Developing a Deterministic Polymorphic Circuit Generator Using Boolean Logic Representation

Trinity L. Stroud
School of Computing
University of South Alabama
tls1627@jagmail.southalabama.edu

J. Todd McDonald
School of Computing
University of South Alabama
jtmcdonald@southalabama.edu

ABSTRACT

As the presence of software and hardware becomes steadily more pronounced in the many and varied applicable areas of society today, the theft of intellectual property (IP) embedded in such technology has increasingly become a problem. Obfuscation is the act of transforming a piece of software or hardware such that recovering information related to the original function or structure is made more difficult. In the context of circuits and software protection, this process of obfuscation involves the selection of subcircuits within a circuit and the replacement of said subcircuits with functionally equivalent variants.

In this research, we design and implement an obfuscation algorithm that operates by iteratively performing random Boolean logic expansions (RBLE) on a given circuit's Boolean expression in order to preserve and conceal its function. This research extends obfuscation research done by J. T. McDonald, Y. C. Kim, and others by comparing the effectiveness of this RBLE algorithm with their two random selection/replacement obfuscation algorithms involving static circuit libraries and circuit generation.

By studying the distributions of the circuit functions produced by each of the algorithms for a number of random trials, we saw that the circuits used by the algorithms as replacements for the selected subcircuits were generally chosen such that the distributions produced were uniform for every function and obfuscation approach. This demonstrates the RBLE algorithm does approach a uniform random selection from the set of all circuits that implement a specific function.

1. INTRODUCTION

Intellectual property (IP) is currently embedded in both software and hardware that are used in almost every area of society today. As companies can typically have billions of dollars invested in such IP, the theft of IP has become a major concern for tech companies and countries around the world. This research project concerns polymorphism and circuit protection for the purpose of approaching an efficient obfuscation algorithm, which could serve as a defense against adversarial reverse engineering and, ultimately, IP theft.

This would involve transforming programs such that the amount of time and resources necessary for a malicious reverse engineer to recover IP from a circuit program is made undesirable to perform or, in the best case, completely infeasible [2]. A solution to this problem of reverse engineering could be applied in the areas of circuit and software protection to the effect that IP of any manner could be safeguarded to a degree against attacks meant to discern the function and form of said programs [10].

One particular type of obfuscating transformation, known as a selection/replacement algorithm [1], involves taking small parts of a circuit and replacing that small part with a functionally equivalent version, or variant, with some different structure. This process is repeated over and over again (iteratively) until some desired level of overhead or security is reached, and the program is deemed to be obfuscated.

In order to reach this solution, the specific question we look to answer is: Can we create a deterministic circuit generation algorithm that approaches a uniform random selection from the set of all circuits that implement a specific function? Such an algorithm could greatly improve the efficiency and capacity of selection/replacement algorithms over, for example, other approaches that involve the enumeration of and selection of circuits from static libraries that must be stored on disk or randomly generated [8].

In order to answer this question, we plan to implement a Random Boolean Logic Expansion (RBLE) algorithm as a part of our existing Program Encryption Toolkit (PET) software. As combinational circuits are, in essence, no different from Boolean expressions [7], we will be using this algorithm while introducing random choices so that we can create polymorphic circuit variants from their manipulated Boolean expressions as part of a selection/replacement algorithm [5], thereby reducing the quantity of resources required to perform operations and introducing interesting potential avenues for program protection. Our results indicate that the RBLE algorithm does approach a uniform random selection when the distributions produced by its replacements are compared to that of the selection/replacement algorithm and the circuit generator.

This paper is organized in the following manner. In Section 2 we consider related works in obfuscation and netlist recovery. In Section 3 we describe our experimental methodology. In Section 4 we discuss the results of our research. In Section 5 we close by reflecting on our results and sharing our goals for future research.

2. BACKGROUND AND RELATED WORKS

2.1 Boolean Logic Functions

A Boolean function is a function with a domain of values {0,1} and of a finite number of variables of value {0,1}. The following theorem [11] lists some of the properties by which a Boolean function is defined on the set B = {0,1}:

Theorem 1.1: For all x,y,z ϵ B, the following identities hold:

1. $x \vee 1 = 1$ *and* $x\ 0 = 0$;

2. $x \vee 0 = x$ *and* $x\ 1 = x$;

3. $x \vee y = y \vee x$ *and* $x\ y = y\ x$;
(Commutativity)

4. $(x \vee y) \vee z = x \vee (y \vee z)$ *and* $x\ (y\ z) = (x\ y)\ z$;
(Associativity)

5. $x \vee x = x$ *and* $x\ x = x$;
(Idempotence)

6. $x \vee (x\ y) = x$ *and* $x\ (x \vee y) = x$;
(Absorption)

7. $x \vee (y\ z) = (x \vee y)\ (x \vee z)$
and $x\ (y \vee z) = (x\ y) \vee (x\ z)$;
(Distributivity)

8. $x \vee \bar{x} = 1$ *and* $x\ \bar{x} = 0$;

9. $\bar{\bar{x}} = x$;
(Involution)

10. $\overline{(x \vee y)} = \bar{x}\ \bar{y}$ *and* $\overline{(x\ y)} = \bar{x} \vee \bar{y}$;
(De Morgan's Laws)

11. $x \vee (\bar{x}\ y) = x \vee y$ *and* $x\ (\bar{x} \vee y) = x\ y$.

2.2 Boolean Logic Expressions

One way to represent a Boolean function is with a Boolean expression, a logical statement that, upon evaluation, has a value of either 0 or 1, false or true. Our notation for operators within Boolean expression is as follows:

disjunction, or ∨, will be +;
conjunction, or ∧, will be *;
and negation, or $\bar{x}$, will be x'.

There is an additional operator, xor, represented by ^, which is a derived symbol based on disjunction, conjunction, and negation rules as follows:

$$(x\,\bar{y}) \vee (\bar{x}\,y)\ and\ (x \vee y)\,(\bar{x} \vee \bar{y}),$$

with our notation for the above expression being:

$$(x * y') + (x' * y)\ and\ (x + y) * (x' + y').$$

The following table lists a number of Boolean logic laws, equations whose function of the expression on its left-hand side (LHS) is equivalent to the function of the expression on its right-hand side (RHS):

#	Original		Reduction	Law
1	A * A	=	A	Idempotence
2	A + A	=	A	Idempotence
3	A * B	=	B * A	Commutativity
4	A + B	=	B + A	Commutativity
5	A * (B * C)	=	(A * B) * C	Associativity
6	A + (B + C)	=	(A + B) + C	Associativity
7	A * (A + B)	=	A	Absorption
8	A + (A * B)	=	A	Absorption
9	A * (B + C)	=	(A * B) + (A * C)	Distributivity
10	A + (B * C)	=	(A + B) * (A + C)	Distributivity
11	A * 0	=	0	Annihilation
12	A + 0	=	A	Identity
13	A ^ 0	=	A	Identity
14	A * 1	=	A	Identity
15	A + 1	=	1	Annihilation
16	A ^ 1	=	A'	Negation
17	A * A'	=	0	Complementation
18	A + A'	=	1	Complementation
19	(A')'	=	A	Involution
20	(A + B)'	=	A' * B'	De Morgan's
21	(A * B)'	=	A' + B'	De Morgan's
22	(A + B) * (A' + B')	=	A ^ B	Derivation
23	(A' * B) + (A * B')	=	A ^ B	Derivation
24	(A + B)' + (A * B)	=	(A ^ B)'	Negation
25	A ^ A	=	0	Annihilation
26	(A ^ A)'	=	1	Annihilation
27	(A * B') + (A * B)	=	A	Annihilation
28	(A' * B') + (A' * B)	=	A'	Negation
29	(A + B) * (A + B')	=	A	Annihilation
30	(A' + B) * (A' + B')	=	A'	Negation

Table 1: Boolean Logic Laws

2.3 Digital Logic Circuits

A Boolean expression is akin to a combinational circuit whose inputs, outputs, and gates correspond to Boolean variables and operators in an expression.

As a combinational circuit is, in essence, equivalently representable as a Boolean expression by its function [3], we have chosen certain symbols and rules for describing this notation. The table below summarizes the names and symbols used for the unary and binary Boolean operators, which form a basis for logic gates:

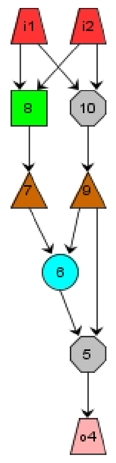
Name	Notation
NOT	A’
AND	A * B
NAND	(A * B)’
OR	A + B
NOR	(A + B)’
XOR	A ^ B
NXOR	(A ^ B)’

Table 2: Gate Types and Notations

where A and B represent Boolean variables with unknown values of either 0 or 1, false or true.

In order to demonstrate the equivalence of Boolean expression and digital gates, we take the following digital logic circuit as an example:

Figure 1: Example Digital Logic Circuit



This circuit is functionally equivalent to the following Boolean expression:

o4 = ((i1 + i2)’ * (i1 ^ i2)’) ^ (i1 ^ i2)’,

where o4 represents the output of the circuit and i1 and i2 represent the two inputs to the circuits.

These digital logic circuits can be grouped together by their input, output, and gate sizes. The notation we will use for a circuit of input size X, output size Y, and gate size Z will be X-Y-Z. As an example, the above circuit, which has 2 inputs, 1 output, and 5 gates, would be represented as 2-1-5. The family of circuits which all have X inputs, Y outputs, and Z gates will be represented as CX-Y-Z. In this example, all those circuits which are 2-1-5 circuits belong to the C2-1-5 family.

Circuits can further be grouped by the functions they each represent. For example, C2-1-1 consists of 6 circuits that together represent 6 distinct functions between them. These functions are as follows:

1.	o1 = (i1+i0)’	(NOR)
2.	o1 = i1^i0	(XOR)
3.	o1 = (i1*i0)’	(NAND)
4.	o1 = i1*i0	(AND)
5.	o1 = (i1^i0)’	(NXOR)
6.	o1 = i1+i0	(OR)

The model below illustrates the relationships between, as an example, the C2-1-1 circuit family and its function partitions:

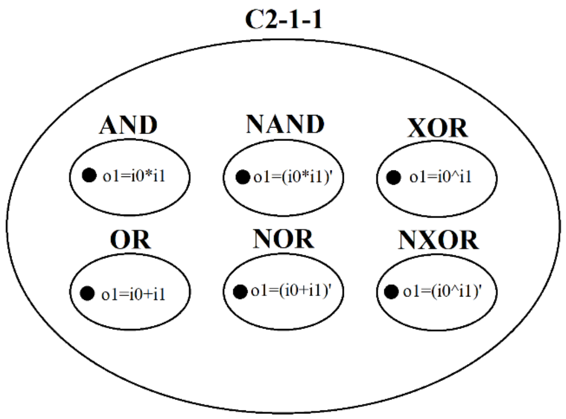


Figure 2: C2-1-1 Partitioned by Function

There are additional characteristics of digital logic gates which can be used to define circuit families. The fan-out of a gate is the number of gates to which the output of a single gate is fed as input. The fan-in of a gate is the number of inputs fed to the gate. Given different values for these gate features, different circuit families can be produced. We define our circuit families as consisting of circuits whose gates have a fan-in of 1, in the case of the unary gate NOT, or 2, in the case of the other gate types defined in Table 2.

There is also the possibility that a circuit family of certain input, output, and gate sizes may contain a partition whose function is a CONST0 or CONST1. A circuit of such a function has, for any mapping of input values, the Boolean value 0 or 1, respectively. Allowing or disallowing the presence of such circuits is another way in which the characteristics of a circuit family can be described. The decisions regarding these gate features and the gate types which may appear in circuits determine the basis set to be used for logic gates.

2.4 Boolean Reduction and Expansion

Obfuscation, as related to Boolean logic, can be performed in a process known as Random Boolean Logic Expansion (RBLE), whereby a Boolean variable or subexpression can be replaced with an equivalent subexpression reached through the application of logic laws on the original subexpression.

After a component of a circuit’s Boolean expression is selected in the process of obfuscation, this RBLE algorithm would be employed in order to randomly determine from a list of Boolean logic laws one to apply to the subexpression that would, as a consequence of being a logic law, alter its form while preserving its function. The resulting subexpression would take the place of the original selected from the circuit’s expression so that a portion of the circuit has been obfuscated with no change made to its overall function. Each output function of the circuit would be represented with separate Boolean expressions, though the functions themselves may be closely connected [9], and this method applied to each.

This process would be repeated, with different portions of the expression being transformed randomly, until such a time as the circuit's expression is considered to be appropriately obfuscated. This condition for completion can be on the basis of either the number of expansions performed or on the size of the resulting Boolean expression.

For any Boolean logic law, the LHS of the expression is functionally equivalent to the RHS of the expression. This fact allows us to substitute a portion of an expression that matches the LHS or RHS of a logic law with its corresponding RHS or LHS. To demonstrate this concept, the following steps detail the expansion of a Boolean expression belonging to the circuit family C2-1-1 to the expression representing the C2-1-2 circuit of the same function:

$$g1 = (i0 * i1)'$$

1: (0 + (i0 * i1))' (law 12)
2: ((i1' * 0) + (i1 * i0))' (law 11)
3: ((i1' * (i0 * i0')) + (i1 * i0))' (law 17)
4: ((i1' * (i0 * i0')) + (i1 * (i0 * i0)))' (law 1)
5: ((i1' * (i0 * i0')) + (i0 * (i1 * i0)))' (law 5)
6: ((i0 * (i1'*i0')) + (i0 * (i1*i0)))' (law 5)
7: (((i1' * i0') + (i1 * i0)) * i0)' (law 9)
8: ((i1 ^ i0)' * i0)' (law 24)

$$g1 = ((i1 \wedge i0)' * i0)'$$

The images below are graphical representations, known as operator trees, of the original Boolean expression and the resulting expression after expansion:

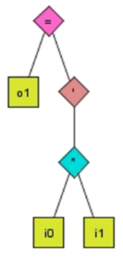


Figure 3: Original Expression

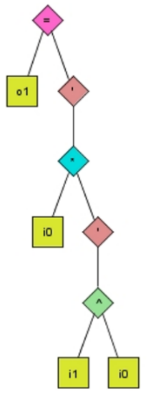


Figure 4: Resulting Expression

The list of Boolean logic laws used by the RBLE algorithm to perform such an operation can be reduced to include only those logic laws which actually change the structure of an expression. Laws which demonstrate associativity or distributivity or which have an effect on the number of variables or values represented in an expression are valuable for this purpose of altering structure, while such laws as those which only demonstrate commutativity are not. Therefore, laws #3 and #4 from Table 1 can be removed, leaving us with this optimized and reordered list of Boolean logic laws:

#	Original		Expansion	Law	Relative Gates
1	0	=	A * 0	Annihilation	CONST0
2	0	=	A * A'	Complementation	CONST0
3	0	=	A ^ A	Annihilation	CONST0
4	1	=	A + 1	Annihilation	CONST1
5	1	=	A + A'	Complementation	CONST1
6	1	=	(A ^ A)'	Annihilation	CONST1
7	A	=	A * A	Idempotence	AND
8	A	=	A + A	Idempotence	OR
9	A	=	A * (A + B)	Absorption	AND,OR
10	A	=	A + (A * B)	Absorption	OR, AND
11	A	=	A + 0	Identity	OR, CONST0
12	A	=	A ^ 0	Identity	XOR, CONST0
13	A	=	A * 1	Identity	AND, CONST1
14	A	=	(A')'	Involution	NOT
15	A	=	(A * B') + (A * B)	Annihilation	AND,OR,NOT
16	A	=	(A + B) * (A + B')	Annihilation	AND,OR,NOT
17	A'	=	A ^ 1	Negation	XOR, CONST1
18	A'	=	(A' * B') + (A' * B)	Negation	AND,OR,NOT
19	A'	=	(A' + B) * (A' + B')	Negation	AND,OR,NOT
20	(A + B)'	=	A' * B'	De Morgan's	NOR
21	(A * B)'	=	A' + B'	De Morgan's	NAND
22	A ^ B	=	(A + B) * (A' + B')	Derivation	XOR
23	A ^ B	=	(A' * B) + (A * B')	Derivation	XOR
24	(A ^ B)'	=	(A + B)' + (A * B)	Negation	NXOR
25	A * (B + C)	=	(A * B) + (A * C)	Distributivity	AND,OR
26	A + (B * C)	=	(A + B) * (A + C)	Distributivity	OR,AND
27	(A * B) * C	=	A * (B * C)	Associativity	AND
28	(A + B) + C	=	A + (B + C)	Associativity	OR

Table 3: Reordered Boolean Logic Laws

The laws have been rearranged such that their original expressions are ordered by lowest-to-highest form. This ordering allows us to easily recognize that, for example, the expression 0 has 3 possible expansions, the expression 1 has 3 possible expansions, and the expression A has 10 possible expansions.

2.5 Related Works

Kim and McDonald [4] examined random and deterministic techniques of obfuscation on logic-level definitions for the purpose of producing semantically-equivalent variants to hinder the efforts of adversaries to analyze protected intellectual property. The metrics used to measure the effectiveness of these obfuscation algorithms included the successfulness of topology, signal, and component recovery. They showed that there are tradeoffs to using either an obfuscation algorithm that implements random choice or one that targets specific hiding properties of a circuit.

McDonald et al. [5] proposed an obfuscation algorithm that produces a circuit variant by enacting repeated small, incremental and random changes to a component that preserve its function. This sequence of changes would serve as a key to the original circuit. The circuit variants produced by the algorithm were intended to have functions semantically equivalent to that of the original circuits, better hide their structural and functional information, and be more easily reproduceable in terms of the time required for completion and the variant's gate size.

McDonald et al. [6] considered the experimental effects of the random or deterministic application of changes by transformation algorithms to circuits for the purpose of protecting certain properties of information. The authors' Random Program Model used whitebox and blackbox transformations for protecting the intent of a circuit such that there would be no correlation between the behavioral information leaked by the obfuscated circuit and the behavior of the original circuit and that there would be no more correlation between the structural topology of the obfuscated circuit and the original circuit as there would be for any randomly selected circuit.

McDonald et al. [8] compared four different white-box transformation algorithms that change the component and signal configurations within combinational logic programs with the aim of hindering reverse engineering. The four algorithms – Pseudorandom Selection and Replacement, Deterministic Boundary Blurring, Deterministic Component Fusion, and Deterministic Component Encryption – were performed on three test circuits to produce variants on which were applied reduction and analysis of signals and components.

Hansen et al. [10] demonstrated the vulnerability of the ISCAS-85 benchmark suite of circuits to reverse engineering by creating for them high-level models from their original gate-level netlist forms. The techniques they used to do this included examining library modules of commonly-occurring components, repeated modules for unknown functions, expected global structures when modules are found, and computed functions for a small number of signals.

McDonald et al. [12] used Boolean logic reduction in order to determine the effectiveness of transformation techniques in hiding structural information with the goal of preventing or hindering reverse engineering. They also sought to determine whether properties of protection occurring within a circuit variant would be dependent upon its number of intermediate gates or the method by which the circuit variant is created. The authors demonstrated that allowing for random choices in the process of circuit generation can result in the presence of properties of protection.

Kim [13] used ten different minimization-by-pattern-analysis techniques and two minimization-by-truth-table-analysis techniques to reduce the occurrence of redundancy within obfuscated circuits by removing those found through the examination of the functions represented in the circuit's truth table. The author also examined the circuits' gate sizes and level counts in order to determine the efficacy of obfuscated circuits while noting the amount of observed redundancies that could not be reduced. A high amount of such would indicate the circuit to be powerfully obfuscated, while a low number would indicate the obfuscation to be weak.

3. METHODOLOGY

To ensure that our RBLE algorithm truly approaches a uniform random selection from the set of all circuits that implement a specific function, we created distributions of the functions of the circuit families produced by each of the following methods: the random and iterative application of Boolean logic laws to a circuit's expression to produce a variant, the enumeration and random selection of a circuit from a circuit family, and the generation of a circuit until a functionally equivalent variant is produced. Comparing the resultant distributions would allow us to verify if our RBLE algorithm is truly as random as these other methods of random selection/replacement and random generation. The following sections describe the software approach behind these obfuscation algorithms and the procedure for generating these distributions.

3.1 Experimental Design

In order to gauge the effectiveness of this RBLE algorithm in relation to the circuit generator and the circuit chooser algorithms, we performed experiments to track which circuits were selected as replacements by these algorithms for 4 of the functions of the C2-1-1 family as their circuits were expanded to become of the C2-1-2 family. We then compared the distributions of circuits selected in order to determine how closely the RBLE algorithm approaches a uniform random selection.

Each of the 3 obfuscation algorithms has 3 strategies for how selection and replacement is performed: size-based, where selections and replacements (i.e., iterations) are applied until the overall specified circuit size is reached; iteration-based, where iterations are applied a fixed number of times; and round-based, where iterations are applied until all original gates in the circuit have been replaced at least once in a round for a number of rounds.

The RBLE approach in particular has 3 policies, or rules, to be followed when generating or creating a replacement circuit given some subcircuit selection. The first policy, Fixed, involves applying E number of expansions to a subcircuit selection. The second policy, Target, involves applying a variable number of expansions until such a time as size T is reached or exceeded for the replacement subcircuit. The third policy, Strict, involves applying a variable number of expansions to the selected subcircuit until a strict size S is reached or exceeded for the replacement subcircuit, with a specified value for maximum divergence allowing for some number of gates above the strict size and a specified value for maximum attempts limiting the number of times a sequence of expansions may be attempted. The Strict policy is the one we implemented in our experiment, as it is the policy which most closely mimics the behavior of the other 2 obfuscation algorithms.

3.2 Data Collection

Data was collected on the distributions of the subcircuit replacements for each of the 4 functions of C2-1-1 which can be expanded to their respective functions within C2-1-2. As this was done for each of the 3 obfuscation algorithms, this resulted in 4 sets of 3 circuit distributions each being produced.

First, the circuits of the C2-1-1 family with function AND were examined. The circuit chooser algorithm was called and a functionally equivalent variant belonging to the C2-1-2 family returned. This process was repeated 1000 times and a distribution made for the circuits of the C2-1-2 family with function AND used as subcircuit replacements.

Second, the circuit generator was called and, similarly, a variant returned for a circuit with function AND and of the C2-1-2 family.

This process was also completed 1000 times and the circuits selected for replacement counted in order to produce a distribution of the circuits in the C2-1-2 family.

Third, the boolean expander was called and its policy set to Strict. Given a 2-1-1 subcircuit with function AND, a 2-1-2 subcircuit of the same function was produced through Boolean expansion for the purpose of replacement. Likewise, this process was repeated 1000 times so that a distribution could be examined of the circuits within the C2-1-2 family which were achieved through expansion for a subcircuit of function AND.

These three steps were repeated for each of 4 functions of the C2-1-1 family which also appear in the C2-1-2 family – AND, OR, NAND, and NOR– with the result that 4 sets of 3 distributions each were produced.

4. RESULTS

The following tables detail the frequencies with which of each of the circuits in each of the 4 functions examined were used as replacements for their functionally equivalent C2-1-1 variants by the 3 different obfuscation algorithms in 1000 trials:

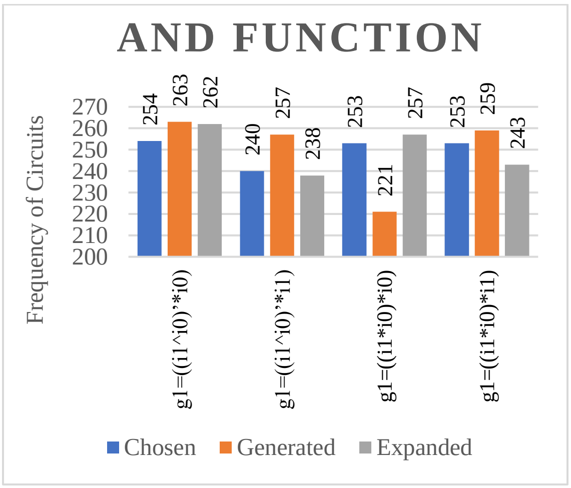


Table 4: Distribution of Circuits for AND Function

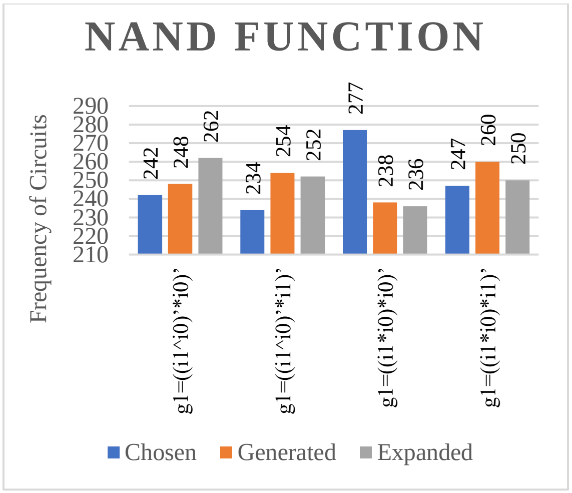


Table 5: Distribution of Circuits for NAND Function

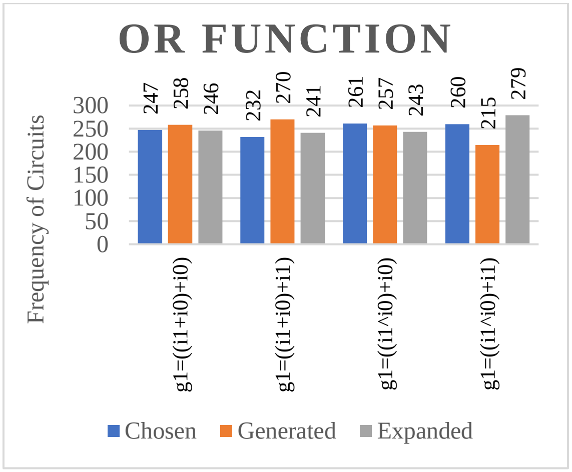


Table 6: Distribution of Circuits for OR Function

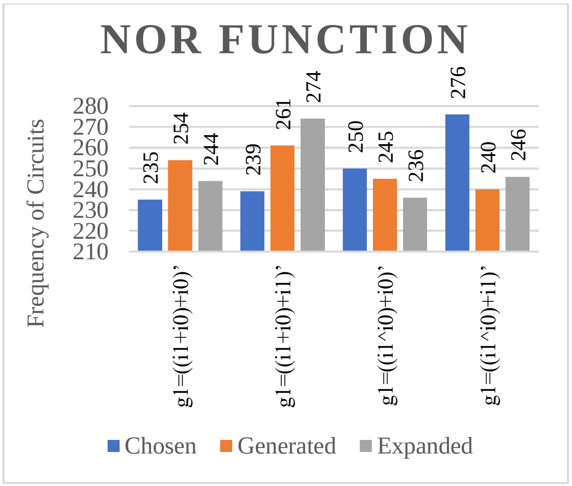


Table 7: Distribution of Circuits for NOR Function

The distribution of circuits generated or chosen by each the obfuscation algorithms are shown to be relatively equal no matter the function, circuit, or obfuscation approach used, with no circuit for any function being heavily favored by any obfuscation approach.

5. CONCLUSION

Through this research we have demonstrated that the RBLE obfuscation algorithm approaches a uniform random selection from the set of all circuits that implement a specific function by comparing the distribution of circuits generated by this algorithm to the distributions produced by the two random selection/replacement obfuscation algorithms involving static circuit libraries and circuit generation.

Our future goals for extending this research include examining the effects of varying the minimum and maximum selection sizes used by these algorithms, manipulating the variation options in order to observe the effects this may have on the circuits and functions represented in certain circuit families, and applying logic reduction algorithms to circuits expanded through Boolean logic operations so as to measure the ability of reverse engineering tools to recover original circuits.

REFERENCES

[1] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677-691, Aug. 1986, doi: 10.1109/TC.1986.1676819.

[2] H. R. Andersen and H. Hulgaard, "Boolean Expression Diagrams," *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-1997)*, Warsaw, Poland, 1997, pp. 88-98. doi: 10.1109/LICS.1997.614938.

[3] H. R. Anderson, "An Introduction to Binary Decision Diagrams," *Efficient Algorithms and Programs*, Lecture, The IT University of Copenhagen, Copenhagen, Denmark, 1999.

[4] J. T. McDonald and Y. C. Kim, "Examining Tradeoffs for Hardware-Based Intellectual Property Protection," *Proceedings of the 7th International Conference on Information Warfare (ICIW-2012)*, University of Washington, Seattle, USA, March 22-23, 2012.

[5] J. T. McDonald, Y. C. Kim, and D. Koranek, "Deterministic Circuit Variation for Anti-Tamper Applications," *Proceedings of the Cyber Security and Information Intelligence Research Workshop (CSIIRW-2011)*, Oak Ridge, TN, Oct. 12-14, 2011.

[6] J. T. McDonald, Y. C. Kim, and M. R. Grimaila, "Protecting Reprogrammable Hardware with Polymorphic Circuit Variation," *Proceedings of the 2nd Cyberspace Research Workshop*, Shreveport, LA, USA, June 2009.

[7] Yasinsac and J. T. McDonald, "Of Unicorns and Random Programs," *Proceedings of the 3rd IASTED International Conference on Communications and Computer Networks (IASTED/CCN-2005)*, Marina del Rey, CA, USA, Oct. 24-26, 2005.

[8] J. T. McDonald, Y. C. Kim, D. Koranek, and J. D. Parham, "Evaluating Component Hiding Techniques in Circuit Topologies," *International Conference on Communications, Communication and Information Systems Security Symposium (ICC-CISS-2012)*, Ottawa, Canada, June 10-15, 2012.

[9] R. E. Bryant, "Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification," *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD-1995)*, San Jose, CA, USA, 1995, pp. 236-243. doi: 10.1109/ICCAD.1995.480018.

[10] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3,

pp. 72-80, July-Sept. 1999, doi: 10.1109/54.785838.

[11] Y. Crama and P. Hammer, "Chapter 1: Foundations," *Boolean Functions: Theory, Algorithms and Applications (Encyclopedia of Mathematics and its Applications) 1st Edition*, Cambridge University Press, Jan. 2006, pp. 21-23.

[12] J. T. McDonald, E. D. Trias, Y. C. Kim, and M. R. Grimaila, "Using Logic-Based Reduction for Adversarial Component Recovery," *Proceedings of the 25th ACM Symposium on Applied Computing*, Sierre, Switzerland, March 2010.

[13] H. Kim, "Optimizing Redundant Logic Pathways in Polymorphic Circuits," Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2009.

[14] P. Svensson, "Chapter 4: Boolean Algebras," *Boolean Algebras*, Växjö, Sweden, Dec. 2009, pp. 21–31.

[15] P. Svensson, "Chapter 5: Boolean Algebras and Electronic Circuits," *Boolean Algebras*, Växjö, Sweden, Dec. 2009, pp. 32–40.

[16] M. Genesereth and E. Kao, "Chapter 2: Propositional Logic," *Introduction to Logic*, Morgan & Claypool Publishers, 2012, pp. 13–22.