A Java Based Component Identification Tool for Measuring the Strength of Circuit Protections *

[Extended Abstract]

James D. Parham
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
james.parham.2@us.af.mil

J. Todd McDonald
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
jmcdonal@afit.edu

Michael R. Grimaila
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
mgrimail@afit.edu

Yong C. Kim
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
ykim@afit.edu

ABSTRACT

Protecting circuitry from reverse engineering is extremely important for the protection of intellectual property and critical technologies. A failure to adequately mitigate reverse engineering risks can result in significant consequences. In commercial environments, the consequences include loss of revenue resulting from the removal of content access restrictions, creation of unlicensed copies of a circuit, and intellectual property theft. In military environments, consequences include an adversary's ability to gain valuable knowledge about the structure, function, and operation of the system enabling them to develop countermeasures to defeat or corrupt the system's intended purpose. When critical systems are compromised, organizations may be required to spend millions of dollars and countless labor hours making changes in the system and redesigning new protection circuitry. While there are multiple protection algorithms and schemes which claim to provide protection against reverse engineering, little research exists on developing methods to measure the effectiveness of protections. Component identification is an essential step of the reverse engineering process of circuits which are increasingly being embedded in modern systems. The task of component identification is not trivial and requires significant effort even for relatively small circuits. For this reason, computer tools are often employed to make the analysis of larger circuits possible. In this paper, we discuss the development and implementation of a Java based tool that can be used to identify components in combinational circuits. The use of a component identification tool provides metrics that can be used to differentially evaluate the"strength" of protections within circuits. We introduce the foundational candidate enumeration algorithm, explain the additional techniques required for identifying components in large Boolean circuits, and demonstrate the utility of the method through the analysis of multiple combinational circuits.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-aided design (CAD)

General Terms

Component identification, Circuit protection, Reverse engineering

1. INTRODUCTION

Reverse Engineering (RE) is the scientific process of developing an understanding of the underlying technical principles of a system through analysis of its structure, function and operation. RE is conducted for a wide variety of legitimate purposes (e.g., modeling, identify opportunities for cost reduction, recreate lost technical documentation, security analysis, education) and illegitimate purposes (e.g., theft of intellectual property, circumventing content protection, creating unlicensed copies, espionage). Protecting circuitry reverse engineering is extremely important in both commercial and military environments. While the conse-

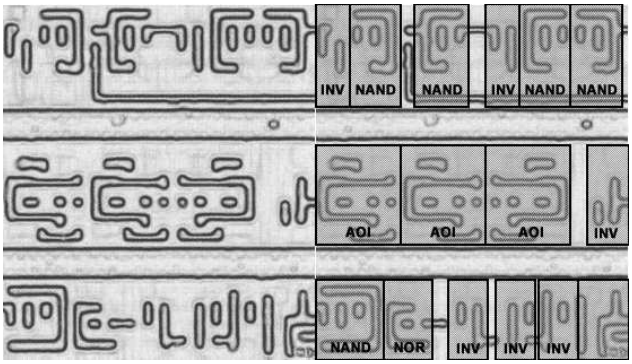

Figure 1: Exposed transistors of the Mifare RFID are shown on the left and results after gate identification are shown on the right [5].

quences resulting from failing to protect from RE in commercial environments are typically characterized in terms of the loss of revenue, in military environments the consequences can be much higher including loss of critical resources and the loss of life.

Without implementing some form of protection, an adversary may easily gain access to sensitive information contained within the circuit with disasterous results. For example, the U.S. National Security Agency has authorized the Advanced Encryption Standard (AES) for encrypting classified information on Unmanned Aerial Systems (UAS). Without providing protection to AES circuitry, adversaries could possibly obtain secret keys enabling interception of the data link to substitute their own video feeds or take control of a UAS [1].

Billions of dollars are spent each year developing technologies. The U.S. Department of Defense Fiscal Year 2009 budget request for procurement research and development was in excess of $183 billion for modernizing and meeting future threats [6]. Adversaries gain access to technologies faster and cheaper when they reverse engineer system circuitry.

Component identification is one computer aided method used by reverse engineers for discovering key circuit elements [3]. Nohl et al. used this technique when revealing the cryptographic cypher of the Mifare Classic Radio Frequency Identification (RFID) tag. They first exposed circuit transistors enabling identification of gate structures and the connections between them. Figure 1 shows a small section of exposed transistors and the results of gate identification. Then they identified the area of the circuit chip containing cryptographic components. Finally, they discovered the cryptographic keys making it possible to access the Dutch transit system where the use of these cards was intended. When they testified before the Dutch government detailing their accomplishments, approximately $2 billion had been invested in the ticketing system [5].

Our previous research focused on creating random circuit variants and used randomness as a proxy for an obfuscation/protection metric [4]. However, we have recently started to explore the the use of the operational parameters (e.g., runtime, memory utilization, convergence) of component identification tools as measures of the effectiveness of circuit protection in combinational circuits. In pursuit of these objectives, we chose to implement a Java based component identification tool. In this paper, we present the development of a component identification tool, discuss foundational algorithms for candidate component identification, introduce a method of candidate equivalence checking, and demonstrate the utility of the method through the analysis of multiple versions of a 16-bit multiplier.

2. MODELING BOOLEAN CIRCUITS

We model combinational logic at the gate level as Boolean circuits using Directed Acyclic Graphs (DAG) where each vertex represents a logic gate and each edge represents the connection between them. Directed graphs are used since flow occurs only in one direction from the output of a gate to the input of one or more gates. Shaped vertices allow visual identification of gate types as well as inputs and outputs. Figure 2 shows an example circuit graph for a five input two output circuit.

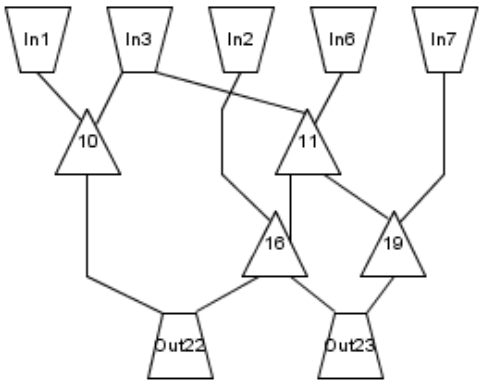

Figure 2: An five input two output example circuit graph.

3. CANDIDATE ENUMERATION

A fully connected graph has $n!$ subgraphs where n is the number of vertices. A fully connected graph is highly unlikely, but serves as an upper bound for the number of possible sub circuits [7]. This shows that even small circuit graphs contain an intractable number of subgraphs. White [8] details a candidate subcircuit enumeration algorithm with runtime $O(n^3)$. Since no source code implementation of the algorithm was available, we implemented their algorithm in Java. The component candidate enumeration algorithm is based upon three rules:

1. **Unique Enumeration** - No subgraph is created more than once.
2. **Fully Specified Candidate** - A subgraph must contain either all or no predecessors for every vertex in the graph.
3. **Fully Contained Candidate** - A subgraph must contain either all or no successors for every vertex in the graph.

When rule two and rule three are satisfied the candidate subcircuit is considered fully contained and is valid for equivalence checking. In Figure 3 we show examples of rule two and rule three violations. Vertex five violates rule two because only one of its predecessors, vertex four, is contained in the subgraph. Vertex four violates rule three because only one of its successors, vertex five, is contained in the subgraph.

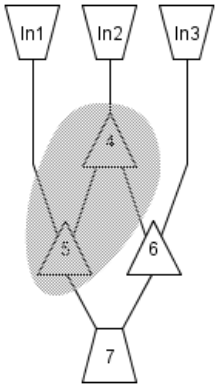

Figure 3: The highlighted subgraph contains two vertices each violating a single rule. Vertex five violates rule two and vertex four violates rule three.

4. EQUIVALENCE CHECKING

When our tool identifies fully contained candidates, we perform equivalence checking before accepting it as a valid circuit component. We accomplish this with truth table analysis between the candidate component and a known library component. Our first check is the input and output (I/O) space of the candidate. If a known library module exists with the same I/O space we perform the truth table analysis. If no library module exists with matching I/O space we discard the candidate. I/O ordering effects the analysis so we permute I/O order comparing all possible combinations. Equivalence checking has a runtime of $O(n!m!)$ where n is the number of inputs and m is the number of outputs. We limit equivalence checking to candidates not exceeding eight inputs or eight outputs because of the runtime, memory, and performance limitations. For best component identification performance, the known library should contain only components of interest during candidate identification.

5. IDENTIFICATION ALGORITHM

Our component identification tool utilizes a multipass process. A circuit of interest, described in Bench file format [2], is supplied to the identification tool. Searching begins with enumeration of candidate components starting with components of larger size since larger components may be composed of smaller ones. When the tool identifies a candidate it is logged until the pass completes. We then remove all identified components from the circuit of interest and repeat the search. This process continues until zero components are identified in a single pass. Figure 4 shows a process flow chart.

6. TEST CIRCUITS

Our primary circuit of interest is the ISCAS-85 C6288 16-bit multiplier [2]. We consider this type of circuit one in which

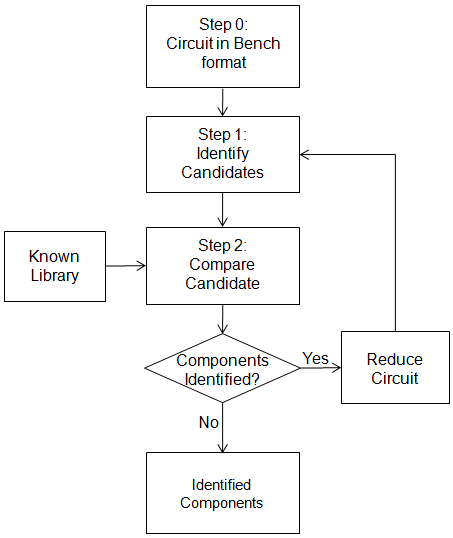

Figure 4: Component identification process.

protection is most difficult because of its repeated gate structure. We created other test circuits from our own custom component library. These components are random circuits with input and output size of six or less. This I/O space is chosen for making equivalence checking feasible. Our largest component is a six input four output circuit containing 145 gates. With this component, the worst case number of comparisons for equivalence is 6!4! = 17280 comparisons.

The 16-bit multiplier contains 240 adder components, 224 full adders and 16 half adders. Full adders are three input two output component containing 12 gates and half adders are two input 2 output components with 11 gates. Using custom components we constructed a large I/O space circuit. The circuit has 70 inputs 28 outputs and contains 1374 gates. Figure 5 shows a high level diagram of the large test circuit. The diagram provides a topological view where inputs are at the top and outputs at the bottom of the diagram. Each box shown is a custom random component. Note that text in this figure are not important to read.

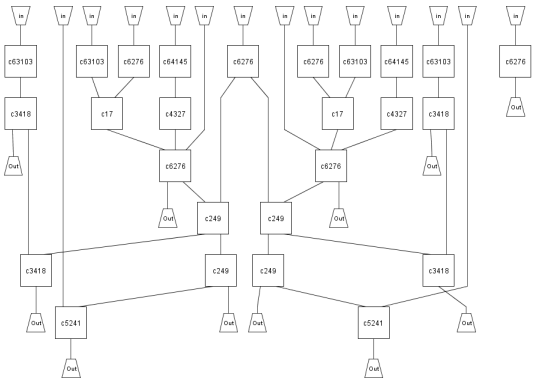

Figure 5: Largest custom test circuit containing 26 custom components.

7. IDENTIFICATION RESULTS

We found the tool best suited for "independent" components. We consider components "dependent" when a single gate supplies input to more than one component. Figure 6 shows a circuit with three custom components. Components B and C share an input from component A. This shared input makes component B and C dependent resulting in identification failure. Components in our test circuits are independent of each other.

Because we know the composition of each test circuit we used a specified search array to decrease overall search time. In C6288, we use the search set {12,11} while in the larger custom circuit we use the set {145,103,76,41,27,18,11,9}. In both circuits 100% of components are identified. Our tool needs a single pass taking 1.167 minutes to identify all 240 adder components. The larger circuit requires multiple passes for 100% identification. Our tool identifies all 26 components after four passes in 40.58 minutes.

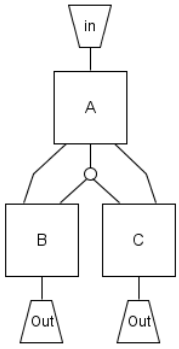

Figure 6: A three component circuit where identification fails from dependent components B and C.

We use a larger search set when using our tool for measuring the strength of circuit protection. This is more realistic to the methods a reverse engineer may use since they would not have full knowledge of circuit composition. We created three protected variants of C6288 and searched for adder components using our component identification tool. A search set of $\{25, 24, \ldots, 11\}$ was used. Table 1 shows the results of circuit protection measurements.

Table 1: Results of component identification on an unprotected C6288 and two protected variants.

Circuit	Gate Size	Components Identified	Identification Time
Unprotected	2448	100%	18.8 minutes
Variant One	2468	92%	18.9 minutes
Variant Two	5784	.02%	44.5 minutes
Variant Three	7052	0	54.3 minutes

8. CONCLUSIONS

Component identification is one step of the reverse engineering process. Up until now, no identification tool existed in our protection toolkit. Using the subcircuit enumeration algorithm developed by [8], we implemented our own component identification tool. The tool is limited to circuits containing independent components and is not efficient when checking equivalence of circuit with large I/O space. However, we have shown computer tools will aid reverse engineering for identifying components of larger circuits. Our component identification tool has been used for measuring effectiveness of current protection algorithms and for developing algorithms which defeat component identification all together.

9. ACKNOWLEDGMENTS

This work was supported by a research grant from the Air Force Office of Scientific Research. The authors would like to thank Dr. Travis Doom at Wright State University for sharing references and discussing his work on component identification with us.

10. DISCLAIMER

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

11. REFERENCES

[1] P. Dillien. Shhh, it's a secret: Encrypting uas communications. *Unmanned Systems*, 28(2):43–44, Feb 2010.

[2] M. C. Hansen, H. Yalcin, and J. P. Hayes. Iscas-85 c6288 16x16 multiplier. World Wide Web. Available at http://www.eecs.umich.edu/ jhayes/iscas/c6288.html.

[3] M. C. Hansen, H. Yalcin, and J. P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *IEEE Design and Test of Computers*, 16(3):72 – 80, 1999. Carry look ahead;Error correcting circuits;Register transfer;.

[4] J. T. McDondald, Y. C. Kim, and M. R. Grimaila. Protecting reprogrammable hardware with polymorphic circuit variation. In *Proceedings of the 2nd Cyberspace Research Workshop*, pages 63–78, 2009.

[5] K. Nohl, D. Evans, S. Starbug, and H. Plötz. Reverse-engineering a cryptographic rfid tag. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 185–193, Berkeley, CA, USA, 2008. USENIX Association.

[6] Office of the Under Secretary of Defense (Comptroller). Fy 2009 budget request. World Wide Web, 2008.

[7] J. L. White. *Candidate Subcircuit Enumeration for Module Identification In Digital Circuits*. Ph.D. dissertation, Department of Computer Science and Engineering, Michigan State University, 2000.

[8] J. L. White, A. S. Wojcik, M.-J. Chung, and T. E. Doom. Candidate subcircuits for functional module identification in logic circuits. pages 34 – 38, Chicago, IL, USA, 2000. Functional module identification;.