3-10-2010

Static and Dynamic Component Obfuscation on Reconfigurable Devices

Camdon R. Cady

STATIC AND DYNAMIC COMPONENT OBFUSCATION ON RECONFIGURABLE DEVICES

THESIS

Camdon R. Cady, Second Lieutenant, USAF

AFIT/GE/ENG/10-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GE/ENG/10-06

STATIC AND DYNAMIC COMPONENT OBFUSCATION ON RECONFIGURABLE DEVICES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Camdon R. Cady, B.S.E.E.

Second Lieutenant, USAF

March 2010

AFIT/GE/ENG/10-06

STATIC AND DYNAMIC COMPONENT OBFUSCATION ON RECONFIGURABLE DEVICES

Camdon R. Cady, B.S.E.E.
Second Lieutenant, USAF

Approved:

//signed//	22 Mar 2010
Dr. Yong Kim Chairman	Date
//signed//	22 Mar 2010
Dr. Michael Grimaila, PhD Member	Date
//signed//	22 Mar 2010
Maj LaVern Starman, PhD Member	Date

Abstract

Computing systems are used in virtually every aspect of our lives. Technology such as smart phones and electronically controlled subsystems in cars is becoming so commonly used that it is virtually ubiquitous. Occasionally, this technology can be exploited to perform functions that it was never intended to perform, or fail to provide information that it is supposed to protect. This can allow for the malicious use of hardware, such as circumventing copyright restrictions or stealing cryptographic keys that must be kept secret.

In order to protect these computing systems and secure the data held within, the individual components must be protected. In order to accomplish this goal, this research performs work in three areas: investigation of circuit vulnerabilities, effectiveness of static protection methods, and the effectiveness and feasibility of using dynamic protection. Circuit vulnerabilities are explored by extending X-Hot Input Analysis (X-HIA), a recently proposed blackbox attack method that reduces the number of input vectors necessary to identify a circuit, allowing for faster identification. Several previously demonstrated static obfuscation techniques are evaluated against X-HIA. Dynamic Polymorphic Reconfiguration (DPR), a previously proposed dynamic protection method that has not been previously implemented, is realized so that it can be evaluated.

X-HIA was shown to be effective at identifying several circuit components, including a multiplexer and multiplier, in a significantly shorter time than previous identification methods. Instead of requiring a number of input/output pairings that grows factorially or exponentially as the circuit size grows, it requires only a number that grows polynomially with the size of the circuit. This allows for the identification

of significantly larger circuits. Static protection techniques that are applied to the circuits under test increase the order of that polynomial, but do not increase the amount of time required to identify the circuit to the point that it is not feasible to perform that identification. DPR is implemented, and it is shown both that the overhead is not prohibitive and that it is effective at causing an identification algorithm to fail. This formalizes a method of protecting circuits from attack: altering a circuit often enough that the most efficient algorithms are never able to identify it.

Acknowledgements

Of course, I must first thank my advisor, Dr. Kim, for all of his inspiration and help. Without him I never would have been able to complete this thesis. The same goes for the rest of my committee. If Jenn, Dan, Adam, and Nate weren't around, the long days at school would have been a lot longer. Finally, I owe a debt of gratitude to my family; without them I would never have been in a position to apply to AFIT.

Camdon R. Cady

Table of Contents

Page

Page

List of Figures

List of Tables

STATIC AND DYNAMIC COMPONENT OBFUSCATION

ON

RECONFIGURABLE DEVICES

I. Introduction

Computing technology pervades nearly every aspect of our lives. Many people carry internet-connected smart phones with them in order to always maintain their link to their digital lives. The entertainment industry relies on MP3 players, Blu-Ray players, and computerized gaming consoles to deliver their content to consumers. The shift to digital media has undoubtedly increased access to many resources for many people, but can also be used to circumvent copyright laws or company policies. Once hackers discovered how the PlayStation 2 (PS2) video-game console functioned, they were able to produce a "mod-chip" for it that allowed them to use the system to play illicit games and software. As Figure 1 shows, the original circuitry was not replaced[2]. The mod chip attaches to existing circuitry and alters its function, bypassing the copyright check. Wireless access points can be modified to allow the broadcasting power to be increased further than the manufacturer recommends. In short, computing technology is here to stay, and if it not secured, someone will exploit it for their own good.

1.1 Motivation

Technology plays an increasingly important role in the United States Military. The military considers any technology that makes a significant contribution to military potential to be a Military Critical Technology (MCT)[1]. Compromise of this

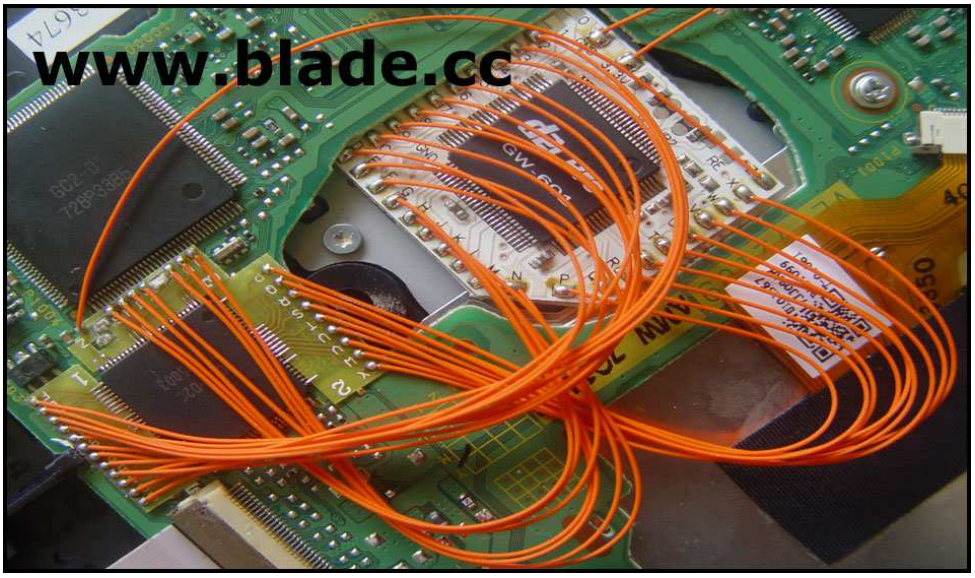


Figure 1. PlayStation 2 Mod Chip Installation

technology can have disastrous consequences for the end user, such as the enemy being able to tap into our intelligence feeds[4]. It is tempting to assume that hardware implementations are invulnerable to attack because they do not feature large easily observable components like mechanical systems do. This is not the case. The PS2 was likely cracked using brute-force methods. An amazing amount of information about a system can be obtained by a determined person in their garage, without fancy equipment. An adversary must only compromise the weakest link in a system in order to gain access to critical technology (they did not have to break the copy protection on a DVD, just the DVD player). For this reason, the hardware itself must be secured against attack.

Attacks come on two varieties: black-box and white-box attacks. While-box attacks occur when the attacker is able to gain access to information about the inside workings of a circuit and use this information to analyze the circuit whereas during black-box attack, an attacker only has information about the input-output behavior of the circuit. Most white box techniques involve partitioning a circuit into likely components, and then performing black-box identification on those components. For

that reason, this research focuses on black-box protection.

The attack against the PS2 was likely a black-box attack. By loading first a legitimate game, then a illegitimate copy of the same game, and monitoring the signals between chips, the attackers could deduce which signals prevented the copied game from being used. Then, a chip was designed that could override these signals and allow any game to be played. The same modification chip works for several generations of the PS2, demonstrating that altering the layout of a circuit board does not provide security if the function of the signals is known.

In the interest of shortening development time and creating more versatile, cost effective systems, reconfigurable devices such as Field Programmable Gate Arrays (FPGAs) are frequently used. The same qualities that make these devices highly desirable can also leave them vulnerable to attack. If the computer controlling the engine in a car can be easily reprogrammed, the dealer can fix design problems simply and quickly. This same functionality may allow for illicit modifications circumventing environmental or safety laws.

1.2 Problem Statement

Hardware protection systems all seek to secure an electronic device that may be out of our physical control. Obviously, the consumer of the technology will be able to examine it at their leisure. Even if the consumer is trusted (perhaps this would be the case with a radio that is only sold to the military), the potential exists for the device to be lost or stolen, and subsequently fall into the hands of a less trustworthy user. The goal of obfuscation is twofold: to make it difficult or impossible for an adversary to discover the function of a captured circuit and to ensure that an adversary cannot easily replicate or modify the captured system.

Current methods of protecting circuitry seek to protect the circuit at either the

design stage or the configuration stage of the system. Because this protection only occurs at a single point in time, and does not change over time, these methods are referred to as static protection. Because a combinational component can be identified by enumerating its truth table, an ideal static transformation will require anyone seeking to identify the circuit to try every possible input combination.

If static protection methods are not adequate, or the cost is too high, then dynamic methods must be employed. A novel protection concept called Dynamic Polymorphic Reconfiguration (DPR) that could successfully protect a circuit against black-box identification methods has been presented but there has been no implementation nor evaluation performed[14]. Obstacles that must be overcome in order to implement DPR include defining the transformation frequency as well as formalizing the structure of the dynamic system, as well as the actual implementation in either an ASIC or some sort of reconfigurable device. The goal of the dynamic method is to increase the number of input vectors that must be applied to identify a component beyond the best static transformation, so that even if all possible inputs are applied to a circuit, it may still not be identifiable without additional information.

1.3 Contribution

In order to successfully protect a device from unwanted analysis, common devices are broken down into building blocks, and efficient analysis methods for each building block are created. The usefulness of static methods against these methods and also more conventional ones is examined in order to define the requirements of a dynamic system. Once the requirements for a dynamic system have been established through analyzing static protection schemes, a framework for DPR (or another dynamic protection method) to be implemented and tested must be created. This framework may be specific to the technology being used to implement the dynamic system. Due to

their speed, flexibility, and widespread use, FPGAs will be used as the platform for implementing and testing the dynamic protection scheme.

II. Literature Review

The typical design process for an electronic system involves first specifying the operation of the system, and then using available technology to meet the requirements. While this can be an expensive and time consuming process, it is well understood and practiced regularly.

Conducting this process backwards, or reverse engineering (RE), can be far more complex. The goal of RE is to discover and describe the operation of a given system. Theoretically, this can always be accomplished through an exhaustive search, however, modern systems are sufficiently large that the world's fastest supercomputer could not exhaustively test a circuit before the sun burned out.

This chapter is structured as follows: first, the strengths and weaknesses of reconfigurable computing in general, and FPGAs in particular, are examined in order to justify their use and the effort of protecting them. Different types of attack methods are then explored to define what the system being protected may encounter. The specific methods of attack that this research hopes to defeat, component identification, is addressed in more detail. Existing methods of component hiding are presented, and finally the systems on which these techniques will be tested are described.

2.1 Reconfigurable Computing

A reconfigurable computing system is one that is capable of having its operation defined after it is manufactured. Changing the operation of a device without remanufacturing allows for rapid prototyping of digital systems, presenting significant decreases in the amount of time and money required to iterate through the design process. As a result, reconfigurable computing platforms are in widespread use. The Field Programmable Gate Array (FPGA) is a commonly used, powerful, and versatile

reconfigurable computing platform.

2.1.1 FPGAs.

An FPGA is an array of reprogrammable gates. The operation performed by each gate is specified prior to programming the device. FPGAs represent a modern improvement on the Programmable Logic Devices (PLDS) and Complex Programmable Logic Devices (CPLDs) that were created in the 70s and 80s. Modern FPGAs may contain millions of circuit elements, as well as on-chip application-specific integrated circuits (ASICs) such as arithmetic units or microprocessors to enhance the function of the device.

Each manufacturer of FPGAs has their own design and nomenclature, but the basic operation of each is similar. The design of the Xilinx Virtex Series FPGAs is presented here, but the discussion applies to any FPGA based on lookup table (LUT) and static random-access memory (SRAM) technology.

The FPGA is constructed of a 2-dimensional array of logic blocks connected by a routing matrix. Both the logic blocks and the routing matrix are reconfigurable. Xilinx refers to each of these logic blocks as a Configurable Logic Block, or CLB. The CLB in turn contains smaller logic elements, termed Slices. Slices within a CLB can communicate directly with other Slices in the same CLB. Within each slice are multiple LUTs. These LUTs act as truth tables, translating multiple inputs to multiple outputs. Each LUT of the Virtex-5 has 6 inputs and 1 output. The LUTs are paired with flip-flops (FFs) to facilitate the creation of sequential circuit elements.

By specifying the operation to be performed by each LUT as well as the interconnections between and within CLBs, virtually any operation can be implemented on an FPGA. The operation is specified using a netlist, register transfer logic (RTL) description, hardware description language (HDL), or circuit description. Proprietary

tools then transform this specification into an FPGA configuration. The output of these tools can be used to program the FPGA and is referred to as a *bitstream*. During its lifetime, an FPGA may be reprogrammed hundreds or even thousands of times.

2.1.2 Run-time and Partial Reconfiguration.

Typically an FPGA is programmed by bringing it offline and transferring the bitstream to the FPGA. One area of considerable interest is *partial reconfiguration*, in which only a portion of the bitstream is modified. This allows the unaffected portions of the FPGA to continue to function while the programming operation occurs, leading to increases in performance[7].

This increased flexibility and performance requires a more complex architecture and tools. Several vendors including Xilinx do offer devices capable of partial reconfiguration, but there are limitations to these capabilities. Typically, a differential bitstream is generated, which is not desirable when producing dynamically reconfigurable circuitry[19]. Current Xilinx tools do not support the exchange of predefined modules, although this capability was present in previous tools[18]. The module support that did exist required each module to be stored on the board when the device is programmed, and new modules could not be introduced without bringing the FPGA offline. Stone introduced a design where individual LUTs could be reconfigured while the FPGA was running[22].

Stone's research is promising but has its own limitations. First, it contains only 16 CLBs, whereas most modern FPGAs contain thousands. Also, while the function of a CLB or LUT may be dynamically altered, the routing cannot be changed. One encouraging result is that the reconfiguration circuit can operate at a higher frequency than the rest of the FPGA, minimizing circuit downtime.

2.2 Circuit Vulnerabilities

Military Critical Technologies (MCTs) and proprietary hardware must be secured. Several classes of attacks are possible against both ASICs and FPGAs. Four of these classes are discussed here: side channel, reverse engineering, invasive techniques, and semi-invasive techniques. Side channel attacks attempt to indirectly gain access to proprietary information about a system. Reverse engineering attempts to create a higher level description of a system in order to better understand it, and therefore exploit it. Invasive and semi-invasive techniques are the most direct attacks, in that they physically attack the circuit in order to exploit it.

2.2.1 Side Channel Attacks.

While an operation is being performed by a digital system, measurements such as time, power consumption, heat signature, and electromagnetic emissions can be recorded. Side-channel analysis uses these measurements to make inferences about the operation of the circuit, which can then be used to exploit the circuit. Information that would not normally be available, such as secret keys, can be obtained using side-channel analysis.

Simple Power Analysis (SPA)[23] and Differential Power Analysis (DPA)[11] are commonly employed against cryptographic circuitry. In order to perform this analysis, the power consumption of a circuit is monitored by recording the voltage drop across a low-valued resistor inserted between the circuit and the power supply. In some cases, attacks can be carried out with little knowledge of the circuit itself[13]. Typically the differences in power consumed by different functional units that carry out arithmetic operations are quantified, and information about the key can be obtained. SPA discerns the key directly from the power traces while DPA uses statistical methods to test likely keys.

It has been shown that an FPGA is susceptible to correlation power analysis (CPA)[21]. A block cipher was implemented on a Xilinx FPGA, and the key was subsequently extracted using statistical analysis of the power consumed by the device.

Sometimes the key can also be extracted by analyzing the amount of time required to complete an operation. In many RSA implementations a choice is made between a squaring operation alone or a square followed by a multiply. Obviously, the square then multiply choice will take longer to execute. Timing analysis was first completed in 1996[12]. The key can be obtained from a 512-bit RSA encryption unit by recording and analyzing the time it takes to complete 5000 encryption operations[15].

2.2.2 Reverse Engineering.

One goal of the reverse engineering process is to identify the components within a system and the relationships between these components[6]. Given a gate level description of a circuit, it is possible to select a subset of gates that may be a distinct component[24].

The goal of an identification algorithm is to determine whether or not a circuit is equivalent to any circuit contained in a library of circuits. Two circuits are equivalent if and only if there exists some ordering of the inputs and outputs such that the truth tables for both circuits are identical[24].

For small circuits, the truth table of a candidate component and a library component may be compared directly. For larger circuits, it is not possible to even enumerate the truth table[14].

It is worth noting that if any two circuits are structurally equivalent, they will also be functionally equivalent. The reverse does not necessarily hold true. Consider as an example two adders of the same size, one a conventional ripple-carry adder and the other a carry-lookahead adder. The two are functionally equivalent, because for

each possible input, they produce the same output (the sum of the inputs). The two are not structurally equivalent, because they are different at the gate or transistor level.

Identification techniques that analyze only the circuit structure to identify a circuit are known as white-box techniques, while techniques that use only the input-output relationships of the circuit to identify it are known as black-box techniques.

2.2.3 Invasive Techniques.

While the previously discussed exploitation techniques passively monitor the circuit to be exploited, this is not always the case. If access to a device is unrestricted, the device may be physically tampered with. Invasive techniques require the de-packaging of components, removing the protective packaging around the circuit. These techniques have not been demonstrated against FPGAs, but the SRAM technology that FPGAs are built on is susceptible to these attacks[25]. In order to conduct these attacks, specialized tools are required.

2.2.4 Semi-invasive Techniques.

These techniques also require that the outer packaging of a circuit be removed, but do not require the same level of technical equipment and expertise that invasive techniques do. In most cases, semi-invasive techniques seek to alter the behavior of the circuit, which may allow for the breaking of a cryptographic system. The state of individual transistors or SRAM cells can be altered using radiation[20].

2.3 Component Identification Techniques

Three methods of component identification are presented in this section: a structural approach that has been used to identify the components in a set of benchmark

circuits, an approach that does not rely on the structure of the circuit to analyze components, and X-HIA, the component identification method that will be expanded upon in the next chapter.

2.3.1 A Structural Approach.

The ISCAS-85 benchmark circuits have been reverse engineered using a structural approach[10]. Because the approach was both successful and well-documented, it makes an ideal case study for structural methods of component identification.

The approach used by Hansen, Halcin, and Hayes utilized eight techniques to describe a circuit at increasingly higher levels.

- The reverse engineers began by looking for library components that were easily recognizable. Examples of this are XOR gates constructed for basic gates, multiplexers, and CLA generators.

- Secondly, they identified non-library logic modules that were repeated throughout the circuit. These elements give insight as to whether or not the circuit performs the same operation on multiple bits.

- After recognizing some of the modules in the circuit, the engineers looked for expected circuit elements, such as an XOR gate accompanying CLA generator logic.

- Small unidentified sections of the circuit were exhaustively analyzed. This is only feasible for sections of the circuit with a small number of inputs.

- High fan-out signals were identified, because they are likely to be control signals. This allows the blocks that the control signals feed to be identified more easily.

- By grouping signals into busses, and then examining where these signals lead, the circuit can be partitioned further.
- In the case of the ISCAS benchmarks, common names were employed for related signals. This allowed the reverse engineers to group these signals together.
- If a block contains truly random logic, it must be represented as a black box.

The authors concluded that it is possible to reverse engineer a circuit using white-box techniques without simply comparing the entire circuit to library circuits. They successfully reverse-engineered circuits with over 200 input lines, and circuits with more than 2000 logic gates. The technique they present relies on the existence of engineers with expertise in the field of circuit design, and plenty of time to work. No completely analytic method is presented.

2.3.2 Semantic Approaches.

An infinite number of combinational circuits can be generated that match a given function or truth table. The proof of this is simple: assume that there were only a finite number of circuits that could represent a given function. After these were enumerated, add a buffer to the outputs of each of them. This will create at least one new combinational circuit that matches the function, violating our assumption. Therefore, there must be an infinite number of circuits to represent any function.

Given the theorem above, no syntactic matching library could ever contain every possible circuit that could implement a function. Therefore, in order to increase the chances of matching a circuit component to a component in the library, semantic methods can be used. Because semantic methods focus only on circuit behavior, the underlying implementation details do not affect the running of the algorithm.

2.3.2.1 Doom's Method.

Doom, White, Wojcik, and Chishold proposed a different method of component identification. Their method relies not on finding exact structural matches, but on finding functionally equivalent circuits. This process begins by narrowing the number of possible candidate circuits by computing a signature for the component(function) to be identified. This signature is not necessarily unique to the component being identified, and all functionally equivalent components will share a signature. This narrows the search space of equivalent components to those components sharing a signature with the target component.

An algorithm is presented which allows for the automation of component identification. The algorithm proceeds through the following five steps:

1. Create Binary Decision Diagrams (BDD) for both the component to be identified and a candidate component.

2. Compare the signature of each component. If they do not match, discard the candidate component as a possible match.

3. Place all of the possible matches into the suspect set.

4. Search each possible correspondence for a match, using the BDD generated in the first step.

5. Determine the legal output correspondences, which are the orderings of the outputs for which the components are equivalent.

This method successfully identified common circuits. It was much faster than a "brute force" method. The brute force method grows factorially as the number of inputs increases, while this method grows exponentially (a significant improvement.)

2.3.2.2 X-HIA.

X-hot input analysis represents one of the newest methods of identifying components[14]. The name is derived from the principal operation performed while identifying components: recording the outputs of the circuit while driving all permutations of X inputs high (the "hot inputs"). While the number of permutations grows quickly as X increases, Porter showed that X can remain small, thereby allowing for very quick component identification compared to testing all input and output permutations in order to match to a library component. X-HIA requires the formulation of a unique identification strategy for each component in the library. It has not been shown that an efficient strategy can be developed for a wide variety of components, but the results are promising. For example, an adder with n inputs can be identified using only about $1.5n$ input vectors, allowing a 64-bit adder (129 inputs) to be identified in approximately 200ns using modern testing equipment.

2.4 Component Hiding Techniques

There are several ways to classify circuit obfuscation techniques. Techniques may be classified according to whether they defend against white-box attacks, black-box attacks, or both. Alternately, they may be divided by whether they preserve the original circuit semantics(the truth table) or alter it. Obfuscation methods can also be grouped by whether they are performed once, when the circuit it created (*static methods*) or occur as the circuit is being used (*dynamic methods*).

2.4.1 White-box methods.

White box methods necessarily involve changing gates or signals inside the circuit. Examples of transformations that could be used are increasing the number of input bits, introducing intermediate gates to create new paths within the circuit, or

introducing new output gates. These transformations may be carried out statically or dynamically. Both techniques that preserve the semantic meaning of a circuit and those that do not may defend against white-box attacks.

Typically, there are several stages to any white-box obfuscation method. First, a gate or signal must be selected for obfuscation. Next, the obfuscation itself must be selected (i.e. a replacement gate type). Finally, the circuit must be altered, and the necessary information computed and stored in order to recover the original function of the component.

2.4.2 Black-box methods.

Because black-box analysis methods rely only upon the truth table of a circuit in order to identify it, techniques that preserve the semantics of a circuit are not effective against black-box attacks. White box transformations that do not preserve the original semantics (function) of the circuit may protect against black-box attacks, as well as methods that simply alter the truth table without relying on white-box circuit information.

2.4.3 Dynamic methods.

It is possible to create an FPGA that can be reprogrammed without halting the operation of a circuit[22]. Alternately, a circuit may be fabricated with gates that perform multiple functions based on the operating conditions[16]. These advances in technology allow for the formulation of dynamic obfuscation techniques in which the circuit is periodically modified in order to make it more difficult to identify.

One method that has been shown to help protect a circuit from unwanted analysis is Dynamic Polymorphic Reconfiguration (DPR)[14]. This technique relies on six functions to obfuscate the circuit:

- GateReplace - performs a gate replacement
- SignalHide - hides a signal
- AddInput - adds an input
- AddGate - adds a gate
- AddOutput - adds an output
- StandAloneKey - creates a standalone recovery key

2.5 Cryptographic Systems

Encryption and decryption of information is absolutely essential to e-commerce, the military, and even personal computer users. Many online transactions require the use of cryptography. Cryptography may also be used to digitally sign documents or perform identify verification. These cryptographic systems all require the use of a key, so methods of generating the key will first be explored, AES and RSA, two of the most commonly used cryptographic algorithms, are presented.

2.5.1 Key Generation and Protection.

All modern cryptographic methods rely on the use of a secure key. If this key is compromised, the security of the entire system is compromised, regardless of the particular algorithm used[17]. If the system can by analyzed sufficiently, the key may be able to be read out of a cryptosystem directly. If this were to occur, the security of the system would be negated.

Key generation techniques fall into two categories: random and pseudo-random. Random bitstreams are difficult to produce, but cannot be accurately predicted by an adversary. Pseudo-random bitstreams are easy to produce using a variety of methods,

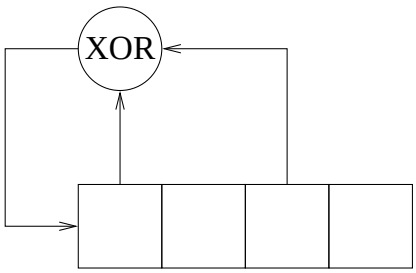


Figure 2. 4-bit LFSR

but can be predicted if some information is known. Many pseudo-random number/bit-stream generators rely on some sort of seed value. One commonly used pseudo-random number generator is a linear feedback shift register (LFSR). A LFSR is a shift register with the input connected to some combination of the current bits in the register. Typically, some of the bits in the register are XORed together to form the next bit to be shifted in. An example of a LFSR is shown in Figure 2.

A LFSR can be described by an associated polynomial. The coefficients of the polynomial are either 0 or 1, and indicate which bits of the shift register are to be "tapped" in order to form the input bit. The polynomial for the shift register shown in Figure 2 is shown in Equation 1.

$$x^3 + x^1 + 1 \tag{1}$$

2.5.2 AES.

The advanced encryption standard (AES) is currently used by the US Government to protect sensitive and classified information. AES operates on 128 bits of ciphertext or plaintext at a time and uses either a 128-, 192-, or 256-bit key to perform the encryption or decryption[3]. AES consists of five functions, each of which is repeated depending on the size of the key used. If the circuit components that implement these functions can be identified, the key can be recovered. The five functions used in AES

are:

- KeyExpansion - This function is only performed once. It is the first function to be performed when an encryption or decryption operation begins. It takes as an input the 128-, 192-, or 256-bit AES key and generates 10, 12, or 14 round keys depending on the key length.

- AddRoundKey - This function takes as an input 2 128-bit operands. It performs a bitwise XOR operation and returns the result.

- SubBytes - This function takes as an input a 4×4 array of bytes. Each byte is replaced by another according to a non-linear lookup table referred to as the substitution box, or S-box.

- ShiftRows - This function takes as an input a 4×4 array of bytes. It shifts each byte in the second row to the left by one position, each byte in the third row left by two positions, and each byte in the fourth row left by three positions.

- MixColumns - This function takes as an input a 4×4 array of bytes. Each column of the array is multiplied by a fixed polynomial.

Each of these functions can easily be implemented in hardware, or using an 8-bit microprocessor. If implemented in hardware, each component can be attacked separately. Once the signals that correspond to the key are identified, the security of the system is compromised. If the round keys are even partially discovered, the original key can be recovered[9].

Encryption and decryption are performed using the same hardware on AES, using the same key. Because the same key encrypts and decrypts a message, it is known as a symmetric key algorithm. The re-use of the same hardware implies that if encryption can be protected from exploitation, so can decryption.

2.5.3 RSA.

RSA is a public-key cryptographic algorithm. As such, two keys are generated, one for encryption and one for decryption. Only one of these must be kept private, the other is made freely available.

The basis for any RSA implementation is modular exponentiation. This is a fairly simple mathematical operation. The repetition required to implement it makes some RSA hardware very vulnerable to attacks, particularly side-channel attacks.

2.6 Summary

Reconfigurable hardware simplifies the design process by allowing designers to rapidly iterate through prototype designs. Unfortunately, they are vulnerable to many of the same attack methods that conventional ASIC circuitry are vulnerable to, as well as some vulnerabilities introduced by the design process for reconfigurable hardware. Two of the more direct attacks are black-box and white-box component analysis. Several examples of these attacks were presented, as well as the existing methods of countering the attacks. Finally, the crypographic systems that can be implemented in hardware and may need to remain secure are discussed.

III. Static Circuit Vulnerabilities

This chapter describes a component identification technique that can be performed with only black-box information. The effectiveness of various static obfuscation methods in defending against this identification technique is then explored. In the next chapter, a dynamic protection technique will be developed using some of the same circuit transformations in order to further protect a component from identification.

The desired output of an identification routine varies depending on the application. If a supplier goes out of business, or the original specification of a system is lost, then all that is necessary is to be able to replicate the original hardware. For this purpose, simply knowing the truth table or circuit netlist is sufficient, because tools exist that can implement a circuit given only the truth table or netlist. On the other hand, if the goal is to understand the working of the circuit (perhaps to locate the key, or check to ensure that no additional hardware was added by an untrustworthy manufacturer) then more knowledge is required. Simply knowing the truth table does not suffice; knowing that the truth table of the component matches the truth table of a multiplier is much more useful.

To define a circuit by applying all possible inputs and recording the outputs (enumerating the truth table line by line) is not feasible. For a circuit with 60 binary input lines, this discovery process would take approximately thirty-six thousand years using a gigahertz tester. Circuits or programs are often composed of common building blocks. This can stem from design processes used by engineers, who are used to working with these components, or from the automated translation from a high-level language to a circuit description. As a result, the search space for an unidentified circuit can be significantly smaller than 2^n, where n is the number of inputs. An algorithm to identify these common components quickly is presented here.

Several methods have been proposed for protecting circuits. These circuit transfor-

mations are performed once, at design time. They seek to change the implementation of the circuit from one that can be easily identified to one that is difficult to identify. These transformations are described and their effectiveness against the matching techniques presented is evaluated.

3.1 A general identification method

Circuitry can be broken down into combinational and sequential elements. In a combinational circuit, the output depends only on the current inputs. In a sequential circuit, the output may depend on the past inputs in additional to the current ones. As a result, sequential circuits are much more difficult to analyze than combinational circuits. Sequential circuits may be broken down into flip-flops with combinational circuitry in between. If the combinational portion can be protected, then the sequential circuit is protected.

Many circuit analysis techniques approach the identification problem rather blindly. Significant increases in performance have been achieved by classifying inputs and outputs into narrow containers prior to the matching algorithm being run[8]. This narrows the search space considerably.

This identification method expands on these techniques by taking into account the likely function of a circuit. Once a likely function is identified, the following steps are tailored to identifying the inputs and outputs of that particular function. If the identification fails, the identifier moves on to the next most likely function, and runs a sequence tailored to that function. If all possible functions are tested and fail, then the algorithm has failed to identify the component.

By following a systematic, easily automated process, the components can be identified using far fewer steps than conventional brute force attacks. This process can be divided into four steps: input/output identification, input/output space analysis,

input/output partitioning, and bit ordering and correspondence discovery. This technique builds upon that presented by Porter in [14] and described briefly in Chapter II.

3.1.1 Input/Output Identification.

As all black-box analysis techniques rely upon the ability to force an input and measure an output, identification of each pin as either an input or an output is a necessary first step. This is easily accomplished using modern technology and techniques.

The ideal electronic device has the following characteristics: infinite input resistance and zero output resistance. While in practice no device can be fabricated that meets this standard, the input resistance is several orders of magnitude higher than the output resistance.

It is possible for Automatic Test Equipment(ATE) to quickly measure the response of a pin to small currents (100 - 250 μA) without damaging the pins. This measurement allows for the identification of a pin as either an input or an output[5].

When using reconfigurable hardware such as an FPGA, this process may be simplified. If any of the original design information is available, it is likely that the I/O configuration will be stored and can simply be read from the file. If not, the same techniques that are used to analyze ASIC circuitry can be utilized.

3.1.2 Input/Output Space Analysis.

Analysis of the I/O space allows for a probabilistic classification of a circuit, that is, it does not absolutely determine the function of a component, but it does lend itself to determining the likely function of the circuit. This in turn allows the search routine to test the most likely functions first, greatly improving the efficiency of the

Table 1. Expected Input/Output Ratios

Device Name	Input and Output Information		
	Input Pins	**Output Pins**	**Ratio**
n-input gate	n	1	$\frac{n}{1} = n$
m-bit adder	$2m + 1$	$m + 1$	$\frac{2m+1}{m+1} \approx 2$
$k : 1$ m-bit MUX	$km + \lceil log_2 k \rceil$	m	$\frac{km+\lceil log_2 k \rceil}{m} \approx k$
m-bit multiplier	$2m$	$2m$	$\frac{2m}{2m} = 1$
decoder	n	2^n	$\frac{n}{2^n} < 1$

algorithm. Both the number and ratio of input and output pins yields information about the likely function of a circuit.

Comparison of the input/output space of a circuit to Table 1 shrinks the number of likely circuit candidates. However, in many cases there will still be multiple candidate circuits. For example, It is impossible to distinguish between the 1-bit 2:1 multiplexer and a 3-input logic gate by looking only at the number of inputs and outputs. For this reason, an *IO Signature* is created for each component in the library. IO Signatures of different components can be compared. Candidate functions can then be ordered from most likely to least likely, and tested in that order.

3.1.3 Input/Output Partitioning.

Input/Output Partitioning organizes the inputs and outputs such that related bits are grouped together. For example, the inputs of a division unit could be grouped into a divisor and a dividend.

Symmetric Groups are groups whose labels may be exchanged without affecting the function of the circuit. Examples of circuits that contain symmetric groups are commutative mathematical functions, such as addition and multiplication. Symmetric groups will be labeled Group A, Group B, etc during the labeling process. *Asymmetric Groups* are groups that are not symmetric, such as a group of control

lines on a multiplexer, or the operands of a divider. Asymmetric groups will be given distinct labels corresponding to their function within the circuit.

3.1.4 Bit Ordering and Correspondences.

For some circuits, the ordering (significance) of bits within a group can be determined (as in an adder or multiplier). In others, such as a multiplexer, no ordering can be made within a group.

If the order of the lines within a group can be established, then the lines of Group A (containing m lines) will be referred to with increasing order of significance as $a_0, a_1, ..., a_{m-1}$. If the absolute ordering cannot be established, but it can be established that bits in different groups share significant positions, then these bits will be said to *correspond* with each other.

In some cases, particularly with control lines, the order cannot be established, but the effect of particular logic values upon the operation of a circuit can be established. As an example, the ordering of the control lines of a 4:1 multiplexer cannot be established, but the combination $(0, 0)$ may lead to group A being selected, $(1, 0)$ lead to group B being selected, and so forth. In this case, a table or other expedient method of presenting this information will be given.

3.1.5 Process Flow.

Figure 3 shows the overall process flow for identifying a component. First, a proper subcircuit must be selected. This subcircuit corresponds to a likely library component. Depending on how the subcircuit was identified, it may be necessary to examine it to discover the function of each pin: either power, ground, input, or output. This step is only necessary when working with ASIC components. Then, the inputs and outputs are partitioned and bit order/correspondences are verified as

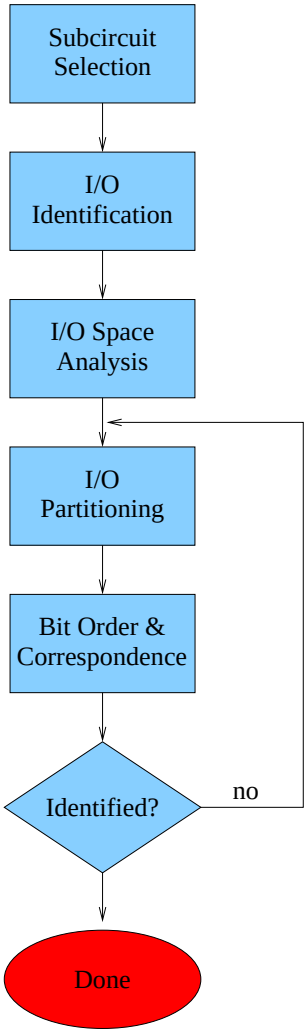


Figure 3. Process Flowchart

described above. These steps are repeated until the component is identified, or no more information can be gathered.

3.2 Static Protection Analysis

Static transformations can be grouped into three categories: those that alter circuit semantics but not the number of input or output bits, those that add input and/or output lines without altering the semantics of the original inputs and outputs, and those that alter both the input/output count and circuit semantics. Obfuscation tech-

niques may also be categorized depending on what level of circuit description they act on. Techniques that only rely only upon the truth table for the circuit description are black box techniques, while those that require information about circuit structure are white box techniques.

Semantic preserving techniques allow the circuit to still perform the original function (such as encryption) with no extra recovery circuitry. If the circuit semantics are not preserved, then additional circuitry must be added in order to recover the original function.

3.2.1 Semantic-Preserving Techniques.

In order to obfuscate a circuit while preserving the original semantics, additional inputs and/or outputs must be added to the circuit.

3.2.1.1 Addition of input bits.

Several input bits can be added to the circuit such that when the correct combination is applied, the function of the original circuit is unchanged. Every other combination will result in a different function being performed. In effect, there is a combinational lock on the circuit, and the circuit will not work properly unless the correct combination is presented.

Consider a component with n inputs and m outputs. If the component is one of the components analyzed above, the number of input vectors required to analyze the component is a polynomial function of n. Extra inputs can be added to a circuit to form a sort of combinational lock. When the correct combination is applied to these inputs, the circuit will function in its original manner. When an incorrect combination is applied, the outputs of the circuit will be garbled. When a combinational lock of size k bits is added to the component, the analysis tool must first identify which input

bits make up the combinational lock. If n, k, and the correct key are known, then the identification routine must be run $(n+k)(n+k-1)...(n+1) > n^k$ times in order to test every possible permutation of input bits of size k. On average, half of the possible permutations will have to be tested before the key bits are located. Even with all information given to the component identification algorithm other than which inputs form the key, the asymptotic complexity of the identification algorithm is increased by a factor of n^k. This is a significant increase, considering that the asymptotic complexity of identifying an adder is $O(n)$ and that of identifying a multiplier is $O(n^2)$ (see Chapter IV).

While the addition of the combinational lock adds security while adding relatively few inputs to the circuit, care must be taken with the implementation of such a system. In order to meet the assumptions listed above, the circuit must behave differently for each combination applied to the lock bits. This can be accomplished by replacing gates with polymorphic gates (described in detail in Chapter IV). This may cause the size of the circuit to grow significantly when the lock is added to the circuit. The analysis above assumes that only black-box information about the circuit is available. If white-box analysis is able to identify the lock bits, significantly less security is added to the system. Similarly, if the adversary can observe the operation of the circuit in a larger system and determine the proper input combination to apply to the lock, the lock adds no security.

3.2.1.2 Adding Output Bits.

The addition of output bits will not significantly affect the algorithms presented previously. The additional outputs will simply be considered unidentified, but the original circuit will still be identified.

3.2.2 Non-Semantic-Preserving Techniques.

As demonstrated in the previous section, semantic-preserving techniques can offer only a limited amount of real-world protection. Therefore, techniques that alter the semantic description of the circuit should be employed in order to increase protection.

The transformations can be made in two different ways. An existing semantic description of the circuit(i.e. a truth table) can be altered directly or a syntactic/structural description of the circuit can be altered. Each set of techniques has advantages and disadvantages.

3.2.2.1 Line inversion.

The simplest black-box transformation inverts the outputs on certain lines of the truth table. Inverting every line would result in the trivial transformation of the circuit from one with active-high outputs to active-low outputs, or vice-verse. A non-trivial transformation utilizes an *inversion schedule*. This schedule determines whether or not a line of the truth table can be inverted. In practice, this may be accomplished through a boolean function which determines whether nor not the output is inverted based on the input. This is equivalent to the replacement of a primary output with a 2-input XOR gate with one input connected to the previous output and one input connected to a combinational logic block or key that selects when that output is to be inverted. Therefore, it can be analyzed in the same manner as a gate addition obfuscation strategy, and has the capability to successfully obfuscate a circuit.

3.2.2.2 Column Exchange.

While row exchange can be applied to the truth table as a whole, column exchange must be applied to only certain rows. This is because inputs and outputs are con-

IN1	IN2	IN3	OUT1	OUT2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Original Truth Table

IN1	IN2	IN3	OUT1	OUT2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

(b) Obfuscated truth table

Figure 4. Column Exchange Operation

sidered in arbitrary order by most identification algorithms. An example of column exchange may be to change the order of bits in an operand depending on the parity of that operand, possibly confusing algorithmic attempts to order the bits within that operand. Alternately, columns could be permuted based on the hamming weight of the input. Figure 4 demonstrates the results of column exchange on a 3-input, 2-output circuit. The columns have been exchanged in the highlighted area.

Performing column exchange is equivalent to the addition of polymorphic switches (see Chapter IV) that are controlled by internal circuit signals (instead of additional pseudo-primary inputs). The algorithm for component identification presented in this chapter is defeated if non-trivial exchanges are performed. Non-trivial exchanges are those that exchange inputs or outputs that do not share logical significance. In other words, the exchange must alter the truth table. In order for the original intent to be recovered, the exchange operation must be performed again before the outputs are used.

3.3 Summary

In this chapter, a black-box component identification method was described. Several white- and black-box transformations intended to protect a component from being identified were presented, and their effectiveness against this new identification method was estimated. If the component identification method can identify many common components, static protection will not provide enough security to make component identification infeasible.

IV. Dynamic Techniques

Dynamic protection techniques can be used to enhance the protection offered by static defenses. It may be possible to provide additional protection while still consuming fewer resources if implemented properly. This chapter describes the requirements of such a dynamic component hiding solution, presents one possible method by which such a solution could be realized, and outlines a method of implementing and testing this method on an FPGA.

Regardless of the algorithms used to identify a circuit, certain bounds are not likely to be broken. When performing black-box identification, the function of each input pin must be identified. This suggests that even the world's best attacker would be forced to apply at least as many input vectors as there are input pins and record the results for analysis. Therefore, the theoretical minimum number of input vectors required to identify a n-input circuit is n vectors. This seems to hold true for even simple components, such as an adder (the required number of input vectors is only about 50 percent higher than this theoretical minimum.) For even the most complicated circuitry, there are upper bounds to the difficulty of circuit identification. Regardless of circuit function, the truth table for a combinational circuit can be enumerated in 2^n cycles, where n is the number of inputs.

The bounding of static solutions leads to a dynamic approach. The bounds on the number of cycles necessary to identify a given circuit dictate the requirements for a dynamic system. In addition, the frequency of reconfiguration is of critical importance. If reconfiguration occurs infrequently so that each configuration can be fully analyzed and understood by an adversary, then the dynamic system provides little or no additional security. If the reconfiguration occurs much more frequently than necessary, then unnecessary overhead (measured in wasted clock cycles and power consumption) is incurred in order to implement the security measure. If the

proper frequency is selected, then maximum security can be provided at the lowest possible cost.

4.1 Requirements and Constraints

4.1.1 Adaptability to a Key.

Prior to each reconfiguration, a change or changes must be made to the circuit. The method of selecting these changes is subject to the same trade-off as the change selection methods for a static implementation. If changes are too predictable, then more information is leaked from the system. If changes are too random, it is possible that no additional security will be added. For this reason, a key will be used to determine the changes made to the system. There are several reasons for this. First, as long as the key is properly protected, the changes being made are unpredictable to an adversary. Second, use of a key allows for the repetition of a set of changes, which is useful when analyzing a system that may be used. Third, if some sort of hardware signature (digital fingerprint) is available, this may be used to generate the key. This will allow for a system to work predicably on a given piece of hardware, but if a design is stolen and applied to a different piece of hardware, then it will not work as expected, further complicating the task of replicating the function of a component.

4.1.2 Recoverability.

In most cases, the original semantic intent of the circuit must be recoverable. This is analogous to the ability of an encrypted message to be decrypted. In very limited situations, such as "shredding" a component to protect it from compromise, the intent does not have to be recoverable. Due to the protection of a key (i.e. if the key is lost, a circuit is effectively shredded), this research focuses on techniques that allow for the recovery of the original circuit semantics.

4.2 Gate Replacement

Gate replacement essentially involves 2 steps: selecting a gate to be replaced and selecting a replacement gate. Each of these steps can be accomplished dynamically using a key-based system. Gates can be numbered in a predictable fashion using netlist IDs or circuit layering techniques. Assuming that only AND, OR, XOR, and their inverse gates are allowed, all gate types can be represented using 3 bits. Then, a portion of the key can be used to select the gate that will be replaced, and the next 3 bits can be used to determine the replacement gate type. If the key is partitioned into 16 bit sections, then circuits with up to 8192 gates can be obfuscated.

This is no different from random gate selection and replacement; it does not guarantee that the gate replacement makes the circuit harder to identify. This technique relies on the sheer number of replacements being made to obfuscate the circuit, instead of the quality of individual transformations.

There is a limitation of this technique: after a few replacement operations the key will be exhausted. Even a relatively large 1024 bit key will only last for 64 iterations. This can be overcome by using the key value supplied to generate a much longer key bitstream. This method is discussed in detail later in this chapter.

4.3 Input and/or Output Reordering

Because of the nature of black-box identification algorithms, changing the order if inputs or outputs once does not affect the running of the algorithm. Changing the order dynamically can dramatically affect the results. Take an adder for example. In the algorithm presented previously, the least significant bits of the input and output are the first identified. If the bits change order midway through the identification of the adder, then the process of identification must begin again. Repeatedly changing the order of input or output lines can prevent an attacker from ever gaining a full

picture of the operation of the adder.

This can be implemented using a key as well. For example, consider a crossbar switch inserted in series with the component to be obfuscated. If the switch is inserted before the component, any possible ordering of the inputs is possible. Similarly, if the switch is inserted after the component, any possible ordering of the outputs is possible. The key can serve as the input to the switch. Assuming that the component proceeding or preceding the obfuscated component also has access to the key, the lines can be re-ordered as necessary so that the circuit still functions correctly.

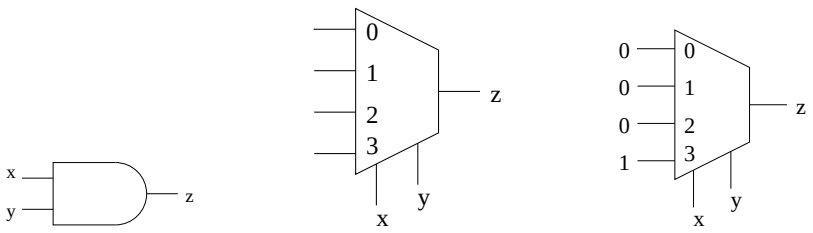
The same problems with key length arise when reordering is used. There are $n!$ possible ordering of n lines, so if every possible ordering is possible, a large key will be needed to select even one ordering.

4.4 Row Exchange and Line Inversion

As presented in the previous chapter, the row exchange and line inversion operations can be an effective obfuscation techniques. When properly applied, properties such as parity can be hidden.

With the advent of reconfigurable hardware, this operation is fairly easy to complete. Because FPGAs use lookup tables to implement the desired operations, the outputs of these LUTs can simply be exchanged or complemented. Implementation without run-time reconfigurable hardware requires the addition of many gates to the circuit, increasing the area used, power consumed, and the delay through the component.

Selection of proper rows to swap or invert is a tremendous hurdle that makes this operation difficult to implement. The number of rows in a truth table grows exponentially as the number of inputs is increased. The number of possible orderings grows factorially as the number of rows is increased. Therefore, this cannot be implemented



(a) 2-input AND gate

(b) 2-input Polymorphic Gate

(c) Polymorphic gate configured as an AND

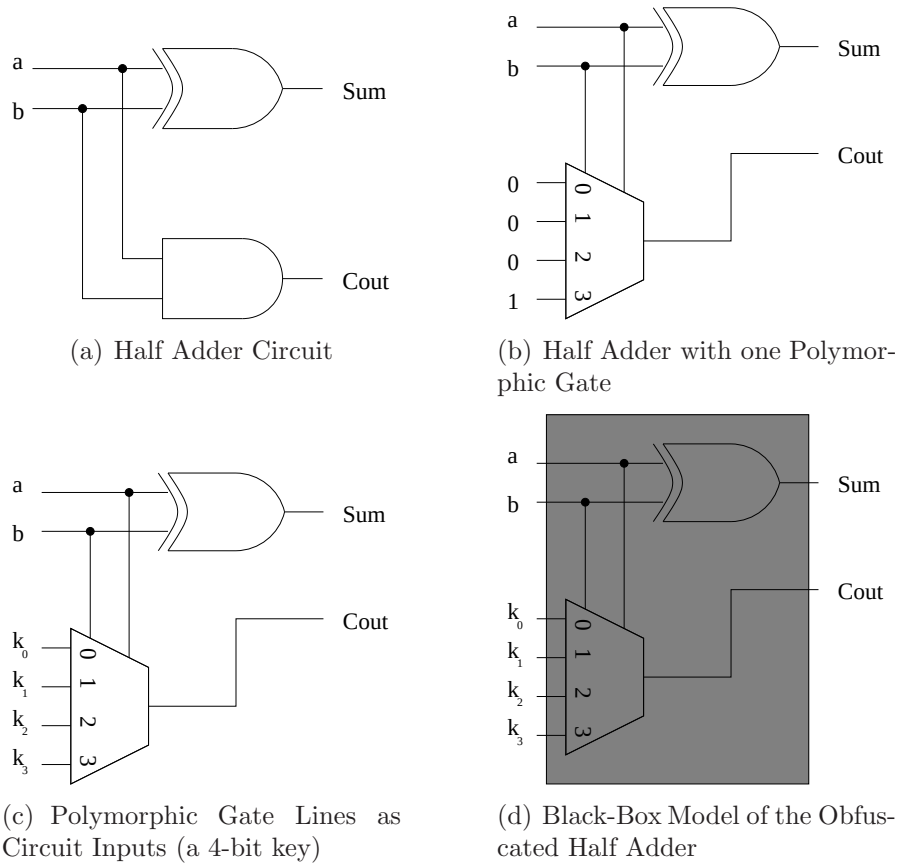
Figure 5. Polymorphic Gate Conversion

except on very small cut-sets.

4.5 Polymorphic Gates and Networks

One method of implementing a key based system is the use of polymorphic gates. A polymorphic gate can be formed by replacing any 2-input gate with a multiplexer. The gate's inputs will become the MUX select lines, while the inputs to the MUX will become the additional input lines. An example showing an AND gate, a polymorphic gate, and a polymorphic gate acting as an AND gate is shown in Figure 5. Adding a single polymorphic gate can add as many as 4 key bits to the circuit. By tying the middle two inputs of the MUX together any of the 6 basic gate types can be formed while only adding 3 inputs to the circuit. If several polymorphic gates are added, then a key of reasonable size can be added to the circuit in a very simple manner, as demonstrated in Figure 6. Implemented statically, this functions as a combinational lock. Inverters can be introduced between the input of the circuit and the MUX if desired.

Polymorphic gates are very adaptable, and require a very low overhead. Polymorphic gates can be implemented using 11 primitive gates, including inverters. In comparison, an XOR can be implemented using 4 NAND gates. The cost to convert a primitive gate to a polymorphic gate is 10 gates, while the cost of converting an XOR



(a) Half Adder Circuit

(b) Half Adder with one Polymorphic Gate

(c) Polymorphic Gate Lines as Circuit Inputs (a 4-bit key)

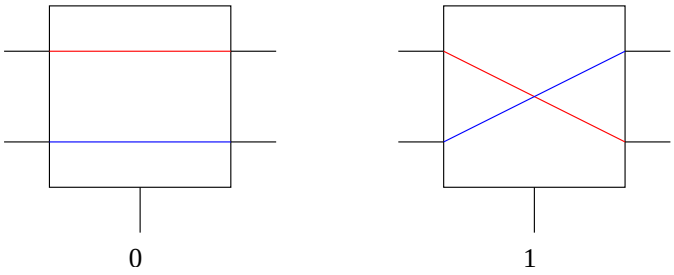
(d) Black-Box Model of the Obfuscated Half Adder

Figure 6. Adding A Key to a Half Adder

gate is only 7 gates. Additionally, they are equally easy to implement in both ASIC designs and reconfigurable hardware. The maximum delay through the polymorphic gate is 4 gate delays plus 2 inverter delays, so care must be taken if the gate is inserted into the critical path. If the gate is not inserted into the critical path then the circuit timing will likely remain unaffected.

Polymorphic switches may also be used. A 2-input, 2-output switch requires a single control line. The operation of such a switch is shown in Figure 7.

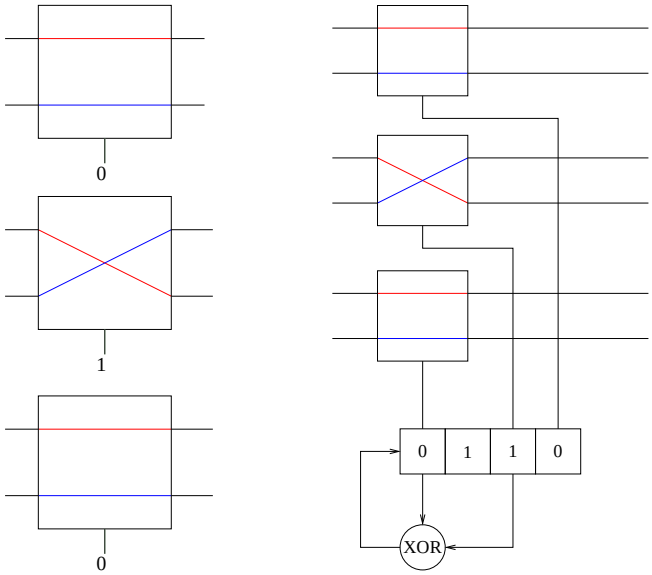
Polymorphic switches can be connected to form polymorphic networks. An example of how a LFSR could be used to form a polymorphic network that switches the order of inputs over time is shown in Figure 8. When combined with the obfuscated half adder, a dynamically changing, obfuscated, circuit is formed, shown in Figure 9.



(a) Switch with 0 applied at control line

(b) Switch with 1 applied at control line

Figure 7. Polymorphic Switch



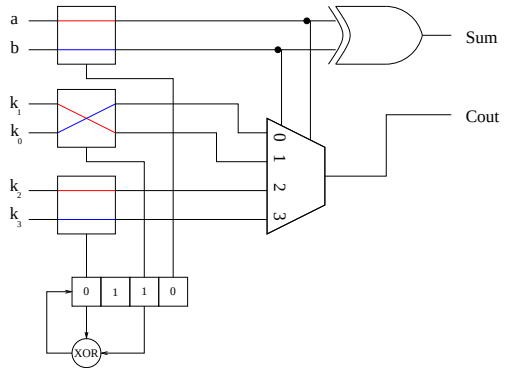
(a) Polymorphic Network

(b) Polymorphic Network controlled by LFSR

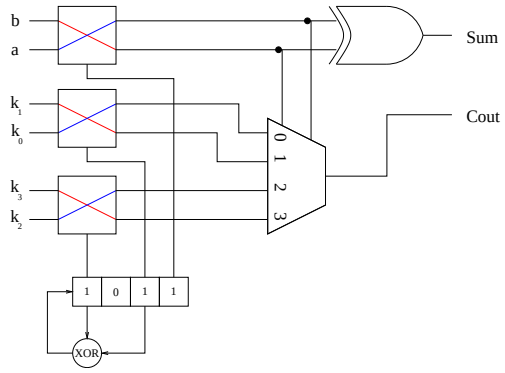
Figure 8. A Dynamically Changing Polymorphic Network

4.5.1 32-Bit Adder Cost Analysis.

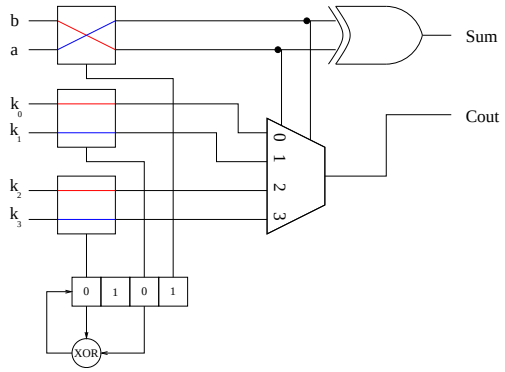
In order to estimate the cost of polymorphically obfuscating a component, a 32-bit adder is examined. As a basis, a ripple-carry adder using 32 full adders, each constructed from 2-input gates, is used. Assuming that one gate from each adder is randomly selected to be replaced, the increase in area and delay through the adder is computed. Table 2 gives the results of replacing each gate in the circuit with a polymorphic gate, and Table 3 gives the best case, worst case, and average increases



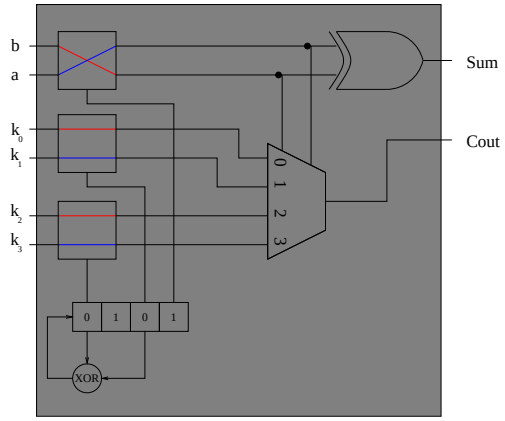
(a) Network altering the order of the inputs to the half adder



(b) After one transition of the LFSR, the order changes



(c) The order changes again after another LFSR transition



(d) Black-Box Model of the Obfuscated Half Adder and Polymorphic Network

Figure 9. A Polymorphic Obfuscated Half Adder

in area and critical path delay if random gate selection is used to replace a single gate in each circuit. The critical path of a ripple-carry adder is through the carry chain, so only the delay from the carry-in to the carry-out is considered. Approximately 80% penalty is expected to be incurred by replacing one gate in each adder.

4.6 Polymorphic Key Generation

It is not always possible for the user to provide extremely large secure keys. Generating long streams of random numbers is a complex and difficult process. Even if large secure bitstreams are available, entering the key into a device could be significantly

Table 2. Gate Replacement Costs

Gate Replaced	Delay (gates)	% Increase	Area (gates)	% Increase
None	2	0 %	11	0 %
XOR1	2	0 %	18	63.4 %
XOR2	2	0 %	18	63.4 %
AND1	2	0 %	21	90.9 %
AND2	6	200 %	21	90.9 %
OR	6	200 %	21	90.9 %

Table 3. Adder Cost Analysis

	% Increase in Delay	% Increase in Area
Best Case	0 %	63.4 %
Worse Case	200 %	90.9 %
Average	80 %	79.9 %

more difficult than any other operation that the device was designed to perform. If the key is generated from some sort of hardware signature, the area consumed by the generator grows with the signature size. Therefore, it is sometimes necessary to use the supplied key as a starting point, and have the dynamic obfuscation unit generate its own bitstream.

4.6.1 Truly Random Bitstream Generation.

The generation of a truly random bitstream is possible. If random noise is sampled, a random bitstream is produced. This is not feasible for most applications because it requires specialized hardware and a source of randomness. An example of a good source of random noise is the emission of a star.

Even if it is possible to generate a random bitstream, it is not necessarily desirable. In order for the original circuit semantics to be retained, the key bits would have to be provided to some sort of correction unit. If these key bits were read by an adversary, the alterations to the circuit could be reversed and the circuit no longer obfuscated.

4.6.2 Pseudo-random Bitstream Generation.

Various methods of generating pseudo-random bitstreams are available. This implementation uses a 64-bit LFSR to generate the key. The LSFR is initially seeded with a key of the user's choice. At whatever intervals the user desires, the LFSR can be stepped through all possible 64-bit patterns.

Several options are available as to how the polymorphic key is used. One method would be to have the initial seed hard-coded into the device. The user would have to provide a correction key in order for the device to function properly. After a set number of clock cycles, or perhaps a number depending on the inputs to the device, the next 64 bit key would be generated by the LFSR, and the user would have to provide a correction key for the new key. In this way, only a user that knows both the initial seed, LFSR configuration, and proper operating key would be able to use the device.

4.7 Level Of Protection

Consider a component with n inputs and m outputs. If the component is one of the components analyzed above the number of input vectors required to analyze the component is a polynomial function of n. When a combinational lock of size k bits is added to the component, it was previously shown that the number of vectors required to identify the circuit grew by a factor approximating $0.5n^k$. Given an 8-bit adder (16 inputs) with a 4-bit combinational lock, the circuit could likely be identified using 49152 input vectors, taking approximately 50 microseconds using a 1 GHz tester. Let T_i be the number of vectors required to identify the circuit before the combinational lock was applied (the identification period).

When that circuit is implemented polymorphically, the location of the key bits as well as the key can change with each circuit evolution. Even if the circuit semantics

remain unaltered, the identification problem is made much more difficult. Assume that after T_p (the polymorphic period) inputs are applied, the circuit is altered. Then in order to identify the circuit, only T_p traces are available. Assume that $T_p > T_i$, so each evolution of the circuit has the potential be fully identified.

After testing x configurations, the probability of having guessed the correct key configuration are computed by Equation 2. During any given circuit evolution, only T_p/T_i configurations can be tested. If the adder described above evolves every 1000 clock cycles, then the probability of it being identified during any clock cycle is about .000636.

$$P \approx 1 - (1 - (n^k - x)^{-1})^x \tag{2}$$

4.8 Evaluation

In order to demonstrate that DPR is a viable solution for protecting hardware, an evaluation platform must be constructed, and then DPR must be tested. First, the evaluation platform itself is described, followed by the general testing methodology.

4.8.1 Evaluation Platform.

Due to the cost and time associated with ASIC manufacturing techniques, and the lack of reconfigurable computing platforms that support full run-time reconfiguration, a system for testing both dynamic and static techniques is necessary. This system must meet the following requirements:

- The system must be able to be implemented on a readily available commercial FPGA
- The system must be able to take input from the user, and be able to provide feedback

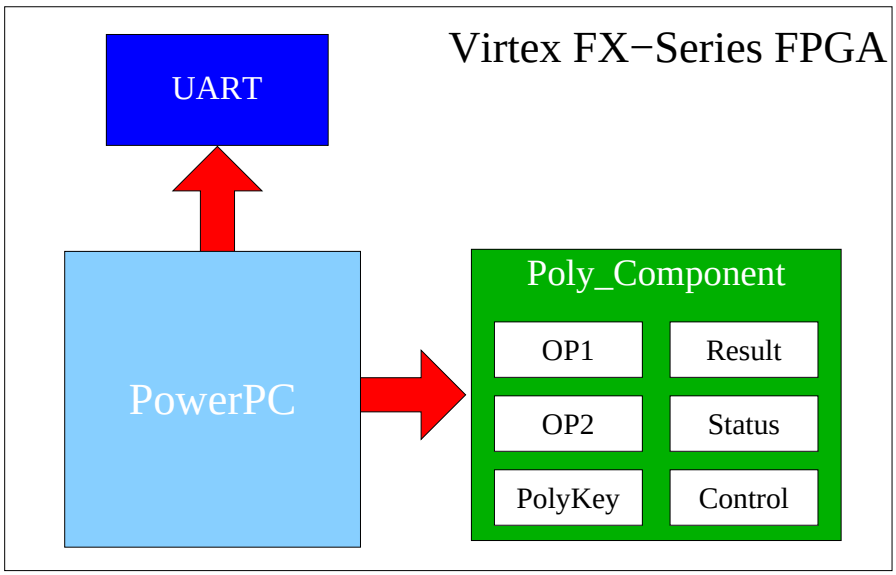


Figure 10. Evaluation System

- The system must be able to test multiple obfuscation strategies on multiple base circuits

The system that was created for test is shown in Figure 10.

The test bench is implemented on a Xilinx Virtex-5 FX series FPGA. A ML507 development and evaluation board is used, which contains a XC5VFX70TFFG1136 FPGA and supporting hardware. The components making up the obfuscation test bench are described here:

- The PowerPC is a 32-bit microprocessor that is able to interface with the FPGA on Xilinx FX series boards. The PowerPC interfaces via a bus with both the RS-232 serial UART and the polymorphic component. It receives input from and returns information to the user via the UART, and provides the inputs and records the outputs of the component under test via the polymorphic component.

- The UART is a RS-232 compliant serial UART that allows for bidirectional communication with any laptop or desktop computer with the correct port.

- The Poly_Component is a wrapper for the component under test (CUT). It includes registers for storing the operands that the CUT will operate on, the polymorphic key, the result returned by the CUT, the current status, and the control commands from the PowerPC. Any type of combinational or sequential component can be contained inside of this wrapper, with any sort of obfuscation technique applied to it.

4.8.2 Evaluation Methodology.

Obfuscated and non-obfuscated hardware will be compared by first implementing a non-obfuscated component, and verifying its functionality. The area used by the component and its speed will be recorded. Then, the component will be obfuscated, and again the functionality will be verified. The area and speed of the obfuscated component will then be compared to that of the original component in order to quantify the cost of the obfuscation.

4.9 Summary

A previously presented method of dynamically protecting components was refined so that it is suitable for implementation on an FPGA. The requirements of any implementation were proposed, and a set circuit alterations that could realize DPR was demonstrated. The additional protection achieved by these alterations, and the overhead required to implement them, were estimated. Finally, a system was described that will allow DPR to be tested on an FPGA.

V. Results

5.1 X-HIA Extension Results

The steps outlined in Chapter III were followed in identifying an adder, multiplexer, multiplier, and primitive gate. Each component was successfully identified. For clarity, the flowchart describing the identification process is shown again in Figure 11. The results, in terms of information gleaned and required number of input vectors, is presented at the end of this section.

5.1.1 Circuit 1: 4-bit Adder.

The first circuit to be identified is a 14-pin circuit (excluding power and ground connections). The circuit is shown in Figure 12.

5.1.1.1 Input/Output Identification.

The procedure previously described was followed, yielding the information listed in Figure 13. The pin numbers listed are arbitrary.

5.1.1.2 Input/Output Space Analysis.

There are 9 input pins and 5 output pins. Referencing Table 1, likely circuit candidates are an adder or a multiplexer. Because the number of input and output lines correspond exactly to a 4-bit adder with carry, an adder is selected as the most likely component. If the identification stalls under that assumption, the list of probable components will be expanded until a suitable ID is found.

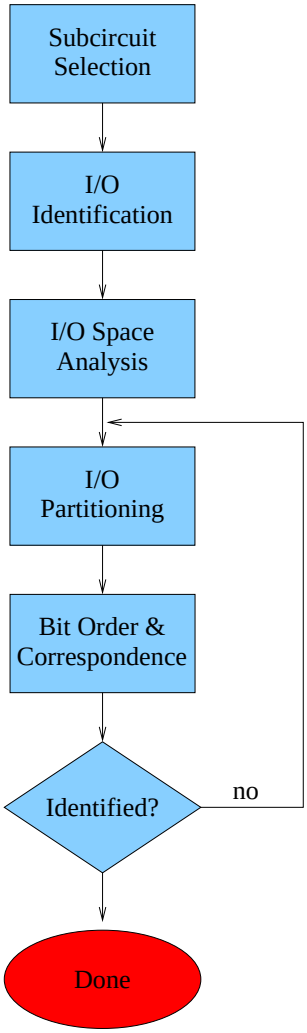


Figure 11. Process Flowchart

5.1.1.3 Input/Output Partitioning.

A generalized technique for partitioning circuits and finding bit ordering is X-hot-input analysis (X-HIA), where X refers to the number of input lines that are logically high at one time. X-HIA relies on the application of inputs such that small groups or individual signals can be identified as having an effect on specific outputs. Through 1-HIA, where a single input signal is high, it is possible to determine relationships between the inputs and outputs. These relationships are shown in Figure 14.

From the 1-HIA results, groupings can be established. The inputs can be divided

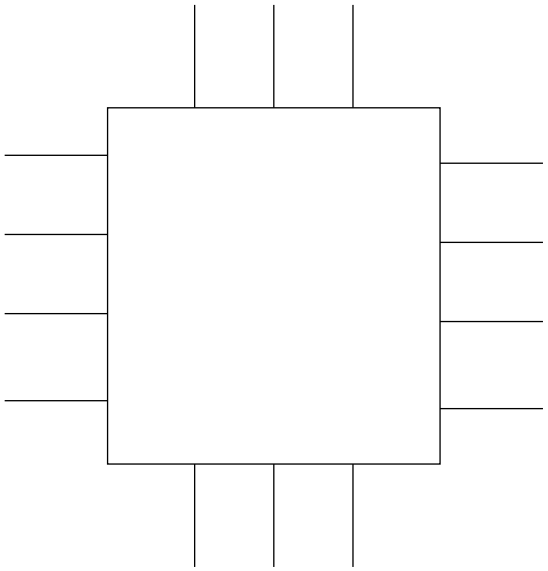


Figure 12. Circuit 1. The first circuit to be identified contains 14 input and output pins. At this point, nothing is known about the circuit other than this count.

into four groups: LSB (pins 4, 5, and 6), $group_1$ (pins 1 and 9), $group_2$ (pins 2 and 8), and $group_3$ (pins 3 and 7). The group LSB contains the 2 least significant bits of the inputs, as well as the carry-in. The two bits within each of the other three groups share significance, and cannot be distinguished. The output can be divided into two asymmetric groups: Sum and C_{out}.

5.1.1.4 Bit Ordering.

Some bit ordering can be found from the 1-HIA results as well. Pin 13 can be labelled Sum_0, as it is excited 3 times. Additionally, it can be determined which groups correspond to which bits of the sum, which will be very useful later.

The next and final step of identifying the adder is to exercise each group individually. Both signals contained within $group_1$ will be excited, then $group_2$, then $group_3$, as shown in Table 4 This will determine the relative order of each of the output pins, which will in turn determine the order of each of the input pins.

From the results above, $group_1$ can be identified as the MSB, $group_2$ as significant

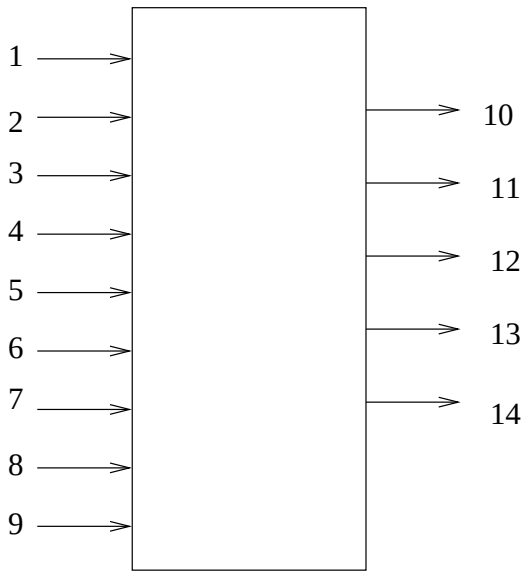


Figure 13. The ATE determines that Circuit 1 contains 9 input pins and 5 output pins.

Table 4. 2-HIA indicates the order of the input and output lines relative to each other by causing a carry operation to occur

Inputs									Outputs				
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0
0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0
0	0	**1**	0	0	0	**1**	0	0	0	0	0	0	**1**

position 2, and $group_3$ as position 1.

5.1.1.5 Identification.

The component was successfully identified as a 4-bit adder. A validation test(see [5]) can be performed to confirm that the component is in fact an adder.

Due to the nature of an adder, its inputs cannot be grouped into an A input and a B input. All that can be identified is the positional significance of each of the inputs and outputs. The circuit can be characterized as shown in Table 5.

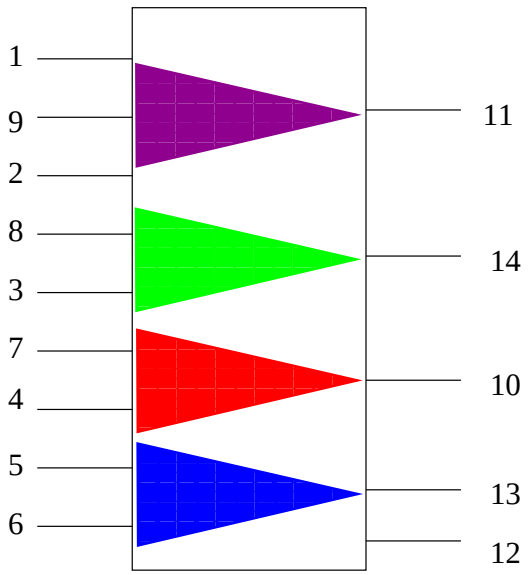


Figure 14. 1-HIA Analysis shows that each of the inputs stimulates exactly one output when applied individually.

5.1.2 Circuit 2: Multiplexer.

The second sub-circuit consists of 11 pins (again excluding power and ground connections). While this circuit is small enough that the truth table could be enumerated, the identification algorithm is used to identify it to demonstrate its feasibility.

5.1.2.1 Input/Output Identification.

The procedure listed previously was followed, indicating that the circuit contains 8 input pins and 3 output pins. The inputs are labelled 1-8, and the outputs 9-11.

5.1.2.2 Input/Output Space Analysis.

Analysis of the number and ratio of inputs and outputs would lead us to think that this is some sort of multiplexer or encoder. Since the number of inputs and outputs precisely matches that of an 8-3 encoder, the identification algorithm will start assuming that the component is an encoder.

Table 5. Circuit 1 Pin Summary

Pin Number	Pin Direction	Group Assignment	Semantic Meaning
1	In	$group_1$	Position 3
2	In	$group_2$	Position 2
3	In	$group_3$	Position 1
4	In	LSB	Position 0
5	In	LSB	Position 0
6	In	LSB	Position 0
7	In	$group_3$	Position 1
8	In	$group_2$	Position 2
9	In	$group_1$	Position 3
10	Out	Sum	$Sum.1$
11	Out	Sum	$Sum.3$
12	Out	C_{out}	Carry Out
13	Out	Sum	$Sum.0$
14	Out	Sum	$Sum.2$

Table 6. 1-HIA shows that circuit 2 is not an encoder, but may be a multiplexer

Input Excited	Output Excited
1	10
2	-
3	-
4	11
5	-
6	-
7	-
8	-

5.1.2.3 Bit Ordering: First Pass.

An encoder consists of only two groups: input and output. Therefore, no further group analysis is needed for an encoder.

Pin identification of an encoder is simple: perform 1-HIA analysis and record the output combination. The encoder is then completely described. The results of 1-HIA can be seen in Table 6.

From the 1-HIA analysis, the component is clearly not a simple encoder. However, the results are consistent with a 2-bit-wide MUX (with at least one extra pin). The

Table 7. 2-HIA shows that pins 7 and 8 are the MUX control lines

Input Excited	Output Excited
1-2	10
1-3	10
1-5	10
1-6	10
1-7	-
1-8	-

Table 8. Circuit 2 Control Analysis

Input Excited	Control Value		
	(0,1)	(1,0)	(1,1)
2	10	-	9
3	-	10	9
5	11	-	9
6	-	11	9

identification process will continue to see if the component could possibly be classified as a multiplexer. Proceeding with this new information, the process of group identification must be repeated.

5.1.2.4 Group Identification and Bit Ordering.

From the 1-HIA analysis, pins 1 and 11 will be classified as group A, and pins 10 and 11 placed into group Out. One of these will be held high, and then 2-HIA performed, pulling one other pin high at a time, and recording the results.

From the first round of 2-HIA, inputs 7 and 8 can be grouped together in the group $Control$. Now, for each possible permutation that can be placed on the control line, each of the other un-grouped inputs will be enabled, one at a time. The results are given in Table 8.

Table 9. Circuit 2 Pin Summary

Pin Number	Pin Direction	Group Assignment	Semantic Meaning
1	In	A	$A.0$
2	In	B	$B.0$
3	In	C	$C.0$
4	In	A	$A.1$
5	In	B	$B.1$
6	In	C	$C.1$
7	In	$Control$	Control Line
8	In	$Control$	Control Line
9	Out	Unassigned	Invalid Indicator
10	Out	Out	$Out.0$
11	Out	Out	$Out.1$

5.1.2.5 Identification.

These results confirm that the component does in fact act like a multiplexer. An automated system would likely ignore the extra output line, for the sake of simplicity. Manual analysis shows that this extra line is an "invalid" indicator, since only 3 of the 4 possible control line settings are valid.

The analysis of this component demonstrates the iterative nature of the identification process. Some steps, particularly group identification and pin ordering, are repeated several times as more information is gleaned about the circuit.

A summary of the information that is learned about Circuit 2 is given in Table 9. Note that unlike the adder, in which the ordering of the inputs could be determined but not the partitioning, in this case the partitioning can be determined but no ordering information can be found.

5.1.3 Circuit 3: Multiplier.

The final circuit to be analyzed in this paper contains 16 bits.

5.1.3.1 Input/Output Identification.

The procedure listed previously was followed. The circuit contains 8 inputs, labelled 1-8, and 8 outputs, labelled 9-16.

5.1.3.2 Input/Output Space Analysis.

Referring to Table 1, the circuit is likely a 4-bit $\times$ 4-bit multiplier.

5.1.3.3 Input/Output Partitioning.

Performing 1-HIA upon the circuit does not cause any outputs to be excited, which is consistent with the assumption that the circuit is a multiplier. Holding the first input line high, and "walking" a one down the rest of the inputs, allows the inputs to be partitioned into group A and group B. If the first input line is placed in group A, then all input lines do not cause an output line to be excited can be placed in group A as well, and those that do can be placed in group B.

5.1.3.4 Bit Ordering and Correspondences.

To find which bits in groups A and B share logical significance, first one group, then the other, must be held to the pattern 1111, and 1-HIA performed on the other group. The results are recorded, and the test repeated again, with the two groups interchanged. Bits that cause the same output pattern when excited share logical significance.

This process also allows for the MSB of the output, designated out_7, to be determined. It is the only bit never to be excited in the previous test. The determination of out_7 allows a_0, b_0, a_3, and b_3 to be found. This is accomplished by holding all bits of A to be logical 1, and performing 2-HIA upon B. b_3 will be high during all combinations that excite out_7. Only 1 bit, namely b_0, can be excited along with b_3

without causing out_7 to be excited. Because the correspondences between the two input groups are known, the least and most significant bits of A are now known as well.

By counting the number of times each bit went high during the modified 1-HIA analysis, output bits can be assigned two possible positions. Multiplying 1111 by 1 allows for the determination of the lower 4 bit positions(with no ordering), and so yields the significance of each output bit. This knowledge, along with the LSB of one of the inputs, allows for the significance of every input bit to be determined.

5.1.3.5 Identification.

The component is fully identified as a multiplier. Both the partitioning of the inputs, and the order of the bits, are completely discovered.

5.1.4 Circuit 4: Logic Gates.

Any of the 4 primitive logic gates(AND, OR, NAND, and NOR) of arbitrary input size may be identified with only 2 tests applied. If XOR gates are treated as primitive gates, then even/odd parity gates can be detected with the addition of one more test, for a total of 3.

The *zero response* of a gate may be defined as the output of the gate when all inputs are forced to logic zero. Only a single test vector, comprised of all zeros, is required to determine the zero response.

After determining the zero response of a gate, first 1 arbitrary input bit, then 2 should be pulled high. Table 10 indicates that 3 test vectors (all zeros, a single 1, and two 1s) can differentiate between 6 different gates.

Table 10. Logic Gate Properties

Gate Type	Zero Response	1-HIA Response	2-HIA Response
AND	0	0	0
OR	0	1	1
XOR	0	1	0
NAND	1	1	1
NOR	1	0	0
XNOR	1	0	1

Table 11. Component Identification Performance

Device Name	Identification Performance			
	Partition Input	Order Input/ Output	Define Control Lines	Running Time
n-input gate	N/A	N/A	N/A	$O(1)$
m-bit adder		$\surd$	N/A	$O(m)$
$k:1$ n-bit MUX	$\surd$		$\surd$	$O(nk)$
m-bit multiplier	$\surd$	$\surd$	N/A	$O(m^2)$
decoder	N/A		N/A	$O(2^n)$

5.1.5 Circuit 5: Decoder.

Due to the nature of a decoder, exhaustive testing must be performed to identify its operation. If more information about the circuit is known, i.e. the order of the outputs, then identification may be faster. For this reason, a decoder should be examined as part of a larger system, such as a register file, in order to glean pertinent information about it.

5.2 Identification Performance

Table 11 shows the capabilities of this method for identifying common components.

5.3 DPR Evaluation Results

DPR was implemented and tested. Two circuits were tested: an adder and an AES encrypter. First, the non-obfuscated versions of each circuit were implemented

and verified, and the area required and maximum speed were recorded. Then, the circuits were obfuscated and verified again. The area and speed of the obfuscated circuits was recorded and compared to the original.

5.3.1 Configuration.

The system described in Chapter IV was constructed. The key is stored in a LFSR, in order to facilitate a simple and repeatable key pattern. A 64-bit LFSR was used with the characteristic polynomial given in Equation 3. This is a maximal-length LFSR[17].

$$x^{64} + x^4 + x^3 + x^1 + 1 \tag{3}$$

The PowerPC and associated busses and peripheral components all operate at 125 MHz.

5.3.2 Adder Test.

The first circuit tested was a 32-bit ripple-carry adder. This adder was constructed using 32 full adders, each of which was constructed using five two-input gates. The original circuit uses conventional gates, while the polymorphic version replaces every gate with a polymorphic gate. The original and polymorphic circuits are shown in Figure 15.

The system was tested under a variety of conditions. After proper operation was verified, the polymorphic component was tested using a variety of keys. The results of this test were very promising. As expected both the size of the circuit and the maximum propagation delay increased (see Table 12). Because the increase in delay was so small, the clock frequency of the system as a whole did not have to be changed. Analysis of the output of the Xilinx implementation tools shows that the reason for

Table 12. Adder Comparison

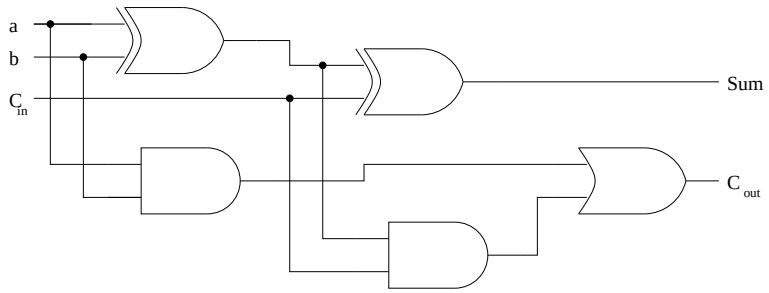
	Original Adder	Polymorphic Adder	% Increase
FF Used	255	383	50.2%
LUTs Used	268	847	216%
Max Delay	7.936ns	7.975ns	.491%
Max Freqnency	126 MHz	125 MHz	-.8%

such a small increase in the delay is because the majority of the delay is a result of signals traveling between components on the FPGA, not due to the delay of the components.

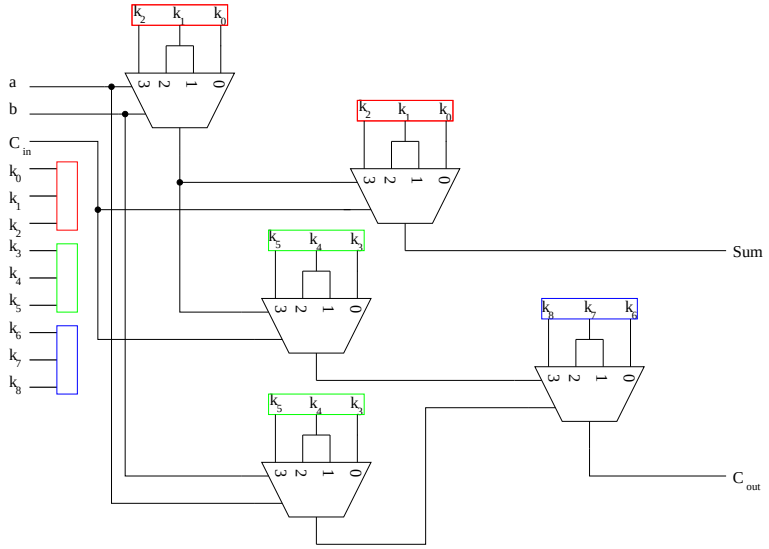
The 32-bit adder has a 9-bit key. The identification period (the number of clock cycles required to identify the adder) is 96 cycles. If the key is moved every 1000 clock cycles, then 10 identification attempts may be made during each polymorphic period. The probability of identifying the adder during any given polymorphic period is 1.11×10^{-15}. If the adder is tested continuously on a 1 GHz tester (not possible with this hardware) the probability of the adder being identified within the first year of testing is 3.4%. The probability of the adder being identified within the first decade of testing is 30%.

5.3.3 AES Test.

Polymorphic networks were added to a pipelined implementation of AES in order to make it harder to identify. The order of the inputs and outputs was altered using polymorphic switches. The component that performs the AddRoundKey operation was altered to include polymorphic gates. This component was selected because it is the simplest component to identify (it essentially consists of an array of XOR gates). Proper encryption was verified when the correct key was applied. Additionally, it was confirmed that when the key was changed, the output of the encrypter changed. The number of clock cycles required to perform the encryption was unaltered by the addition of the obfuscation elements. The difference in LUT and FF count utilized



(a) Original Adder Circuit



(b) Each gate replaced with a polymorphic gate, and a 9 key bits added to the input(3 for each gate type).

Figure 15. Test Circuits

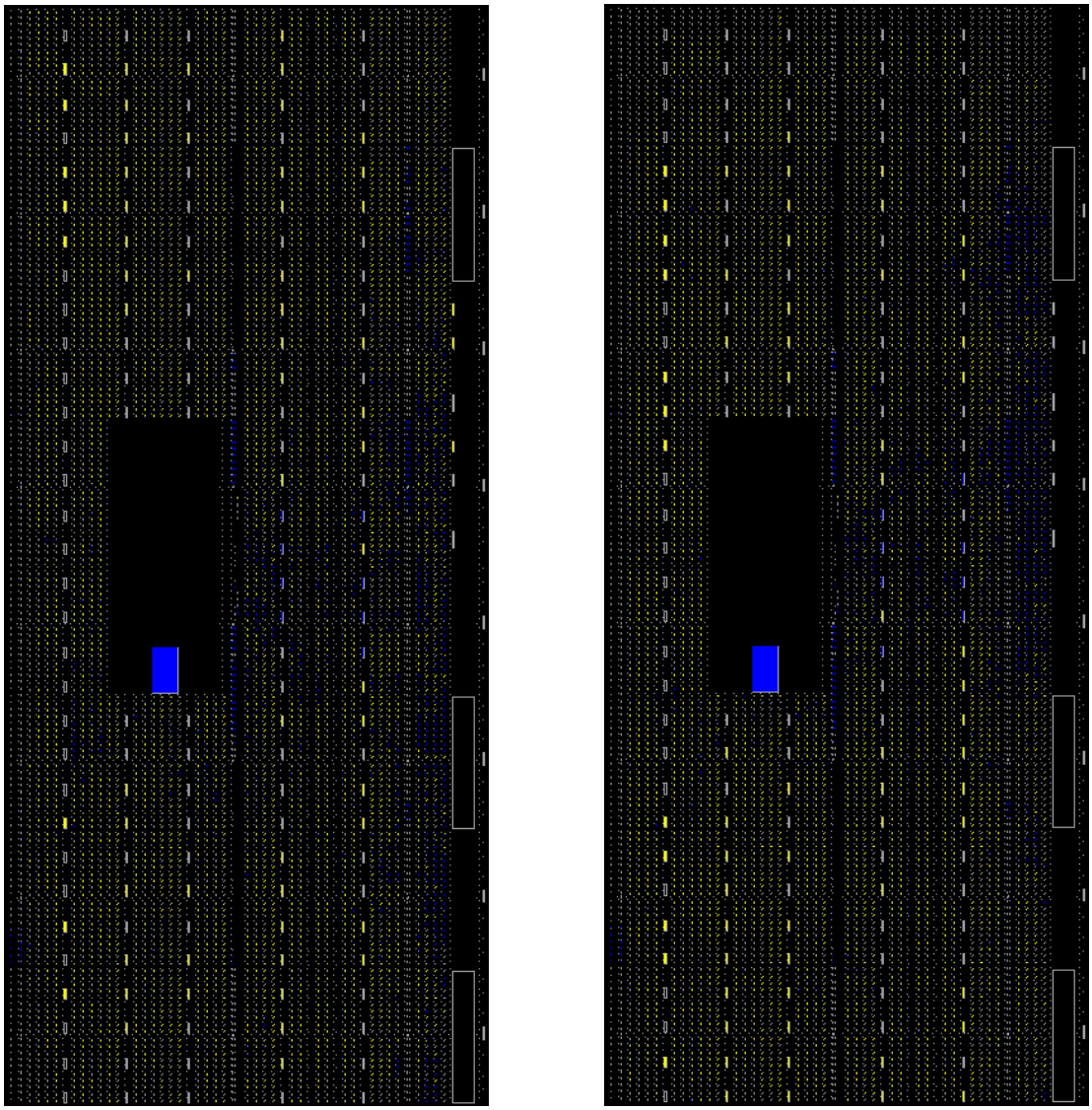
is given in Table 13. Figure 16 shows the implementation of both the original and obfuscated AES circuit on a Virtex-5 FX FPGA. The additional logic consumed is not visible by examining the layout of the FPGA without comparing the circuit netlists line by line.

The modified AES encryption component was used under the same conditions as the original encryption component. There was no observable difference in behavior other than the changes in the order of the inputs. The alterations in order to make the circuit dynamic appear to result in a robust and useful circuit. In the as-tested configuration, there was no need to insert "stalls" in the code in order to get correct

Table 13. AES Comparison

	Original Adder	**Polymorphic Adder**	**% Increase**
FF Used	13065	13193	.97%
LUTs Used	12870	13070	1.12%

results from the encryption unit.



(a) Original AES Encryption circuit implemented on a Xilinx Virtex-5 FX FPGA. The yellow portions are the sections of the FPGA containing the AES circuit.

(b) Obfuscated AES Encryption circuit implemented on a Xilinx Virtex-5 FX FPGA. There is very little difference between the original and obfuscated circuits.

Figure 16. AES Device Usage

5.4 Summary

X-HIA was extended to identify several additional components: a multiplexer, multiplier, encoder, and basic gates. The number of input vectors required to identify each of these components grows polynomially as the number of inputs grows, a significant improvement over the exponential or factorial growth rate of previously presented methods.

DPR was successfully implemented and tested on a commercial FPGA. The overhead required to implement DPR was less than predicted, due to the optimization that takes place during the FPGA design process.

VI. Conclusion

Significant progress was made in the characterization of black-box analysis threats and formalization of a DPR system. This chapter presents the conclusions drawn, covers the contributions made, and provides suggestions for future work on related topics.

6.1 Conclusions

X-HIA was extended to encompass several common circuit sub-components instead of just an adder. It was successfully shown to identify a multiplier and several simpler components in polynomial time, instead of exponential or factorial time like other methods.

After showing that many components could be efficiently identified, static protection was applied to these components and evaluated. It was shown that the static protection was only able to increase the order of the polynomial, not increase the amount of time necessary to identify the current beyond the polynomial level.

Using the information gained from extending X-HIA and implementing static protection, DPR was implemented on an FPGA and evaluated. It was shown that the amount of time that we can expect a circuit to be identified in was increased significantly over the amount of time required if only static protection is used. The overhead associated with implementing DPR was measured.

The contributions made by this research are in the following areas:

- Black-box Identification - By extending X-HIA to more components, this new identification method is shown to be viable
- Threat Characterization - Analyzing the effects of static and dynamic obfuscation against X-HIA allows for a better understanding of the threats to circuitry

- DPR System Formalization - While DPR was previously described by Porter, no system for its use had been formalized

6.2 Future Work

The work presented here leads to several exciting future research opportunities: white-box characterization of the polymorphic components, implementation of the X-HIA identification algorithm, and automation of the adaptation of an arbitrary component to DPR.

6.2.1 White-Box Characterization.

The polymorphic components added to the circuit may present a white-box vulnerability. If an attacker has access to white-box information about the circuit the topology of the gates may be uncovered. This could leak information about the circuit to the adversary. In order to further protect the circuit, white-box obfuscation may need to be performed on the circuit once it is prepared for DPR.

6.2.2 X-HIA Implementation.

Using the basic system-level framework that was used to analyze DPR, the X-HIA process could be automated. The benefits of this would be twofold: the availability of an efficient component identification technique and the ability to quickly quantify the obfuscatory effects of black-box transformations.

6.2.3 Automation of DPR preparation.

Currently, a gate-level netlist must be modified by hand in order to implement DPR. This is not practical for very large and complex circuits. This process could be automated, significantly decreasing the amount of time necessary to protect a circuit

and also allowing for the testing of many different selection routines against each other.

6.2.4 Uncloneable Functions.

The latest research in digital fingerprinting allows for the generation of keys that are linked to individual pieces of hardware. If these keys are used to control the DPR system, then a FPGA bitstream could be generated that would only function properly on a single device.

6.3 Summary

X-HIA can be extended to identify several common components. Static protection techniques can slow down the identification process, but it is still feasible to identify a component using currently available technology. DPR can be used to significantly decrease the chances of a component being identified.

Bibliography

[1] "Military Critical Technologies List". Http://www.dtic.mil/mctl/MCTL.html.

[2] "ps2 mod mars modchip". Http://www.repair-modchip-modchips-ps2-playstation-xbox-wii-brisbane.com/product-pages/ps2-mars-install-diagrams.htm,.

[3] "Specification for the Advanced Encryption Standard (AES)". Federal Information Processing Standards Publication 197, 2001. URL `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[4] Benson, Pam and Michael Sefanov. "Iraqi insurgents hacked Predator drone feeds, U.S. official indicates", december 2009. Http://edition.cnn.com/2009/US/12/17/drone.video.hacked/index.html.

[5] Bushnell, M. L. and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, volume 17 of *Frontiers in Electronic Testing*. Springer, 2000.

[6] Chikofsky, E.J. and II Cross, J.H. "Reverse engineering and design recovery: a taxonomy". *Software, IEEE*, 7(1):13 –17, jan 1990. ISSN 0740-7459.

[7] Compton, Katherine and Scott Hauck. "Reconfigurable computing: a survey of systems and software". *ACM Comput. Surv.*, 34(2):171–210, 2002. ISSN 0360-0300.

[8] Doom, T., J. White, A. Wojcik, and G. Chisholm. "Identifying high-level components in combinational circuits". 313–318. Feb 1998. ISSN 1066-1395.

[9] Halderman, J. Alex, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. "Lest We Remember: Cold Boot Attacks on Encryption Keys." Paul C. van Oorschot (editor), *USENIX Security Symposium*, 45–60. USENIX Association, 2008. ISBN 978-1-931971-60-7. URL `http://dblp.uni-trier.de/db/conf/uss/uss2008.html#HaldermanSHCPCFAF08`.

[10] Hansen, M.C., H. Yalcin, and J.P. Hayes. "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering". *Design & Test of Computers, IEEE*, 16(3):72–80, 1999. ISSN 0740-7475.

[11] Kocher, Paul, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". 388–397. Springer-Verlag, 1999.

[12] Kocher, Paul C. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". *CRYPTO*, 104–113. 1996.

[13] Miyamoto, A., N. Homma, T. Aoki, and A. Satoh. "Enhanced power analysis attack using chosen message against RSA hardware implementations". 3282 –3285. may 2008.

[14] Porter, Roy. *Critical Technology Tamper Protection Through Dynamic Polymorphic Reconfiguration*. Master's thesis, Air Force Institute of Technology, 2009.

[15] jacques Quisquater, Jean, Franois Koeune, Werner Schindler, and Werner Schindler. *Unleashing the Full Power of Timing Attack*. Technical report, Universite Catholique de Louvain – Crypto Group, 2001. 120, 130 BIBLIOGRAPHY 159, 2001.

[16] Ruzicka, R., L. Sekanina, and R. Prokop. "Physical Demonstration of Polymorphic Self-Checking Circuits". 31 –36. july 2008.

[17] Schneier, Bruce. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995. ISBN 0-471-11709-9.

[18] Sedcole, P., B. Blodget, T. Becker, J. Anderson, and P. Lysaght. "Modular dynamic reconfiguration in Virtex FPGAs". *Computers and Digital Techniques, IEE Proceedings* -, 153(3):157 – 164, may 2006. ISSN 1350-2387.

[19] Silva, Miguel L. and Joo Canas Ferreira. "Support for partial run-time reconfiguration of platform FPGAs". *Journal of Systems Architecture*, 52(12):709 – 726, 2006. ISSN 1383-7621. URL `http://www.sciencedirect.com/science/article/B6V1F-4KBDWT1-1/2/463c0451f1fdc863474e4b72b654f373`.

[20] Skorobogatov, Sergei. *Semi-invasive attacks - A new approach to hardware security analysis*. Technical report, University of Cambridge, Computer Laboratory, Technical Report UCAM-CL-TR-630, 2005.

[21] Standaert, O.-X., E. Peeters, G. Rouvroy, and J.-J. Quisquater. "An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays". *Proceedings of the IEEE*, 94(2):383 –394, feb. 2006. ISSN 0018-9219.

[22] Stone, S.J., R. Porter, Y.C. Kim, and J.V. Paul. "A dynamically reconfigurable Field Programmable Gate Array hardware foundation for security applications". 305 –308. dec. 2008.

[23] Thomas S. Messerges, Ezzy A. Dabbish and Robert H. Sloan. "Investigations of Power Analysis Attacks on Smartcards". 1999.

[24] White, J.L., M.-J. Chung, A.S. Wojcik, and T.E. Doom. "Efficient algorithms for subcircuit enumeration and classification for the module identification problem". 519 –522. 2001.

[25] Wollinger, Thomas and Christof Paar. "How Secure Are FPGAs in Cryptographic Applications?" *Proceedings of International Conference on Field Programmable Logic and Applications (FPL 2003), Lecture Notes in Computer Science Volume 2778*, 91–100. Springer-Verlag, 2003.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2010	Master's Thesis	Sept 2008 — Mar 2010

4. TITLE AND SUBTITLE

Static and Dynamic Component Obfuscation
on
Reconfigurable Devices

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Camdon R. Cady, 2d Lt, USAF

5d. PROJECT NUMBER

10-299

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/10-06

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Robert L. Herklotz
Air Force Office of Scientific Research, AFMC
801 North Randolph Street, Rm 732
Arlington VA 22203-1977
703-696-9544 (DSN: 426) robert.herklotz@afosr.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFOSR/NL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Computing systems are used in virtually every aspect of our lives. Technology such as smart phones and electronically controlled subsystems in cars is becoming so commonly used that it is virtually ubiquitous. Sometimes, this technology can be exploited to perform functions that it was never intended to perform, or fail to provide information that it is supposed to protect. X-HIA was shown to be effective at identifying several circuit components in a significantly shorter time than previous identification methods. Instead of requiring a number of input/output pairings that grows factorially or exponentially as the circuit size grows, it requires only a number that grows polynomially with the size of the circuit. This allows for the identification of significantly larger circuits. Static protection techniques that are applied to the circuits do not increase the amount of time required to identify the circuit to the point that it is not feasible to perform that identification. DPR is implemented, and it is shown both that the overhead is not prohibitive and that it is effective at causing an identification algorithm to fail.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Yong Kim
U	U	U	U	145	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 4620 yong.kim@afit.edu