6-2021

Probability Based Logic Locking on Integrated Circuits

Michael Yue

SANTA CLARA UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

I HEARBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

MICHAEL YUE

ENTITLED

Probability Based Logic Locking on Integrated Circuits

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING

Fatemeh Tehranipoor *Sara Tehranipoor* 06/06/2021

Thesis Advisor date

PROBABILITY BASED LOGIC LOCKING ON INTEGRATED CIRCUITS

BY

MICHAEL YUE

MASTER THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

OF

SANTA CLARA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

SANTA CLARA, CALIFORNIA

June, 2021

ACKNOWLEDGEMENTS

I would like to thank my professor, Dr. Sara Tehranipoor for her support and guidance throughout the course of my research and thesis assignment. I am thankful for everything that she has taught me and for giving me the opportunity to be involved in the research community.

ABSTRACT

The demand of integrated circuits (IC)s are increasing and the industry has outsourced the fabrication process to untrusted environments. An adversary at these untrusted facilities can reverse engineer parts of the IC to reveal the original design. IC piracy and overproduction are serious issues that threaten the security and integrity of a system. These ICs can be copied illegally and altered to contain malicious hardware. The pirated ICs can be placed in consumer products which may harm the system or leak sensitive information. Hardware obfuscation is a technique used to protect the original design before it gets fabricated, tested, assembled, and packaged. Hardware obfuscation intends to hide or alter the original design of a circuit to prevent attackers from determining the true design. Logic locking is a type of hardware obfuscation technique where additional key gates are inserted into the circuit. Only the correct key can unlock the functionality of that circuit otherwise the system produces the wrong output. In an effort to hinder these threats on ICs, we have developed a probability-based logic locking technique to protect the design of a circuit. Our proposed technique called ProbLock can be applied to combinational and sequential circuits through a critical selection process. We used a filtering process to select the best location of key gates based on various constraints. The main constraint is based on gate probabilities in the circuit. Each step in the filtering process generates a subset of nodes for each constraint. We also integrated an anti-SAT technique into ProbLock to enhance the security against a specific boolean satisfiability (SAT) attack. We analyzed the correlation between each constraint and adjusted the strength of the constraints before inserting key gates. We adjusted an optimized ProbLock to have a small overhead but high security metric against SAT attacks. We have tested our algorithm on 40 benchmarks from the ISCAS '85 and ISCAS '89 suite. ProbLock is evaluated using a SAT attack on the benchmark and measuring how well the attack performs on the locked circuit. Finally, we compared ProbLock to other logic locking techniques and discussed future steps for this project.

CONTENTS

List of Figures

List of Tables

I. Introduction

The semiconductor industry is constantly changing, from the production of integrated circuits (IC)s to the complexity of their design. The industry has moved to a fabless model where most of the fabrication for a chip is outsourced to a less secure and less trusted environment. From intellectual property (IP) design to manufacturing, an IC has go to through an extensive process before it reaches the end user [1]. The supply chain stages are shown in Figure 1. First, the IP owner designs a module at the RTL level, gate level, and layout level. Multiple IP designs get integrated onto a single system on chip (SoC). Next, a foundry fabricates the IC die and an assembly will package the die with pins and wires into a complete package. The final package gets manufactured and distributed out to end users and consumers. These environments in the supply chain include testing and fabrication facilities that are necessary for the pipeline. Testing and fabrication facilities are usually outsourced to other countries where it is cheaper to finish the work. While this model does improve production costs and development, it has also led to the consequence of piracy, overproduction, and cloning. An IP owner does not have control over these untrusted facilities so IP piracy is a common issue. The chips are also vulnerable to various attacks that attempt to extract the design of the chip or other information from the device. Due to these security issues, researchers have developed techniques to counter these attacks. Other research topics including developing attacks to evaluate the security and privacy of ICs at different stages of the supply chain.

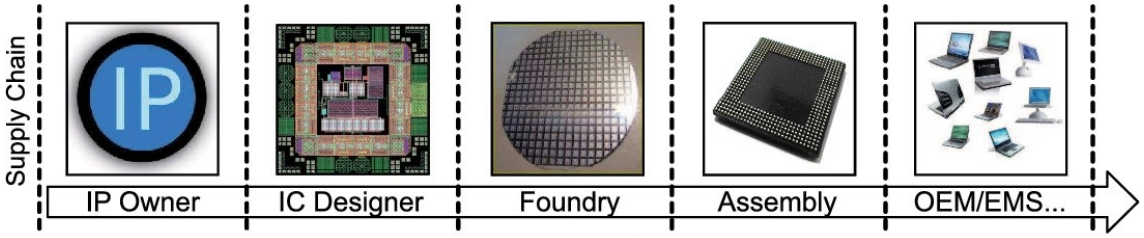


Figure 1: IC Supply Chain Stages [1]

The main threats in the IC supply chain are reverse engineering, IP piracy, and tampering [1]. The main goal of reverse engineering is to determine the design and behavior of the IP modules on an IC or SoC. Reverse engineering attacks will exploit the weaknesses and security vulnerabilities of an IC to recreate the original design of an IP. The reverse engineered design can then be used to sell counterfeit hardware on the black market. Reverse engineering can occur anywhere along the IC supply chain including the design house, foundry, testing site, and at the end user. Reverse engineering attacks are also usually destructive and sometimes require chemical and physical alteration of the IC to recover the design. IP piracy is another major concern in the IC supply chain. Facilities in the IC supply chain that use the IP illegally are violating piracy rules. Piracy usually includes overproduction and counterfeit production. A foundry with access to the IP from the designer can produce more ICs than what was ordered and sell the extra for profit. Adversaries at these sites can also clone or copy the design and sell that for profit as well. Tampering is the idea of modifying the design for other than its intended purpose. An attacker can accomplish this by inserting hardware Trojans that will exploit sensitive data on the IC and transmit that data to an outside party [2]. Trojans can also sabotage the IC by targeting critical path data such as power and timing modules on the circuit.

A. Background

One technique to improve the security of ICs is hardware obfuscation [3]. Hardware obfuscation is a technique that modifies the structure or description of a circuit to make it harder for an attacker to reverse engineer the hardware. Obfuscation techniques can be applied at the register-transfer level (RTL), gate level, and layout level of a design.

At the RTL level, IP design is usually in the form of a hardware description language (HDL) such as Verilog or VHDL. The HDL is used to create a high level implementation of a circuit. Attackers will also want to reverse engineer the original RTL code for piracy. This design can be obfuscated using a traditional encryption technique such as advanced encryption standard (AES) and Rivest-Shamir-Adleman (RSA) algorithms. As shown in figure 2, the RTL code or plain text can be encrypted through a formula

and decrypted using a generated key. This makes it harder for an adversary to retrieve the original design unless they have the secret key needed for decryption. At the RTL level, code can also be represented in a graph format with a state machine. Additional states can be added to obfuscate how the circuit behaves under a certain condition. Only the correct key applied to the circuit will make the design function properly. Otherwise, the functional states may stall or go to an incorrect state.

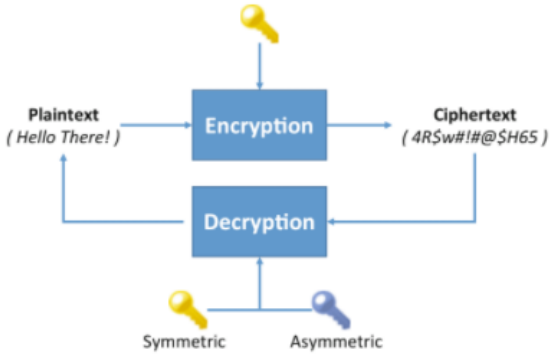
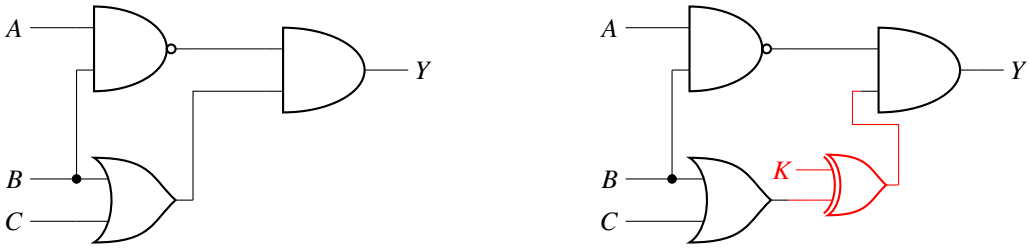


Figure 2: Encryption [3]

At the gate level, an IP design is formed with standard logic cells and components. Logic locking is a technique that inserts additional gates and logic components into a circuit which will lock the circuit and produce an incorrect output unless the proper key is provided to the circuit [3]. The IC will be considered locked or functionally incorrect until the correct key unlocks the additional gates. The key is a generated binary value that corresponds to key gates in the circuit. Using XOR and XNOR components as key gates, the proper key value will make the gate act as a buffer and do not affect the rest of the logic. If the wrong key value is provided, the key gate will produce a wrong value and make the circuit nonfunctional. Figure 3 shows an example of logic locking. A key gate is added in between logic gates with one input connected to the key bit value. The addition of these key gates adds a small overhead to the overall circuit while increasing the security of the device. These security concerns have piloted the field of hardware security and have raised serious issues for IP designers.



(a) Unlocked Circuit (b) Locked Circuit

Figure 3: An Example of a Logic Locking Circuit.

Other gate level obfuscation techniques use finite state machines (FSM) and authentication. A FSM can be embedded into the gate level design of a circuit to add extra security to the system [3] [4]. Additional states are added to the original states to obfuscate the design. Only an authorized user will be able to traverse the correct states by applying a specific input pattern. If the wrong value is applied, the state machine will enter an infinite loop state and behave incorrectly. As shown in figure 4, only verified users from a verified state will be allowed to pass through the additional states into the original design. Authentication can also be used to prevent adversaries in an untrusted foundry from IC overproduction or IP piracy [3]. Only authenticated users at a facility will have access to a secret key to unlock the circuit. One way to generate a unique key is through physical unclonable functions (PUF). A PUF uses the random process variation on each chip to generate a key unique to that system. This key is used in a series of challenge response pairs to authenticate the IC. As shown in figure 5, a certain input pattern challenge will produce a corresponding response for a given PUF generated sequence. This same challenge response pair should be replicated at the foundry to authenticate the user and the IC. This process keeps adversaries in check in untrusted environments.

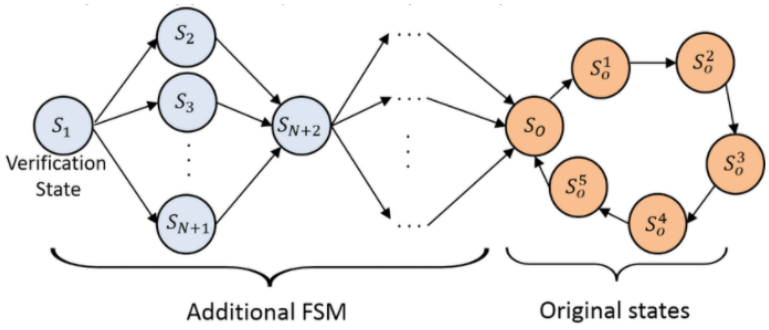


Figure 4: FSM-based Locking [4]

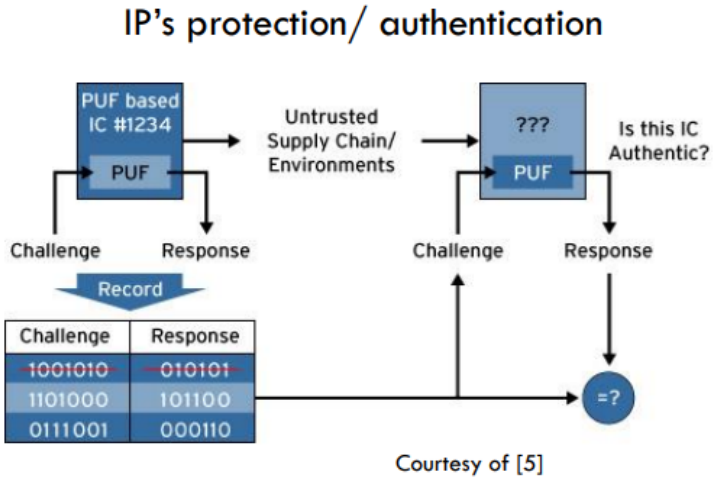


Figure 5: PUF Based Authentication [3]

Layout specific obfuscation techniques prevent attacks and tampering at the layout level of an IP design. At the layout layer, the geometric and spatial information can be used to reverse engineer an IP so obfuscation can be used to protect that information. This type of obfuscation can be used to prevent or detect hardware Trojans. One technique is called split manufacturing [3]. In split manufacturing, the front and back end of the layout are fabricated at different facilities. At the untrusted foundry, the front end of line layer (FEOL) is fabricated. The FEOL is the first portion of the IC fabrication where individual devices are patterned into the semiconductor. FEOL includes transistors and active layers from the layout. The trusted design house will create the back end of line layer (BEOL). The BEOL is the second half of the IC fabrication where the individual devices get interconnected to the wiring on the wafer. The BEOL is less costly to manufacture and can easily be done by the design house. Figure 6 shows the cross section of a layout during split manufacturing.

Camouflaging is a layout obfuscation technique that makes the layout appear the same regardless of gate implementation [3]. Figure 7 shows dummy contacts at the intersection of the dielectric layer in the layout which creates an ambiguity of the gates. Dummy contacts may or may not connect parts of the layout which creates confusion for an adversary. An attacker would have trouble determining the netlist with these false contacts. Another similar technique is used to prevent hardware Trojans. Figure 8 shows how the whitespace of a layout can be filled to prevent Trojans. Normally, an adversary will insert a hardware Trojan into a layout where there is whitespace. A hardware Trojan is additional hardware inserted into the layout so whitespace is the only place to insert it. If a designer fills the whitespace with dummy modules, it will be harder for an attacker to insert a Trojan.

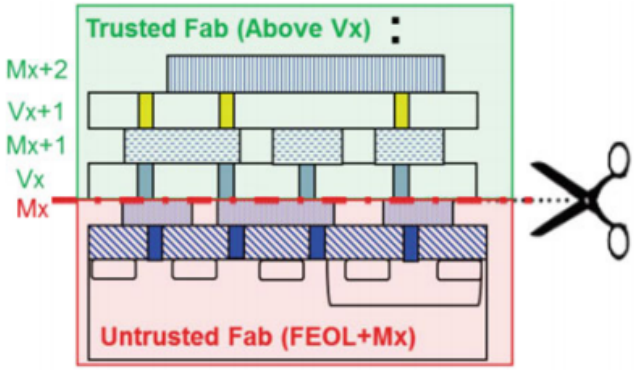


Figure 6: Split Manufacturing Obfuscation [3]

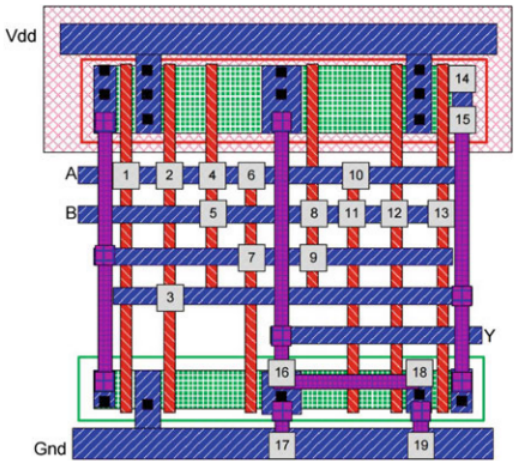
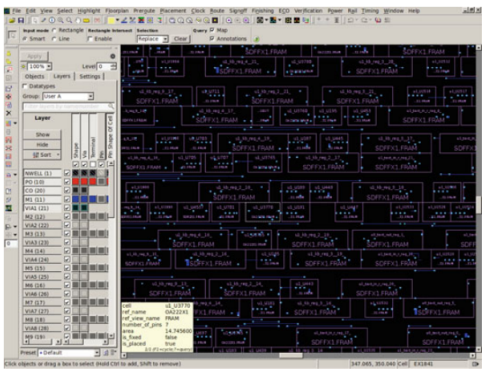
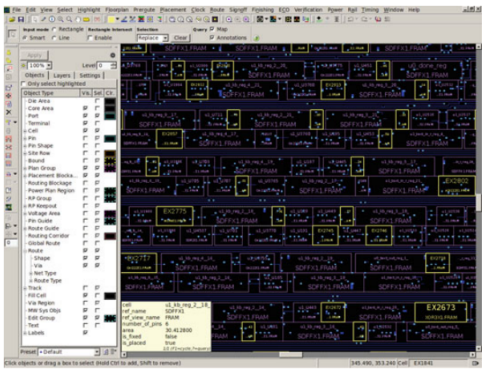


Figure 7: Layout Camouflaging Obfuscation [3]



(a) Layout With Whitespace [3]



(b) Layout With Whitespace Filled [3]

Figure 8: Trojan Resistant Obfuscation [3]

B. Motivation and Outline

The field of hardware security is relatively new and the research is very important to the semiconductor and electronics industry. Technology is constantly changing and the demand for ICs and semiconductor devices is increasing. The industry has moved to a fabless model where the fabrication and testing of ICs are performed at separate facilities instead of the design house. These facilities are untrusted and open opportunities for an adversary to exploit the IC design. Overproduction refers to a facility producing

more ICs than ordered by a company and selling the extra on the black market. IP piracy is the general idea of stealing the IP design or cloning the design for other malicious intent. Any piece of consumer technology can contain pirated IP. This may not directly impact consumers if the adversary intends to sell the IP for profit. However, it is possible where the system can leak sensitive information or cause a system failure in the product. The issues associated with hardware security affect all modern technology consumers and can be easily prevented with hardware protection schemes. Overproduction and IP piracy are serious issues that need to be addressed immediately. This research is motivated by this issue in the field of hardware security.

This research explored and developed hardware obfuscation techniques specifically for gate level netlists using logic locking. This thesis will start with a literature review of previous hardware obfuscation topics such as logic locking techniques, attacks on logic locking, and evaluation of logic locking security. Different types of logic locking will be discussed in the later sections including a novel technique derived from the probability of gate outputs in a circuit. This section will also discuss the design for hardware security against specific attacks. The evaluation of probability based logic locking is explored in experimental tests. The results will be analyzed and evaluated in the next section. Finally, this thesis will end with a conclusion about the research and possible future works on the topic.

II. Literature review

A. Attacks on Logic Locking

Researchers have developed strong logic locking techniques to increase security in ICs. These techniques usually provide some type of key or have a golden circuit that represents the true design of the system. An attack on logic locking will try to decipher the key or the golden circuit. As presented in [12], many attacks on logic locking exist to show the strength of the technique against various types of attacks. There are many types of attacks that have been developed against logic locking. An oracle refers to an unlocked circuit or a functionally correct circuit where an input would provide the correct output. Oracle-guided attacks use IO information to generate an algorithm while an oracle-less attack doesn't rely on an oracle. Structural attacks are based on using structural analysis to identify components of the system. The information gathered is used to create an algorithm for the attack. Functional attacks use functional analysis to determine various properties of the system. The basis of functional and structural analysis is used in many modern attacks on logic locking.

Boolean satisfiability or SAT attack is a type of algorithm that intends to determine the key based on certain inputs and corresponding outputs to the circuit. A distinguishing input pattern (DIP) is the metric used to determine the viability of a SAT attack. Input patterns are tested on an oracle and a computer algorithm will control the whole process. DIPs will eliminate wrong key values to unlock the circuit. Different input output patterns reveal potential key bit values for the locked circuit. SAT attacks are strong against obfuscation techniques that are not designed specifically to counter them.

In 2015, Subramanyan et al. [13] proposed a SAT based algorithm to evaluate the security of hardware obfuscated circuits. They were successful in decrypting 95% of the evaluated benchmarks and determining the secret key used to unlock the circuit. This SAT attack is an oracle based attack that uses the SAT formula to eliminate wrong key values using DIPs. The attack algorithm uses a formula to determine information about the key bits used in the circuit. However, for each iteration, the formula increases in size which makes it unscalable for larger circuits. This attack still provides partial information about the locked circuit and also established a baseline for future SAT attacks and evaluation methods on logic locking.

In 2017, Shamsi et al. [14] proposed a functional attack called AppSAT, an approximate deobfuscation algorithm based on the traditional SAT attack. Their attack can successfully approximate the key values of several benchmark circuits with high accuracy. Their attack shows that in SAT attack defenses will usually have a tradeoff between exact attack resiliency and approximate attack resiliency.

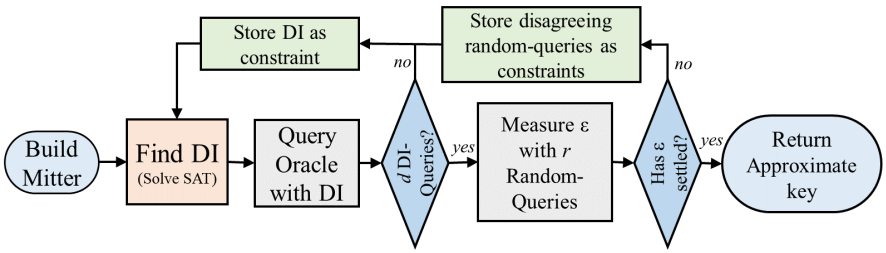


Figure 9: AppSAT Attack Flow [14]

The AppSAT algorithm employs the probably-approximately correct (PAC) setting, where the algorithm will return, with a certain probability, an approximation of the deobfuscated circuit. The attack flow is shown in Figure 9. Whereas for exact SAT attacks, the algorithms continue until no more discriminating inputs (DI) can be found, AppSAT can be terminated at any point. In order to know when to stop, after every d DI queries, r random input patterns are queried, and the accuracy of the current key is calculated. If the error stays below a specified threshold for a specified number of times, the algorithm stops, and the approximated key has been found. This intermediate querying allows for random query reinforcement, where input samples that disagree with the unlocked circuit can be added to the SAT formula as constraints, which significantly speeds up the running time of the attack. In compound obfuscation schemes. AppSAT can deobfuscate the high corruptibility portions of the circuit as if the low corruptibility point functions are not present in the circuit. An attacker has access to the netlist of the obfuscated circuit, as well

as an unlocked circuit with correct keys to be used as a black box. The purpose of the attack relaxes the exactness constraint of the traditional SAT attack and allows the key of the obfuscated circuit to be approximated with high accuracy, as achieving perfect accuracy for key bits is not always necessary for some applications. AppSAT targets the high corruptibility portion of a circuit and converges on an acceptable key value faster than a traditional SAT attack.

Sirone et al. [15] published a paper in 2020 detailing their latest attack on logic locking, calling it Function Analysis Attack, or FALL. This attack is a combination of structural and functional attacks and does not need an oracle, or an unlocked version of the circuit, to determine the correct key. The attack is broken into three sections, the first being the structural attack, the second being the functional attack, and the final portion being key confirmation. These stages are shown in Figure 10. This attack uses multiple algorithms to find the key and is designed to defeat SFLL-hd [16], which is claimed to be one of the most robust forms of logic locking to date. These algorithms use concepts like set analysis and equivalence checking to identify candidate gates within the circuit and determine which key bits are correct.

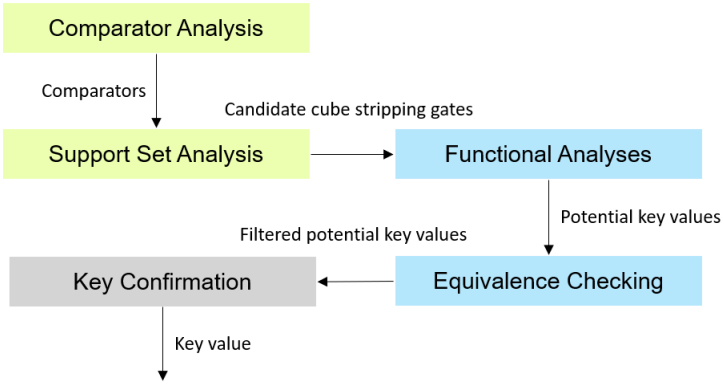


Figure 10: The 3 Stages of the FALL [15] attack

First, comparators are used to find key gates within the circuit and to build a list of key gates and their inputs, outputs, and key bits. Support set matching is used to generate a list of candidate gates. Three algorithms are then used to solve for the key, each one corresponding to different dimensions of SFLL-HD ^{h}. For SFLL-HD ^{0}, the sign of the unateness of the inputs of a candidate gate can reveal the key bit. Unateness describes a relationship between the input and output of a logic function. Positive unate describes the output of a logic cell that is the same as the input, and negative unate is when the output is the inverted version of the input. The paper proves that for the ith key bit, k_i, a positive unate candidate key gate means that $k_i = 1$ and a negative unate candidate key gate causes $k_i = 0$. For SFLL-HD ^{h} where $h < m/2$ and m is the length of the key, two satisfying assignments are made for the candidate gates that have a support hamming distance of 2h. It can be concluded that any matching bits in both satisfying assignments are correct key bits. The remaining bits can be found by looping through the remaining bits and applying a SAT equation. For SFLL-HD ^{h} where $4h \leq m$ and m is the length of the key. This algorithm works like the previous one at first, but once all matching bits are found by looking for two more satisfying assignments such that the bits which were not equal in the first pair of assignments are now equal.

An attacker has access to gate level netlist. Sequential circuits can be attacked if flip flop inputs and outputs are treated as combinational inputs and outputs. This attack specifically targets circuits that are locked with SFLL. The objective is to recover the correct key and restore functionality to the circuit. FALL was able to produce the key for 65 of 80 circuits that were tested, including the 7 largest benchmark circuits with 128 bit keys. All successful attacks were completed under the time limit of 1000 seconds. If SFLL-HD ^{h} where $0 < h \leq m/2$ or $h > m/4$ where m is the length of the key, then this attack fails and cannot deduce the key.

FALL is one of the most recent attacks discussed in this paper, making it one of the more capable and powerful attacks. It can defeat SFLL, which is said to be one of the stronger logic locking integration techniques, for most cases. The idea of using a structural attack to generate candidate gates and to identify

important inputs and outputs is what helps this algorithm be so potent. Another strength of FALL is that it is oracle-less and can even be used on combinational circuits. However, an oracle is required to do the key confirmation, which is another strong tool that allows attackers to confirm that the key is correct. In addition, FALL is magnitudes faster than SAT in experiments, making it time efficient, an important trait for attacking logic locked circuits.

B. Logic Locking Papers

Many techniques of logic locking have already been proposed and tested against certain attacks and on circuit benchmarks. The survey presented in [5] discusses the rise in popularity of logic locking as a hardware obfuscation technique and talks about the weaknesses of current protection schemes. One of the earliest logic locking techniques inserts key gates randomly into the circuit. This provided some security, but many attacks were developed to break this method. Another obfuscation technique was developed using logic cone analysis in [6]. Sections of a circuit can be grouped into logic cones by calculating the fan-in and fan-out values of a gate. Inserting key gates at certain logic cones areas will increase the security of the system. Logic cone analysis is good for countering logic cone attacks. Certain attacks will exploit these weak logic cones and try to discover the key to unlock the circuit. Logic cone analysis is vulnerable to other types of attacks such as SAT and functional attacks.

Strong logic locking (SLL) is another obfuscation technique, but it is also vulnerable to SAT attacks [7]. SLL is based on interference graphs that show how inserted key gates interfere with each other. The interference graph shows the relationship between an inserted key gate and its surrounding key gates and wires. The interference graph shows if key gates are on a cascading path, parallel path, or if they don't interfere with each other at all. The interference graph along with other information makes it harder for an attacker to unlock the circuit even with a SAT attack model. The type of key gate inserted can either be a run of key gates, isolated key gates, or dominating key gates. A run of key gates is inserted where the output of one key gate connects to the input of another key gate. A run of key gates reduces the effort an attacker needs to determine the key bit because multiple key bits can unlock the same run of gates. An isolated key gate occurs when there is no path from a key gate to other key gates. An attack prefers isolated key gates because once a pattern is determined to sensitize the bit to an output, they can determine the correct value of that key bit. Dominating key gates occur if one key gate lies on every path between another key gate and the output. An attacker can only determine the value of the dominating key gate if the effect of the other key gate is muted. All of the information between the type of key gates and their relationship provides a logic locking framework that is scalable and provably secure.

More recent techniques have been developed to counter SAT attacks and other related schemes. The obfuscation technique needs to be strong enough to resist certain attacks otherwise the integrity of the IC would be compromised. The goal of an adversary during an attack is to determine the secret key to unlock the circuit or gain other important information from the system. SARLock was developed to make the SAT attack model inefficient [8]. SARLock employs a small overhead strategy that exponentially increases the number of DIPs needed to unlock the circuit. SARLock is very strong against SAT attacks since it uses the basis of the attack model to determine where to insert key gates. In this way, the encryption technique exploits the weakness of the SAT algorithm to insert key gates. The input pattern and corresponding key values can be analyzed during the insertion process of the obfuscation technique.

In 2017, TTLock was proposed that resisted all known attacks including SAT and sensitization attacks [9]. TTLock would invert the response to a logic cone to protect the input pattern. The logic cone would be restored only if the correct key is provided. The small change to the functionality of the circuit would maximize the efforts needed for the SAT attacks. The generalized form of stripping away the functionality of logic cones and hiding it from attackers is known as stripped-functionality logic locking (SFLL). However, the design of TTLock didn't account for the cost of tamper-proof memory which could lead to high overhead in the re-synthesis process [10] [11]. Another group automated the general process of TTLock to identify the parts of the design that needed to be modified efficiently. They used ATPG tools to develop a scalable and more efficient way of protecting these patterns from attackers. Overall, a 35% improvement in overhead was achieved with the automated process. Later, a modified version of SFLL was proposed based on the hamming distance of the key. This was referred to as SFLL-hd [17]. The hamming distance metric was used to determine which pattern to modify in the SFLL scheme. Depending on the type of attack, the hamming distance can be adjusted accordingly. In 2019, the idea of

exploring high-level synthesis (HLS) with logic locking was proposed with SFLL-HLS [18]. SFLL-HLS was proposed to improve the system-wide security of an IC. The design resulted in faster validation of design and higher levels of abstraction. The HLS implementation in this technique was used to identify the functional units and logic cones to be operated on with respect to SFLL. They observed low overhead and power results from their analysis.

Many forms and variations of SAT attacks have been created to show the weaknesses of various hardware obfuscation techniques. As a result, an anti-SAT unit was developed as a general solution to the SAT attack [24]. The anti-SAT block consists of a low overhead unit that can be added to any obfuscation technique to help counter the SAT attacks. The unit requires the key length for the locked circuit to increase as inputs to the anti-SAT block. The number of DIPs and input patterns that an adversary needs would grow exponentially due to this change. This would make the complexity of the SAT attack exponential instead of linear and therefore inefficient. The overview in figure 11 shows how the anti-SAT block can be implemented with any existing logic locking technique. The locked circuit contains inserted key gates from any type of logic locking algorithm. The key is split into two parts where some key bits unlock the locked circuit while other key bits are used in the anti-SAT block. The anti-SAT block also takes in structurally obfuscated wires from the locked netlist. An attack algorithm would have trouble eliminating wrong key values due to the addition of the anti-SAT block and the number of iterations in the SAT attack would increase exponentially. Due to the light overhead, scalability, and provable security of the anti-SAT block, researchers have developed similar techniques to fight against SAT attacks. The recent innovation in anti-SAT has inspired us to develop a technique that will be resistant to various SAT attacks. We designed constraints that should minimize the effects of a SAT algorithm.

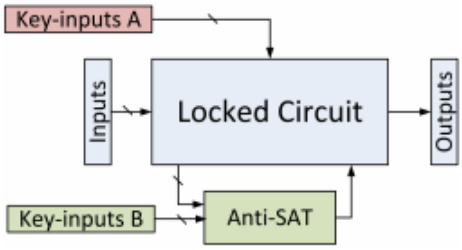


Figure 11: Anti Sat Block Overview [24]

III. Logic Locking Hardware Obfuscation

Logic locking is a specific form of hardware obfuscation that inserts additional logic components or key gates into a netlist. The addition of key gates adds a small overhead price to the design of the circuit, but it can greatly increase the security and privacy of the hardware. The insertion technique and algorithm of key gates determine the security against certain types of attacks. Each locked circuit requires a specific key to unlock the functionality of the circuit, otherwise, the circuit provides incorrect output behavior. Logic locking encryption techniques can be designed to counter specific attacks such as SAT attacks, key sensitization attacks, and side channel attacks. Researchers have also combined ideas from different techniques to create stronger and more secure netlists.

A. Types of Logic Locking

There are two main types of logic locking, functional and structural logic locking. Functional logic locking aims to protect the functionality of an IC. The design of an IC can be simplified into basic functionality or logic behavior. An IC is designed to perform certain tasks depending on the input vectors to the circuit. An adversary can use the functional design of a circuit to produce another copy or clone of the IC. This leads to IP piracy and overproduction. Functional logic locking aims to protect the functional design by inserting key gates in a way where only the correct key value reveals the true behavior of the design. Structural logic locking aims to protect the structure of the IC. The wires and connections between logic components in a netlist determine the structure of an IC. The structural information can help an attacker perform side channel attacks, probe attacks, and hardware Trojan attacks. All of these attacks exploit the

connectivity of wires in a design. Structural logic locking will be used to hide how modules are connected in a design. This will help to prevent attackers from isolating specific components and prevent certain attacks from executing. Developed logic locking techniques usually combine some form of structural and functional logic locking to create a protection scheme that maximizes the security of the system.

B. Threat Model

Logic locking is based on using a secret key to protect the circuit design. The secret key unlocks the correct functionality of the circuit. An attacker can be located anywhere on the supply chain including the foundry or at the end user [3]. The attacker's main objective is to determine the secret key used in logic locking. The secret key also allows for the attacker to determine both the functional and structural design of the design and allows them to perform IP piracy, overproduction, or hardware Trojan insertion. Most attacks require several components to uncover the secret key. First, an encrypted design or netlist is needed as the main victim of an attack. This encrypted design can come from the IC design house or reverse engineering. The second component is a functionally correct IC that can be obtained from the open market. The functionally correct IC is used to verify behavior or tested key bits during the attack algorithm. Some advanced attacks do not require a functionally correct IC and can determine the secret key from an encrypted netlist. This threat model is the basic framework for logic locking attacks.

C. Logic Locking Metrics

Several metrics are used to evaluate the success of a logic locking technique. If the technique meets the standard of these metrics, it is considered a strong scheme against certain attacks [3]. The first metric is correctness. The circuit produced from a logic locking technique should produce the correct output if the correct secret key is applied. If the technique does not do this, the design is considered defective and violates design rules. The next metric is resilience against specific attacks. For the SAT attack, a logic locking technique should prevent the SAT attack from determining the correct key. Other evaluations of this metric include increasing the number of iterations the SAT attack needs to produce the key and increasing the amount of time needed to execute the attack. Entanglement is another metric needed to ensure the success of a logic locking technique. An attacker should not be able to remove the key gates inserted by logic locking to reveal the natural design. This metric can be met by using structural obfuscation or other techniques to entangle the wires and connections of key gates into the netlist. The final metric is overhead. Overhead refers to the amount of additional area, delay, and power required when key gates are added to the circuit. Additional logic that supplements the key gates are also considered in overhead. A strong logic locking technique should have minimal overhead without compromising the security of the design.

IV. Probability Based Logic Locking

Probability based logic locking or ProbLock is a novel functional logic locking technique based on filtering out nodes in a circuit to find the best location to insert key gates. The final stage of the filtering process uses the probability of gate outputs to determine where to insert key gates. ProbLock is an algorithm where the key gates are either XOR or XNOR gates and a key is used to unlock the circuit. We used four constraints to determine the best candidate nodes to insert our XOR or XNOR key gate; **longest path, non-critical path, low dependent nodes, and best probability nodes**. The first three constraints find the set of nodes that lie on the longest path, non-critical path and have low dependent wires. The last constraint uses probability to find the set of nodes equal to the key length where we will insert the key gates. We chose the longest path and non-critical path constraint to avoid critical timing elements and to insert key gates on parts of the circuit that was being used the most. We chose the low dependent wires and probability constraint to determine locations where the output would be changed the most. This would make it harder for an attack to generate the golden circuit using an oracle based attack. Once we determine the location of the key nodes, we can insert key gates into the netlist and re-synthesize the circuit. In Equation 1 the candidate nodes are determined from a function of all four constraints. *LP* is

the set of nodes on the longest paths while *NCP* is the set of nodes on non-critical paths. *LD* represents the set of low dependent nodes and *P* are the set of probability nodes.

$$selectedNodes \subset P \subset LD \subset NCP \subset LP \tag{1}$$

For our obfuscation technique, we decided to lock a set of combinational and sequential circuit netlists using the ISCAS '85 and ISCAS '89 circuit benchmarks [19] [20]. We obfuscated a total of 40 benchmarks using ProbLock. For some of the constraints, we had to use an unrolling technique described in [21] to accurately filter out nodes. This unrolling technique was only used in sequential circuits to simplify the concepts of flip flops and other sequential logic. The sequential logic can be replaced by the main stage and a k number of sub-stages depending on the number of times unrolled. This results in a k-unrolled circuit that has the same functionality as the regular circuit. For this process, we generated a set of unrolled ISCAS '89 benchmarks which we used in some constraint algorithms. We unrolled these circuits once to prevent inaccuracies in constraints such as the longest path and non-critical path. Finally, we integrated a custom low overhead anti-SAT block based on an established anti-SAT block method [24].

A. Longest Path Constraint

The longest path constraint isolates a subset of nodes that lie on the longest paths in a circuit netlist. The subset of nodes is different for each circuit and is a function of the key length determined for each circuit. We represent the netlist of each benchmark as a directed acyclic graph (DAG) and perform the longest path analysis on each DAG. Each vertex in the DAG is a gate element from the netlist and each vector represents the wire connecting to the next gate element. Once the DAG is constructed for each benchmark, we calculated the longest paths of the DAG using a depth first search (DFS) technique. We then calculate the next longest path to generate a subset of nodes along the longest paths. Each unique node in the longest path gets added to a subset during each iteration until the size of the subset is bigger than two times the key length for that circuit. The structure of this theory is shown in Algorithm 1 which uses the DFS in Algorithm 2. Figure 12 shows the longest path for the circuit to be 3 since there are 3 gates between input A and output Y. The next longest path would also be 3 from input B to output Y. All of the nodes along both longest paths would be added to a subset of the longest path nodes. Once this subset of longest path nodes is determined, that subset gets used in the next filtering constraint. This subset can be adjusted to include more or fewer nodes depending on other filtering constraints. If more nodes are needed, this constraint is the first to be modified.

We chose to use the longest path constraint to counter oracle guided attacks. Oracle guided attacks will query the IC with various inputs and observe the output. This gives the attacker information about how the circuit behaves and the adversary can use this information to determine the secret key. We want to insert key gates where most of the logic and activity occur in the circuit. An oracle guided attack will most likely pass data through the longest paths of a circuit so we want to protect these parts of the IC by inserting key gates on the longest path.

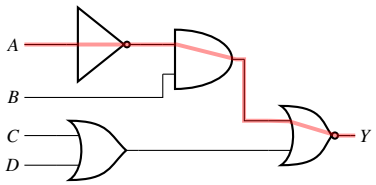


Figure 12: Longest Path (in red)

B. Critical Path Constraint

The critical path constraint is similar to the longest path; however, rather than considering logic depth, we look at timing information. This constraint is essential, as adding gates on the critical path could break the circuit functionality or change timing specifications. We used Synopsys Design Compiler (DC) [22] to compile the benchmarks and calculate the critical paths. In DC, the critical path of a circuit can

Algorithm 1: Get Longest Path

```
input  : Circuit Graph and Key Length
output : List of nodes on the longest path
G← circuit graph;
V← source vertex of G;
overallNodes ← [];
while true do
    allPaths ← DFS(G,V);
    for p in allPaths do
        if len(p) = maxLength then
            maxPath ← len(p);
        end
    end
    for p in maxPath do
        if p not in overallNodes then
            overallNodes.append(p);
        end
    end
    if len(overallNodes > keyLength * 3) then
        return overallNodes;
    end
end
```

Algorithm 2: Depth First Search

```
input  : Circuit Graph G and Vertex Source V
output : Longest path in Circuit
mark V as visited;
for all neighbors W of V do
    if W is not visited then
        DFS(G,W);
    end
end
```

be determined with several simple steps. First, the benchmark in Verilog format is imported into the software. Next, the delay timing and timing constraints are set for certain specifications. Finally, DC will calculate the critical path from a primary input to a primary output. The results are a set of nodes that represent the critical path. For this step, ProbLock calculates multiple critical paths and exports the data to be used in the filtering process. Figure 13 shows an example of the timing analysis of a basic circuit. The figure shows that the two critical paths lie from input C to output Y and from input D to output Y. The propagation timing for the critical path is 60ps as opposed to 65ps on the other paths. The nodes selected often overlapped with other constraints (e.g. the longest path was often the critical path), though oftentimes the critical path would involve gates with large fan out. Determining the critical path is largely technology-specific; different process design kits (PDKs) will have different timing information which can affect which paths are critical paths. We removed any nodes that were on the critical path from the set of nodes passed into this constraint. The resulting subset results in nodes that are on the longest, non-critical path.

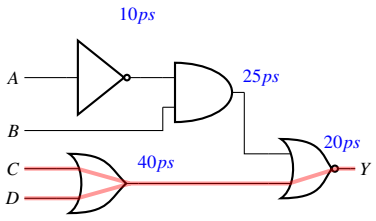


Figure 13: Critical Paths (in red)

C. Low Wire Dependency Constraint

The next constraint generates a subset of nodes that are connected to low dependent wires. The output wire of a gate is considered low dependent if the input wires to that gate have little influence on the value of output. This idea is modified from a technique called FANCI where suspicious wires can be detected in a Trojan infected design [25]. A functional truth table is created for each output wire of each gate in the circuit. The inputs of the truth table correspond to the inputs of the gate being analyzed. For each input column, the other columns are fixed and each row is tested with a 0 or 1 to determine the output. This results in two functions when setting the value to either 0 or 1. The boolean difference between these two functions results in a value between zero and one that can be further analyzed. The value for each input gets stored as a list for each output wire. We take the average value of the entire list to determine the dependency of an output wire. The algorithm logic is shown in Algorithm 3. This analysis can determine if certain inputs are low dependent or if they rarely affect the corresponding logic. Low dependent wires are weak spots in the circuit so this constraint isolates those locations to improve the security. An example of this algorithm is shown in Figure 14 and 15. The multiplexer with a suspicious wire has direct influence over the logic of the circuit. We insert key gates next to low dependent wires to fortify any weaknesses. The filtering process passes the subset of nodes to the final constraint.

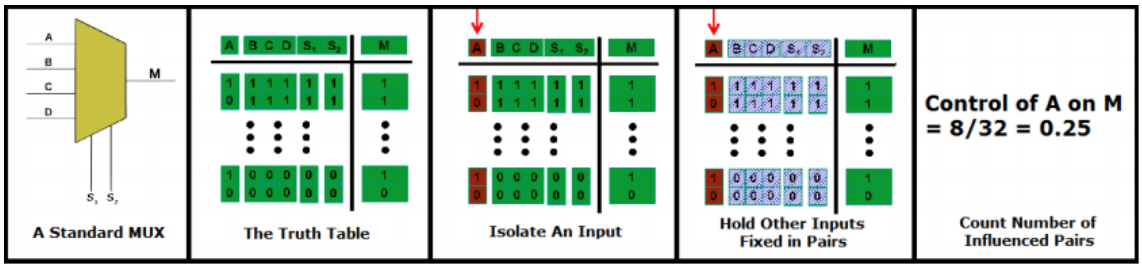


Figure 14: Standard Multiplexer with Calculation of Dependency Control [25]

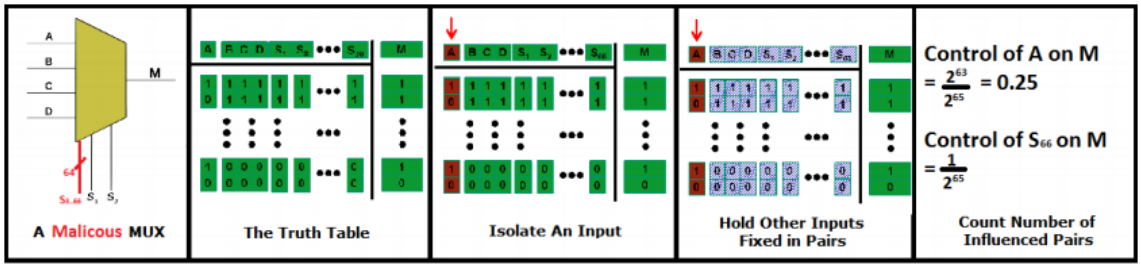


Figure 15: Malicious Multiplexer with Calculation of Influence from Suspicious Wire [25]

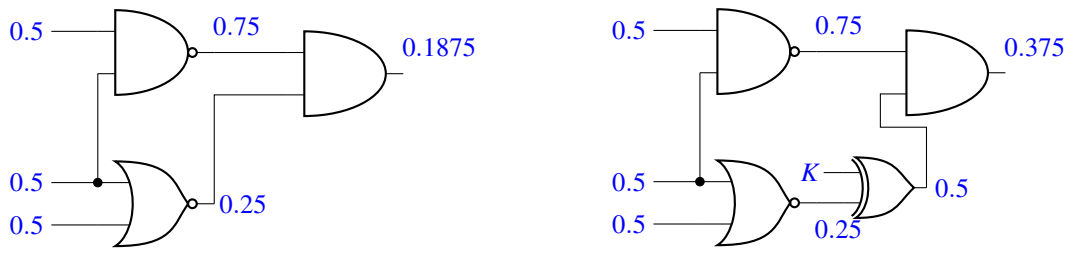
Algorithm 3: Find Low Dependent Wires

```
input  : Circuit Graph
output : List of low dependent wires
G ← circuit graph;
foreach gates in G do
    foreach output wire w do
        T ← Truthtable(w);
        L ← empty list of control values;
        foreach column c in T do
            count ← 0;
            foreach row r in T do
                x0 ← Value of w when input value = 0;
                x1 ← Value of w when input value = 1;
                if x0 != x1 then
                    count++;
                end
            end
            L.append count/size(T);
        end
        avg ← average(L));
        if avg < 0.5 then
            controlNodes.append(gate);
        end
    end
end
```

D. Biased Probabilities Constraint

The probability constraint focuses on reducing the effectiveness of the SAT attacks. In a SAT attack, a distinguishing input (DI) is chosen and the attacker runs through various key values, eliminating any which yields an incorrect output. Thus, to reduce the effectiveness of a SAT attack, the number of wrong keys produced for a given DI must decrease. This can be done by bringing the probability of any given node being 1 closer to 0.5, since any node which is biased towards 0 or 1 will propagate through to the output nodes, making it easier for SAT attacks to eliminate key values. Since a two-input XOR/XNOR has an output probability of 0.5, we can insert our key gates at nodes heavily biased towards 0 or 1 and "reset" the probability to 0.5.

The algorithm used to obtain the N nodes with the most biased probabilities is shown in Algorithm 4. The output probabilities of a gate are dependent on the type of logic gate and the probabilities of its inputs. The probability of incoming input wires will influence how the output probability is calculated for each gate type. It is worth noting that while generating node probabilities for combinational circuits is trivial, sequential circuits pose a potential problem because of the D flip flops (DFFs). However, giving the DFF outputs a starting probability of 0.5 and propagating running a few iterations (three is sufficient) will asymptotically approach the correct probability for the DFF node.



(a) Pre-Insertion Probabilities

(b) Post-Insertion Probabilities

Figure 16: Key Gate Insertion Probabilities

Algorithm 4: Find Biased Nodes

```
input  : Circuit Graph and Circuit Input Probabilities
output : List of N most biased nodes
for i ← 1 to N do
    DFF initial probability←0.5;
    while any probability unknown do
        foreach node with unknown probability do
            if all node input probabilities known then
                Compute node output probability;
            end
        end
    end
    Node ← max(abs(circuitprobabilities − 0.5));
    Add Node to output list;
    Insert XOR/XNOR gate at Node in Circuit Graph;
end
```

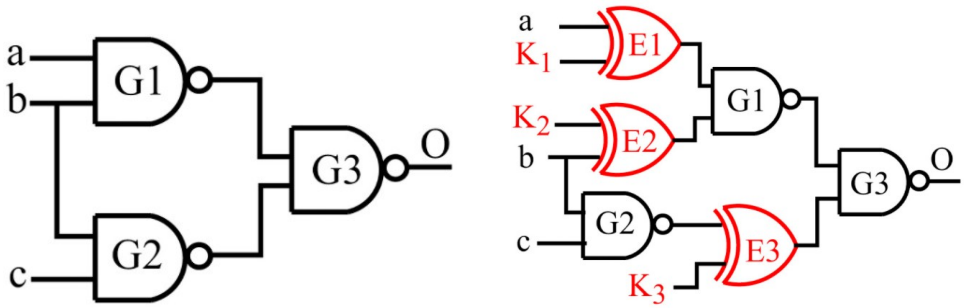
An example of this is illustrated in Figure 16. Figure 16 a shows a sample circuit with each node annotated with the probability of that node being a logic 1. The output shown is heavily biased toward logic 0, which makes it more susceptible to SAT attacks. Strategically adding a key gate, as shown in Figure 16 b brings the output probability closer to 0.5, reducing the effectiveness of the SAT attack.

This idea of using probability calculations for logic locking is a novel concept that has not been tested yet. After the first three filtering constraints, the ProbLock algorithm has a set of nodes that lie on the longest path, non critical path and are low dependent. From this set of nodes, we generate a new subset equal to the bit size of the key value based on the biased probabilities of each node. The final set of nodes should produce the best location to insert key gates for a circuit netlist. This is the final step for the ProbLock algorithm that results in a low overhead locked benchmark.

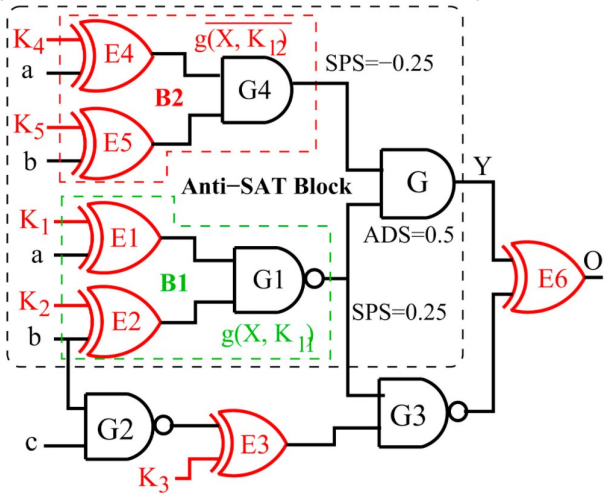
E. Anti-SAT

ProbLock is a low overhead logic locking technique that aims to counter SAT based attacks on integrated circuits. However, other anti-SAT techniques have been developed to be provably effective against newer SAT attacks. The lightweight anti-SAT block (ASB) in [26] is a modification of the regular anti-SAT block in [24] that reuses overlapped key gates from the logic locking technique in the new lightweight anti-SAT block. The regular ASB described in [24] is a low overhead module that can connect to wires and key inputs of the original locked circuit. This block exponentially increases the attack time and iterations of a SAT attack. The ASB is composed of additional key gates and logic components that are integrated with the original logic of the locked circuit. The lightweight ASB minimizes overhead by creating an ASB that reuses the key gates from the original locked circuit and only adds the additional key gates to complete the full ASB. This method reduces the overhead of the ASB while maintaining the same level of security against SAT attacks. In Figure 17a, an example circuit is shown. Figure 17b shows the locked version of that circuit by inserting key gates $E1$, $E2$, and $E3$. In Figure 17c, the lightweight ASB implementation is shown by reusing key gates $E1$, $E2$, $E3$, and adding gates $E4$, $E5$, $E6$, $G4$, and G, to create the full ASB. For this specific example, the overhead is reduced by almost 50% and maintains the effectiveness against a full SAT attack.

We used the same idea to integrate a strong anti-SAT solution into ProbLock. During the ProbLock algorithm, the best nodes are selected to be candidates for inserted key gates. After the key gates are established, the algorithm parses for patterns that would be suitable to construct a lightweight ASB. If a combination of key gates can be used to create a lightweight ASB, those nodes are flagged until later. Once all of the overlapped nodes are determined, the final step of ProbLock constructs a final locked circuit with an integrated ASB.



(a) Original Circuit [26] (b) Logic Locked Circuit [26]



(c) Lightweight Anti-SAT Block Circuit [26]

Figure 17: Comparison of lightweight anti-SAT block integrated with logic locking [26]

V. Experimental Procedure

A. Objective and Evaluation Metrics

The purpose of the experimental procedure is to evaluate ProbLock against known SAT attacks and compare the effectiveness of the technique against known logic locking schemes. We encrypted 40 different combinational and sequential circuit benchmarks from the ISCAS' 85 and ISCAS '89 suite. Each of these locked circuits was evaluated against a SAT attack from 2015 [13]. We compared and analyzed the results against other logic locking techniques such as SLL, logic cone analysis, and random logic locking [7] [27].

Several metrics are used to evaluate the strength of ProbLock and the effectiveness against SAT attacks. First, the complexity of ProbLock itself should be very efficient. ProbLock includes multiple algorithmic steps that have various complexity. The most inefficient step has a complexity of $O(n^2)$. Even with the inefficiencies of ProbLock, we were able to encrypt the largest benchmarks within minutes. The next metric is overhead which is the measure of how much additional space, power, and energy is added by the encryption algorithm. ProbLock aims to add security to the ISCAS benchmarks by only adding a maximum of 10% overhead. The overhead is calculated by the number of additional key gates divided by the total number of gates in a benchmark. After optimizing the logic locking algorithm, the overhead of the ASB was also minimized to be included within the 10% maximum overhead.

To evaluate the effectiveness of ProbLock against a SAT attack, we used a public SAT algorithm from [13] to attack the encryption technique. The SAT attack takes real time and iteration cycles to determine the key used to unlock the circuit. The number of iterations is the main metric used to evaluate how well the SAT attack performs. If the number of iterations is low, then the SAT attack is very effective against the benchmark and can break the protection scheme quickly. If the SAT attack needs a large number of iterations, then the benchmark has higher security and is more effective against SAT attacks. There is

also the case where the SAT algorithm cannot determine the correct key in a given time and therefore the circuit has the highest security against SAT attacks.

B. Setup and Procedure

The first step to implement ProbLock is to gather the benchmarks for testing and analysis. We used combinational and sequential benchmarks from the ISCAS ’85 and ISCAS ’89 suite which was in Verilog format [19] [20]. Figures 18 and 19 shows an example of the C17 benchmark used in this step. To obfuscate the benchmarks, we created a Python script that implemented the ProbLock algorithm. The script takes in a benchmark netlist in Verilog format and returns an obfuscated netlist in the same format. The obfuscated netlist included the key gates inserted as well as the key defined to unlock the circuit. We created a function to parse each netlist for information. The information was organized into lists of inputs, outputs, and gate types. We used this information to determine the key size relative to the number of gates in a netlist. We also created functions for each constraint in our algorithm. A set of overall nodes was passed through each function and then narrowed down to a set of best nodes for key gate insertion. Another function was created to insert key gates from a data structure into a new netlist. We specified the key inputs, key gates, and the key value in the header of the new netlist for development purposes. A final function was used to calculate and construct the ASB for ProbLock. We reformatted the benchmarks and refactored the Python code when needed during this stage. Throughout the development process, we ran tests to verify the intention of our script and to make sure each new netlist was correct.

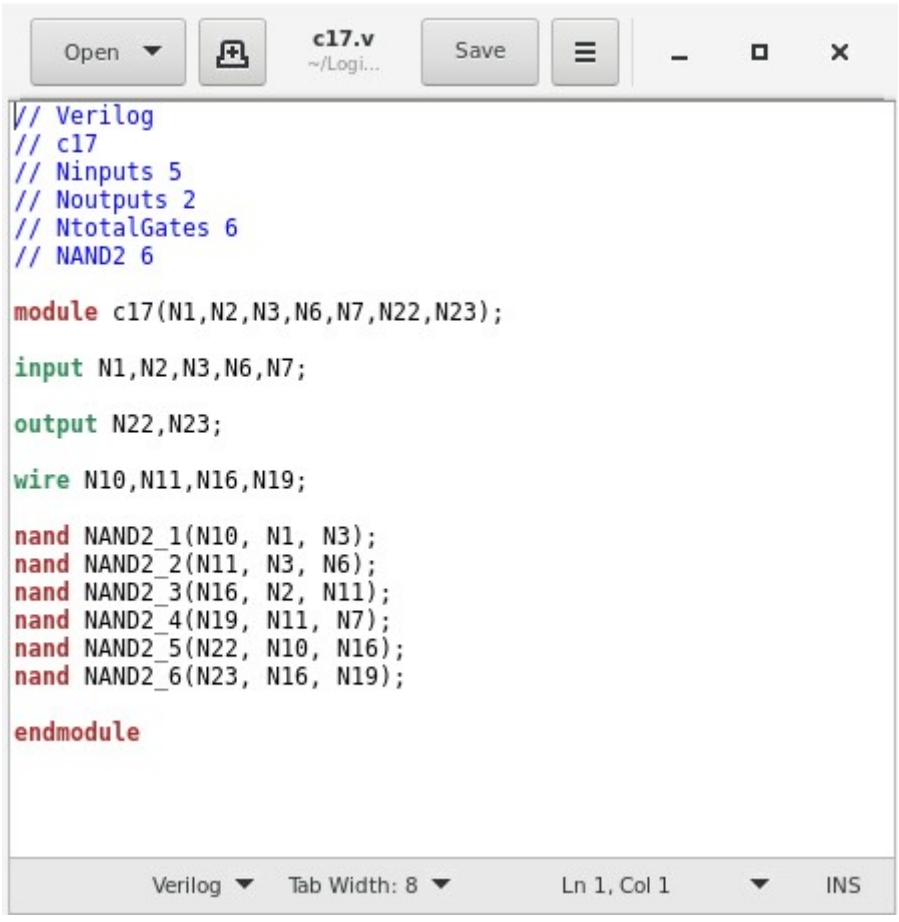


Figure 18: ISCAS ’85 C17 Circuit Netlist

We used Synopsys Design Compiler to synthesize and view the netlists before and after obfuscation [22]. The Synopsys tool allowed us to see the gate level representation of each benchmark during the analysis process. We were able to see the location of the logic components as well as the inserted key gate after the obfuscation process occurred. We also used the Design Compiler for critical path analysis in our second constraint. The tool allowed for timing analysis between different logic components of the netlist.

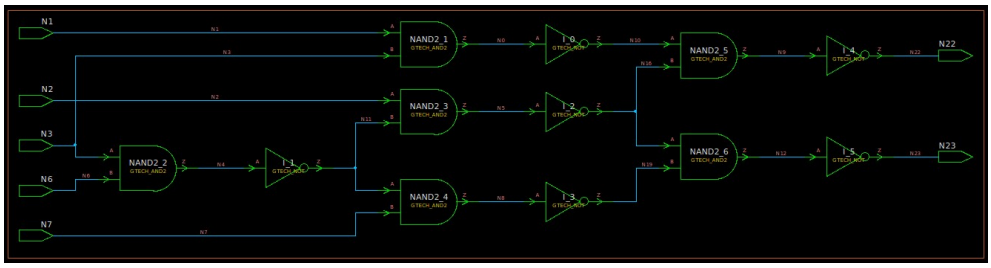


Figure 19: ISCAS ’85 C17 Circuit Schematic

We used this to calculate the critical paths for our constraint and removed any nodes that lie on this critical path. An example of the critical path for the c17 ISCAS ’85 benchmark is shown in Figure 20. We integrated the results from Synopsys by passing it through a text file that gets parsed in the main script. Finally, we used the Design Compiler to structurally obfuscate our benchmarks. ProbLock does not focus on structural obfuscation so we used a simple solution to add this component to the algorithm. Resynthesis of a circuit will change the wires and connections in a netlist which provides a minimal amount of structural obfuscation. We used the Design Compiler to resynthesize after functionally obfuscating the benchmarks with ProbLock. We did not analyze the strength of the structural obfuscation since SAT attack is a functional attack. This step was performed for good practice and to provide a framework for future analysis.

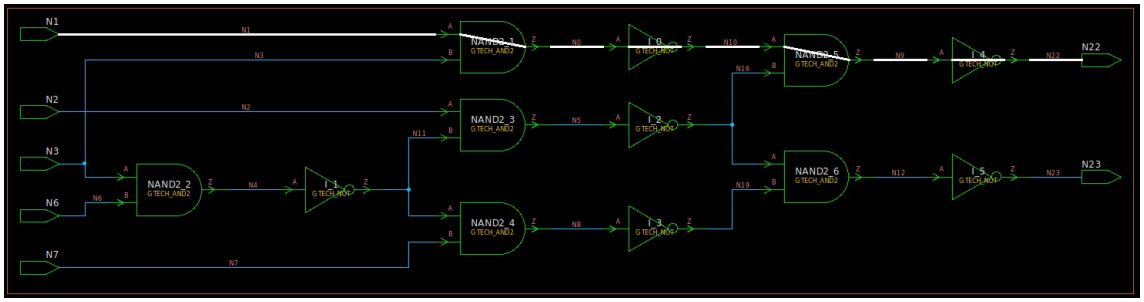


Figure 20: Using DC to Determine Critical Path of ISCAS ’85 c17

We considered multiple SAT attacks that were publicly available or open sourced. Lingelinge is well renowned SAT algorithm that won multiple SAT competitions [23]. Lingelinge is part of a suite developed by Armin Biere that has solved multiple SAT based problems. Lingelinge is a strong SAT solver but there was no specific application for circuits and netlists. The SAT solver provided by [13] is another simple program that we ran on a Linux machine to attack the benchmarks. This SAT solver was used in earlier papers to evaluate the SAT resistance of logic locked circuits. Since the application is the same for our experiment, we decided to use the [13] SAT solver for our experiment. The program requires access to an oracle in the form of the original unlocked benchmark as well as the locked benchmark. The two benchmarks are inputs to the SAT attack program, and the output results are either a key to unlock the circuit or an un-SAT model. The SAT attack program also shows the evaluation time in seconds and the number of iterations needed to execute the attack. An example of the SAT attack running is shown in Figure 21. For a small simple circuit, the execution time is a fraction of a second and only takes one iteration. We ran the attack on all of the ISCAS benchmarks and analyzed the results to show how effective ProbLock was against SAT attacks.

C. Results

During the development process of ProbLock, we analyzed the correlation and relationship between constraints. We chose two constraints based on path elements and two constraints based on nodes and wires. Due to this design, we were able to analyze the correlation between constraints and adjust the strength of the filtering process based on this analysis. Overall, we wanted the correlation between constraints to be large enough to remove any nodes that didn’t belong in both sets. This would allow the filtering process from each constraint to generate a subset of nodes each time until only the best

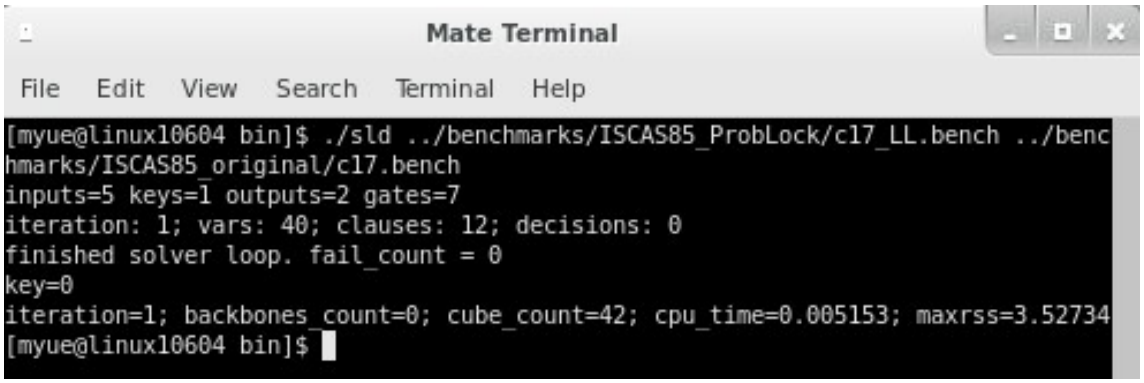


Figure 21: SAT Evaluation of ISCAS ’85 c17

candidate nodes remain to be inserted. The strength of the correlation varies between benchmarks because of the shape and functionality of each circuit. Each subsequent constraint filtered out a set of nodes based on the relationship between the constraint and the overall set of nodes. Table I shows the experimental correlations for ISCAS ’85 and ’89 benchmarks. We only show some of the results in the table as all 40 benchmarks are not included. The longest path (LP) length and critical path (CP) count shown in blue are based on the path constraints. The rest of the categories including non-critical path (NCP), low depending wires (LD), and biased probabilities (Prob) are based on nodes. In the ISCAS ’85 benchmark suite, the correlation between nodes on the longest path, and the original set of nodes was 36% on average. Between nodes on the non-critical path and nodes on the longest path, the correlation was about 63% on average. The correlation between low dependent nodes and nodes on the non-critical path was about 73% and the final correlation between biased probabilities and low dependant nodes was about 65%. For the ISCAS ’89 benchmark suite, the correlation between the longest path and overall nodes is 27%. The correlation between the critical path and the longest path is 84%. The correlation between low dependent nodes and non-critical path is 85% and the final correlation between biased probabilities and low dependent nodes is 45%. The numbers that we analyzed were ideal for the filtering process. Enough nodes were removed with each subset until the final set of best candidates were discovered. For the final biased probabilities constraint, the final set of nodes was equal to the size of the key. For the other constraints, we adjusted the filtering threshold accordingly. Depending on the situation, the strength of the constraints can be adjusted which allows flexibility in our algorithm.

After analyzing and adjusting the constraints of ProbLock, we evaluated the security of combinational ISCAS ’85 circuits against a SAT attack model. The toc13xor and dac12 benchmarks are encrypted using fault analysis and hamming distance techniques to determine the insertion of key gates [13]. LCSB is a logic locking benchmark that inserts key gates based on the logic cone’s fan-in and fan-out value of each gate [6]. Finally, ProbLock is the logic locking technique we developed based on gate probability and a lightweight ASB unit. We ran the SAT attack presented in [13] on each combinational benchmark 5 times and averaged the results in Table II. The decryption time is the amount of time the SAT attack takes to determine the correct secret key to unlock the circuit. The decryption iteration is the number of iterations that the SAT attack takes to determine the secret key. The longer decryption time and iteration imply that the benchmark is more resistant to SAT attacks. If the SAT attack cannot determine the correct key in 30 minutes, the benchmark is determined unbreakable. The most secure benchmarks will take the longest time for the SAT attack to complete. As shown in the table, the toc13xor, dac12, and LCSB benchmarks have a lower decryption time and iteration number as opposed to ProbLock. ProbLock is significantly more secure for most circuits. All encryption techniques used different techniques to limit overhead, so for every encryption technique, the overhead was maintained between 5% and 8%.

D. Evaluation and Analysis

After running the SAT attack on various combinational benchmarks, we saw that ProbLock had great security compared to other logic locking techniques. The benchmarks presented in [13] are older and more simplified techniques. These techniques do not implement any ASB unit and are easier to implement. During the early developments of ProbLock, we did not integrate an ASB unit and only used probability

Table I: ISCAS '85 & '89 Constraint Correlation

ISCAS 85	Key Size	LP Length	CP Count	Total Nodes	LP Subset	NCP Subset	LD Subset	Prob Subset
c432	16	18	7	160	88	60	33	16
c499	16	12	32	202	186	104	99	16
c1355	32	25	32	546	485	253	53	32
c1908	64	39	25	880	205	145	129	64
c2670	64	31	100	1269	217	200	75	64
c3540	128	42	22	1669	260	173	151	128
c5315	128	47	100	2307	411	226	180	128
c7552	256	35	100	3513	532	341	278	256
ISCAS 89	**Key Size**	**LP Length**	**CP Count**	**Total Nodes**	**LP Subset**	**NCP Subset**	**LD Subset**	**Prob Subset**
s298	8	10	6	75	26	26	18	8
s344	8	21	11	101	21	19	19	8
s382	8	10	6	99	29	29	21	8
s386	8	12	7	118	49	32	24	8
s400	8	10	6	106	30	30	22	8
s444	8	12	6	119	38	38	30	8
s641	8	75	24	107	80	53	51	8
s713	8	75	23	139	84	61	56	8
s838	16	18	1	288	45	29	26	16
s1238a	32	23	14	428	132	76	64	32
s1488	32	18	19	550	89	60	48	32
s5378a	64	15	46	1004	134	96	92	64
s9234a	128	19	37	2027	264	261	213	128
s13207a	256	28	100	2573	573	521	482	256
s15850a	256	22	100	3448	553	544	506	256
s38584	256	15	100	11448	717	716	571	256

Table II: ISCAS '85 SAT Evaluation

	toc13xor (5-8%) Encryption		dac12 (5-8%) Encryption	
ISCAS '85 Benchmark	**Decryption time (s)**	**Decryption iteration**	**Decryption time (s)**	**Decryption iteration**
c17	n/a	n/a	n/a	n/a
c432	0.057259	3	0.051281	1
c499	0.1015	1	0.115953	5
c880a	0.144556	5	0.157413	25
c1355	0.188653	1	0.393672	2
c1908	3.93094	2	0.673176	29
c2670	0.285035	14	unbreakable	unbreakable
c3540	1.20809	13	3.51718	65
c5315	0.947552	9	9.74864	27
c6288	n/a	n/a	n/a	n/a
c7552	1.51212	10	28.5707	88

	LCSB (5-8%) Encryption		ProbLock (5-8%) Encryption	
ISCAS '85 Benchmark	**Decryption time (s)**	**Decryption iteration**	**Decryption time (s)**	**Decryption iteration**
c17	0.015533	1	0.005433	1
c432	0.05647	3	0.10256	3
c499	0.116477	3	0.2273	9
c880a	0.178235	11	1.6596	43
c1355	0.05869	1	unbreakable	unbreakable
c1908	2.66502	31	2.0384	54
c2670	2.66502	74	145.8	245
c3540	1.51982	18	8.291	82
c5315	4.65455	73	94.024	178
c6288	6.01653	3	1.2284	31
c7552	3.08791	29	674.5	561

based constraints for logic locking. These results are compared equally to existing basic logic locking techniques with the same overhead and same complexity. Due to this analysis, we decided to integrate an ASB unit with minimal additional overhead. This increased the security with only a slight cost of overhead and complexity. We also ran an incomplete experiment and compared ProbLock to a full ASB logic locking scheme [24] and SarLock [8]. Only some ISCAS '85 benchmarks were used in both studies so there is no equivalent comparison for our experiment. The experiments used in other studies had a larger key value and greater overhead than ProbLock. They also proved to have better security metrics due to the bigger key size. In the future, we aim to optimize ProbLock to increase the security metrics and compare it to the best logic locking research currently published.

We also did not complete the experiment for sequential circuits. For sequential circuits, we decided to unroll each flip flop used in the benchmark to create a fully combinational circuit. For the probability constraint, this was a necessary step because the probabilities change for each loop of the flip flop. This also led to inconsistent results during the SAT attack because the number of unrolled flops was different for each circuit. Without a consistent way to test the evaluation of sequential circuits, we were not confident in the experimental results. In the future, we want to test the security of ProbLock on sequential circuits as well as combinational circuits.

During the experimental phase, we also tested large key size and observed the trend of security increasing with larger key size. The main issue with a large key size is the overhead. The overhead is an additional circuit that a designer must include to protect their original design. This will cost the design more time, power, and money. This is why ProbLock was implemented with an overhead metric of less than 8%. Future implementations of ProbLock aim to reduce the overhead to a maximum of 5% with the same or improved level of security.

VI. Conclusion

In this paper, we presented ProbLock, a novel logic locking technique using gate probabilities in circuit netlists. We explored this idea and tried to optimize the performance and overhead of ProbLock. ProbLock uses a filtering process to determine the best set of nodes to insert key gates. The filtering process is based on several constraints including longest path, non-critical path, low dependent wires, and biased gate probabilities. ProbLock also integrated an ASB unit to further enhance its resistance to SAT attacks. We evaluated the strength of ProbLock by attacking the locked ISCAS '85 combinational benchmarks with a SAT attack and recorded the results. Compared to early logic locking techniques, ProbLock is more effective against SAT attacks. Compared to newer and more complex logic locking, ProbLock can be improved for better results. There are still tasks that need to be done in the future to increase the security metrics of ProbLock and to complete comprehensive testing. Overall, we show that ProbLock can be a simple way to improve hardware security through logic locking.

REFERENCES

[1] D. Zhang, X. Wang, M. Tauhidur Rahman, and M. Tehranipoor, "An On-Chip Dynamically Obfuscated Wrapper for Protecting Supply Chain Against IP and IC Piracies", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2018, pp. 2456-2469.

[2] S. Bhasin, J. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware Trojan Horses in Cryptographic IP Cores", in Proceedings - 10th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013, pp. 15-29.

[3] D. Forte, S. Bhunia and M. Tehranipoor, Hardware Protection through Obfuscation, 1st ed. Springer, 2017.

[4] C. W. Hsieh, C. Y. Lin, Z. Li, T. Y. Ho, "IP Protection for Digital Microfluidic Biochips", in The 28th VLSI Design/CAD Symposium (VLSI-CAD), 2017.

[5] M. Yasin and O. Sinanoglu, "Evolution of logic locking," 2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Abu Dhabi, 2017, pp. 1-6, doi: 10.1109/VLSI-SoC.2017.8203496.

[6] Y. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," 2015 16th Latin-American Test Symposium (LATS), Puerto Vallarta, 2015, pp. 1-6, doi: 10.1109/LATW.2015.7102410.

[7] M. Yasin, J. J. V. 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, Sept. 2016, doi: 10.1109/TCAD.2015.2511144.

[8] M. Yasin, B. Mazumdar, J. J. V. Rajendran and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2016, pp. 236-241, doi: 10.1109/HST.2016.7495588.

[9] M. Yasin, B. Mazumdar, J. J. V. Rajendran and O. Sinanoglu, "TTLock: Tenacious and traceless logic locking," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, 2017, pp. 166-166, doi: 10.1109/HST.2017.7951830.

[10] A. Sengupta, M. Nabeel, M. Yasin and O. Sinanoglu, "ATPG-based cost-effective, secure logic locking," 2018 IEEE 36th VLSI Test Symposium (VTS), San Francisco, CA, 2018, pp. 1-6, doi: 10.1109/VTS.2018.8368625.

[11] M. Yasin, A. Sengupta, M. Thari Nabeel, M. Ashraf, J. J. V. Rajendran and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory To Practice", in CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Texas, 2020, pp. 1601-1618.

[12] J. Mellor, A. Shelton, M. Yue, and F. Tehranipoor, "Attacks on logic locking obfuscation techniques", in 2021 IEEE International Conference on Consumer Electronics (ICCE), 2021, pp. 1–6.

[13] 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), 2015, pp. 137–143.

[14] 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), McLean, VA, 2017, pp. 95-100, doi: 10.1109/HST.2017.7951805.

[15] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2514-2527, 2020, doi: 10.1109/TIFS.2020.2968183.

[16] F. Yang, M. Tang and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) – Unlocked," in IEEE Transactions on Information Forensics and Security, vol. 14, no. 10, pp. 2778-2786, Oct. 2019, doi: 10.1109/TIFS.2019.2904838.

[17] F. Yang, M. Tang and O. Sinanoglu, "Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-hd) - 2013 Unlocked," in IEEE Transactions on Information Forensics and Security, vol. 14, no. 10, pp. 2778-2786, Oct. 2019, doi: 10.1109/TIFS.2019.2904838.

[18] M. Yasin, C. Zhao and J. J. V. Rajendran, "SFLL-HLS: Stripped-Functionality Logic Locking Meets High-Level Synthesis," 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 2019, pp. 1-4, doi: 10.1109/ICCAD45719.2019.8942150.

[19] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in Proc. of the International Symposium on Circuits and Systems, 1985, pp. 663-698.

[20] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in Proc. of the International Symposium of Circuits and Systems, 1989, pp. 1929-1934.

[21] N. Miskov-Zivanov and D. Marculescu, "Modeling and Optimization for Soft-Error Reliability of Sequential Circuits," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 5, pp. 803-816, May 2008, doi: 10.1109/TCAD.2008.917591.
[22] Design Compiler Graphical. Synopsys, 2018.
[23] A. Biere, "Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016", in Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions, volume B-2016-1 of Department of Computer Science Series of Publications B, University of Helsinki, 2016, pp 44–45.
[24] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT Attack on Logic Locking," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 2, pp. 199-207, Feb. 2019, doi: 10.1109/TCAD.2018.2801220.
[25] A. Waksman, M. Suozzo and S. Sethumadhavan, "FANCI: Identification of Stealthy Malicious Logic Using Boolean Functional Analysis," in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 697-708, Sept. 2013, doi: 10.1145/2508859.2516654.
[26] V. Rathor, B. Garg, G. Sharma, "New lightweight Anti-SAT block design and obfuscation technique to thwart removal attack", in Integr., vol. 75, pp. 178-188, 2020.
[27] S. Amir, B. Shakya, X. Xu, Y. Jin, S. Bhunia, M. Tehranipoor, D. Forte, "Development and Evaluation of Hardware Obfuscation Benchmarks," J Hardware System Security 2, pp. 142–161, 2018, doi:10.1007/s41635-018-0036-3