6-30-2020

Hardware Intellectual Property Protection Through Obfuscation Methods

Jason Blocklove
jmb8314@rit.edu

Hardware Intellectual Property Protection Through Obfuscation Methods

Jason Blocklove

Hardware Intellectual Property Protection Through Obfuscation Methods

Jason Blocklove
June 30, 2020

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

Hardware Intellectual Property Protection Through Obfuscation Methods

Jason Blocklove

Committee Approval:

Dr. Marcin Łukowiak *Advisor* Date
Department of Computer Engineering

Dr. Stanisław Radziszowski Date
Department of Computer Science

Dr. Michael Kurdziel Date
L3Harris Technologies

To all of my family and friends who helped me on this journey

Acknowledgments

I would like to thank my family for their immense support throughout my whole college career. Without them I could not have accomplished this milestone and I cannot thank them enough, especially my parents, Jon and Linda, and my sister Katie.

I would like to thank all of the friends I've made throughout the years. So many people throughout my time at RIT have helped to push me forward and I would not have made it here without them.

I would like to thank my many friends who worked alongside me in the Applied Cryptography and Information Security lab including Stephanie Soldavini, Braeden Morrison, Eric Scheler, Kevin Millar, and Prathibha Rama. The insights and help I received were invaluable in the completion of this work and I wish them all the best in their various endeavors.

Finally, I would like to thank all of the professors and advisers I've had the pleasure of working with throughout the years. I'd like to thank Professor Cliver, who helped truly inspire my love of teaching and my goals to always be a better TA.

Thank you especially to the members of my committee, without whom none of this work would have been possible: Dr. Kurdziel for his help throughout the project, Dr. Radziszowski for his incredible expertise in dealing with the mathematics behind cryptography, and Dr. Łukowiak for helping me really understand the research process and always being available for help.

Abstract

Security is a growing concern in the hardware design world. At all stages of the Integrated Circuit (IC) lifecycle there are attacks which threaten to compromise the integrity of the design through piracy, reverse engineering, hardware Trojan insertion, physical attacks, and other side channel attacks — among other threats. Some of the most notable challenges in this field deal specifically with Intellectual Property (IP) theft and reverse engineering attacks. The IP being attacked can be ICs themselves, circuit designs making up those larger ICs, or configuration information for the devices like Field Programmable Gate Arrays (FPGAs). Custom or proprietary cryptographic components may require specific protections, as successfully attacking those could compromise the security of other aspects of the system. One method by which these concerns can be addressed is by introducing hardware obfuscation to the design in various forms. These methods of obfuscation must be evaluated for effectiveness and continually improved upon in order to match the growing concerns in this area.

Several different forms of netlist-level hardware obfuscation were analyzed, on standard benchmarking circuits as well as on two substitution boxes from block ciphers. These obfuscation methods were attacked using a satisfiability (SAT) attack, which is able to iteratively rule out classes of keys at once and has been shown to be very effective against many forms of hardware obfuscation. It was ultimately shown that substitution boxes were naturally harder to break than the standard benchmarks using this attack, but some obfuscation methods still have substantially more security than others. The method which increased the difficulty of the attack the most was one which introduced a modified SIMON block cipher as a One-way Random Function (ORF) to be used for key generation. For a substitution box obfuscated in this way, the attack was found to be completely unsuccessful within a five-day window with a severely round-reduced implementation of SIMON and only a 32-bit obfuscation key.

Contents

List of Figures

List of Tables

Acronyms

AES

Advanced Encryption Standard

ASIC

Application-Specific Integrated Circuit

ATPG

Automatic Test Pattern Generation

BDD

Binary Decision Diagram

BFS

Breadth First Search

BIL

Bitstream Interpretation Library

CDFG

Control and Data Flow Graph

CNF

Conjunctive Normal Form

CRP

Challenge/Response Pair

DIP

Distinguishing Input Pattern

EDA

Electronic Design Automation

FPGA

Field Programmable Gate Array

FSM

Finite State Machine

HDL

Hardware Description Language

IC

Integrated Circuit

IP

Intellectual Property

KPG

Key Programmable Gate

LUT

Look-Up Table

NIST

National Institute of Standards and Technology

NLFSR

Non-Linear Feedback Shift Register

NSA

National Security Agency

ORF

One-way Random Function

PCB

Printed Circuit Board

PIP

Program Interconnect Point

PLP

Programmable Logic Point

PSM

Parallel State Machine

PUF

Physical Unclonable Function

RTL

Register Transfer Logic

s-a-0

stuck-at-0

s-a-1

stuck-at-1

SAT

satisfiability

SEM

Scanning Electron Microscope

SLL

Strong Logic Locking

SoC

System on Chip

SPS

Signal Probability Skew

SRAM

Static Random Access Memory

STG

State Transition Graph

TEM

Transmission Electron Microscope

VHDL

Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL)

Chapter 1

Introduction

1.1 Motivation

Hardware security is the concept of protecting circuit information against many possible types of attacks, such as reverse engineering, inserting malicious hardware Trojans, or creating counterfeit devices based on the original.

These attacks can differ at the various stages of the Integrated Circuit (IC) lifecycle, and each entity involved has its own assets to protect, as well as its own possible malicious behaviors as shown in Figure 1.1.

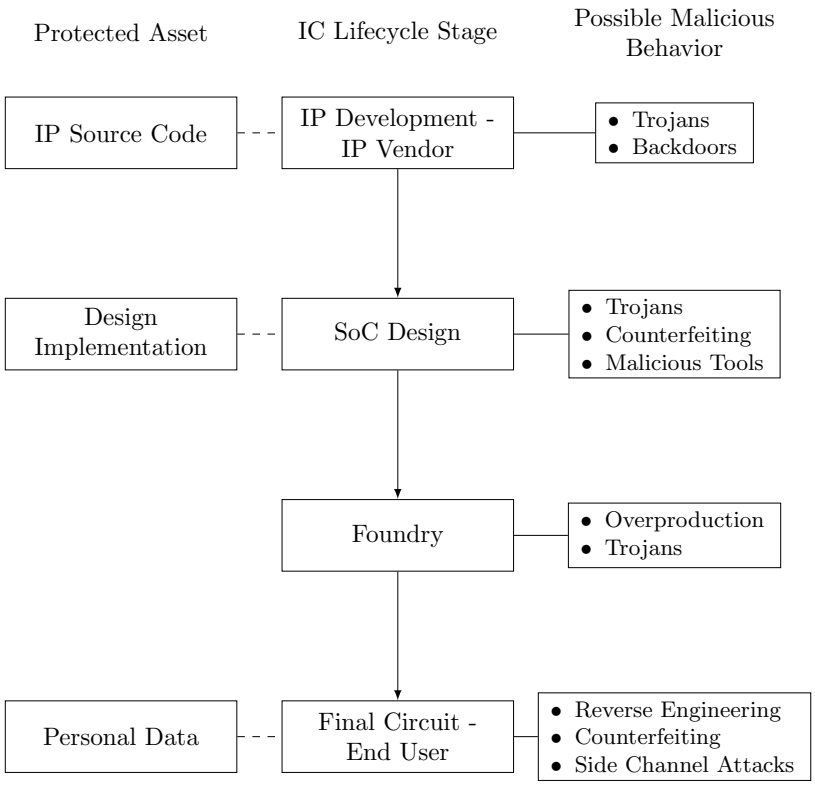


Figure 1.1: IC Lifecycle Concerns. The middle column shows the stage in the cycle, the left column represents the assets each stage wants to protect, and the right column shows the types of attack each stage can perform.

The Intellectual Property (IP) vendor is concerned with protecting the IP itself. One method of protecting their Hardware Description Language (HDL) IP is to encrypt the HDL itself before sending it to future stages. This encryption is done in such a way that tools are able to decrypt the design and modify it, but the actual circuit data is never exposed [1]. As an attacker, the vendor/designer could insert a backdoor or hardware Trojan to compromise the security of further stages in the process.

The System on Chip (SoC) design house can be vulnerable to maliciously modified Electronic Design Automation (EDA) tools that could insert hardware Trojans or gather data on their implementation of the design, as well as the malicious behaviors from the IP providers, since complex IC designs can be comprised of multiple instances of IPs. These attacks cannot be easily prevented, but can be detected through trust

verification methods [2]. Once detected, the toolchain can be modified to remove the malicious tools. The SoC designers would also be able to attack the end user by inserting their own Trojans or attack the vendor by counterfeiting the design.

Currently, foundries are not particularly susceptible to attacks by other entities in the IP lifecycle. However, once a design is ready to be manufactured, the foundry has the ability to exploit it in many ways, as it is inherently given access to low level chip representation information in order to fabricate it, often in the form of GDSII files. They are then able to insert their own hardware Trojans or overproduce a chip with the intention of selling the extras.

Finally, the end users are concerned with protecting their own personal data. They are vulnerable to a number of malicious behaviors from other stages, like Trojans and backdoors inserted into the design. End users are able to attack a design by reverse engineering the IP to counterfeit the device or performing side-channel attacks to acquire secret information stored in the IP. These attacks primarily affect the IP vendor/designer [2, 3]. Each of these possible attacks has its own protections, several of which will be explored further in this document.

When manufacturing new circuits, a significant concern is that most designers creating these systems do not have direct access to a fabrication method. As a result, most chip fabrication is done through external services that cannot necessarily be trusted. These outside manufacturers could be performing many of the hardware attacks noted earlier by reverse engineering the design. Even without reverse engineering, untrusted manufacturers could still overproduce a device and sell the extra for their own profit, either to end users or to the initial designer's competitors [4].

A major-area of concern with IP piracy is that the configuration information, the bitstream in the case of Field Programmable Gate Arrays (FPGAs), could be reverse

engineered. While the issue of manufacturing piracy mostly concerns the design of Application-Specific Integrated Circuits (ASICs) or other custom designs, FPGA programming is widespread and accessible, so the effect of these attacks could have a substantial reach. Protections against these attacks must be evaluated for effectiveness, and more advanced countermeasures must be found to guard against increasingly complex attacks.

One common technique used to protect hardware designs is to obfuscate the design itself, such that reverse engineering becomes too expensive, in either time or resources, for an adversary to realistically accomplish. This obfuscation can be done at different stages of the lifecycle, ranging from the HDL being modified to make it more difficult to understand, to the netlist, which is used to either generate a bitstream — in the case of FPGAs — or generate the hardware design for manufacture — in the case of ASICs, being manipulated. Other methods of protection rely on encrypting certain aspects of the design, or even the design description itself, such that only trusted users with the correct key can utilize the sensitive information [1].

1.2 Thesis Objective

The purpose of this thesis is to evaluate some of the currently proposed hardware obfuscation techniques, both on existing benchmark circuits as well as on specific non-linear cryptographic components. The cryptographic components analyzed include substitution boxes, namely the 8-bit S-box from Advanced Encryption Standard (AES) and a larger 16-bit S-box from MK-3 [5, 6]. Further, it will explore satisfiability (SAT) attacks as a method of defeating hardware obfuscation, and the effects which different obfuscation methods have on protecting the components from this type of attack.

1.3 Novel Contributions

The novel contributions of this work primarily focus on implementing different methods of obfuscation on the S-box component of the configurable block cipher MK-3. The effectiveness of these obfuscation methods on this component were evaluated with the SAT attack. Finally, the use of the lightweight block cipher SIMON was tested for additional obfuscation as an One-way Random Function (ORF).

Chapter 2

Background

2.1 Hardware Reverse Engineering

Reverse engineering is a necessary precursor to most attacks on hardware IP designs; the ultimate goal of this approach is usually to determine most or all of the underlying functionality of the targeted design. This can be accomplished through a number of methods, including inferring the function of the system and extracting the circuit implementation details [7]. Reverse engineering also differs based on the type of system being targeted; determining the functionality of a system on a FPGA can be very different from reverse engineering an ASIC.

2.1.1 FPGA Reverse Engineering

Reverse engineering a FPGA relies on the adversary being able to extract and decipher the bitstream used to program the device [8]. It is the job of the FPGA vendors to make this as difficult as possible, through offering measures like bitstream encryption as well as by ensuring that their bitstream specifications are not publicly available.

There have been several projects that have attempted to thwart these measures, though, and some have been able to at least partially reverse engineer a bitstream back into a usable netlist.

In [9] the use of a tool called "debit" is discussed as a means of reverse engineering the bitstream format for several of Xilinx's older FPGA devices — namely the Virtex 4 and Virtex 5, among a few others. This work was partially successful, as some of the configuration information for the netlist could be recovered. Over 90% of the bitstream in these cases was made up of switching box logic — specifically Program Interconnect Point (PIP) data — and the tool was only able to recover some of that data.

Another tool was later called Bitstream Interpretation Library (BIL) [10]. This tool expanded upon the previous work and created a database that contains many of the component definitions and mappings from Xilinx's older bitstreams. This method uses files which can be obtained through the Xilinx toolchain for Xilinx ISE, called the XDL and XDLRC files. These are netlist files that describe the design and structure of the device being used. Unlike the work done using the "debit" tool, this work was able to completely decipher a large segment of the PIP information using the information from the XDLRC file. It was not, however, able to reverse the PIP commands for primitive site and several other types of tiles [8].

Finally, an effort was put forth by Mathias Lasser to reverse engineer both Lattice's iCE40 bitstream and Xilinx's 7-series FPGA bitstream, which is for their most recent family of devices [11]. Lasser was able to completely reverse engineer Lattice's iCE40 FPGA bitstream, which is a very small device compared to Xilinx's FPGAs. This was accomplished by debugging and slightly modifying the vendor-supplied bitstream generation tool. This is the best-case scenario for FPGA reverse engineering, but it is very unlikely to work on most applications where the bitstream mapping information is more carefully hidden. Reversing the Xilinx 7-series bitstream was a more substantial challenge, as this bitstream was notably more complex and included features like error correction that made the bitstreams difficult to analyze. It was ultimately done

through careful analysis of the bitstream by converting the data to a bitmap image and identifying patterns. The full mappings have not been released; however, it can be assumed that an adversary — given enough time and resources — could manage to reverse engineer a bitstream for an FPGA using a combination of these methods.

2.1.2 ASIC Reverse Engineering

Reverse engineering of ASICs is more focused on analyzing the physical chip after production. This can be done at a number of different levels depending on the adversary and their capabilities [7].

At the IC level, an attacker has several options available to reverse engineer a fabricated device, both nondestructive and destructive. For a nondestructive attack, the adversary can use X-ray tomography and ptychography to analyze the internal structure of a chip layer-by-layer without having to alter the device itself [2, 12]. If the adversary is willing to alter or destroy the IC in the process of reverse engineering it, the attack could entail delayering the chip and using microscopy (optical, Scanning Electron Microscope (SEM), Transmission Electron Microscope (TEM), or any combination thereof) to analyze each layer individually [7].

Once the chip has been fabricated and mounted on a Printed Circuit Board (PCB), the reverse engineering can extend to that PCB itself. This would allow an attacker to gain a greater understanding of the chip's outward interactions. The attacks against PCBs are largely the same as those against ICs, in both the nondestructive and destructive attack approaches [13].

A full system would consist of a number of ICs, PCBs, and other configuration information to allow the device to work as intended. This is what most end-users would receive. This configuration information is often stored in some form of non-volatile memory. A reverse engineering attack on a completed system would likely

include attacks on the individual components, as mentioned above, as well as reverse engineering the configuration data, which can contain information about the system's final operation and timing.

An understanding of the functionality of the IP gained through reverse engineering can lead to a number of attacks, such as counterfeiting and Trojan insertion, that go beyond the initial issues of stealing the design or other trade secrets [2].

2.2 Intellectual Property Attacks

2.2.1 Hardware Trojans

Hardware Trojans are additional components added to a design that act in a malicious way. They are intended to be difficult to detect and can cause problems with the affected circuit ranging from minor malfunctions to early system death to leaking critical information. Trojans are characterized by three main features: malicious intent, difficulty to detect, and rare activation [14].

A Trojan must have malicious intent to be considered a Trojan. It must be inserted by an adversary with the intention of compromising the design in some way, otherwise it cannot be considered a Trojan.

A Trojan must also be placed in such a way that it is difficult to detect. It would not benefit an attacker if a Trojan could be easily identified and removed, so the intention is that they be well disguised from normal testing and analysis of the modified device. This relates to the third characteristic: by crafting a Trojan so that it only triggers on rare occurrences, the likelihood of it being detected in normal testing is greatly reduced. In theory, these triggering conditions could be entirely skipped in ordinary fault testing, but are likely to arise under extended operation in the field [15].

There are several kinds of Trojans that could be inserted into a design with different intended outcomes. One form is a basic device that would, when triggered, cause a malfunction at the output. An example of this from [15] is shown in Figure 2.1.

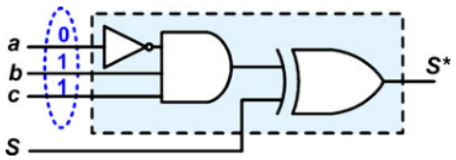


Figure 2.1: Combinational Trojan Example [15]

This Trojan would be inserted into the path of signal `S` and change the output to be `S*`. When the trigger condition (`a`=0, `b`=1, `c`=1) is met, `S*` becomes the inversion of `S` and likely introduces a malfunction in the design. If this Trojan is placed strategically, this error could propagate through much of the circuit and leave the output completely unusable.

Another form of Trojan is one that leaks critical information from the design when triggered. A Trojan of this kind is proposed in "MOLES: Malicious Off-Chip Leakage Enabled by Side-Channels" [16], which can leak critical information through side channels. The given example depicts a MOLES-based Trojan inserted into a cryptographic processor to leak the key information, shown in Figure 2.2.

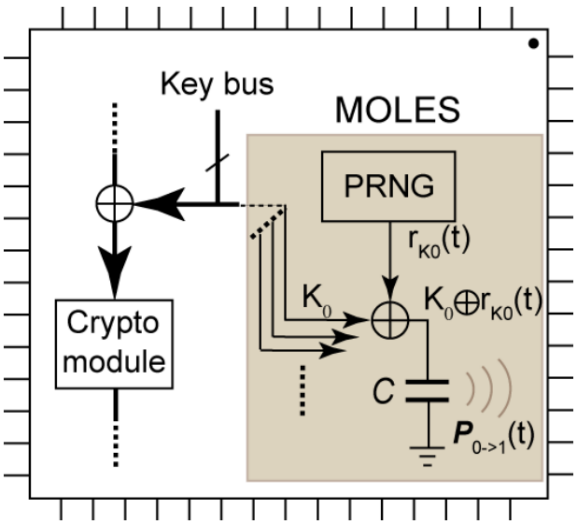


Figure 2.2: MOLES Example Trojan [16]

2.2.2 Counterfeiting

Counterfeiting a design is the process of creating illegal copies of IP. The issue of counterfeiting has become greater with the increased use of offshore foundries and the lack of effective avoidance mechanisms [17]. Often, this approach requires reverse engineering of the design, however it links closely to the problem of overproduction — which does not necessarily require full reverse engineering to create counterfeits of a design.

2.2.3 Overproduction

Overproduction is mainly a concern with ASIC production at untrusted foundries. It is possible that a bad actor at a foundry/fab could produce more of a design than necessary and sell the extra for profit. This sale could be to new end users or to the initial designer's competitors, who could then reverse engineer the design to uncover secrets [2].

The counter to this attack is called metering, and it refers to carefully controlling the number of devices that are produced, for whom they are made, and their properties.

Obfuscation can be enacted to help prevent this attack by ensuring that a produced device is unusable without some final configurations that would be done by a trusted actor or the design house itself [4, 18].

2.3 Hardware Model Encryption

One approach to protecting hardware designs is to encrypt the HDL using cryptographic methods. This prevents the end users of these models from accessing the plaintext of the design and being able to maliciously influence it. The main method for this HDL encryption is defined in IEEE Standard P1735 [1]. This document provides standards of practice for encrypting the IP using both symmetric and asymmetric cryptography, as well as recommendations for how to manage licensing and distribution of the encrypted IP.

These standards allow vendors to send encrypted versions of their designs to customers who have properly purchased them, and for those customers to have their EDA tools decrypt the IP internally. This ensures that the end user never has access to the plaintext, but is still able to use the secured component [19].

By encrypting the design, vendors are preventing some of the more basic attacks from end users — namely counterfeiting the design or stealing secrets from within it. This encryption is useful to that extent, but it does very little to protect against reverse engineering or any attacks at the manufacturing stage, should the IP be intended for use in an ASIC.

However, vulnerabilities were discovered in IEEE standard P1735 roughly two years after it was adopted. It was prone to several attacks that could, in the worst cases, allow for full circuit definition recovery of the IP [20].

2.4 Hardware Authentication

Though it doesn't necessarily directly protect the IP, hardware authentication is often considered one of the main methods of stopping IP piracy. Authentication involves techniques like digital watermarking, which can prove who created the initial design by embedding digital signatures in the design that should only be known to the IP vendor [21]. This, however, is only useful in litigation should the design be stolen.

2.5 Hardware Obfuscation

Hardware obfuscation consists of a series of methods by which the function of a design is modified to make it infeasible to understand or use without the proper information, such as a key. These are intended to protect the IP at almost all stages of the hardware development process. Obfuscation of this kind is able to offer protections against reverse engineering of the hardware itself by either the manufacturer or end user, insertion of hardware Trojans by the manufacturer, and overproduction of the device. Since functionality is affected, without the secret necessary for proper operation the device will not behave normally and most attacks are, ideally, prevented.

These techniques can vary significantly in their implementation, from introducing extra elements into the design, to swapping signal paths based on differences from chip-to-chip.

2.6 Hardware Obfuscation Methods

There are a number of different methods of obfuscating the function of a design. Some rely on inserting additional combinational logic along paths in the design to act as locks, while others involve adding Finite State Machines (FSMs) for encryption and authentication.

2.6.1 Netlist Logic Locking

The following methods obfuscate hardware designs by adding combinational logic into the original design to create false outputs if not configured properly. This configuration usually comes in the form of key inputs to the circuit — additional inputs that function as a secret key, similar to a cryptographic function.

2.6.1.1 Random Locking

The first method of netlist level logic locking, proposed in "EPIC: Ending Piracy of Integrated Circuits," relies on randomly placing XOR and XNOR gates throughout a circuit to act as the obfuscation [22]. Each locking gate would be connected to an internal signal and a key bit, as shown in Figure 2.3.

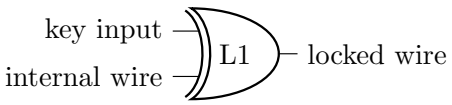


Figure 2.3: XOR Lock Gate

These locking gates would be picked based on the key bit input — XOR for '0', XNOR for '1' — and would then be placed randomly throughout the circuit. The key bit/gate combinations are determined by the logical characteristics of the gates: when one input of an XOR is a '0' it acts as a buffer for the other input, and the opposite is true for XNOR.

It is unlikely that a circuit obfuscated in this way would be unlocked by multiple keys, though is technically possible in certain situations. In [22] this is said to be possible in a circuit consisting entirely of XOR and XNOR gates, as an XOR tree can be unlocked by 50% of all key combinations. However, it is possible in a circuit consisting of any number of XOR and XNOR gates; should two locking gates be made the inputs of another XOR or XNOR gate those key bits could be invalidated in

certain combinations. Another instance where multiple keys could unlock the circuit comes when employing any form of stacking locking gates or surrounding an inverter, as shown in Figure 2.4.

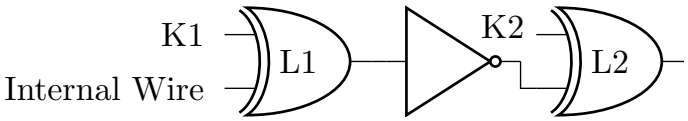


Figure 2.4: Lock Gates Surrounding an Inverter

With the configuration in Figure 2.4, if both lock gates are given the wrong input the inversions will cancel out and the proper value will be passed. This has the capability to invalidate some key bits in certain combinations and allow for multiple functional keys. The likelihood of this occurring should be low, especially in a large enough circuit, and the risk of this becoming a significant problem is mitigated by having a suitably large key. Flaws like this can also be avoided entirely by modifying the placement algorithm to account for similar configurations of gates; in this case, the gates would no longer be placed truly randomly, but by removing the possibility of this occurring the locking would on average be stronger with fewer invalidated key bits.

It was determined in [22] that a "suitably large key" is larger than 64 bits, as that can withstand a brute force attack with today's technology. That does not, however, guarantee that the circuit is secure, as there are several other attacks currently developed, and possible new attacks, that can break this encryption much more quickly than a brute force attack would be able to on average.

2.6.1.2 Fault Analysis-Based Logic Locking

A logic locking technique is created in [23] in which the location of XOR/XNOR locking gates are determined by using similar methods to those used in fault-analysis

of completed circuits. As such, incorrect inputs to locking gates can be pictured as stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1) faults depending on the input pattern of the initially locked circuit.

The metrics that take advantage of this definition were designed to combat possible inadequacies with the random locking scheme from [22] — specifically those dealing with fault propagation and fault masking problems, as shown by the previously discussed lock gate interactions with other XOR and XNOR gates.

Fault propagation in this context is the idea that not all incorrect keys will propagate their "error" through the circuit. Given circuit input patterns, some key gates might be on unused paths, which would render them useless. An example of this is given in Figure 2.5.

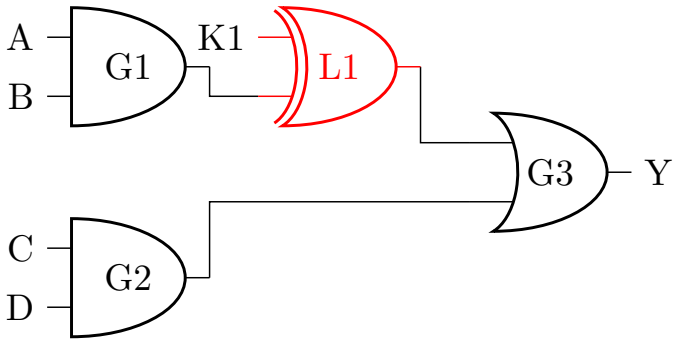


Figure 2.5: Fault Propagation Example

In the circuit in Figure 2.5, for any set of inputs to the circuit with both `C` and `D` as `1`, the output of `G3` will output a `1` and the effect of the locking gate `L1` will not be propagated to the output.

Fault masking refers to a property which states that when exciting multiple stuck-at faults some of the faults' effects could be covered up by other faults later in the circuit. The same thing can happen with key gates in a locked netlist.

Using fault analysis tools, the ideal locations for locking gates can be determined such that the hamming distance between the correct and incorrect outputs is 50%, meaning that for an incorrect key half of the output bits will be incorrect. This is achieved by looping through the netlist and determining the fault impact of each gate and then locking the gate with the highest fault impact until the correct number of key bits have been achieved. The fault impact is determined using Equation 2.1.

$$\begin{aligned} Fault\ impact = (No.\ of\ Test\ Patterns_{s-a-0} \times No.\ of\ Outputs_{s-a-0}) \\ + (No.\ of\ Test\ Patterns_{s-a-1} \times No.\ of\ Outputs_{s-a-1}) \end{aligned} \tag{2.1}$$

2.6.1.3 Strong Logic Locking

In [24] a logic locking technique, referred to as Strong Logic Locking (SLL), was proposed that utilizes key gates in a way similar to [22], but that offers other heuristics for where those gates should be placed to be as effective as possible. These methods of gate placement are determined by a gate's "mutability," or the ability to determine its key bit by "muting" another locking gate in the circuit — similar to fault masking from [23]. There are several definitions for each locking gate based on its placement within the circuit: isolated, dominating, concurrently mutable, sequentially mutable, and non-mutable. Each of these gate definitions has an associated attack that could allow the attacker to determine its key bit.

Isolated Key Gates An isolated key gate is one that does not have a path to all other key gates. For example, if two key gates input directly into the final gates for two separate outputs. These gates cannot have any effect on each other or any other key gates, so they are both isolated.

Dominating Key Gates A dominating key gate is one that is on every path between another key gate and an output. This key gate's output would "dominate" the output of the earlier gate.

Concurrently Mutable Convergent Gates Mutable convergent gates are gates that converge at another gate whose outputs can be "muted" by placing a certain bit on an input to another gate such that the effect of the lock cannot be seen. For two gates to be concurrently mutable, one gate's bit must be able to be determined by muting the other, and vice-versa. This is shown in Figure 2.6

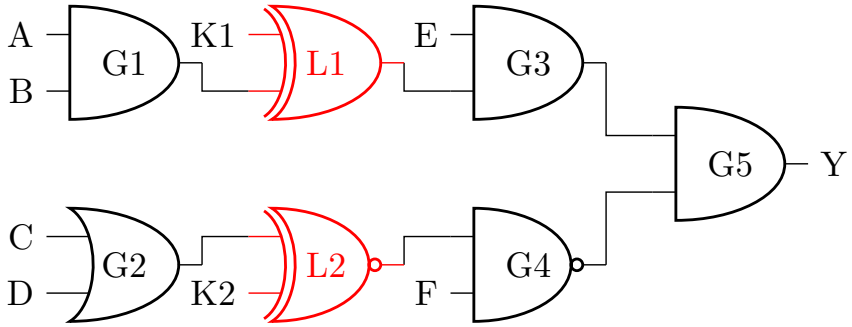


Figure 2.6: Concurrent Key Gate Muting

The effect of `K1` can be muted by setting `E=0`, and the effect of `K2` can be muted by setting `F=1`. Once a gate is muted, the key value of the other can be sensitized.

Sequentially Mutable Convergent Gates Sequentially mutable gates are mutable gates which only go in one direction. If the key bit for `K2` can be found by muting `K1`, but not the other way around, the gates are sequentially mutable.

Non-Mutable Convergent Gates Non-mutable convergent gates are key gates that converge to another gate, but no key gate values can be found by muting another key gate's effect.

The method for inserting key gates for increased circuit protection is then to add gates

such that the number of non-mutable gates is maximized. This is done by representing the key gates of a circuit in a graph, referred to as an "interference graph," in which each locking gate is a node, dashed edges connect mutable gates, and solid edges connect non-mutable gates. Sequentially mutable gates are connected by both a solid and a dashed edge, to show that only one of the gates can be muted to find the value of the other.

A circuit with multiple types of mutable and non-mutable gates is shown in Figure 2.7.

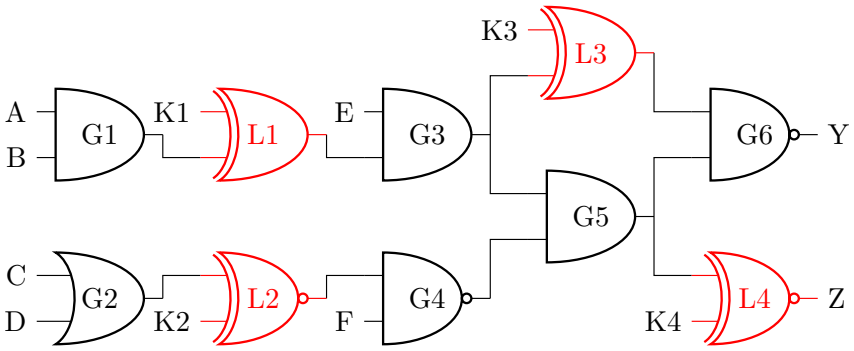


Figure 2.7: Circuit with Locking Gates for Demonstrating SLL Graph

The interference graph representation of this circuit is shown in Figure 2.8.

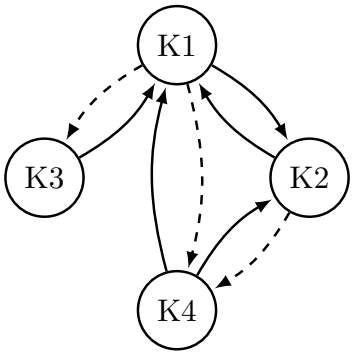


Figure 2.8: SLL Interference Graph

Gates `L1` and `L2` are connected to each other with non-mutable edges, as they are non-mutable convergent gates. Both gates can be muted by setting `E=0` for `L1` and `F=1` for `L2`; however, neither can be used to determine the value of the other.

The other gate connections are sequentially mutable, with the exceptions of `L2` connecting to `L3` and `L3` connecting to `L4`. As an example: if `E=0` then the effect of key bit `L1` can be muted, which can be used to help determine the values of `L3` and `L4`. This cannot be done in the opposite direction, which makes that connection sequentially mutable. Similarly, `L2` being muted cannot be used to determine `L3`, and `L3` cannot be used to find `L4`, so in the interference graph they are disconnected.

By using this method of representing the key gates in a circuit, stronger logical obfuscation is achieved by maximizing the number of non-mutable edges in the circuit. This is done by initially inserting a certain percent of the key gates randomly — Rajendran et. al. used 10% — and then constructing the interference graph of the key gates. Each further gate was then introduced iteratively, such that for every gate in the netlist the type of edges connecting to the previous gates were determined. The type of edge was then assigned a weight, with non-mutable edges being given the higher weight, and the sum of the weights was calculated. The gate which maximized this sum was selected for obfuscation and a key gate was inserted at its output.

2.6.1.4 SARLock

SARLock [25], or SAT Attack Resistant Logic Locking, is another extension of the locking method proposed in EPIC that specifically focuses on increasing resistance to the SAT attack introduced in [26]. The SAT attack, which is described in detail in Section 2.7.1, relies on finding Distinguishing Input Patterns (DIPs) that can rule out incorrect key values, ideally multiple at a time. The worst-case scenario for the SAT attack then, is that each DIP is only able to rule out a single incorrect key. This is the goal of SARLock.

This type of obfuscation is done by introducing a comparator into the design, which compares the key to the circuit input. For certain combinations of the key and input,

the comparator produces a *flip* signal, which is XORed with a primary output, thus inverting it. This is shown in Figure 2.9.

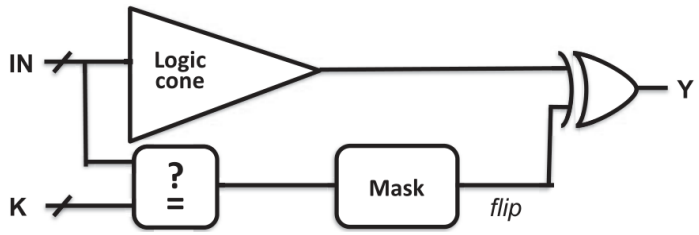


Figure 2.9: SARLock Circuit [25]

The intended effect of this is to ensure that for each input pattern only a single key will produce an incorrect value. While only the correct key will produce the correct output for all inputs, other keys will produce an incorrect value for one input pattern. This effect is shown in Table 2.1.

Table 2.1: Example Truth Table for a SARLock Circuit

				Output for Each Key Value							
a	b	c	Y	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0
0	1	1	1	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0	1	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	0

The truth table defines the behavior of a circuit: $Y = (ab) + (bc) + (ac)$, with key gates inserted. Only a single output bit will change for each possible key value, ensuring that each DIP can only rule out a single incorrect key. This is done by creating a function, $flip = F(in, k)$, where in is the primary input to the circuit and k is the

key. This *flip* bit is passed into an XOR gate on the output of the circuit, as shown in Figure 2.9.

SARLock on its own is not enough to provide protection to a circuit, as it only serves to increase protection against SAT attacks. As the additional components for SARLock are isolated from the main circuit, there are several attacks that could remove that functionality if no additional protections were added. Yasin et. al. recommend a two-layer locking approach, in which SARLock is combined with SLL from [24]. This method splits the key into two separate parts, one for locking the logic cone using SLL, called `K1`, and one for interacting with the SARLock block, called `K2`. `K2` is scrambled with `K1` for two reasons. First, it ensures that the flips do not all occur on pre-determined combinations of input and key values. Second, it creates a dependency between the two keys, which ensures that a simple removal attack cannot reduce the number of the key bits. In order to extract `K2` from the scrambler function, the value of `K1` must be known; however, in order to determine the value of `K1` an attack — possibly a SAT attack — must be performed on the locked logic cone, which the SARLock block should prevent. This combined functionality is shown in Figure 2.10.

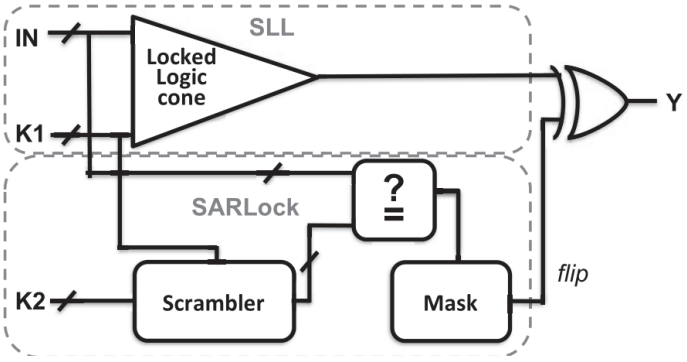
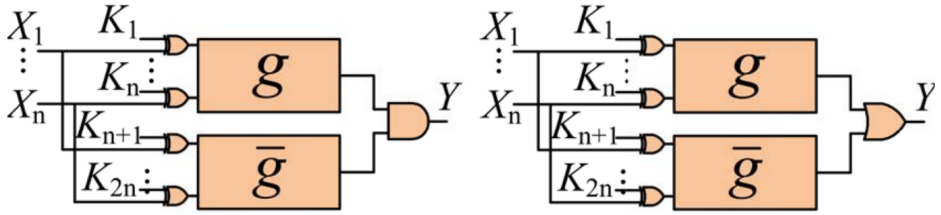


Figure 2.10: SARLock+SLL [25]

2.6.1.5 Anti-SAT

Anti-SAT is another netlist locking method of hardware obfuscation designed to attempt to thwart the SAT attack [27]. The aim of Anti-SAT is to include a small additional circuit that greatly increases the number of iterations the SAT attack will take to break the function. The method of obfuscation has two configurations, known as *type-0* and *type-1*. Both are made of two additional functional blocks, g and $\overline{g}$, which share the primary inputs but have differing keys. In *type-0* Anti-SAT the output of these blocks is then tied to an AND gate, while in *type-1* they are tied to an OR gate, as shown in Figure 2.11.



(a) *type-0* Anti-SAT Block **(b)** *type-1* Anti-SAT Block

Figure 2.11: Anti-SAT Configurations [27]

By feeding the output of the g and $\overline{g}$ blocks into an AND or an OR gate, this circuit now produces a single bit output, defined by the following functions:

$$Y = g(\vec{X} \oplus \vec{K_1}) \wedge \overline{g(\vec{X} \oplus \vec{K_2})} \tag{2.2}$$

$$Y = g(\vec{X} \oplus \vec{K_1}) \vee \overline{g(\vec{X} \oplus \vec{K_2})} \tag{2.3}$$

Equation 2.2 gives the equation for the output of a *type-0* Anti-SAT block, and Equation 2.3 gives the equation for a *type-1* block. The inputs and keys in this case were only XORed for simplicity, but in a realistic implementation it would be

a combination of XORs and XNORs in accordance with the key bits, similar to the implementation in EPIC [22].

These equations, if the correct key is given, should then always output either a 0 in the case of the *type-0* block, or a 1 in the case of a *type-1* block. These blocks are then integrated into a circuit as shown in Figure 2.12.

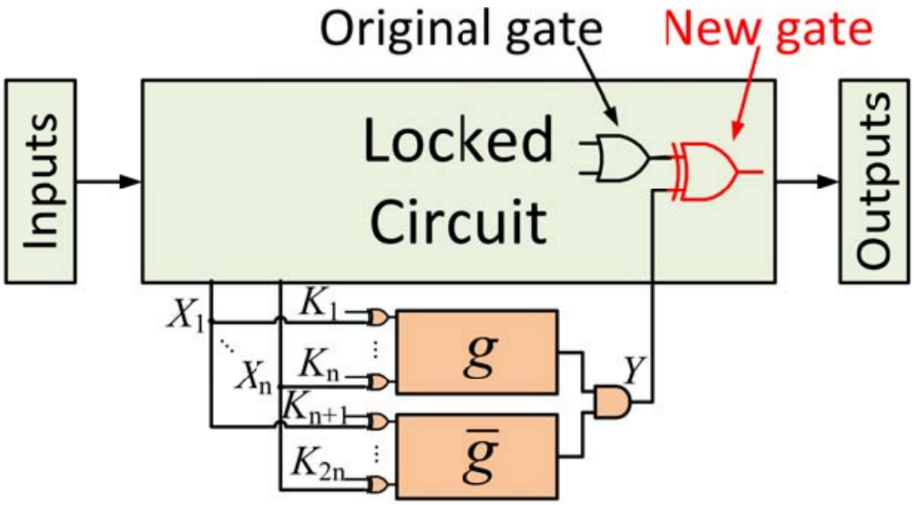


Figure 2.12: Anti-SAT Type-0 Integrated into a Locked Circuit [27]

Based on whether a *type-0* or *type-1* block is used, the "New Gate" highlighted in the figure could either be an XOR or an XNOR to guarantee that the output is correct with the correct key.

How these Anti-SAT blocks connect to the locked circuit is important for increasing the level of complexity of the SAT attack. It is recommended that the inputs to the block are connected to primary inputs from the original circuit, and the output is connected to a randomly selected internal wire within the top 30% observability.

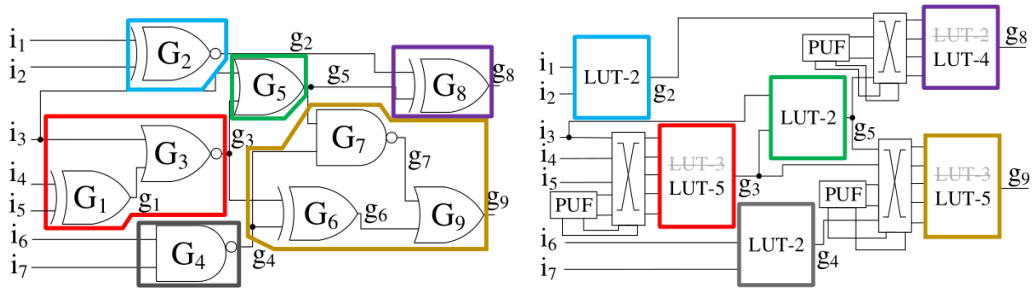
In [27], it is recommended that the Anti-SAT blocks also be combined with another obfuscation method to provide resilience against other types of attacks. SLL is again recommended for this purpose, as it has been shown to be strong against key

sensitization attacks [24].

2.6.1.6 LUT-Lock

In [28], a method of netlist-level logic locking is proposed which uses Look-Up Tables (LUTs) to obfuscate the netlist rather than simpler gates, as were used in other methods [22–25, 27]. This proposed method also includes specific heuristics for which gates should be obfuscated by these LUTs so that the design is secure against SAT attacks.

The research in [28] points to a number of different ways to utilize LUTs for stronger logical obfuscation, some of which differ between FPGAs and ASICs. In FPGAs the hardware itself is fixed, but can be reconfigured. This gives the designer an abundance of LUTs and other hardware resources that can be used for obfuscation. When routing for a FPGA bitstream, individually defined gates are mapped to LUTs of various sizes. This can be leveraged to insert obfuscation logic into the design. Gates can be placed in larger-than-necessary LUTs, with the additional inputs being fed by Non-Linear Feedback Shift Registers (NLFSRs) or Physical Unclonable Functions (PUFs). These can act as internal keys, so that the bitstream will function properly only on a specific device that is configured to produce that particular output. This method of utilizing LUTs in FPGAs is shown in Figure 2.13.



(a) Unmodified Circuit **(b)** Obfuscated Circuit with LUTs

Figure 2.13: FPGA Obfuscation LUT Example [28]

The red, purple, and yellow groups of gates all became larger-than-necessary LUTs with PUFs leading to some inputs. These are the obfuscated gates in the design.

The use of LUTs for obfuscation in ASICs differs substantially from their use in FPGAs, as they are not readily available resources for a fully custom design. LUTs can lead to substantial area and delay overheads with modern technology, so they need to be used more sparingly and should be placed more specifically so as not to interfere with timing or power specifications. Their use is rather simple, though: the gates selected for obfuscation are removed and replaced with equivalent LUTs. The functionality of these LUTs can be hidden from the fab, and are then programmed at a trusted facility later.

The heuristics for where to best place these LUTs are what provide more substantial protection against SAT attacks. There are five algorithms established in [28] on which LUT-Lock is based, each used to remove weaker gates for obfuscation from a list of possible candidates. These algorithms are:

1. FIC: Fan-In Cone of minimum number of primary outputs
2. HSC: Focusing on Higher Skew gates in FIC
3. MFO-HSC: Focusing on gates with Minimum Fan-Out
4. MO-HSC: Focusing on gates with least impact on primary outputs
5. NB2-MO-HSC: Avoiding back-to-back insertion of LUTs

FIC The first algorithm is focused on reducing the output corruption of the locking gates, which increases the difficulty of the SAT attack. This is done by selecting only fan-in cones that affect the fewest number of primary outputs of the circuit. Due to intersections in these fan-in-cones, it is rare that only only a single output

will be affected; however, it should be minimized as much as possible. This can be accomplished by doing a Breadth First Search (BFS) to find the closest cells to the selected output for obfuscation. After all gates in the selected fan-in cone are replaced, a new output is selected and the process is repeated. The selected output must have a large fan-in cone so as to offer more candidate gates for obfuscation.

HSC This algorithm relies on creating a candidate gate list based on the Signal Probability Skew (SPS) of all gates in the selected output's fan-in cone. Through the original researchers' testing, it was found that gates with higher SPS are better for obfuscation. SPS is defined as $\mid P_r(0) - P_r(1) \mid$, where $P_r(0)$ is the probability that the gate will produce a 0 and $P_r(1)$ is the probability that the gate will produce a 1 based on possible inputs. This is a measurement of how controllable a gate is by the primary inputs of the circuit. By selecting gates with a high skew, it raises the chances that a SAT attack will select inputs that will test the output of that particular gate.

MFO-HSC This is another step that focuses on lowering the output corruption of the locked gates. It finds the gate to obfuscate by determining the gates with minimum fan-out. If multiple gates have the same fan-out, the gate with the higher SPS is selected. This is done to attempt to reduce the corruption to only gates within the initial fan-in cone, with the intention of reducing the corruption at intersections. When a gate is selected for obfuscation, the gates in its fan-in cone are added to the list of candidate gates for the next iteration of the method.

MO-HSC This is an incremental improvement over MFO-HSC, in which the number of affected primary outputs is accounted for. Some gates can have larger fan-out but only affect a single primary output. MO-HSC, instead of finding the fan-out cone of a gate, calculates how many outputs are affected by this gate. This approach requires additional processing to find each candidate gate, but should further reduce the output

corruption. Similarly to MFO-HSC, once a gate is selected to be obfuscated its fan-in cone is added to the list of candidate gates for the next iteration.

NB2-MO-HSC This final algorithm utilizes MO-HSC, but additionally accounts for the back-to-back insertion of LUTs. When multiple LUTs are placed sequentially multiple keys could be able to unlock those obfuscated gates, due to De Morgan's Laws. Each additional locked gate in a fan-in cone increases the number of possible keys for unlocking, which leads to an exponential increase in the number of successful keys. This is avoided by ensuring that no new obfuscated gate feeds into a previously added one.

The NB2-MO-HSC algorithm is the most complete algorithm for finding the gates to be obfuscated, so this is ultimately the algorithm which was used in the final LUT-Lock design.

2.6.2 ORF Insetion

To provide additional obstruction between the primary inputs of a circuit and a locked netlist, Yasin et. al. proposed inserting an ORF between the key inputs of the circuit and the key inputs of the logic locked device [29]. This is shown in Figure 2.14.

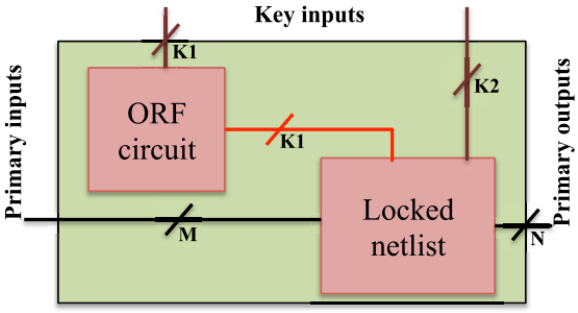


Figure 2.14: ORF Insertion [29]

An ORF is a random function whose inverse is infeasible to compute, meaning its input pattern cannot reasonably be determined by its output pattern. These functions

prevent attackers from determining the key inputs to the function from its outputs by disassociating them.

The researchers discuss using a modified AES block cipher with a hard-coded key as the ORF for these locking schemes, as they are reliable one-way functions [29]. The hard-coded key value is implemented to prevent possible removal attacks. Since the structure of AES is well known, it is possible that an attacker could identify and remove it; however, by including the key in the design itself, once it is synthesized it would become much harder to identify and remove.

2.6.3 Binary Decision Diagram Logic Locking

Xu et. al. proposed a method of logic locking that does not function on the base netlist like those previously discussed, but instead relies on representing the circuit using a Binary Decision Diagram (BDD) and inserting obfuscation in that before synthesizing into a netlist [30]. A BDD is a method of representing logic, often used in verification and synthesis, that shows the logic as a tree of decisions. Figure 2.15 represents the BDD for an XOR gate.

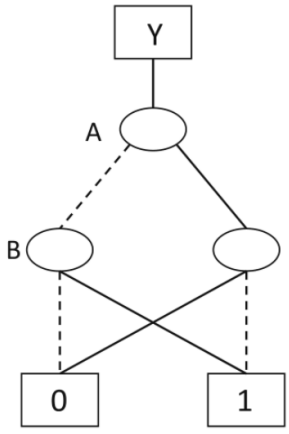


Figure 2.15: BDD Representation of $Y = A \oplus B$ [30]

In the diagram, dashed edges represent the variable being input as a 0, and solid

edges represent a 1. The final boxes show the output of the function when that node has been reached. These diagrams can easily be expanded to represent more complex logical functions.

Using these BDDs, the authors of [30] use Figure 2.16 to describe the obfuscation method.

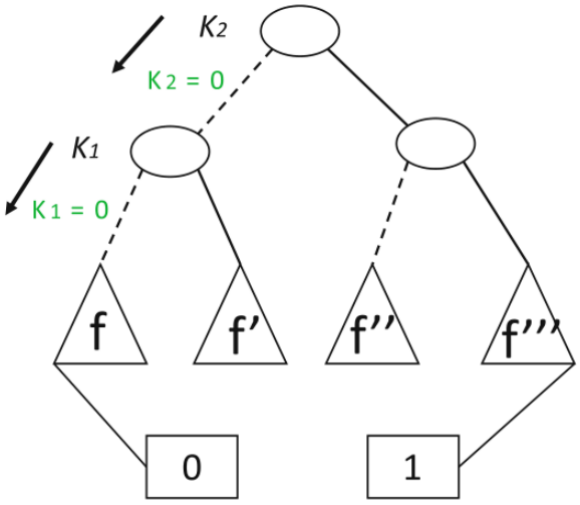


Figure 2.16: Obfuscated BDD [30]

`K1` and `K2` are new variables added to the function which act as key bits. The use of the proper key, (0,0), leads to f, the original function being obfuscated. Any other key combination leads to a modified version of the circuit f', f'', or f'''. These modified circuits provide incorrect functionality, and allow the design to be obfuscated.

This obfuscation method was designed as a countermeasure to the bypass attack that was introduced in the same paper.

In order to provide resistance from the SAT attack, each incorrect function must have a Hamming Distance from the original function of 1. This creates only a single incorrect bit on the output when the wrong key is used, which increases the difficulty of SAT attacks finding a DIP.

2.6.4 Finite State Machine Insertion

Another prevalent method of hardware obfuscation is the act of inserting additional control FSMs, which can work with a key to obscure the functionality of the design.

2.6.4.1 Adding an Isolation State Space

In [15], a method of obfuscation was proposed that uses a separate and isolated "obfuscated state space" that locks the proper functionality of the design behind a key sequence. This was designed specifically to protect against hardware Trojan insertion, as a Trojan inserted into the incorrect state space would be rendered benign. This concept is shown in Figure 2.17.

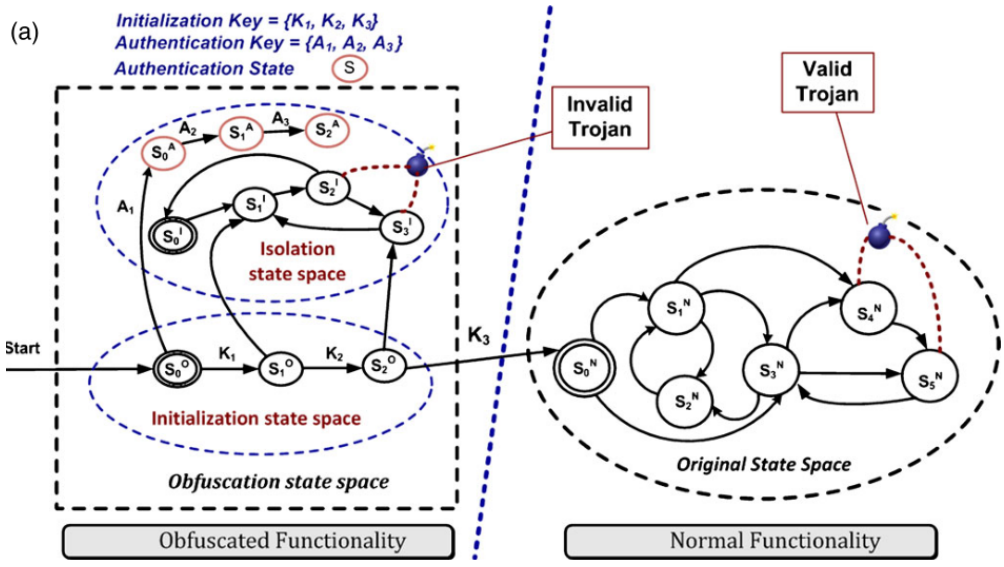


Figure 2.17: Obfuscated State Space [15]

A notable benefit of this method of obfuscation is that it offers both obfuscation and authentication, depending on how the designer chooses to set up the state spaces. These state spaces come from the following modifications of the State Transition Graph (STG):

1. The size of the reachable state space is increased with additional state elements.

2. Certain states which were initially unreachable are used in the isolated state space.

When the device is booted up, it starts in the "initialization state space," and a key must be entered before moving on to the proper circuit functionality. If the entered key is correct, the device transitions to the "original state space" and is able to function as intended; however, if it is incorrect the functionality is moved to an "isolation state space" that offers no way to return to the original state space without rebooting the device and attempting another key.

The initialization sequence can be made very short, and as such can also be hidden from the end-user as part of the normal "power-on" latency. This could be particularly helpful if the key was derived from a PUF so that the design could only work for a single device, and the user would not need to influence the key.

To greatly increase the size of the obfuscated state space, the authors recommend inserting a Parallel State Machine (PSM) that defines transitions between the extra states. This can be folded into the normal functionality so it cannot be extracted and removed easily. These states that exist in the obfuscated functionality are also what allow for authentication with this design. If a specific "incorrect" key is given in the initialization sequence, it can transition to a specific series of states that function as a digital watermark.

These additional state spaces should make hardware Trojan insertion incredibly difficult, as effective Trojans must be inserted at locations which will only be reached on rare events, so the proper functionality must be understood and accessible. If the attacker is unable to access the normal state space then they cannot insert an effective Trojan. A randomly placed Trojan is very unlikely to be located at a rare state in the normal functionality — it is more likely that it would be placed at a common state

and be found or placed in the isolated state space and would be rendered useless.

2.6.4.2 Utilizing the CDFG

In [31], a similar obfuscation method was suggested; however, it was accomplished by converting the Register Transfer Logic (RTL) into a Control and Data Flow Graph (CDFG) to determine where to add the obfuscation state space and mode-control state machine. First, the RTL is converted to the CDFG and the graph is flattened. This means that smaller blocks are merged where possible, creating larger CDFGs. These larger graphs are useful for integrating the mode control logic with the already existing state control logic. By integrating them, the mode change logic becomes more difficult to remove.

Once the CDFGs have been created and flattened, branches with large fan-out are selected for modification to allow for the obfuscated mode of functionality.

The CDFGs are then converted back into RTL with the additional obfuscation added, which can then be implemented for a device.

2.6.5 Signal Path Obfuscation

A further form of defense through obfuscation relies on modifying the signal paths between components, as discussed by Wendt et. al. [32]. Figure 2.18 shows a method for obfuscating the signal paths between components such that they cannot be easily followed without knowing the Challenge/Response Pairs (CRPs) of a specific PUF.

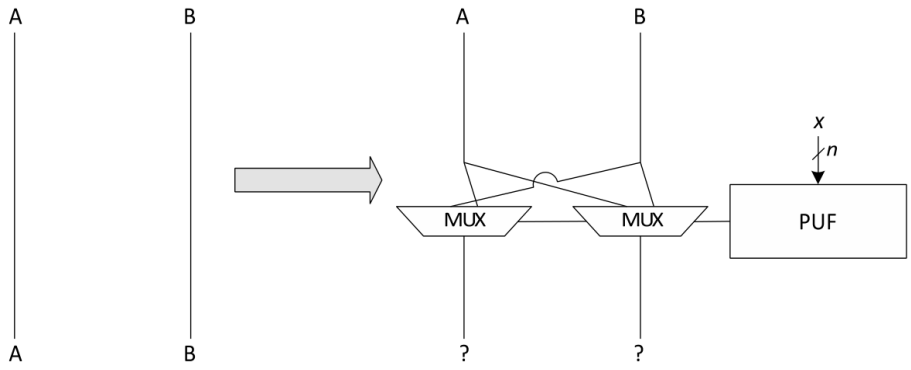


Figure 2.18: Obfuscated Signal Path [32]

This method ensures that it cannot easily be determined how different components connect without knowledge of the set of CRPs for the PUF being used as the select for the multiplexers. On its own, this method would likely not be enough to protect a design, though in conjunction with other methods it could help fully protect a design.

2.6.6 ASIC Cell Camouflaging

Another method of obfuscation, specific to the manufacture process for ASICs uses modified layout-level standard cells for design [33,34]. This approach does not obfuscate the function, per se, but instead obfuscates the physical appearance of the material being manufactured. This was done using a camouflaged standard cell library with modifications to only the dopant layer, keeping all other layers identical. Camouflaging in this way protects mostly against attacks at a foundry while laying out the design, but is also a key defense against attacks where the device is being reverse engineered through use of an X-ray or Scanning Electron Microscope to see the silicon layout of the design. Another obfuscation technique used with this design is to populate the empty space on the chip with arbitrary materials laid out in such a way that makes viewing the design difficult, but does not affect functionality.

2.6.7 Reconfigurable Logic Barriers

Similar to obfuscation, reconfigurable logic barriers can be utilized to protect the design from adversaries [4]. This method is done by sectioning off part of the design to be left reconfigurable after manufacturing. The separation is done in such a way that each device requires a unique design placed in the reconfigurable area to restore proper functionality. The main purpose of this approach is to protect against overproduction at the manufacturing stage, since producing more boards offers the adversary no benefit without the correct reconfigurable designs.

2.7 Attacks on Hardware Obfuscation

2.7.1 Satisfiability Attacks

One of the main attacks on hardware obfuscation is the SAT attack, which uses the Boolean satisfiability problem to attempt to determine the key of an obfuscated circuit. Originally proposed in [26], this attack rules out equivalence classes of keys that do not lead to the correct output. To accomplish this attack, the adversary must have access to both the locked netlist and a functional unlocked IC.

Keys are separated into equivalence classes, which are sets of keys where for each key in the set, the inputs and outputs are equivalent, as denoted in Equation 2.4.

$$\vec{K_1} \equiv \vec{K_2} \iff \forall \vec{X_i} : C(\vec{X_i}, \vec{K_1}, \vec{Y_i}) \wedge C(\vec{X_i}, \vec{K_2}, \vec{Y_i}) \tag{2.4}$$

In the equation, $\vec{K_1}$ and $\vec{K_2}$ refer to the two keys in the equivalence class, and $C(\vec{X_i}, \vec{K}, \vec{Y_i})$ is the function of the circuit with inputs $\vec{X_i}$ and outputs $\vec{Y_i}$.

By considering these equivalence classes, the SAT attack is able to rule out sets of keys rather than individual keys, thus increasing the efficiency of the search. Once

equivalence classes are defined, the attack iteratively rules out incorrect keys through the use of Distinguishing Input Patterns (DIPs). A DIP is defined as an input vector where, for two different keys $\vec{K_1}$ and $\vec{K_2}$ the output is different. This output can then be compared to the output of the unlocked circuit for that input vector and one or both equivalence classes can be ruled out. This allows the attack to rapidly reduce the size of the key space in order to narrow in on the correct key.

A miter-like circuit is created to find the DIPs, which creates two copies of the circuit with the same primary inputs and two different keys [35]. The outputs of these circuits are XORed to determine if they are the same, and then ORed into a single `diff` signal. If any output between the circuits differs, the `diff` signal is asserted high, as shown in Figure 2.19.

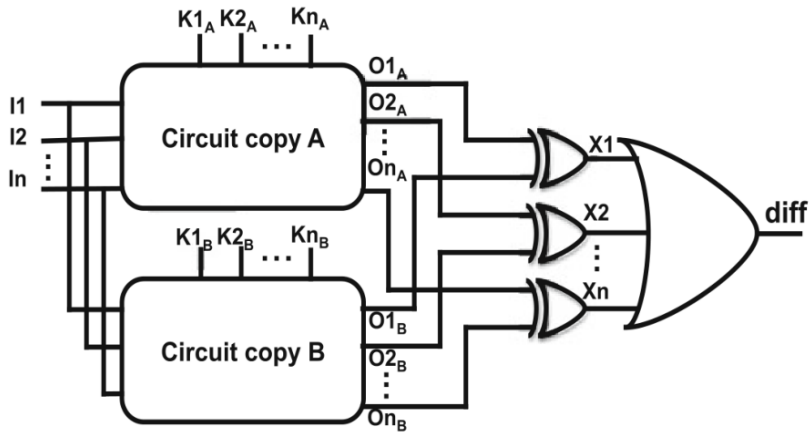


Figure 2.19: DIP Miter Circuit [35]

To use this to find the DIPs, the full miter circuit is converted to its Conjunctive Normal Form (CNF) and a SAT solver is run on the description of the circuit. A DIP has been found once a satisfiable assignment has been found.

Once a DIP has been found, the output of the activated IC can be examined to rule out either or both of the keys used — and by extension, those keys' equivalence classes. This process is repeated either until no further DIPs can be found, indicating that the

equivalence class of the key can no longer be narrowed down and the correct key has been found.

Algorithm 1 describes the full process of the attack.

Algorithm 1 SAT Attack [26]

Inputs: C and $eval$
Output: $\vec{K_C}$

1: **function** DECRYPT
2: $i := 1$
3: $F_1 = C(\vec{X}, \vec{K_1}, \vec{Y_1}) \wedge C(\vec{X}, \vec{K_2}, \vec{Y_2})$
4: **while** $sat[F_i \wedge (\vec{Y_1} \neq \vec{Y_2})]$ **do**
5: $\vec{X_i^d} := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y_1} \neq \vec{Y_2})]$
6: $\vec{Y_i^d} := eval(\vec{X_i^d})$
7: $F_{i+1} = F_i \wedge C(\vec{X_i^d}, \vec{K_1}, \vec{Y_i^d}) \wedge C(\vec{X_i^d}, \vec{K_2}, \vec{Y_i^d})$
8: $i := i + 1$
9: **end while**
10: $\vec{K_C} := sat\_assignment_{\vec{K_1}}(F_i)$

Once the `while` condition (*line 3*) is no longer true, meaning the miter circuit is unsatisfiable, the correct key value $\vec{K_C}$ is output.

2.7.1.1 Tseitin Transformation

The general conversion of a Boolean formula into CNF is an NP-hard problem and can result in an exponential explosion of clauses depending on the circuit being transformed. To resolve this, the tool uses a Tseitin, alternatively known as Tseytin, transformation, which is a method of easily converting gates into proper CNF by including the output of the gate as a variable [36]. This transformation also reduces the complexity of the satisfiability problem by providing only clauses with a maximum of three variables.

The following equations show the derivation for the Tseitin transformation of an AND gate.

An AND gate can be defined by:

$$(C \rightarrow (A \wedge B)) \wedge (\overline{C} \rightarrow (\overline{A} \vee \overline{B})) \tag{2.5}$$

Implications are replaced with equivalent logic using only AND, OR, and NOT:

$$(\overline{C} \vee (A \wedge B)) \wedge (C \vee (\overline{A} \vee \overline{B})) \tag{2.6}$$

The equation is then be put in CNF by using distribution on the leftmost clause:

$$(\overline{C} \vee A) \wedge (\overline{C} \vee B) \wedge (C \vee \overline{A} \vee \overline{B}) \tag{2.7}$$

Table 2.2 shows the applicable Tseitin transformations for all two-input gates and an inverter.

Table 2.2: Tseitin Transformations [37]

Type	Operation	CNF Expression
AND	$C = A \cdot B$	$(\overline{C} \vee A) \wedge (\overline{C} \vee B) \wedge (C \vee \overline{A} \vee \overline{B})$
NAND	$C = \overline{A \cdot B}$	$(C \vee A) \wedge (C \vee B) \wedge (\overline{C} \vee \overline{A} \vee \overline{B})$
OR	$C = A + B$	$(C \vee \overline{A}) \wedge (C \vee \overline{B}) \wedge (\overline{C} \vee A \vee B)$
NOR	$C = \overline{A + B}$	$(\overline{C} \vee \overline{A}) \wedge (\overline{C} \vee \overline{B}) \wedge (C \vee A \vee B)$
XOR	$C = A \oplus B$	$(\overline{A} \vee B \vee C) \wedge (A \vee \overline{B} \vee C) \wedge (A \vee B \vee \overline{C}) \wedge (\overline{A} \vee \overline{B} \vee \overline{C})$
XNOR	$C = \overline{A \oplus B}$	$(\overline{A} \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee C)$
NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$

Figure 2.20 is an example circuit, encoding the equation $(AB + \overline{C})$ obfuscated with two locking gates, to be converted to CNF for a SAT solver using the Tseitin transformation.

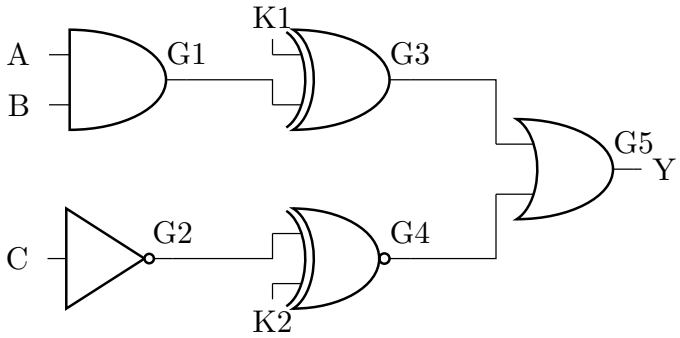


Figure 2.20: Example Circuit for Tseitin Transformation

Each gate output was given a label, G1-G5. These labels are then used to find the Tseitin transformation of each gate, as shown in Table 2.3.

Table 2.3: Tseitin Example Transformations

Gate	CNF Sub-Expression
G1	$(\overline{A} + \overline{B} + G1)(A + \overline{G1})(B + \overline{G1})$
G2	$(\overline{C} + \overline{G2})(C + G2)$
G3	$(\overline{G1} + \overline{K1} + \overline{G3})(G1 + K1 + \overline{G3})(G1 + \overline{K1} + G3)(\overline{G1} + K1 + G3)$
G4	$(\overline{G2} + \overline{K2} + G4)(G2 + K2 + G4)(\overline{G2} + K2 + \overline{G4})(G2 + \overline{K2} + \overline{G4})$
G5	$(G3 + G4 + \overline{G5})(\overline{G3} + G5)(\overline{G4} + G5)(G5)$

Connecting each of the individual CNF formulas with AND operations yields a full CNF equation for the circuit. As gate G5 represents the output of the circuit, an additional lone ($G5$) clause was added to force the circuit output to be one and yield the correct results from a SAT solver.

2.7.2 Bypass Attack

The bypass attack proposed in [30] is an attack that was designed to capitalize on the vulnerabilities of various SAT attack-resistant locking mechanisms, such as SARLock [25]. Based on a SAT attack, once the attacker has found all DIPs for an incorrect key they would be able to use that key and reverse the outputs to become

correct. A "bypass circuit" could be attached alongside the original locked netlist to monitor the DIPs to change incorrect outputs to the correct outputs based on SARLock's *flip* bit.

2.7.3 Key Sensitization Attack

Proposed in [24], the key sensitization attack is performed by determining input patterns that allow key bits to propagate to the outputs of the circuit. This is done using Automatic Test Pattern Generation (ATPG) tools, which are ordinarily used to generate input patterns for fault analysis of circuits [38]. They can, however, be leveraged to determine input patterns that propagate key bits to the output.

A basic example of a circuit that allows a key bit to propagate to the output is shown in Figure 2.21

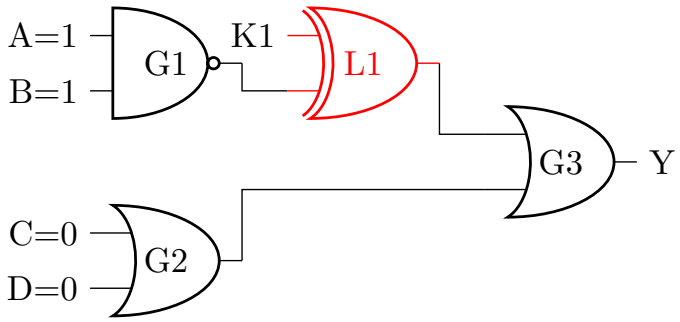


Figure 2.21: Circuit Vulnerable to the Key Sensitization Attack

The input pattern shown in the figure would allow an attacker to sensitize the key bit `K1` to the output `Y` and observe its value.

The SLL method of obfuscation was developed as a means to prevent this type of attack. By ensuring that the gates are connected in such a way that they cannot expose key values, this attack could be rendered ineffective [3].

2.7.4 Signal Probability Skew Attack

An attack was formulated in [39] known as the Signal Probability Skew (SPS) attack, designed to specifically attack the Anti-SAT method of obfuscation. This attack is used to analyze the SPS of gates g and $\overline{g}$. These gates will very likely have the highest skews of the circuit and can be easily singled out and guessed. These gates can also be removed, and the value of their output can be determined by analyzing the skew of the output of the block itself.

2.8 Physical Unclonable Functions

Physical Unclonable Functions (PUFs) are components that produce a unique value per-chip. They utilize variations in the silicon and manufacturing process to ensure that each will give a different response. This is called a Challenge/Response Pair (CRP), and must be unique for each PUF [40].

The space of this CRP can vary based on the type of PUF, either weak or strong. A strong PUF is defined as one with, "so many CRPs such that an attack (performed during a limited amount of time) based on exhaustively measuring the CRPs has only a negligible probability of success" [41]. A weak PUF is one with a restricted CRP space, sometimes only a single CRP. These are often used for creation of a single value per-chip.

PUFs are useful for designs that need a unique identifier or key that differs from instance-to-instance such that no two devices will ever share a key. These values cannot be detected during the manufacturing process/bitstream programming. LUT-Lock utilizes these devices in obfuscation on FPGAs to act as keys for obfuscated gates [28].

2.8.1 PUF Definitions

A PUF must be provably unclonable, unique, and random in order to guarantee security. Barbareschi offers a number of equations for determining if a design fits these characteristics, as well as several definitions that help simplify the language surrounding the components [40].

First, a PUF is formally defined by Equation 2.8.

$$\theta \in \Theta : C \rightarrow R | \theta(c) = r, c \in C, r \in R \tag{2.8}$$

The function θ takes a challenge, c, as an input and returns a response r. C is the set of all challenges, and R is the set of all responses that map to these challenges. The CRP in this case would be defined as the pair $(c, \theta(c)) = (c, r)$, and a PUF is a device which is able to provide a unique CRP for each device.

Unclonability is one of the main properties of PUFs, which means that they cannot be mimicked on another device. This ensures that a design that is dependent on a specific PUF output will only work for a certain challenge on a certain chip. Unclonability is defined by the following series of functions [40]:

$$\nexists f : \theta(c) = f(c) = r, \forall c \in C \tag{2.9}$$

$$\nexists \theta' : \theta(c) = \theta'(c) = r, \forall c \in C \tag{2.10}$$

Both equations state similar properties, but with slightly differing contexts. Equation 2.9 states that a mathematical function f cannot be found that would produce the same set of CRPs as the initial PUF. Equation 2.10 similarly states that it would be hard to create a device with a PUF that can be swapped out for the initial PUF with function θ.

Another major characteristic that all PUFs must adhere to is uniqueness, which is similar to unclonability and can often be mistaken for such, but has a key difference. This property is defined by Equation 2.11.

$$\nexists \tilde{\theta} \in \Theta : \tilde{\theta}(c) = \theta(c) = r, \forall c \in \widehat{C} \subseteq C \tag{2.11}$$

The key distinction of Equation 2.11 is that uniqueness requires that a $\tilde{\theta}$ which belongs to Θ and produces the same CRP cannot exist. It also must only hold valid for a subset of C.

These properties of PUFs allow them to make ideal keys which cannot be discerned at any stage of the manufacture process.

2.9 Substitution Boxes

Security in block ciphers relies on the incorporation of components that add to both the confusion and diffusion of the data. Confusion is the process of making each bit of ciphertext dependent on multiple key bits, thus obscuring the connection between the key and the encrypted data. Diffusion spreads the information in the plaintext out across a substantial amount of the ciphertext, such that if a single bit of plaintext is changed half of the bits of ciphertext will change, on average [42]. A major component in block ciphers that helps create these security measures, specifically confusion, is the substitution box (S-box), which is the main component being obfuscated in this work.

An important quality of S-boxes is that they are nonlinear components, and are typically the only nonlinear components in a block cipher [6]. These components can be designed in a number of ways, as long as they offer nonlinear functionality to provide strong confusion.

2.9.1 MK-3 S-Box

The S-box used in the MK-3 cipher uses 16-bit inputs and outputs which was, at the time, believed to be the largest S-box used in a cryptographic algorithm [43]. As with most other ciphers, this S-box is the main source of confusion and is the only nonlinear element in the algorithm.

The algorithm for MK-3 is designed to be customizable, so that a number of different S-boxes are available for different implementations. This results in the need for the component to be protected, as it is important to guarantee the security of the cipher for its users [44].

This S-box design from Wood's Master's thesis on the topic [6] was used because it could be efficiently implemented in hardware. It functions very similarly to the AES S-box, as it is also created by calculating a multiplicative inverse in a finite field and then performing an affine transformation. This function can be represented as $S(\alpha) = A \cdot \alpha^{-1} + b$, where α denotes the 16-bit input to the function, which is treated as a polynomial in $GF(2^{16})$. The affine transformation is handled by the matrix A and the 16-bit vector b. The multiplicative inverse of α is calculated using an irreducible polynomial $f(x) = x^{16} + x^5 + x^3 + x + 1$.

As this S-box is customizable, the values for $f(x)$, A, and b can be selected to create a unique S-box without jeopardizing the security of the cipher [44].

2.9.2 AES S-Box

AES utilizes an 8-bit S-box component for its confusion. This S-box's function is in two steps: the multiplicative inverse of the input value is taken in the Galois Field of $GF(2^8)$ and then an affine transformation is performed over a binary Galois Field [45]. The design of this particular S-box was chosen based on both nonlinearity — leading

to the lowest possible correlation between inputs and outputs and the lowest possible difference propagation probability — and the algebraic complexity of determining the values [46]. These criteria were chosen to make the confusion for the cipher as strong as possible.

2.10 SIMON Block Cipher

In 2013 the National Security Agency (NSA) designed two families of lightweight block ciphers for use in highly resource constrained devices, SIMON and SPECK [47]. The SIMON family was optimized to be implemented on hardware devices, while the SPECK family of ciphers was optimized to be run in software. Each cipher could be configured to use varying block and key sizes, ranging from 32/64 — denoting a 32-bit block and 64-bit key — to 128/256.

This research will focus on making use of the SIMON family of ciphers as One-way Random Functions (ORFs) for obfuscation.

The SIMON cipher is designed as a Feistel network, thus removing the need for separate encryption and decryption logic. This network consists of a number of rounds and a key scheduler, each of which is determined by the block and key size being implemented.

The round consists of multiple rotation, logical AND, and logical XOR operations, and is shown visually in Figure 2.22.

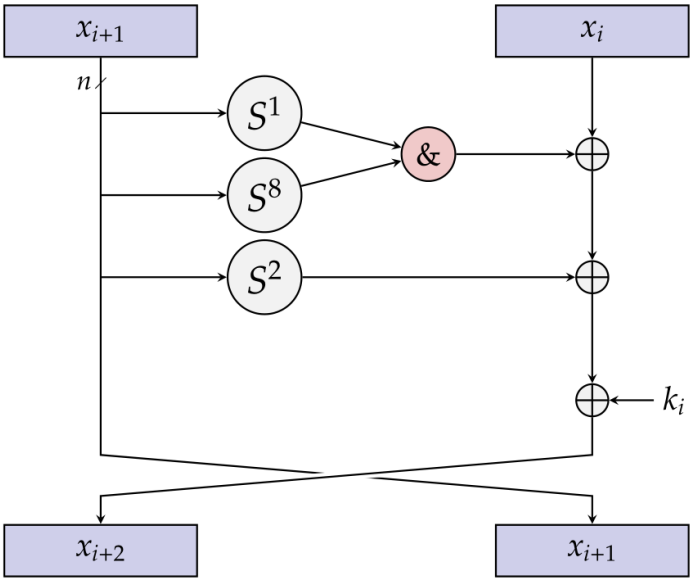


Figure 2.22: One Round of SIMON [47]

In the figure, n is the number of bits being used for the operations, which is half of the configured block size; x_i and x_{i+1} are the upper and lower halves of the block being used in round i; and S^n represents a left circular shift by n bits.

The key scheduler, which produces an n-bit k_i for each round i, is what gives the SIMON cipher the majority of its confusion. It defines five round sequences, $z_0 \dots z_4$, which are used as round constants to remove slide properties and circular shift symmetries. Each of these sequences is used for different block/key size combinations to provide additional separation between similar configurations. The scheduler also uses a round constant c that is defined as $c = 2^n - 4 =$`0xff`$\cdots$`fc`.

The equation for each round key is defined by the piecewise function in Equation 2.12.

$$k_{i+m} = \begin{cases} c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+1}, & m = 2, \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+2}, & m = 3, \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}), & m = 4, \end{cases} \tag{2.12}$$

In this equation, m is defined as the number of key words being used, which varies based on the configuration; and $(z_j)_i$ represents the i^{th} bit of the sequence z_j that is used for that configuration. These configuration-specific values are defined in Table 2.4.

Table 2.4: SIMON Configurations [47]

block size **2n**	key size **mn**	word size **n**	key words **m**	constant **seq**	rounds **T**
32	64	16	4	z_0	32
48	72	24	3	z_0	36
	96	24	4	z_1	36
64	96	32	3	z_2	42
	128	32	4	z_3	44
96	96	48	2	z_2	52
	144	48	3	z_3	54
128	128	64	2	z_2	68
	192	64	3	z_3	69
	256	64	4	z_4	72

Using the information from the table and the function in Equation 2.12, a round key can be made for each round of operation and the block cipher is complete.

The intention is to use SIMON as an ORF, similar to the use of AES from [29].

2.10.1 Round-Reduced SIMON

A round-reduced version of SIMON could possibly be used as an ORF as long as it still provides strong confusion and diffusion properties.

Several works have been published that look at the cryptanalysis of round-reduced SIMON and determine the minimum number of rounds that cannot be broken by several common attacks. In [48] the results were summarized in Table 2.5.

Table 2.5: Summary of results on SIMON. CP = chosen plaintexts, CC = chosen ciphertexts, Att. = attacked, Succ. = success, Ref. = reference. [48]

Cipher	Rounds		Time	Data	Memory	Succ.
	Full	Att.			Bytes	Rate
Differential						
SIMON32/64	32	18	$2^{46.0}$	$2^{31.2}$CP	$2^{15.0}$	0.632
SIMON48/72	36	19	$2^{52.0}$	$2^{46.0}$CC	$2^{20.0}$	0.981
SIMON48/96	36	19	$2^{76.0}$	$2^{46.0}$CC	$2^{20.0}$	0.981
SIMON64/96	42	26	$2^{63.9}$	$2^{63.0}$CP	$2^{31.0}$	0.863
SIMON64/128	44	26	$2^{94.0}$	$2^{63.0}$CP	$2^{31.0}$	0.863
SIMON96/96	52	35	$2^{93.3}$	$2^{93.2}$CP	$2^{37.8}$	0.632
SIMON96/144	54	35	$2^{101.0}$	$2^{93.2}$CP	$2^{37.8}$	0.632
SIMON128/128	68	46	$2^{125.7}$	$2^{125.6}$CP	$2^{40.6}$	0.632
SIMON128/192	69	46	$2^{142.0}$	$2^{125.6}$CP	$2^{40.6}$	0.632
SIMON128/256	72	46	$2^{206.0}$	$2^{125.6}$CP	$2^{40.6}$	0.632
Related-Key Rectangle						
SIMON32/64	32	18	$2^{54.55}$	$2^{30.86}$CP	$2^{32.86}$	0.632
Impossible Differential						
SIMON32/64	32	13	$2^{50.1}$	$2^{30.0}$CP	$2^{20.0}$	≈ 1
SIMON48/96	36	15	$2^{53.0}$	$2^{38.0}$CP	$2^{20.6}$	≈ 1
SIMON64/128	44	17	$2^{71.0}$	$2^{52.0}$CP	$2^{21.0}$	≈ 1
SIMON96/144	54	20	$2^{111.0}$	$2^{84.0}$CP	$2^{19.6}$	≈ 1
SIMON128/256	72	25	$2^{195.0}$	$2^{119.0}$CP	$2^{23.0}$	≈ 1

The results in Table 2.5 suggest that, depending on the attack used and the size of the cipher, as few as 13 rounds can be an effective cipher. Given this, a severely round-reduced implementation could be effective as an ORF.

One reason a round-reduced implementation may be wanted is to account for overhead. Though SIMON is a much smaller block cipher than AES, it can still add substantial overhead to the circuit even with fewer rounds. As this research is being conducted, National Institute of Standards and Technology (NIST) is running a competition to find a new lightweight cipher for use in resource-constrained applications [49]. Though the finalists and winner have not been announced at this time, this could lead to other promising ciphers as ORFs.

Chapter 3

Methodology

Much work has been done in the last several years on improving the security of logic locking algorithms for hardware obfuscation; however, little research has been done that analyzes the effects of these obfuscation methods on cryptographic components. This research seeks to analyze the effectiveness of several methods of obfuscation on non-linear cryptographic components, specifically S-boxes, against SAT attacks. Obfuscating these components would be particularly valuable for any proprietary or customizable cryptographic operations implemented in hardware. Similar work in analyzing different obfuscation methods has been done previously, however this research only tests against a series of relatively basic components from the ISCAS '85 benchmark set [50].

3.1 Circuit Obfuscation Program

A Python program was written which can obfuscate a Verilog gate-level netlist using several logic locking methods. This software was able to parse these netlists into lists of gates and wires and then modify those to implement an obfuscation method determined by the user.

To prepare the circuits for testing, if they were not already in the form of a gate-level

netlist, the circuits were synthesized using Leodardo Spectrum. For this synthesis a generic ASIC standard gate library was used, and a helper script was written in Python which converted the gate definitions from that library into standard Verilog primitives. This process is shown in Figure 3.1.

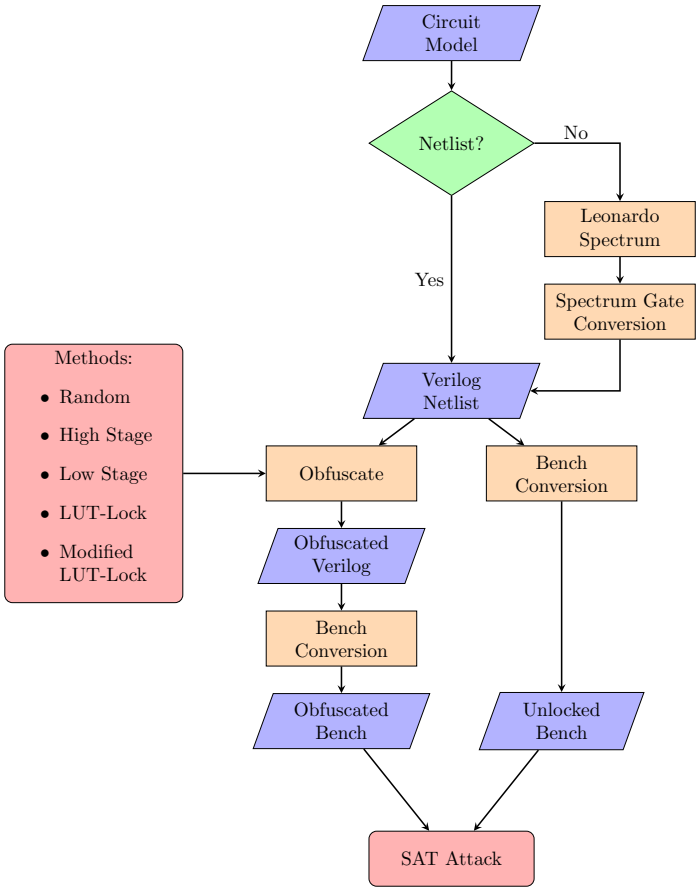


Figure 3.1: Process Workflow

The locking methods implemented were the random locking method proposed in [22], as well as two which place the locking gates at either the highest or the lowest stages of the circuit, and LUT-Lock — a more recently proposed locking method made to specifically defend against SAT attacks [28], and a small modification of LUT-Lock.

The two methods which utilize gate stage as metrics for locking are based on the fully random locking methods, but place all locking gates in either the highest or lowest

stages possible. The stage of a gate is determined by the largest number of gates from primary inputs; so a gate with two primary inputs has a stage of 0, a gate with one primary input and one input which passes through two levels of gates has a stage of 2, and so on. Gate stage values are shown in Figure 3.2.

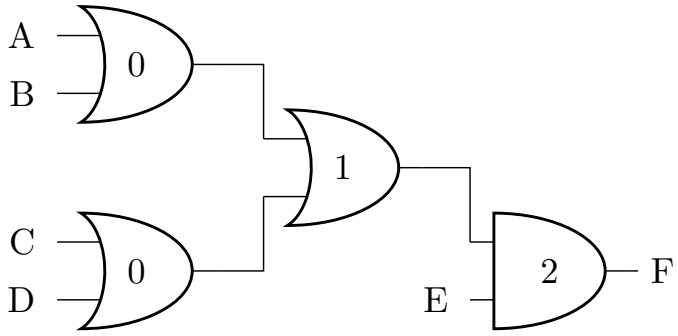


Figure 3.2: Circuit Stages

Each gate is labeled with its stage in the design, which starts from 0 — with primary inputs — and increments with each gate that must be passed through from those inputs.

These obfuscation methods were used to protect both the 8-bit AES S-box as well as the 16-bit MK-3 S-box [43] with key sizes ranging from 16-bits to 128 bits. These methods were also used to obfuscate a subset of the ISCAS ‘85 combinational benchmarks to attempt to replicate some of the work done in [50].

The obfuscated circuits then had a SAT attack run against them using the attack tool from [26]. To use this tool, the netlists had to be converted into bench format, which was accomplished with another Python script.

For each circuit, each obfuscation method was run and then attacked ten times, and the amount of time the attack took to complete was recorded. A timeout was set so that if the attack ran for more than five days it was stopped. This was a requirement set by the server cluster being utilized for testing.

The results for a SAT attack run on the ISCAS benchmarks with random locking gates inserted closely mimicked the results from [50], though that work tested fewer key sizes so a full comparison could not be made.

3.1.1 Modified Random Locking Implementation

The random locking method of encryption is theoretically vulnerable to physical reverse engineering attacks where the attacker could X-ray or otherwise analyze the traces in a chip to find and remove the locking gates. This is possible because the locking gates in this method would only be added into wires, so replacing them with buffers could invalidate the security.

As a countermeasure to this, a method was developed which would randomly either insert locking gates into the internal signals — as was done in the original method — or replace inverters with locking gates. For any inverters being replaced, the locking gate would correspond to the opposite gate — either XOR or XNOR — as was used to obfuscate the internal wires. This should ultimately have no impact on the SAT attack as the effect of each key bit remains the same, it should only increase the difficulty of physical analysis attacks.

3.1.2 LUT-Lock Implementation

The final algorithm proposed in LUT-Lock [28] was referred to as NB2-MO-HSC, which was a combination of several heuristics used to determine which gates should be obfuscated to increase security against SAT attacks. The algorithm which was put forward in the research relied on narrowing down which outputs could be obfuscated based on their timing requirements before locking specific gates. When implementing this algorithm for testing, the timing aspect was ignored, as the designs were eventually intended for use on FPGAs which will adjust routing to ensure timing constraints are met if possible. Should it become clear that these considerations are important for

the security of the design, they could be re-added.

3.1.2.1 Key Programmable Gate Insertion

To test the functionality of a LUT-based obfuscation method, Key Programmable Gates (KPGs) needed to be added. These are gates which, when given the correct key act in the way that they are intended, otherwise they behave differently [51]. As with previous methods, this was most easily accomplished using XOR and XNOR gates, but it can be done with multiplexers or LUTs. XORs and XNORs were ultimately used for this purpose again, as the attack tool would not be able to parse anything other than primitive gates; however, should this algorithm be used on actual devices, the gates would ideally be rolled into LUTs.

Gates of this nature are only required for testing when dealing with FPGAs, since an implementation on an actual device would use PUFs or NLFSRs for key inputs rather than primary inputs.

3.1.2.2 LUT-Lock Modification

A further modification done to the LUT-Lock method was based on how the candidate list of gates was initially formed. In the originally proposed algorithm, a single output would be chosen to be obfuscated first based on the timing requirements. This port's fan-in cone would be obfuscated through until no candidates in that cone could remain, and once that occurred the next output would be obfuscated. This creates a potential issue of incomplete obfuscation in the circuit with keys below a certain size — determined by the size of the circuit itself. If smaller keys were used with this algorithm as it was presented, not all outputs would necessarily be obfuscated before the number of key bits was exceeded. To fix this, rather than fully iterating through the fan-in cone of each output and obfuscating based on the metrics, all outputs were added as candidates at the beginning. This ensured that each output received some

level of obfuscation, as long as the size of the key was greater than the number of output bits.

The algorithm followed the original from this point: finding the child gates from the fan-in cone with the lowest effect on primary outputs and highest SPS, obfuscating that gate, forbidding its children from being obfuscated, and adding the obfuscated gate's children's children to the candidate list. Both this modified form of the LUT-Lock algorithm and the original NB2-MO-HSC method were used for obfuscation testing.

3.2 SIMON as an ORF Experiments

Variations of the SIMON block cipher [47] were used as One-way Random Functions (ORFs) for obfuscating circuits in the manner presented in [29]. The configuration used, prior to any modifications, was the 32/64 implementation of the cipher with a 32-bit block size and a 64-bit key. These netlists were derived from a Very High Speed Integrated Circuit (VHSIC) HDL (VHDL) implementation of the 64/128 SIMON block cipher [52]. The key scheduler from that project was modified to accommodate the 32/64 size cipher and the rounds were resized appropriately.

To attempt to establish a trend that relates the number of rounds of SIMON to its strength as an ORF for this application, several round-reduced implementations of the cipher were created and tested.

Another experiment involved fixing both the plaintext and the key inputs to the cipher, respectively. The purpose of fixing these values for certain tests was to protect against the possibility of removal attacks. As noted in the original paper [29], in reference to the AES, a cipher of this nature could have a known synthesis profile and an adversary could simply remove the relevant gates if it were discovered. By fixing these values prior to synthesis, the post-synthesis model would be effectively unrecognizable.

Before using a possibly round-reduced SIMON for this purpose, the SAT attack's effectiveness against it must be analyzed. To do this, the SAT attack tool from [26] was used. The "unlocked" netlist given was a netlist with a fixed key, as that key was the value being searched for, and the locked netlist was the standard SIMON block. This test was done using round-reduced implementations ranging from 1–8, 16, 24, and 32 rounds.

Following these experiments, the three variations of SIMON were used as ORFs for the MK-3 S-box. The actual number of locking gates used for the obfuscation itself was fixed at 32 to match the size of the SIMON block. For each input variation of SIMON; 2, 4, 6, 8, 10, 12, 14, 16, 24, and 32 round instances were tested, each 10 times.

3.3 Obfuscated Models

Several models were used in the obfuscation experiments, covering a range of sizes and structures.

The first six models come from a set of benchmarks referred to as the ISCAS '85 benchmarks [53]. These benchmarks are a series of standard combinational circuits used for gate-level ATPG testing and fault simulation. The original work did not, however, disclose the functionality of the designs, merely their netlists, so further work [54] has gone into discerning their operations as described in the "Description" column of Table 3.1. This subset of circuits from the full ISCAS '85 set were chosen so that comparisons could be made with the results of the work which created specific obfuscation benchmarks of these particular designs [50]. The netlists used as the base for obfuscation were from [55].

The final two models are instances of S-boxes from both the AES and MK-3 block

ciphers [43,45]. Both were created using composite field techniques to make the results comparable between them. Both of these S-boxes were originally implemented in VHDL; however, as the obfuscation tool only works with files based off of Verilog netlists, the designs needed to be converted to Verilog. This conversion was done using Leodardo Spectrum, which converted the design fully into two-input standard cell gates, plus inverters, based on a standard ASIC library. This differs from the ISCAS '85 circuits used, as they use n-input standard cell gates, as well as including buffers in several designs. This can lead to a misleading understanding of the comparison of number of gates in the circuits, so it must be accounted for in analysis.

These models are described in Table 3.1.

Table 3.1: Obfuscated Models

	Model Name	# Inputs	# Outputs	# of Gates	# of Stages	Description
ISCAS '85	c432	36	7	160	16	27-channel interrupt controller
	c880	60	26	383	23	8-bit ALU
	c1908	33	25	880	39	16-bit error detector/corrector
	c3540	50	22	1669	46	8-bit ALU with binary and BCD arithmetic, logic, and shift operations
	c5315	178	123	2406	48	9-bit ALU
	c7552	207	108	3512	42	34-bit adder and magnitude comparator with input parity checking
S-Boxes	AES S-Box	8	8	182	29	Composite field S-Box for AES block cipher
	MK-3 S-Box	16	16	699	61	Composite field S-Box for MK-3 block cipher

Chapter 4

Results

4.1 Test Setup

To test the effectiveness of several obfuscation methods on protecting against SAT attacks, the attack tool from [26] was used. Each base circuit being tested was obfuscated with each tested key size and then attacked 10 times so averages could be taken. Initially five different key lengths were used: 16-, 32-, 64-, 96-, and 128-bits.

These tests were conducted on a server with Intel Xeon Gold 6150 CPUs running at 2.70GHz. Each instance of a test was run on a single thread and given access to 24GB of RAM with a timeout of five days.

Tests were generated using the Python programs described in Chapter 3 to obfuscate the given netlists and convert them to the appropriate bench format for the attack tool.

4.2 Basic Logic Locking

The first series of tests conducted were basic applications of different locking methods on Verliog netlists using a custom program written in Python. These methods all followed the initial direction of the papers from which they were derived: "EPIC" [22],

in the case of random-based methods, and "LUT-Lock" [28].

"Random" refers to the original obfuscation described in "EPIC". "High Stage" and "Low Stage" refer to the modifications made to the random locking scheme in which the locking gates are places at the highest and lowest stages respectively. "LUT-Lock" is the original NB2-MO-HSC method from the "LUT-Lock" paper, with the only modification being removal of the timing requirements. "Mod. LUT-Lock" describes the modified instance of the NB2-MO-HSC locking scheme where all outputs are added as candidates in the initial formation of the candidate gate list.

4.2.1 ISCAS '85 Benchmarks

These netlist locking methods were first tested on six benchmark circuits from the ISCAS '85 set of benchmarks. This subset of benchmarks was used to serve as a point of comparison against results collected when developing a series of hardware obfuscation benchmarks [50].

Table 4.1 shows the results of the SAT attack breaking various obfuscation methods applied to these circuits.

Table 4.1: Average SAT Attack Break Time (seconds) on ISCAS '85 Benchmarks

Circuit	Key Size	Random	High Stage	Low Stage	LUT-Lock	Mod. LUT-Lock
	16	0.026	0.025	0.021	0.032	0.023
	32	0.041	0.035	0.095	0.081	0.054
c432	64	0.162	0.164	0.208	0.281	0.269
	96	0.480	0.285	0.585	0.304	0.471
	128	0.509	0.563	0.603	0.159	1.101
	16	0.048	0.025	0.055	0.035	0.020
	32	0.062	0.074	0.074	0.131	0.021
c880	64	0.116	0.197	0.143	0.108	0.047
	96	0.306	0.347	0.333	0.198	0.103
	128	0.386	0.616	0.695	0.221	0.347
	16	0.077	0.049	0.067	0.095	0.034
	32	0.103	0.076	0.285	0.237	0.046
c1908	64	0.336	0.172	1.518	0.990	0.367
	96	0.748	0.162	4.904	2.325	0.514
	128	1.677	0.303	7.172	3.352	1.193
	16	0.210	0.102	0.558	0.133	0.077
	32	0.312	0.192	1.084	0.265	0.109
c3540	64	0.982	1.498	1.581	0.666	0.260
	96	1.142	5.069	1.788	0.873	0.559
	128	1.484	6.939	2.498	1.924	0.752
	16	0.219	0.150	0.244	0.189	0.113
	32	0.365	0.213	0.353	0.358	0.113
c5315	64	0.541	0.506	0.545	0.647	0.114
	96	0.824	0.962	0.822	0.665	0.115
	128	0.924	2.294	1.089	1.006	0.135
	16	0.265	0.241	0.304	0.282	0.169
	32	0.488	0.346	0.455	0.401	0.170
c7552	64	0.709	0.640	0.746	3.599	0.168
	96	1.149	1.101	1.345	0.970	0.168
	128	1.486	1.826	1.750	26.661	0.338

The results of random locking on the benchmarks closely mimic the results of similar tests from [50]; however not all key sizes tested in Table 4.1 are represented in that work. For five of the six tested ISCAS benchmarks, only 32-bit keys were tested, with c7552 having testing results for 32-, 64-, and 128-bit keys.

Results of the tests on these benchmarks show that as the number of gates in the circuits increased, as did the general resilience to the SAT attack. However, all circuits were broken exceedingly quickly, regardless of obfuscation method, with only some outliers taking more than 1 second.

4.2.2 Substitution Boxes

The same tests which were performed on the ISCAS benchmarks were then performed on the S-boxes from both the AES and MK-3. The results of these tests are shown in Table 4.2.

Table 4.2: Average SAT Attack Break Time (seconds) on S-boxes. "Timeout" indicates unsuccessful attack after 5 days (432,000 seconds).

Circuit	Key Size	Random	High Stage	Low Stage	LUT-Lock	Mod. LUT-Lock
AES S-Box	16	0.194	0.084	0.118	0.121	0.552
	32	3.098	3.582	0.408	0.561	5.788
	64	86.166	157.058	1.613	15.021	25.137
	96	401.362	215.286	102.879	158.688	307.201
	128	1,114.959	220.349	2,689.538	219.187	1,048.692
MK-3 S-Box	16	582.454	281.531	165.035	490.311	946.375
	32	692.276	253.747	81.716	24,973.522	2,676.279
	64	270,519.970	32,865.659	482.897	281,140.660	59,992.760
	96	Timeout	298,616.340	43,736.577	414,580.100	333,862.340
	128	Timeout	Timeout	Timeout	Timeout	Timeout

As these show the averages over 10 tests, any tests which timed out show that all tests

with a certain key size and locking method timed out, while tests with values nearly at the timeout (such as LUT-Lock with a 96-bit key on the MK-3 S-box) indicate that some tests timed out and some did not.

Figures 4.1 and 4.2 show the results from Table 4.2 as bar charts. The y-axis uses a logarithmic scale, as the amount of time required to break the obfuscation increased drastically in most cases as the keys increased in size.

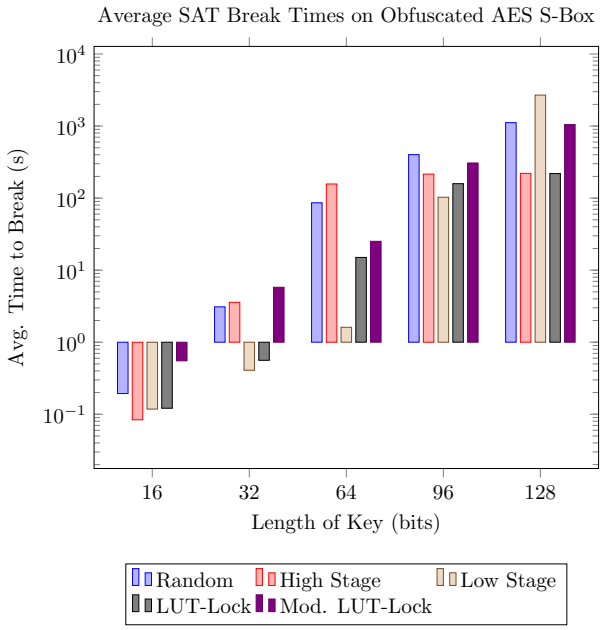


Figure 4.1: SAT Attack on AES S-Box obfuscated with different methods

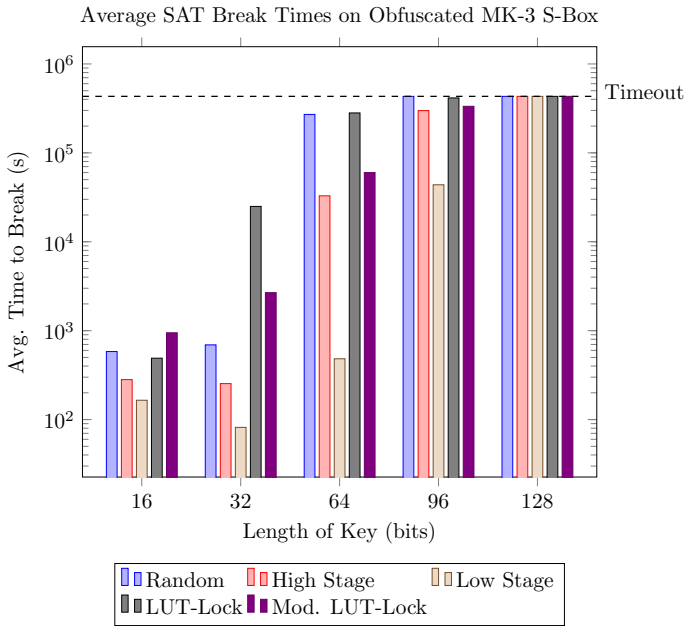


Figure 4.2: SAT Attack on MK-3 S-Box obfuscated with different methods. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

As the results show, random locking is generally the most effective defense against SAT attacks on S-boxes such as these, despite LUT-Lock being an algorithm designed specifically to combat this attack. LUT-Lock and its modified version both performed rather well, though, depending on the test. The modifications made to the algorithm appear to have improved its performance with the smaller AES S-box, while the MK-3 S-box benefited notably more from the LUT-Lock algorithm as it was initially proposed.

There were some anomalies in the results which currently do not have explanations — specifically the low stage tests on the AES S-box with a 128-bit key. This obfuscation method, though generally performing much worse than all other methods tested, took significantly longer to break than the other methods in that test instance.

To further analyze the effect of key size on obfuscation effectiveness and possibly help determine a suitable minimum key size for some of the tested obfuscation methods,

a series of tests were run with finer granularity of key sizes — shown in Figures 4.3 and 4.4. Between the 32-bit and 64-bit keys, additional keys spaced out by 8-bits were tested, but from 64 to 128-bits additional keys were only separated by 4-bits.

Figure 4.3 shows the finer grain keys from 32 to 96-bits, and Figure 4.4 shows the finer grain keys from 96 to 128-bits.

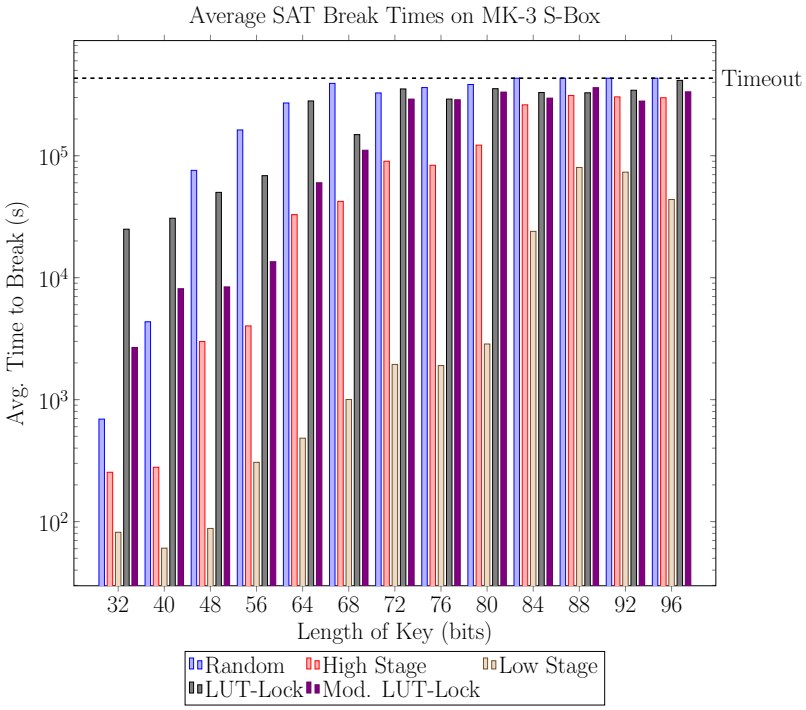


Figure 4.3: SAT Attack on MK-3 S-Box obfuscated with different methods. Finer grain keys between 40 and 96-bits "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

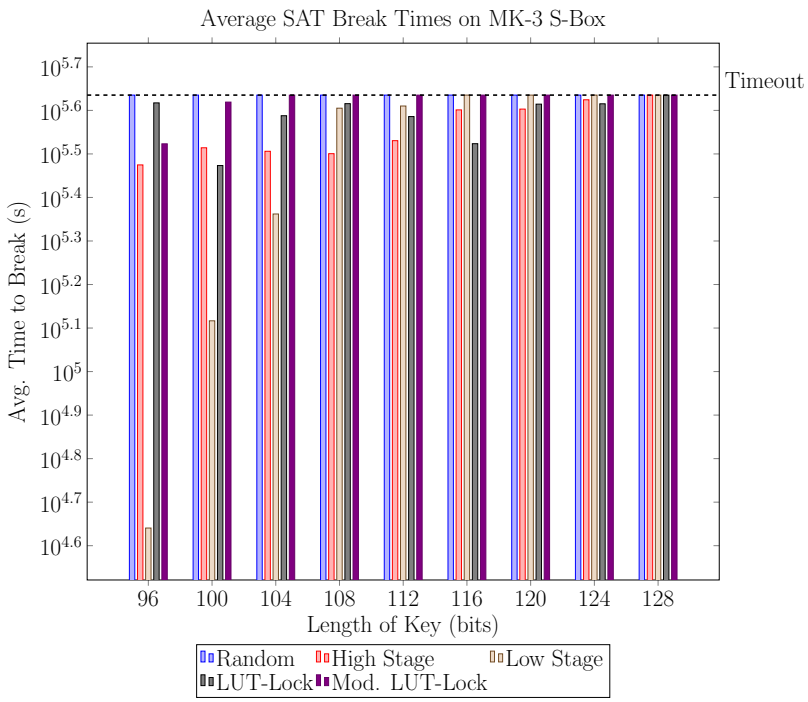


Figure 4.4: SAT Attack on MK-3 S-Box obfuscated with different methods. Finer grain keys between 96 and 128-bits "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

As shown in the figures: past 84-bits, random locking was consistently reaching the five day timeout on all tests, and all other methods showed a general positive trend as the number of key bits increased. They did not all consistently reach the timeout until a 128-bit key was used. The modified implementation of LUT-Lock, though generally performing worse than the unmodified implementation, did reach a point of consistently timing out across all tests with only a 104-bit key while the unmodified version was still being broken for some tests. A similar phenomenon was seen with the low-stage random locking: though performing substantially worse than all other methods for almost all key sizes tested, it began consistently reaching the timeout of the tests with a 116-bit key, where both high-stage and normal LUT-Lock were still being broken.

These results generally aligned with what was expected based on previous tests, and

confirmed that for random locking of the MK-3 S-box an 84-bit key may be considered suitably large, where other methods may require up to a 128-bit key. Further analysis may be necessary given additional testing, though, especially given the number of tests which were being broken close to the timeout value. If a longer timeout could be set, then the additional information gained could further point to what would be a suitably large key size for all tested methods.

4.2.2.1 Comparing ISCAS and Substitution Box Results

These results, when compared to those of the ISCAS benchmarks, indicate that there is an element of the design of these S-boxes which makes them naturally more resistant to SAT attacks. The two ISCAS benchmarks which are closest in size to the S-boxes are c432 and c1908, which correspond to the AES and MK-3 S-boxes respectively. When comparing the amount of time these circuits were able to withstand the SAT attack, it is clear that there is a difference of orders of magnitude.

This large difference for similarly sized designs may indicate that the inherent non-linearity of the S-boxes may increase the complexity of the Boolean SAT problem. Further analysis of these circuit structures would be required, but may be able to indicate types of modifications which could be made which will naturally increase a design's resiliency against the SAT attack.

4.2.3 Modifications to Locking Methods

Three modifications to previously published locking methods were tested with both the ISCAS benchmarks and the S-boxes: high stage random locking, low stage random locking, and a modified NB2-MO-HSC algorithm from "LUT-Lock". These each performed generally distinctly from the locking methods that they are derived from, which could further indicate the effects of lock placement on SAT attack effectiveness.

4.2.3.1 Stage-Based Locking

The high and low stage random locking methods both performed generally much worse than the original purely random locking method; however, placing the obfuscating gates in higher circuit stages almost always performed substantially better than placing them in lower stages. This is supported by several of the preliminary algorithms from "LUT-Lock" [28]. Being closer to the outputs generally results in much lower output corruption and higher SPS, which has been shown to increase the difficulty of the SAT attack. The opposite should then be true of placing gates at lower stages, where output corruption would be maximized and the locking gates would be much more easily controlled by changes in the input patterns.

An interesting phenomenon occurred when using the high stage locking on the MK-3 S-box: each successful SAT attack only took a single iteration, and each unsuccessful one was stuck on processing that first iteration. This behavior was not seen with any other circuit being tested. This would seem to indicate that an aspect of the design of this S-box combined with the locking gates being placed as close to the outputs as possible creates a situation where finding a DIP might be difficult, but once a single DIP is found it will always lead to the correct key.

4.2.3.2 Modified LUT-Lock

The modification to LUT-Lock which included all output gates in the initial obfuscation candidate list seemed to behave inconsistently in relation to the original implementation. The purpose of this modification was to ensure that all outputs would have at least one gate of obfuscation if the number of key bits allowed for it; however this generally seemed to decrease the effectiveness of the algorithm. This dynamic is reversed when obfuscating the AES S-box, though it is unclear why that would be the case.

4.3 ORF Added Logic Locking

Following the tests of basic logic locking on the ISCAS benchmarks and S-boxes, tests following the ORF insertion scheme from [29] were performed. The ORF suggested in that research was an implementation of AES with a fixed key; however, AES is a large block cipher and could incur too much overhead to be feasible. It is for this reason that SIMON, a small hardware-optimized block cipher, was used as an ORF for these tests [56].

To attempt to find a trend between number of rounds and effectiveness as an ORF, several round-reduced implementations on SIMON were tested. The first tests run were attacks on round-reduced implementations of the SIMON cipher itself. 1–8, 16, 24, and 32 round implementations of the SIMON 32/64 block cipher were tested using the SAT attack. For this attack, the "unlocked" netlist was a netlist with a fixed key synthesized in, and the "obfuscated" netlists were the various implementations of SIMON with the SAT attack searching for the key. The tool requires two netlists since an important step in the SAT attack algorithm is a comparison of the output of locked netlists against the correct output of the circuit. These netlists are depicted as blocks in Figure 4.5.

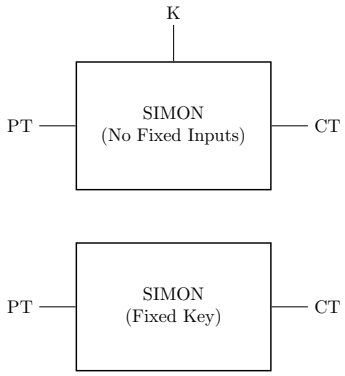


Figure 4.5: SAT Attack on SIMON Netlists

The top block shows the SIMON netlist with no fixed inputs — this is treated by the tool as the "obfuscated" or "locked" circuit. The bottom block is an implementation of SIMON with a fixed key; this key value is what the attack is trying to determine for the "locked" implementation.

The results of running these tests are shown in Table 4.3.

Table 4.3: SAT Attack Break Time (seconds) on SIMON. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

# of Rounds	Time to Break
1	0.010
2	0.017
3	0.034
4	0.082
5	2.596
6	1,845.920
7	Timeout
8	Timeout
16	Timeout
24	Timeout
32	Timeout

The implementations with 1–6 rounds were broken rather quickly, but 7 rounds or more were not broken in the given timeout range with this attack. Given that, implementations used as ORFs for obfuscation should not be broken if 7 or more rounds are implemented.

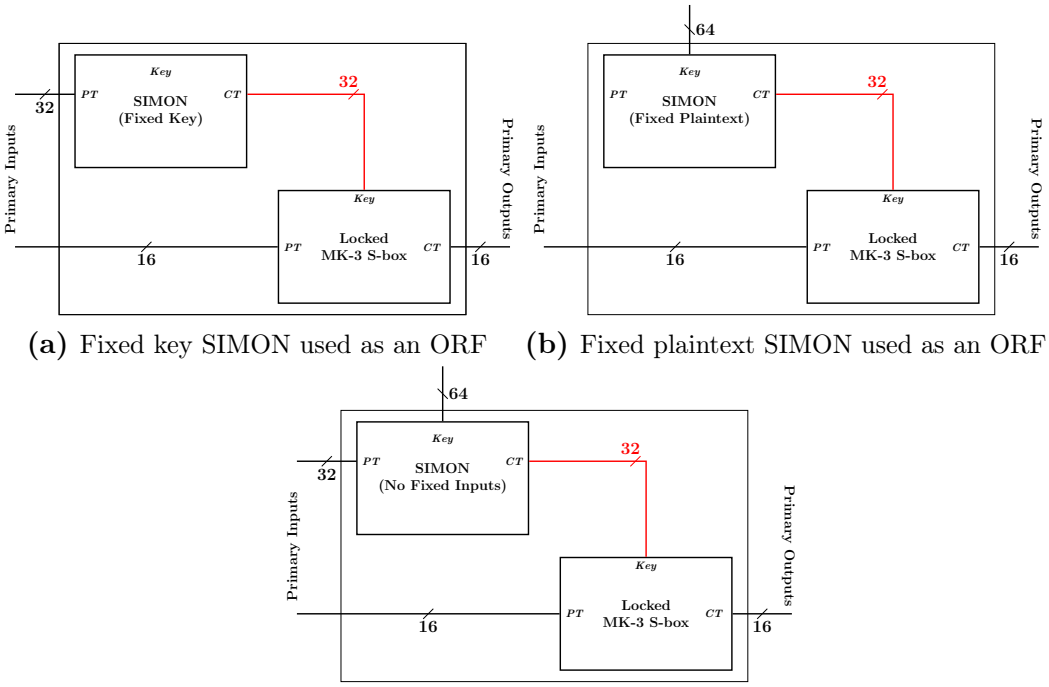
Tests similar to those done with basic logic locking were conducted using SIMON as an ORF between the primary key inputs of the circuit and the keys to the gates.

Three base configurations of SIMON were used for this test: no fixed inputs, fixed key, and fixed plaintext. The fixed input implementations were tested as ways to possibly combat removal attacks as well as provide additional security against SAT attacks. As the base configuration of this cipher is published, it is possible that an adversary could identify the structure in a netlist and remove it, thus invalidating the protection. By fixing an input before synthesis it should create an unrecognizable structure and possibly prevent the removal. The standard implementation was then also examined to determine if having the full cipher gave additional security against SAT attacks, despite being more susceptible to removal attacks.

Rather than modulate key size in the tests, the number of rounds of SIMON was changed. This is done to analyze the trends in the amount of security each implementation offers and to determine if the full cipher is required or if a round-reduced implementation could be used and still provide substantial additional security.

The locking method used for these tests was the random locking scheme with a 32-bit key. The random method was chosen as it generally offered the highest security for most circuits tested, and a 32-bit key was used to correspond to the 32-bit output of the SIMON block.

Figure 4.6 gives example block diagrams for the test setups on the MK-3 S-box.



(a) Fixed key SIMON used as an ORF **(b)** Fixed plaintext SIMON used as an ORF

(c) No fixed input SIMON used as an ORF

Figure 4.6: SIMON ORF MK-3 S-box Test Setups

Table 4.4 shows the results of these tests on the MK-3 S-box.

Table 4.4: Average SAT Attack Break Time (seconds) on MK-3 S-box with SIMON as an ORF. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

# of Rounds	Fixed Key	Fixed Plaintext	No Fixed Inputs
2	927.226	461.267	832.390
4	1,359.334	934.231	2,047.327
6	4,108.339	Timeout	1,612.064
8	1,683.022	Timeout	2,462.465
10	5,228.145	Timeout	70,289.770
12	7,944.420	Timeout	416,078.200
14	10,679.470	Timeout	Timeout
16	24,125.180	Timeout	Timeout
24	313,615.620	Timeout	Timeout
32	388,226.900	Timeout	Timeout

Table 4.5 shows the results of these tests on the AES S-box.

Table 4.5: Average SAT Attack Break Time (seconds) on AES S-box with SIMON as an ORF. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

# of Rounds	Fixed Key	Fixed Plaintext	No Fixed Inputs
2	6.222	4.409	10.256
4	13.433	14.359	12.317
6	105.381	336,388.860	39.082
8	3,888.765	Timeout	6,980.247
10	21,231.967	Timeout	141,253.720
12	90,308.115	Timeout	419,342.900
14	280,292.180	Timeout	Timeout
16	319,193.110	Timeout	Timeout
24	414,411.000	Timeout	Timeout
32	Timeout	Timeout	Timeout

Table 4.6 shows the results of these tests on the ISCAS '85 c432 benchmark.

Table 4.6: Average SAT Attack Break Time (seconds) on ISCAS c432 with SIMON as an ORF. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

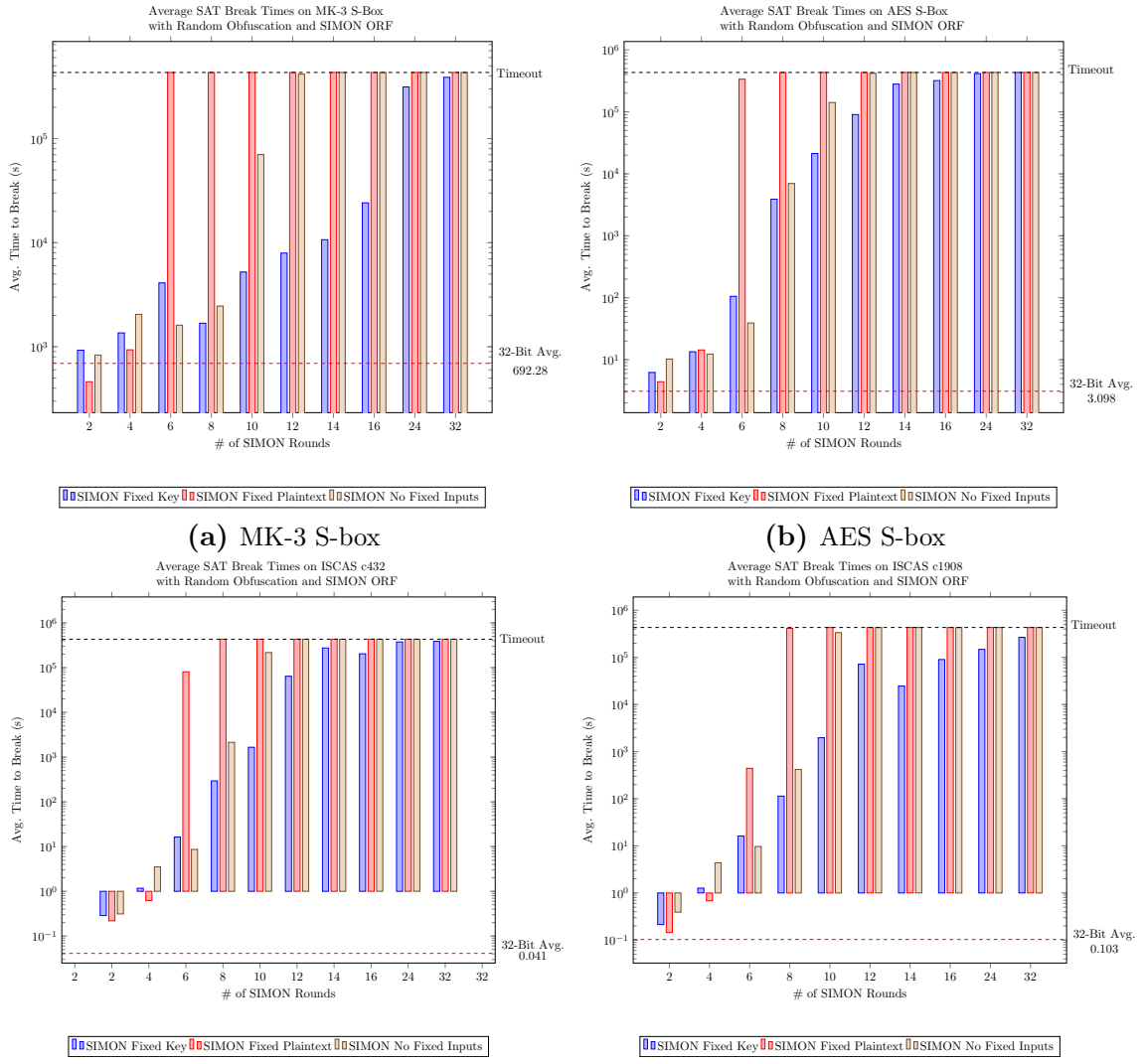
# of Rounds	Fixed Key	Fixed Plaintext	No Fixed Inputs
2	0.287	0.219	0.312
4	1.170	0.620	3.518
6	16.349	80,477.414	8.638
8	293.527	Timeout	2,139.059
10	1,660.054	Timeout	217,207.610
12	64,247.510	Timeout	Timeout
14	274,615.330	Timeout	Timeout
16	203,958.718	Timeout	Timeout
24	374,895.330	Timeout	Timeout
32	386,263.700	Timeout	Timeout

Table 4.7 shows the results of these tests on the ISCAS '85 c1908 benchmark.

Table 4.7: Average SAT Attack Break Time (seconds) on ISCAS c1908 with SIMON as an ORF. "Timeout" indicates an unsuccessful attack after 5 days (432,000 seconds).

# of Rounds	Fixed Key	Fixed Plaintext	No Fixed Inputs
2	0.213	0.145	0.391
4	1.266	0.682	4.372
6	16.230	442.660	9.673
8	113.747	413,649.900	417.311
10	1,974.259	Timeout	335,260.050
12	71,911.856	Timeout	Timeout
14	24,669.020	Timeout	Timeout
16	90,233.257	Timeout	Timeout
24	148,126.070	Timeout	Timeout
32	266,980.260	Timeout	Timeout

Figure 4.7 gives these data in bar charts.



(a) MK-3 S-box

(b) AES S-box

(c) ISCAS '85 c432

(d) ISCAS '85 c1908

Figure 4.7: SAT Attack Results on Locked Circuits with SIMON Configurations as ORFs

Most implementations with as few as two rounds showed a greater resistance to the attack than the average amount of time the SAT attack took to be successful with only basic logic locking, but continuing to increase the number of rounds showed substantial differences between each configuration. Across all circuits and implementations the tests which used a fixed key performed worse than the other two methods, though still offered notable added security over the basic implementations without an ORF.

The tests which did not fix any input value, timed out above 14 rounds and offered

drastic security improvements for all implementations above 10 rounds. This may not have performed perfectly at 8 rounds, despite the cipher itself not being broken with that many rounds, because there are a greater number of possible keys which could lead to the correct output. As the outputs of the cipher are connected to the obfuscation key gates, any input combinations which give the cipher that output would be valid. This should lead to multiple possible input vectors, at least 2^{32} as that is the maximum possible plaintext values, that could give the correct output. This should remain a rather small subset of all 2^{96} possible input vectors (both key and plaintext), but could weaken the cipher enough to make implementations with fewer rounds more vulnerable than they would otherwise be.

The test which showed the most rapid improvement in SAT attack resistance with increasing rounds was the implementation of SIMON synthesized with a fixed plaintext input. Almost all tests conducted with an implementation of SIMON using 8 or more rounds and a fixed plaintext timed out at the maximum 5 days. This method is also susceptible to the issue of multiple valid keys, however with a fixed plaintext there will be fewer such keys, which may indicate the increased security.

Fixing the plaintext seems to retain the most cryptographic strength as well as possibly providing some protection against a removal attack, though more testing with additional plaintext values must be conducted. The removal protection may also be extremely minimal, as fixing the plaintext values should only affect the first 2 rounds and would then have a very small impact on the synthesized design of the circuit.

Ultimately, adding SIMON as an ORF seems to vastly increase security in both the cases of fixed plaintext and no fixed inputs regardless of the circuit it was applied to. All ISCAS benchmarks tested with only logic locking were broken on the order of seconds, but by adding this additional circuit the two tested benchmarks were able to withstand the attack for at least five days.

4.3.1 Overhead

Including SIMON does incur some overhead which must be accounted for when designing a circuit. Table 4.8 gives data on the gate overhead for each configuration of SIMON that was tested.

Table 4.8: SIMON 32/64 Overheads (# of 2-Input Gates)

Rounds:	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**24**	**32**
Fixed Key	107	211	320	433	551	659	770	870	1338	1765
Fixed Plaintext	67	200	435	675	919	1155	1391	1628	2583	3540
No Fixed Inputs	128	256	495	717	955	1194	1432	1672	2634	3586

Each implementation and number of rounds adds a different amount of overhead to go with the different amount of security offered. The configuration which uses a fixed key is the smallest for more than 8 rounds, but also offers the least security. Fixed plaintext, while substantially stronger than fixed key, greatly increases the number of gates necessary to implement the function as the number of rounds increases. The implementation with no fixed inputs, which had similar results to that of the fixed plaintext, has the largest implementations for all numbers of rounds, but is only slightly more than the fixed plaintext configuration.

When using an ORF, the overhead of the obfuscation method itself must also be accounted for. For all of the discussed netlist logic locking methods, the overhead is one gate per key bit.

Other lightweight ciphers could also be considered for use as ORFs moving forward. The NIST competition for lightweight ciphers [49] is ongoing at the time of this research and could lead to ciphers which provide more security than SIMON with a smaller footprint.

Ultimately, the utility of each of these implementations and ciphers must be determined on a case-by-case basis, based on the size of the design being obfuscated, the security requirements of that component, and the resources available.

Chapter 5

Conclusions

The inclusion of different hardware obfuscation techniques has been shown to be an effective method of preventing SAT attacks. The methods explored in this work offered varying levels of success, with random gate placement and ORF insertion ultimately offering the most security for the tested components, in some cases increasing the attack time from fractions of a second to several days. Additionally, substitution boxes showed a much higher resistance to SAT attacks than the standard ISCAS '85 benchmarks, which could indicate that non-linear components are, by design, harder to break in this way. Further work needs to be done to analyze what specific aspect of the S-boxes increased the complexity of the SAT attack and what other types of components could offer this same strength.

Other future work might include examining other lightweight ciphers for use as ORFs for obfuscation, analyzing the effectiveness of other attacks on these cryptographic components, or better determining the effectiveness of different configurations of SIMON as an ORF.

Bibliography

[1] "IEEE recommended practice for encryption and management of electronic design intellectual property (IP)," 2015. [Online]. Available: http://dx.doi.org/10.1109/ieeestd.2015.7274481

[2] S. Bhunia and M. Tehranipoor, *Hardware Security*. Elsevier, 2019. [Online]. Available: http://dx.doi.org/10.1016/c2016-0-03251-5

[3] K. Zamiri Azar, H. Mardani Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," pp. 471–476, 05 2019.

[4] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design & Test of Computers*, vol. 27, pp. 66–75, 1 2010. [Online]. Available: http://dx.doi.org/10.1109/mdt.2010.24

[5] C. A. Wood, S. P. Radziszowski, and M. Lukowiak, "Constructing large S-boxes with area minimized implementations," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*. IEEE, 10 2015, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/milcom.2015.7357417

[6] C. A. Wood, "Large substitution boxes with efficient combinational implementations," mastersthesis, Rochester Institute of Technology, 2013. [Online]. Available: https://scholarworks.rit.edu/theses/5527

[7] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, pp. 1–34, 12 2016. [Online]. Available: http://dx.doi.org/10.1145/2755563

[8] H. Yu, H. Lee, S. Lee, Y. Kim, and H.-M. Lee, "Recent advances in FPGA reverse engineering," *Electronics*, vol. 7, p. 246, 10 2018. [Online]. Available: http://dx.doi.org/10.3390/electronics7100246

[9] J.-B. Note and E. Rannaud, "From the bitstream to the netlist," in *the 16th international ACM/SIGDA symposium*. ACM Press, 2 2008, inproceedings. [Online]. Available: http://dx.doi.org/10.1145/1344671.1344729

[10] F. Benz, A. Seffrin, and S. A. Huss, "Bil: A tool-chain for bitstream reverse-engineering," in *2012 22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 8 2012, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/fpl.2012.6339165

[11] M. Lasser, "Reverse engineering FPGAs," 12 2017. [Online]. Available: https://media.ccc.de/v/34c3-9237-reverse_engineering_fpgas

[12] M. Holler, M. Guizar-Sicairos, E. H. R. Tsai, R. Dinapoli, E. Müller, O. Bunk, J. Raabe, and G. Aeppli, "High-resolution non-destructive three-dimensional imaging of integrated circuits," *Nature*, vol. 543, pp. 402–406, 3 2017. [Online]. Available: http://dx.doi.org/10.1038/nature21698

[13] N. Asadizanjani, M. Tehranipoor, and D. Forte, "PCB reverse engineering using nondestructive x-ray tomography and advanced image processing," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, pp. 1–8, 2017. [Online]. Available: http://dx.doi.org/10.1109/tcpmt.2016.2642824

[14] J. Vosatka, *Introduction to Hardware Trojans*. Springer International Publishing, 11 2018, pp. 15–51. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-68511-3_2

[15] R. S. Chakraborty and S. Bhunia, "Security against hardware trojan attacks using key-based design obfuscation," *Journal of Electronic Testing*, vol. 27, pp. 767–785, 12 2011. [Online]. Available: http://dx.doi.org/10.1007/s10836-011-5255-2

[16] L. Lin, W. Burleson, and C. Paar, "Moles," in *the 2009 International Conference*. ACM Press, 11 2009, inproceedings. [Online]. Available: http://dx.doi.org/10.1145/1687399.1687425

[17] U. Guin, D. Forte, and M. Tehranipoor, "Anti-counterfeit techniques: From design to resign," in *2013 14th International Workshop on Microprocessor Test and Verification (MTV)*. IEEE, 12 2013, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/mtv.2013.28

[18] Y. M. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *16th USENIX Security Symposium (USENIX Security 07)*. Boston, MA: USENIX Association, August 2007, inproceedings. [Online]. Available: https://www.usenix.org/conference/16th-usenix-security-symposium/active-hardware-metering-intellectual-property-protection

[19] *Vivado Design Suite User Guide: Creating and Packaging Custom IP*, Xilinx Inc., 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1118-vivado-creating-packaging-custom-ip.pdf

[20] G. Wassermann. (2017) IEEE P1735 implementations may have weak cryptographic protections. [Online]. Available: https://www.kb.cert.org/vuls/id/739007/

[21] A. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design IP protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 1236–1252, 2001. [Online]. Available: http://dx.doi.org/10.1109/43.952740

[22] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *2008 Design, Automation and Test in Europe*. IEEE, 3 2008, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/date.2008.4484823

[23] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic encryption: A fault analysis perspective," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE 2012)*. IEEE, 3 2012, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/date.2012.6176634

[24] ——, "Security analysis of logic obfuscation," in *the 49th Annual Design Automation Conference*. ACM Press, 6 2012, inproceedings. [Online]. Available: http://dx.doi.org/10.1145/2228360.2228377

[25] 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)*. IEEE, 5 2016, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/hst.2016.7495588

[26] 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)*. IEEE, 5 2015, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/hst.2015.7140252

[27] 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, pp. 199–207, 2 2019. [Online]. Available: http://dx.doi.org/10.1109/tcad.2018.2801220

[28] H. Mardani Kamali, K. Zamiri Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2018, inproceedings, pp. 405–410.

[29] 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, pp. 1411–1424, 9 2016. [Online]. Available: http://dx.doi.org/10.1109/tcad.2015.2511144

[30] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," in

Cryptographic Hardware and Embedded Systems – CHES 2017, W. Fischer and N. Homma, Eds. Cham: Springer International Publishing, 2017, inproceedings, pp. 189–210.

[31] R. S. Chakraborty and S. Bhunia, "RTL hardware IP protection using key-based control and data flow obfuscation," in *2010 23rd International Conference on VLSI Design: concurrently with the 9th International Conference on Embedded Systems Design (VLSID)*. IEEE, 1 2010, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/vlsi.design.2010.54

[32] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 11 2014, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/iccad.2014.7001362

[33] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *the The 51st Annual Design Automation Conference*. ACM Press, 6 2014, inproceedings. [Online]. Available: http://dx.doi.org/10.1145/2593069.2602554

[34] S. Malik, G. T. Becker, C. Paar, and W. P. Burleson, "Development of a layout-level hardware obfuscation tool," in *2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 7 2015, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/isvlsi.2015.118

[35] J. Rajendran and S. Garg, *Logic Encryption*. Springer International Publishing, 1 2017, pp. 71–88. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49019-9_3

[36] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*. Springer Berlin Heidelberg, 1983, pp. 466–483. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-81955-1_28

[37] Wikipedia, "Tseytin transformation — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Tseytin\%20transformation&oldid=962672219, 2020, [Online; accessed 17-June-2020].

[38] D. Vontela and S. Ghosh, "Methodologies to exploit atpg tools for de-camouflaging," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 3 2017, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/isqed.2017.7918324

[39] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-SAT," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 1 2017, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/aspdac.2017.7858346

[40] M. Barbareschi, *Notions on Silicon Physically Unclonable Functions*. Springer International Publishing, 1 2017, pp. 189–209. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-44318-8_10

[41] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA Intrinsic PUFs and Their Use for IP Protection*. Springer Berlin Heidelberg, 2007, pp. 63–80. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74735-2_5

[42] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[43] M. Kelly, A. Kaminsky, M. Kurdziel, M. Ł ukowiak, and S. Radziszowski, "Customizable sponge-based authenticated encryption using 16-bit S-boxes," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*. IEEE, 10 2015, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/milcom.2015.7357416

[44] P. Bajorski, A. Kaminsky, M. Kurdziel, M. Lukowiak, and S. Radziszowski, "Customization modes for the Harris MK-3 authenticated encryption algorithm," in *MILCOM 2018 - IEEE Military Communications Conference*. IEEE, 10 2018, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/milcom.2018.8599712

[45] N. I. of Standards and Technology, *FIPS PUB 197: Advanced Encryption Standard (AES)*. Gaithersburg, MD, USA: National Institute of Standards and Technology, November 2001. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

[46] J. Daemen and V. Rijmen, *The Design of Rijndael*. Springer Berlin Heidelberg, 2002. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-04722-4

[47] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," 2013. [Online]. Available: https://github.com/nsacyber/simon-speck

[48] F. Abed, E. List, S. Lucks, and J. Wenzel, "Differential and linear cryptanalysis of reduced-round simon," Cryptology ePrint Archive, Report 2013/526, 2013. [Online]. Available: https://eprint.iacr.org/2013/526

[49] NIST, "Lightweight cryptography," 2018. [Online]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography

[50] S. Amir, B. Shakya, X. Xu, Y. Jin, S. Bhunia, M. Tehranipoor, and D. Forte, "Development and evaluation of hardware obfuscation benchmarks," *Journal of Hardware and Systems Security*, vol. 2, pp. 142–161, 6 2018. [Online]. Available: http://dx.doi.org/10.1007/s41635-018-0036-3

[51] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan, "Benchmarking the capabilities and limitations of SAT solvers in defeating obfuscation schemes," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 7 2018, inproceedings. [Online]. Available: http://dx.doi.org/10.1109/iolts.2018.8474189

[52] J. Wetzels and W. Bokslag, "Simple simon: FPGA implementations of the simon 64/128 block cipher," *arXiv*, Jul 2015. [Online]. Available: http://arxiv.org/abs/1507.06368v1

[53] F. Brgles and H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in fortran," in *IEEE International Symposium on Circuits and Systems*, 06 1985, inproceedings.

[54] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the iscas-85 benchmarks: a case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, pp. 72–80, 1999. [Online]. Available: http://dx.doi.org/10.1109/54.785838

[55] M. Jenihhn, 2007. [Online]. Available: http://www.pld.ttu.ee/~maksim/downloads.html

[56] A. Aysu, E. Gulcan, and P. Schaumont, "SIMON says: Break area records of block ciphers on FPGAs," *IEEE Embedded Systems Letters*, vol. 6, pp. 37–40, 6 2014. [Online]. Available: http://dx.doi.org/10.1109/les.2014.2314961