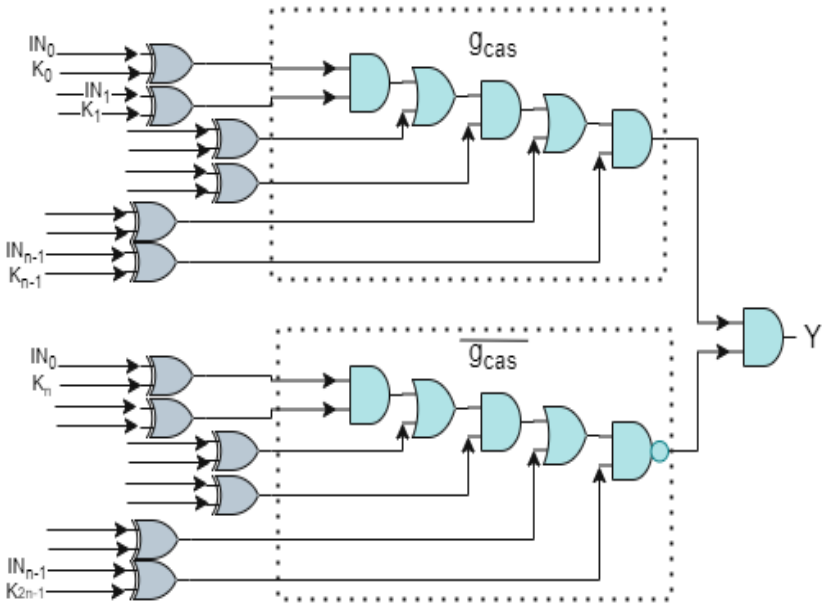

Figure 2.7: An instance of a CAS-Lock. IN are the primary inputs, K_{n-1} are the keys to the upper function (g_cas), K_{2n-1} are the keys to the lower function $\overline{(g_cas)}$ and Y is the output of these two functions combined.

AND-Tree Detection (ATD) [23] is also a derivative of Anti-SAT. However, instead of using an additional circuit, this technique identifies and utilizes these functions, AND-Trees, existing within the original netlist. For example, a structure like the one shown in Figure 2.8 is searched within the circuit and, if found is utilized for this purpose. This reduces at least the area overhead [5]. These AND-Trees are then locked by inserting XOR/XNOR in their inputs. The security of this technique

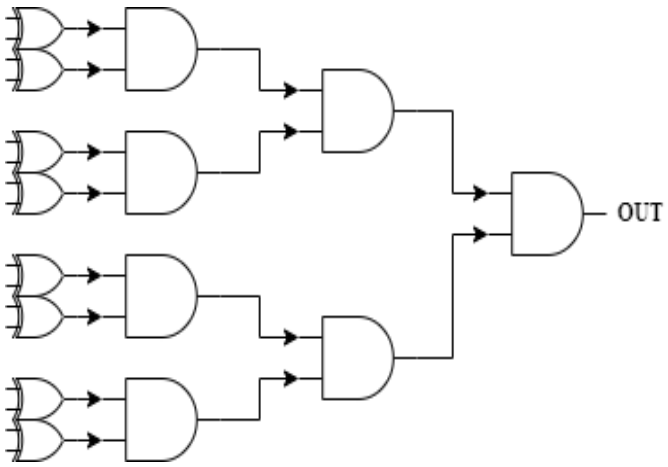

Figure 2.8: Locked AND-Tree with XOR at its input.

against the conventional SAT attack, similar to its counterparts SARLock and anti-SAT, grows exponentially with the key size. Cascaded Locking (CAS-Lock) [24] works similar to Anti-SAT. It comprises two complementary functions augmented to the original netlist. They differ in their ways of implementing these functions. Anti-SAT implements it in a tree-like manner whereas CAS-Lock implements it using a cascaded AND/OR, as shown in Figure 2.6. CAS-Lock increases the SAT attack complexity and protects logic circuits against bypass attack 2. This logic locking technique has been broken by [21].

Removal attacks target logic locking techniques with less structural obfuscation, that is those with a separate and distinguishable locking block from the original circuit, for example, the complementary functions in Anti-SAT and CAS-Lock, comparator, and mask in SARLock. AppSAT-Guided Removal (AGR) and Sensitization-Guided SAT (SGS) are removal attacks that target Anti-SAT and ATD, respectively [5]. The design flaw of these types of circuits allows a removal attack to identify the locking unit and eventually remove/isolate it from the circuit. One way to isolate the signal

is to ground it that is to make it to be always 0.

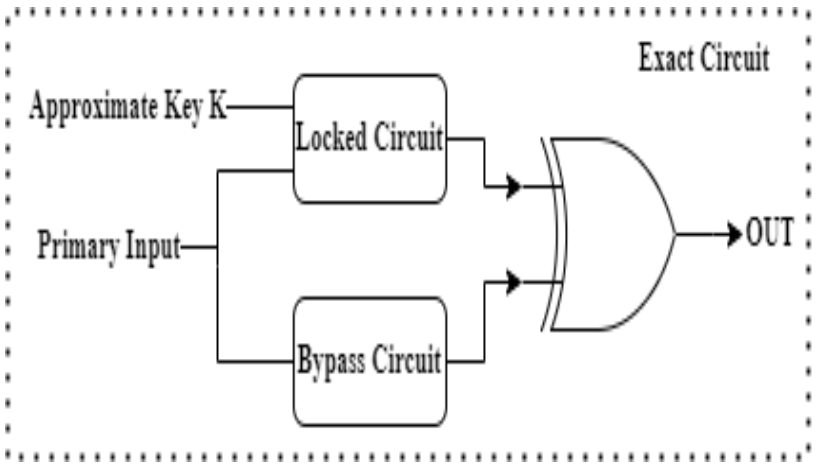

Figure 2.9: Bypass attack principle.

Bypass attack first launches a double-DIP attack and recovers the approximate circuit. Double-DIP is an attack that forms a miter circuit using four circuits [5]. Using this miter circuit a SAT solver is employed to generate all input patterns that make the circuit satisfiable. These inputs are then fed to the functional circuit and the outputs are recorded. Among these inputs, those that generate incorrect output are indicated as error-injecting inputs. Then a circuit is constructed that corrects the output of these error-injecting inputs. This is referred to as a bypass circuit. The bypass attack principle is illustrated in Figure 2.9 as explained above.

Cyclic Obfuscation [25] logic locking introduces logical cycles in the circuit that will make the SAT attack unresolvable, under the assumption that cycles are difficult to encode in conjunctive normal form (CNF). If a circuit is depicted as a directed acyclic graph (DAG), cycles can be generated by introducing reverse edges that counteract the typical flow of the circuit graph. This technique inserts key gates (usually AND/OR and multiplexers (MUX)) and dummy edges, that can be removed, into

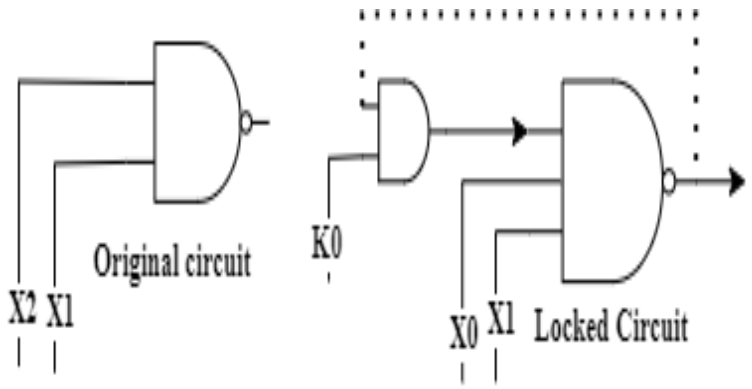

Figure 2.10: Using AND gate to introduce cycle in a circuit.

a circuit. When the right key is used, the dummy edges do not interfere with the operation of the circuit, and the circuit operates correctly. The dummy edges introduced into the circuit corrupt the behavior of the circuit when a wrong key is applied. Figure 2.10 illustrates how cycles can be introduced in the circuit using an AND gate. For example, for the input pattern 11 (that is when X0 and X1 are both 1) when the wrong key is applied, in this case 0, the output of the original circuit will be different than the output of the locked circuit. The SAT attack is perfect for directed acyclic circuits, meaning without any combinational loop or feedback in the circuit, but may or may not generate the correct keys for some cyclic encrypted circuits [26]. The SAT attack is indefinitely trapped in the loop. SAT-based attack on cyclic locking (CycSAT) [26] preprocesses the circuit by first checking the key conditions that may result in a circle in the netlist, which are then converted to a set of "No Cycle (NC)" constraints/clauses. These clauses are added to the circuit CNF formula, and then a SAT attack is invoked to recover the correct key. This attack can be thwarted if the SAT attack is thwarted. The pre-processing step of CycSAT involves detecting and avoiding cycles in the netlist before invoking the SAT solver. This is done by

generating cycle avoidance clauses that are added to the list of clauses that represent the circuit SAT problem. The pre-processing step is important for generating cycle avoidance clauses because a successful SAT attack on a cyclic circuit requires these clauses, and the time it takes to generate such clauses has an exponential relation with the number of inserted feedbacks.

SAT-Resistant Cyclic Logic Locking (SRCLock) [27] came up to prevent CycSAT. CycSAT adds extra conditions to a cyclic encrypted acyclic circuit that breaks the cycles in the circuit. SRCLock attempts to increase the time required to form these conditions and make it exponential with the number of loops in the circuit.

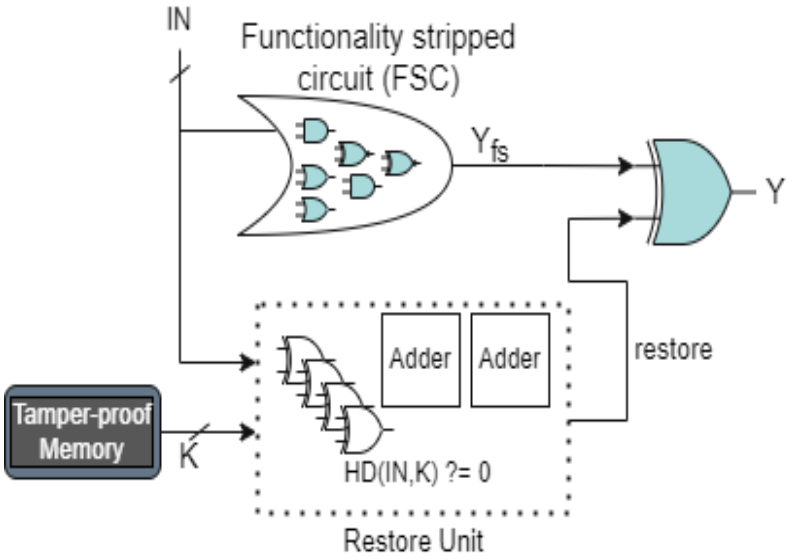

Figure 2.11: SFLL protection circuit, the restore unit is a hamming distance checker or basically a comparator that checks equality between the key and the input pattern. It cancels the error produced by the FSC for the correct key

Stripped functionality logic locking (SFLL) [3] is another logic locking technique that guards against the SAT attack. It injects a controllable error into a design by logic gate insertion or replacement. and the error is nullified by the restore unit when

the correct key is applied, the architecture of this technique is shown in Figure 2.11. This error is used to invert the design (circuit) output for $\binom{k}{h}$ selected (protected) input pattern. Thereby, protecting all input cubes that are of h hamming distance to the secret key. Where, k, h, and n are the key size, hamming distance, and input size respectively, and $k \leq n$. SFLL-HD countermeasure and its variants have been broken by structural and functional analysis attacks [28] and [29].

The study presented in [7], dubbed hide-and-seek, focuses on the security of provably secure logic locking (PSLL) techniques used to protect intellectual property in integrated circuits. The authors propose a generalized attack on hard-coded and non-hard-coded PSLLs. This attcak is capable of recovering the secret key across various PSLL techniques. It successfully breaks the security of fourteen logic locking techniques; SARLock [19], Anti-SAT [20], SFLL-HD^0 [30], SFLL-flex [3], SFLL-rem [31], CAS-Lock [24], error-controlled encryption (ECE) [32], Strong Anti-SAT (SAS) [33], Generalized (G-)Anti-SAT complementary and Non-complementary [34], Corrupt and Correct (CAC) [35], and Diversified Tree Logic (DTL) by Shamsi *et al.* [35] SARLock-DTL, Anti-SA-DTL, and CAC-DTL. In the case of hard-coded PSLL techniques like SFLL, the recovery of the secret key depends on the key-revealing logic gates. There might be logic cones within the locked circuit, and applying protected patterns as inputs to these cones could invert the output response of the oracle, leading to corruption. The output of these logic cones is termed key-revealing logic gates. For non-hard-coded PSLL techniques like the Anti-SAT, it first identifies the critical wire Y, the wire at the output of the protection circuitry, and then applies an algorithm to recover the value of each individual key.

CHAPTER 3

METHODOLOGY

3.1 Design Method

This section demonstrates how the ECCLock circuit is obtained from the original unencrypted circuit. Our technique employs ECC to logic lock combinational circuits. To the best of our knowledge, this is the first study in which an ECC-based locking technique is presented to logic-lock a combinational circuit. The ECC is well-known for its strength especially when its parameters are carefully chosen. This work consists of two stages: (1) locking the combinational logic circuit by encrypting its output, which is performed offline (in software) using the public key. (2) unlocking the encrypted output, performed online (in hardware while the circuit is in operation) with the private key stored in a tamperproof memory. Once the locked circuit is re-synthesized, the unlocking unit is attached to it as shown in Figure 3.1. The locked circuit takes the primary input and produces a locked output, which is then unlocked by the unlocking unit using the private key stored in the tamperproof memory. Silicon foundries will have no idea what the circuit does without knowledge of the private key, which is kept

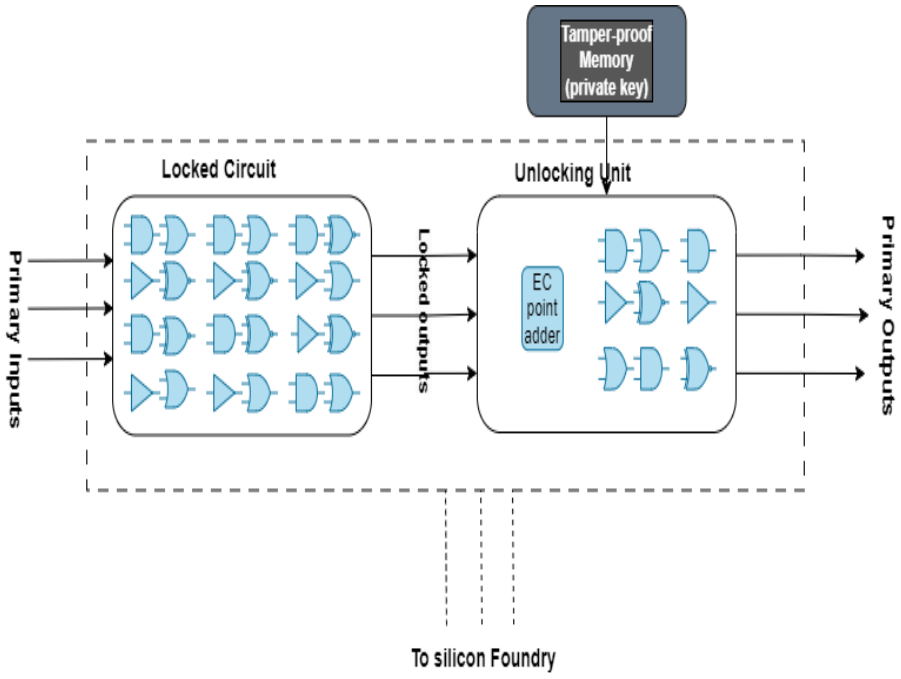

Figure 3.1: The overall logic-locked circuit block description. The original circuit can either be partially or completely locked. For whichever is used, the unlocking unit should recover the original functionality of the circuit.

secret and stored in tamperproof memory.

The flow for obtaining the ECCLock circuits starts with the BENCH file. The BENCH file is converted into a PLA and, the PLA is then converted into its equivalent truth table -Algorithm 1. If the circuit is large, such that the PLA cannot be obtained, a cone is extracted from the circuit for this purpose. The truth table output is encrypted using the Elgamal EC encryption Algorithm (Algorithm 3). The output equations of this encrypted truth table are then obtained in Verilog format. These Verilog equations are the original circuit in its encrypted form (LOCK). To obtain the ECCLock of the circuit, a decryption circuit (which is an EC point addition implementation in Verilog) is attached to the LOCK. Therefore, the decryption unit together with the LOCK makes up the ECCLock logic circuit. A flowchart of locking an oracle

(unencrypted logic circuit) from BENCH to ECCLock is shown in Figure 3.2. The oracle which is in bench format is obtained, if the circuit is small (circuit with less than or equal to 11 PIs), it is directly converted into a programmable logic array format, and from this format, it is easily transformed into truth table. For all the input of the truth table, the corresponding output binary string is encrypted using the ElGamal EC encryption algorithm. This results in a new truth table with the input patterns and their corresponding encrypted outputs. The output equations of this encrypted truth table are then obtained in Verilog format. This is the encrypted circuit (the LOCK). The encrypted circuit together with the decryption unit, which is an ECC point adder makes up the ECCLock circuit. If the network is large, a suitable cone (a cone that affects most of the output) is extracted for encryption. A cone is a subset of the circuit that shares a common output. This includes the direct inputs to the output gate, the gates feeding those inputs, and so on, cascading back until the primary inputs of the circuit are reached. It encompasses all gates and logic that contribute, directly or indirectly, to that specific output. The extracted cone is replaced with an ECCLock module into the rest of the circuit. The extraction of these cones is due to the prohibitively large truth table that will be generated from these circuits if they are to be encrypted as a whole since truth table size depends on the size of the PIs of a circuit. The *integrate* process between the LOCK and the decryption circuit feeds the output of the LOCK to one of the two inputs of the decryption circuit as shown in Figure 3.1. The *integrate* process between the remaining unencrypted cones of the oracle and the encrypted cone combines them into a single module.

The *ABC* tool [36] is used to obtain PLA, BENCH, BLIF and, Verilog format

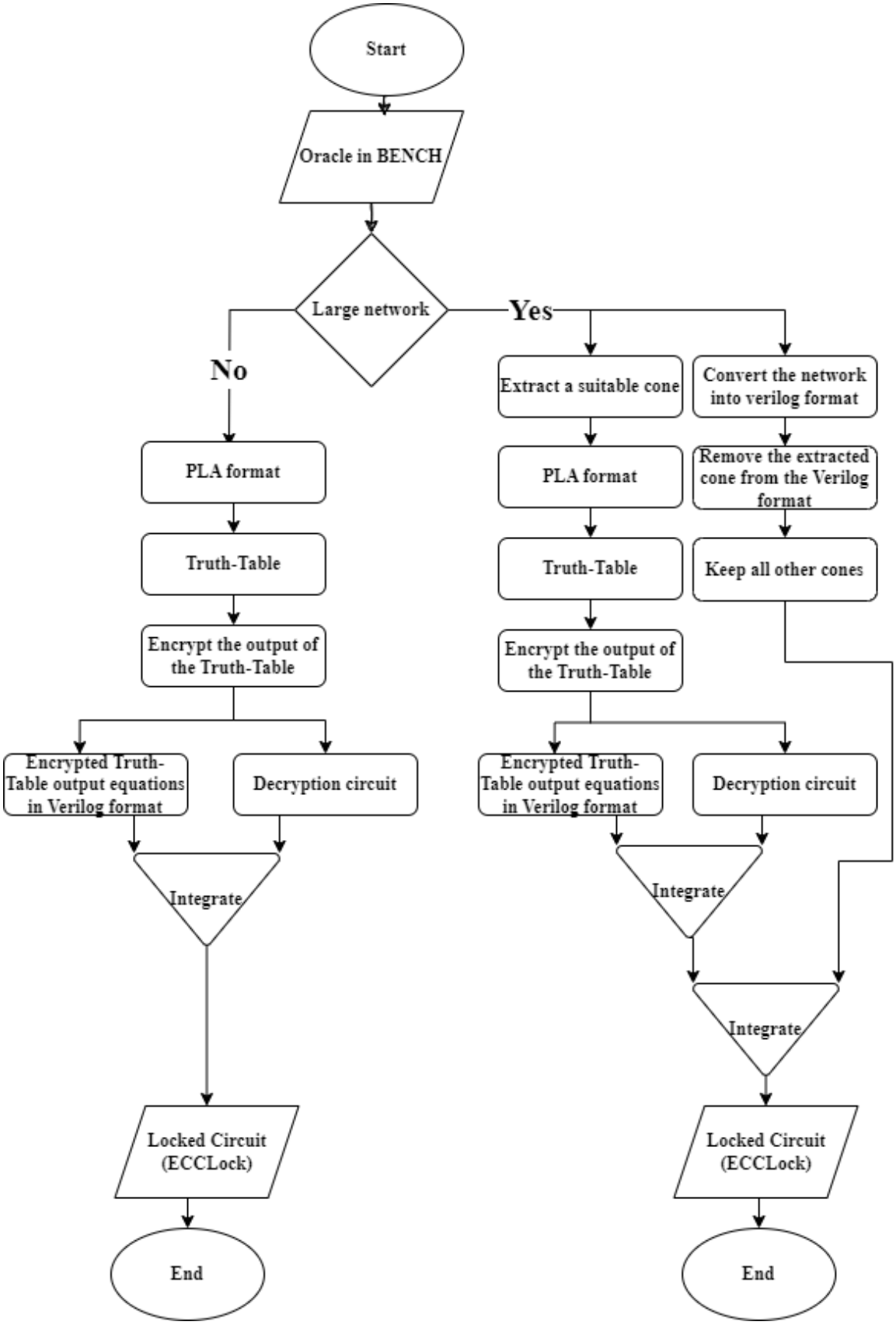

Figure 3.2: Locking a circuit from BENCH format to ECCLock in Verilog

of a logic circuit interchangeably. The tool *yosys* [37] is used in this work mainly to convert Verilog to BLIF in order to obtain the BENCH format of the ECCLock for security evaluation purposes. The algorithms for the ECCLock circuit are presented next.

Extracting a suitable cone from a large circuit using ABC:

- Extract the statistics of the number of affected primary outputs and the number of supports (primary input) for each cone in the circuit. It is convenient to use the BENCH format of the circuit for this purpose.

- To Extract the statistics of the number of affected primary outputs by each cone in the circuit, iterate through all the cones in the circuit. For each cone, propagate forward through other cones that depend on the given cone until all the primary outputs that depend on these cones are reached.

- To Extract the statistics of the number of supports for each cone in the circuit, iterate through all the cones in the circuit. For each cone, propagate backward through other cones that the cone depends on until all the primary inputs that drive these cones are reached.

- Determine a suitable cone for extraction, say *suitable_cone*. This cone should have sufficient support (nominal value of 10) and should affect a reasonable number of primary outputs of the circuit.

- Import the whole circuit into *ABC*; *read_bench aa.bench*, aa.bench is the name of the bench file

- Extract the suitable cone; *conesuitable_cone*
- To view the structure of this cone; show
- Collapse the extracted cone; collapse
- Write the PLA format of the extracted cone to a file; *write_pla aa.suitable_cone_pla*
- *aa.suitable_cone_pla* is now the extracted cone.

Generating domain parameters for binary extension fields in ECC is critical to ensure robustness and security. ECCLock uses the F_{2^m} field for its cryptographic computation. For this study, we utilized the *ecgen* tool [38] to generate random curves across varying bit sizes, enabling the selection of a more suitable curve in terms of the curve parameters a and b, the field order n, the base point coordinates x and y, and the cofactor h. For example, selecting a curve that has larger n will allow the encryption of more messages than a curve that has smaller n, and also message encrypted using the latter is more secure than the one encrypted using the former. The extension fields (m) used are 5, 11, 13, 19 and 29. All these extension fields are chosen to be prime and the selected curve generated from these extension fields has a cofactor of one. The selection of the curve domain parameters was complemented by adopting recommended irreducible polynomials (F) for each field, as described in [8]. This is done to strengthen the security of the ECCLock.

Algorithm 1 translates a Programmable Logic Array (PLA) representation of an original circuit, denoted as *oracle.pla*, into its corresponding truth table, saved as

oracle_truth_table.pla. The procedure involves reading the number of inputs, outputs, and implicants from *oracle.pla*. The algorithm initializes the truth table and iterates through all possible input combinations. For each input combination, it checks if the current implicant covers the input and computes the corresponding output. The algorithm then writes the input and output values to the truth table. This process is repeated for all input combinations, resulting in the generation of the truth table from the PLA representation of the circuit.

Algorithm 1 PLA to Truth Table

```
1: procedure PlaToTruthTable(oracle.pla)
2:     Inputs: PLA representation of the original circuit oracle.pla
3:     Outputs: Truth table of the oracle PLA oracle_truth_table.pla
4:     tt ← createFile(oracle_truth_table.pla)
5:     plaInN, plaOutN, inPla, outPla, implicants ←
   read from read (number of inputs, output and implicants)oracle.pla
6:     tt ← write(plaOutN, inPla, outPla)
7:     ttInput ← 2^plaInN
8:     for j in range(0, ttInput − 1) do
9:         OutputTruth ← 0          ▷ Initialize the truth table implicant output
10:        inputTruth ← Convert j to binary representation
11:        for i in range(first implicant, last implicant) do
12:            if the implicant[i] covers inputTruth then
13:                OutputTruth ← OR(OutputTruth, Implicant Output)
14:            end if
15:        end for
16:        tt ← write(inputTruth, outputTruth)
17:    end for
18:    return tt
19: end procedure
```

Algorithm 2 is used to generate the keys for encryption and decryption; these keys are used by the ElGamal encryption algorithm (see Algorithm 3) and the decryption circuit of the ECCLock, respectively. The point function in line 15 of Algorithm 2 creates a point for P -the base point-, scalar multiplication of the point is obtained

Algorithm 2 Key Pair Generation Algorithm

procedure KeyPairGeneration(m, x, y, F, a, b, n)
    Input:
        m - The extension field
        x - x-coordinate of the primitive point
        y - y-coordinate of the primitive point
        F - The irreducible polynomial
        a - Curve parameter
        b - Curve parameter
        n - Field order
    Output:
        Q - Public key
        N - Encryption key
        d - Private key
        $dC1Negative$ - Decryption key
    $P \leftarrow \text{point}(x, y)$
    Randomly choose d from the set $[1, n-1]$
    Randomly choose k from the set $[1, n-1]$
    $Q \leftarrow \text{scalarMultPoint}(P, d)$
    $C1 \leftarrow \text{scalarMultPoint}(P, k)$
    $N \leftarrow \text{scalarMultPoint}(Q, k)$
    $dC1 \leftarrow \text{scalarMultPoint}(C1, d)$
    $dC1Negative \leftarrow \text{negate}(dC1)$
    return $(Q, N, d, dC1Negative)$
end procedure

using the $scalarMultPoint$ function in lines 18, 19, and 20 of Algorithm 2, and the *negate* function returns the negative of a given point (line 22). The output of this algorithm is an integer d and three points Q, N, and $dC1Negative$ (or their coordinates). The Elgamal decryption algorithm subtracts the product $d \times C1$ from the ciphertext $C2$ and then extracts the message from the result of this subtraction. The fourth contribution of our study is to reduce the hardware computation in the Elgamal decryption Algorithm. To achieve this, the $dC1Negative$, which is the decryption key, is precomputed offline and it will be added to the ciphertext $C2$ to decrypt the ciphertext and consequently extract the original message. The pre-computation of - $d \times C1$, which is $dC1Negative$, offline enhances the performance of ECCLock in terms of hardware, latency, and power consumption and at the same time, the security of the system is increased. The performance is because the scalar point multiplication is performed offline (in software) and it is performed only once. The security of the system is enhanced by the fact that if d, the private key, is n bits then $dC1Negative$ will be $2 \times n$ bits (since $C1$ is a point consisting of x and y coordinates), x is n bits and y is n bits.

Algorithm 3 ElGamal Encryption Algorithm (modified)

```
1: procedure ElGamalEncryption(m)
2:     Input:
3:         m - The message and N - The encryption key
4:     Output:
5:         C2 - The ciphertext
6:     M ← messageToPoint(m)
7:     C2 ← addPoints(M, N)
8:     return C2(x, y)                    ▷ the x and y coordinates of C2
9: end procedure
```

The encryption circuit in our logic locking method employs Elgamal EC encryption

and decryption algorithms in the F_{2^m} field. Algorithm 3 depicts a modified version of the original basic Elgamal EC encryption algorithm, Algorithm 4 is the original Elgamal encryption algorithm. Line 6 of Algorithm 3 represents/converts the message to be encrypted into a point in F_{2^m} field for a given curve, to do so, Algorithm 5 is invoked. $C2$ is the ciphertext of the message and the *addPoints* function adds two EC points in F_{2^m} field, see Algorithm 6. The x and y coordinates of the ciphertext $C2$ are then returned as the encryption of the message m. The modification is the precomputation of N which is kQ. In the original Elgamal EC encryption algorithm, $C2$ is the addition of M and the product kQ. Instead of computing the scalar point multiplication for each message to be encrypted, it is done once and it is used to encrypt all messages. The reusing of N is valid because neither k nor Q is changing. Another modification to the ElGamal encryption algorithm is that in our modified algorithm only $C2$ is returned as the ciphertext while in the original algorithm, $C1$ and $C2$ are returned as the ciphertext. The part of the ciphertext $C1$ is only used to decrypt the message in the decryption algorithm so we compute it and use it only in the decryption algorithm, where $C1$ is the product of k and the base point P, and k is a random number between the field order and 1.

Algorithm 5 - Representing a Message As a Point M in F_{2^m} Field - takes a message as input along with parameters defining the extension field, curve, and field order to convert the message into a point on the given curve in the F_{2^m} field. The algorithm starts by setting the x-coordinate of the point as the input message, and padding it if necessary. Padding in line 13 of the algorithm refers to the process of adding extra random bits to the message before it undergoes encryption. This is done to distinguish

Algorithm 4 Original Elgamal Encryption Algorithm

procedure ElgamalEncryption(m, n, P, N)
    Inputs:
        m - The message
        n - The field order
        P - Public key point
        N - Encryption key point
    Outputs:
        $C1$ - Ciphertext point
        $C2$ - Ciphertext point
    $M \leftarrow$ messageToPoint(m) ▷ Convert message to point
    Randomly choose k from $[1, n-1]$
    $C1 \leftarrow$ scalarMultPoint(k, P) ▷ Compute first part of the ciphertext
    $C2 \leftarrow$ addPoints(M, N) ▷ Compute second part of the ciphertext
    return $(C1(x_1, y_1), C2(x_2, y_2))$ ▷ Return ciphertext coordinates
end procedure

between any two messages that are the same so that their ciphertext differs. If the message is zero, it sets the y-coordinate to 1 and checks if the resulting point lies on the curve. If successful, it returns the point. In the case the message is non-zero, it iterates through possible paddings, calculating the point coordinates based on the F_{2^m} field equations. This loop iteratively pads all possible combinations of binary patterns to the variable x one at a time, continuing until either the size of the extension field is reached or until x has a corresponding y-coordinate on the curve. The algorithm returns the resulting point if it satisfies the curve conditions.

Algorithm 6 provides a method to perform the addition of two points on the same curve; all operations are carried out in F_{2^m} finite field. The algorithm takes two points, $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, as inputs and computes their sum $P + Q$ on the elliptic curve F_{2^m}. The output of this algorithm is the result of the point addition. The algorithm first checks if Q is the point at infinity, in which case the sum is the point P. It then handles the cases where P is not equal to $\pm Q$, computing

Algorithm 5 Representing a Message As a Point M in F_{2^m} Field

procedure MessageToPoint(message, m, a, b, n, x, y, F)
    Input:
        message - The message
        m - Extension field
        a - Curve parameter
        b - Curve parameter
        n - Field order
        F - Irreducible polynomial
    Output:
        A point M in F_{2^m} Field
    $x \leftarrow$ message
    if length of the message $<$ 80% of the extension field size **then**
        Pad x from its MSB with 20% of the extension field size bits
    end if
    if $x == 0$ **then**
        $y = 1$
        if the point at (x, y) is on the curve **then**
            return point(x, y)
        end if
    end if
    for i in range 0 to $2^{(m-\mathrm{x})}$ **do**
        Pad x with i from its MSB
        Calculate $\beta = x + a + \frac{b}{x^2}$
        Solve for z from $z^2 + z = \beta$
        Solve for y from $y = z \cdot x$
        if the point at (x, y) is on the curve **then**
            return point(x, y)
        end if
    end for
    return (None)
end procedure

Algorithm 6 Point Addition in F_{2^m}

1: **procedure** POINTADDITION($P = (x_1, y_1) \in E(F_{2^m})$, $Q = (x_2, y_2) \in E(F_{2^m})$)
2: **Input:**
3: $P = (x_1, y_1)$ - A point in $E(F_{2^m})$
4: $Q = (x_2, y_2)$ - Another point in $E(F_{2^m})$
5: **Output:**
6: $P + Q$ - The result of point addition
7: **if** $Q = \infty$ **then**
8: $P + Q = Q + P = P$
9: **end if**
10: **if** $P \neq \pm Q$ **then**
11: $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$
12: $y_3 = \lambda \cdot (x_1 + x_3) + x_3 + y_1$, where $\lambda = \frac{y_1 + y_2}{x_1 + x_2}$
13: **end if**
14: **if** $P \neq -P$ **then**
15: $2P = (x_3, y_3)$, where $x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2}$ and $3y_3 = x_1^2 + \lambda x_3 + x_3$ where $\lambda = x_1 + \frac{y_1}{x_1}$
16: **end if**
17: **return** $P + Q$
18: **end procedure**

the coordinates x_3 and y_3 using the formula in line 11 and 12 respectively as stated in [8]. The point P added to its negative $-P$ results to the point at infinity,∞. To negate a point P for example where $P = (x_1, y_1) \in E(F_{2^m})$, $-P = (x_1, x_1 + y_1)$ and $(x_1, y_1) + (x_1, x_1 + y_1) = \infty$, and $-\infty = \infty$ where ∞ is the point at infinity. This algorithm is implemented in hardware using Verilog which is our third contribution. The program in the [39] repository is used with some modifications to perform these finite fields and elliptic curve point arithmetic offline.

Algorithm 7 highlight the process of obtaining the ECCLock circuit using the original circuit's Verilog description. It begins by creating a new file for the ECCLock module and extracting the primary inputs and outputs from the original circuit. The module is then populated with the output equations of the encrypted circuit, and a point addition operation is performed on the obtained ciphertext and decryption key.

Algorithm 7 Obtaining The ECCLock Circuit

1: **procedure** ObtainECCLock(encrypted_verilog, keyinput, oracle.v)
2: **Input:**
3: encrypted_verilog - Verilog representation of the encrypted truth table
4: keyinput - Decryption key
5: oracle.v - Verilog description of the original circuit
6: **Output:**
7: The ECCLock module
8: $ECCLock$ = createFile()
9: Obtain the primary inputs and outputs of the circuit from oracle.v: PI and PO
10: Create a module with ports(PI, keyinput, and PO) in the file $ECCLock$ created in line 8
11: Write the output equations of the encrypted circuit from encrypted_verilog into the module
12: Obtain the ciphertext from these equations: $C2$
13: O = addPoint($C2$, keyinput)
14: **if** encrypted_verilog is the encryption of the whole circuit **then**
15: $PO = O$
16: **else**
17: encrypted_cone = the output of the encrypted cone
18: Replace the cone by O obtained in line 13: encrypted_cone = O
19: Write the remaining cones of the original circuit to the ecc module
20: $PO = PO$
21: **end if**
22: **return** $ECCLock$
23: **end procedure**

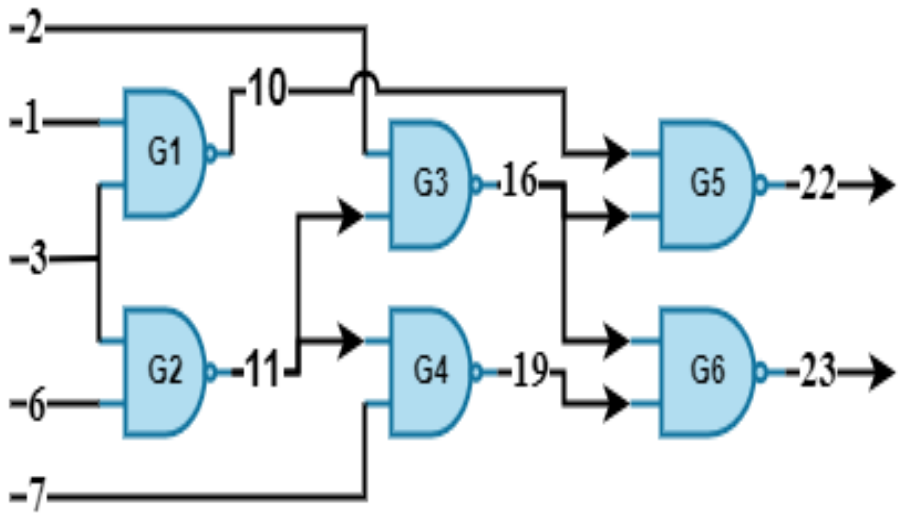

Figure 3.3: c17 ISCAS-85 benchmark circuit.

The result of the addition of $C2$ with the *keyinput* is used to extract the primary output of the circuit. If the circuit is encrypted as a whole, the primary outputs are the least significant bits of O (line 13) that correspond to the number of primary outputs. If a cone is extracted, then the last bit of O is the output of the cone and the primary outputs of the ECCLock are then the primary outputs of the original circuits.

The *addPoint* function in line 13 of Algorithm 7 is implemented in Verilog using Algorithm 6. The decryption unit is just an EC point adder. If a cone is encrypted, not the entire circuit, the *encrypted_verilog* is its Verilog description (line 3 of Algorithm 7). Algorithm 7 returns the locked oracle; therefore, it is the implementation of our logic locking technique, ECCLock.

Table 3.1: Cone 16 of c17 PLA representation

Input			Output
2	3	6	16
0	-	-	1
-	1	1	1

Table 3.2: Cone 16 of c17 truth table

Input			Output
2	3	6	16
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

3.1.1 An Example To Demonstrate Our Technique

We use c17 ISCAS-85 to illustrate our locking technique and to also illustrate how a suitable cone is selected. Since we want to illustrate the cone selection, we will follow the flowchart path that says "*YES*" in the ECCLock flowchart 3.2. Cone 16 which fan out to the output of gate G3 is selected as the suitable cone. Because the cone output impacted both the primary outputs of the circuit and it only has 3 supports viz 2, 3, and 6. The cone PLA is shown in Table 3.1. The selection and extraction of this cone is based on the procedure listed in A.2 section of the Appendix. The PLA is then unrolled into its equivalent truth table using Algorithm 1 and the resulting truth table is shown in Table 3.2.

For each of the values (message) in the truth table output, it is represented as a point using an extension field of 5. The resulting transformation is shown in Table 3.3. The x and y in the table represent the x and y coordinates of the point. As can be

Table 3.3: Representing each of the bits in the truth table output to a message using Algorithm 5

Output 16	x					y				
1	0	1	0	1	1	0	0	1	1	1
1	0	0	0	0	1	1	1	0	1	1
1	0	1	0	1	1	0	0	1	1	1
1	0	0	0	0	1	1	1	0	1	1
0	0	1	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	1	0	1
0	0	1	0	0	0	0	0	1	0	0
1	0	0	0	0	1	1	1	0	1	1

Table 3.4: Representing each of the bits in the truth table output to a message using Algorithm 5

Input											
2	3	6	out1	out2	out3	out4	out5	out6	out7	out8	out9	out10
0	0	0	0	0	0	1	1	1	1	1	0	1
0	0	1	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	1	1	0	1	0
0	1	1	0	1	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	1	1	1	0
1	0	1	0	0	0	1	0	1	0	0	1	1
1	1	0	1	0	0	0	0	0	1	1	1	0
1	1	1	0	1	0	0	0	0	0	0	1	0

seen from the table, for any two outputs that are the same, the represented message point returned may differ. For example, the first and the second rows have the same output but their represented message points differ, this is the effect of the padding used in Algorithm 5. The messages in Table 3.3 are encrypted using extension field of 5 with the curve parameters; m is 5, and the variables a, b, x, y, n, and F are defined as 10000, 11110, 19, 8, 36, and 100101, respectively. The parameter a, b, and F are in binary while m, x, y, and n are in decimal. Table 3.4 shows the input patterns of the truth table with their corresponding encrypted outputs. The equations for this encrypted output are then obtained as follows;

$$\text{out1} = (c_2\& \sim c_3\& \sim c_6)|(c_2\&c_3\& \sim c_6);$$

$$\text{out2} = (\sim c_2\& \sim c_3\&c_6)|(\sim c_2\&c_3\&c_6)|(c_2\&c_3\&c_6);$$

$$\text{out3} = 0;$$

$$\text{out4} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6)|(c_2\& \sim c_3\&c_6);$$

$$\text{out5} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6);$$

$$\text{out6} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6)|(c_2\& \sim c_3\&c_6);$$

$$\text{out7} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6)|(c_2\& \sim c_3\& \sim c_6)|(c_2\&c_3\& \sim c_6);$$

$$\text{out8} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6)|(c_2\& \sim c_3\& \sim c_6)|(c_2\&c_3\& \sim c_6);$$

$$\text{out9} = (\sim c_2\& \sim c_3\&c_6)|(\sim c_2\&c_3\&c_6)|(c_2\& \sim c_3\& \sim c_6)|(c_2\& \sim c_3\&c_6)|(c_2\&c_3\& \sim c_6)|(c_2\&c_3\&c_6)$$

$$\text{out10} = (\sim c_2\& \sim c_3\& \sim c_6)|(\sim c_2\&c_3\& \sim c_6)|(c_2\& \sim c_3\&c_6);$$

$$C2 = \{\text{out1, out2, out3, out4, out5, out6, out7, out8, out9, out10}\}.$$

The subscript in the output of equations for each c indicates the primary input of the circuit for example, c_2 indicates that 2 is the primary input. This is done to abide by the naming rule in Verilog. The variable $C2$ (the ciphertext) is responsible for reproducing all the encrypted messages in Table 3.4. To recover the original output of the extracted cone, $C2$ is added to the *key input* (as indicated in line 13 of Algorithm 7), and the least significant bit of the resulting addition is the recovered output. The output of this point addition will replace the extracted cone in the original circuit, line 18 of Algorithm 7.

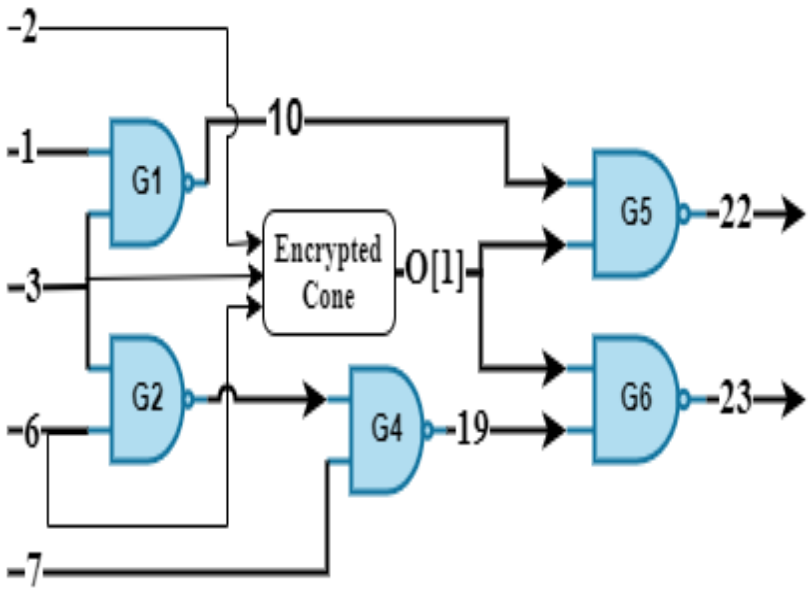

Figure 3.4: Illustration of the Second *integration* process in Flowchart 3.2

Figure 3.4 illustrates how the extracted cone is replaced with its encrypted version and how the remaining cones of the circuit are integrated into a single module. This example showcases how Flowchart 3.2 can be implemented using the various algorithms discussed in this chapter.

CHAPTER 4

RESULTS AND CONCLUSION

This section discusses the results of the study with respect to security, delay, hardware, and power. A total of 11 ISCAS-85 benchmark circuits [40] were considered for this purpose, and information about these benchmark circuits is shown in Table 4.1. The dash in the table entry indicates that the corresponding circuit is encrypted as a whole. The extracted cone column provides the name of the extracted cone in the circuit for encryption. The experiments are run on Intel(R) Xeon(R) CPU E5-2680 at 2.70GHz with 32 cores and 128 GB RAM server.

4.1 Result Method

For a given benchmark circuit, different curves with different curve parameters were used to generate the encrypted version of the circuit. Five different extension fields, all primed from 0 to 30, are selected. These m's are 5, 11, 13, 19 and 29. For extension field 5, c17 has a different domain parameter and thus has a different original key than the rest of the circuits. This is because the domain parameters for the other

Table 4.1: ISCAS-85 Benchmark information.

S/n	Name	PI	PO	PI + PO	Extracted Cone	Cone PI	Affected POs (%)
1	c17	5	2	7	-	-	100
2	c432	36	7	43	171	2	100
3	c499	41	32	73	350	10	100
4	c880	60	26	86	543	11	34.62
5	c1355	41	32	73	700	8	100
6	c1908	33	25	58	1741	8	100
7	c2670	233	140	373	1606	6	10.71
8	c3540	50	22	72	2723	11	77.27
9	c5315	178	128	306	6153	9	19.53
10	c6288	32	32	64	2230	8	53.13
11	c7552	207	108	315	8348	7	30.56

circuits have no suitable points to represent all the outputs of the c17 circuit. Before encrypting any output/message, at least 20% bits of the extension field are padded to the bits of the message. For example, if m is 5 then 20% of 5 is 1, so only one bit is padded to the bits of the original message. To invoke the SAT attack, the ECCLock circuit in Verilog is converted to BLIF using *yosys*, then to BENCH using *ABC*. The ECCLock and the oracle both in BENCH format are then used as inputs to the SAT attack tools. For each encrypted circuit, the SAT attack is invoked to decrypt it to find a key that renders the two circuits (the locked and oracle) functionally equivalent. The number of iterations and time spent (in seconds) extracting the correct key are recorded. We set the timeout time to be 4.32×10^5 s (5 days), this means that the SAT attack failed if it cannot find the correct key pattern in 5 days, we label this as "timeout". The timeout time is the time elapsed during the execution of the SAT attack exclusively on the CPU, excluding waiting time. The SAT attack is invoked five times for a particular circuit and extension field, and the average of the metrics (execution time and number of iterations) are then taken.

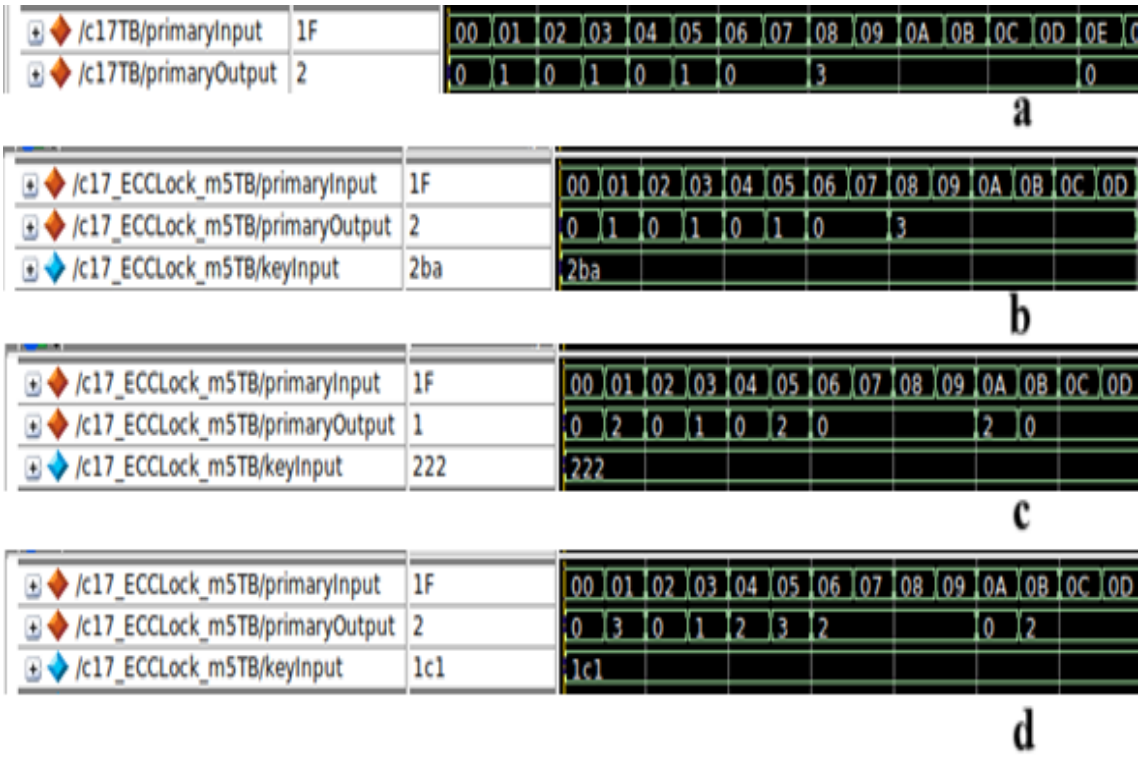

Figure 4.1: ECCLock verification using c17 benchmark circuit with extension field of 5

The ECCLock circuit and the oracle are compiled in Quartus Prime Pro Edition on 10CX220YF780I5G of the Cyclone 10 GX family. The latency of the circuit is extracted after compilation, the path with the longest delay from the PIs to the POs with a "slow 990mV 100C model". The estimated power of the circuit module reported by the Quartus power analyzer is extracted. The hardware, in terms of gate count, is extracted from the locked circuit benchmark representation. We compared our technique's security metrics with existing state-of-the-art encryption algorithms.

To verify the correctness of our locking technique, c17 which is a small circuit is simulated using the correct key and some incorrect keys. Extension field 5 is used for this purpose and the result is shown in Figure 4.1. All the key inputs shown in the figure are in hexadecimal. Figure 4.1a is the original circuit output, Figure 4.1b shows

the output of the locked circuit when the correct key (2ba) is applied, Figure 4.1c is the locked circuit when an incorrect key (222), and Figure 4.1d shows the output of the circuit when another wrong key is used. The figure shows that when the key used is not the correct key, the output of most of the input patterns differs from that of the original circuit.

4.1.1 Security

The SAT attack described in [12] is used. Which is developed and modified by [41] and [30]. If the SAT attack returns UNSAT, it means that the attack fails and may need to be retried several times. If it is invoked several times and still returns UNSAT, then the attack cannot find a satisfying assignment for the formula and therefore fails to find the decryption key. The SAT attack returned UNSAT for c432 for all the extension fields, whereas it returned UNSAT for c5315 for 19 and 29 extension fields. The ISCAS-85 circuit c5315 returns the key for extension fields 5 and 11 with a failed count of 1, meaning that the key returned by the SAT attack when used to decrypt the locked circuit cannot, in all cases, produce an output equivalent to the oracle for all input patterns. The timeout time is relative, it depends on the speed of the processor and the machine specifications from which the attack is run. Therefore SAT attack timeout time is not only the function of the attack procedures, Table 4.2 illustrates that. The dash lines in the table indicate that the key is not returned. c499 locked using an extension field of 11 is used to illustrate this. The SAT attack is launched on this circuit on a personal computer (PC) and a server with the following specification Intel(R) Celeron(R) J4125 CPU @ 2.00GHz with 4 cores and 4GB RAM and Intel(R)

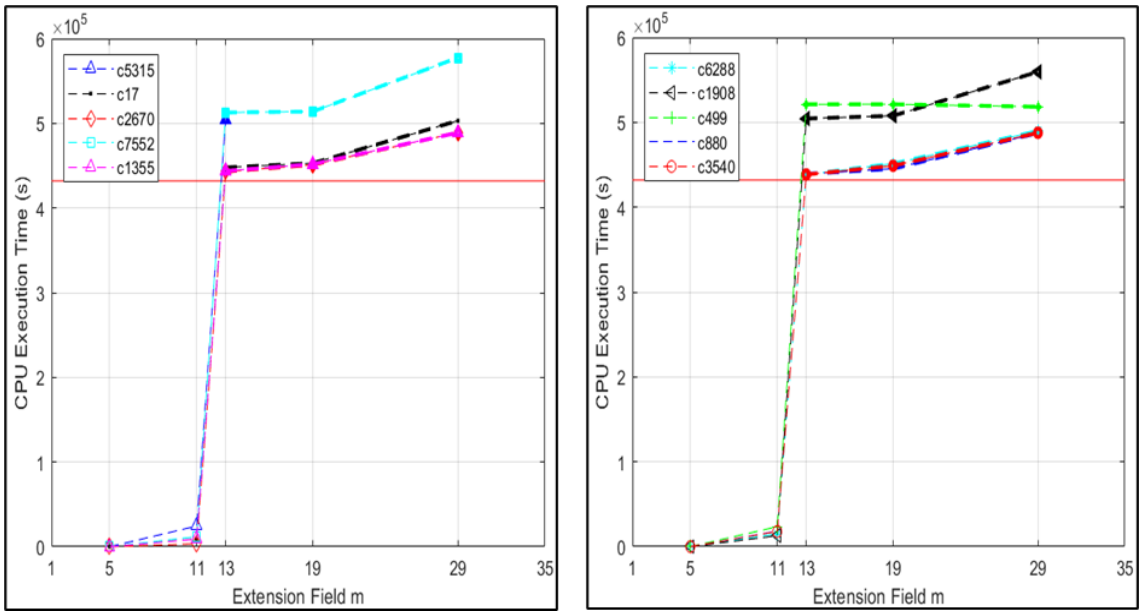

Figure 4.2: SAT attack results on the benchmark circuits excluding c432 using our logic locking configuration: ECCLock. Timeout is 5 days (4.32×10^5 s) indicated by the solid straight horizontal line. The marked dash lines above the solid straight horizontal line are the results when the SAT attack has time-outed, this occurs from extension field 13 and above.

Xeon(R) CPU E5-2680 at 2.70GHz with 32 cores and 128 GB RAM respectively. When run on the server the SAT attack always returns the correct key in less than 10 h but when run on the PC it always fails to return the correct key within 10 h despite that locked circuit security is breakable.

Table 4.2: Timeout time dependency

PC		Server	
Time (s)	Key	Time (s)	Key
36005.12	-	24354.4	399ed8
36011.33	-	22709.6	399ed8
44744.02	-	21145.2	399ed8

The results of the SAT attack on the ISCAS-85 benchmark circuits locked using our method are shown in Figure 4.2. The solid straight horizontal line indicates the timeout time. The region above the timeout line indicates that the SAT attack is

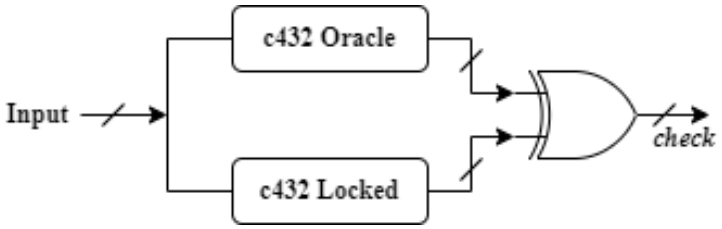

Figure 4.3: c432 verification miter

time-outed, and the region below indicates that the SAT attack succeeded. As shown in Figure 4.2, for extension fields 5 and 11, the SAT attack succeeded in finding a key that can be used to unlock the locked circuit. For extension fields from 13 (key size 2×13) and above, the SAT attack time-outed, that is, the SAT attack fails to unlock the locked circuit within five days or, in other words, fails to find a decryption key that will render the oracle and the locked circuit equivalent. It can also be observed that the CPU execution time of the SAT attack increases as m increases. Therefore, it is correct to say that extension field 11 is more secure than extension field 5. The SAT attack could return a key, not necessarily the original decryption key but any key pattern that, when applied to the locked circuit, will render the circuit functionally equivalent to the oracle. As presented in Table 4.3, when considering the extension fields in which the SAT attack can break, the key returned most of the time is different from the original decryption key (second row). Despite that the extension fields 5 and 11 are not secured, it will be interesting to explore why the SAT attack sometimes returns a correct key different from the secret key and also to further investigate the circuit c432 which results in an UNSAT for all the extension fields.

The ECCLock version of the circuit c432 is tested to verify its correctness to be able to see whether the correct key works, even though the SAT attack returns UNSAT for

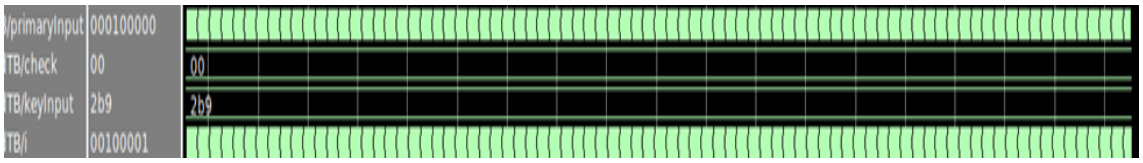

Figure 4.4: c432 UNSAT verification simulation result using the correct key 2b9

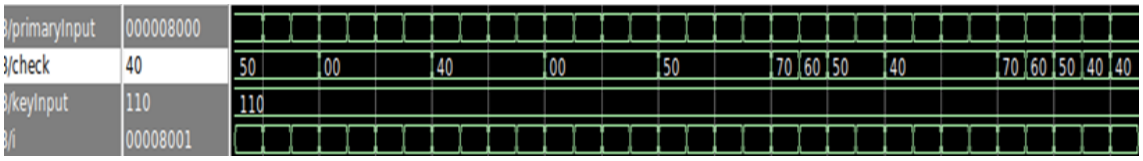

Figure 4.5: c432 UNSAT verification using incorrect key

the circuit. A miter-like circuit is constructed with the c432 oracle and its ECCLock for an extension field of 5 as shown in Figure 4.3. The output of the miter circuit (*check*) will be a string of 0's when the two circuits are equivalent. The circuit has a total of 36 inputs. The circuit is simulated with 1M input patterns from 0 to 1M (0x100000), that is more than half of the circuit number of inputs were used. The result of the simulation is shown in Figure 4.4, it can be seen from the figure that for all these input patterns the *check* output signal is always 00, that is the circuits are equivalent for all these input combinations. The correct key 2b9 is used for this purpose and all the values shown in the figure are in hexadecimal. The equivalence is also checked for 1G (0x40000000) input patterns and the result shows that they are also equivalent. When the incorrect key is applied, the circuits are not equivalent, for most of the input patterns. Figure 4.5 shows the result of the equivalence checking when an incorrect key (110) is applied. The result shown is for the input patterns from 0 to 32k. From the figure, the output signal *check* is not always equal to 00.

Most of the countermeasures sets at most 48 h for their timeout time. The Anti-SAT countermeasure [20] which is run on Intel Core i5-2400 CPU with 16 GB RAM

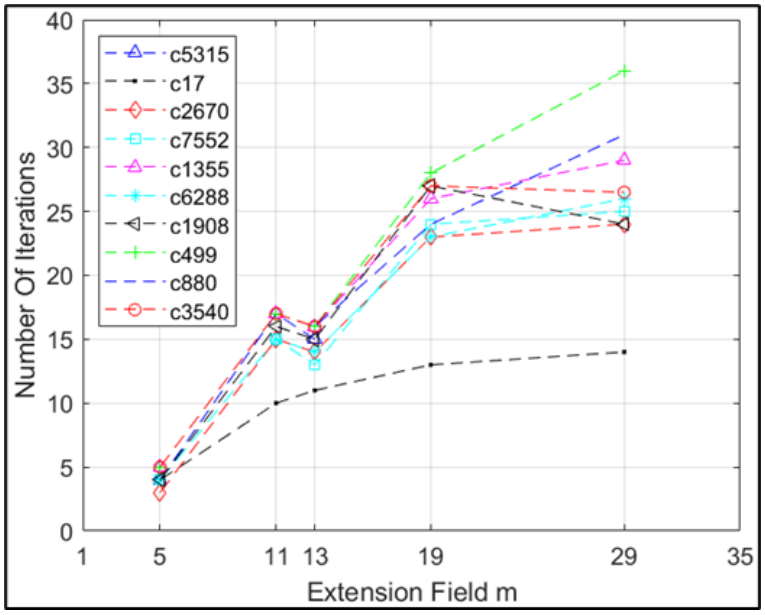

Figure 4.6: SAT attack iterations on the locked benchmark circuits.

Table 4.3: SAT attack returned keys after successful circuit unlocking.

m	5	11	13
Original Key (Hex)	2ba/2b9	399ed8	1298955
c17	2ba	399ed8	timeout
c2670	321	8cdc5	timeout
c1355	154	106e43	timeout
c6288	2b9	15750e	timeout
c880	3cd	6c1e7	timeout
c3540	130	2ba877	timeout
c499	398	399ed8	timeout
c5315	1bd	af077	timeout
c1908	36a	ec905	timeout
c7552	84	3e7071	timeout
c432	UNSAT	UNSAT	UNSAT
c17_no_pad	2ba	3e3ee9	timeout
c1355_no_pad	2e8	20208	3bc78cd

sets their timeout to 10 h. The study uses six benchmark circuits for its experiment. Of the 6 circuits, 3 are from the ISCAS-85 benchmark; c1355, c1908, and c3540. The SAT attack on these circuits time-outed at 10 h for key sizes 55, 72, and 111 respectively. In our method, only a key size of 26 (m 13) is sufficient to guard against the SAT attack. The ORF-Lock One-Way Function-Based Logic Locking method described in [5] integrates RLL/FLL with an AES block. The experimental results show that using a one-way function, such as AES, increases the execution time of the SAT attack. The result of the SAT attack using this logic locking technique with the c7552 ISCAS-85 benchmark circuit shows that with 10 fixed keys connected to the AES block, an execution time of about 2×10^5 s (2.3 days) can be achieved. However, this locking technique is prone to removal attacks and it is anticipated to have considerable overheads.

$$T_e = \sum_{n=1}^{\lambda} t_n \tag{4.1}$$

The SAT attack execution time T_e is given in equation 4.1, where λ is the number of iterations and t_n is the time required to complete n^{th} iteration. To increase the execution time of the SAT attack, either λ, t_n or both need to be increased. As shown in Figure 4.6, the SAT attack is able to complete a maximum of 17 and 27 iterations on circuits locked using ECCLock for extension fields 13 and 29 respectively. This indicates that the locking technique extends the execution time of the SAT attack by extending the time for each iteration. The complexity of the SAT attack relates to the complexity of the locked circuit CNF formula. When the formula is complex, the SAT attack takes time to find a satisfying assignment. The length of the formula

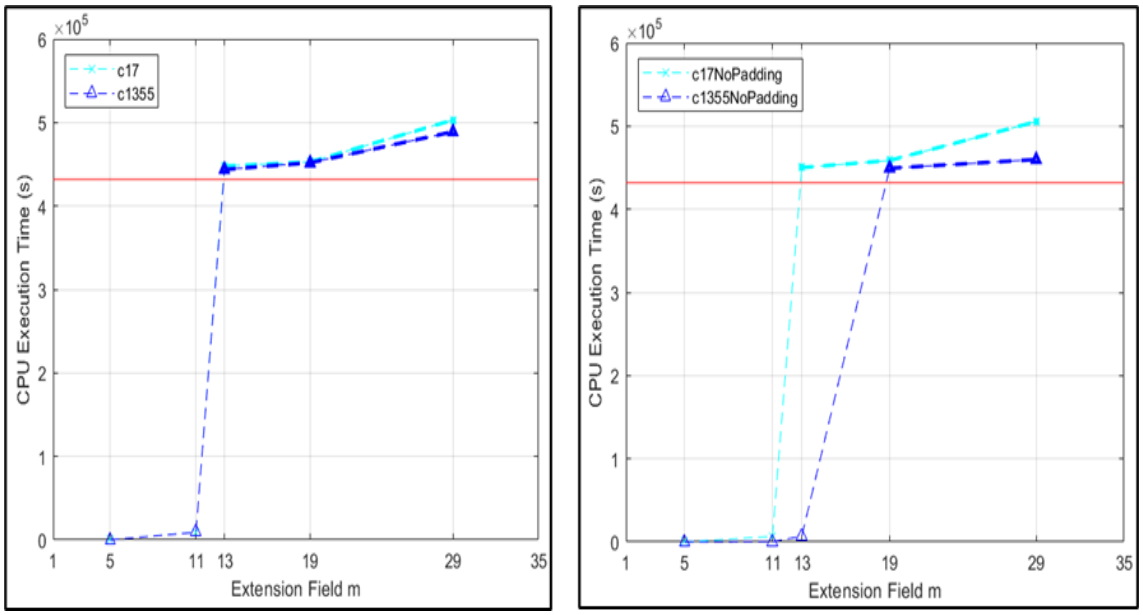

Figure 4.7: Comparison of the SAT attack result on c17 and c1355 the benchmark circuits with padding (left) and without padding (right). Timeout is 5 days (4.32×10^5 s) indicated by the solid straight horizontal line. The marked dash lines above the solid straight horizontal line are the results when the SAT attack has time-outed

in terms of the number of clauses and the number of literals affects its complexity. Therefore the length of the key and the number of inputs to the locked circuits directly affect the length of the formula and consequently, the complexity of the SAT attack. The security provided by the ECC makes the CNF formular very complex such that it hinders the easy solution to the formula. The number of inputs to the encryption unit, and the support of the encrypted cone, will impact the complexity of the SAT attack because as the literals increase so as the complexity of the CNF formula. For the encrypted cone, having impacting more primary output will result in the creation of more clauses that will be added to the CNF formula and hence increase its complexity.

It can be observed from Figure 4.7 that padding the message to be encrypted before it is represented as a point on a curve can be more secure than representing the message to a point directly without padding. The circuit c17 in both cases is secured

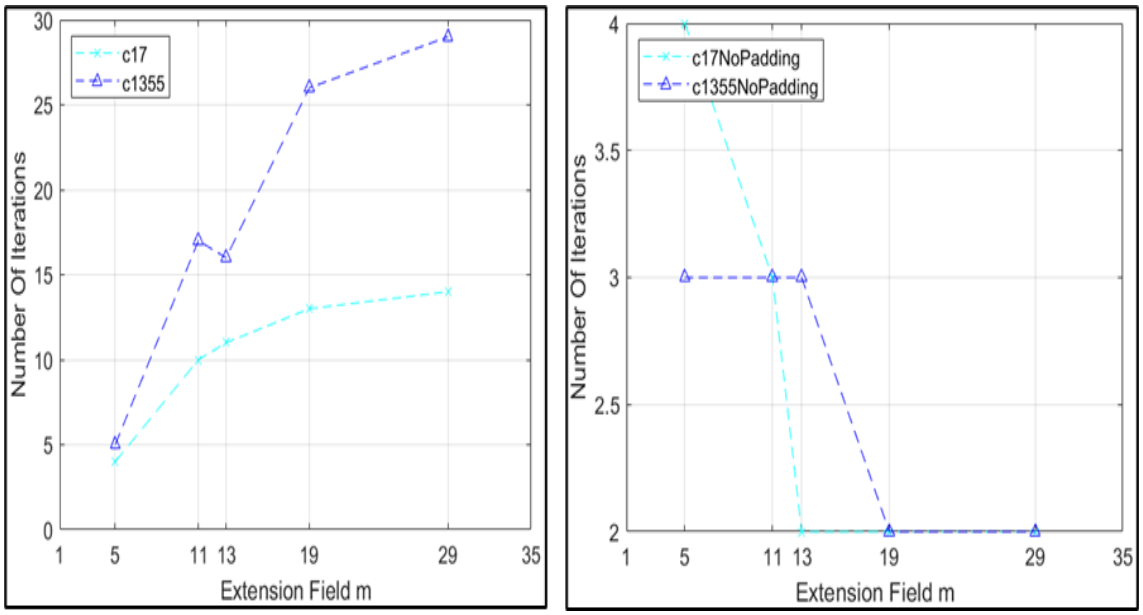

Figure 4.8: SAT attack iterations on the locked benchmark circuits c17 and c1355 with padding (left) and without padding (right).

against the SAT attack with respect to the timeout for at least an extension field of 13, this is evident in Table 4.3, whereas its counterpart c1355 is only secured, when padding is not used, for an extension field of at least 19. From Figure 4.8, it can be seen that the number of iterations monotonically decreases for no padding with respect to m, but the opposite is true for circuits locked after being padded.

4.1.2 Security Analysis of ECCLock Against Other Forms of Attacks

In this section, the security analysis of our technique is presented, evaluating its robustness against attacks discussed in Chapter 2, and examining its resistance to these attacks. Hill climbing attack relies on random guesses, and the probability of choosing a pattern from the k bit key at random is 2^{-k}. As k increases, the

probability decreases. Fault-based logic locking uses divide and conquer to sensitize individual keys to one of the circuit's primary outputs. In our proposed technique, all the k bits of the key input must be used simultaneously to decrypt the ciphertext. Hence, sensitizing any key bit to the output necessitates controlling all key bits. The cryptographic mechanisms in the Elgamal ECC algorithm [8] used in our technique can increase the complexity of the SAT solver and prevent it from solving the CNF of a locked circuit within a reasonable amount of time. The result of the resiliency of our method is presented above. From the result, ECCLock is secured against the SAT attack for at least an extension field of 13. The resilience of our method to the approximate-based SAT attack is similar to that of the SAT attack. Our technique does not have the problem of probability skew, because it does not use an AND/OR tree or complementary functions. Therefore, resilient to signal probability skew attack. ECCLock is not a victim of removal attacks because the protection logic is intertwined with the original circuit. Double-DIP works the same way as the SAT attack, the bypass attack will only succeed if the SAT attack is successful and the incorrect key pool is relatively small. Hide-and-seek is a detrimental attack that exploits the specific features provably secure logic-locking techniques. Our countermeasure, being non-provably secure, adopts a distinctive approach by preventing a key from propagating to a primary output. The hide-and-seek attack targets mainly the point function logic locking techniques while ECCLock is not a point function. Therefore, ECCLock is not a victim of this attack and can be used to circumvent this attack. We will test the security of our technique using the hide-and-seek attack in the future.

4.1.3 Hardware, Delay, and Power Overhead

Table 4.4: The number of gates, estimated latency (delay), and estimated power consumption of each of the ISCAS-85 benchmark circuits used.

Circuit Name	Gate Count	Delay (ns)	Power (mW)
C432	160	10.923	0.72
C17	6	8.152	0.19
C1355	546	11.932	1.39
C6288	2416	18.179	1.27
C1908	880	11.757	1.12
C499	202	11.61	1.39
C880	383	11.84	1.49
C3540	1669	14.344	1.27
C5315	2307		
C2670	1193		
C7552	3512		

The hardware (gate count), delay, and power of each of the ISCAS-85 benchmark circuits are obtained as shown in Table 4.4. The gate count is the number of the circuit's logic gates. The ECCLock circuit in Verilog format is transformed into Berkeley Logic Interchange Format (BLIF) representation using *yosys*, the resulting BLIF is imported into *ABC* and it is converted to BENCH and the gates in the resulting BENCH is obtained. The delay, and power of c5315, c2670, and c7552 are not obtained because the board used to synthesize these circuits (10CX220YF780I5G in the Cyclone 10 GX family) has a maximum of 284 user I/O pin locations and these three circuits has I/O exceeding this threshold. The percentage overhead is calculated using Equation 4.2. The O in the equation is the overhead in percentage, L is the absolute locked circuit parameter (hardware, delay, or power) and R is the oracle (base) parameter. The absolute gate count of the locked circuits is shown in Table 4.5. From Table 4.5 the number of gates in the locked circuit is directly proportional to m. The

Table 4.5: The absolute gate count of the locked circuit for different extension fields.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	1268	10663	19949	60171	194941
c17	1216	11058	20392	60760	196041
c1355	3291	14632	24443	66228	203777
c6288	7012	18391	28079	69964	207568
c1908	3208	14636	26500	69886	209120
c499	6865	23522	49081	81967	226684
c880	12539	35768	49770	103135	258907
c3540	13941	37381	49851	105053	260286
c5315	6492	19985	28931	73868	213569
c2670	2428	12455	21810	62419	198002
c7552	5134	15650	24948	66270	202410

lowest gate count among the oracles is c17, with 6 gates, and the highest is c7552 with 3,512 gates. The hardware overhead of the ECCLock logic locking is as shown in Table 4.6. It can be seen from the table that these overheads increases as m increases for all the benchmark circuits. The benchmark circuit, c17 has the highest hardware overhead compared to the other circuits because it is a small circuit of six NAND gates only.

$$O = \frac{L - R}{R} \times 100\% \tag{4.2}$$

The hardware overhead decreases as the size of the circuit to be locked increases. Circuits c6288, c7552, and c5315 have the lowest hardware overhead for a given extension field (see Table 4.6) compared to the rest of the circuits because they are larger (in terms of the number of gates) than the others. This large overhead is mainly due to the decryption unit, although in some cases it is also affected by the locking unit that produces the ciphertext. The absolute gate count of the decryption unit for each

Table 4.6: Hardware overhead on ISCAS-85 benchmark circuits locked using ECCLock logic locking technique for different extension fields in percentage.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	692.50	6564.38	12368.13	37506.88	121738.13
c17	20166.67	184200.00	339766.67	1012566.67	3267250.00
c1355	502.75	2579.85	4376.74	12029.67	37221.79
c6288	190.23	661.22	1062.21	2795.86	8491.39
c1908	264.55	1563.18	2911.36	7841.59	23663.64
c499	3298.51	11544.55	24197.52	40477.72	112119.80
c880	3173.89	9238.90	12894.78	26828.20	67499.74
c3540	735.29	2139.72	2886.88	6194.37	15495.33
c5315	181.40	766.28	1154.05	3101.91	9157.43
c2670	103.52	944.01	1728.16	5132.10	16496.98
c7552	46.18	345.62	610.36	1786.96	5663.38

Table 4.7: Decryption Unit Gate Count for Different Extension Fields

m	Gate Count
5	1147
11	11217
13	20767
19	61791
29	198677

extension field is shown in Table 4.7. From Table4.8 it can seen that these overheads increase as the extension field size increases. For extension field 13 for most of the circuits, the decryption unit constitutes more than 70% of the locked circuit. In some extension fields, the overhead is more than 100%, for example, extension fields 11, 13, 19, and 29 for c432 and c17. This repeats also for c2670 for the extension field 29. This is due to the fact that the *ABC* tool optimizes the locked circuit together with the unlocking unit than when the unlocking unit is a standalone module. The overhead generated by the decryption unit is mainly due to the finite field inversion that is involved in the EC point addition. We intend to reduce this overhead in our future work.

Table 4.8: Decryption circuit overhead compared to the locked circuit in percentage

Circuit Name	5	11	13	19	29
c432	90.46	105.20	104.10	102.69	101.92
c17	94.33	101.44	101.84	101.70	101.34
c1355	34.85	76.66	84.96	93.30	97.50
c6288	16.36	61.00	73.96	88.32	95.72
c1908	35.75	76.64	78.37	88.42	95.01
c499	16.71	47.69	42.31	75.39	87.64
c880	9.15	31.36	41.73	59.91	76.74
c3540	8.23	30.01	41.66	58.82	76.33
c5315	17.67	56.13	71.78	83.65	93.03
c2670	47.24	90.06	95.22	98.99	100.34
c7552	22.34	71.67	83.24	93.24	98.16

Table 4.9: The absolute delay of the locked circuit for different extension fields in ns.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	14.35	40.71	54.27	100.91	213.78
c17	13.21	37.35	54.06	106.41	221.24
c1355	16.63	44.62	62.15	115.58	227.91
c6288	20.64	48.66	66.63	123.60	236.71
c1908	14.92	43.18	57.05	111.51	230.72
c499	18.62	50.47	64.77	120.21	235.06
c880	25.26	54.61	69.88	124.27	240.22
c3540	24.68	53.26	73.97	125.30	235.70

The absolute delay of the locked circuit is shown in Table 4.9. From the table, the delay in the locked circuit is directly proportional to m because the number of iterations involved in the decryption unit increases with the increase in the extension field. The highest delay is observed for the extension field of 29 for the c880 benchmark circuit which has 240.22 ns. The delay overhead of the benchmark circuits is almost the same for each m (see Figure 4.10). The highest delay overhead is experienced by the c17 benchmark circuit.

The absolute power of the locked circuit is given in Table 4.11. The power in the locked circuit increases with an increase in the size of the extension field. The

Table 4.10: Delay overhead on ISCAS-85 benchmark circuits locked using ECCLock logic locking technique for different extension fields in percentage.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	31.34	272.73	396.84	823.81	1857.19
c17	62.08	358.15	563.11	1205.36	2613.95
c1355	39.41	273.93	420.83	868.64	1810.08
c6288	13.54	167.69	266.51	579.91	1202.10
c1908	26.90	267.25	385.26	848.42	1862.40
c499	60.40	334.72	457.86	935.37	1924.60
c880	113.32	361.26	490.23	949.60	1928.87
c3540	72.03	271.33	415.71	773.51	1543.17

Table 4.11: The absolute power of the locked circuit for different extension fields in mW.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	0.85	1.00	1.06	1.21	1.50
c17	0.32	0.48	0.53	0.69	0.95
c1355	1.52	1.70	1.76	1.91	2.17
c6288	1.40	1.56	1.61	1.79	2.06
c1908	1.25	1.40	1.46	1.64	1.90
c499	1.52	1.70	1.76	1.91	2.17
c880	1.65	1.81	1.86	2.01	2.30
c3540	1.40	1.58	1.63	1.79	2.05

Table 4.12: Power overhead on ISCAS-85 benchmark circuits locked using ECCLock logic locking technique for different extension fields in percentage.

Circuit Name	Extension field m				
	5	11	13	19	29
c432	18.06	38.89	47.22	68.06	108.33
c17	68.42	152.63	178.95	263.16	400.00
c1355	9.35	22.30	26.62	37.41	56.12
c6288	10.24	22.83	26.77	40.94	62.20
c1908	11.61	25.00	30.36	46.43	69.64
c499	9.35	22.30	26.62	37.41	56.12
c880	10.74	21.48	24.83	34.90	54.36
c3540	10.24	24.41	28.35	40.94	61.42

highest power is 1.49 mW for the c880 oracle. For the extension field of 29, the c880 benchmark circuit still consumes more power compared to the rest of the benchmarks. The power overhead for all the circuits increases as the extension field becomes larger. As shown in Table 4.12, the benchmark circuits c499 and c1355 have power overheads of less than 10% for extension field 5. For the highest extension field, the majority of the circuits have a power overhead of less than 70%. The power overhead of c17 which is more than 100% is because the original circuit consumes very little power compared to the rest of the circuits. The power overhead reduces as the power consumption of the original circuit increases. The power overhead can also be reduced by reducing the delay overhead using a cache. Because when there is a cache hit the dynamic power that the encryption and the decryption logic will consume will not be turned on. Therefore, reducing the delay overhead can reduce the power overhead as well.

The takeaway message is that the security gain provided by ECCLock outweighs the overheads incurred in areas that have zero tolerance for security threats like the military field and areas related to national security.

A good point worth noting is the effect of quantum computing with respect to logic locking techniques. Despite that quantum computing is in its early stage, when it comes to a breakthrough, will change many things. ECC, which stands out today as the most secure and efficient asymmetric cryptosystem, could be no longer secured with the advent of quantum computers [42]. The computation power of quantum computers will enable the execution of any attack on combinational circuits faster, and in no time the secret key could be revealed. The timeout time of most of the logic locking techniques is 36,000 s (10 h) and the timeout time of our technique is 432,000

s (5 days). Thus our technique could be more reliable than other forms of techniques, and as can be seen from our results, the security of the circuit can be increased by increasing the size of the extension field.

4.2 Conclusion

This work proposes a new logic locking technique dubbed ECCLock by utilizing cryptographic algorithms of elliptic curve cryptography, which, before this time, was mostly used in software. We assess this method for its effectiveness in terms of security, delay, hardware requirements, and power consumption. We show that our logic locking technique can increase the SAT attack execution time. The length of the extension field has an effect on the security of the locked circuit with respect to the SAT attack. While our technique has considerable hardware overhead, in 10^4x magnitudes on average, the high security provided is a good tradeoff. In the future, we intend to optimize the technique to reduce these overheads, especially the hardware overhead. Overall, ECCLock is capable of thwarting the well-known attack on combinational logic circuits, the SAT attack, and at the same time is resilient to structural attacks such as removal attacks.

APPENDIX A

SUPPLEMENTARY MATERIAL

A.1 Generating random curve domain parameters for F_{2^m}:

- Using the tool `ecgen` [38] to generate a random curve of any bit size.
- In the terminal window where the folder of the tool resides, type `ecgen --f2m -r -cx m`, where m is the extension field.
- For example, `ecgen --f2m -r -c5 5` generates random curve domain parameters for the 2^5 field.
- The `-cx` option generates x different curves of the same bit length but with different domain parameters, enabling the selection of a more suitable curve.
- Obtain the parameters a, b, n, x, and y from the output of this command.
- select a curve that has a cofactor of 1

- The recommended irreducible polynomial F is then selected. They are given in the Appendix section of [8].

A.2 Extracting a suitable cone from a large circuit

To extract a suitable cone in PLA format from a given network, the steps below are followed.

- Extract the statistics of the number of affected primary outputs and the number of supports (primary input) for each cone in the circuit.
- Determine a suitable cone for extraction, say $suitable_cone$. This cone should have sufficient support (nominal value of 10) and should affect a reasonable number of primary outputs of the circuit.
- Import the whole circuit into *ABC*; *read_bench aa.bench*, aa.bench is the name of the bench file
- Extract the suitable cone; *conesuitable_cone*
- To view the structure of this cone; show
- Collapse the extracted cone; collapse
- Write the PLA format of the extracted cone to a file; *write_pla aa.suitable_cone_pla*
- *aa.suitable_cone_pla* is now the extracted cone.

REFERENCES

[1] "Integrated Circuits Market Size, Trends and Global Forecast To 2032." [Online]. Available: https://www.thebusinessresearchcompany.com/report/integrated-circuits-global-market-report

[2] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, Sep. 2016, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[3] 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 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 1601–1618. [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3133985

[4] W. P. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit Level IC Protection Through Direct Injection of Process Variations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 5, p. 791, 2012. [Online].

Available: https://www.academia.edu/es/50513039/CLIP_Circuit_Level_IC_Protection_Through_Direct_Injection_of_Process_Variations

[5] M. Yasin, J. Rajendran, and O. Sinanoglu, *Trustworthy Hardware Design: Combinational Logic Locking Techniques*, ser. Analog Circuits and Signal Processing. Cham: Springer International Publishing, 2020. [Online]. Available: http://link.springer.com/10.1007/978-3-030-15334-2

[6] A. Jain, U. Guin, M. T. Rahman, N. Asadizanjani, D. Duvalsaint, and R. D. S. Blanton, "Special session: Novel attacks on logic-locking," in *2020 IEEE 38TH VLSI TEST SYMPOSIUM (VTS 2020)*, ser. IEEE VLSI Test Symposium. IEEE; IEEE Comp Soc; IEEE Philadelphia Sect; IEEE Comp Soc Test Technol Tech Council, 2020, iEEE 38th VLSI Test Symposium (VTS), ELECTR NETWORK, APR 06-08, 2020.

[7] S. Patnaik, N. Limaye, and O. Sinanoglu, "Hide and Seek: Seeking the (Un)-Hidden Key in Provably-Secure Logic Locking Techniques," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3290–3305, 2022, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/document/9893861/footnotes#footnotes-id-fn5

[8] D. R. Hankerson, A. J. Menezes, S. A. Vanstone, D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, ser. Springer professional computing. New York, NY: Springer, 2004.

[9] "Crypto Basics," in *Information Security*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Sep. 2011, pp. 17–49. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/9781118027974.ch2

[10] M. A. Mehrabi, C. Doche, and A. Jolfaei, "Elliptic curve cryptography point multiplication core for hardware security module," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 69, no. 11, pp. 1707–1718, NOV 1 2020.

[11] "Public Key Crypto," in *Information Security*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Sep. 2011, pp. 89–123. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/9781118027974.ch4

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

[13] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *2008 Design, Automation and Test in Europe*, Mar. 2008, pp. 1069–1074, iSSN: 1558-1101.

[14] 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, Feb. 2015. [Online]. Available: https://ieeexplore.ieee.org/document/6616532/

[15] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," Jan. 2018, arXiv:1801.04961 [cs]. [Online]. Available: http://arxiv.org/abs/1801.04961

[16] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, Jun. 2015, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. [Online]. Available: https://ieeexplore.ieee.org/document/7045595

[17] "ATPG-Guided Fault Injection Attacks on Logic Locking | IEEE Conference Publication | IEEE Xplore." [Online]. Available: https://ieeexplore.ieee.org/document/9337734

[18] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 95–100, May 2017, conference Name: 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) ISBN: 9781538639290 Place: Mclean, VA, USA Publisher: IEEE. [Online]. Available: http://ieeexplore.ieee.org/document/7951805/

[19] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *2016 IEEE International Symposium*

on Hardware Oriented Security and Trust (HOST), May 2016, pp. 236–241. [Online]. Available: https://ieeexplore.ieee.org/document/7495588

[20] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, Feb. 2019, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. [Online]. Available: https://ieeexplore.ieee.org/document/8279462

[21] A. Sengupta, N. Limaye, and O. Sinanoglu, "Breaking CAS-Lock and Its Variants by Exploiting Structural Traces," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 418–440, Jul. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8981

[22] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517–532, Apr. 2020, conference Name: IEEE Transactions on Emerging Topics in Computing.

[23] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably Secure Camouflaging Strategy for IC Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1399–1412, Aug. 2019, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[24] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, Nov. 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8397

[25] 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 on Great Lakes Symposium on VLSI 2017*. Banff Alberta Canada: ACM, May 2017, pp. 173–178. [Online]. Available: https://dl.acm.org/doi/10.1145/3060403.3060458

[26] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 49–56, iSSN: 1558-2434. [Online]. Available: https://ieeexplore.ieee.org/document/8203759

[27] S. Roshanisefat, H. M. Kamali, and A. Sasan, "SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, May 2018, pp. 153–158, arXiv:1804.09162 [cs]. [Online]. Available: http://arxiv.org/abs/1804.09162

[28] F. Yang, M. Tang, and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, Oct. 2019, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/document/8666809

[29] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, vol. 15, pp. 2514–2527, 2020.

[30] Y. Muhammad and J. Knechtel, "SAT attack SFLL Attack Framework," https://github.com/DfX-NYUAD/CCS17, 2023, accessed: 2023-03-28. [Online]. Available: https://github.com/DfX-NYUAD/CCS17

[31] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, Dec. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8967131/

[32] Y. Shen, A. Rezaei, and H. Zhou, "A comparative investigation of approximate attacks on logic encryptions," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2018, pp. 271–276, iSSN: 2153-697X. [Online]. Available: https://ieeexplore.ieee.org/document/8297317

[33] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong Anti-SAT: Secure and Effective Logic Locking," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*. Santa Clara, CA, USA: IEEE, Mar. 2020, pp. 199–205. [Online]. Available: https://ieeexplore.ieee.org/document/9136983/

[34] J. Zhou and X. Zhang, "Generalized SAT-Attack-Resistant Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2581–2592,

2021. [Online]. Available: https://ieeexplore.ieee.org/document/9354192/

[35] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the Approximation Resiliency of Logic Locking and IC Camouflaging Schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, Feb. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8395439/

[36] B. L. Synthesis and V. Group, "ABC: A System for Sequential Synthesis and Verification," https://people.eecs.berkeley.edu/~alanmi/abc/, 2023, accessed: 2023-02-03.

[37] C. X. Wolf, "yosys –yosys open synthesis suite," https://github.com/YosysHQ/yosys, 2023, accessed:2023-10-08.

[38] K. A. K. Ján Jančár, "GitHub - J08nY/ecgen: Tool for generating Elliptic curve domain parameters," https://github.com/J08nY/ecgen, 2023, accessed: 2023-03-28.

[39] "ECDH/ECDHKE.py at master · elesiuta/ECDH · GitHub." [Online]. Available: https://github.com/elesiuta/ECDH/blob/master/ECDHKE.py

[40] M. Jenihhin, "Bench/BLIF/verilog Mark formats," https://pld.ttu.ee/~maksim/benchmarks/, 2023, accessed:2023-04-30.

[41] P. Subramanyan and S. Ray, "modified SAT attack tool," https://github.com/descyphy/Modified_SAT_Attack_on_Logic_Locking, 2023, accessed: 2023-03-28. [Online]. Available: https://github.com/descyphy/Modified_SAT_Attack_on_Logic_Locking

[42] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang, "The Impact of Quantum Computing on Present Cryptography," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018, arXiv:1804.00200 [cs]. [Online]. Available: http://arxiv.org/abs/1804.00200

VITAE

- Name: Aliyu Abubakar Habib
- Nationality: Nigerian
- Date of Birth: 11/29/1996
- Email: *aliyuhabibabubakar004@gmail.com*
- Permenant Address: No. 165 Babayaro Quarters Street, Rimingata, Besides Bayero University (New Campus), Kano, Kano state, Nigeria. Postal code: 700105.

- M.Sc. Computer Engineering.

  King Fahd University of Petroleum and Minerals (KFUPM) Dammam, Saudi Arabia.

- B. ENG. Computer Engineering.

  Bayero University, Kano. Kano, Nigeria.

- A paper titled "A Systematic Literature Review on Vulnerabilities, Mitigation

Techniques, and Attacks in Field-Programmable Gate Arrays" submitted to the IEEE Acess. Manuscript ID: Access-2023-26768.

- Awards:

  Best level 100 Student Electrical Engineering Department, 2015.

  Best Graduating Student, Computer Engineering Department, 2019.