6-17-2010

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

Daniel F. Koranek

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

THESIS

Daniel Koranek, CIV

AFIT/GCO/ENG/10-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/10-15

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

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

Daniel Koranek, B.S.

CIV

June 2010

AFIT/GCO/ENG/10-15

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

Daniel Koranek, B.S.

CIV

Approved:

Dr. Yong C. Kim (Chairman)	date
Lt Col Jeffrey T. McDonald, Ph.D. (Member)	date
Dr. Michael R. Grimaila (Member)	date
Maj Eric D. Trias, Ph.D. (Member)	date

AFIT/GCO/ENG/10-15

Abstract

Reverse-engineering is a threat to U.S. Military technology; obfuscation protects circuits and software programs by transforming them in a way that makes them harder to reverse-engineer. The Random Program Model (RPM) proposed by Yasinsac and McDonald hypothesizes that an obfuscated program is indistinguishable from one with random structure and output.

One existing method for whitebox obfuscation based on the RPM consists of randomly selecting subcircuits from a circuit and replacing these subcircuits with a randomly selected, semantically equivalent replacement. This method enumerates all possible replacement subcircuits that it could select, and randomly selects one. The number of semantically equivalent subcircuits grows intractably for increasing circuit size, making subcircuit replacement intractable.

To improve upon the efficiency and security of this implementation of the RPM, this research identifies two alternative methods for generating semantically equivalent circuit variants which leave the circuit's internal structure pseudo-randomly determined. In *component fusion*, random selection employed by the previously mentioned RPM implementation is replaced with a deterministic selection based upon component identification, and random replacement is replaced with a deterministic algorithm that generates canonical logic forms. *Component fusion* is demonstrated to produce security and efficiency improvements over random subcircuit selection-and-replacement, but is bounded by intractable growth of runtime and circuit replacement size in relation to the number of inputs in the subcircuit selection. *Component encryption* seeks to alter the semantics of individual circuit components using an encoding function, but preserves the overall circuit semantics by decoding signal values later in the circuit.

Experiments were conducted to examine the performance of component fusion and component encryption against representative trials of subcircuit selection-and-replacement and Boundary Blurring, two previously defined methods for circuit obfuscation. Overall, results support the conclusion that both component fusion and component encryption generate more secure variants than previous methods and that these variants are more efficient in terms of required circuit delay and the power and area required for their implementation. Component encryption is shown experimentally to require shorter runtime than other methods offering similar protection, showing increased usability.

Acknowledgements

I would like to thank my advisor Lt. Col. Todd McDonald, who was deployed prior to the completion of my thesis but provided invaluable guidance in the selection of my thesis topic and advanced my knowledge of the subject area; Dr. Yong Kim, for spending many hours mentoring me and refining the concepts of my thesis; and the rest of my committee, Major Eric Trias and Dr. Michael Grimaila, for their patience in working with a computer science student trying to complete a thesis requiring a background in algorithms and electrical engineering.

There are also many AFIT students without whose help I would not have succeeded. These include Lt. Jamey Parham, Miles McGee, my fellow SFS students, and VLSI students Lt. Jeff Falkinburg and Capt. Robert Trejo. Thank you all for your help and for listening patiently to long descriptions of circuit experiments.

I owe a large debt of gratitude to my family for their love and support, and to my parents specifically for homeschooling me through high school and for encouraging me to complete college and graduate school. Also, I thank the folks at my church for their love and support of me as a single grad student, for teaching me to properly maintain my car, and for providing me many free lunches. Most importantly, though, I would like to thank God for giving me any talents that I have, for guiding me to the Air Force Institute of Technology, and for giving me the strength and perseverence I needed to complete this Masters program.

Daniel Koranek

Table of Contents

Page

Page

Page

List of Figures

Figure Page

Figure Page

List of Tables

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

I. Introduction

Computing technology in the form of circuits and software is used across the entire Department of Defense (DoD). The DoD estimates its 2010 research and development budget at almost 79 billion dollars, encompassing all of Research, Development, Test, and Evaluation [21]. While this budget is not allocated specifically for development of computing technologies, it represents a significant investment. Technologies developed through research and development are deployed across the United States armed services and are used around the globe in the course of military operations. The success of Joint operations often depends upon the coordinated use of computing technologies across military branches in communications, intelligence collection, weapons deployment, and protection and dissemination of sensitive information.

Adversaries with the ability to understand the designs of technology belonging to the United States can identify inherent flaws or weaknesses (e.g., software and network vulnerabilities) which might be turned into an attack, manipulate the portions of the system within their control (e.g., feeding intelligence to an intelligence collection system), or clone the good qualities of the system to copy them. In one respect, if parties representing threats to the United States are able to understand U.S. military technology designs, a strategic advantage has been lost because some advantage lies in possessing superior technology. In another respect, some critical information is inherent to the design of systems themselves. DoD Directive 5200.39 describes some information pertaining to systems as "Critical Program Information":

> *. . . [information], technologies, or systems that, if compromised, would degrade combat effectiveness, shorten the expected combat-effective life of*

the system, or significantly alter program direction. This includes classified military information or unclassified controlled information about such programs, technologies, or systems.

1.1 Motivating Scenarios

Specifically, the United States Air Force is seeking protection mechanisms for protecting hardware from adversaries. While raw data can be encrypted to protect it against an eavesdropper (rendering it indistinguishable from random data), circuits and software, collectively referred to as *programs*, are limited by their nature: a program must be able to perform a function and all of the information necessary to compute that function must (traditionally) be available within the program. This makes computing technology a weak link in information security - functional information stored in hardware and software is vulnerable to static and dynamic analysis.

The scenarios below serve to motivate the development of software and circuit protections in this thesis.

1.1.1 Reverse Engineering a Cryptographic RFID Tag. In a well-known study presented at the Black Hat 2008 security conference, Nohl et al. [32] examined the Mifare Classic RFID tag used in ticketing for access control in several large public transportation systems because the chip is relatively inexpensive. The Oyster card used in London by TranSys formerly used the Mifare chip [1], as did the SmartRider card used in Australia [40].

Nohl et al. reverse-engineered the cipher implementation used in the Mifare Classic RFID tag by dissolving the plastic card surrounding the tag using acetone and mechanically polishing away successive layers of the RFID tag. Using a microscope, Nohl et al. photographed individual layers of the chip. Since only about 70 different gate implementations are used within the chip, templates for each gate type were created and a template matcher identified additional instances of the type within the chip.

Focusing their efforts on reverse-engineering the cipher implementation, Nohl et al. found that the cipher implementation consists of a 48-bit linear feedback shift register. A brute-force attack to discover the key used by the card and reader was estimated to take about 50 minutes, but cheaper attacks are also possible because of weaknesses found in the cipher design. Nohl et al. reported that technology capable of breaking 52-bit keys has been available since 1998. Because of the weaknesses that Nohl et al. found in the Mifare Classic, TranSys made the decision to start using a more robust Mifare RFID chip based upon the DES algorithm [1].

In essence, Nohl et al. have shown that merely implementing a circuit design in silicon offers little protection for the physical circuit design. Without security protections, reverse-engineers will be able to easily recover the gate-level and transistor-level schematics of circuitry.

1.1.2 Technology Cloning. In the private sector, a company may attempt to reverse-engineer a competitor's product in order to gain a competitive edge; *Popular Science* reported that selling cloned versions of a product even before the real product is sold is common overseas, and that cloned versions of the Apple iPhone and American automotive models are sold at reduced prices [25]. Outsourcing manufacturing facilities to other nations has aggravated this problem of intellectual property theft because in many instances it allows untrusted individuals access to the schematics for constructing a product.

A study conducted by the House of Representatives in 1999 observed a foreign nation which seeks to become technologically independent; in order to accomplish this, scientists from this nation are pressured to reverse-engineer technology rather than purchase it. In one instance, scientists reverse-engineered a high-performance U.S. computer rather than spending a fraction of the cost to buy it. Reverse-engineering products may incur significantly more expense than importing them, but this nation appears willing to expend resources specifically in order to become less dependent on foreign technology. In order to obtain access to more foreign technology, the

study claims that this nation depicts itself as more technologically advanced than it really is (making it seem less as though obtaining U.S. technology will improve their capabilities) [41].

1.1.2.1 Hainan Island Incident. In light of foreign interest in copying U.S. technology, there was considerable tension in April of 2001 when an EP-3E surveillance aircraft collided with a Chinese aircraft and was forced to land on Hainan Island because of damage. The EP-3E contained cryptological equipment, sensors, and other sensitive computing equipment. While the crew of the EP-3E was able to destroy some of the equipment, the Chinese retained possession of the plane for several months. During that time, Chinese scientists were free to analyze the technology with the end goal of reverse-engineering it [34, 44]. The incident on Hainan Island serves to motivate the protection of critical computing technologies; in the case that a foreign entity ever obtains military technology belonging to the United States, the technology should be protected to such a degree that analyzing it is as difficult as possible (or hopefully impossible).

1.2 Scope

The primary focus of this thesis is on the problem of generating versions of combinational circuits which convey less information to an adversary about the circuit's purpose than the original circuit, i.e., the problem of generating variants which are *obfuscated*. For background on existing circuit obfuscators, see Section 2.5.

This thesis is constructed to answer two questions. The primary research question regards the feasibility of replacing random circuit obfuscators with deterministic circuit obfuscators, and the second research question regards the capabilities of random circuit obfuscators. Both questions are elaborated in the following sections.

1.2.1 Primary Goal: Construction of New Circuit Obfuscators. The primary research question of this thesis is:

1. Is it possible to construct efficient methods that generate securely obfuscated versions of combinational logic circuits for the purpose of protecting software and hardware systems?

This research question is significant for several reasons; firstly, though, because it concerns the usability and quality of combinational logic obfuscators. If obfuscators do not produce secure combinational logic variants, then there is no benefit to obfuscation. If obfuscators or the variants they produce are too inefficient to be used, then the obfuscators are unusable. In total, there are three requirements which can be derived from this research question to define the type of combinational logic obfuscators that are sought:

1. **Semantic Equivalence**: New circuit variants must compute the same function as the original circuit. This is also described as "functional correctness."
2. **Improved Security**: New circuit variants should be more secure than the original circuit. This means that the structural and internal functional information of the original circuit should be protected.
3. **Equal or Improved Efficiency**: New circuit variants should be produced efficiently, and the variants themselves should be efficient. For obfuscated circuit variants, this means that the amount of power and area required to implement the circuit should not increase significantly. For the obfuscators themselves, efficiency means that usability should not be limited by runtime.

The research goals for this thesis were formulated from the research question, and thus accomplishing the research goals requires the creation of the obfuscator described by the research question. The primary goal of this research is to:

1. Develop new deterministic methods for whitebox obfuscation which are both efficient and secure.

Accomplishing this research goal requires that new deterministic methods be created, and that they demonstrably have the characteristics of the obfuscators in the research

question. To demonstrate that deterministic obfuscators developed in this thesis meet the requirements of the primary research question, this research attempts to achieve the following sub-goals:

1. **Semantics-Preserving Algorithms**: Obfuscated circuits will be verificably semantically equivalent to the original circuit.
2. **Secure Variants**: Obfuscated circuits will be more secure than the original circuit. Security will be measured by:
   - The elimination of functional information in the circuit. This will be measured by the percentage of common truth-table values produced by the gates in both the obfuscated and the original circuits.
   - The elimination of structural information in the circuit. This will be measured by the number of components in the original circuit which cannot be detected in the obfuscated circuit.
3. **Efficient Algorithms and Variants**: Obfuscated circuits will be as efficient or more efficient than the original circuit. This will be measured by
   - The number of hierarchical levels in the obfuscated circuit, representing the delay with which the circuit produces results.
   - The number of gates in the obfuscated circuit, representing the area required to implement the circuit and the power required for using the circuit.

   In addition, obfuscation algorithms will require less time for generating obfuscated circuits than previous algorithms. This will be measured by
   - The amount of time required for the generation of obfuscated circuits.

To accomplish the primary research goal two new methods for obfuscation, *component fusion* and *component encryption*, are presented in Sections and respectively.

1.2.2 Secondary Goal: Evaluation of Preexisting Circuit Obfuscators. As mentioned above, this research constructs new combinational logic obfuscators in the light of two preexisting methods for generating obfuscated digital logic circuits. The first, termed random subcircuit selection-and-replacement (SSR), consists of randomly selecting gates from a logic circuit and replacing them with a random equivalent replacement [33]. Random SSR is unfortunately limited in efficiency by its requirement on enumerating large circuit families in order to generate subcircuit replacements randomly. In addition, the security of random subcircuit selection-and-replacement has not yet been validated, and whether it implements the theoretical security model on which it is based is still an open question. This research thus seeks to further clarify the properties of a truly random circuit variant generator in relation to subcircuit selection-and-replacement.

Thus the secondary research question, centered around the analysis of random SSR, is formulated as follows.

2. Does the random subcircuit selection-and-replacement method successfully implement the Random Program Model (RPM) for security, such that the security provided by implementing the RPM is present in a random subcircuit selection-and-replacement method?

This evaluation of random SSR is provided in Appendix B.1.

1.2.3 Assumptions. Several assumptions are made in the course of this research to better describe the problem that this thesis is attempting to solve. Firstly, the domain of this problem has been restricted to only combinational logic circuits. Restricting the domain of this problem is necessary because all sequential circuits are composed of combinational components; the problem of combinational logic obfuscation must be examined before the problem of sequential logic obfuscation can be addressed. Defining the capabilities of an adversary is also helpful because it identifies ways in which information can be leaked from a circuit. This research, secondly, assumes that information is leaked by the existence of copies of known subcircuits

within the circuit. An adversary will identify information gained by analysis of the structure of the circuit, including that of the existence of individual components and the ways in which they are related, and use this information to reconstruct the intent of the circuit.

1.3 Organization

Chapter II will focus on providing background material for this thesis, including digital logic concepts, concepts related to hardware and software security, program transformation frameworks, and reverse-engineering. In Chapter III, two schemes for deterministic white-box variation (component fusion and component encryption) are proposed, and Chapter IV details how the implementations of these schemes are evaluated according to their effectiveness in protecting circuit components. Chapter V details the conclusions which can be drawn from the results and proposes future work areas.

The appendices contain primarily reference material. Appendix B.1 describes the methodology used in evaluating the strength of random subcircuit selection-and-replacement, which is the second research goal. It does this by comparing random subcircuit selection-and-replacement against an ideal system for circuit manipulation. Appendix C contains pseudo-code for algorithms described in this thesis.

II. Background

This chapter serves to establish the importance of hardware and software obfuscation, describe background concepts related to these areas, and describe previous research into combinational circuit obfuscation systems. In following chapters, this thesis will seek to make improvements to the circuit obfuscation framework described in Section 2.5. A brief summary of this chapter is as follows.

Section 2.1 begins by discussing the relationship between hardware and software. Because the hardware and software domains overlap, a holistic protection strategy is sought. Further, because of the functional relationship between hardware and software, advances in the protection of circuits can be applied to software protection.

Section 2.2 considers reverse-engineers: individuals who will attempt to recover information about the intent of a system. This is examined in both the hardware and software domains, and strategies for reverse-engineering hardware modules are explored. Also, this section examines paradigms for program intent and paradigms for the reverse-engineer's ability to understand a program. Section 2.3 examines previous research into formal theoretical models for circuit and software obfuscation, and Section 2.4 examines the framework of Abstract Interpretation for modeling the relationship between program syntax and program semantics, and for modeling the domains that a reverse-engineer must consider in order to understand a program. Section 2.5 provides background on previous efforts at implementing combinational logic obfuscators and previous research into quantifying the security of combinational circuit variants.

Lastly, Section 2.6 provides background into the alternative representation of logic circuits as Binary Decision Diagrams.

2.1 Blurring the Distinction Between Hardware and Software

A traditional, conceptual divide has been drawn between the domains of *hardware* and *software*, with the term *hardware* usually referring to logic circuit designs manifested in physical technology and *software* referring to bit patterns that can be

executed on hardware-based multiprocessors. However, with the emergence of reconfigurable computing technologies such as Field Programmable Gate Arrays (FPGAs), circuits are increasingly being implemented in a software-like manner.

Rather than manufacturing application specific integrated circuits (ASICs) to perform specialized processing tasks such as video and audio processing, compression, encryption, and other tasks, FPGAs are programmed to perform these tasks. Vahid [42] has suggested that this allows task executions at speeds which are magnitudes faster than similar implementations in register-transfer language on microprocessors, but that the reconfigurable nature of FPGAs allows new algorithms to be implemented without the additional cost in chip manufacturing associated with developing ASICs.

Vahid suggests that it is important to recognize that FPGAs are just another platform for software, and that recognizing this fact will improve the design of embedded systems which are traditionally implemented as software running on microprocessors. Particularly, the only difference between traditional software and an FPGA circuit design is the modeling domain - traditional software development in languages such as C is temporally oriented, while circuit design is a spatially oriented form of software development.

Kim and McDonald [23] build on this assumption, and describe how viewing circuits as software should impact the way in which hardware and software protections are considered. Hardware protection is traditionally viewed as the task of providing successful anti-tamper measures, and software protection remains difficult to define but is usually characterized in terms of watermarking, obfuscation, and tamperproofing as described by Collberg [10]. If Vahid is correct, then many software protection concepts should be adapted to the design of hardware because there will soon cease to be a distinction between logic circuit designs for implementation in physical hardware and logic circuit designs implemented as software on an FPGA. In other words, every logic circuit design will be potentially software.

Thus, rather than viewing the two as distinguished categories, logic circuit designs and traditional software program code both being considered software) and physical circuit designs should be considered under the category of *computing technology*. Techniques for protecting critical computing technology will require a new, holistic strategy which integrates the protection of physical circuitry with the protection of non-physical software that is executed on it.

2.1.1 Modeling Software as Circuits. From a more theoretical viewpoint, there are certain classes of functions which can be computed by both logic circuits and software, and serve to further make circuits and software indistinguishable. Wee gives one example of such a function (computing equality) in [46].

Any combinational circuit easily maps to a piece of software which computes the same function; logic gates map directly to code instructions which compute the same function, and gate outputs map to program variables. Translating software structures to circuit structures is also possible for most programming constructs; a limited subset of constructs is provided by VHDL (**V**ery High-speed Integrated Circuit **H**ardware **D**escription **L**anguage), which is used widely for circuit design and simulation. A larger set of programming constructs was mapped to circuitry by Wirth, who defined methods for translating variable declarations, subroutines, and high-level control statements directly into circuit structures and implemented an experimental compiler targeting logic circuits [52].

Another mapping was proposed by Norman [33], who demonstrated that combinational logic circuits compose a small language, which can be represented in Backus-Naur form using the grammar

$$B ::= true|false|(!B)|(B\&B)|(B||B)|(E < E)$$

where B represents any Boolean expression and E represents any integer expression. This logic grammar allows for all combinational logic to be mapped to some software

program in a grammar which contains the same constructs. Further, sequential circuits may be decomposed into combinational components, where sequential circuits retain state and memory through feedback in addition to performing computations, allowing sequential circuits to also be represented, though not feasibly for large sequential circuits, in a Backus-Naur grammar.

2.2 Reverse-Engineering

Seminal works in the field of reverse-engineering were written to describe methods for recovering design from systems for which designs have been lost. Sometimes, a system needs to be modified or repaired in the course of normal maintenance, but the documentation of the system provides inadequate design information. To recover the design, a brief phase of reverse-engineering must occur before maintenance can occur.

One of the earliest references to reverse-engineering defines the reverse-engineering process as "the act of creating a set of specifications for a piece of hardware by someone other than the original designers, primarily based upon analyzing and dimensioning a specimen or collection of specimens." [38, page 244] A more general definition has been given by Chikofsky and Cross [8, page 14]: "Reverse engineering is the process of analyzing a subject system to ... create representations of the system in another form or at a higher level of abstraction."

2.2.1 Traditional Software Reverse-Engineering. The process of compiling a software executable typically consists of several steps which progressively transform high-level code into assembly code for a particular processor architecture. As shown in Figure 2.1, source code is parsed into a syntax tree, intermediate code is generated and control flow is optimized, and assembly code is generated from intermediate code. The last step in this process is the assembling of assembly code into machine code for execution on a target architecture.

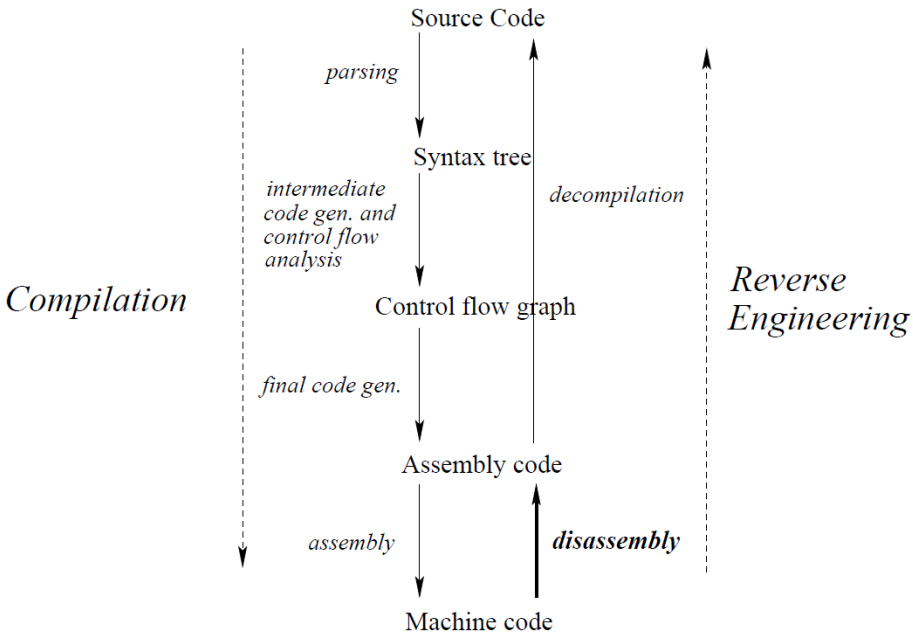


Figure 2.1: The processes of software compilation and software reverse-engineering. [27]

Linn and Debray [27] described the process of statically reverse-engineering executable software programs as a combination of two steps; the first being the re-creation of assembly code from machine code in a process termed disassembly, and the second being the recreation of high-level source code from assembly code through decompilation. Because of information lost in executable creation, source code recreated through reverse-engineering attempts will never precisely recreate the original source code, but recreated source code will better communicate to the reverse-engineer the functionality of the software.

Dynamic software reverse-engineering through the use of software debuggers or virtualization techniques is also possible. This provides reverse-engineers with the ability to examine how a piece of software affects a running processor. By allowing a reverse-engineer to control the execution of a piece of software, analysis of running program states can be performed which cannot be performed in real-time. Commercial reverse-engineering tools such as IDA Pro [19] and OllyDbg [53] (and many others) exist for helping with these tasks.

2.2.2 Traditional Hardware Reverse-Engineering.

2.2.2.1 Reverse-Engineering using Circuit Schematics. The ISCAS-85 benchmark circuits are a set of 10 combinational logic circuits introduced at the International Symposium of Circuits and Systems in 1985, consisting of circuits whose functions and high-level designs were not published for confidentiality reasons and for the purpose of allowing them to be used as random logic circuits for test purposes. In 1995, Hansen et al. [18] conducted a study in reverse-engineering in which the high-level design for each of the benchmark circuits were recreated, and the functionality of each circuit was described.

Hansen et al. described the techniques they used in reverse-engineering the functionality of the set of benchmark circuit designs. The principles they followed offer a level of insight into the process that an attacker would use in uncovering the functionality of a circuit:

1. *Library modules* - Common components such as multiplexors, decoders, adders, and other mathematical functions often appear in circuit designs.
2. *Repeated modules* - Circuit designs often include particular modules repeated many times. Regardless of whether the function of a repeated module is known, this offers a visual indication to a reverse-engineer that that the subcircuit computing that function is significant.
3. *Expected global structures* - After recognizing structures which are commonly used, a reverse-engineer can search for other circuit structures, signals, or functions which usually accompany these common structures.
4. *Computed Functions* - The logic function that a circuit computes can be constructed from individual components, offering insight into the particular mathematical function being computed.
5. *Control Functions* - Signals which are reused within a circuit are probably being used as control logic.

6. *Bus structures* - The signals produced as output by many repeated modules can often be grouped into buses, and noting where these signals lead can help to partition the circuit design.
7. *Common names* - In cases where names are available and included in the circuit design, signal names or structure names (even non-descriptive ones) can be used as identifiers. If a name is used more than once to refer to different structures, then the structures are probably connected.
8. *Black boxes* - Grouping unknown functionality together into a "black box" is used to separate known functionality from unknown functionality.

2.2.2.2 Reverse-Engineering based on Side-Channel Attacks and Gray-Box Attacks. Kim and McDonald [23] identify a set of attacks which provide information to a reverse-engineer, even in the absence of circuit schematics. Running circuitry either leaks information in specific ways or can be attacked in ways which provide information not available through examination of schematics (or in the absence of available schematics):

1. **Focused Ion Beams** - This tool is similar to a scanning electron microscope, but uses gallium ions instead of electrons. Using a FIB, it is possible to set specific intermediate signals within a circuit to be a specific value (0 or 1). This makes it possible to examine internal structure by holding particular signals constant.
2. **Optical Equipment** - Optical probes rely on the interactions of photons with silicon devices and allow circuit examination by looking at transistor states to provide an adversary with the ability to observe a signal propagated by means of applied input values.
3. **Power Consumption** - By observing the power usage across a circuit, an attacker can gain insight into what signals are changing the most in the circuit.

Particularly, if a circuit is being used to compute some mathematical function, more power will be used in the area of the circuit dedicated to this function.

4. **Timing Analysis** - Timing attacks are the circuit version of software debugging. In a timing attack, a reverse-engineer modifies the speed of the circuit clock, either slowing it down to allow additional analysis or speeding it up to achieve a desired effect.

5. **Fault Injection** - Primarily, fault injection is used to merely prevent the correct function of a circuit; however, it can aid reverse-engineering or aid in circuit tampering. Using fault injection to prevent correct key selection in a cryptographic circuit is one particular example of the way fault injection could be used in circuit tampering.

This list is not exhaustive, but does serve to illustrate the threat of physical circuit tampering. For more information concerning side-channel and gray box attacks on circuitry, see [23].

2.2.3 Program Understanding. McDonald and Yasinsac [30] considered a reverse-engineering adversary and reason about the goals of a reverse-engineer. They considered that a reverse-engineer can achieve understanding of the intent of a program (either a piece of circuitry or a piece of software) in the light of four different paradigms:

1. Program understanding manifested by the ability to predict a program's future output.
2. Program understanding based upon comparing a program's code segments to those of code libraries whose function is known.
3. Program understanding manifested by the ability to gain any information at all from a program's structure.
4. Program understanding based upon information present in program code.

McDonald and Yasinsac classify these four paradigms into two domains of program understanding, *black-box understanding* and *white-box understanding*. These two domains provide a basis for classifying the information which can be observed about a program.

Definition 1. ***Black-Box Understanding/Obfuscation:*** *Program $P \leftarrow X, Y$ is black-box understandable if and only if, given an arbitrarily large set of pairs $IO = (x_i, y_i)$ such that $y_i = P(x_i)$ and y_j an arbitrary element of Y with (x_j, y_j) not an element of IO, an adversary can efficiently guess x_j such that $y_j = P(x_j)$ with greater than negligible probability. Otherwise, we say P is black-box obfuscated. [30]*

Definition 2. ***White-Box Understandable:*** *The program p'_{n+1} is white box understandable if the probability is greater than or equal to $\frac{1}{2} + \epsilon$ (where ϵ is the negligible error probability), that the adversary A is able to distinguish whether p'_{n+1} is either $E(p_{n+1})$ or is a random program P_R. Therefore, the program encryption algorithm $E(p)$ provides white box obfuscation if and only if an adversary is able to distinguish the encrypted program (p'_{n+1}) from a random program (P_R) with probability less than or equal to $\frac{1}{2} + \epsilon$, where ϵ is the negligible error probability. [28]*

These two concepts are combined to form *intent protection*:

Definition 3. ***Intent Protection:*** *Program P is intent protected if and only if it is protected against black-box and white box analysis.*

In other words, a program P is understandable in the black-box sense if an adversary can predict the input that will cause some arbitrary output with odds better than random guessing. A black-box obfuscated program represents one in which an adversary has no ability to predict I/O relationships. This does not necessarily mean that the adversary cannot understand anything about P; a black-box obfuscated program may still leak information by its program structure.

McDonald and Yasinsac's second, third, and fourth paradigms above describe program understanding based in the white-box domain. A encrypted program P', then, is white-box understandable if it can be distinguished from a randomly generated

program of the same size with greater than negligible probability. Reasoning that an encrypted program P should contain the random properties found in encrypted data, McDonald and Yasinsac suggest that random selection of a program P' from the set of all possible programs implementing the function of P guarantees the randomness of P's structure and provides a baseline for measuring the entropy of program structure.

2.3 Obfuscation

Obfuscation is defined by Varnovsky et al. [43] to be "any efficient semantic-preserving transformation of computer programs aimed at bringing a program into such a form, which impedes the understanding of its algorithm and data structures or prevents the extracting of some valuable information from the plaintext of a program." More generally, Barak et al. define an obfuscator to be "an efficient, probabilistic 'compiler' that takes as input a program or circuit P and produces a new program O(P) that has the same functionality as P yet is 'unintelligible' in some sense." [2]

2.3.1 Virtual Black Box Obfuscation. Barak et al. were able to produce a seminal impossibility result in the field of obfuscation by demonstrating that constructing a Virtual Black Box (VBB) obfuscator is impossible. [2] According to the definition of a VBB obfuscator, an obfuscator O must satisfy two conditions, with a third condition defined by the domain:

- **functionality**: $O(P)$ must compute the same function as P
- **"virtual black box" property**: Anything that can be efficiently computed from $O(P)$ can be efficiently computed given oracle access to P.
- **polynomial slowdown**: $O(P)$ must be at most polynomially slower than P. In the domain of circuits C, this means that $|O(C)| \leq p(|C|)$; in the domain of Turing Machines M, this means that the description length and running time of $O(M)$ are at most polynomially larger than those of M.

To demonstrate the impossibility of building a VBB obfuscator, Barak et al. constructed a family of functions F which are unobfuscatable under the VBB model because there exists a property $\pi : F \leftarrow 0, 1$ such that

(a.) Given any program that computes a function $f \in F$, the value $\pi(f)$ can be efficiently computed.

(b.) Given oracle access to a randomly selected function $f \in F$, no efficient algorithm can compute $\pi(f)$ much better than random guessing.

In other words, the obfuscation of functions in F is impossible because there is some property which can be computed from F, but not from oracle access to F. Barak et al. were able to first construct 2-Turing Machine and 2-Circuit functions within F (showing that no 2-Turing Machine or 2-Circuit VBB obfuscator exists), and then extend the result to show that no VBB Turing Machine or Circuit obfuscator exists. Because of this impossibility proof, Barak et al. concluded that obfuscators, if they exist, are only possible under definitions of obfuscation other than VBB.

2.3.2 Indistinguishability Obfuscation. Following an impossibility proof for the VBB model, Barak et al. proposed an alternative definition of obfuscation to show that some (weaker) definition for obfuscation could exist: *indistinguishability obfuscation.* The indistinguishability obfuscation model is identical to VBB obfuscation model, with the exception that the VBB property is replaced with an indistinguishability property:

- **indistinguishability**: For any PPT A, there is a negligible function a such that for any two circuits C_1, C_2 which compute the same function and are of the same size k,

$$|Pr\left[A\left(O\left(C_1\right)\right)\right] - Pr\left[A\left(O\left(C_2\right)\right)\right]| \leq a(k)$$

Restated, under indistinguishability obfuscation, if the Turing Machine A finds the difference between $O\left(C_1\right)$ and $O\left(C_2\right)$ to be less than $a\left(k\right)$ according to some metric,

then the obfuscations are indistinguishable to A. This definition is weaker than the VBB definition, but allows for the possibility that an obfuscator might exist. Under VBB, an obfuscated circuit cannot leak any information at all that cannot be derived from oracle access, but under indistinguishability obfuscation, an obfuscated circuit can leak more than negligibly more information than any another circuit implementing the same function would leak. Barak et al. proved that inefficient indistinguishability obfuscators do in fact exist.

2.3.3 Best-Possible Obfuscation. Goldwasser and Rothblum identified a problem with the Barak et al.'s definition for indistinguishability obfuscation: indistinguishability obfuscation does not provide any guarantee that obfuscation prevents the leakage of information. (It is possible that both of the two obfuscated variants leak about the same amount of information, but that this amount is rather large.) To improve upon this definition, Goldwasser and Rothblum proposed best-possible obfuscation:

- **Best-Possible Obfuscation**: An algorithm O, which takes as input a circuit in C and outputs a new circuit, is said to be a (computationally/statistically/perfectly) best-possible obfuscator for the family C if it has the preserving functionality and polynomial slowdown properties as above, and also has the following property (instead of the virtual black-box property).
  - **Computational/Statistical/Perfect Best-possible obfuscation**: For all large enough input lengths, for any polynomial size circuit adversary A, there exists a polynomial size simulator circuit S such that for any circuit $C_1 \in C_N$ and for any circuit $C_2 \in C_N$ that computes the same function as C_1 and such that $|C_1| = |C_2|$, the two distributions $A(O(C_1))$ and $S(C_2)$ are (respectively)computationally/statistically/perfectly indistinguishable.

Restated, a circuit obfuscated under best-possible obfuscation will leak only the information that can be leaked by all other circuits that compute the same function.

This means that all circuits may theoretically be obfuscated under the definition of best-possible obfuscation, but makes no claim about the feasibility of constructing such an obfuscator.

2.3.4 Random Program Model. McDonald and Yasinsac [29] distinguish *obfuscation* from *program encryption*. While obfuscation and program encryption both seek to make a program more difficult to understand, program encryption takes on some properties of traditional cryptographic ciphers. Cryptographic ciphers seek to produce ciphertext which leaks no non-trivial information regarding the original plaintext, and thus produce output which is indistinguishable from a stream of random bits.

Given that encrypted data shares many of the characteristics of a random bitstream, McDonald and Yasinsac reason that a random program can be used to measure successful program encryption. If an encrypted program shares the characteristics of an encrypted ciphertext, then that encrypted program should also resemble a random bitstream. However, there are several other characteristics which a random program must possibly also satisfy.

If a stream consists of random bits, then there is no guarantee that it consists entirely of legal instructions from a particular instruction set architecture. Further, there is no guarantee that a sequence of legal instructions will run successfully (and not fail due to a runtime error). A random program must be legal both syntactically and grammatically, not fail at runtime, and must be able to halt. Without being syntactically and grammatically correct, a program will fail to compile; without being fail-safe and able to halt, a program will not appear functionally correct.

According to McDonald and Yasinsac, a program producing encrypted output (representing a semantic transformation applied to the program) which is pseudo-random can be said to have *Strong Black-box Security*:

- **Strong Black-box Security** - Given program p', which

1. Implements program encryption algorithm $\mathbb{E}$ $p' = \mathbb{E}(p)$
2. Takes input x
3. Produces output $y = p(x)$

After knowing any I/O pairs $(x_1, p'(x_1)), (x_2, p'(x_2)) \ldots, (x_{n-1}, p'(x_{n-1})), (x_n, p'(x_n))$, an adversary that supplies any set of subsequent input $x_k, x_{x+1}, x_{k+2}, \ldots$ cannot predict in polynomial time either the correct outputs $p'(x_k), p'(x_k + 2), \ldots$ of the obfuscated program p' or the correct outputs $p(x_k), p(x_{k+1}), p(x_{k+q} \ldots$ of the original program p.

In addition, McDonald and Yasinsac provide a white-box security requirement for the RPM which is essentially based around the definition for White-box understandability they provide in [30]. If a program or circuit is not White-box understandable, then that program or circuit has a measure of White-box security.

2.3.5 Measuring Obfuscation. In [9], Collberg et al. designed a Java program obfuscator which took as input a Java program and produced as output an equivalent Java program containing opaque constructs in order to increase the difficulty of reverse-engineering. The modified (but equivalent) Java program contained additional, irrelevant statements to hide the real control flow of the program, contained object code sequences for which no high-level constructs exist, and modified existing control flow abstractions. To evaluate the quality of obfuscating transformations, Collberg et al. described four qualities which describe obfuscating transformations in the general sense:

- **Potency**: The amount of obscurity (or complexity, or unreadability) added to the program. This definition is distinct from *potency* as defined by Cousot and Cousot [12], but the two definitions are not mutually exclusive.
- **Resilience**: The difficulty of constructing an automatic deobfuscator to reverse obfuscating transformations.

- **Stealth**: The degree to which obfuscated code blends in with the rest of the program.
- **Cost**: The amount of computational overhead added to the program.

Potency measures the benefit to program security gained by introducing obfuscation into a program; cost represents the penalties incurred by introducing obfuscation (e.g., program runtime and executable size). Resilience indicates the difficulty of constructing an automated tool to detect and remove obfuscation, while stealth indicates the difficulty overall of detecting and removing obfuscation. A resilient construct might not be stealthy if the construct is difficult to detect automatically, but a human could easily discover it.

2.4 Abstract Interpretation

2.4.1 Design of Program Transformation Frameworks. AI [12] is a framework for formalizing the correspondence between a program's syntax and its semantics at different layers of abstraction. Cousot and Cousot distinguish syntax and semantics by reasoning that a program's syntax (described by physical representations of program code, such as source code and assembly code) is an abstraction of the program's semantics (which may consist of additional qualities such as the sequence of values held by a variable during runtime or other properties of an actual program execution).

Cousot and Cousot model layers of abstraction of a program's semantics using individual domains modeled as partially ordered sets.

- **Concrete semantics** describe a syntactically correct program $P \in \mathbb{P}$ and belong to a concrete domain $\mathbb{D}$ which is a partially ordered set $\mathtt{po} < \mathbb{D} :\sqsubseteq >$ when ordered by the approximation ordering $\sqsubseteq$ formalizing a loss of information.
- **Abstract semantics** describe a safe/conservative approximation of the concrete semantics which belongs to an abstract domain $\overline{\mathbb{D}}$ and is also a partially ordered set $\mathtt{po}\,\langle\overline{\mathbb{D}} : \overline{\sqsubseteq}\rangle$.

- **Abstraction** is a *Galois connection*, a mapping between two partially ordered sets, which indicates how entities in the concrete domain $\mathbb{D}$ correspond to entities in the abstract domain.
- **Concretization** is also a Galois connection which describes how entities in the abstract domain map to entities in the concrete domain.

Using the AI framework, Cousot and Cousot reason about transformations applied to programs at varying levels of abstraction.

- **Syntactic transformations** t transform the syntax of a program.
- **Semantic transformations** transform the semantics of a program.
- **Potency**: A syntactic transformation t is *potent* if there is some semantic property which t does not preserve.
- **Correctness**: A syntactic transformation t is *correct* if, at some level of abstraction, t preserves meaning.

While any syntactic transformation will transform the semantics of a program to some degree, a correct syntactic transformation will preserve the operational semantics of the program. Figure 2.2 shows a transformation t which transforms the semantics of the program P but preserves observational semantics at an abstract level.

2.4.2 Code Obfuscation using Abstract Interpretation. Dalla Preda and Giacobazzi use AI as a basis for modeling software obfuscation in the white-box domain [13, 36, 37]. Obfuscation preserves the correct operation of software on some level, but modifies its source. Because of this, Dalla Preda and Giacobazzi choose to model a program obfuscator as a δ-obfuscator if it is a potent program transformation $\tau : \mathbb{P} \rightarrow \mathbb{P}$ such that every program is equivalent to its obfuscated version with respect to a particular observational semantics δ.

Based upon this model for obfuscation, Dalla Preda and Giacobazzi present schemes for obfuscating program control flow [13] and for detecting opaque predicates in program code [37].

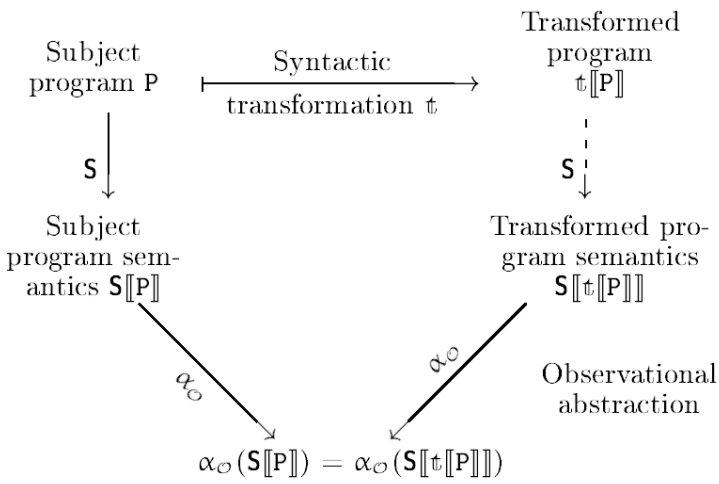


Figure 2.2: Correctness of a syntactic transformation [12]

2.4.2.1 Opaque Predicate Detection. *Opaque predicates* are defined to be predicates whose values are known prior to program execution, because the program will always cause those predicates to evaluate to the same value. Dalla Preda and Giacobazzi present several examples of opaque predicates; an `if` statement in program code whose conditional will always evaluate to *true* or *false* (causing dead code to exist in the program) or functions whose values will always retain a certain property (for example, $\forall x \in \mathbb{Z} : n|f(x)$ - a function f always returns a multiple of n). Opaque predicate insertion is one method for obfuscating program code because it is thought that inserting opaque predicates makes it more difficult for a reverse-engineer to analyze a program. The technique opaque predicate insertion was implemented in 2003 when Linn and Debray [27] proposed a method of this type based on opaque predicate insertion for preventing static code analysis.

Dalla Preda and Giacobazzi specify an attacker who is able to break an opaque predicate (that is, detect that the program construct is, in fact, opaque) by modeling the abstract domain in which the opaque predicate resides. Once the level of abstraction of the attacker reaches the same level as that on which the opaque predicate is opaque, the attacker is able to recognize the predicate's opaqueness.

While a particular attacker might be able to detect certain opaque predicates but not others, Dalla Preda and Giacobazzi suggest that opaque predicates requiring a level of abstraction different from that of the attacker could interact with predicates that *are* within the domain of the attacker, preventing the attacker from identifying the inserted opaque predicates. If opaque predicates were drawn from a large enough set of domains, an attacker might be required to combine all domains used in the program opaque predicates in order to break any single opaque predicate.

2.5 Circuit Obfuscation

2.5.1 Algorithms for White-box Obfuscation Using Randomized Subcircuit Selection and Replacement. Norman [33] implemented a simplified white-box program randomizer for obfuscating combinational logic circuits. This obfuscator models combinational logic circuits as directed acyclic graphs (DAGs) and attempts to satisfy the white-box obfuscation requirement of the Random Program Model. An obfuscator satisfying the requirements of the RPM should transform the white-box structure of a circuit C into any other white-box implementation of C in a circuit family $\delta_{i-o-g-\Omega}$ with equal probability, and to accomplish this, the architecture described by Norman enumerates all of the possible white-box implementations within $\delta_{i-o-g-\Omega}$. However, Norman recognizes that as circuit size increases, the size of possible replacements from the family $\delta_{i-o-g-\Omega}$ grows too large to enumerate. While estimating the size of semantically-equivalent circuit replacement families remains an open problem, Simonaire [39] provided empirical results which measure the size of circuit families with given input/output and gate count.

In absence of a tractable method for circuit family enumeration, a different method for replacing all of the gates within a circuit was developed, based around SSR:

Definition 4. ***Subcircuit selection and replacement:*** *Given a circuit C which is to be white-box obfuscated, select a subcircuit, C_{sub}. Retrieve a randomly chosen circuit C_{rep} from a library of circuits which contains a set of all circuits semantically*

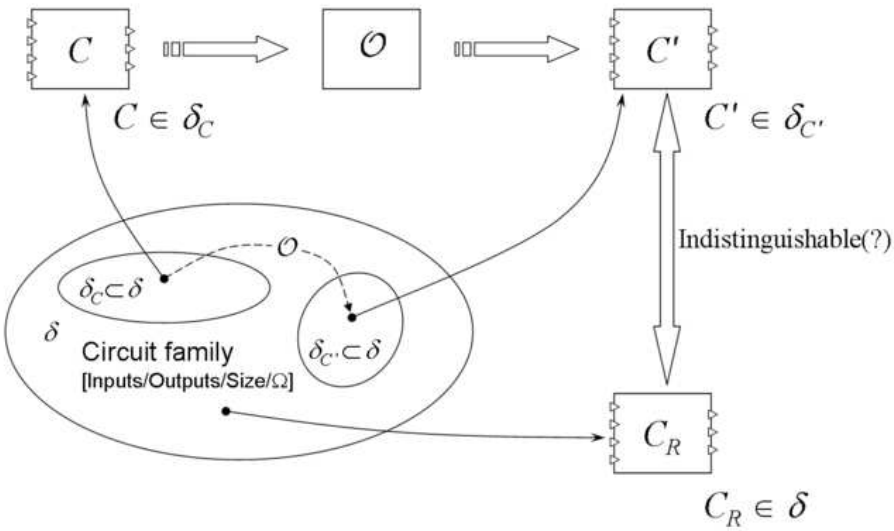


Figure 2.3: The intent protection model: Obfuscation of a subcircuit C by the selection of C, application of black-box changes to produce the black-box behavior of C', and white-box randomization to select from all circuits implementing the function of C'. [33]

equivalent to C_{sub}. Finally, remove C_{sub} from C and insert C_{rep} in its place. As long as C_{sub} and C_{rep} are semantically equivalent (and the order of inputs and outputs is preserved), then semantic equivalence exists for C, all C'_i, and C'_n.

Norman notes that developing an algorithm which uses a library of *all* circuits semantically equivalent to C_{sub} would violate the polynomial slowdown property of the RPM. To overcome this limitation, bounds are placed on the size of the circuits in the library, and a given C_{rep} is selected from among all size-bounded circuits semantically equivalent to C_{sub}.

2.5.1.1 CORGI. A Java-based tool named CORGI (Circuit Obfuscation via Randomization of Graphs Iteratively) was developed as an implementation of the white-box circuit randomizer Norman described. Several strategies for selecting subcircuits were developed, providing methods for random selection of individual gates. Smart selection strategies were also developed that select subcircuits with

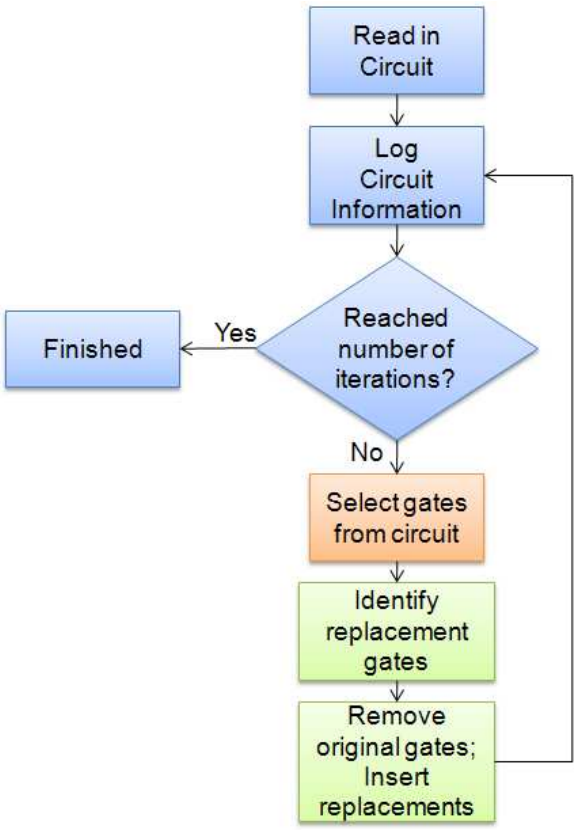


Figure 2.4: The process flow of random subcircuit selection-and-replacement.

given metrics (input size, output size, circuit size, gate basis, and/or truth table) or subcircuits isomorphic to some other subcircuit.

2.5.2 Removing Redundant Logic Pathways in Polymorphic Circuits. Artifacts of subcircuit selection-and-replacement often still exist following certain obfuscation experiments. These artifacts are visible as small patterns which are easily optimized (e.g., buffers, double inversion, or using 0/1 as inputs to `AND` or `OR` gates). Kim [22] observed these patterns after analyzing obfuscated circuits produced by the random select-2 replace-with-3 algorithm (2-3SSR), and sought to describe and implement methods for automatically minimizing these small logic patterns. In all, twelve algorithms for identifying and reducing small logic patterns were created. These al-

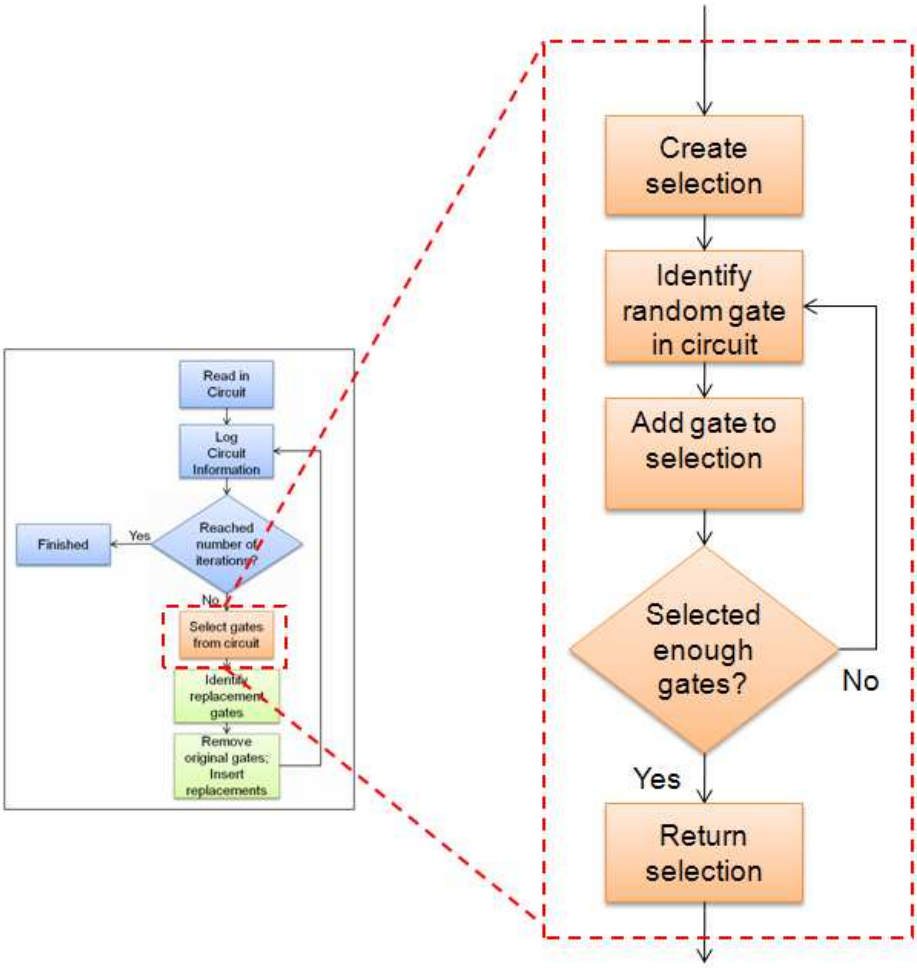


Figure 2.5: The process flow of random subcircuit selection

gorithms represent a minimizer that an adversary might use to simplify circuit logic, and were added to the functionality of CORGI to

2.5.2.1 Other Work in Pattern-Based Reduction. The concept behind Kim's pattern-based logic reduction, local gate pattern optimization, has long been a part of commercial circuit minimization algorithms. In 1981, Derringer et al. described a logic optimization system based upon traditional compiler optimization that identified and minimized local logic patterns [15]. In 1984, a more formalized tool termed the *Logic Synthesis System*(LSS) was developed which utilized these techniques [14]. Brayton et al. note when describing the ESPRSSO optimization algo-

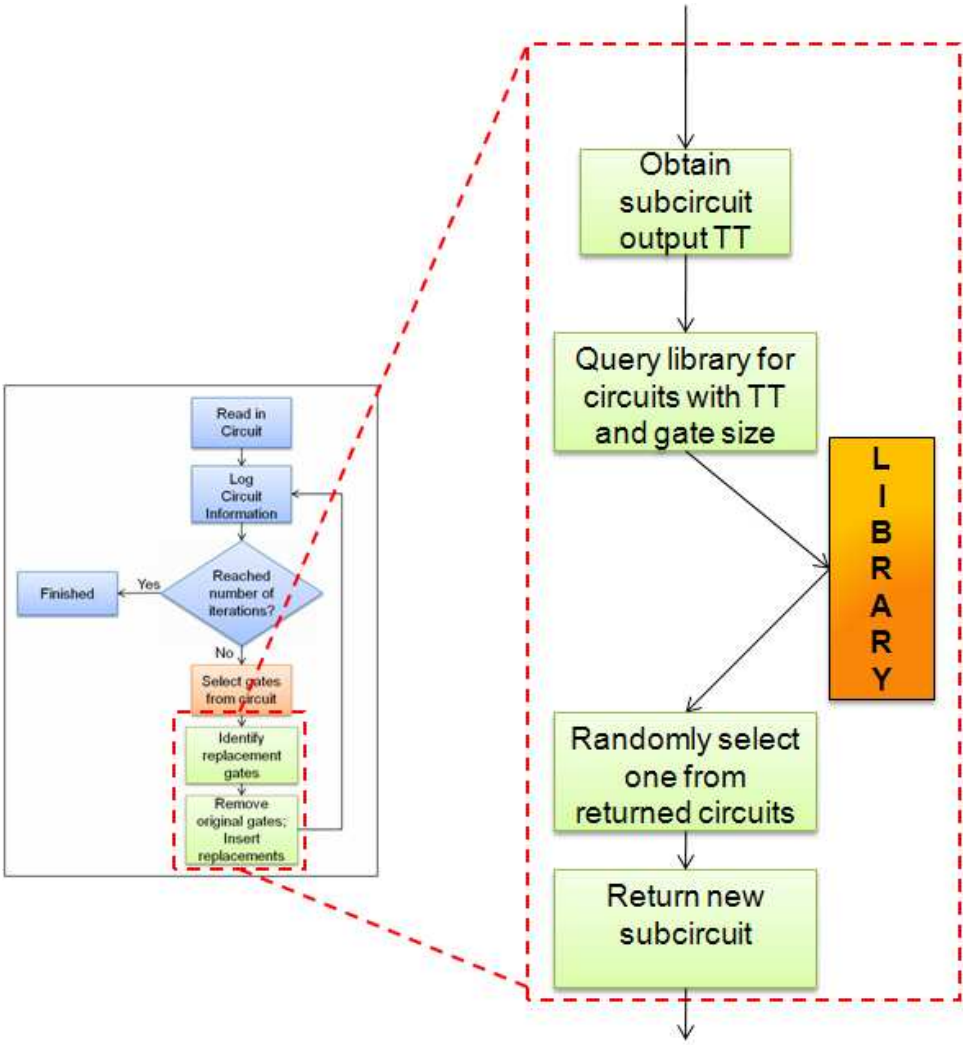


Figure 2.6: The process flow of random replacement.

rithm that two-level optimization is most useful when combined with this type of local logic pattern reduction techniques [5]. The ESPRESSO two-level optimization algorithm is more fully described in Section 3.3.2.1.

2.5.3 Ancestral Entropy. Williams [51] described a measurement for circuit obfuscation which attempted to describe changes in a circuit that occur during iterations of subcircuit selection and replacement. This metric, termed "ancestral entropy," was proposed to measure the uncertainty regarding the component to which a particular gate in a circuit belongs, following variation applied by a subcircuit se-

lection and replacement tool. Williams used the following definitions to describe the concept of ancestral entropy:

Definition 5. ***Circuit:*** *A directed, acyclic graph $D(V, E)$ of inputs, logic gates and outputs where 1) V is the set of nodes of the circuit where each node is either an input with an assigned value or a gate with a corresponding Boolean function $f : \{0, 1\}x\{0, 1\} \rightarrow \{0, 1\}$, 2) E is the set of wires that connect nodes, 3) the gate set G of a circuit is the set of all gate nodes within V, 4) inputs are nodes in the graph with no fan-in and 5) outputs are distinguished nodes.*

Definition 6. ***Component:*** *Given a gate set G and an input set I of a circuit P and an integer $k > 1$, where k is the number of components, a set C of components $\{c_1,...,c_k\}$ partitions G and I into k disjoint sets of inputs and/or gates.*

Definition 7. ***Ancestry:*** *Given m number of gates in a circuit, k number of defined components in a circuit, and a tuple $T = \{t_1, t_2, ..., t_{k-1}, t_k\}$ assigned to each gate g_m in a circuit, where t describes the composition of g_m in terms of the nth component.*

Definition 8. ***Ancestral Entropy:*** *Let $P = (p_i, ..., p_n)$ be a probability distribution on the set of $N = (1, ..., n)$ components in a circuit. The entropy of P is the function $H(P) = -\sum_{i \in N} p_i log_2(p_i)$, where N is the number of components in a circuit and p_i is the probability of a gate being from a defined component i. Maximum ancestral entropy, $H(P)_{max}$, is achieved when $p_i = p_{i+1} = ... = p_{n-1} = p_n$. $H(P)_{min}$ is achieved when $p_i = 1$.*

2.5.4 Circuit Family Size. Simonaire applied Walenstein et al.'s result [45] in normalizing metamorphic malware to the problem of normalizing circuits obfuscated by the CORGI subcircuit selection and replacement engine. To do this, he modeled possible subcircuit selection and replacements as a series of term-rewriting system operations [39]. The rule set of term-rewriting operations was examined for convergence, which would have indicated the existence of a perfect normalizer for obfuscated circuits. However, the subcircuit selection and replacement rule set was found to contain critical overlaps which prevented the rule set from converging.

An important result produced by Simonaire was the empirical measurement of the size of certain circuit families through enumeration. (These enumerated circuit families were enumerated by input count, output count, and gate count, without regard to the function computed by any particular circuit.) The circuit families consisting of 1-input, 1-output, n-gate circuits (the δ_{1-1-n} family) constructed from a one-gate basis were found to be identical in size to corresponding integers in the AT&T Research Labs series A000366. This correlation is one of the first theoretical results quantifying the cost of circuit family enumeration.

The integer series A005439 (counting the number of Boolean functions of n variables whose ROBDD (see Section 2.6) contains at least n branch nodes, one for each variable) is equivalent to A000366 multiplied by 2^{n-1}, and Simonaire's empirical count of circuit family constructed using a six-gate basis (AND, OR, NAND, NOR, XOR, and XNOR) is equivalent to A000366 multiplied by 6^{n-1}. This is because the counting of possible BDD configurations correlates to the counting of gate families based on only two possible gate identities. The first seven values of these three integer series are shown in Table 2.1.

Table 2.1: Integer series corresponding to 1) the δ_{1-1-g} circuit families, 2) the δ_{1-1-g} family with a 2-gate basis, and 3) the δ_{1-1-g} family with a 6-gate basis

g	1	2	3	4	5	6	7
A000366	1	2	7	38	295	3098	42271
A005439	2	8	56	608	9440	198272	5410688
A000366 * 6^{n-1}	6	72	1,512	49,248	2,293,920	144,540,288	11,833,174,656

2.5.5 Component Identification. Parham [35] described an attacker using an algorithm described by White [48]. This algorithm was created for the purpose of identifying modules within the context of a larger circuit, and avoids the *NP*-completeness problems associated with isomorphic subgraph identification [16] by focusing on the more restricted problem of identifying gate clusters (connected subgraphs composed of gates within a circuit) which satisfy three rules:

1. **Unique Enumeration** - No subgraph is identified more than once.
2. **Output only subcircuits** - All vertices within identified subgraphs are fully specified.
3. **Output only contained subcircuits** - All vertices within identified subgraphs are fully contained.

Upon enumerating a subcircuit, this algorithm compares the subcircuit to a library of known components to identify whether the candidate subcircuit has the same semantics (matching input/output count and the same truth table) as some library component. If the candidate matches a library component, then the library component was identified inside the circuit. Parham's implementation allowed for the relaxing of the second and third rules to allow for the identification of subgraphs that do not match White's definition of a subcircuit. The effect of relaxing these rules has not been experimentally verified.

The practical importance of this algorithm lies in its use to automate the individual tasks described by Hansen et al. [18] as necessary in the hardware reverse-engineering process. Rather than forcing a reverse-engineer to visually search a circuit schematic for components, this algorithm makes it possible to more efficiently and automatically identify library modules.

2.5.6 Boundary Blurring. Parham recognized that to defeat the component-identification adversary, a circuit obfuscation system needed to modify the semantics of circuit components. To do this, he developed two algorithms for logic modification which he deterministically applied at component boundaries. These algorithms, which Parham described as *blurring algorithms*, blur internal circuit logic. Definition 9 describes the blurring process.

Definition 9. ***n-level Blurring Algorithm (condensed):***

1. *A blurring algorithm selects a* ***replacement gate*** g_{rep}.

2. *The gate type of g_{rep} is changed, and at n levels closer to the output of the circuit, a* ***recovery gate*** *g_{rec} is selected.*

3. *New combinational logic is added to the circuit which computes the function originally computed by the recovery gate g_{rec}. The new gate which computes the function of g_{rec} is g_{new}.*

4. *g_{rec} is replaced in the circuit by g_{new}.*

The two blurring algorithms developed by Parham are *Multilevel blurrring* and *Don't-care blurring.* (For a more detailed description of both algorithms see [35].) These algorithms differ in the type of new combinational logic used to recover the original function of the recovery gate. While Multilevel blurring only generates new combinational logic using the original signals of the original and recovery gate, Don't-care blurring generates new logic which depends upon signals selected randomly from other locations in the circuit.

Blurring algorithms become *boundary blurring algorithms* when used deterministically to blur the logic of a component boundary. In these cases, the replacement and recovery gates are chosen to fall on either side of a component boundary. Parham reported perfect success in using Don't Care blurring to protect against component identification. His success was due to the fact that Don't-care blurring modifies number of inputs and the number of outputs of individual components in the circuit. Because the component identification algorithm requires that a subcircuit s exist with the same number of inputs and outputs as a known component c in order to identify s as an instance of c, the component identification algorithm will fail when the input or output count of a subcircuit has been modified.

2.6 Combinational logic represented as binary decision diagrams

Binary Decision Diagrams (BDDs) [24] are a type of directed acyclic graph structure for representing and manipulating Boolean functions consisting of two sink

nodes (representing the values of true and false) and many branch nodes, each with two links pointing to other nodes. Branch nodes represent a single variable v, and links emanating from branch nodes are `high` and `low`, respectively to indicate the path that will be evaluated if v evaluates to `true` or `false`, respectively. Each branch node n computes a truth table (described by Knuth as a string termed a `bead`) which represents the sink node (`true` or `false` reached by every possible combination of values that individual variables could evaluate to. One specific node, termed the *root*, indicates a specific branch node whose bead computes the function of the BDD.

Variants BDDs include ordered BDDs (in which nodes from the root to the sink follow some specific ordering based upon the variable they represent) and reduced BDDs (in which nodes that compute the same `bead` are combined). Further, shared BDDs allow for the possibility of multiple root nodes (that is, multiple functions computed) that share common branch nodes.

2.7 Background Summary

Reverse-engineering is a field in which the intent, purpose, or design of a system is recovered from the system itself, and a reverse-engineer seeks to achieve a level of understanding about the system itself based upon the system's behavior and internal structure. Several definitions for program understanding have been presented, and several positive and negative results in developing metrics for the field of program obfuscation have been reviewed. Cousot and Cousot's use of the abstract interpretation framework in program understanding has been surveyed, as has Dalla Preda and Giacobazzi's method based upon abstract interpretation for detecting opaque predicates. Lastly, results in circuit obfuscation have been reviewed. These included methods for mapping software to circuits, the subcircuit selection and replacement framework developed by Norman, metrics for evaluating circuit obfuscation, and the component identification and boundary blurring tools developed by Parham.

Using this background, the next chapter will seek to analyze the security offered by the subcircuit selection and replacement method and will seek to improve the security and efficiency of the algorithm proposed by Norman.

III. Methodology

As noted in the first chapter, this research seeks to provide better static circuit protection by proposing more secure and more efficient methods for white-box circuit obfuscation in order to improve the obfuscation of individual circuit variants. Better security and better efficiency are achieved by implementing deterministic methods that improve over the purely random methods designed to implement the Random Program Model.

Chapter I established that this thesis seeks to provide security improvements in two areas: functional and structural information hiding. Practically, these improvements are measured using the metrics of signal hiding and component hiding. The metric of signal hiding is first described in this thesis, and is detailed in Section 4.5; the metric of component hiding was first described by Parham, and is detailed in Section 2.5.5. Both new obfuscation algorithms described in this chapter were designed to maximize component hiding, and the second obfuscation algorithm is designed to deterministically achieve signal hiding.

The random SSR system mentioned in Chapter II suffers from limitations related to the cost associated with enumeration and indexing of large, equivalent circuit families in order to provide a theoretical measure of security. Parham's work in [35] is the first to propose a more efficient means for defeating component hiding than random SSR. Both new algorithms in this thesis use a circuit synthesizer to achieve faster generation times than random replacement used in the random SSR method.

A brief summary of this chapter is as follows:

Section 3.1 provides preliminary definitions for use in discussing random, deterministic, and efficient algorithms. Also, this section seeks to clarify terminology used by this thesis for describing characteristics of combinational logic circuits.

Section 3.3 proposes a new deterministic method for subcircuit selection-and-replacement. This method, *component fusion*, deterministically selects subcircuits which contain components. If none are available, then *component fusion* partitions

the circuit into internally connected subcircuits for replacement. Subcircuits are then forced to overlap by adding the predecessor gates of each subcircuit to the subcircuit itself. Deterministic replacement is accomplished using a circuit synthesizer built on top of the ESPRESSO two-level optimizer. This synthesizer randomly chooses whether to implement a product-of-sums-based logic form or a sum-of-products-based logic form, and also chooses one of four possible gate type configurations for circuit variants.

While component fusion guarantees that subcircuits overlap, component fusion does not provide a guarantee that the semantics of individual components will be protected. In order to provide this guarantee, Section 3.4 provides another deterministic method for whitebox circuit obfuscation - *component encryption.* Component encryption performs semantics-changing operations on a circuit, but preserves the semantics of the overall circuit. To protect component semantics a simple combinational logic encryption scheme, *Signal Value Permutation* (SVP), is provided to deterministically hide both the semantics of signals between components and the count of signals between components. SVP modifies the count and semantics of signals between internal components by introducing encoding logic where signals are produced and decoding logic where signals are used, effectively preserving the semantics of the circuit. Lastly, component encryption synthesizes encoding and decoding logic into the original circuit components to prevent an adversary from discovering the encoding and decoding logic.

The metrics used to evaluate component fusion and component encryption are examined in Chapter IV, and results from the evaluation will be presented in that chapter as well.

3.1 Definitions

For clarification, definitions are provided for key algorithm terms and key circuit vocabulary. Section 3.1.1 provides algorithm definitions, and Section 3.1.2 provides circuit definitions.

3.1.1 Algorithm definitions. To better describe circuit generation algorithms, the following terms are more precisely defined:

Definition 10. ***Random:*** *A random algorithm returns an output value such that every possible output value is equally statistically probable. [11]*

Definition 11. ***Deterministic:*** *An algorithm whose behavior can be completely predicted from [its] input. [3]*

Definition 12. ***Pseudorandom:*** *A deterministic algorithm returning output values that "look" statistically random. [11]*

From the definitions above, almost all algorithm implementations on physical systems can be classified as deterministic. If all inputs from the run of an algorithm are held constant, that algorithm should produce the same output every time. Many otherwise deterministic algorithms use random number generators implemented in high-level computing languages as a means of introducing randomness into their result. While producing as output sequences of numbers that appear random, most random number generators are actually deterministic because they rely on an initial seed value from which to generate a random number sequence; "random" numbers will be reproduced if any given seed value is reused. According to the National Institute of Standards and Technology (NIST), algorithms which use random numbers to produce random output are not usually considered deterministic, but algorithms which use numbers produced by a pseudo-random number generator may exhibit deterministic behavior [3].

In this thesis, the primary distinction will be drawn between pseudo-random algorithms (ones which would be completely random if given a source of randomly generated values) and deterministic algorithms (ones whose behavior would not be completely random even if given a source of randomly generated values). In other words, pseudo-random algorithms would become random algorithms if they were seeded with random values, and deterministic algorithms exhibit non-random prop-

erties even though they may have randomized routines and even when seeded with random seeds.

3.1.2 Circuit Definitions. For clarity, the following definitions are used in this thesis to describe the domain of logic circuits. Several of the sources for this thesis provide conflicting definitions for each of these terms, so defining these terms is necessary to avoid confusion.

Definition 13. ***Circuit:*** *A directed, acyclic graph $D(V, E)$ of inputs, logic gates, and outputs where 1) V describes a set of nodes (or vertices) representing logic gates and 2) E represents the set of edges representing wires connecting the output of some logic gate (or an input to the circuit) to the input of some other logic gate (or an output of the circuit).*

Definition 13 is essentially the same as that defined by Williams, [51] and corresponds to White's definition of a *circuit graph.* [48]

Definition 14. ***Subcircuit:*** *A subgraph of some circuit C.*

Definition 14 is most similar to that implied by Norman and James, but conflicts with the definition of a subcircuit provided by White. This definition is also similar to Williams' definition of a *component.*

Definition 15. ***Candidate Component:*** *A specified subcircuit which is fully specified, fully contained, or both.*

Definition 15 is used most often by Parham. This definition corresponds to White's definition of a *subcircuit.* The terms *fully specified* and *fully contained* are described in more detail by White in [48].

Definition 16. ***Component:*** *A subgraph with a specified input/output flow whose semantics, i.e., truth-table input/output mappings, are known. In addition, there is some notion of intent attached to a component, indicating some knowledge of the higher-level purpose of a subgraph with those semantics.*

A component as defined by Definition 16 is one of the best descriptions of what a reverse-engineer is searching for when attempting to identify the function of a circuit, but has not yet been proposed. This definition is implied by Parham, and conflicts with the definition of *component* provided by Williams. Several of the features that Hansen et al. [18] used as aids during the reverse-engineering of the ISCAS85 benchmark circuits (such as library modules or features present in the circuit topology) can be reduced to special cases of the component identification problem.

3.2 Pseudorandom Methods for Circuit Variation

Norman [33] and James [20] designed the subcircuit selection and replacement system for circuit transformation under the assumption that a random transformation system would offer the best circuit protection, and derived this assumption from the Random Program Model (RPM). The RPM [29] suggests that a program variant with the best possible white-box security will be indistinguishable from a random program. Applied to circuits, this would mean that a circuit variant will possess a randomized circuit structure that is the most likely to be indistinguishable from a circuit generated entirely at random. With the requirement that black-box properties be preserved, a circuit P will be replaced with some other circuit implementing the same function as P, as illustrated in Figure 3.1. Without the requirement that black-box properties be preserved, P will be replaced with some other circuit implementing an encrypted version of P (designated $E(P)$) and with randomized white-box structure.

The SSR system designed by Norman and James was intended to provide *semantics preserving* white-box circuit randomization. However, the circuit transformation system designed by Norman and James only approximates the generation of random equivalent circuit variants. Because the implementation suggested by James requires the enumeration of large subcircuit families, only very small replacement candidates (size measured by logic gate count, and corresponding to similarly small selection sizes) of a fixed size can be considered. Not all circuit implementations are reachable by small logic gate selection and replacements of a fixed size, making circuit variation

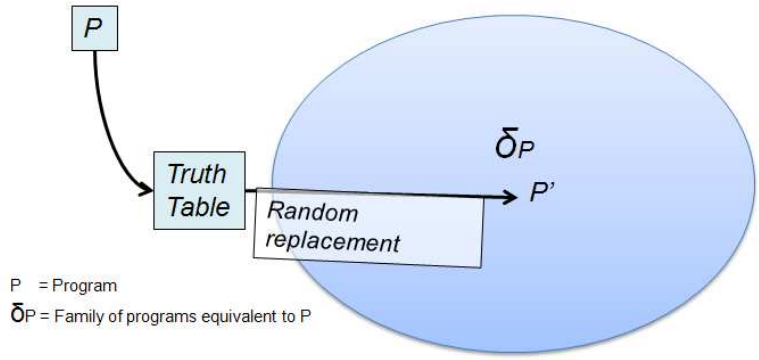


Figure 3.1: The Random Program Model: a semantics-preserving transformation on P replaces P with any other circuit in the family δ_P.

by SSR slightly deterministic. This hypothesis is examined in more depth in Section B.1.

Furthermore, random selection offers no guarantee that all portions of a circuit will be selected in any particular run of the variant generator, and random replacement offers no guarantee that the replacement of a circuit will exhibit hiding properties. The SSR described by Norman and James contains a strategy for randomizing individual subcircuits during the iterative subcircuit selection-and-replacement process, but not a strategy for obfuscating a complete circuit.

While the security of a random replacement can be argued by an appeal to the RPM (i.e., a randomly generated equivalent circuit variant is as indistinguishable from a randomly-generated circuit as possible), generating and querying replacement libraries was empirically shown to be intractable by Simonaire [39]. Simonaire characterized the growth of circuit families as *hyper-exponential*. In the absence of a method for randomly choosing circuit variants without enumerating all possible candidates, fully random replacement allowing potential replacement from any variant of a given size may only be performed on candidates within a bounded size.

Prior to this research there has been no investigation of whether random subcircuit selection-and-replacement actually achieves the security that the Random Pro-

gram Model proposes. An elementary examination of the capabilities of random SSR is provided in Appendix B.

Overall, the security and efficiency limitations of SSR motivate the construction of the *component fusion* and *component encryption* methods in Sections 3.3 and 3.4, respectively.

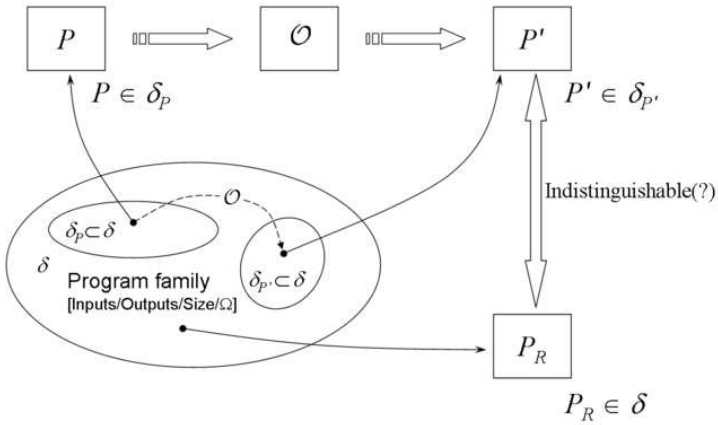


Figure 3.2: The Random Program Model: Program variant production through selection from all circuits in a family $\delta_{i-o-s-\Omega}$ [33].

3.3 Component Fusion

The first new method for obfuscating logic circuits, *component fusion*, improves on both the efficiency and the measurable security offered by random subcircuit selection-and-replacement by abandoning the requirement that all possible subcircuits be considered for selection, and also by abandoning the requirement that all possible replacements for a subcircuit be considered. Instead of random selection, a component-identification-based method for deterministic selection is used. Combined with a method for selecting portions of neighboring components, this selection strategy yields security by redefining component boundaries.

Component-based selection selects subcircuits too large to replace with a fully enumerated circuit library. To address this, a new, efficient method for subcircuit

replacement is proposed. This replacement method obtains a normalized form of the subcircuit, removing white-box structural information.

As Figure 3.3 shows, the concept behind component fusion is to avoid requiring random iteration to move from a circuit to a good variant. The process will still use iteration, but not as the progress metric for measuring obfuscation. The circuit P will be efficiently replaced with a circuit P' from a set of circuits implementing the same function as P, but with good function hiding properties. Component fusion will still

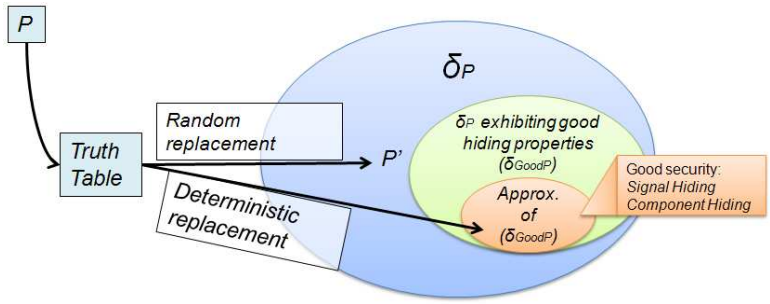


Figure 3.3: Using a deterministic method, replacements for P come only from δ_{GoodP}, but the deterministic method can produce only a subset of replacements in δ_{GoodP}.

involve subcircuit-selection-and-replacement, and will be based upon a deterministic selection routine using component identification and deterministic replacement using a minimizer to obtain the normal form of the selection. The algorithm for component fusion is diagrammed in Figure 3.4

3.3.1 Deterministic Selection. The new algorithm for identifying subcircuit selections is based upon identifying small subcircuits which require protection. These may include either components identified by a component identifier, components already known to exist in the circuit, or arbitrary partitions of the circuit.

As shown in Figure 3.5, prior to running a variation routine the obfuscator will search the circuit for any identifiable components using White's component identification algorithm. If any can be identified, then these components will be the first

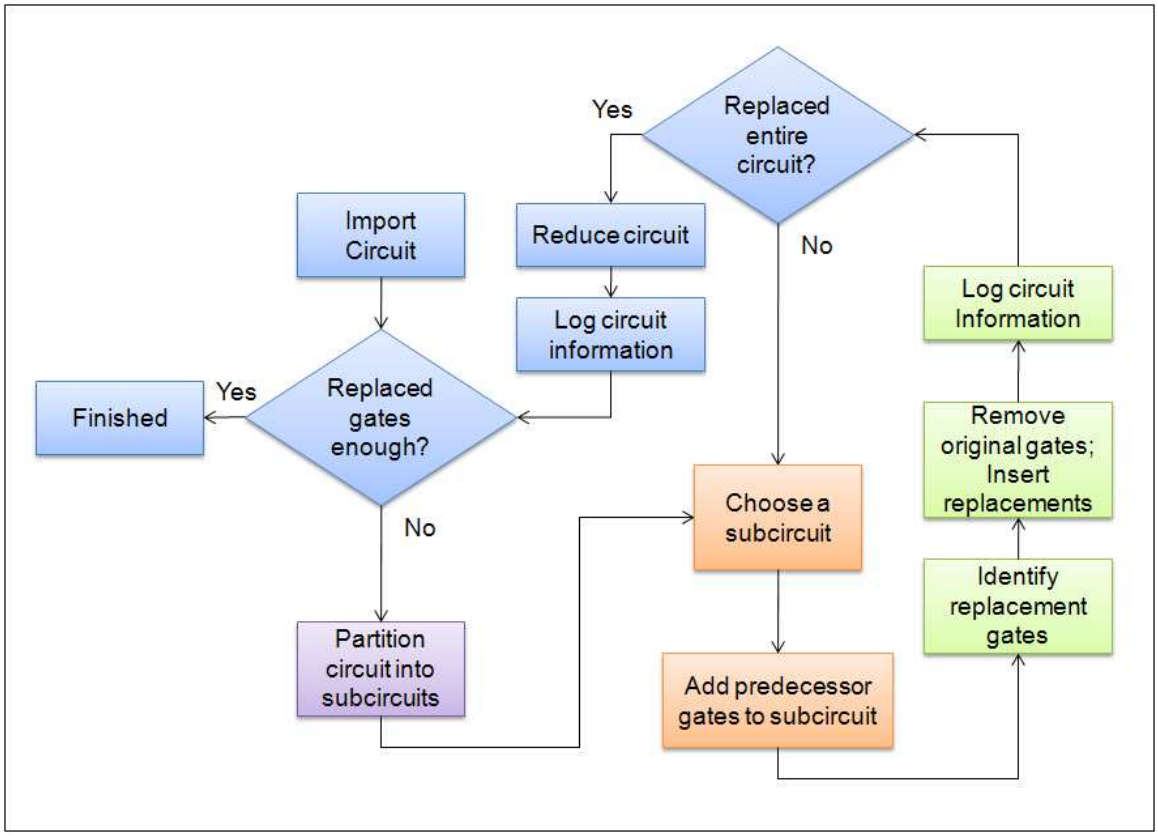


Figure 3.4: The process flow of component fusion.

subcircuits selected for replacement. Identified components are placed in a queue to be selected and replaced.

Next, the circuit variant generator identifies gates in the circuit that do not yet belong to a component and partitions them into connected subcircuits with a number of inputs small enough that the truth table can be enumerated in reasonable time. Once the circuit is divided into subcircuits, the algorithm begins selection and replacement. By partitioning the circuit into subcircuits prior to obfuscation, the circuit obfuscation algorithm guarantees that every gate in the circuit will be replaced, a first step towards ensuring that topology will be obfuscated.

Figure 3.6 depicts the selection process. During each run of the variation algorithm, a subcircuit will be removed from the queue. All gates preceding this subcircuit will be selected and added to the subcircuit. Then, the new subcircuit will be given

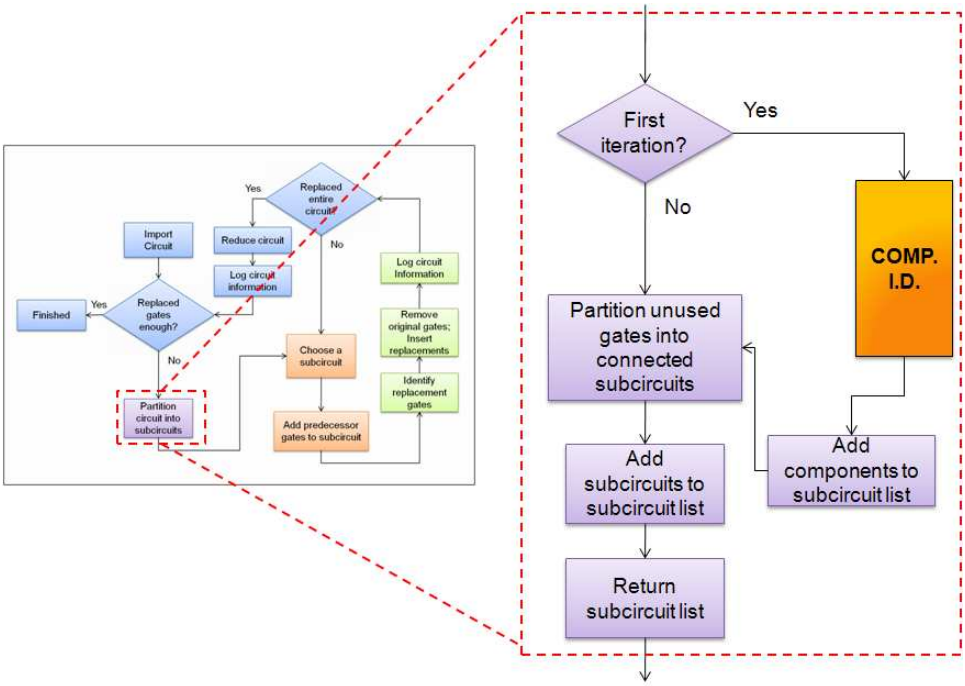


Figure 3.5: The process flow for circuit partitioning in component fusion.

to the replacement algorithm. By adding predecessor gates to each subcircuit prior to replacement, deterministic selection ensures that selection-and replacement operations will overlap. This ensures that signals bordering components will be replaced at some time during obfuscation, helping to prevent component identification.

3.3.2 Deterministic Replacement. The algorithm for producing circuit variants consists of coupling variation with a circuit normalizer. This algorithm is depicted in Figure 3.7. For this research, the ESPRESSO-II logic minimizer developed by Berkeley [5] was chosen because both the ESPRESSO-II algorithm and original ESPRESSO-II source code were available from Berkeley. Surveying known logic minimization algorithms (such as Quine-McCluskey, PRESTO, OPTIMA, [6] or Latte [26]) was considered to be beyond the scope of this thesis because any other two-level logic minimizer can easily replace ESPRESSO in the deterministic replacement strategy described in this thesis, and most logic minimizers will exhibit performance benefits on a similar order relative to circuit family enumeration.

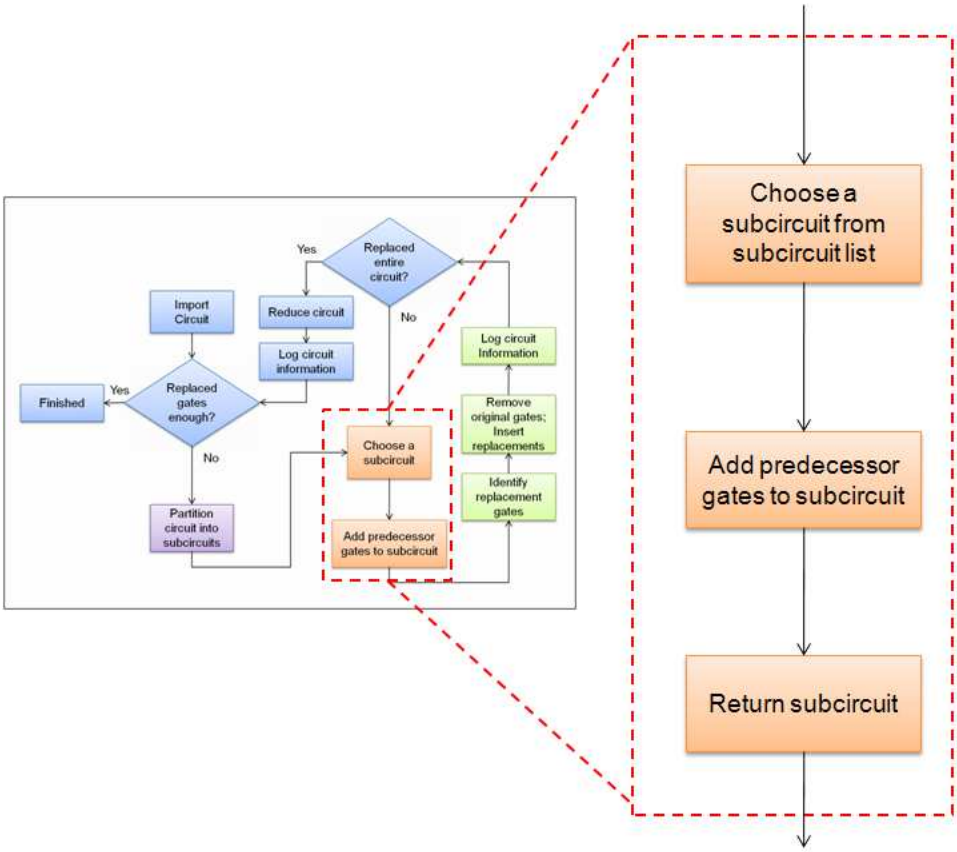


Figure 3.6: The process flow for subcircuit selection in component fusion.

3.3.2.1 Espresso heuristic minimization . The Espresso algorithm is designed to manipulate programmable logic array (PLA) representations of logic circuits. A PLA is used to implement two-level combinational logic through adjacent pairs of rectangular arrays of logic gates. PLAs are easily represented using a pair of matrices, an input array and corresponding output array which specify the input signals that cause certain circuit outputs. An example truth-table and minimized PLA which describes the outputs of the ISCAS benchmark c17 logic circuit are shown in Figure 3.8.

The Espresso algorithm is used without modification in deterministic replacement. For background, eight core functions are used in the computation of the Espresso algorithm: [5]

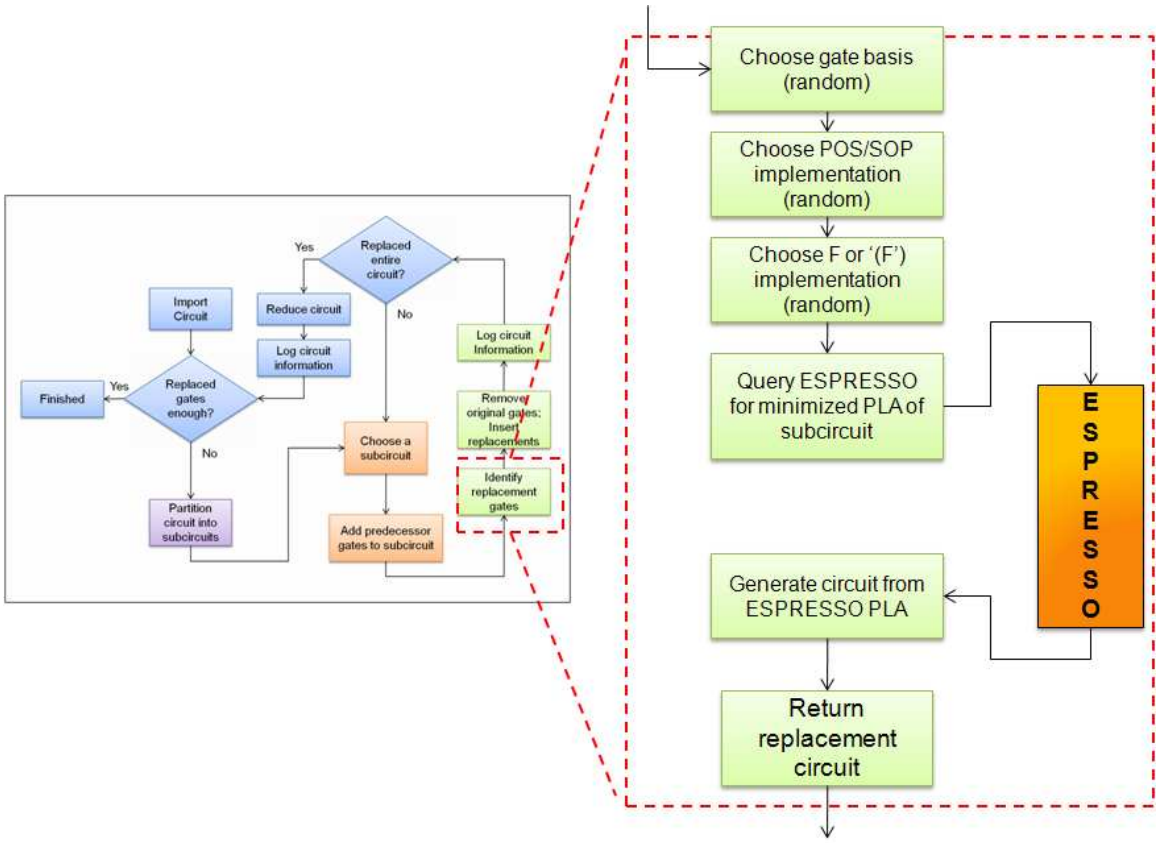


Figure 3.7: Deterministic replacement through the use of chosen minimized forms.

1. Complement (Compute the complement of the PLA and the don't-care set, i.e., compute the off-set)
2. Expand (Expand each implicant into a prime and remove covered implicants.)
3. Essential Primes (Extract the essential primes and put them in the don't-care set).
4. Irredundant Cover (Find a minimal (optionally minimum) irredundant cover).
5. Reduce (Reduce each implicant to a minimum essential implicant).
6. Iterate 2,4, and 5 until no improvement.
7. Lastgasp (Try reduce, expand, and irredundant cover one last time using a different strategy. If successful, continue the iteration).

```
.i 5
.o 2
.ilb 1 2 3 6 7
.ob 22 23
00000 00
00001 01
00010 00
00011 01
00100 00
00101 01
00110 00
00111 00
01000 11
01001 11
01010 11
01011 11
01100 11
01101 11
01110 00
01111 00
10000 00
10001 01
10010 00
10011 01
10100 10
10101 11
10110 10
10111 10
11000 11
11001 11
11010 11
11011 11
11100 11
11101 11
11110 10
11111 10
.e
```

```

F:\Pen-Drive>espresso.exe c17-tt.txt
.i 5
.o 2
.ilb 1 2 3 6 7
.ob 22 23
.p 5
1-1-- 10
--0-1 01
---01 01
-10-- 11
-1-0- 11
.e

```

Figure 3.8: (From left to right) The unminimized PLA of the c17 ISCAS-85 benchmark circuit and the Espresso minimized PLA of the same circuit.

8. Makesparse (Include the essential primes back into the cover and make the PLA structure as sparse as possible).

3.3.2.2 Variation of two-level forms. The PLAs returned by the Espresso algorithm are natively in sum of products form (also described as *disjunctive normal form* in Boolean algebra or *OR-AND* form in circuit form, indicating that the implementation is an OR of ANDs). However, other normal forms are also achievable. The same PLA may be implemented in product of sums form (conjunctive normal form or AND-OR form) and compute the same function. In addition, variation can be introduced by implementing circuits using NAND or NOR gates (by DeMorgan's theorem, OR-AND form is equivalent to NAND-NAND form, and AND-OR form is equivalent to NOR-NOR form). Lastly, it is possible to implement the negation of a function and invert the output, yielding an additional four variants. All in all, there are eight possible variants derivable from two-level logic. The replacement algorithm, Algorithm III.1, is diagrammed in Figure 3.9. Two-level optimizations of the PLA form may implement functions using gates of each fundamental type, (AND, OR, NAND, NOR) and these gates may have an arbitrary number of inputs. It is uncommon to find a physical implementation of a nine-input AND gate, but it is

Algorithm III.1 CHOOSE-CANONICAL-FORM(Boolean negate, Boolean sum-of-products, Boolean apply-demorgans)

```
if !negate then {Implement function F}
  if sum-of-products then {Implement sum-of-products form}
    if apply-demorgans then
      return  NAND of NANDs form
    else
      return  OR of ANDs form
    end if
  else {Implement product-of-sums form}
    if apply-demorgans then
      return  NOR of NORs form
    else
      return  AND of ORs form
    end if
  end if
else {Implement the negated function F', and then invert it at each output.}
  if sum-of-products then {Implement negated sum-of-products form}
    if apply-demorgans then
      return  AND of NANDs form
    else
      return  NOR of ANDs form
    end if
  else {Implement product-of-sums form}
    if apply-demorgans then
      return  OR of NORs form
    else
      return  NAND of ORs form
    end if
  end if
end if
```

possible that a nine-input PLA would specify an implementation requiring such a gate. Decomposing these gates to only require two gates (through application of the circuit form of the associativity rule) may yield clusters of gates of the same type. An example is shown in Figure 3.10. In the cases where more than two input gates are used, this type of gate clustering would be less of a problem.

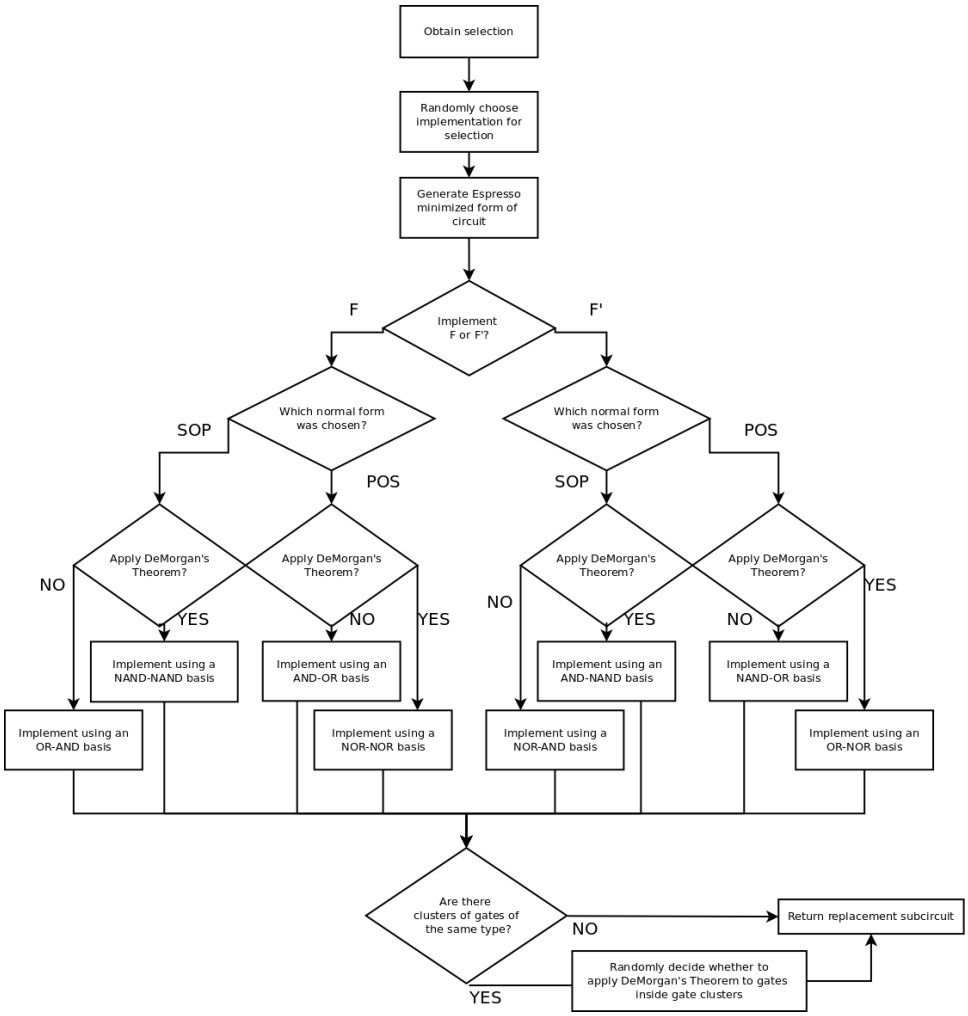


Figure 3.9: Deterministic replacement through the use of chosen minimized forms.

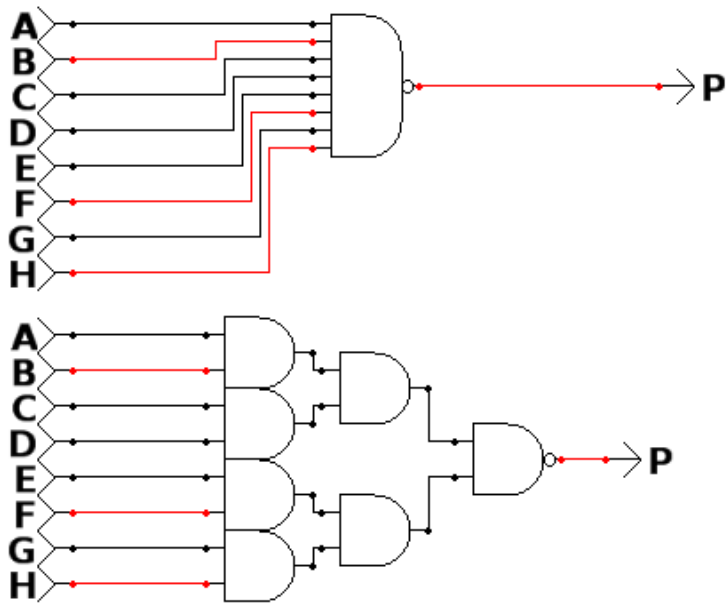


Figure 3.10: Clustering of AND gates resulting from the decomposition of an 8-input NAND gate.

3.4 Component Encryption

In considering the goals of a reverse-engineering adversary, several educated assumptions are made about the strategies used by the adversary to recover the intent of a circuit. These assumptions are based upon measures of information leakage.

What information will be leaked by a combinational logic circuit c? An adversary can monitor the signals being used as input by c and the signals being produced as output by c. Furthermore, it is assumed that the adversary will be able to apply values to the inputs of c and will be able to monitor c's outputs (i.e., the adversary will be given oracle access to the c). This means that the adversary will be able to perform black-box analysis on c. However, it is assumed that the number of inputs to c will be large enough that enumerating the truth table of c will be infeasible. This research will assume, as Parham assumed, that the infeasibility of black-box analysis will drive an adversary to white-box means for identifying the intent of c.

What white-box information, then, will be leaked by a combinational logic circuit? Most combinational logic circuits c can be decomposed some number of smaller

combinational subcircuits $s_1, s_2, \ldots s_n$ used individually to compute functions because their individual functions can be used to compute a larger function. Any subcircuit s_i is said to have a particular semantics which defines its function, and the semantics of s_i can be broken down into individual characteristics (the number of s_i's inputs, the number of s_i's outputs, and s_i's truth table). Further, subcircuits have topology characteristics related to the manner in which they are implemented, including the type and number of gates in the subcircuit, and how they are connected.

Some subcircuits are well-known in the computing industry because they compute a function that is useful and are used frequently. These types of circuits have been referred to in this research as *components*. Such subcircuits are often implemented using the same gate structures. It is also possible that equivalent circuits that compute the same functions will be used.

Parham reasons that the primary goal of a reverse-engineer will be component identification. This is defined as (firstly) finding some subcircuit s_{ident} with the same input/output flow as a known circuit s_{known} , and (secondly) validating that the semantics of s_{known} and s_{ident} are equivalent. Once the reverse-engineering adversary has done this, the next step will be to identify the intent of c from the relationships between the subcircuits of c.

To summarize, it is assumed that an adversary driven to white-box analysis will attempt to use information of any type available in the process of recovering the intent of c, and that the ability to identify components and the communications between them is the way that an attacker will be able to attain a higher-level abstraction of a circuit. For this reason, this research seeks to both modify the semantics of individual components and to modify the signals that carry communications between the components, with the additional goal of modifying the internal structure of components to prevent topology-based identification. Parham's research on boundary blurring techniques also had these goals. Parham succeeded in defeating a component identification adversary based upon a component identification algorithm developed

by White, [48] [50] but no framework existed prior to this research to evaluate the security of components which cannot be identified using White's algorithm.

A new, alternative component protection system will be termed Signal Value Permutation (SVP). SVP is a simple encoding scheme which modifies the semantics of components in a way that renders the semantics of individual components sufficiently different to cause a component identification algorithm to fail, except in the case of isolated components (described by Parham as case I components). Case I components receive all of their inputs from the circuit inputs, and every output of a Case I component is a circuit output.

SVP consists of five steps: Identifying signals which are common to a set of components, generating a map for encoding these signals, applying the encoding to the component truth tables, generating new components based upon the new component truth tables, and connecting the new components together in a way that preserves the original component communications.

In comparison, component fusion guarantees that the boundaries of identified components will be contained within some replacement. However, at a fundamental level, component fusion is still a semantics-preserving selection-and-replacement methodology, so after every selection-and-replacement, the input and output signals of the selection contain the same semantics as before the replacement. Overlapping replacements are the only way that a semantics-preserving selection-and-replacement algorithm can hide all of the signals internal to a circuit. What component fusion provides (that random subcircuit selection-and-replacement does not) is a guarantee that every replacement will overlap.

Though overlapping selection-and-replacement operations is the only way that all of the signals can be hidden, deterministically overlapping these operations does not inherently provide a guarantee that a signal will be hidden. A poorly picked replacement may still internally contain a signal that the original circuit contained.

Signal Value Permutation aims to provide a better security guarantee, by preserving the semantics of an original circuit, but purposefully changing the topology and semantics of each identified component in the circuit to defeat component identification methods. The encoding function detailed in section 3.4.2) is applied to the output of each component to encode the component's output, and the decoding function detailed in Section 3.4.4 is applied to the input of each component to decode the input.

Figure 3.11 illustrates the process of component encryption. Notional encryption and decryption functions are inserted on the signals between components A and B, and then component A', containing both the logic of A and E_k, is synthesized to absorb the encryption logic. Similarly, the component B', containing both the logic of B and D_k, is synthesized to absorb the decryption logic. The new notional component boundary between A' and B' carries an encrypted signal and has a different number of signals than the original boundary.

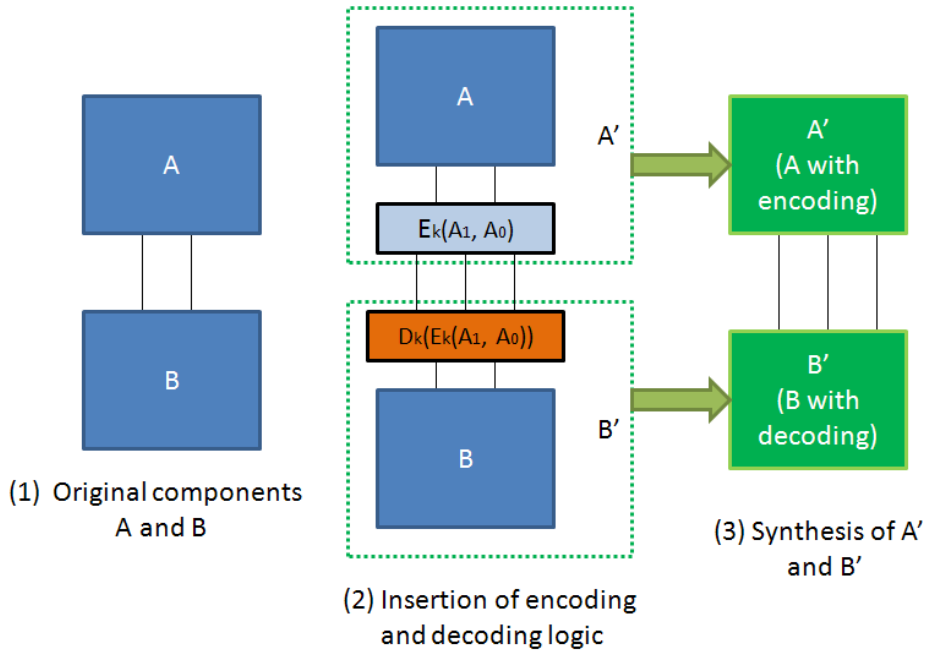


Figure 3.11: Encryption of signals between components A and B: (1) The original configuration, with signals from A feeding B; (2) insertion of encryption and decryption functions, and (3) synthesis to combine A with E and D with B.

3.4.1 Generating Signal Value Mappings for Encoding/Decoding. Figure 3.12 shows the generation of the mapping table which remaps all possible values of the original 2 signals to new values in a set of 3 signals. This mapping table is then used in the generation of encoding and decoding logic. There are many ways of generating a mapping table of this sort; it is possible to map the original two signals to a new set of three signals by generating a one-to-one mapping between the original truth table rows and some of the the new truth table rows as shown in Figure 3.12 (a.), but this leaves some of the truth table rows of the new signals unused. If the old signals are mapped to new signals in this manner, it would be possible to tell by truth-table analysis that the new three signals will only ever carry four possible values. See section 3.4.5 for more information on information leakage.

On the other hand, a one-to-many mapping is also possible. In this mapping, there is a one-to-many relationship between the old and new signal values. In case (a.), inputs that caused the the wires A and B to carry the value `11` will cause the new wires A, B, and C to carry the value `000`, but in case (b.), the inputs that cause A and B to carry `11` will cause A, B, and C to carry a value which is either `000`, `100`, or `101`. Sections 3.4.2 and 3.4.4 describe in detail how this mapping is applied.

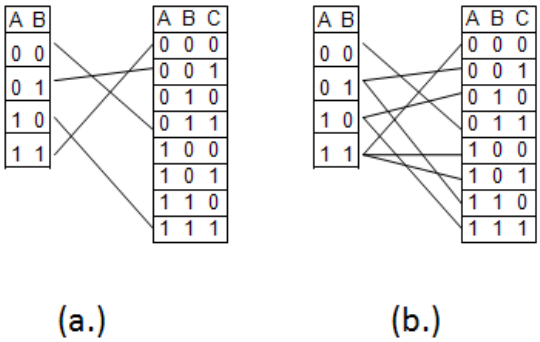


Figure 3.12: Mapping the possible values of two signals to the possible values of three signals (a.) in a one-to-one manner and (b.) in a one-to-many manner.

3.4.2 Encoding Outputs. Applying encoding to the outputs of a component is accomplished by replacing each occurrence of an output value with its counterpart in the mapping table. It is possible that an output value may be mapped to more than one new output value, as noted in the previous section. It should be noted that it is possible that not all output values are produced by a given component, e.g. if a component produces a constant value, because in that case only that constant will be produced.

In the encryption scheme defined here, for every output value of the original signals, a new output value is randomly chosen from the values that the original output maps to, and this new output value replaces the original output value in the truth table.

Figures 3.14 (a.) illustrates a one-to-two mapping used to hide the signal `Out1` which is produced as output by a component and also used as input by a component. The truth tables of both components are depicted in (b.). In (c.), two possible encryptions of the output `Out1` are depicted. Note that because there are only two possible mappings for the `Out1` value `0` but there are three occurrences of the value `0`, the replacement `00` is used twice. Similarly, note that two possible replacements are possible for the value `1`, but only one is used because the value `1` occurs only once in the output `Out1`.

3.4.3 Input Decoding. In applying the decryption function to the truth table of a circuit, the number of rows in the truth table is usually increased. The decryption function modifies the truth table by identifying all inputs which produce a certain output and creating rows in the truth table which equate the input values to their corresponding output values. The modification of a truth table to incorporate a decryption function is shown in Figure 3.14 part (c.) and the algorithm used to accomplish this mapping is provided as Algorithm C.2. The mapping depicted in 3.14 (a.) is used to modify the input `Out1` to create a truth table which produces correct output from encrypted input.

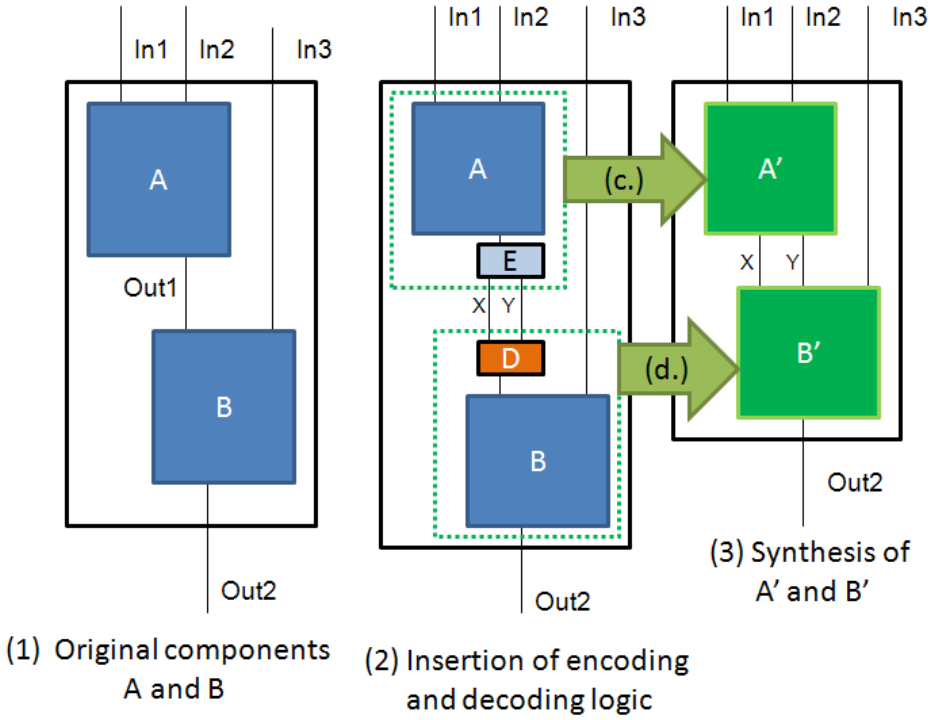


Figure 3.13: (1) The original components A and B, (2) components A and B with encoding and decoding logic to replace signal `Out1`, and (3) new components A' and B' which contain the encoding and decoding logic, respectively. Note that transformations (c.) and (d.), detailing the combination of components with encoding and decoding logic, are depicted in more detail in Figure 3.14

3.4.4 Identifying Signals for Encoding. Only signals between components can be encrypted. This means that all of the signals $s_1, s_2, \ldots s_n$ which are the output of a single component and every component which takes $s_1, s_2, \ldots s_n$ as input takes *all* of $s_1, s_2, \ldots s_n$. If not all of the signals in a group are used by every component, then the group of signals is reduced in size until every component that uses any of its signals uses all of them. Once the group is of size 1, this is always true.

Figure 3.15 illustrates the encryption of a signal group - an encryption function is used to encrypt the signals exiting component A. Note that in this case the decryption function D which is absorbed into C is the same D which is absorbed into B. Figure 3.16 shows a different configuration in which there are two groups of signals exiting

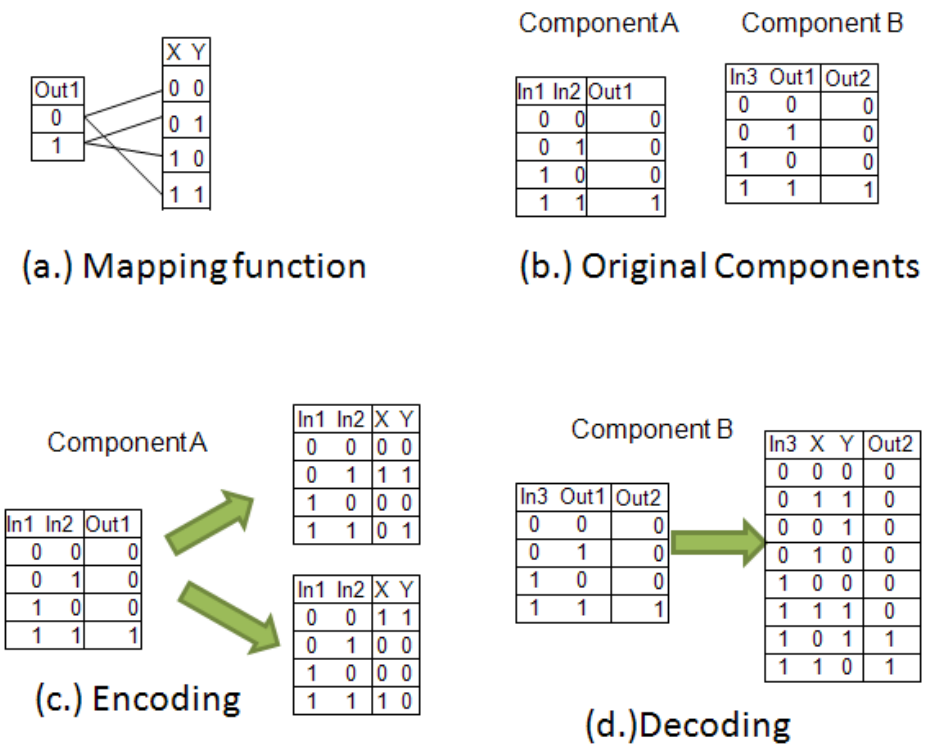


Figure 3.14: The mapping table (a.) is applied to the signal `Out1` (depicted as both an input and an output in (b.)). In (c.), enccoding logic`Out1` is an output which is encoded, and in (d.) `Out1` is an input which is decoded. Figure 3.13 graphically depicts these operations.

component A which are encrypted separately, and the decryption functions for the signals entering components B and C are different.

In order to preserve the semantics of the circuit boundary (but hide the signals of individual components), no encryption function is applied to signals exiting the circuit, and no decryption function is applied to inputs entering the circuit.

3.4.5 Attacks on Signal Value Permutation. Perhaps the most obvious cryptanalytic attack on this component manipulation system would be to examine the components closest to the inputs of the circuit boundary. If an attacker had foreknowledge of what type of components to look for (including an input count and the output function), he could attempt to identify these components by identifying

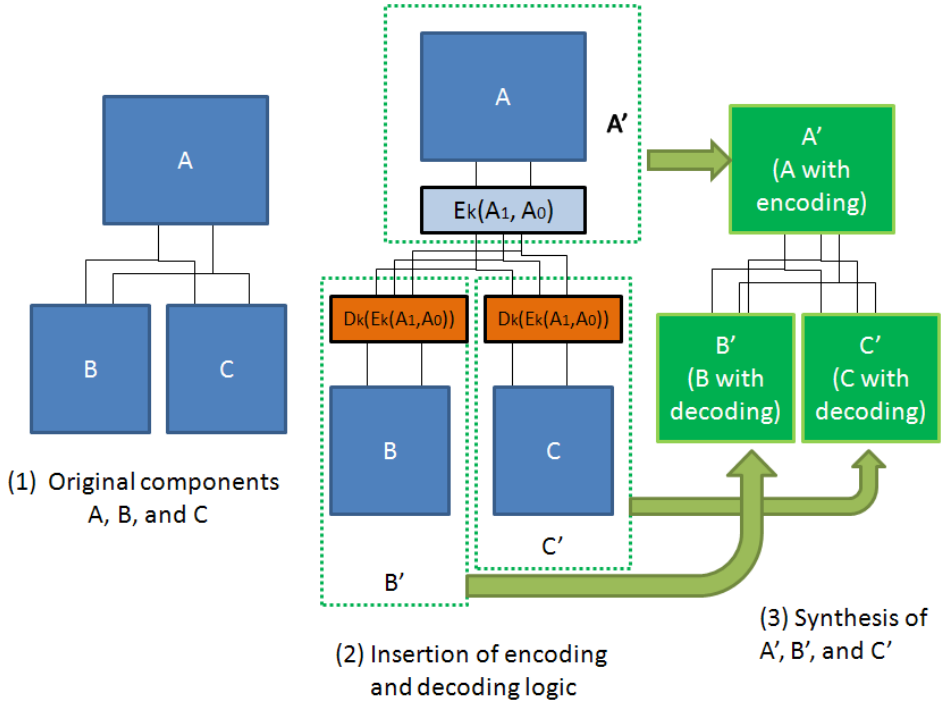


Figure 3.15: In (1), B and C share inputs from A; in (2), an encryption function is used to encrypt the output of A, and the same D decrypts both B and C. In (3), the encryption and decryption functions are synthesized into A, B, and C.

subcircuits with an input count equal to that of the component he is searching for. However, the attacker must guess the way in which the encrypted output of the component matches the original output of the component.

In addition to this, the attacker must guess the correct ordering of the component inputs. The probability of selecting the mapping which is used to translate the original circuit output to the new set of possible outputs, using only random guessing, is easy to identify for encryption which uses a one-to-one map like the one shown in Figure 3.12 (a.). A one-to-one mapping is simply a permutation of the input rows, and the number of possible permutations of an i input truth table is

$$\left(\frac{(2^i)!}{1}\right)$$

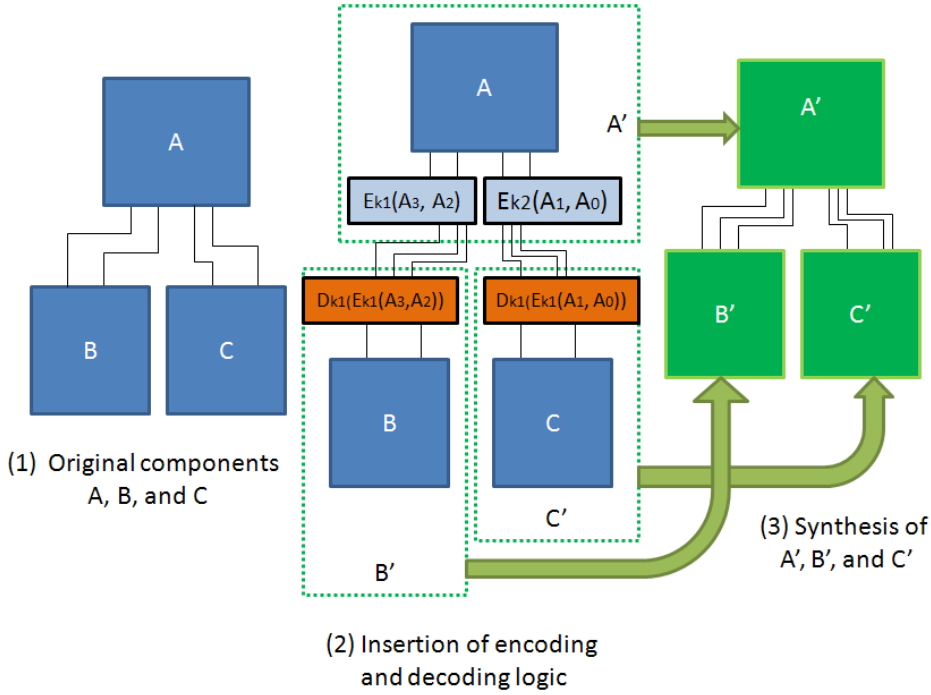


Figure 3.16: In (1), B and C share different inputs from A. In (2), different encryption and decryption functions are used to protect signals between A and B and between A and C, and in (3) these functions are synthesized into A, B, and C.

The number of possible one-to-many mappings, like the one shown in Figure 3.12 (b.), is much larger. Finding the number of one-to-many mappings (without respect to the original signal value) is easily modeled as the problem of partitioning a set of n elements (the new signal values) into m nonempty subsets. The series of the Stirling numbers of the second kind computes the number of possible partitionings of this type:

$$\mathcal{S}_n^m = \frac{1}{m!}\sum_{k=0}^{m}(-1)^{m-k}\begin{pmatrix} m \\ k \end{pmatrix} k^n$$

To compute the number of possible ways that the new truth table can be partitioned into $2^{s_{orig}}$ subsets (one for every row of the original truth table), every instance of m in the formula for Stirling numbers is replaced by $2^{s_{orig}}$, and every instance of n is replaced by $2^{s_{new}}$, where $s_{orig} \leq s_{new}$, s_{orig} represents the number of signals in the original truth table, and s_{new} represents the number of signals in the new truth table.

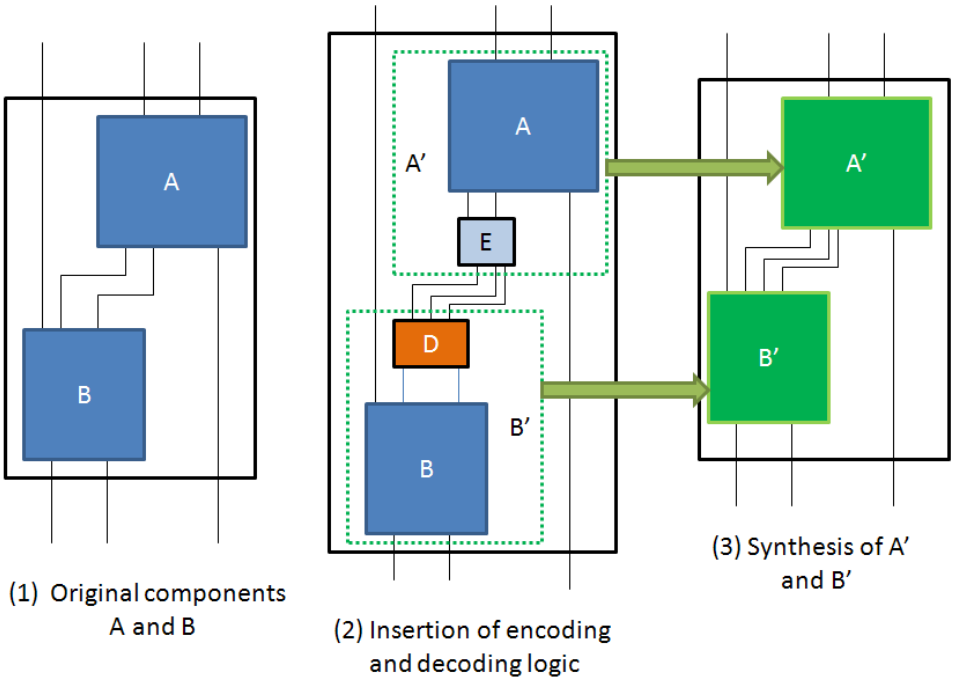


Figure 3.17: An illustration of semantic preservation of inputs and outputs that cross the circuit boundary. In (1), an input to B and an output of A cross the circuit boundary. In (2), these signals are preserved semantically, so (3) shows encrypted components which still use some unencrypted input signals and unencrypted output signals.

This yields the formula

$$Partitionings(s_{orig}, s_{new}) = \frac{1}{(2^{s_{orig}})!} \sum_{k=0}^{2^{s_{orig}}} (-1)^{2^{s_{orig}}-k} \begin{pmatrix} 2^{s_{orig}} \\ k \end{pmatrix} k^{2^{s_{orig}}}$$

There are $2^{s_{orig}}!$ ways that the original truth table rows can be assigned to $2^{i_{orig}}$ subsets, so the number of possible mappings is

$$Mappings(s_{orig}, s_{new}) = 2^{s_{orig}}! * Partitionings(s_{orig}, s_{new}) =$$

$$\sum_{k=0}^{2^{s_{orig}}} (-1)^{2^{s_{orig}}-k} \begin{pmatrix} 2^{s_{orig}} \\ k \end{pmatrix} k^{2^{s_{orig}}}$$

Thus the number of ways to map the possible values on a set of s_{orig} signals onto the set of values that could appear on a set of s_{new} signals is counted by $Mappings(s_{orig}, s_{new})$. It is fairly easy to evaluate the correctness of this formula for the the number of mappings of one signal onto two signals; 14 such mappings exist, each of which can be easily generated by hand. However, there are 40824 mappings of two signals onto three signals, and there are 86355926616960 mappings of three signals onto four signals.

At this point, it becomes necessary to consider what an adversary would do if no subcircuit with an input/output size matching that of a known component existed in the circuit. If this occurred, the next logical step for an adversary would be to find a component with input flow matching that of a known component (but with a different output flow) and attempt to find a set of outputs as large as the number of outputs on the known component for which all produced values are the same as those of the known component, or perhaps the same as those of the known component, but inverted. The adversary might try the same thing by finding a component whose output flow matches that of the original but whose input flow is different, and see whether applying a permuted set of signals or inverted signals on the inputs will produce the original outputs values.

What this means is that there will be some mappings that will not effectively prevent an adversary from identifying a component. These 'bad' mappings will be mappings in which the original truth table is either present or is present with any number of columns inverted. The number of possible 'bad' mappings can be seen by examination. For every s_{orig}-permutation of the s_{new} columns, there is some mapping which results in the old signal values being mapped to new signal values which contain exactly the same values on s_{orig} columns, and there are $2^{s_{orig}}$ mappings which contain some combination of the old signals inverted (but contain the old signals nonetheless). The number of bad mappings, then is

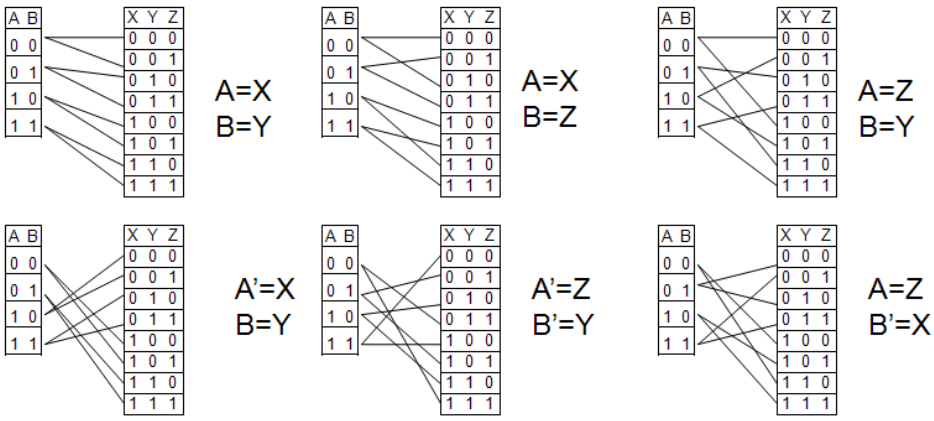


Figure 3.18: A set of 2-to-3 signal mappings which all contain the original component signals

$$BadMaps(s_{new}, s_{orig}) = 2^{s_{orig}} \frac{s_{new}!}{(s_{new} - s_{orig})!}$$

If any of these bad mappings were applied, an adversary would have an advantage because the signals of the original component would still produced by the circuit. Several examples of a bad mapping between two signals and three signals are shown in Figure 3.18. In each mapping, it can be clearly shown that the original signals are always equivalent to some signal in the new signal set, and in a combinational circuit, these signals would be easily identifiable as the original outputs of a component.

Avoiding these mappings is expensive, depending upon the number of old and new signals. Because generation of mappings is random, new mappings can be generated efficiently given a source of randomness, even though the number of mappings increases intractably on the number of signals in the mapping. However, the number of bad mappings grows exponentially on the number of old signals and factorially on the number of new signals, and each bad mapping must be checked for individually. If an 8-bit bus were encrypted into 9 bits, then the number of permutations would be $(2^8 * 9!) = 92897280$ permutations. (To put this into perspective, there are $1.6005 * 10^{1214}$ possible encodings of all 8-bit values into all 9-bit values.) In the example of the 8-signal to 9-signal mapping, work required by the mapping generator

is 92897280 checks for bad mappings, but the adversary must consider a space of $1.6005 * 10^{1214}$ possible encodings.

Storing mappings so that they can be applied to all components which use the original signals requires space that increases on the order of the size of the table of new signal values (it increases with respect to $2^{s_{new}}$). The number of mappings will increase linearly with the number of component outputs whose signals do not exit the circuit.

3.4.5.1 Switching Activity. Circuits utilizing encryption of signals between components will still be vulnerable to side-channel analysis. Because of the use of one-to-many mappings, it is possible that the switching activity of component outputs could be observed. If a certain set of inputs was observed to switch very often, information could be gleaned about the importance of the component. On average, though, encrypting outputs in the manner described in this section will increase the amount of switching behavior because a circuit which produces the same output for different input combinations has the potential to change if the inputs change. In Figure 3.14 part (c.), the signals X and Y can carry either 00 or 11 when the signal $Out1$ would originally have only carried 0 for the inputs 00, 01, and 10.

In practice, not all of possible output values are produced by any given circuit. For example, a 4-bit multiplier circuit multiplies two 4-bit numbers together and produces a product of eight bits. This circuit will not produce as output any prime number greater than four bits, and there are 48 such primes. Out of 256 possible outputs, 48 will be unused. An 4-bit adder, on the other hand, does produce all possible outputs. In these cases, it will be possible for an adversary to tell that some signal value is repeated within the set of possible signals.

Another example is an eight-input one-output `AND` gate, which produces the value 0 for 255 possible input combinations, and produces the value 1 for a single input combination. If the single output of an 8-input `AND` gate was mapped to two signals, there could be 1, 2, or 3 replacements for the value 0. If the value 0 has only

one possible replacement, then the new component will still produce the same value for 255 input combinations. This will leak a large amount of information. For an 8-input AND gate, the output 0 having three possible replacements is the ideal case because it will most reduce the amount of repetition in the possible output values.

It should be noted that if a mapping maps the original value of a set of outputs to multiple new values, a new method for adding inputs to a component is possible. As long as the new inputs only cause variation between values which the mapping designates as equivalent, it is possible to add inputs into components (either from some other random signal in the circuit or even a state machine which regularly produces pseudo-random output) which will serve to increase the switching activity of the component's outputs.

3.4.6 Conclusions Regarding Signal Value Permutation. In conclusion, a new methodology has been proposed for protecting signals between components inside of a circuit which would offer an increase in the difficulty of intent recovery. While the class of insecure SVP grows intractably, the number of possible ways that a set of signals can be protected also grows intractably, and to a much greater degree. Non-semantics-preserving operations applied to circuit components will serve to make component identification more difficult because part of the information necessary for the computation of a component's function will now be held by that component's successor components.

3.5 Summary

In summary, in this chapter two new, clearly deterministic methods have been proposed, and these methods are named *component fusion* and *component encryption*. Component fusion involves partitioning the circuit into components and taking measures to hide these components. Initial selections consist of limited-size components already known to exist in the circuit (thus measurably hiding known whitebox information), and later selections merely guarantee uniform selection-and-replacement

across the circuit. Replacements consist of random normal forms which can be generated efficiently and placed into the circuit. Chapter 4 will focus on the results of implementing this deterministic method and applying it to several test cases.

Also, a method named *component encryption* has been proposed. This method deterministically protects signals between components using a component signal encryption scheme to hide the outputs of certain components. This scheme, named *Signal Value Permutation*, offers measurable security by potentially changing both the number of signals between components and the semantics of the gates producing these signals. In Chapter IV, both *component fusion* and *component encryption* are measured against previous white-box circuit obfuscation algorithms in terms of security and efficiency.

IV. Evaluation of Component Fusion and Component Encryption

The previous chapter described two new methods for whitebox circuit obfuscation, *component fusion* and *component encryption*; this chapter establishes their validity and describes the characteristics of previously existing obfuscation methods in relation to component fusion and component encryption.

This thesis attempts to accomplish two goals; part of the first research goal (description of new methods for circuit obfuscation) is met in Chapter III, but the new methods described in Chapter III must be validated. That is accomplished in this chapter. The second goal (evaluation of subcircuit selection-and-replacement) is relegated to Appendix B.

This chapter validates that the first goal has been accomplished by describing the metrics necessary to evaluate the security and efficiency of random circuit variants produced by the component fusion and component encryption algorithms, and by then evaluating the validity of both algorithms according to their performance in efficiently and securely obfuscating three test circuits. A brief summary of the chapter is as follows:

Section 4.3 describes the variables used to control each obfuscation algorithm, and Section 4.1 describes the metrics (both desirable and undesirable) which are measured in the circuit variants. Section 4.2 describes the three test circuits used to test all of the obfuscation algorithms, and Section 4.4 presents raw results for each test circuit.

Lastly, Sections 4.5 through 4.9.5 examine the reasons why the results produced by experiments in this thesis exhibit certain properties related to security and efficiency.

Following this chapter, conclusions will be drawn from the results in this chapter to determine the success of the research presented in this thesis, and these conclusions will be used to identify future research areas.

4.1 Evaluation

Component fusion and component encryption were benchmarked against Parham's Boundary Blurring algorithm and against random subcircuit selection-and-replacement. This was done to highlight the differences in security and efficiency of each method.

Firstly, each method was examined to ensure that obfuscated circuits are semantically equivalent to the original circuit.

Secondly, each method was examined with regard to security:

- **Internal signal existence:** Do the original internal signals of the circuit exist anywhere? If they do (at some location other than the circuit inputs or outputs), then it is possible that these signals indicate a component boundary or a portion of the original circuit which contains its original semantics. This metric is acquired by analyzing the truth-table value of every gate in the circuit variant and comparing each truth-table to the truth-tables of the gates in the original circuit. Examining internal signals is only possible for circuits with a reasonably small number of inputs. Section 4.3.5 examines this concept in more depth.
- **Boundary hiding:** Do the original component boundaries exist? Is there some structure with the original input-output flow which computes the function of one of the original components? If these boundaries do not exist, then what would be required to identify the new boundaries? This metric was analyzed using the component identification tool.

Lastly, each method was examined with regard to efficiency:

- **Level Count/Circuit Delay:** Has the depth of the circuit increased? By how much? As noted in Chapter I, this metric represents the increased or decreased circuit delay of an obfuscated circuit.
- **Gate Count/Power and Area Requirements:** Has the number of gates in the circuit increased? By how much? As noted in Chapter I, this metric

represents the increased or decreased power and physical area required for an obfuscated circuit

- **Algorithm Runtime:** How long does each method take to complete? As noted in Chapter I, this metric represents the cost to a circuit designer in securing a circuit design.

For reference, all runtimes provided in this thesis were measured on quad-core Intel Xeon processors running at 3.00 GHz with 3.14 or 4 GB of RAM. Minor processing steps involved in preparing data (such as component identification and reduction) were performed off-line on other computers.

4.2 Test cases

Three test cases were chosen to compare the characteristics of circuit variants produced by subcircuit selection-and-replacement to variants produced by implementations of deterministic SSR and component SVP. These circuits represent two different domains of circuits which would benefit from protection, and the variants provide a wide sampling of the effects produced by each algorithm.

4.2.0.1 c264: 4-bit multiplier. The c264 circuit is a scaled-down version of the c6288 16-bit multiplier circuit which is easier to view because of its smaller size. (For more information on the significance of the c6288 circuit, see section 4.2.2.) This circuit multiplies two four-bit numbers, and is composed of 132 gates with 12 major functional blocks (4 half-adder cells and 8 full-adder cells). This circuit better depicts the structure in the c6288 circuit, and serves as another test case which is likely whose protection scales to the protection of c6288.

4.2.1 Polymorphic Circuitry.

4.2.1.1 Background. Cady [7] describes a key-based system for completely protecting the identities of logic gates within a circuit. Reasoning that all

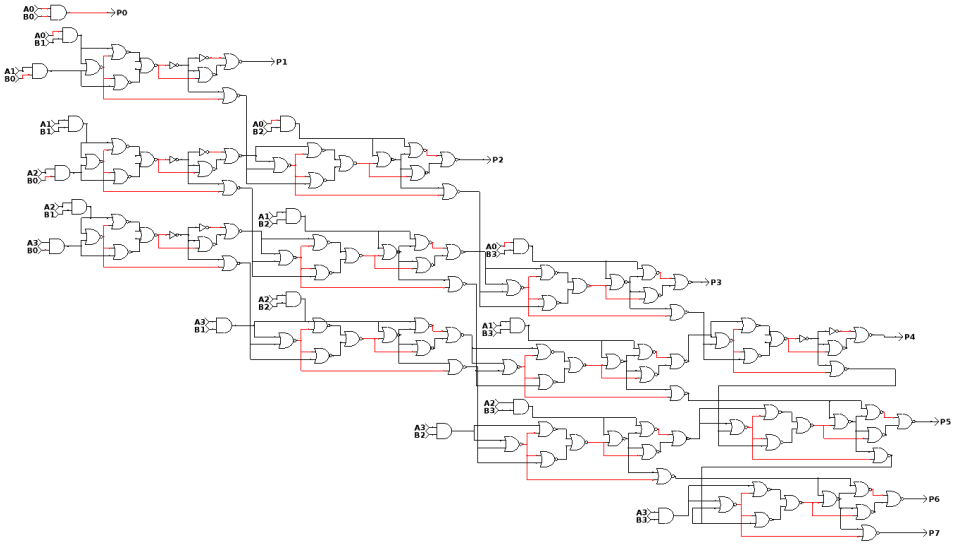


Figure 4.1: The 4-bit multiplier represented using circuit logic gates

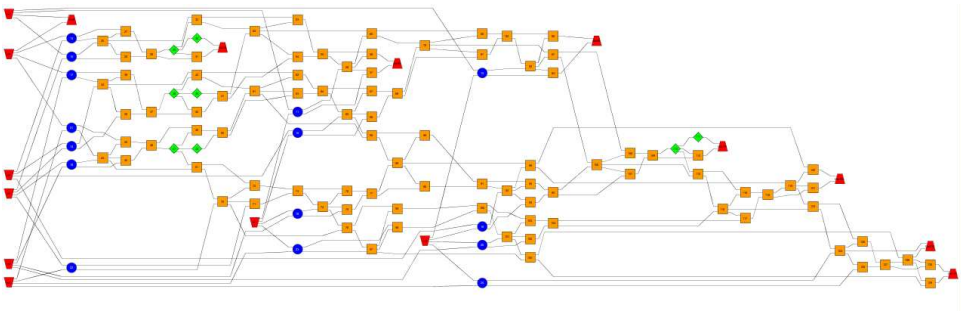


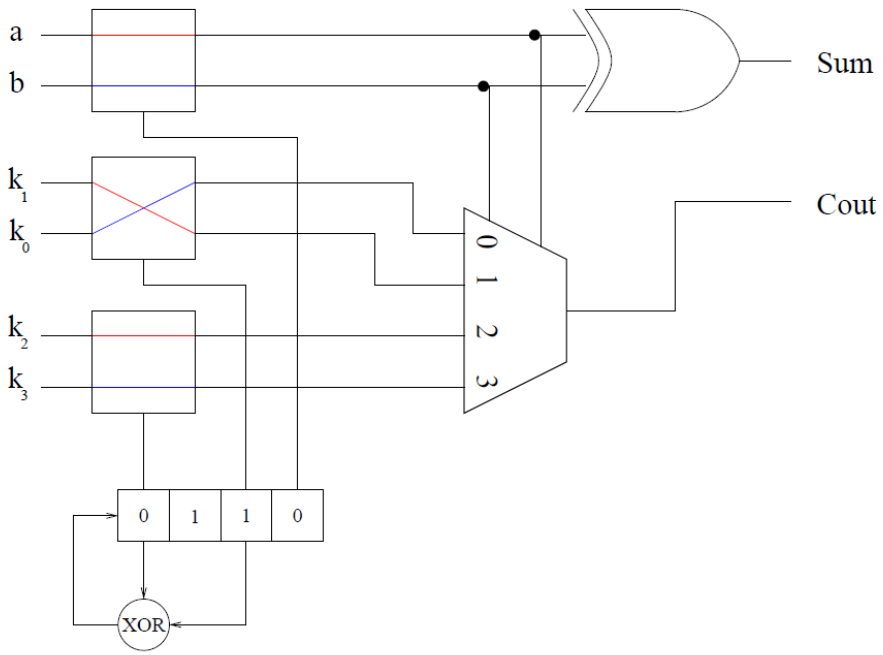
Figure 4.2: A graph-based representation of the 4-bit multiplier circuit.

the functions of all two-input logic gates can be specified using a truth table with 4 rows, Cady proposes the use of 4-to-1 multiplexors (which are 6-input, 1-output components) as replacements for 2-input logic gates.

Definition 17. ***Polymorphic Gate****: A multiplexor used to implement an n-input logic function, having n select lines, 2^n inputs, and one output, and whose logic function is specified by the truth table applied to the 2^n multiplexor inputs. The elementary logic function computed by a polymorphic gate is entirely dependent on the inputs to the multiplexor.*

Polymorphic gate structures can be used as part of a more comprehensive dynamic black-box protection using dynamic input reordering. Cady describes a polymorphic switching network which connects its input signals to its output signals based upon the value applied on a control line. Figure 4.3 shows Cady's intended use of a linear feedback shift register (LFSR) to change the values applied to each of the switching networks, thus causing the black-box semantics of the circuit component to change dynamically. If the inputs to a circuit component come from a component whose outputs are being dynamically reordered using the same LFSR sequence, then dynamic encryption has been been provided to the circuit.

4.2.1.2 Circuit composition. A circuit composed of polymorphic gates will be constructed at least partly using multiplexors. Individual logic gates are replaced by multiplexor components, and these multiplexors are fed by keys that define their logic function. Because multiplexors have a well-defined semantics, identifying subcircuits with the semantics of a multiplexor is easy. To make the process even more easy, the X-Hot Input Analysis technique described by Porter can easily be extended to apply to multiplexor components. An attack on the polymorphic gate component (assuming that an adversary has knowledge of the structure of a polymorphic gate) would be similar in many ways to the work on reverse-engineering the ISCAS-85 benchmarks, in which identifying well-known components was helpful in reconstructing higher-level functionality.



Network altering the order of the inputs to the half adder

Figure 4.3: A Linear Feedback Shift Register (LFSR) used to control keys of polymorphic gates [7]

Multiplexors may either be constructed optimally from gate logic, or constructed from smaller optimally implemented multiplexors arranged in a tree structure. If multiplexors are constructed from smaller multiplexors, there is an even better chance that a reverse engineer will identify some component because two types of components will exist, the smaller multiplexor and the larger multiplexor (and the smaller multiplexor will exist inside the larger).

4.2.1.3 Test circuit. The test circuit (depicted in Figure 4.4)implemented for this test case is a single full-adder similar to the one that Cady used to create the 32-bit full adder used in his experimentation. Unlike Cady's full-adder (depicted in Figure 4.5, the full-adder used in this thesis provides only a two-bit key to each multiplexor. A fully polymorphic gate would have a four-bit key (unlike the implementations in either this thesis or Cady's), and a fully polymorphic circuit would have a key for every gate (unlike the implementation of the full-adder in either this thesis or Cady's). If the full-adder circuit used as a test case in this thesis were fully polymorphic, it would have 23 inputs (four inputs for each of the five gates in the full-adder, and 3 for the inputs to the circuit).

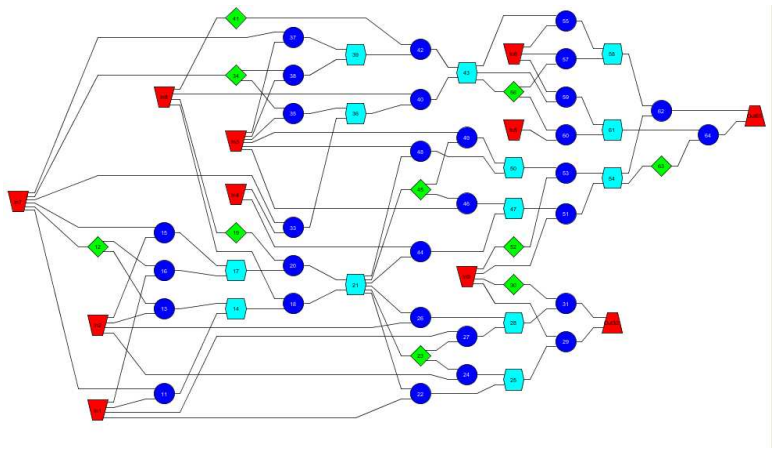


Figure 4.4: A test full-adder composed of polymorphic gates.

However, in this thesis the test circuit provides each gate with only the signals 0 and 1. Cady similarly limits the capability of the polymorphic gate by tying together

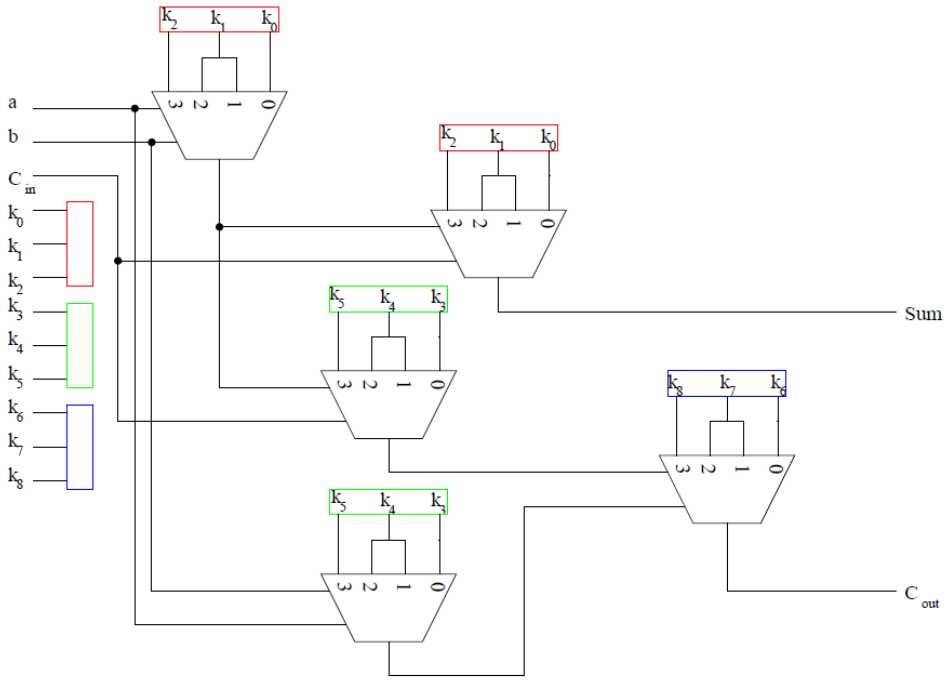


Figure 4.5: A full-adder circuit composed entirely of polymorphic gates [7]

the two middle inputs. (providing each gate with three signals instead of four) Figure 4.6 shows an AND gate, an OR gate, and an XOR gate constructed in this fashion. The concept of polymorphism is still used - if the key is changed, the circuit semantics change.

4.2.2 ISCAS Benchmark 6288: 16-bit multiplier.

4.2.2.1 Background. The ISCAS-85 circuits [18] are a set of industrial circuit designs whose designs were not published, both for confidentiality reasons and for the purpose of providing a random set of logic circuit designs to be used as benchmarks.

However, Hansen et al. were able to reverse-engineer this set of benchmarks to reveal the function of each of the benchmark circuits. While Hansen et al.'s work is significant because it is a case of reverse engineering, the ISCAS-85 circuits are

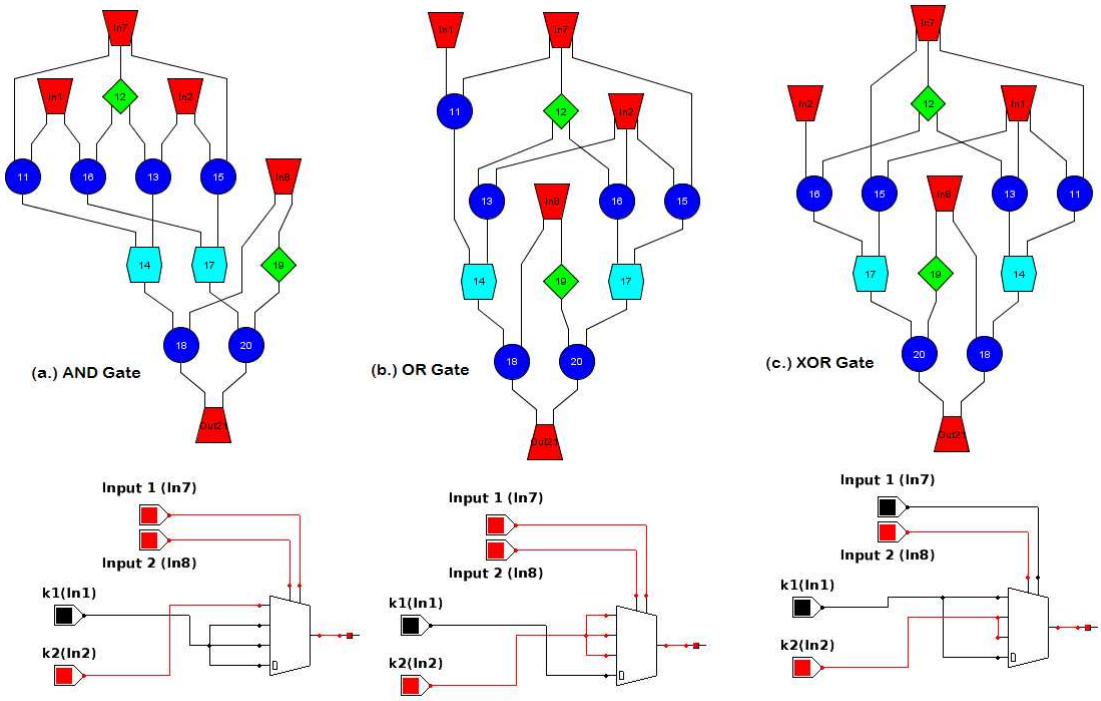


Figure 4.6: (a.) A polymorphic AND gate, (b.) a polymorphic OR gate, and (c.) a polymorphic XOR gate.

primarily useful as a set of benchmark circuits (which is the original purpose of the ISCAS-85 circuits).

4.2.2.2 Circuit composition. The c6288 circuit is a 16-bit by 16-bit multiplier circuit with 32 inputs and 32 outputs and containing 2,406 gates. Hansen et al. note that c6288 is a 124-level circuit composed of 240 major functional blocks, which are both full- and half-adder cells. This circuit presents an ideal scenario for a deterministic circuit variant generator because it contains a large number of small modules which are individually easily minimized. Given advances in computing technology, it is possible that a c6288 circuit (or any 32-bit circuit) could be completely synthesized (which would perfectly protect all internal circuit structure of c6288), but examining the feasibility of completely minimizing benchmarks was outside the scope of this research.

Like the polymorphic-gate circuit proposed by Cady, c6288 consists of a large number of identical modules. Unlike the polymorphic gate structure, however, modules in c6288 are largely all connected in the same manner. Because of the number of functional blocks in the circuit exceeds the number of possible component variants, it is still possible that an adversary might be able to visually identify the location of components but not be able to easily identify the original component boundaries (that is, the white-box obfuscation algorithm will provide measurable security, but won't provide topological randomness). Using Collberg et al.'s terminology, [9] it is possible that the obfuscation algorithm will be resilient but not potent.

The version of c6288 used in this experimentation was a variant of the original c6288 in which the entire c6288 circuit was first decomposed entirely into NOR gates. This did not affect the identification of the full-adder or half-adder components in any way (both components were identifiable in the circuit), but it did affect the gate constitution and size of the circuit. Unlike the original c6288, the nor-decomposed

c6288 has 2928 gates and 125 levels, the result of decomposing each AND gate into Figure 4.7 shows a graph representation of the gates in the c6288-nor circuit.

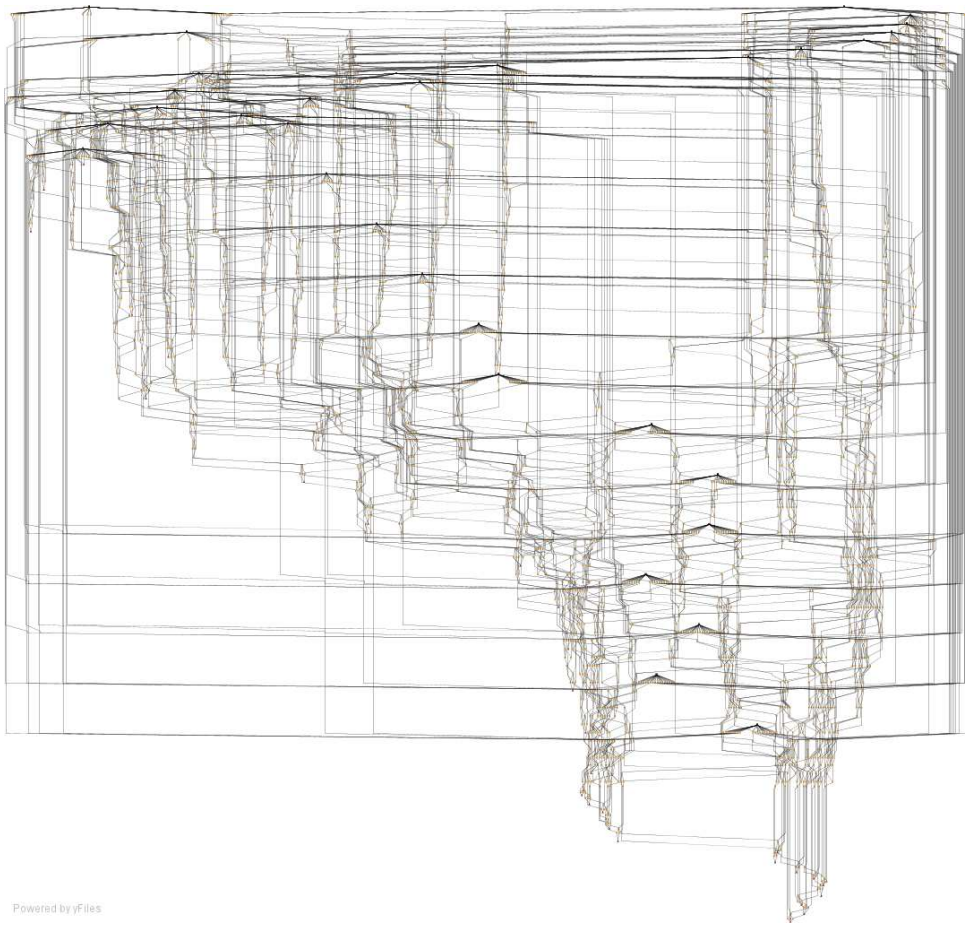


Figure 4.7: A graph-based representation of the c6288 ISCAS benchmark circuit.

4.3 Experimental Setup

4.3.1 Random Subcircuit Selection-and-Replacement. As a benchmark, the random subcircuit selection-and-replacement implementation developed by Norman and James was run on each of the benchmark circuits to provide reference statistics about the performance of SSR algorithms in terms of efficiency and security properties. This tool was implemented in the Java programming language and uses the JGraphT library to represent circuits; this tool was described in Section 2.5.1.1. For optimum results, the settings that have produced the best results in previous research

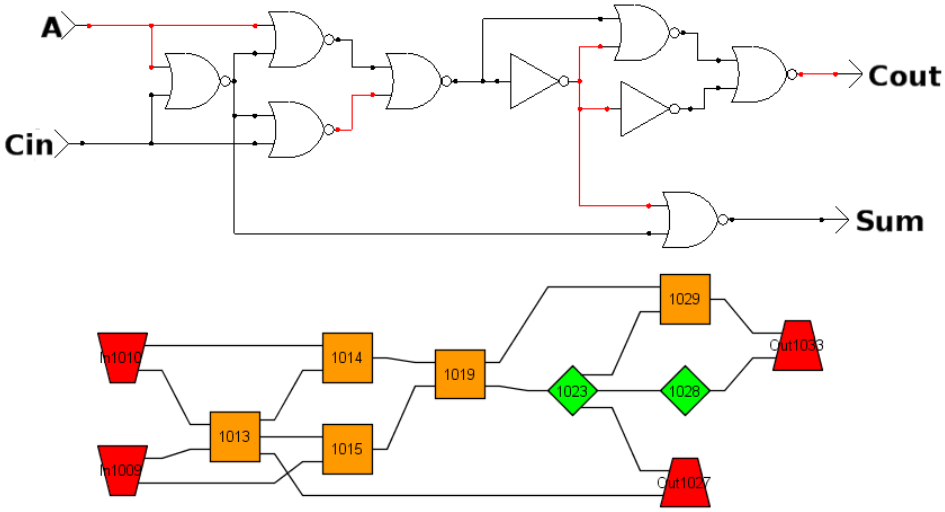


Figure 4.8: Gate- and graph-based representations of the half-adder component.

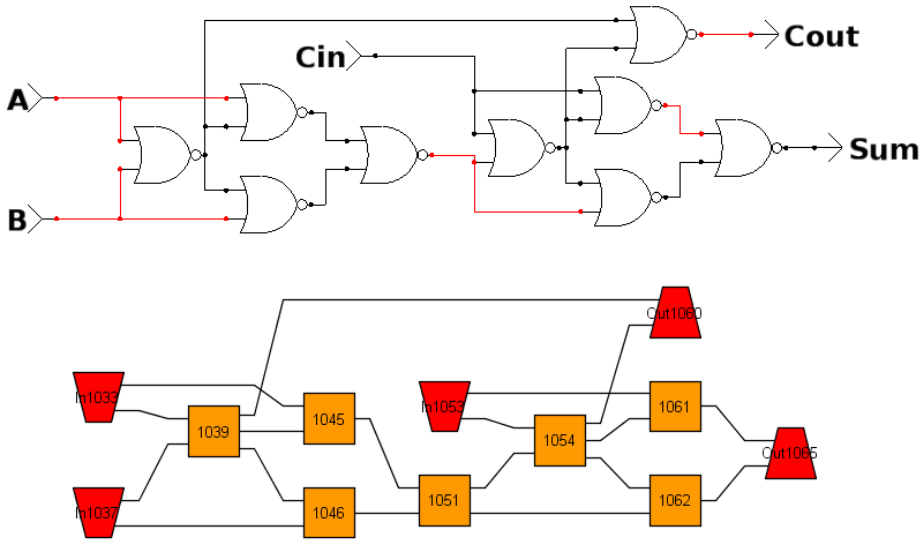


Figure 4.9: Gate-and graph-based representations of a full-adder component.

efforts were used. This research examines four different selection strategies. Because Williams noted that the RandomTwoGates selection strategy seemed to perform as well or better than other algorithms at increasing ancestral entropy, the variants used in this experimentation were generated using two different strategies used in different experiments:

1. Select two gates randomly.
2. Select three gates randomly.

Six options govern the choosing of random replacements, and broadly eliminate or include categories of random circuits. A seventh option allows a choice of the gate basis for replacement. A seventh option allows a limitation of the gate types chosen; this research assumes that all replacements should consist of two-input logic functions from the set of AND, OR, NAND, NOR, XOR, and XNOR gates. [20]

1. **RedundantGates** - Allow gates identical to one another in the replacement circuit?
2. **AllowConstants** - Allow access to the constants 0 and 1 in the replacement circuit?
3. **DoubleInputs** - Allow gates with both inputs coming from the same signal?
4. **SymmetricGates** - Allow symmetric gates? That is, should replacement families consider a gate with inputs (X_1, X_2) different from a gate with inputs (X_2, X_1)?
5. **SimpleOutputs** - Don't allow gates in the replacement that aren't used in the computation of the outputs?
6. **ExactCount** - Only allow replacements that are exactly the size requested? (If false, allow smaller gate counts in the replacement).

In his analysis of CORGI variants, Kim observed that the configuration FFFTTT (false, false, false, true, true, true, with respect to the six options above) provided

less redundancy than the other option settings that he had considered. [22] For this reason, the FFFTTT configuration was chosen for SSR experiments in this research. Table 4.1 shows the SSR variants which were analyzed in this thesis.

Table 4.1: Subcircuit Selection-and-replacment trials

Selection Strategy	**Selection Size**	**Replacement Size**	**Iteration Count**	**Circuits**
Random Gates	2	3	500	All
			1000	All
			2000	All
			3000	All
		4	500	All
			1000	All
	3	4	0	c264
			25	c264
			50	c264
			100	c264
Related Gates		4	500	c264/polymorphic
			1000	c264/polymorphic
			2000	c264/polymorphic
			3000	c264/polymorphic

4.3.1.1 Pattern-based reduction. Kim developed a pattern-based logic reducer for the minimization of circuit variants, and used this reducer to make observations about the resilience of variants produced by subcircuit selection-and-replacement. [22] This reducer was applied to all circuit variants generated by SSR routines, and was also applied to all circuit variants generated by algorithms developed in this thesis. The optimum pattern reduction order identified by Kim was used in all cases.

4.3.2 Deterministic selection-and-replacement. Deterministic SSR occurs in rounds, and during each round the entire circuit is replaced. To accomplish complete circuit replacement, DSSR relies on two methods of deterministically identifying subcircuit selections: (a.) component identification and (b.) circuit partitioning. In the first round, component identification is used to determine subcircuits which must be

replaced; then, any gates which remain in the circuit are partitioned into subcircuits. All subcircuits are placed in a queue for replacement. Each round is made up of many iterations; during an iteration, a subcircuit is removed from the queue. All of the predecessor gates (if there are any) to that subcircuit are added to the subcircuit, and then the subcircuit is replaced. Control variables for deterministic SSR are (a.) the number of rounds of SSR the circuit undergoes, (b.) the size of the subcircuit partitioning, and (c.) the algorithm used to generate replacements.

Because the algorithm used for circuit minimization in this research is a two-level minimizer, replacements often contain a decomposed version of a two-level PLA form with decomposed versions of larger gates (this problem is depicted in Figure 3.10. For anything other than a small circuit selection, the replacement circuit increases the size of the overall circuit. The consequences for this are that some circuit variants become unmanageably large after even a small number of rounds. Table 4.2 shows that only one round of variation was executed on c6288, but two rounds were executed on c264 and three rounds were executed on the small polymorphic fulladder circuit.

Table 4.2: Trials of the deterministic subcircuit selection-and-replacement algorithm.

Circuit	**Components**	**Rounds**	**Trials**
Polymorphic full-adder	5	3	3
c264 (4-bit multiplier)	12	2	3
c6288	240	1	3

4.3.3 Component Encryption. Like component fusion, component encryption completely covers a circuit. However, component encryption does not protect anything other than identified components. This means that only one iteration of component encryption is possible unless a component is discovered to still be in the circuit. The controllable variables for component encryption are (a.) the number of signals by which to increase the connections between any two components (that is, the number of signals that the signal mapping function increases by) and (b.) the algorithm used to generate replacements. In the test implementation, component

encryption identifies the connections between individual components and increases the number of signals by either 1 or 2; component encryption chooses to increase the number of signals in a connection by 1 with a $\frac{2}{3}$ probability and by 2 with a $\frac{1}{3}$ probability. It would also be possible to choose not to increase the number of signals (which would be equivalent to merely permuting the possible truth-table values of the signals of the connection between components), but this thesis focused on schemes for modifying the number of signals between components.

4.3.4 Verification of Variant Correctness. To verify that circuit variants are in fact semantically identical to the original circuit, the outputs of the c264 4-bit multiplier and the polymorphic full-adder circuit variants were verified against the outputs of the original c264 and polymorphic full-adder circuits. (c6288 was not verified in this manner because its input count of 32 made the size of output truth-table comparisons intractably large.)

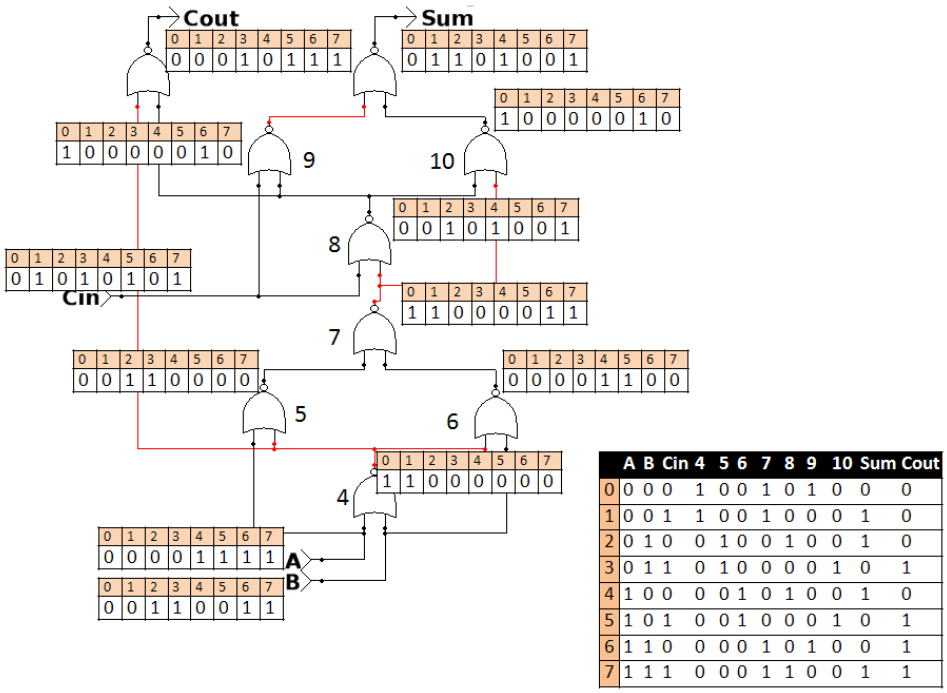
The output ordering of a circuit is not reliably preserved by the circuit variants produced by the component encryption implementation used for this thesis (but the input ordering is preserved, and circuit variants produce signals with semantically equivalent output), so the algorithm for verifying semantic equivalence checks that the circuit variant contains every one of the original outputs. The worst case runtime for this equivalence check is $O(n^2)$, where the variable n is the number of outputs.

Every c264 4-bit multiplier circuit variant and polymorphic fulladder variant produced by the component fusion and component encryption routines was verified to be semantically equivalent to the original c264 and polymorphic full-adder (respectively) though the outputs of circuits produced by the component encryption routine were ordered differently due to the generation process.

4.3.5 Identifying Common signals. The word *signal* is used to refer to some gate output with a designated truth-table value. An illustration of this concept is provided in Figure 4.10, which illustrates the signals on every wire in a full-adder

circuit. Essentially, the re-appearance of wire's truth tables and their inversions in circuit variants can be used to indicate how much of a circuit is identical to the original. Figure 4.15 shows the relative number of signals preserved by each algorithm.

Signal analysis is significant because it represents the amount of information that an adversary would be able to find out through side-channel analysis. If the original circuit signals are present in the variant, then the adversary searching for the side-channel characteristics (switching activity, for example) of those signals will be able to find them in the circuit.



	A	B	Cin	4	5	6	7	8	9	10	Sum	Cout
0	0	0	0	1	0	0	1	0	1	0	0	0
1	0	0	1	1	0	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0	1	0	0	1	0
3	0	1	1	0	1	0	0	0	0	1	0	1
4	1	0	0	0	0	1	0	1	0	0	1	0
5	1	0	1	0	0	1	0	0	0	1	0	1
6	1	1	0	0	0	0	1	0	1	0	0	1
7	1	1	1	0	0	0	1	1	0	0	1	1

Figure 4.10: Example signals in a full-adder circuit.

4.3.6 Identifying Boundaries using Component Identification. The component identification tool implemented by Parham was used to identify the components in each of the Parham's benchmark circuits, demonstrating that each of the benchmarks contains components which can be identified automatically using White's algorithm. [35] [48]. This tool was used (with some modifications) to identify the components in each of the test case circuits in this thesis. The results of running

component identification on these circuits is shown in Table 4.3. As the table shows, each of the 240 full- and half-adder cells were identified in the c6288 circuit, each of the 12 full- and half-adder cells were identified in the c264 circuit, and each of the 2-input 1-output multiplexors (3-input 1-output components) and each of the 4-input 1-output multiplexors (6-input, 1-output components) was identified. Identification of the 4-1 multiplexors required specifying the polymorphic AND, OR, and XOR gates individually as components (For example, to identify both of the XOR gates in the circuit, the semantics of a polymorphic XOR gate were placed in the module library; to identify both of the AND gates, the semantics of a polymorphic AND gate were placed in the module library. If a polymorphic gate as specified by Cady had been used, then only one polymorphic gate definition would have been needed.).

Table 4.3: Components in each test case circuit

Circuit	**Component Type**	**Components Identified**
c264	Full-adder	8
	Half-adder	4
Polymorphic Full-adder	2-1 Multiplexor	15
	Polymorphic AND	2
	Polymorphic OR	1
	Polymorphic XOR	2
c6288	Full-adder	224
	Half-adder	16

4.4 Results

As described in Section 4.3, each of four algorithms was executed on three test circuits for varying trial counts. Tables 4.4, 4.5, and 4.6 provide reference summary statistics for each test circuit, and a single diagram of the c264 variant is shown to illustrate the variant at the terminations of the select-2 replace-3 SSR algorithm (Figure 4.11), the Level 3 Don't Care boundary blurring algorithm (Figure 4.12), the component fusion algorithm(Figure 4.13), and the component encryption algorithm (Figure 4.14). An analysis of the metrics from each of these trials begins in Section 4.8.

Table 4.4: Metrics of c264 variants

	Select-2 Replace-3 SSR				**Select-2 Replace-4 SSR**		**Don't Care Boundary Blurring**	**Component Fusion**		**Component Encryption**
Metrics	**500**	**1000**	**2000**	**3000**	**500**	**1000**	**Level-3**	**1**	**2**	**1**
Trials	2				1		10	3		3
Inputs	8	8	8	8	8	8	8	8	8	8
Outputs	8	8	8	8	8	8	8	8	8	8
Original gates	124	124	124	124	124	124	124	124	124	124
Original level count	34	34	34	34	34	34	34	34	34	34
Gates in variant	143.5	151	154.5	158.0	833	1542	352.5	631.66	2082.66	539.66
New level count	43.5	46	46.5	48	249	505	59.1	30.33	52	30.66
Gates increase	19.5	27	30.5	34	709	1418	228.5	507.66	1958.66	415.66
Levels increase	9.5	12	12.5	14.0	215	471.0	25.1	-3.66	18.0	-3.33
Unreplaced Gates	12	8	6.5	6	21	18	N/A	N/A	N/A	N/A
Unique original signals in new	124	124	124	124	123	47.0	124	61	73.33	61
Unique original signals inverted in new	34.5	35.5	36.5	37.0	77	39.0	66.8	54.66	65.66	52.66
New signals copied from original signals	131	135	135.5	137.5	341	318	213.2	121.66	308.66	119.66
New signals copied from inverted original signals	32.5	37	40	42.5	270	305	95.8	110.66	242.66	85.66
Signals preserved	100%	100%	100%	100%	99.19%	37.9%	100%	49.19%	59.13%	49.19%
New gates producing original signals	91.34%	89.68%	87.97%	87.12%	40.93%	20.62%	60.66%	19.35%	14.87%	22.83%
Full-adders identified ($\leq$ 15 gates)	5	4	4	3.5	0	0	0	0	0	0/8
Half-adders identified ($\leq$ 15 gates)	3.5	3	3	3	0	0	0	0	0	.66/4
Runtime (s)	243	628	1715.5	3678	14050	28859	14	4	159.33	~1

Table 4.5: Metrics of select-2, replace-with-3 replacement on a polymorphic full-adder

	Select-2 Replace-3 SSR				**Select-2 Replace-4 SSR**		**Don't Care Boundary Blurring**	**Component Fusion**			**Component Encryption**
Metrics	**500**	**1000**	**2000**	**3000**	**500**	**1000**	**Level-3**	**1**	**2**	**3**	**1**
Trials	2				1		10	3			3
Inputs	9	9	9	9	9	9	9	9	9	9	9
Outputs	2	2	2	2	2	2	2	2	2	2	2
Original gates	60	60	60	60	60	60	60	60	60	60	60
Original level count	13	13	13	13	13	13	13	13	13	13	13
Gates in variant	73.5	78	94.5	95.5	799	1452	223.7	65	147.66	343.66	199.33
New level count	20	22.5	33.5	34	289	496	44.3	13	15.33	22.0	21.66
Gates increase	13.5	18	34.5	35.5	739	1392	163.69	5	87.66	283.66	139.33
Levels increase	7	9.5	20.5	21	276	483	31.3	0	2.33	9	8.66
Unreplaced Gates	3	2	2	2	5	4	0	N/A	N/A	N/A	N/A
Unique original signals in new	57	57	59	59	47	27	60	24	19	21.33	32.33
Unique original signals inverted in new	16	16.5	18	18.5	39	25	20	13.33	16.33	14.33	21.33
New signals copied from original signals	61	63	73	73	88	80	109.5	19.33	21.66	33.66	30.0
New signals copied from inverted original signals	17	17.5	19.5	20	74	69	23.2	11.66	13.66	22.33	24.0
Signals preserved	95%	95%	98.33%	98.33%	78.33%	45.0%	100%	39.99%	31.66%	35.55%	53.88%
New gates producing original signals	82.85%	80.52%	77.2%	76.33%	11.01%	5.5%	49.15%	29.8%	15.0%	10.71%	24.2%
2-in multiplexors identified (≤ 16 gates)	10.5	11.5	13	13	13	25	0	0	1.33	3.66	0
4-in multiplexors identified (≤ 16 gates)	3	3	3.5	3	0	1	0	0.33	0	0	0
Runtime (s)	208	482.5	1391	2896.5	13344	27888	5.5	8.33	9.33	13	< 1

Table 4.6: Metrics of select-2, replace-with-3 replacement on c6288

	Select-2 Replace-3 SSR				**Select-2 Replace-4 SSR**		**Don't Care Boundary Blurring**	**Component Fusion**	**Component Encryption**
Metrics/Iterations	**500**	**1000**	**2000**	**3000**	**500**	**1000**	**Level 3**	**1**	**1**
Trials	1				1		5	3	3
Inputs	32	32	32	32	32	32	32	32	32
Outputs	32	32	32	32	32	32	32	32	32
Original gates	2928	2928	2928	2928	2928	2928	2928	2928	2928
Original level count	125	125	125	125	125	125	125	125	125
Gates in variant	2444	2446	2455	2451	3061	3726	7668.4	15606.33	26275.67
New level count	124	124	124	124	494	593	248.6	174	198
Gates increase	-484	-482	-473	-477	133	798	4740	12678.33	23347.67
Levels increase	-1	-1	-1	-1	369	468	123.6	49	73
Full-adders identified ($\leq$ 15 gates)	224	224	224	224	109	62	0	0	*
Half-adders identified ($\leq$ 15 gates)	16	16	16	15	7	2	0	0	*
Runtime (s)	1306	3375	10313	20545	12280	28297	271.8	21956	127.33

4.5 Security Analysis: Internal Signal Hiding

4.5.1 Subcircuit selection-and-replacement. Because SSR involves small selections and replacements, many times SSR does not effectively hide internal circuit signals. This is certainly the case for select-2, replace-with-3 SSR.

Select-2, replace-with-4 does better; 1000 iterations of this algorithm succeeded at providing signal hiding in both the c264 4-bit multiplier and the polymorphic full-adder circuit.

4.5.2 Boundary Blurring. Signal analysis reveals a unique property of Don't Care boundary blurring: in all of the trials performed in this research, boundary blurring preserves all of the signals in the original circuit by including them in the variant. This is one of the worst possible results. Likely, this result is produced because the boundary blurring algorithm is set up in such a way that it preserves all of the original circuit signals.

4.5.3 Component Fusion and Component Encryption. Component Fusion and component encryption produce similar results when analyzed for preserved signals. Neither method preserves all signals, and neither method completely hides all signals. It is possible that some signals could be completely hidden (barring their appearance on the circuit output boundary), but this is probably statistically unlikely.

4.6 Security Analysis: Component Hiding

Boundary hiding is measured by the presence of components with the same number of inputs and outputs and the same semantics as the components in the original circuit. The component identification tool implemented by Parham in [35] uses the *Order Limited Focused Enumeration Algorithm* created by White in [49]. White reports that the worst-case performance of this algorithm is $O(2^c)$, where c represents the upper bound on the size of the subcircuits which can be enumerated. White also indicates that portions of the algorithm have worst-case performance $O(n^3)$ increasing

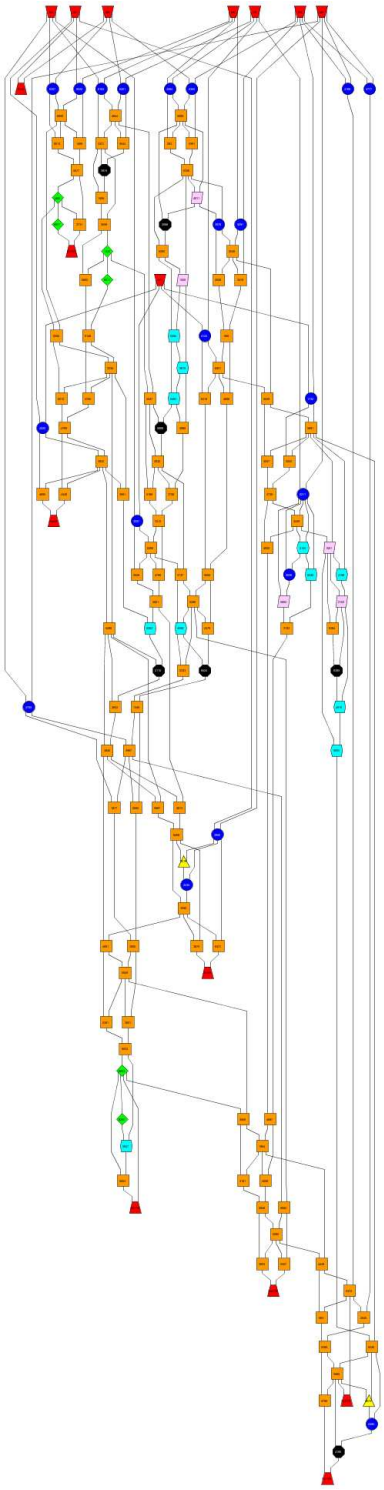


Figure 4.11: Select-2, Replace-3: c264 after 3000 obfuscation rounds and pattern-based reduction. Note that several full-adder components are entirely preserved.

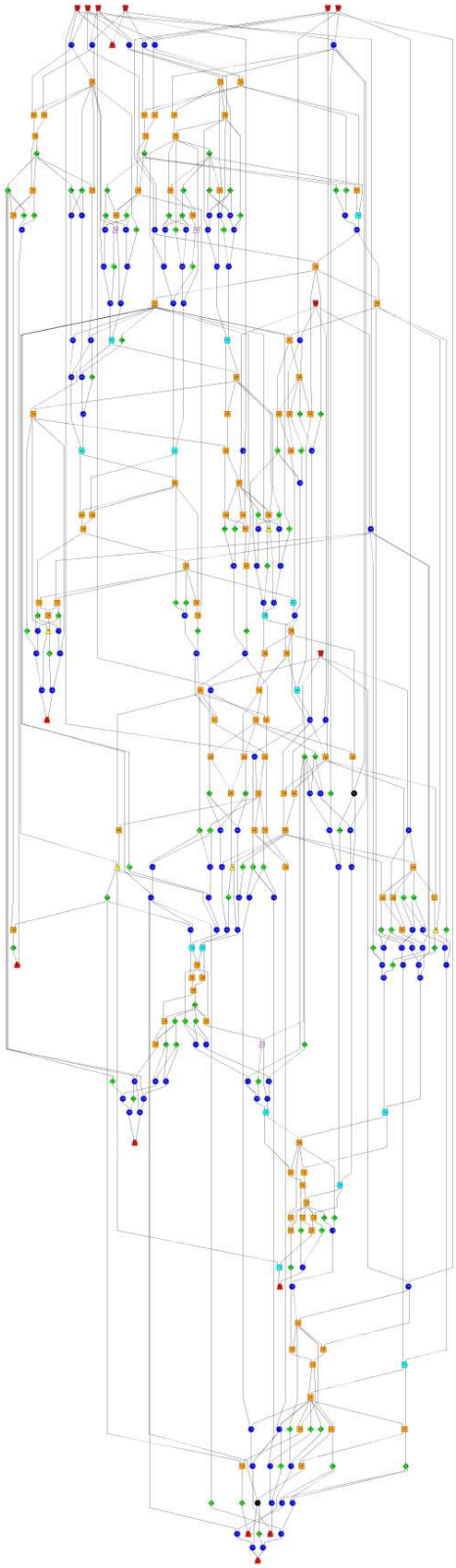


Figure 4.12: Boundary Blurring: 4-bit multiplier after a level-3 blur has been applied to all boundaries. Note that the number of circuit levels have increased, but the graph largely maintains its shape.

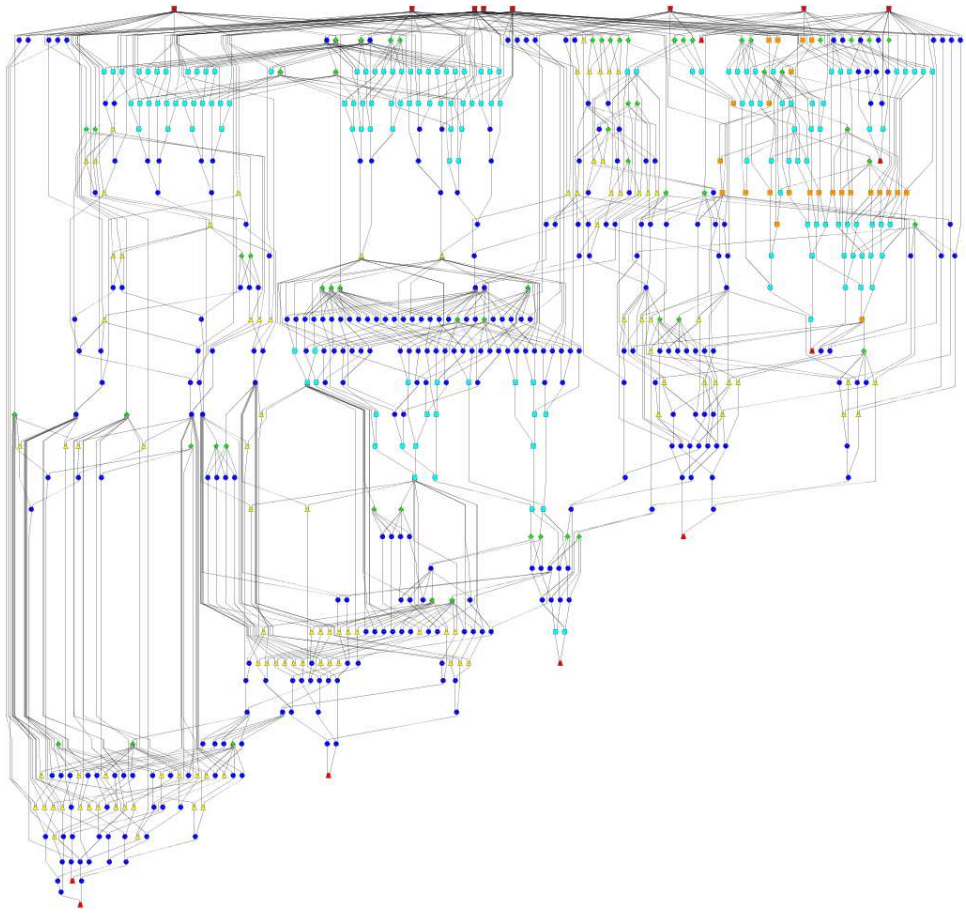


Figure 4.13: Component Fusion: 4-bit multiplier after one round of replacing all gates. Note the characteristic binary tree structure produced by the synthesis tool which increases the circuit width.

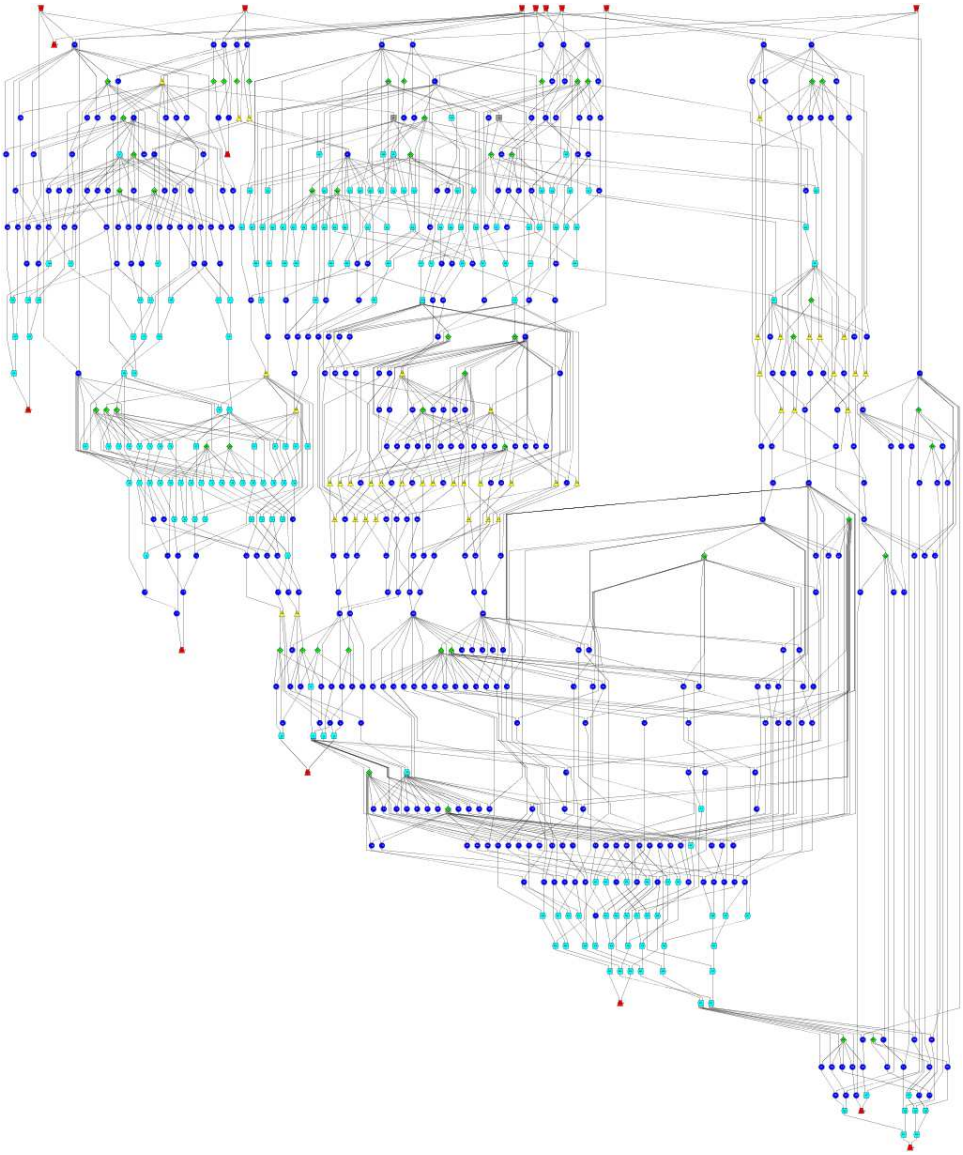


Figure 4.14: Component Encryption: 4-bit multiplier after signal value permutation. Note the characteristic binary tree structure produced by the synthesis tool which increases the circuit width.

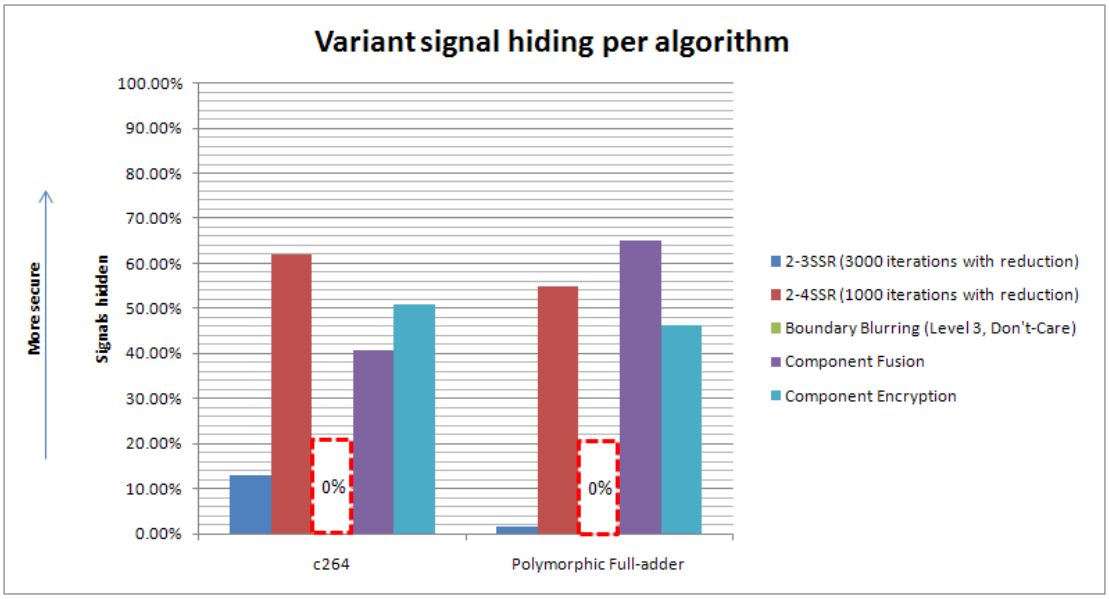


Figure 4.15: Signal Hiding: signals hidden after variant generation trials.

on the number of circuit gates n. Because of this, running component identification on circuit variants with gate counts greater than 15,000 was not attempted. Figures 4.16, 4.17, and 4.18 depict the percentage of components of each type that were hidden.

4.6.0.1 Subcircuit Selection-and-replacement. As seen in Tables 4.4 through 4.6, there were no cases where select-2, replace-3 SSR hid a component. In each case, it appears that reduction removed all added patterns and allowed the identification of some subcircuit.

The select-2 replace-4 algorithm did better. Successive iterations of select-2 replace-4 SSR were observed to hide some components. However, this experiment does not allow reasoning over whether additional iteration would ever completely hide all components.

4.6.1 Boundary Blurring. The boundary blurring algorithm successfully hid all of the components in each test circuit. This is expected because the boundary blurring algorithm was built for the express purpose of preventing component identification, and Parham documented its success at doing this.

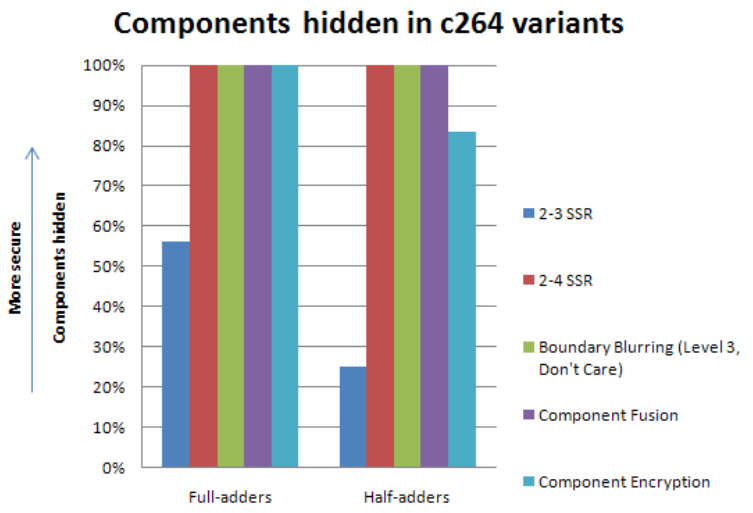


Figure 4.16: Component hiding in c264 across five different algorithms.

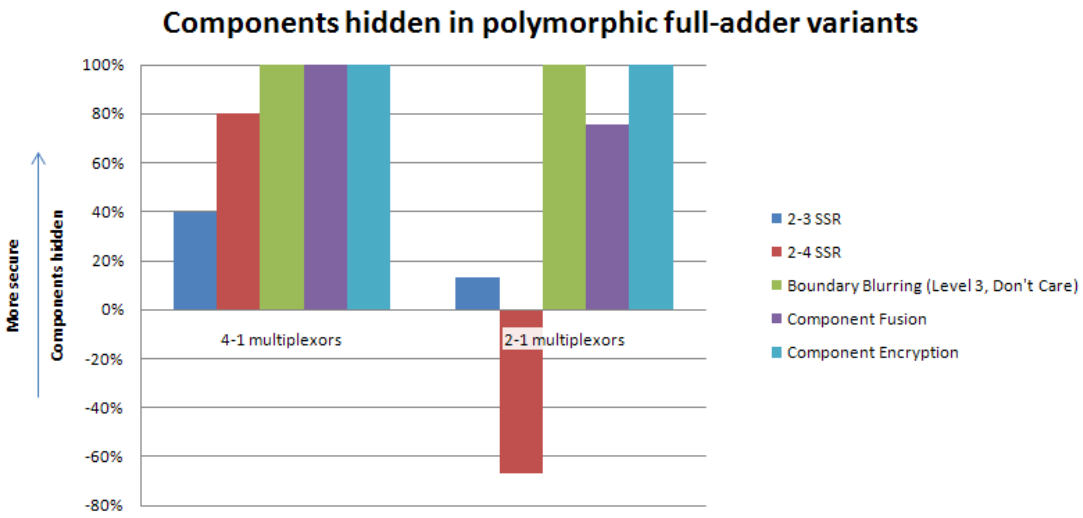


Figure 4.17: Component hiding in the polymorphic full-adder circuit across five different algorithms

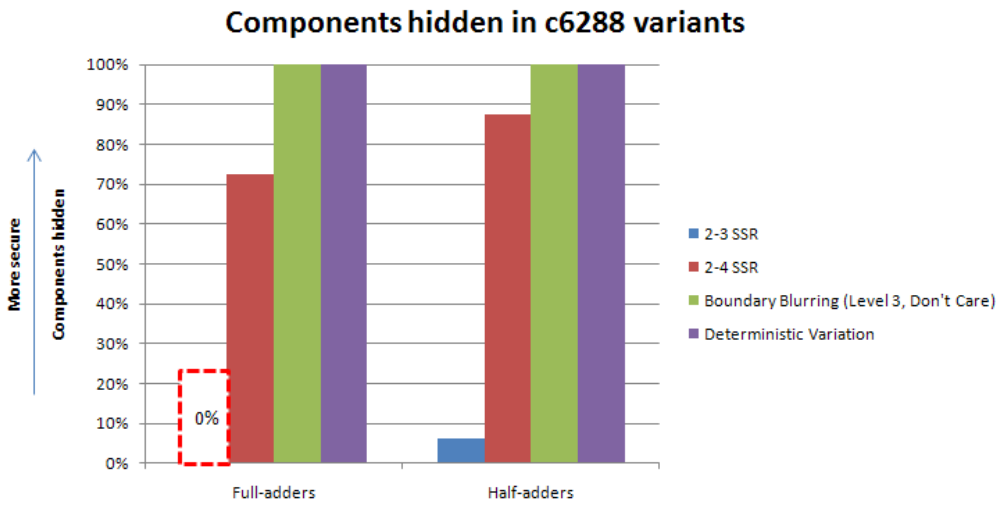


Figure 4.18: Component hiding in the c6288 circuit across four algorithms

4.6.2 Component Fusion. Component Fusion succeeds at hiding full-adder and half-adder components in the c264 circuit. However, it does not always succeed at hiding 4-in 1-out multiplexors. Most interesting is the fact that in the second and third passes through the polymorphic full-adder, components matching the semantics of a 2-in 1-out multiplexor seem to appear in the circuit variants. Of similar note is that, after 1000 iterations of select-2 replace-4 SSR, 25 2-in 1-out multiplexors are identified, even though there were only 15 in the original circuit, and a single 4-in, 1-out multiplexor was identified. This means that identifying 2-in, 1-out multiplexors will be difficult for an adversary because many cutsets with the semantics of this component will exist in the circuit.

4.6.3 Component Encryption. No components are identified in the the polymorphic full-adder circuit, and no full-adders are identified in the 4-bit multiplier. A half-adder was identified in the c264 4-bit multiplier circuit in two of three trials. There are four half-adder components in the 4-bit multiplier, and these are located at the input and output boundaries of the circuit. If a 'bad' signal mapping is used to permute the signals between components, it is plausible that a component with inputs unchanged and a bad output mapping would be identified. This can be prevented

if mappings are checked for 'badness' when they are generated. (See Section 3.4.5 for information on bad signal mappings.) If a system is in place to check maps for 'badness,' then it can reasonably be expected that no component with an internal boundary will be identified.

Note that under the component encryption scheme, only the known components are encrypted. Therefore, it is possible that there will be gates in the circuit which are not replaced. This means that if the component identifier used in the component encryption tool does not identify a component, that component will not be protected - it will be easily identified by an adversarial component identifier. It was computationally infeasible to perform component identification on the c6288 circuit because variants produced by this algorithm contained, on average, more than 15,000 gates. When run on the largest variant of 42856 gates, the component identifier was halted after 96 hours without completing the enumeration of a single circuit family size. It was about 50% complete with an attempt to semantically identify a subcircuit with 9 inputs and 5 outputs, which in worst case would require more than 43,500,000 comparisons using Parham's algorithm.

4.7 Efficiency Analysis: Levels increase

As a side characteristic of an increased gate count, the number of levels of the circuit will generally also increase. Increase in the number of hierarchical circuit levels is notable both because it is a characteristic of circuit topology and also because it is a measure of the physical gate delay - the amount of time between application of inputs to a circuit and the updating of the circuit output values is directly related to the number of levels in the circuit. Figure 4.19 shows the relative level count of variants produced by each algorithm.

4.7.1 SSR. The number of levels in variants produced by the random select-2, replace-with-3 algorithm showed an increase for both the c264 4-bit multiplier and the Polymorphic full-adder circuits. However, in the single trial of select-2, replace-3

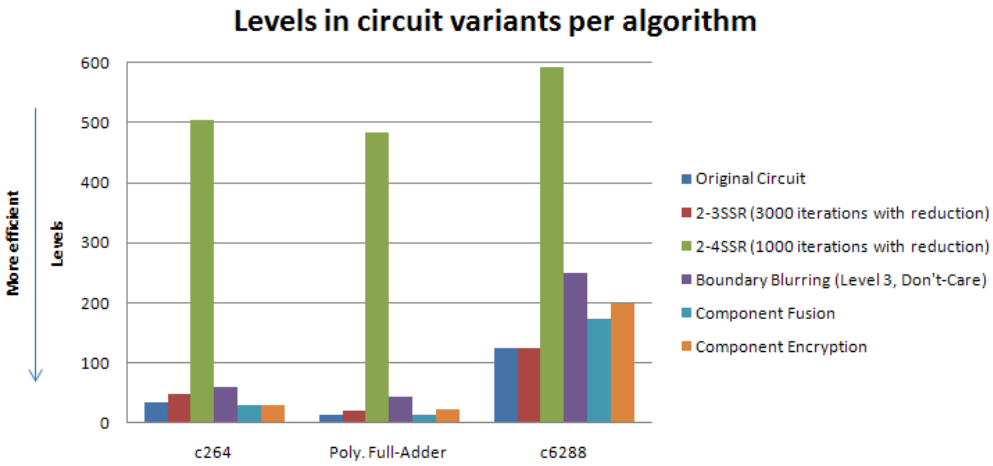


Figure 4.19: Level Count (Circuit Delay): Relative increase in number of levels of variant generation trials.

on the NOR-decomposed c6288 circuit, the number of levels decreased because most of the variation introduced into the c6288 circuit was reduced out by Kim's pattern-reduction tool.

In the random select-2, replace-with-4 test cases, the number of levels increased dramatically, growing faster than any other algorithm. This is because Kim's pattern-reduction tool does not reduce patterns introduced by this algorithm.

4.7.2 Boundary Blurring. Boundary blurring algorithms showed a modest increase in level count. This is likely due to the logic gates which are added to the circuit to replace the 'recovery gate.' As with gate size, level count is most affected in circuits which have a high ratio of boundary gates to total gates. Boundary blurring, when executed to hide the boundaries of the 2-1 multiplexors (4-gate components) increased the number of levels to 340.76% of the original count; when executed to hide the boundaries of full and half-adders (9-gate components) increased the number of levels to 173.82% of the original (for c264) and 198.88% (for c6288)

4.7.3 Component Fusion and Component Encryption. Component fusion and component encryption (SVP) both show modest level increases. Using the above example of a worst-case component for minimization, it is expected that the level

count of a component will have $O(log_2(i_{comp}) + i_{comp})$ complexity, where i_{comp} represents the expected number of inputs to each component. This is due to the of the binary-tree structure used for decomposing gates with greater than two inputs; the depth of a binary tree increases respective to the number of nodes n it contains with $O(log_2(n))$ complexity. (More precisely, in the worst case, one notional level of the component will contain $log_2(i_{comp})$ levels (the result of decomposing PLA gates with i_{comp} inputs), and one notional level will have $i_{comp} - 1$ levels, (the result of decomposing PLA gates with $2^{i_{comp}}/2$ inputs), and one level will be used for providing the inverted forms of every input signal. This value is computed as

$$log_2 i_{comp} + i_{comp} - 1 + 1$$

The number of levels by which each variant produced by these algorithms increases is less predictable than with other algorithms because the increase depends upon the way in which the original circuit components were implemented (which may have been optimal, but might not have been).

A rough estimate of the worst-case expected level count of c6288, then, is the depth of c6288 in terms of components (31 component-levels) multiplied by the expected depth of each component. The average level count for each of 5 c6288 variants

Table 4.7: Upper-bound on level count for the c6288 variants produced by the execution of component encryption

	Expected Inputs	**Expected levels**
1 component	5.56	8.16
c6288 (31 components deep)		252.96

was 198 levels, and the most extreme c6288 variant produced by the component encryption algorithm contained 242 levels. This is less than the upper bound predicted in Table 4.7.

4.8 Efficiency Analysis: Size increase

Size increase is primarily a metric related to the efficiency of the final circuit variant. For the core trials, Figure 4.20 depicts the relative gate count increase across all four algorithms and the c264 and polymorphic full-adder test cases. Figure 4.21 depicts the gate count increase for the c6288 circuit across all four algorithms.

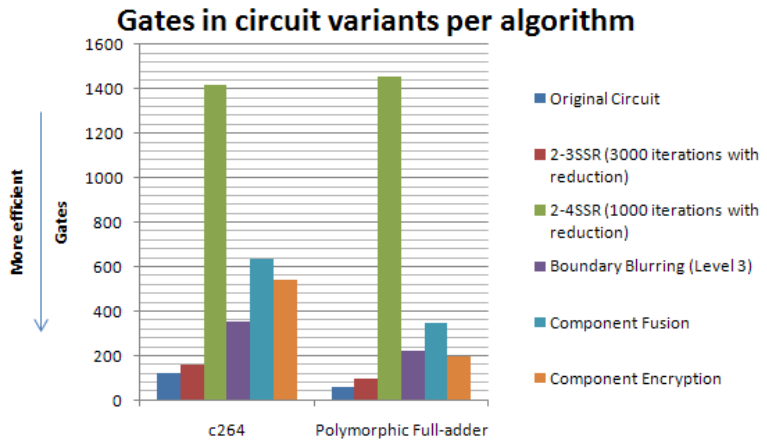


Figure 4.20: Gate Count: Relative size increase of variant generation trials.

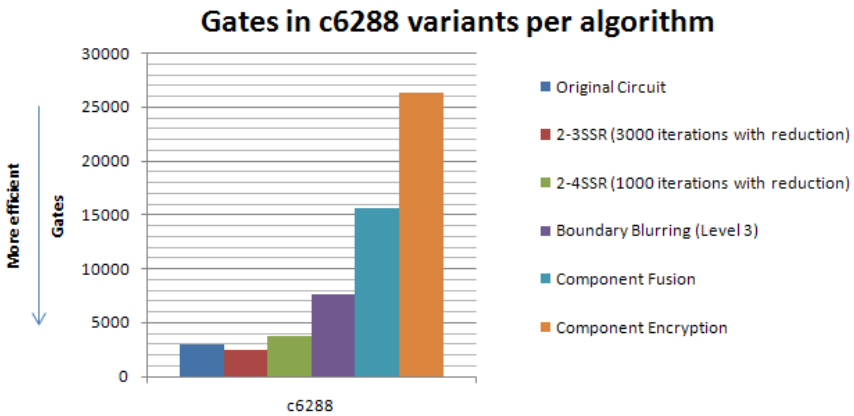


Figure 4.21: Gate Count (Power and Area Requirement): Relative size increase of variant generation trials for c6288.

4.8.0.1 SSR. Fundamentally, it is easy calculate how many gates are added to a circuit by random subcircuit selection and replacement; if 2 gates are

selected and replaced with 4 gates, then it is expected that the circuit size would increase by two gates. However, when there are occasions when random selection aborts rather than choosing a selection due to unsuccessful attempts at identifying a large enough set of random gates; these instances are considered in the iteration count.

In addition, upon algorithm completion, Kim's pattern reduction technique is used. This reduces circuit size if the SSR algorithm was biased towards introducing patterns that Kim's pattern reduction technique could find. Pattern reduction was constructed to target patterns introduced by the select-2, replace-with-3 algorithm. These techniques perform poorly at reducing patterns introduced by select-2, replace-with-4, as shown by the results from the test cases involving this algorithm. Thus, executing a select-2, replace-with-4 algorithm on each circuit causes near-linear growth on the number of iterations of SSR used, while executing a select-2, replace-with-3 algorithm on each circuit causes much less circuit growth. Notably, because the c6288 circuit used in this research was first decomposed into NOR gates, executing a select-2, replace-with-3 algorithm on this c6288 circuit reduced the circuit back almost into its original size of 2406 gates.

4.8.0.2 Boundary Blurring. It is expected that boundary blurring will increase circuit size because the algorithm adds additional logic to each circuit to hide the boundary. This is exhibited in every test case. Across the three test circuits, boundary blurring increased circuit size to 315.22% of its original size. As circuit size increased across the test circuits, percent gate increase became smaller. It is unknown exactly what caused this, but it is likely due to the relationship between the original circuit size and the original number of boundary gates; the number of boundary gates is determined by the number of input and output gates of each component in the circuit.

For this reason, it makes a difference what size component is used; in the case of the polymorphic full-adder circuit, 4-1 mulitiplexors are composed of 2-1 multiplexors.

There are 5 4-1 multiplexors in the circuit, but these can be decomposed into 15 2-1 multiplexors. To provide the boundary blurring algorithm with the largest number of boundaries, the boundaries of the 2-1 multiplexors were blurred.

4.8.0.3 Component Fusion and Component Encryption. Large size increase is noted in the variants produced by the component fusion and the component encryption algorithms. These algorithms both use the ESPRESSO engine to generate minimal forms. Component fusion uses the ESPRESSO engine to generate replacements deterministically, and the component encryption engine uses the ESPRESSO engine to generate a circuit which implements a new truth table with encoder/decoder logic applied to it. Because both routines described in this work uses a two-level minimizer, results are optimized for use in a PLA, but not for implementation using two-input gates. To use two-input gates, every gate specified by the two-level solution is decomposed into a binary tree of two-input gates.

For small subcircuits, the two-level minimizer does produce more minimal subcircuit implementations. However, beyond a certain threshold this minimizer consistently produces implementations which are larger than the original subcircuit. Both product-of-sums (POS) implementations and sum-of-products (SOP) implementations will cause subcircuit gate count to increase by the same magnitude, and this magnitude is capped by the number of minterms(or maxterms, in a POS circuit implementation) in the truth table of the subcircuit.

After ESPRESSO returns a minimized PLA to either routine, a circuit is generated to implement that minimized PLA. The first level of the new two-level circuit consists of all of the minterms(maxterms) which are used by the outputs of the subcircuit. The ESPRESSO algorithm allows the signals that produce individual minterms(maxterms) to be reused, so no minterm(maxterm) will ever be generated twice in the same subcircuit. It is, however, possible that in the worst case all of the minterms in a subcircuit's truth table could be produced as signals. If the subcircuit has n inputs and produces 2 outputs where one output is an XOR function on n

signals and the other output is an $XNOR$ function on n signals, then there will be 2^n first-level gates and two second-level gates. This is shown for a $n = 3$ inputs in Figure 4.22.

If all of the gates in the circuit are decomposed into 2-input gates, then each $n-$input gate will be decomposed into $n-1$ gates. (This decomposition is shown in Figure 3.10). This means that for the worst case, shown above, for i inputs and o outputs there will be $2^i * (i-1)$ gates on the first level. For this worst case to happen, every output gate must depend upon minterms which are not redundant; redundant terms will be factored out. The maximum sets of irredundant minterms(maxterms), of which the XOR and XNOR functions consist, are half the size of the number of subcircuit truth-table rows. This means that in the very worst case, there will be $2^i * (i-1) + 2^{i-1} * o$ gates (with i inverters) in a subcircuit replacement.

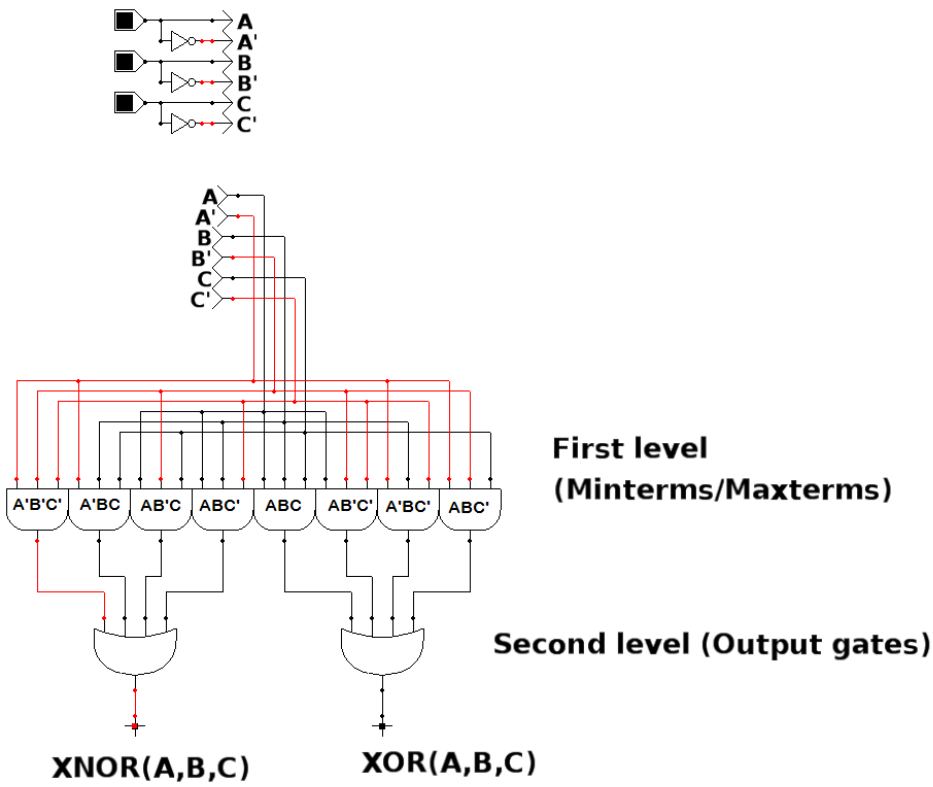


Figure 4.22: The worst-case gate size for deterministic replacement.

4.8.0.4 c6288. As will be noted in a later section, the c6288 circuit variants generated by encrypting each internal circuit component are too large to tractably execute the component identification algorithm on them. The expected number (mean number) of inputs and outputs for individual components in a c6288 variant are depicted in Table 4.8.

Table 4.8: Expected inputs and outputs for each component in the c6288 circuit

Count	**Original Inputs**	**Original Outputs**	**Expected Inputs**	**Expected Outputs**
13	2	2	2	4.66
14	3	2	4.33	4.66
14	3	2	5.66	3.33
13	3	2	7	3.33
1	2	2	2	4.66
1	2	2	2	3.33
1	3	2	5.66	2
1	2	2	4.66	3.33
182	3	2	5.66	4.66
Expected Overall	2.93	2	5.56	4.488

Using the formula for calculating component gate count that is listed above, an expected worst-case figure for the number of gates in the c6288 circuit variant produced by the component encryption algorithm is provided in Table 4.9. This would be the case in which the output of every component is the XOR function. The mean gate counts of c6288 variants produced by component fusion and component encryption, (25606.33 and 26275.67, respectively) are both significantly lower than this worst-case estimate. The largest c6288 variant produced, consisting of 42856 gates, is the closest to this estimate that any of the variants came.

4.9 Efficiency Analysis: Algorithm Runtime

Algorithm runtime is a characteristic of circuit variant generation, and not a characteristic of the variant itself. This metric represents the amount of work required

to protect a circuit using an algorithm. Figure 4.23 shows a comparison of the trial runtimes of each algorithm.

4.9.1 Subcircuit Selection-and-replacement. Random subcircuit selection-and-replacement requires time for (1.) selecting a subcircuit, and (2.) selecting a replacement. Time required for random subcircuit selection is usually small because it effectively only involves removal of the subcircuit gates. Replacement, however, requires querying a library of subcircuit replacements in order to retrieve a semantically equivalent subcircuit. It is unknown how to predict the number of semantically equivalent subcircuits there are for a particular circuit, but it is possible to predict the number of possible subcircuits that there are of a given size; the growth of this number was characterized by Simonaire to be hyper-factorial [39]. For the c264 circuit in Table 4.4, note that for an equivalent number of iterations (1000 iterations), select-2 replace-3 requires about one-eighth the runtime of the select-2 replace-4 algorithm. In a separate experiment, a single trial of select-3 replace-4 was attempted on c264. 100 iterations of this algorithm required 81718 seconds, more than 336 times the 243 seconds required for 500 iterations of select-2 replace-3 obfuscation. It is pre-

Table 4.9: Expected worst-case gate count for components in the c6288 circuit (Execution of component encryption)

Count	**Expected Size**	**Expected inverters**
13	13.32	2
14	113.83	4.33
14	319.80	5.66
13	981.12	7
1	13.32	2
1	10.66	2
1	286.18	5.66
1	134.62	4.66
182	353.432	5.66
Weighted Average per component:	293.72	5.42
All components:	70493.75	1301.3
Other non-component gates:	241	0
Expected total gate count:	70734.75	1301.3

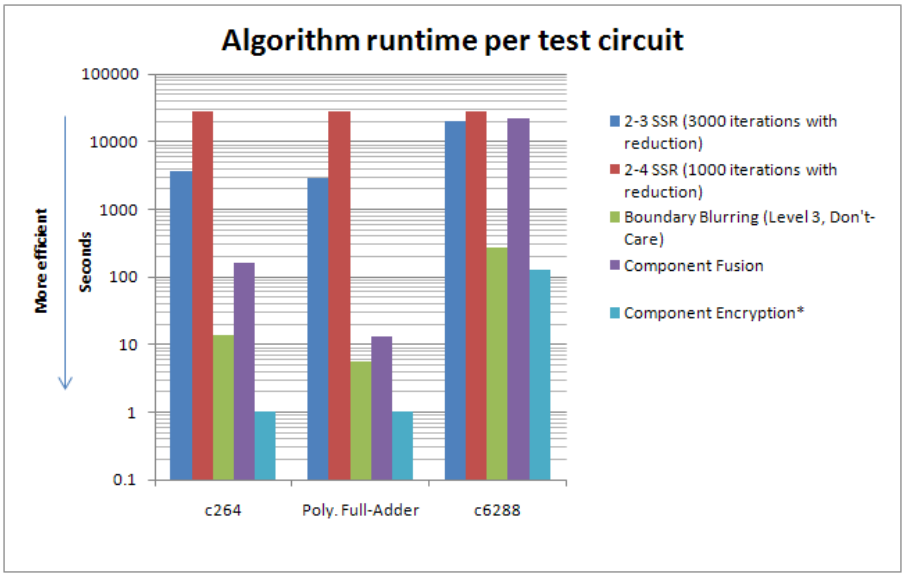


Figure 4.23: Runtime: Relative times required for variant generation trials. **Minimum measurement was 1 second.*

dicted that algorithms with selection-and-replacement at sizes greater than select-3, replace-4 would require even longer to execute.

4.9.2 Boundary Blurring. Boundary blurring exhibited significantly shorter runtime than subcircuit selection-and-replacement in these trials, but the runtime of SSR is dependent upon the number of iterations chosen (an experiment variable) while the runtime of boundary blurring algorithms is dependent solely upon the number of boundaries in a test circuit (a constant). For the c264 circuit, select-2, replace-3 SSR took 262 times as long as boundary blurring, and select-2, replace-4 took 2061 times as long.

The execution of boundary blurring in these trials had a runtime that increased approximately linearly (in these test results) with relation to gate count. Despite this correlation, the runtime of boundary blurring most likely dependent upon the number of component boundaries in the circuit, which increases at about the same rate as the number of gates in these test circuits. If more test cases were run, it is likely that boundary blurring algorithms would exhibit runtime linearly related to the number

of boundaries available to blur (and that the number of boundaries would increase on the same order as the number of gates if component size remained constant).

4.9.3 Component Fusion. Component fusion required only a small runtime for the polymorphic full-adder circuit (60 gates), but runtime increased with seemingly exponential growth, increasing runtime 1688 times when executed on c6288 while gate count increased relatively by only 48.8 times. In fact, component fusion required longer to execute on the c6288 circuit than 3000 iterations of select-2, replace-3 SSR did. This is likely due to overhead in the graph library used for implementation because the component fusion algorithm requires a workload in the worst case which is related linearly to the number of components in the circuit and the number of gates not in a component.

4.9.4 Component Encryption. Component encryption required the smallest runtime of all obfuscation algorithms tested. This is probably because the primary graph operation required was insertion and connection of gates, and few large queries on the graph data structure were required. All components were extracted from the circuit, encrypted, and then all components were inserted into the new circuit in succession and reconnected.

4.9.5 Summary of Circuit Algorithm Analysis. To summarize, component encryption consistently ran in shorter time than any other algorithm, and component fusion ran faster than subcircuit selection-and-replacement, but only for small circuits.

Component encryption and component fusion tend to produce large circuit variants due to inefficiencies in the circuit synthesis algorithm, which is a modified two-level synthesis algorithm. Variants of c6288 produced by these algorithms consistently contain more gates than variants produced by other algorithms, making component identification difficult.

Boundary blurring and select-2 replace-with-4 SSR consistently add more levels to circuit variants than either component fusion or component encryption do,

despite component fusion (CF) and component encryption (CE) algorithms adding more gates to circuit variants. This means that, on average, variants produced by CF and CE will exhibit less overall circuit delay but will require more power and area for implementation than variants produced by Boundary blurring and SSR.

Boundary blurring prevents component identification in every trial where component identification could be performed on the circuit variant. CF and CE tend to do this as well, improving 37% over select-2 replace-4 SSR, but CF seems to introduce gate patterns that cause 2-1 multiplexors to be identified in the circuit; the implementation of CE does not prevent bad mappings, so occasionally components on the input boundary are identified.

Lastly, signal analysis reveals that CF and CE outperform boundary blurring in creating variants that hide the internal signals of circuits. Both algorithms provide at least 48% signal hiding over all three test cases. This has implications for preventing both structural analysis and side-channel analysis.

4.10 Chapter Summary

This chapter has presented metrics for evaluating circuit variant generation algorithms which measure the efficiency and security properties of circuit variants. Then, this research described experimental setup for examining four circuit variant generators using three test circuits composed of known components for the purpose of measuring the effects of each algorithm on circuits with internal components.

One circuit variant generator proposed by this thesis (component encryption) consistently outperformed all other generators in terms of required runtime. Both new circuit generators will likely produce variants with larger gate counts than the other generators once the size of internal circuit components passes a certain threshold, but produce variants with smaller level counts than boundary blurring variants and select-2 replace-4 variants. Variants produced by deterministic selection-and-replacement occasionally contain multiplexor definitions which were not originally in the circuit,

which is bad from the adversary's viewpoint if it suggests that there are components in the circuit that were not originally there.

Overall, there is evidence to support the hypothesis that component encryption is more efficient (in terms of generation time and level count) and more secure in terms of signal hiding than almost all other algorithms. If this algorithm were modified to filter out bad signal mappings, it would most likely also be more secure in terms of component hiding than boundary blurring is.

The next chapter will provide conclusions from this thesis and present suggestions for future work.

V. Conclusions and Future Work

In summary, this thesis has examined questions regarding the security properties of random circuit variant generation and the feasibility of deterministic circuit variant generation. Section 5.1 provides conclusions drawn from this research effort and Section 5.2 examines future research opportunities that stem from this thesis.

5.1 Conclusions

In conclusion, this thesis had two goals, one primary and one secondary. These goals, respectively, were to:

1. Describe more efficient, more secure methods for whitebox circuit obfuscation and metrics for characterizing the increased efficiency and security of the variants which these methods produce, and analyze the methods described in this thesis to evaluate whether the new methods provide security and efficiency improvements over previously described circuit obfuscation methods.
2. Examine whether subcircuit selection-and-replacement achieves the security that a Random Program Model obfuscator would ideally achieve.

5.1.1 Construction of a secure, efficient method for generating logic circuit variants. The first goal (construction of a method for creating combinational logic circuit variants) was met by constructing two deterministic methods for circuit variant generation, one of which (component encryption) was efficient and deterministic. Component encryption performs non-semantics-preserving circuit transformations on circuitry but preserves the semantics of the circuit overall. Component fusion adheres to the model of subcircuit selection and replacement but does so in a non-random way. Both methods serve as useful case studies in the use of deterministic rather than random circuit generation.

Accomplishing the first goal required analyzing the security and efficiency properties of both new methods for variant generation. Metrics were described for the purpose of examining the generation of the variants in this thesis. In describing

new methods for generation of circuit variants and describing metrics for measuring security and efficiency, the first research goal was met.

5.1.2 Analysis of circuit variant generation methods. To summarize, new obfuscation methods for circuit protection were developed in this thesis. Security metrics analyzed were

1. **Signal Hiding**: Both algorithms hide a significant number of internal circuit signals, which improves over both the select-2 replace-3 and the Don't-Care boundary blurring algorithm.
2. **Component Hiding**: Both component fusion and component encryption hide most circuit components. The component identifier identifies spurious (and possibly incompletely hidden) 2-1 and 4-1 multiplexors in variants produced by component fusion, and component encryption has the potential to incompletely hide a component due to the use of a bad mapping.

Efficiency metrics analyzed were

1. **Relative circuit delay in terms of level count**: Both component fusion and component encryption are more efficient in terms of circuit delay than other existing obfuscation methods offering comparable security.
2. **Relative power and area requirements in terms of gate count**: Both component encryption and component fusion do not increase efficiency in terms of gate count. However, this may be acceptable given the benefit of increased security that both methods provide.
3. **Relative algorithm runtime**: The runtime of component encryption improves over the runtime of all other algorithms tested. Component fusion also runs quickly for small circuits.

In analyzing the circuit variant generation methods, the first goal was met. This confirms that the methods created as the first research goal were in fact more secure and more efficient.

5.1.3 Examining the properties of SSR against those of an ideal random circuit variant generator. In Appendix B, this thesis has also examined the possibility that random subcircuit-selection-and-replacement could transform a circuit into any other possible equivalent circuit. A complete set of operations, BAIC, was identified and *could* transform a circuit into any other equivalent circuit, but current experimental models for SSR do not provide completeness because certain operations necessary for completion are not provided by random SSR. In performing this examination, the secondary goal of this thesis was met.

It can thus be concluded that because research goals were successfully achieved, all major research goals were successfully accomplished.

5.1.4 Contributions. This thesis has built upon the subcircuit selection and replacement obfuscation method developed by Norman and James, and evaluates the boundary blurring technique developed by Parham. This research was not the first to use deterministic methodologies to protect circuits; Parham's boundary blurring technique deterministically targeted internal circuit information and deterministically protected this internal information using boundary blurring. As shown in Figure 5.1, the research documented in this thesis is the first to propose semantics-changing obfuscation as a technique, however. Further, this research is the first to identify the separation of deterministic methodologies from random methodologies. Both of the future goals of this research effort are expanded upon in Section 5.2.

5.2 Future research areas

5.2.1 Design and construction of a more robust component encryption algorithm. There are at least two ways in which the component encryption algorithm could be made more secure. The first proposed method for improving the strength of component encryption, filtering insecure mappings, fixes a known weakness. The second proposed method introduces reliance on external signals to increase switch-

ing activity. Using additional randomness will help to improve the strength of an otherwise purely combinational encryption algorithm.

5.2.1.1 Filtering of insecure mappings. The component encryption algorithm implemented in this thesis allows any randomly generated mapping to change the signal values that travel between components. However, many of these mappings allow the original component to be identified because its outputs contain the signals of the original component. An improved component encryption algorithm would check for insecure signal mappings and would not use these mappings to protect component signals.

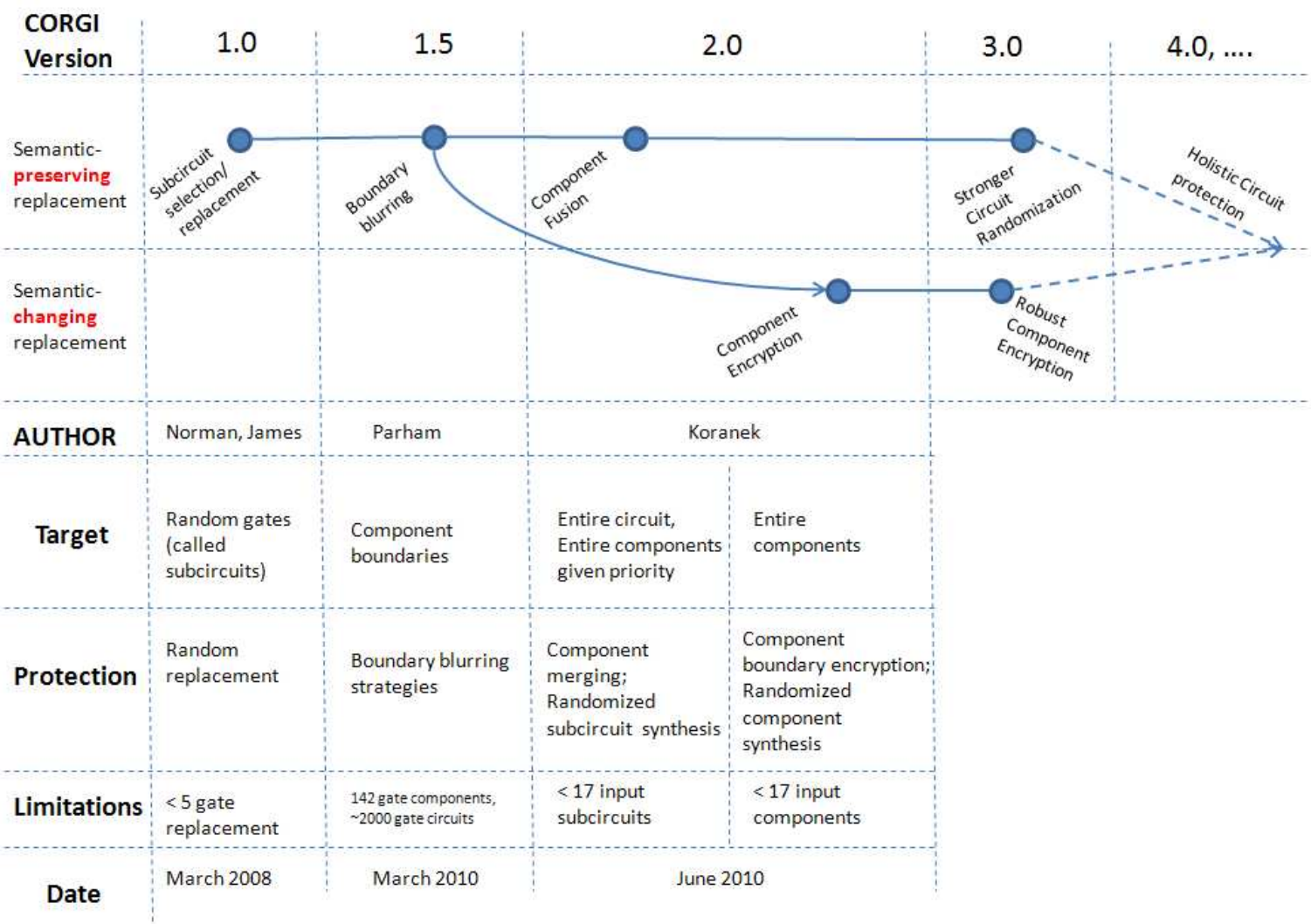


Figure 5.1: Generations of Program Encryption Group (PEG) research on circuit obfuscation.

5.2.1.2 Use of external signals to facilitate switching activity. Occasionally, a mapping function f will map a signal value v onto several new signal values $v'_1, v'_2, \ldots v_n$, but v is never produced or v is used fewer than n times in the original component truth table. To help with the case where a value is used fewer than n times in the original truth table, a signal external to the component component could be used to randomly change v between values that the decoder circuit implementing D_f will decode correctly. This would increase the switching activity between internal components and would serve to further alter the semantics of each component.

5.2.2 Design and construction of a more robust component ID algorithm. The component encryption algorithm described in this thesis synthesizes components which use any randomly generated mapping, even ones which have been identified as insecure. However, the component ID tool developed by Parham is not constructed to identify encrypted components, or even insecurely encrypted ones. A useful modification to this tool would enable it to identify candidate components whose input and output count have been altered.

This tool could use a set of predefined decoder functions to examine combinations of component outputs which could have been insecurely encoded. The decoder functions could include all one-to-one or one-to-two signal mappings, as both sets of mappings are easily enumerable.

Overall, by implementing such a tool, a significant advance will be made in knowledge regarding the capabilities of a component identification adversary.

5.2.3 Implementation of better circuit minimization algorithms. Variants produced by the component fusion and component encryption algorithms in this thesis suffer from the overhead of implementing subcircuit replacements generated by a two-level minimizer using two-input gates. A two-level minimizer produces subcircuits with gates taking possibly any number of inputs (e.g, a two-level minimizer could produce a solution using a 15-input `AND` gate, even there is no gate of that type)

because a two-level minimizer targets a PLA implementation. If a better multiple-output optimizer were available for research use, the gate count of the components produced using both the component fusion and component encryption methods would be dramatically reduced.

5.2.4 Design and construction of a circuit variant distance measurement. One of the significant contributions of this thesis was the use of BAIC in evaluating the capabilities of a system for random circuit generation. Section B.4.10 proposes the use of these identities to measure the distance between circuit variants, but this idea is not developed. Future work in this area could consist of designing and implementing a system for determining the distance between equivalent variants. Because an automatic distance measurement would require finding a path of identities that prove the equivalence of two variants, background research in automated theorem provers using a language like Prolog would probably be useful.

5.2.5 Better visualization of circuit security metrics. An automated tool for visualizing the characteristics of circuit variants in relation to the original circuit would make the design of circuit variant generators much easier. Such a tool could have the following features:

1. The ability to display circuits using traditional logic gate symbols.
2. The ability to visualize the identification of components in a circuit variant.
3. The ability to visually compare identical signals in two circuits.
4. A system for tracking manipulations between circuit versions.

Appendix A. Algorithm Benchmarks

This section contains the results of three experiments which are not mentioned in the main body of this thesis. The experiments are:

1. Running select-3-gates, replace-with-4-gates subcircuit selection-and-replacement on the c264 4-bit multiplier. The results of this experiment are shown in Table A.1.
2. Running select-3-connected-gates, replace-with-4-gates subcircuit selection-and-replacement on the c264 4-bit multiplier. The results of this experiment are shown in Table A.2.
3. Running select-3-connected-gates, replace-with-4-gates subcircuit selection-and-replacement on the polymorphic full-adder circuit. The results of this experiment are shown in Table A.3.

Table A.1: Metrics of select-3, replace-with-4 replacement on the c264 circuit (1 trial)

Metrics/Iterations	**25**	**50**	**100**
Original gates	124.0	124.0	124.0
Gates in variant	128.0	135.0	139.0
Gates increase	4.0	11.0	15.0
Unreplaced Gates	71.0	43.0	30.0
Inputs	8.0	8.0	8.0
Outputs	8.0	8.0	8.0
Original level count	34.0	34.0	34.0
New level count	34.0	35.0	35.0
Levels increase	0.0	1.0	1.0
Unique original signals in new	124.0	124.0	124.0
Unique original signals inverted in new	20.0	25.0	25.0
New signals copied from original signals	127.0	129.0	132.0
New signals copied from inverted original signals	22.0	27.0	28.0
Signals preserved	100.0%	100.0%	100.0%
New gates producing original signals	99.21%	95.55%	94.96%
Full-adders identified ($\leq$ 15 gates)	5	3	2
Half-adders identified ($\leq$ 15 gates)	4	2	2
Runtime (s)	17411	34150	81718

Table A.2: Metrics of select-connected-3, replace-with-4 replacement on the c264 circuit(1 trial)

Metrics/Iterations	**500**	**1000**
Original gates	124.0	124.0
Gates in variant	426.0	747.0
Gates increase	302.0	623.0
Unreplaced Gates	16.0	13.0
Inputs	8.0	8.0
Outputs	8.0	8.0
Original level count	34.0	34.0
New level count	168.0	276.0
Levels increase	134.0	242.0
Unique original signals in new	116.0	86.0
Unique original signals inverted in new	68.0	65.0
New signals copied from original signals	225.0	278.0
New signals copied from inverted original signals	148.0	221.0
Signals preserved	93.54%	69.35%
New gates producing original signals	52.81%	37.21%
Full-adders identified ($\leq$ 15 gates)	0	0
Half-adders identified ($\leq$ 15 gates)	0	0
Runtime (s)	49377	98566

Table A.3: Metrics of select-connected-3, replace-with-4 replacement on a polymorphic full-adder(1 trial)

Metrics/Iterations	**500**	**1000**
Original gates	60.0	60.0
Gates in variant	383.0	720.0
Gates increase	323.0	660.0
Unreplaced Gates	2.0	2.0
Inputs	9.0	9.0
Outputs	2.0	2.0
Original level count	13.0	13.0
New level count	100.0	266.0
Levels increase	87.0	253.0
Unique original signals in new	51.0	49.0
Unique original signals inverted in new	46.0	50.0
New signals copied from original signals	76.0	119.0
New signals copied from inverted original signals	82.0	116.0
Signals preserved	85.0%	81.66%
New gates producing original signals	19.84%	16.52%
2-in multiplexors identified ($\leq$ 16 gates)	15	24
4-in multiplexors identified ($\leq$ 16 gates)	1	0
Runtime (s)	33636	56941

Appendix B. Analysis of Subcircuit Selection-and-Replacement

This appendix examines whether subcircuit-selection-and-replacement implements the Random Program Model by comparing its capabilities to those of a system based upon Boolean algebra laws. This system is actually capable of transforming a circuit into any logically equivalent circuit. Further, applications of identities in this system could allow for a metric of distance to be used measuring an empirical distance between logically equivalent circuit variants.

B.1 Identifying the limits of bounded-size SSR

Current implementations of SSR fail to allow larger than a bounded size space of possible replacements due to the infeasibility of enumerating replacement libraries. Under the Random Program Model, measurable white-box security is derived from pure randomness; every P' is an equally probable replacement for P. While conceptually perfect randomness can be achieved on the level of selection/replacement, there has not yet been any proof that selection/replacement has the ability to transform any circuit P into any possible replacement P', assuming that the selection size is not the size of the entire circuit. Furthermore, there is no existing metric to describe whether additional selection/replacement operations provide additional security. Because of this, we seek metrics to help answer the following questions:

- Can subcircuit selection/replacement be used to transform any size P into any other equivalent P'?
- If not, then what limitations does selection/replacement impose on this transformation?
- Regardless of the answer to the first two questions, what measurable distance does a selection/replacement move P' from the original P?

B.2 Desirable characteristics of logic systems

The first question, regarding the capabilities of the SSR transformation system, can be restated using concepts already well-defined in mathematical logic. The Oxford Dictionary of Philosophy [4] defines two terms which describe desirable characteristics of logic systems. These are *soundness* and *completeness*:

Definition 18. ***Interpretation****: Informally, an interpretation of a logical system assigns meaning to formulas and their elements.*

Definition 19. ***Soundness:*** *Informally, a notion of validity (a formula is valid if it is true in all interpretations).*

Definition 20. ***Completeness:*** *Informally, a notion of semantic consequence (a formula is a semantic consequence of a set of formulae if it is true in all interpretations in which they are true.)*

From these definitions, the following informal definitions for soundness and completeness of an algebra are derived:

Definition 21. ***Sound algebras:*** *An algebra is sound if the identities that equate an expression with any other expression preserve the semantics of the expression.*

Definition 22. ***Complete algebras:*** *An algebra is complete if it can be shown that any two semantically equivalent expressions are actually equivalent using identities. (Or, every valid equality can be derived using the axioms of the algebra. [31])*

Soundness is significant because it indicates that any application of identities will not influence the semantics of an expression. Completeness is significant because it indicates that any two semantically equivalent expressions can be demonstrated to be equivalent. This is significant because, since Boolean algebra identities are both sound and complete [31], 1) applying Boolean algebra identities will always yield other semantically equivalent expressions, and 2) there is some sequence of Boolean algebra identities which can be used to transform one Boolean algebra expression into

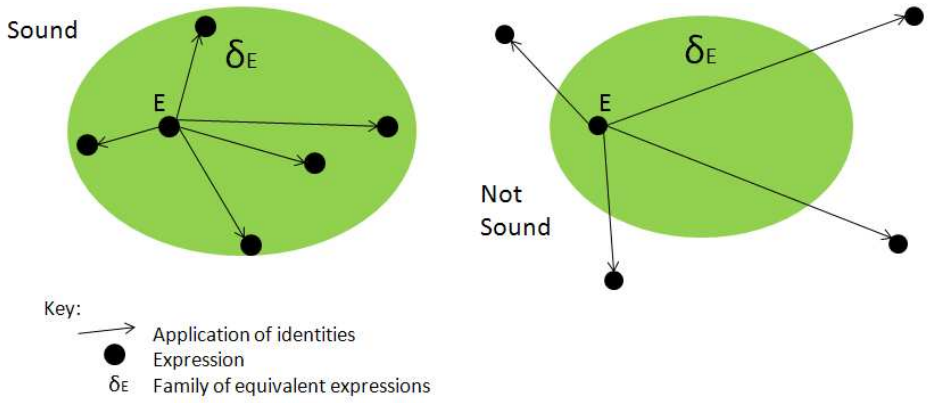


Figure B.1: In a sound algebra, an application of identities will always yield an equivalent expression.

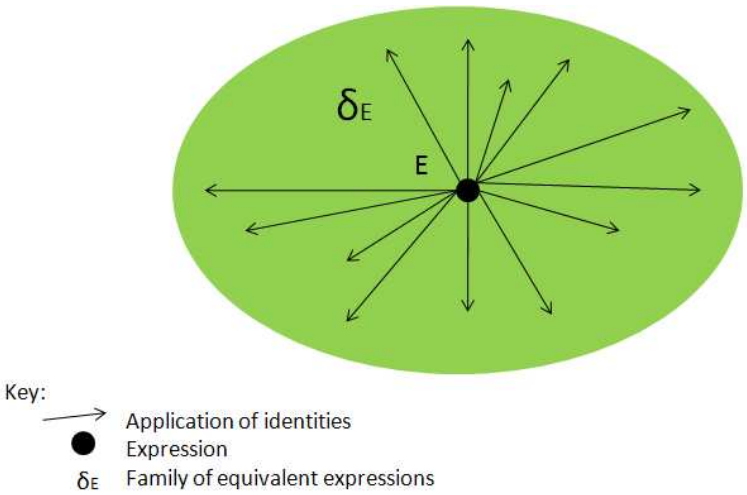


Figure B.2: In a complete algebra, any two equivalent expressions can be demonstrated equivalent using identities (and an expression can be transformed into any equivalent expression using identities).

any other equivalent expression. The concepts of soundness and completeness are illustrated in Figures B.1 and B.2.

Proving the soundness and completeness of subcircuit selection-and-replacement would establish that it is possible to use SSR to obtain from a circuit P any other equivalent circuit P. If, on the other hand, SSR is not sound and complete, then SSR cannot be used to implement the Random Program Model.

B.3 The Soundness and Completeness of Boolean Algebra

As a first step in identifying whether SSR is sound and complete, we examine the soundness and completeness of Boolean algebra, which can be used to express any Boolean Function. For reference, Wolfram MathWorld uses the following identities to define a Boolean algebra: [47]

1. The two operations on a Boolean algebra B, `AND` and `OR`, satisfy the three properties
   - **Idempotence** - $A * A = A + A = A$
   - **Commutativity** - $A * B = B * A$; $A + B = B + A$
   - **Associativity** - $A * (B * C) = (A * B) * C = A * B * C$; $A + (B + C) = (A + B) + C = A + B + C$
2. The `AND` and `OR` operations satisfy the absorption law:
   - **Absorption**: $A * (A + B) = A + (A * B) = A$
3. The `AND` and `OR` operations are mutually distributive:
   - **Distributivity** - $A * (B + C) = (A * B) + (A * C)$; $A + (B * C) = (A + B) * (A + C)$
4. **0 and 1** - Universal bounds (0 or the empty set) and I (1 or the universal set) must exist, s.t.
   - $0 * A = 0$

- $0 + A = A$
- $1 * A = A$
- $1 + A = 1$

5. **Complementation** - A unary operation (the NOT operator or the $'$ operator) must exist, s.t.
   - $A * A' = 0$
   - $A + A' = 1$
   - **Involution** - $(A')' = A$ [17]

Ninomiya and Mukaidono demonstrated the completeness of this algebra with a subset of these identities [31] , as did Huntington [47]. Because Boolean algebra is complete, these identities can be used to prove the equivalence of any Boolean expression with any other equivalent Boolean expression. An additional identity (which can be derived from Boolean algebra identities) known as DeMorgan's theorem is also commonly used to simplify Boolean algebra. DeMorgan's Theorem states that

- $(A + B)' = A'B'$
- $(AB)' = A' + B'$

Functions expressed in the form of Boolean algebra can be transformed into other equivalent expressions through the repeated application of Boolean algebra identities. One common use of identities is to reduce Boolean algebra expressions to a minimal or normal form. Certain identities form the definition of a Boolean algebra itself; other identities are derived from the core ones.

Theorem 1. ***Boolean Algebra identities as a transformation system:*** *Using identities from Boolean algebra, any Boolean algebra expression can be transformed into any other equivalent Boolean algebra expression using Boolean algebra identities.*

Proof: All Boolean algebra expressions implementing the same function can be transformed into an arbitrary normal form (for example, *disjunctive normal form*)

through repeated application of Boolean identities. Imagine that there exist two Boolean expressions F' and F'' implementing the same function F, and a canonical Boolean expression F_{dnf} also implementing F. There exists a sequence of Boolean algebra identities S that transform F' into F_{dnf}, and there exist a sequence of Boolean algebra identities T that transform F'' into F_{dnf}. Because all Boolean algebra identities are equivalences, the sequence T will also transform F_{dnf} into F''. Composing S with T results in a sequence of transformations that transform F' into F''.

This theorem is an additional demonstration of the completeness of Boolean algebra. If identities can transform an expression into some normal form, then identities can transform that expression into any other equivalent expression. Soundness of Boolean algebra can be demonstrated because the system of Boolean algebra identities are all axioms or derived from axioms, which define the domain of Boolean algebra. That is, no application of Boolean algebra identities can yield an expression not equivalent to the original *because equivalence is defined using Boolean algebra identities.* This means transformations based on identities will be sound.

B.3.1 Extending Soundness and Completeness to Digital Logic Circuits. The soundness of Boolean algebra identities have thus been demonstrated. Knowing that subcircuit selection-and-replacement operations are both sound and complete would be useful because this would give credence to the assumption made by Norman that SSR possesses the ability to transform a circuit into any other possible circuit. Figure B.3 provides an illustration of the transformations that a sound and complete SSR system would apply to a circuit - these random transformations would always yield a semantically equivalent circuit, and these transformations could possibly transform P into any other variant P'. For this reason, we seek to map the soundness and completeness of Boolean algebra identities onto the domain of digital logic circuitry. It is possible that some implementations of SSR are sound and complete, but not others. If this is the case, then we seek to discover which implementations of SSR satisfy soundness and completeness and which do not.

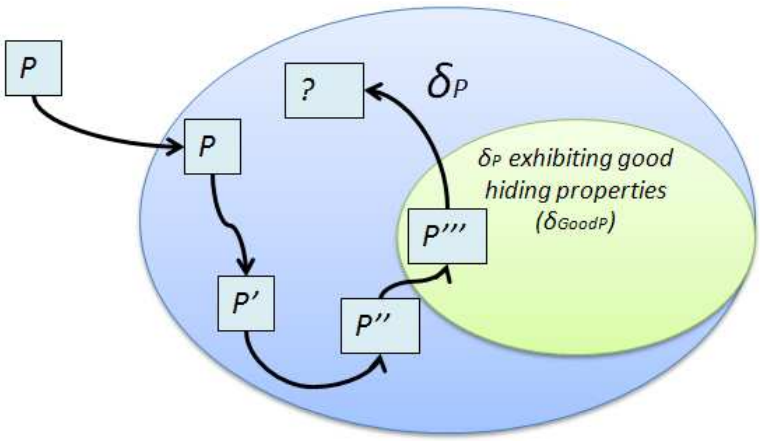


Figure B.3: An ideal circuit manipulator allows incremental manipulations which allow P to be transformed into any circuit in δP.

B.3.2 Mapping Boolean algebra syntax trees to Digital Logic Circuits. In the field of digital logic, the computation of a circuit output is described as a function of the circuit inputs using many functional representations. Functions are described using truth tables (which list the outputs for particular combinations of input values), Karnaugh maps (which map the space of possible output values), Binary Decision Diagrams (with related forms like OBDD, ROBDD, etc.), Boolean algebra expressions, and others.

The parsing of most languages yields a syntax tree such as the one in Figure B.4. This structure can be mapped one-to-one back to the language expression from which it came. Boolean algebra expressions can be similarly parsed into a syntax tree using rules described in Backus-Naur form, as Norman described. [33]

With one operation (defined here as the MERGE operation), a parse tree can be mapped onto a graph structure that directly translates to a one-input digital logic circuit composed of gates with two inputs.

Definition 23. ***MERGE:*** *The MERGE operation takes as input two nodes or logic gates A and B whose semantics are used in the computation of C and D, respectively, and which, upon evaluation, are semantically equivalent. MERGE produces as output*

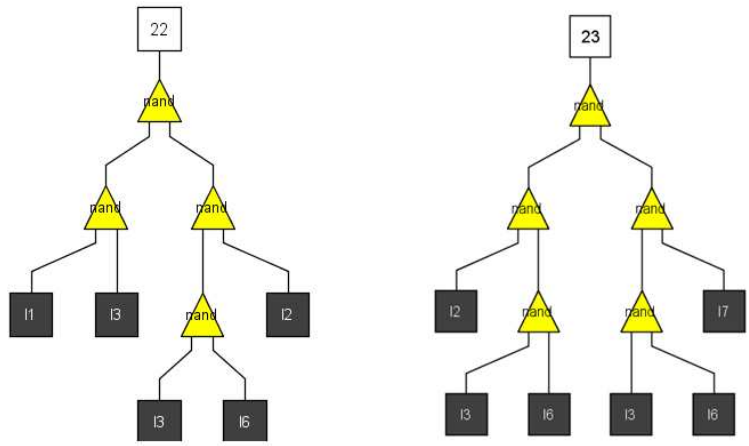


Figure B.4: Syntax trees for expressions computing outputs 22 and 23 of the ISCAS-85 benchmark c17 circuit.

a single node or logic gate E whose value is used in the computation of both C and D.

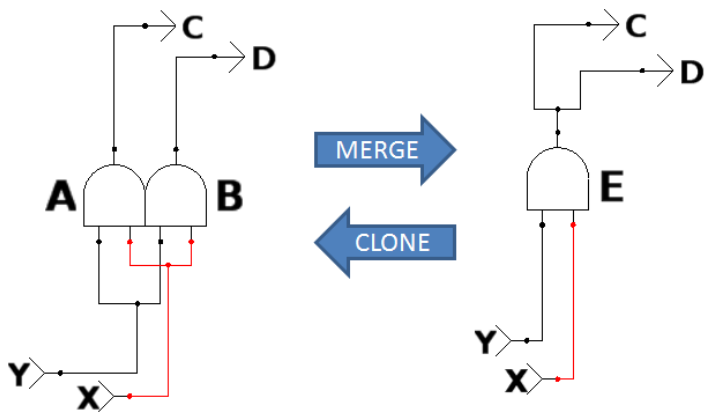


Figure B.5: Complementary operations MERGE and CLONE on an `AND` gate.

To map the parse tree to this intermediate structure, MERGE is applied to all variables with the same name used in multiple places in the parse tree. After applying MERGE, the resulting intermediate structure has the same operator nodes as the parse tree, but only one node representing each variable. Figure B.6 shows the result when the MERGE operation is applied to the Boolean algebra syntax tree in Figure B.4.

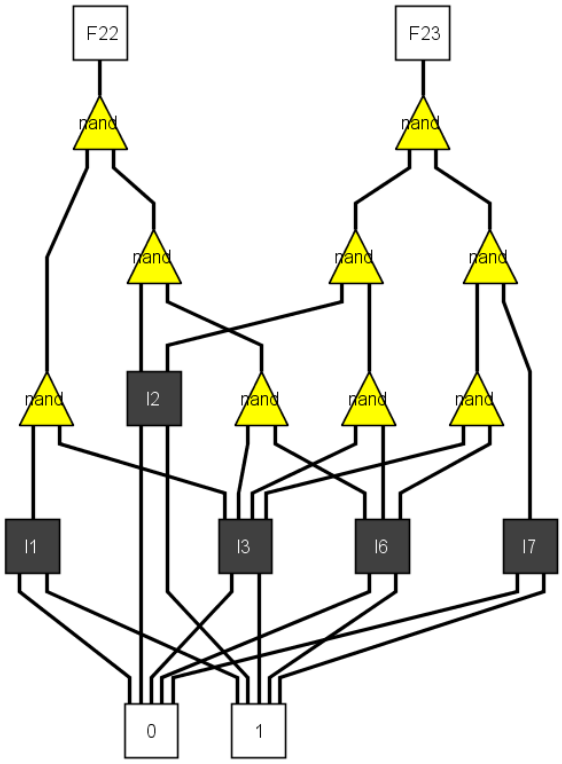


Figure B.6: Syntax trees for outputs 22 and 23 of the c17 circuit from Figure B.4 after the MERGE operation is applied to variable nodes.

This intermediate structure can be mapped directly to a digital logic circuit by translating every operator node to a two-input gate which computes the same function and mapping every variable node to a circuit input.

The inverse of the MERGE operation is the CLONE operation:

Definition 24. ***CLONE:*** *The CLONE operation takes as input a single node or logic gate E whose value is used in the computation of both C and D and outputs two nodes or logic gates A and B, both semantically equivalent, whose values are used in the computation of C and D, respectively.*

The CLONE operation is useful because it allows any combinational logic circuit to be mapped to a parse tree with no values used in more than one computation (in the circuit domain, no logic gates with multiple fanout).

Both CLONE and MERGE are sound because they preserve the semantics of the digital logic circuit in which they operate (they merely reduce or introduce signals which have the same semantics). A graphical depiction of both rules is provided in Figure B.5.

It can thus be seen that

1. A syntax tree can be mapped to a digital logic circuit with two-input gates.
2. A digital logic circuit can be mapped to a Boolean expression syntax tree through repeated applications of the CLONE operation.

B.3.3 A Proof of Completeness of Boolean Algebra Identities and MERGE/-CLONE.

Hypothesis 1. *Completeness of Boolean Algebra Identities and MERGE/CLONE There thus exists a family of transformations, including Boolean algebra identities and the MERGE/CLONE operations whose repeated application allows the transformation of a digital logic circuit into all other possible digital logic circuits implementing the same function.*

Proof: Figure B.7 shows this method for demonstrating equivalences. Using the operation CLONE, a circuit can be transformed into a circuit directly mappable to a collection of parse trees (one for every output) with no signals reused. This circuit can map directly back to a collection of Boolean equations. This collection of Boolean equations can then be transformed through Boolean algebra identities into some arbitrary normal form of those equations.

The operation CLONE is reversible by MERGE, and all Boolean identities are also reversible; thus the sequence of operations which transforms a digital circuit into a collection of canonical Boolean algebra expressions can also be used to transform the canonical Boolean algebra expressions into all digital logic circuits implementing the same functions.

For two equivalent circuits P and Q, P can be transformed into a collection of equations, which can be transformed into a collection of normal forms S. Q can be transformed into a collection of equations, which can also be transformed into a collection of normal forms T identical to S (that is, T *is* S). Because all of the operations used to transform Q were reversible, P can be transformed $S = T$, which can be transformed into any equivalent Q, proving completeness.

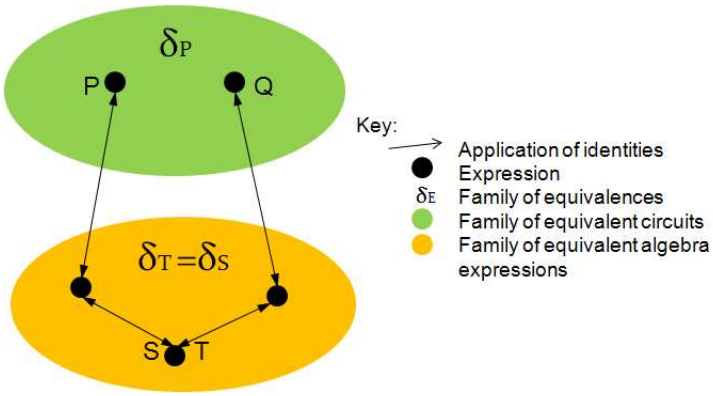


Figure B.7: MERGE/CLONE and Boolean Algebra Identities, together, are complete.

B.3.3.1 Digital Circuit Manipulations derived from Boolean Algebra Identities. Hypothesis 1 describes the fact that digital circuits can be transformed into

any other, equivalent digital circuits *by making use of the Boolean algebra domain.* However, all Boolean algebra identities and the MERGE/CLONE operations have equivalents in the circuit domain which make any mapping to forms outside of the circuit domain unnecessary. This means that a complete set of identities exists within the circuit domain. These identities will be referred to as Boolean Algebra Identities applied to Circuits (BAIC). One good example of a BAIC transformation is a circuit-equivalent form of the law of idempotence. Figure B.8 shows an example of applying idempotence to the circuit which computes the `Sum` signal in a full-adder circuit. In all three of the variants shown in Figure B.8 an `AND` gate takes the same signal twice as input. However, because signals can be shared within a digital logic circuit, circuit idempotence has the potential to introduce cloned signals (signals on which the same value is computed). If we attempt to apply idempotence by inserting an `AND` or `OR` gate that takes as input the same signal, the gates used to compute the source input can be cloned to an arbitrary (and not necessarily uniform) depth.

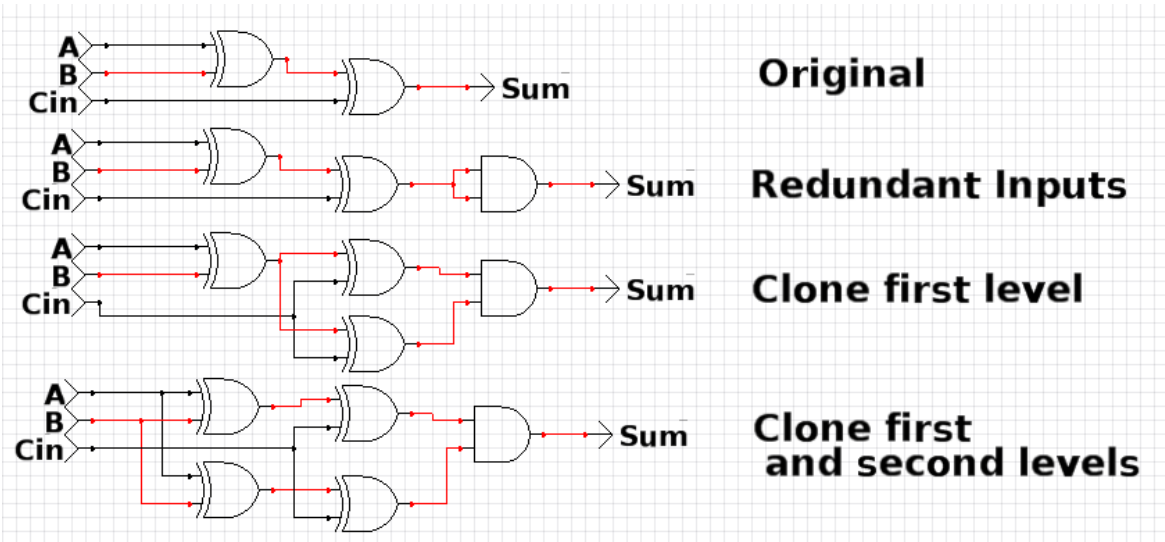


Figure B.8: A circuit computing the `Sum` function and three applications of the idempotent rule on its output signal.

Another example of possible transformations described by Hypothesis 1 is the application of Associativity to circuits. In the circuit domain, this rule is best described as an equivalence of various configurations of the same gate type. Figure B.9

illustrates a set of circuits which can be demonstrated equivalent by applying the Associativity rule to a 4-input AND gate. Other possible rules are illustrated in Figures

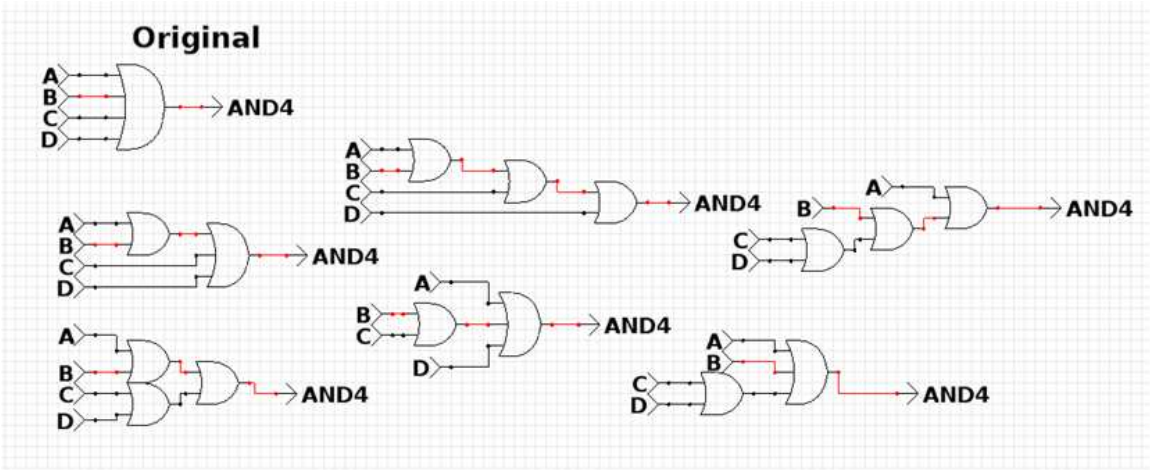


Figure B.9: A 4-input AND gate and six other circuits equivalent by the Associative rule.

B.10 - B.13.

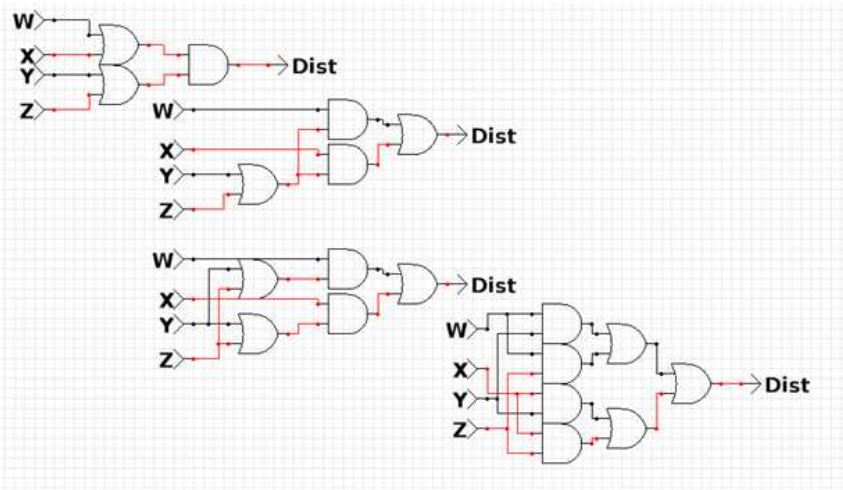


Figure B.10: The circuit implementing (W + X) * (Y + Z) transformed through repeated applications of the Distributive rule.

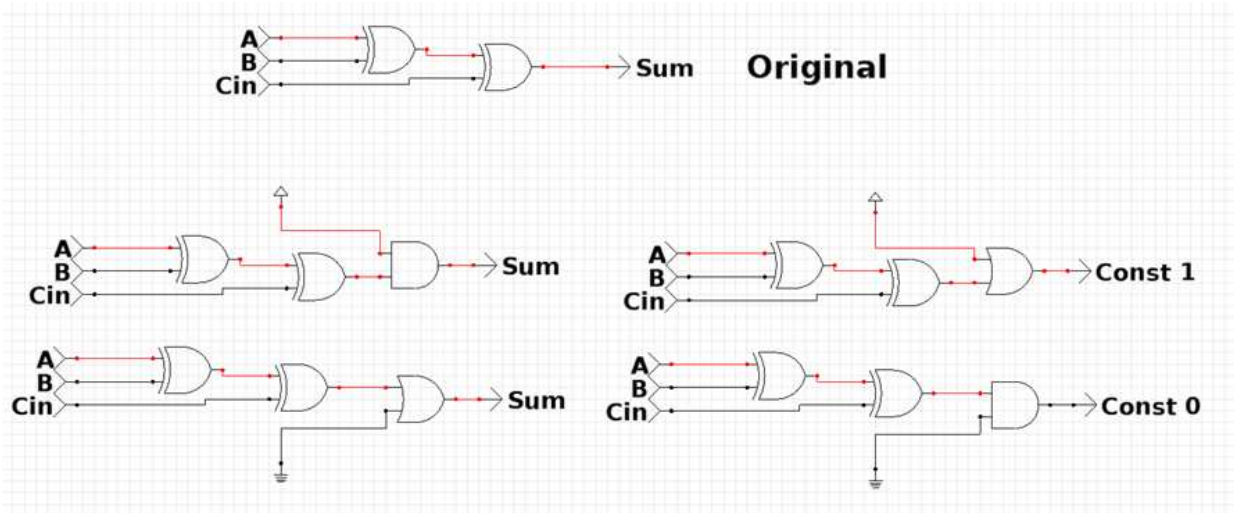


Figure B.11: Circuits illustrating identities involving the 0 and 1 values.

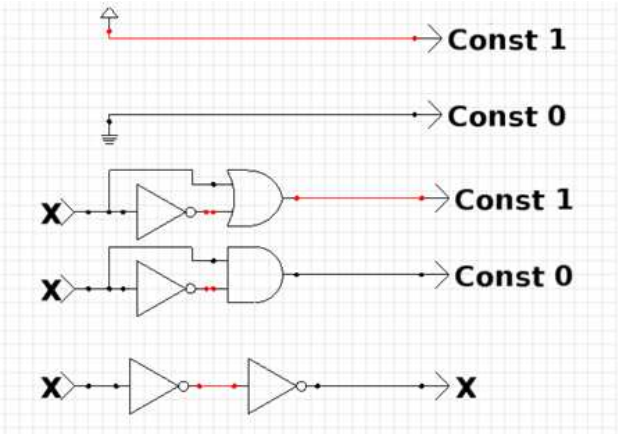


Figure B.12: Two circuits generating constant 1 and 0 signals (respectively), two circuits generating the same two signals (but demonstrating identities of complementation), and a circuit demonstrating the involution property $(X')' = X$.

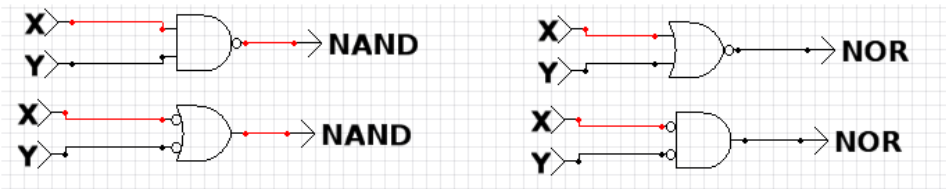


Figure B.13: A 4-input `AND` gate and six other circuits equivalent by the Associative rule.

B.4 Relating Digital Circuit Manipulation to Subcircuit Selection and Replacement

What the previous section describes is a set of circuit equivalences which, if applied, could demonstrate the equivalence of any two equivalent circuit. In Boolean Algebra, several identities are necessary in order to maintain completeness. If these BAIC equivalences are analogous to Boolean algebra identities, then certain identities are likely required to maintain completeness of BAIC identities, as well. Here it is assumed that the BAIC versions of all identities described by Wolfram Research are necessary, and that MERGE/CLONE are necessary.

Each BAIC equivalence requires the selection of a minimum number of gates for its most fundamental application. The numbers required for these equivalences are described in sections B.4.1 through B.4.9.

B.4.1 BAIC Idempotence rule. The minimal form of this rule only requires one-gate selection and two-gate replacement. The inserted gate with redundant inputs will accept as input the output of the single selected gate, so there will be at least two gates in the subcircuit replacement.

B.4.2 BAIC Commutative rule. The minimal form of this rule only requires one gate selection and one-gate replacement. The replaced gate will be exactly the same as the selected gate, but with reordered inputs.

B.4.3 BAIC Associativity rule. The minimal form of this rule requires at least 2-gate selection and one-gate replacement. For this identity to be applied fully, both 2-gate selection with one-gate replacement and one-gate replacement with 2-gate selection must be allowed.

B.4.4 BAIC Absorption rule. The minimal form of this rule requires at least 2-gate selection with 4-gate replacement, and vice versa in order to maintain both possible transformations of the identity.

B.4.5 BAIC Distributive law. The minimal form of this rule requires at least 2-gate selection with 3-gate replacement, and vice versa in order to maintain both possible transformations of the identity.

B.4.6 BAIC Manipulations Involving GND and VDD. Here, it is recognized that a certain, minimum number of gates will be required to generate a signal that will always be 0 or 1. An AND gate coupled with an inverter can generate such a signal, as shown in Figure B.12. Note that Gate propagation delay will often cause the output to momentarily shift when the input changes, but this is logically true. However, the signal used by the AND/inverter pair can be *any* signal at all. This means that, to fully generate the possible identities involving 0 and 1, replacements involving 1) signals not even in the circuit, 2) the GND and VDD signals, or 3) any arbitrarily sized function using inputs within the circuit are all possible. While the SSR method defined by Norman and James does include the use of constant 0 and constant 1 signals as mentioned in Section 4.3, this is disallowed in most experimentation. Bounded-size replacements cap the size of any arbitrarily sized function used by the XOR/inverter pair to compute 1 or 0, so regardless of how large a replacement the selection/replacement algorithm allows, there will always be some function $F_{Gigantic}$ which requires more gates to specify which is impossible for the bounded-size replacement algorithm to generate. No smaller set of operations will be able generate $F_{Gigantic}$ because a smaller set of operations will preserve the semantics of the $F_{Feasible}$ function which is within the range of the selection/replacement algorithm, rather than manipulating the semantics of $F_{Feasible}$ into $F_{Gigantic}$.

B.4.7 BAIC Complementation. One BAIC identity involving complementation (the Involution law) is possible to approximate by inserting or removing double inversion from a circuit. This is possible using one-gate selection and 2-gate replacement, e.g., a NOT-NAND pair is equivalent to an AND gate, or using one-gate selection and 3-gate replacement, e.g., a NOT-NOT-AND is equivalent to an AND gate.

Other identities fall under the limit described in Section B.4.6. For example, to generate an always-1 signal, any function can be ORed with its complement. The generation of any arbitrary-sized function to use as a replacement is bounded by the replacement size.

B.4.8 BAIC DeMorgan's Law. This law requires a selection size which encompasses a gate and all of the gates that generate its inputs (for a 2-input gate, this means 3-gate selection) and an identical replacement size.

B.4.9 BAIC MERGE/CLONE. CLONE requires a 1-gate selection with 2-gate replacement, or (for MERGE) a 2-gate selection with a 1-gate replacement.

B.4.10 Empirically Evaluating the Distance Between P and P'. If it is possible to use individual circuit manipulations to transform a circuit P into any possible variant, P', then any possible selection/replacement operation can be approximated as a sequence of circuit manipulations. In fact, there are probably many unique sequences of selection/replacement circuit operations which can be used to transform P into P'. One open question about circuit randomization using SSR is whether additional circuit variation (through additional SSR iterations) will further protect a circuit. By one reasoning, additional variation makes identification of the precise sequence of variations more difficult. However, additional variation does not necessarily increase the empirical 'distance' between circuit variants.

A metric to measure the distance between equivalent circuit variants could be created from the number of identities used to transform both P and P' into a normal form. In the case of digital logic circuits, the number of BAIC identities required to transform a P into a functionally equivalent P' can provide a sort of distance measurement.

For example, the two circuits shown in Figure B.14 are both random circuits which have four inputs and implement the same function. They can be demonstrated

equivalent through application of rules to transform them into disjunctive normal form.

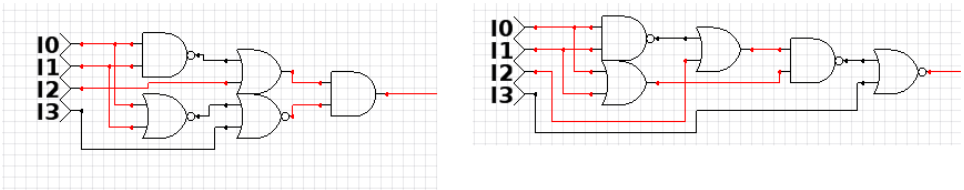


Figure B.14: A 4-input AND gate and six other circuits equivalent by the Associative rule.

Regardless of the number of transformations used to transform P into P', the empirical distance between the two circuits, measured by the combined number of individual identities required to transform both circuits into a normal form, will be the same. If security is measured by the distance between two circuits and all equivalent P are a maximum distance away from the normal form, then at some point, iterative variation which preserves the size of P will provide diminishing security and eventually provide no benefit at all.

To summarize, the number of iterations required to transform P into P' is not a valid measurement of effort required to transform one into the other because the metric for distance between P and P' exists independently of individual effort.

B.5 Conclusions regarding subcircuit selection/replacement

By comparing SSR to a form of canonical digital circuit manipulation, it has been demonstrated that there are several classes of circuits equivalent to any given P which a constant-size selection or constant-size replacement method cannot transform a circuit P into. Figure B.15 illustrates this concept. While a circuit P can be possibly transformed into a subset of the circuits equivalent to P using Random SSR, and a Random SSR method will likely provide some security inherent to randomizing circuit structure, random constant-size selection with constant-size replacement does not provide the security of replacing P with a truly random P'. Furthermore, it is hypothesized that additional random variation is not guaranteed to increase the

empirical distance between the original circuit P and the final circuit variant P'. This means that existing, bounded methods for subcircuit selection and replacement do not implement the Random Program Model, and iterative random variations do not necessarily increase the randomness of a circuit.

There exists an SSR algorithm which is able to transform a circuit into any possible semantically equivalent circuit, but a constant-size-selection, constant-size-replacement implementation of SSR does not allow this type of transformation, even over many iterations. Further, a procedure has been posed (but not empirically demonstrated) to measure the distance between two circuit variants using the number of identities required to transform both into a normal form. Using this procedure as a distance measure, a hypothesis has been made (but not empirically demonstrated) that increasing the number of variant iterations between the original circuit and the final variant does not necessarily increase the distance between the original circuit and the final variant. If this hypothesis is correct, then random variation will yield diminishing security benefit (measured by distance from the original circuit) after some number of iterations.

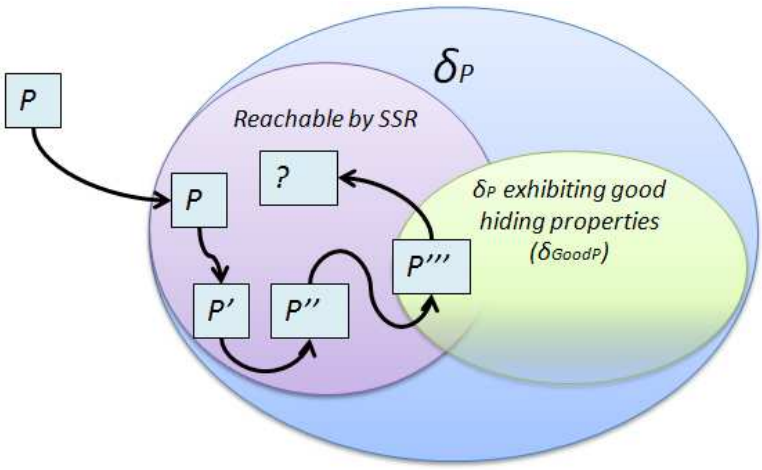


Figure B.15: Subcircuit selection and replacement performs incremental manipulations which allow P to be transformed into a subset of δ_P.

This section contains three algorithms which are used in applying Signal Value Permutation to circuit components by manipulating their truth tables. Algorithm C.1 describes manipulating circuit truth table output rows, Algorithm C.2 describes manipulating circuit truth table input rows, and Algorithm C.3 describes the grouping of signals between components that connect the same components together.

Algorithm C.1 GET-REMAPPED-OUTPUT(boolean [ ][ ] `tt.outputrows`, boolean[ ][ ] `tt.inputrows`, mappings `m`)

The objects ***tt*** *and* ***remapped_tt*** *are both truth tables with i inputs and 2^i rows of input and output values. The number of inputs is the same in both truth tables, but in* ***remapped_tt*** *the number of outputs changes.*

row*,* ***new_row****,* ***original_value*** *and* ***new_value*** *are strings of binary values for convenience.*

Mappings contain a list of ***outputindices*** *(the indices of the outputs it remaps) and a hash mapping a string of binary values to an array of strings of binary values. The hash is accessed using the getReplacementList() method.*

```
    remapped_tt ← boolean [tt.outputrows.size] []
    for all i = 0 to tt.outputrows.size do
      {
      }row ← tt.outputrows [i]
 5:   new_row ← ""
      {}
      for all mapping in m do
        original_value ← ""
        new_value ← ""
10:     for all index in m.outputindices do
          original_value ← originalvalue.row[index]
        end for
        replacement_list ← m.getReplacementList(original_value)
        replacement_index ← Random(0, replacement_list.size − 1)
15:     new_value ← replacement_list [replacement_index]
        new_row ← new_row.new_value
      end for
      remapped_tt.inputrows [i] ← tt.inputrows [i]
      remapped_tt.outputrows [i] ← new_row
20: end for
```

Algorithm C.2 GET-REMAPPED-INPUT(boolean[ ][ ] `tt.outputrows`, boolean[ ][ ] `tt.inputrows`, mappings `m`, Component c)

`tt` and `remapped_tt` both begin as truth tables with i inputs and 2^i rows of input and output values. The number of outputs is the same in both truth tables, but in `remapped_tt` the number of inputs changes, and this causes the number of rows to change.

Mappings contain a list of lists of `inputindices` (the indices of the inputs it remaps, each list indexed by the particular component it is remapping) and a hash mapping a string of binary values to an array of strings of binary values. The hash is accessed using the getReplacementList() method. In addition, mappings contain values `oldsignals` and `newsignals` which indicate the number of signals that the mapping takes in as input and the number of signals contained in all replacement values.

```
remapped_tt ← boolean[ ][ ]
for all mapping in m do
  new_no_inputs ← remapped_tt.noInputs + (mapping.newsignals − mapping.oldsignals)
  expanded_tt.input_rows ← boolean[new_no_inputs][2ñew_no_inputs]
  expanded_tt.output_rows ← boolean[tt.no_outputs][2ñew_no_inputs]
  last_assigned_row ← 0
  for all i = 0 to remapped_tt.inputrows.size do
    row ← tt.inputrows[i]
    equivalent_rows ← boolean[][]
    original_value ← ""
    for all j = 0 to m.inputindices[c.id].size do
      original_value = row[m.inputindices[c.id][j]]
    end forequivalent_rows = mapping.getReplacementList(original_value)
    for all j = 0 to equivalent_rows.size do
      expanded_tt.inputrows[last_assigned_row] ← equivalent_rows[j]
      expanded_tt.outputrows[last_assigned_row] ← tt.outputrows[i]
      last_assigned_row ← last_assigned_row + 1
    end for
  end for
  remapped_tt ← expanded_tt
end for
```

Algorithm C.3 GET-SIGNAL-GROUPINGS(Component c)

```
remaining_outputs ← c.outputs
{}output is an array of arrays of outputs.
output_groupings ← output[][]
while !remaining_outputs.isEmpty() do
  next_output ← remaining_outputs[0]
  remaining_outputs.remove[0]
  {Find the component which uses the signal from next_output and the largest
  number of other signals from c.}
  minfedcomp ← ∅
  minconnections ← −1
  output_set = ∅
  for all fedcomp in c.getOutputComponents() do
    count = 0
    running_output_set = ∅
    for all predecessorGate in fedComp.getPredecessorGates() do
      if c.containsGate(predecessorGate) then
        count = count + 1
        running_output_set.add(predecessorGate)
      end if
    end for
    if count < minconnections||minconnections == −1 then
      minconnections = count
      minfedcomp = fedcomp
      output_set = running_output_set
    end if
  end for
  output_groupings.add(ouput_set)
end while
```

Bibliography

1. Balaban, Dan. "Transport for London to Discard Mifare Classic", January 2010. URL `http://www.nfctimes.com/news/transport-london-discard-mifare-classic-seeks-desfire-sims`.

2. Barak, Boaz, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (im)possibility of obfuscating programs (extended abstract)". *Advances in cryptology—CRYPTO 2001 (Santa Barbara, CA)*, volume 2139 of *Lecture Notes in Comput. Sci.*, 1–18. Springer, Berlin, 2001.

3. Black, Paul E. *"deterministic algorithm"*. U.S. National Institute of Standards and Technology, January 2009. URL `http://www.itl.nist.gov/div897/sqg/dads/HTML/deterministicAlgorithm.html`.

4. Blackburn, Simon. *The Oxford Dictionary of Philosophy: Oxford Paperback Reference*. Oxford; New York Oxford University Press (UK), 1996.

5. Brayton, Robert King, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984. ISBN 0898381649.

6. Brown, Douglas W. "A State-Machine Synthesizer—SMS". *DAC '81: Proceedings of the 18th Design Automation Conference*, 301–305. IEEE Press, Piscataway, NJ, USA, 1981.

7. Cady, Camdon R. *Static and Dynamic Component Obfuscation on Reconfigurable Devices*. Masters, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, March 2010.

8. 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.

9. Collberg, Christian, Clark Thomborson, and Douglas Low. "Manufacturing cheap, resilient, and stealthy opaque constructs". 184 – 196. San Diego, CA, USA, 1998. ISSN 07308566. Code obfuscators;Java programming language;Opaque predicates;.

10. Collberg, Christian S. and Clark Thomborson. "Watermarking, Tamper-Proofing, and Obfuscation-Tools for Software Protection". *IEEE Transactions on Software Engineering*, 28(8):735–746, 2002. ISSN 0098-5589.

11. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-03293-7.

12. Cousot, Patrick and Radhia Cousot. "Systematic design of program transformation frameworks by abstract interpretation". *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 178–190. ACM, New York, NY, USA, 2002. ISBN 1-58113-450-9.

13. Dalla Preda, M., R. Giacobazzi, B.K. Aichernig, and B. Beckert. "Control code obfuscation by abstract interpretation." Dipt. di Informatica, Univ. di Verona Italy, 2005. URL `http://wf2dnvr12.webfeat.org:80/EUfeN1492/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=8732984&site=ehost-live`.

14. Darringer, John A., Daniel Brand, John V. Gerbi, William H. Joyner, and Louise Trevillyan. "LSS: A system for production logic synthesis". *IBM Journal of Research and Development*, 28(5):537 –545, sept. 1984. ISSN 0018-8646.

15. Darringer, John A., William H. Joyner, C. Leonard Berman, and Louise Trevillyan. "Logic synthesis through local transformations". *IBM J. Res. Dev.*, 25(4):272–280, 1981. ISSN 0018-8646.

16. Garey, Michael R. and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

17. Hachtel, Gary D. and Fabio Somenzi. *Logic Synthesis and Verification Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 0387310045.

18. Hansen, Mark C., Hakan Yalcin, and John P. Hayes. "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering". *IEEE Des. Test*, 16(3):72–80, 1999. ISSN 0740-7475.

19. Hex-Rays. "IDA Pro Dissassembler - multi-processor, windows [sic] hosted disassembler and debugger". Website, April 2010. URL `http://www.hex-rays.com/idapro/`.

20. James, Moses. *Obfuscation Framework Based on Functionally Equivalent Combinatorial Logic Families*. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, March 2008.

21. Keller, John. "2010 DOD budget proposes increases for Navy, DARPA spending; Army faces big cuts". Electronic, September 2009. URL `http://www.militaryaerospace.com/index/display/article-display/369172/articles/military-aerospace-electronics/volume-20/issue-9/news/news/2010-dod-budget-proposes-increases-for-navy-darpa-spending-army-faces-big-cuts.html`.

22. Kim, Hanseok. *Removing Redundant Logic Pathways in Polymorphic Circuits*. Master's thesis, Air Force Institute of Technology (AU), March 2009.

23. Kim, Yong C. and J. Todd McDonald. "Considering software protection for embedded systems". *CrossTalk*, 22(9-10):4 – 8, 2009. Hardware description languages;Modern embedded systems;Reconfigurable computing;Software applications;Software protection;.

24. Knuth, Donald E. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 2009. ISBN 0321580508, 9780321580504.

25. Koeppel, Dan. "China's iClone". Electronic, August 2007. URL `http://www.popsci.com/iclone`.

26. Li, Xiao Yu, Matthias F. Stallmann, and Franc Brglez. "Effective bounding techniques for solving unate and binate covering problems". *DAC '05: Proceedings of the 42nd annual Design Automation Conference*, 385–390. ACM, New York, NY, USA, 2005. ISBN 1-59593-058-2.

27. Linn, Cullen and Saumya Debray. "Obfuscation of executable code to improve resistance to static disassembly". *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, 290–299. ACM, New York, NY, USA, 2003. ISBN 1-58113-738-9.

28. McDonald, J. Todd. *Enhanced Security for Mobile Agent Systems*. Doctoral, Florida State University, October 2006.

29. McDonald, J. Todd and Alec Yasinsac. "Of Unicorns and Random Programs". *3rd IASTED International Conference on Communications and Computer Networks (IASTED/CCN)*, October 24-26 2005.

30. McDonald, J. Todd and Alec Yasinsac. "Program Intent Protection Using Circuit Encryption". *Proceedings of 8th International Symposium on Systems and Information Security*. IEEE Computer Society, November 2006.

31. Ninomiya, T. and M. Mukaidono. "Complete and independent sets of axioms of boolean algebra". *Multiple-Valued Logic, 2003. Proceedings. 33rd International Symposium on*, 169 – 174. may 2003. ISSN 0195-623X.

32. Nohl, Karsten, David Evans, Starbug Pltz, and Henryk Pltz. "Reverse-Engineering a Cryptographic RFID Tag". USENIX Security Symposium, July 2008. URL `http://www.cs.virginia.edu/ evans/pubs/usenix08/`.

33. Norman, Kenneth E. *Algorithms for White-box Obfuscation Using Randomized Subcircuit Selection and Replacement*. Master's thesis, Air Force Institute of Technology (AU), March 2008.

34. Online, BBC News. "China paid $34,000 over spy plane". Electronic, August 2001. URL `http://news.bbc.co.uk/2/hi/asia-pacific/1483201.stm`.

35. Parham, James D. *Component Hiding Using Identification and Boundary Blurring Techniques*. Master's thesis, Air Force Institute of Technology, March 2010.

36. Preda, M.D., R. Giacobazzi, L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung. "Semantic-based code obfuscation by abstract interpretation." Dipt. di Informatica, Univ. di Verona Italy, 2005. URL http://wf2dnvr12.webfeat.org:80/EUfeN1509/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=8651146&site=ehost-live.

37. Preda, M.D., M. Madou, K. De Bosschere, R. Giacobazzi, M. Johnson, and V. Vene. "Opaque predicates detection by abstract interpretation." Dept. of Comput. Sci., Verona Univ. Italy, 2006. URL http://wf2dnvr12.webfeat.org:80/EUfeN1552/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=9017996&site=ehost-live.

38. Rekoff, Jr., M.G. "On Reverse Engineering". *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(2):244 – 252, 1985. URL http://wf2dnvr2.webfeat.org:80/QMsTM11594/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=2531943&site=ehost-live.

39. Simonaire, Eric D. *Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems*. Master's thesis, Air Force Institute of Technology (AU), December 2008.

40. Styria, NXP Semiconductors Austria GmbH. "Showcases - London: Easing travel in London's congested public transportation network", 2008. URL http://mifare.net/showcases/london.asp.

41. on U.S. National Security, Select Committee and Military/Commercial Concerns with the People's Republic of China. "HOUSE REPORT 105-851 Chapter 1: PRC Acquisition of U.S. Technology", January 1999. URL http://www.gpo.gov/congress/house/hr105851-html/ch1bod.html.

42. Vahid, F. "It's Time to Stop Calling Circuits "Hardware"". *Computer*, 40(9):106 –108, sept. 2007. ISSN 0018-9162.

43. Varnovsky, N.P., V.A. Zakharov, M. Broy, and A.V. Zamulin. "On the possibility of provably secure obfuscating programs." Lab. for Cryptography, Lomonosov State Univ., Moscow Russia, 2003. URL http://wf2dnvr12.webfeat.org:80/EUfeN1598/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=8116860&site=ehost-live.

44. Verton, Dan. "Spy plane incident raises concerns over access to secret U.S. technology". Electronic, April 2001. URL http://www.computerworld.com/s/article/59203/Spy_plane_incident_raises_concerns_over_ access_to_secret_U.S._technology.

45. Walenstein, A., R. Mathur, M.R. Chouchane, and A. Lakhotia. "Normalizing Metamorphic Malware Using Term Rewriting". *Source Code Analysis and Manipulation, 2006. SCAM '06. Sixth IEEE International Workshop on*, 75–84. Sept. 2006.

46. Wee, Hoeteck. "On obfuscating point functions". *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, 523–532. ACM, New York, NY, USA, 2005. ISBN 1-58113-960-8.

47. Weisstein, Eric W. "Boolean Algebra". *MathWorld - A Wolfram Web Resource*. Wolfram Research, Inc., 2010. URL `http://mathworld.wolfram.com/BooleanAlgebra.html`.

48. White, Jennifer L., Anthony S. Wojcik, Moon-Jung Chung, and Travis E. Doom. "Candidate subcircuits for functional module identification in logic circuits". 34 – 38. Chicago, IL, USA, 2000. ISSN 10661395. Functional module identification;.

49. White, Jennifer Lynn. *Candidate Subcircuit Enumeration for Module Identification in Digital Circuits*. Ph.D. thesis, Michigan State University, 2000.

50. 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." Dept. of Comput. Sci. & Eng., Michigan State Univ., East Lansing, MI USA, 2001. URL `http://wf2dnvr12.webfeat.org:80/EUfeN1625/url=http://search.ebscohost.com/login.aspx?direct=true&db=inh&AN=7087240&site=ehost-live`.

51. Williams, Jason A. *Characterizing Component Hiding Using Ancestral Entropy*. Master's thesis, Air Force Institute of Technology (AU), March 2009.

52. Wirth, N. "Hardware compilation: translating programs into circuits". *Computer*, 31(6):25 –31, jun 1998. ISSN 0018-9162.

53. Yuschuk, Oleh. "OllyDbg v1.10". Website, 2009. URL `http://www.ollydbg.de/`.

Vita

Daniel F. Koranek was homeschooled through high school by his parents, and graduated from high school in 2004. In 2008, he earned his undergraduate degree in Computer Science from Cedarville University in Cedarville, OH. Upon graduation from Cedarville, he was accepted into the Cyber Corps Fellowship under the Scholarship for Service program and began graduate work at the the Air Force Institute of Technology (AFIT). After graduating from AFIT, Daniel will be employed full-time by the Air Force Research Laboratory on the Wright-Patterson Air Force Base in Dayton, Ohio.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
10–06–2010	Master's Thesis	Sept 2008 — June 2010

4. TITLE AND SUBTITLE

Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Daniel F. Koranek, Civ.

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 (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCO/ENG/10-15

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

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research proposes two alternative methods for generating semantically equivalent circuit variants which leave the circuit's internal structure pseudo-randomly determined. Component fusion deterministically selects subcircuits using a component identification algorithm and replaces them using a deterministic algorithm that generates canonical logic forms. Component encryption seeks to alter the semantics of individual circuit components using an encoding function, but preserves the overall circuit semantics by decoding signal values later in the circuit.
Experiments were conducted to examine the performance of component fusion and component encryption against representative trials of subcircuit selection-and-replacement and Boundary Blurring, two previously defined methods for circuit obfuscation. Overall, results support the conclusion that both component fusion and component encryption generate more secure variants than previous methods and that these variants are more efficient in terms of required circuit delay and the power and area required for their implementation.

15. SUBJECT TERMS

software protection, deterministic methods, reverse engineering, circuit obfuscation

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 C. Kim
U	U	U	UU	163	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x 4620

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18