4-Bit Cellular Automata Encryption Analysis

A Master Thesis By

Dallas Leitner

Fulfilling a partial requirement for the Degree of

Master of Science in Engineering Technology

Middle Tennessee State University

August 2017

Thesis Committee:

Dr. Karim Salman, Chair

Dr. Walter Boles

Dr. Saleh Sbenaty

ABSTRACT

Various niches have used forms of cellular automata for decades. One such use is random number generation for data encryption. Of the numerous methods developed for encryption, most use the 3-bit rule space. While this rule space has been tested and proven to possess the desired traits, limiting CA to only the 3-bit rule space severely limits the potential for extreme levels of complexity. To that end, this research aims to explore the 4-bit rule space to find new potential rules that possess the desired level of complexity with the hope that the methods used will allow for exploration of even greater rule spaces in the future. Through mass testing of the rule space in the Diehard test suite, has shown 362 4-bit rules that show good potential for use in cellular automata encryption.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE: RESEARCH FOCUS

Cellular automata have been around since the 1960's [1] and research has proven it is useful for use in data encryption [2]. However most work done has had a focus on the 3-bit rule space. While there are various applications of the 3-bit rules they still utilize the same 256 rule space. A large contributing factor to why other rule spaces hare not used is due to the computational power required to test larger rule spaces. While a 3-bit rule space only contains 256 unique rules, a 4-bit rule space contains 65536 unique rules and 5-bit is in the millions. This research has 2 goals. The first of which is to test the entire 65536 rule space to find suitable rules for encryption applications. The second more tangential goal is to discover potential efficient ways to find good rules in higher bit rule spaces without the requirement of mass testing in the rule space.

CHAPTER ONE: A BRIEF OVERVIEW OF ELEMENTARY CELLULAR AUTOMATA

Section 1.1: Elementary Cellular Automata Introduction

Cellular Automata (CA) are grid based models that use colored cells to display data with a finite number of states, and change according to set rules across a number of discrete time steps. Elementary cellular automata (ECA) is the simplest class of CA. ECA neighborhoods are 3-bit, one dimensional, and each cell contains a binary value of 1 (high) or 0 (low). The cells are arranged horizontally in a single row to represent the current state of the automaton, because of this each subsequent row can be used to represent the next state generated by the given rules. This creates a two-dimensional table from one-dimensional data that shows changes over time. The data contained within the first row is predetermined in order to initialize the automaton but can consist of any desired series of binary data.

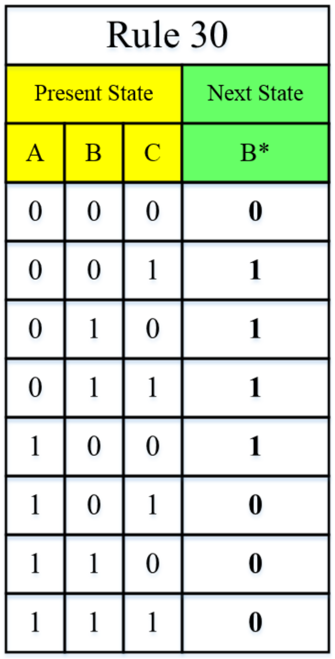
Section 1.2: Cellular Automata Structure

The basic structure of ECA consist of 4 key parts. The first is seed. The seed is simply the name of the initial data used for the automaton. As previously mentioned, the seed can contain any data desired in addition to being any length. The length of chosen seed determines the length for the entire system. If the seed contains 150 bits of data, every row in the CA will contain 150 bits of data. Lastly although the seed may contain any data, for practical use for data encryption, it is best to use a random seed. The second major component is the neighborhood. The neighborhood is simply the subsection of local data used to determine the next state. In ECA systems, the neighborhood is 3 bits long. This means that the current state of the 3 cells in the neighborhood define the next state for the center bit in each cell. As an example table 1 shows a seed with a length of 7 bits. The label of each bit from left to right is A, B, C, D, E, F, and G. For bit B, the neighborhood would consist of bits A, B and C. For bit C, the neighborhood would contain bits B, C, and D. The exceptions to this are the two end bits for which the rules can differ slightly. However this study only uses periodic systems. In a periodic system the end bits simply use the bit on the opposite end for their vacant spot in the neighborhood. Thus A's neighborhood would be bits G, A, and B, while G's would be bits F, G, and A.

Table 1: 7 Cell Cellular Automaton seed.

A	B	C	D	E	F	G
1	0	1	1	0	1	0

The third part is the rule. The rule determines how the data changes based on the data contained in each neighborhood. The 3-bit CA rules pace contains a total number of 256 rule which may be used. The rules use binary logic to determine their functions. For a 3-bit system there is a total possible number of 8 different states the 3 bits could be in. These states range from 000 to 111. The next state has a value of 1 or 0 for each of the possible present states, as can be seen in figure 1 below.



Rule 30			
Present State			Next State
A	B	C	B*
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

Figure 1: Truth table for rule 30.

In this case if the data in a neighborhood contains 000, then the next state for the center bit would 0. If the neighborhood contains 001, the next state is 1 and so on. The name for a rule is simply the binary number produced by the output state. The figure above shows rule 30. The outputs listed from bottom to top produce the binary string of 00011110, which when read as a binary number is 30. The last part required for a CA system is the time step. The time step is when the change actually happens. A time step is when the data changes during each clock cycle. When the clock triggers, each cell's state changes based on the rule and the data contained in each of the neighborhoods, and all changes happen at once.

Section 1.3: Previous Works

Jon Neumann first published work on CA in the 1960's and later Stephen Wolfram published findings showing the potential use of CA in random number generation. Since then extensive studies have attempted to further understand and expand upon CA applications and potential. Building upon the results from previous works Dr. Karim Salman has narrowed the 256 rule space of 3-bit ECA to a set of only 16 rules which are chaotic in nature and thus are able to produce good random numbers [3]. The chaotic rules are 30, 45, 60, 75, 86, 89, 90, 101, 102, 105, 135, 149, 150, 153, 165, and 195. These 16 rules proven in other studies to have use for random number generation, set the baseline for other CA applications to reach in terms of complexity.

Stephen Faulkenberry wrote a thesis providing strong evidence to back up the claims of Stephen Wolfram from the 1980's that CA has use in random number generation. His thesis focused on the application of two modern day random number testers, Die Hard and NIST, to prove that CA random number generation is complex enough to still play a role in modern day encryption and other applications. Stating that CA's simplicity and exponential scalability is enough reason to pay attention to CA's potential [2].

CHAPTER TWO: 4-BIT STRUCTURES

Section 2.1: 4-Bit Clusters

The cluster structure for 4-bit rules is the same as for 3-bit rules described in [3]. A 4-bit rule clusters are created by applying the appropriate operation or combination of operations to the base rule. The differences in the clusters is simply the structure of how the minterms change. Although the application of the operations is still the same, the increased number of minterms in a 4-bit rule effect how minterms are changed. The minterms for a 3-bit rule can be represented on a 3-D plain as a single cube, but the diagram for a 4-bit rule has two cubes, with the two cubes representing a shift on a fourth axis, where the same points on the two cubes are one step away from each other. Figure 2 shows the cubic representation.

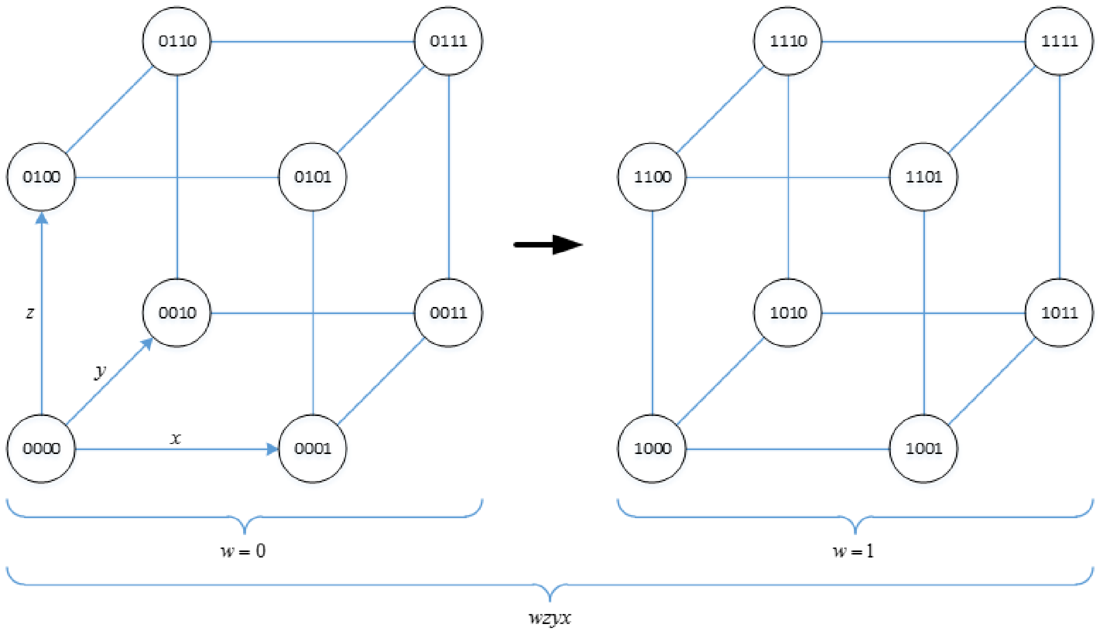


Figure 2: 4-bit minterm cube

Section 2.2: Left and Right

In typical 3-bit CA, the center of each neighborhood is the one being changed. However, in any even bit system there is no center bit. This means that 4-bit CA can either have a left hand structure or a right hand structure. Figure 3 shows a 4-bit neighborhood. For a left handed structure the central bit would be the B bit, while C is the center bit for right handed structures. All rules have a left handed and a right handed variant, since the structure changes how the rule is applied but not actually modifying the rule itself, which effectively double the rule space.

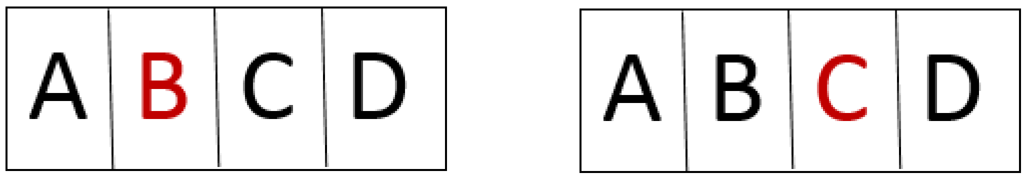


Figure 3: 4-bit neighborhood for left and right handed structures.

Section 2.3: Behavior Differences

A rule applied as a right hand structure tends to behave similarly but different from the same rule when applied to the left. This is due to how the minterms react to the changing of the center bit. The most common change seen in the CA data graphs is a variation between straight and diagonal lines. Figure 4 shows an image of rule 25957 graphed for both left and right structures. Although there are many differences in the graphs, they both follow a similar pattern with the exception that the left hand variant runs the pattern with diagonals, while the right hand variant creates vertical drops. Although different, they are both share the similar traits of having 2 bit wide drops. In addition to the width, the length of the drops are typically the same. The horizontal location of these drops change between rules due to the left hand structures diagonal tendency, but they still share vertical placement.

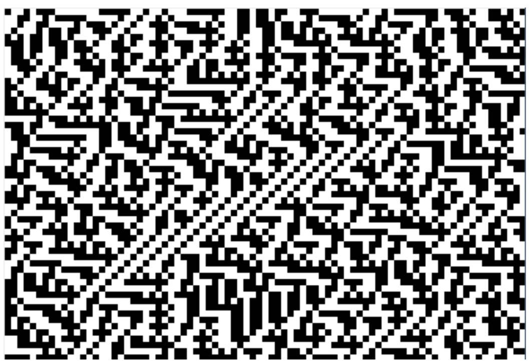
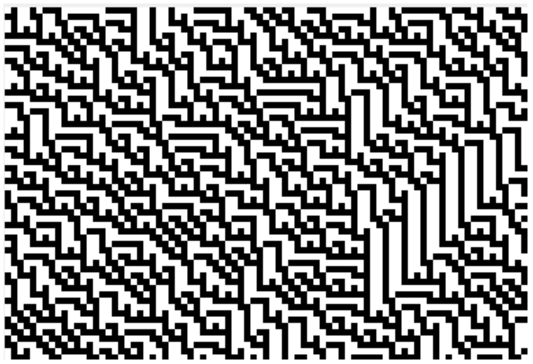



Figure 4: Automation of rule 25957 for left and right handed structures (left on left, right on right).

Section 2.4: 3-Bit Equivalents

For any rule space of any size, there will always be an overlap of rules that use a smaller neighborhood. This happens when the logic of a rule from a larger rule space ignores one or more bits in the neighborhood. 2-bit rule 6 and 3-bit rule 60 demonstrate this equivalence in how they both share the logic of an exclusive OR relationship between bits A and B. The only difference between the two is that rule 60 ignores the C bit, while rule 6 does not possess a third bit, which means that rule 60 effectively emulates rule 6. Similarly a 4-bit rule with the same logic, can emulate any 3-bit rule.

Since 4-bit rules have two structures, the rules that emulate 3-bit rules differ for each structure. The binary number of a 3-bit rule plays a key role in finding the 4-bit equivalent. To find the rule simply duplicate the binary number in one of two ways, depending on if it is a left or right hand structure.

For a right hand structure the equivalent rule will be the binary number for the rule written from start to finish twice. For instance 3-bit rule 30's, 00011110, 4-bit equivalent is rule 7710 which in binary is 000111100001110 or rule 30 twice (00011110 00011110). This is because for a right handed rule, the bit that needs to be ignored is bit A, and since the first half of the logic represents A in the digital off state (0) and the second half represents A in the digital on state (1), if the logic of bit A on and bit A off match, the bit can be ignored.

A similar process can produce the rule for a left hand structure. The difference between the two is that for a left hand structure, the bit that needs to be dropped is bit D. For this configuration the product looks more complicated, but is actually just as simple to produce. The state of D = 0 is represented by every other row on a truth table, with D=1 being represented on the rows between those. No other logic changes between each of these pairings, so if the logic is the same for each pair of rows, the D bit will not affect the logic. This means that to make a left hand structure equivalent rule for a 3-bit rule, one simply needs to write each digit of the binary number twice. This means that the 4-bit rule 1020 (0000001111111100) emulates the 3-bit rule 30 (00011110). If the binary number is divided up into pairs it appears the same as rule 30 only every digit is used twice, 00 00 00 11 11 11 11 00. Table 2 shows the 4-bit equivalent rules for all 16 of the 3-bit chaotic rules.

Table 2: 3-bit equivalent rules for left and right handed structures.

Rule 30 cluster			Rule 45 Cluster		
30	1020	7710	45	3315	11565
86	22102	13116	101	25957	15411
135	49215	34695	75	12495	19275
149	38293	49971	89	22873	13251
Rule 150 Cluster			Rule 90 cluster		
105	15555		90	13260	
105	26985		90	23130	
150	49980		165	52275	
150	78550		165	42405	
Rule 60 Cluster					
60	4080	15420	right hand equivalint		
102	26214	15420	Left hand equivalent		
195	61455	50115	3 Bit Equivalent		
153	39321	50115			

CHAPTER THREE: BIT SIGNIFICANCE

Section 3.1: The Need for a New Method

In previous works in addition to Diehard and NIST test, state diagrams helped isolate potential rules. Mapping out the state diagrams catalogues the number of unique cycles, transient lengths and the number of Garden of Eden states. Generating state diagrams for the 4-bit analysis proved less productive, because there was a far greater amount of variation in how the rules' ranked in these categories, in contrast to rule 45's cluster's near dominance in all categories for the 3 bit rules. This made comparing the effectiveness of a rule much more difficult and it is likely this complexity will increase with size of the neighborhood. This difficulty created a need for another way to compare the rules. The ideal method needed to be extremely quick to perform, filter out likely useless rules, highlight rules which would have the highest potential, and be as resource light as possible to make it scalable to larger neighborhoods.

Section 3.2: The Theory for Bit Significance

The process began with one simple question "what makes an output random?" In order for an output to be random, it needs to have no correlation to the input. However, the very concept of CA is systematic and thus the input directly influences the output. Therefor the best approach would be to find rules that showed as little correlation between the input and output as possible. The term "bit significance" references correlation between a single input bit and the output.

Section 3.3: 3-Bit Analysis

The method used for measuring bit significance is simple and only requires a quick look at the truth table of any given rule. For this experiment each bit in a 3-bit system, bits A, B, and C, begins with a base significance of 0. This implies that the bit has 0 correlation with the output. Then by comparing input value of each bit with the output value they produce the significance changes. If the bits are the same the bit significance is incremented by 1 or if the bits are different the significance value is decremented. Table 3 shows the truth tables for rules 45 and 30. The first three columns show the value of A, B, and C respectively, with the last three simply showing the output produced by each rule. The output value repeats 3 times to compare with each of the three input bits. If the cell is white, the bits are different and the significance level goes down, when the cell is red, the bits are the same and the significance level goes up. The ideal significance level is 0. This would imply that the bit is the same as and different from the input exactly half the time. Meaning they have very little correlation with each other.

Table 3: Output bit comparison to present state for rules 45 and 30 (left to right).

A	B	C	45		
0	0	0	1	1	1
0	0	1	**0**	**0**	0
0	1	0	1	**1**	1
0	1	1	1	**1**	**1**
1	0	0	0	**0**	**0**
1	0	1	**1**	1	**1**
1	1	0	0	0	**0**
1	1	1	0	0	0

A	B	C	30		
0	0	0	**0**	**0**	**0**
0	0	1	1	1	**1**
0	1	0	1	**1**	1
0	1	1	1	**1**	**1**
1	0	0	**1**	1	1
1	0	1	0	**0**	0
1	1	0	0	0	**0**
1	1	1	0	0	0

In both cases with these two rules, they end up showing a theoretical correlation with bit A at -4, and 0 for both B and C. This is because the A bit is only red twice and white 6 times, while both C and B's are equally split between white and red cells. Since there are a possibility of 8 combinations and 6 of those 8 show a negative correlation, when this rule is used the output should theoretically be the inverse of the A input approximately 75% of the time. Additionally the output should be the same as B and C approximately 50% of the time. To test this, rules 30 and 45 used random seeds to generate data. Tables 4, 5 and 6 show a seed of length 25 with 9 time steps for rule 45, and is colored based on if the bit follows the trend or not. The summing of the two possible states ignores the seed state since the seed does not have an input. This means that there are 9 rows of 25, which leads to a total of 225 cells. The first table colors the cells if the cell is the inverse of the A input for the neighborhood and remains white if it is the same.

Table 4: Bit A comparison to next state for rule 45.

!A		cell is green if it equals !A																							
1	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	
1	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	
1	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	
1	0	1	0	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	1	0	0	
1	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	
1	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	0	0	0	
1	0	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	0	
1	1	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	
0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	0	1	1	
0	1	1	0	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	
9	6	9	6	9	7	5	7	6	9	5	9	5	7	7	5	7	7	8	6	7	4	8	8	2	168

The table highlights and counts the number of cells which are equal to the inverse value of the respective cell's A input in the neighborhood, and the value is displayed in the bottom right corner. Since there is a 168 out of the total 225 that are the inverse, the cells output appears to have a 74.667% (168/225) correlation to the inverse of A, as predicted.

The relationships for B and C also follow the predicted pattern, showing a near perfect 50% correlation to the respective cell's input and inverse, which can be seen in the following tables.

Table 5: Bit B comparison to next state for rule 45.

B		Cell is green if it equals B																							
1	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	
1	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	
1	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	
1	0	1	0	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	1	0	0	
1	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	
1	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	0	0	0	
1	0	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	0	
1	1	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	
0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	0	1	1	
0	1	1	0	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	
8	2	3	4	3	3	5	6	5	5	4	5	3	5	3	5	6	3	3	4	6	6	5	6	7	115

Table 6: Bit C comparison to next state for rule 45.

C		Cell is green if it equals C																							
1	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	
1	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	
1	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	
1	0	1	0	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	1	0	0	
1	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	
1	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	0	0	0	
1	0	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	0	
1	1	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	
0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	0	1	1	
0	1	1	0	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	
3	5	4	3	2	4	3	6	6	5	3	2	4	6	6	7	5	7	4	4	5	4	4	8	3	113

This correlation could suggest a relationship between the truth table and the data generated using said the rule. However a single test is too small of a sample to be the bases of any assumptions. To this end, multiple test were performed with multiple rules, and they all showed the same relationship between the truth table's "bit significance" and the data produced by a given rule. Table 7 shows the bit significance for every balanced rule in the 3-bit rule space. The highlighted rules are the rules that produced good results as shown in [3].

Table 7: List of all balanced 3-bit rules

Rule	A	B	C	Balance
15	-8	0	0	0
23	-4	-4	-4	0
27	-4	-4	0	0
29	-4	0	-4	0
30	-4	0	0	0
39	-4	-4	0	0
43	-4	-4	4	0
45	-4	0	0	0
46	-4	0	4	0
51	0	-8	0	0
53	0	-4	-4	0
54	0	-4	0	0
57	0	-4	0	0
58	0	-4	4	0
60	0	0	0	0
71	-4	0	-4	0
75	-4	0	0	0
77	-4	4	-4	0
78	-4	4	0	0
83	0	-4	-4	0
85	0	0	-8	0
86	0	0	-4	0
89	0	0	-4	0
90	0	0	0	0
92	0	4	-4	0
99	0	-4	0	0
101	0	0	-4	0
102	0	0	0	0
105	0	0	0	0
106	0	0	4	0
108	0	4	0	0
113	4	-4	-4	0
114	4	-4	0	0
116	4	0	-4	0
120	4	0	0	0
135	-4	0	0	0
139	-4	0	4	0
141	-4	4	0	0
142	-4	4	4	0
147	0	-4	0	0
149	0	0	-4	0
150	0	0	0	0
153	0	0	0	0
154	0	0	4	0
156	0	4	0	0
163	0	-4	4	0
165	0	0	0	0
166	0	0	4	0
169	0	0	4	0
170	0	0	8	0
172	0	4	4	0
177	4	-4	0	0
178	4	-4	4	0
180	4	0	0	0
184	4	0	4	0
195	0	0	0	0
197	0	4	-4	0
198	0	4	0	0
201	0	4	0	0
202	0	4	4	0
204	0	8	0	0
209	4	0	-4	0
210	4	0	0	0
212	4	4	-4	0
216	4	4	0	0
225	4	0	0	0
226	4	0	4	0
228	4	4	0	0
232	4	4	4	0
240	8	0	0	0

All the highlighted rules follow one of two patterns in terms of their significance. Either it has a 0 significance for each bit or it has a -4 significance for exactly one of the neighboring bits of the center bit for the neighborhood. This suggest that good rules will show either a low or no correlation to the input bits.

For rules similar to rule 15 with any bit significance of -8 for any bit, this means that the rules produces the invers of that bit, in this case A, 8 more times than it produces a bit that is the same. Since there are only a possible 8 states for a 3-bit neighborhood, this means the logic for such a rule would simply be that the next state equals the inverse of A. The inverse is implied with a +8 significance, that the output is always equal to bit A. This is not random data, the data is simply being shifted one cell to the right in the second case, while in the first case that data is shifted and inverted.

The next scenario is the case of rule 29 and similar rules. These rules show a -4 significance on two bits, bits A and C in this case. While this relationship is a bit more complicated, the truth table reveals why it is not random.

Table 8: Truth table for rule 29.

A	B	C	out
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

The cells highlighted in red show that every time that bits A and C hold the same state, the next state is always the inverse of the state that those two bits hold. This leads to the next state bit having a strong inverse correlation to both A and C, which leads to very weak random number generation.

During the extended testing phase, a rule was discovered that possessed the desired characteristics of a perfect 50% ratio between all input bits and their inverse or a bit significance of 0 for all three bits. The rule 150's logic is a three way exclusive or relationship ($A \oplus B \oplus C$) between all three inputs. This creates a perfect balance between all three inputs, which could theoretically create better test results.

Section 3.4: Testing the Theory

Although the discovery of this potential relationship could be significant, it amounts to little if there is no proof to suggest that a rule with a 0 bit significance performs any better than other rules. To test this, various data was generated and tested with various rules with various bit significances. Because Die Hard test results did not produce any clear results, a more strenuous testing method was used. The tests RaBiGeTe test runs are much more difficult to pass and thus getting an overall pass requires the data to have an exceptionally high perforce. Below are graphs displaying the resulting P-value charts for RaBiGeTe as compared to Die Hard for the exact same data for rules 45, 30, and 150. These graph show that the rule with the net 0 bit significance does noticeably better on these test.

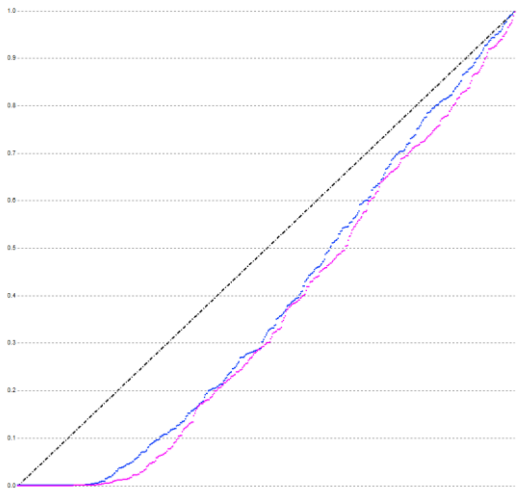
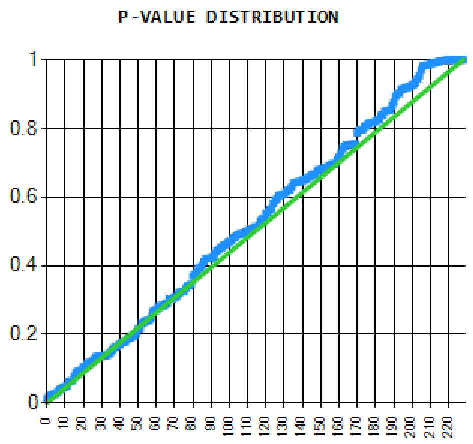



Figure 5: Test results for RaBiGeTe (left) and Die Hard (right) for rule 45.

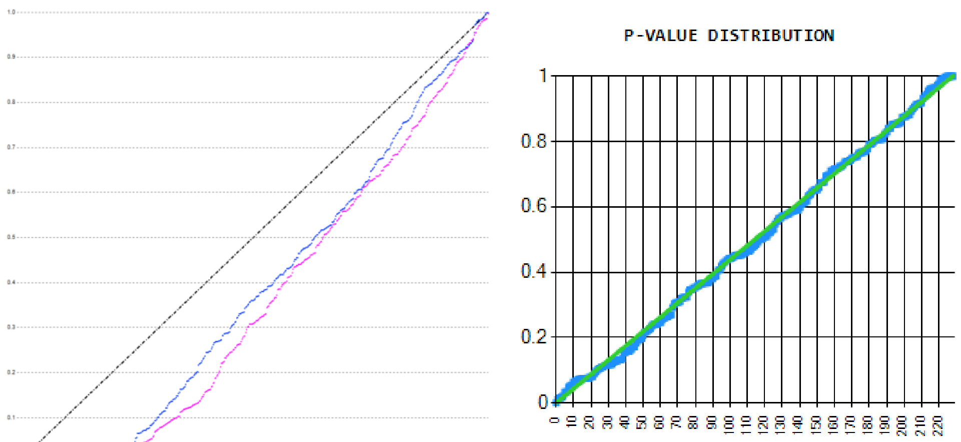


Figure 6: Test results for RaBiGeTe (left) and Die Hard (right) for rule 30.

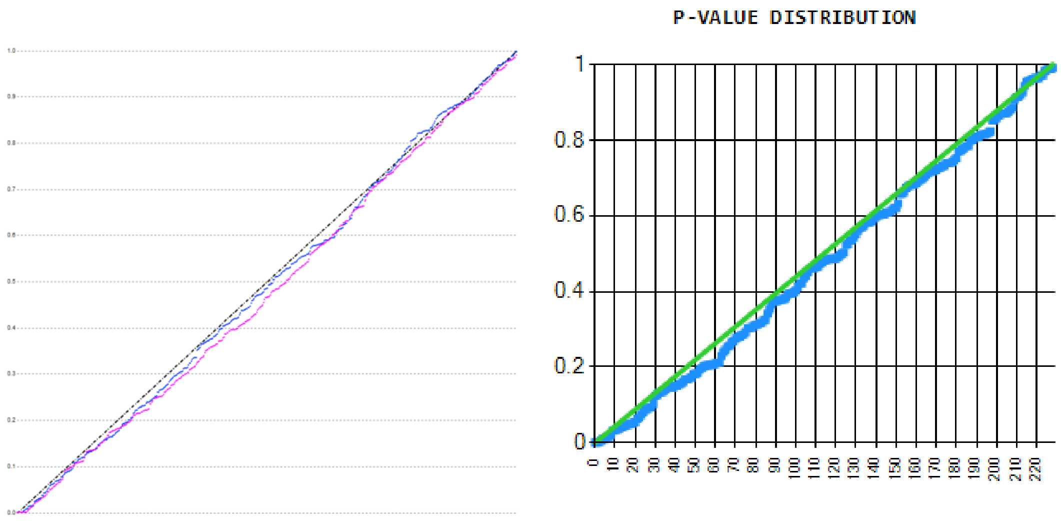


Figure 7: Test results for RaBiGeTe (left) and Die Hard (right) for rule 150.

Although it is not conclusive, these results would suggest that it would be worthwhile to continue these test on larger rule spaces.

CHAPTER FOUR: CODING

Section 4.1: Workbook and Driver

Various programs were used for the generation and testing of the data. Most of the initial testing was done with the use of two programs, the CA workbook and driver. Created by Stephen Faulkenberry [2], the workbook and driver are specially designed programs created to work in tandem to make generating 3-bit CA data easy and efficient. The workbook, written in C#, is an interface program that allows the user to choose from various configurations, lengths, starting string, and test type. The program has three main operating modes: diehard test, NIST test, and automation. The first two modes are designed to run a single test for the corresponding test type, and give a detailed report of the specific test. The automation page is designed to run one specific test various times, for different parameters and give a simplified report for each test run. These reports are much smaller for automation and consist only of the number of passes in each test and the overall P-value for the test. Using this test method, massive amounts of test could be run with relative ease.

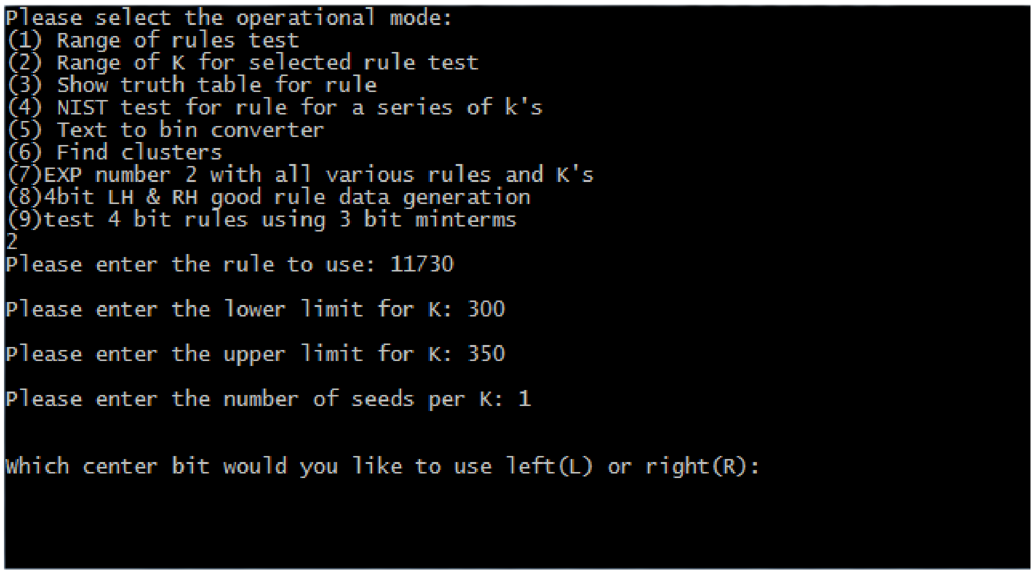
The driver is the program that generated the data and ran the Diehard test. The driver works based on a configuration string generated by the workbook. The string contained all the information for the test to be performed, such as the ID for the test type, seed length/content, and test type. The driver starts by reading in this string, breaking it down into its base components, and generate the data based on the parameters given. This structure is what allows the driver to be used independently, without the workbook when require or with use of another program.

While many minor changes and functions were added to the workbook throughout the course of the research, the largest changes happened within the driver. The driver was initially designed only for windows systems and when the need arose for use on a Linux based system for use on a super computer, compatibility became an issue. While functionality was not changed, many of the core functions in the program were windows reliant and had to be recoded. Additionally the workbook had no functionality for the super computer since the computer had to be controlled via command prompt. This required a new automation program to be created that could run the driver only use text based command and would lead to the development of a program that would eventually become the 4BitAllRules program used for all future 4BitTesting with Die Hard.

Section 4.2: 4BitAllRules

Although the workbook and driver combination boasted a lot of features and functionality, certain limitations made adding new functionalities range from difficult to impossible. Due to the focus of the research when the workbook was being developed, most of the functionality was designed around 3-bit designs and variations. In addition, since the workbook was GUI based, it could not work on systems that did not support GUIs. Due to these issues, a new program had to be designed. A command prompt based program that could perform similar functionality to the workbook but designed around 4-bit CA.

4BitAllRules was essentially the culmination of several smaller programs designed with a much smaller scope. The initial testing of the 4-bit rule space had many approaches that involved testing subsection of the rule space, but when the decision was made to test the entire rule space, this program was created.



```
Please select the operational mode:
(1) Range of rules test
(2) Range of K for selected rule test
(3) Show truth table for rule
(4) NIST test for rule for a series of k's
(5) Text to bin converter
(6) Find clusters
(7)EXP number 2 with all various rules and K's
(8)4bit LH & RH good rule data generation
(9)test 4 bit rules using 3 bit minterms
2
Please enter the rule to use: 11730

Please enter the lower limit for K: 300

Please enter the upper limit for K: 350

Please enter the number of seeds per K: 1


Which center bit would you like to use left(L) or right(R):
```

Figure 8: 4BitAllRules program main menu and testing input screen.

Figure 8 shows the main menu and inputs for running a series of tests for rule 11730. The program has the ability to run automated test based on a range of rules, ranges of seed lengths, create truth tables, find rule clusters and show bit significance. Although not as user friendly as the workbook, 4BitAllRules is generally less finicky and more easily modified.

Section 4.3: Support Programs

Although many features are built into 4BitAllRules, since the initial goal was to create a workbook replacement, the CADriver program is still required to run any sort of testing. In addition to CADriver, several other small support programs were created in order to take advantage of 4BitAllRules, but not add any complexity to the code for 4BitAllRules. Although most were variations on how to test subsections of the rule space, others include a K-map generator, multithread support, and a program to take a large list of rules and organize them into clusters and remove duplicates.

CHAPTER FIVE: RESULTS

Section 5.1: Overview

The ultimate goal of this research was to determine which rules in the 65536 rule space provide the best use for CA based encryption. To do this several rounds of testing and organizing were required. The first task, which was the largest and most difficult to deal with, was running the initial battery of tests. Since results from previous tests showed that CA based random number generation tends to have a stable level of complexity when the seed length is greater than 300, the testing preformed was all done at lengths below 300 [4]. This was done with the assumption that if a rule was able to produce good results at seed lengths lower than 300, then results at length greater than 300 would continue to be satisfactory.

After the initial list of rules was obtained, it was important to organize the rules into clusters. This is due to the relationship between rules in a cluster. To do this, a modified version of 4BitAllRules' cluster finder program was made to sort and organize the rules. During this step an additional measure was taken to filter out linear rules. This was done to limit the number of rules that would require further testing. Nonlinear rules were chosen over linear due to the fact that linear rules possess a liability of being able to be reversed.

The last step in the process was to organize the list of rules into three categories. The three categories are left hand rules, right hand rules, and rules that work with both. For this step in the process, every rule produced was tested at a seed length of 350 for both left hand and right hand structures. Once the results were obtained, the data was organized and all rules were placed in the appropriate category based on the results of the test.

Section 5.2: Initial Testing

The initial testing was performed on the entire rule space for both left and right handed structures. This meant that for every seed length that would be tested, a minimum of 131072 tests would need to be performed with 65536 tests being ran on the right hand structure and 65536 being run for the left handed structure. Since this quickly produced an unwieldy amount of data, the test was designed to give usable results with the smallest amount of testing required.

The testing size was ultimately decided to be limited to seed length between 30 and 300, with every test being performed at increments of 10. These seed lengths were decided based on the results of [4] which showed that in classical 3bit 1D CA, once the seed length no longer had a noticeable effect on the passes of test using Diehard. This would mean that testing above 300 would likely show similar results to the test done at seed length of 300 itself. Consequently all data that would distinguish the rules, would be seen at seed lengths lower than 300. The interval of 10 was selected to keep testing to a minimum as this would still be able to give a sample of how the rule performed at various given lengths while still being able to have the testing done within a reasonable amount of time. Even this limited testing took over 2 weeks per structure while running nonstop on MTSU's super computer. This meant that testing every seed length between 30 and 300 would take roughly 5 months per structure which required too much time for this study.

In order to organize this data, the data was divided up into sub sections based on the rules numerical values. The subsection contained 1000 rules each, starting with the first subsection consisting of rules 0-999, the next was 1000-1999, and it continued until 65536 was reached. Since the data for rules 64000 and higher was predicted to not produce any passing results, the 64000 group contained the remaining 536 rules from 65000-65536.

Once all the data was appropriately divided up, a pivot table was designed for each subsection. The table listed each rule as a row, and used the columns to display the number of test passed in a run of diehard for each given seed length. At the end of the table in the final column, the averages of all the passes was calculated. Additionally the P-values produced from the test were used as a filter to farther limit the data to be analyzed. When a test fails diehard, a P-value of either 0 or 1 is produced. Using this output, all rules that did not pass a single test could be filtered out before the analysis. An example of these tables can be seen in table 9 below which shows the results from the 21000-21999 range.

Table 9: Results of Die Hard test for left hand structure rules 21000-21999.

Row	80	110	120	140	150	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300	(bla	Grand To
21165					209	219	226	226			223	215	214	213	223	225	214	224	223	223		**219.786**
21420				215	219		222	223		210	227	219	224	211	224	222	220	221	225	225		**220.467**
21675					224	218	218	220		215	225	219	226	213	218	222	226	219	225	221		**220.6**
21850	222	222	226	227	226		224	224	223	221	227	225	223	223	227	223	224	222		226		**224.167**
21862	219	225	227	227	223		226	225	221	222	223	229	222	229	227	228	224	224		227		**224.889**
21865	213	223	228	220	223		222	224	219	219	223	223	222	222	224	227	226	221		221		**222.222**
21866	225	228	225	219	228		225	225	226	222	229	228	226	227	223	226	227	223		222		**225.222**
21910	217	224	226	227	223		222	220	225	224	216	225	224	222	223	222	228	217		224		**222.722**
21913	217	224	224	218	220		228	223	224	220	226	224	224	224	223	224	227	226		217		**222.944**
21914	219	226	228	225	223		223	222	224	222	227	226	227	221	226	219	226	226		226		**224.222**
21925	219	226	219	218	219		226	224	224	222	226	220	225	223	227	223	219	223		212		**221.944**
21926	226	223	219	227	225		227	227	225	221	225	228	225	226	221	220	223	222		222		**224**
21929	222	222	219	220	224		225	226	223	227	218	225	228	222	224	222	225	224		223		**223.278**
21930		225							219			226	217		222				223			**222**

Once the table is made, it is much easier to tell which rule may be worth looking at. Rules such as 21850 show a very high potential with multiple passes before seed length of 150 and with passes on most seed lengths. While a rule like 21930 would require more testing to determine its usefulness as it does show some potential but on a much smaller scale.

Section 5.3: Clusters

Once all the base rules had been found, it is important to group them into their respective clusters. This step is important because rules from the same clusters share many of the same characteristics from the number of GOE’s and cycle lengths, to how they tend to perform when being tested. Additionally there is the potential that do to the limited nature of the test, a rule that belongs in the same cluster as a rule in the list may not have met the criteria from the initial test, but may still be useable for data encryption. Furthermore, since rules from the same cluster perform similarly in test, a cluster leader can be used for test to compare the cluster to other clusters. In addition, linear rules were phased out during this process, since the goal of this research was to find the best rules for use in encryption and linear rules are not good for independent encryption use.

Section 5.4: Left and Right Applications

After reorganizing all of the rules into their appropriate clusters, some new rules were added, which required some testing to determine if the new rules could meet the criteria. Additionally during the clustering phase, the data for left and right hand structures was combined in order to make the list all inclusive. This means that all rules needed to go through one final round of testing to determine their usefulness.

For this test, all the rules were ran in Die Hard on 5 different seeds for both left and right hand structures. The average of the 5 test was taken and each rule was placed into one of 4 categories and placed into table 10.

The first of these is the useless rules that failed both left and right. These rules were left uncolored in the final table. Next is the green rules in the table, which are the rules that passes only when applied to a left hand structure. The red rules represent the right hand structure rules, while yellow is used for rules that work in both structures. The final count was 57 clusters with 11 right hand only rules, 60 left hand only rules, and 291 rules that work with both. This brings the count of good rule up from 16 to 378.

Table 10: All passing results grouped into clusters.

1020	22102	49215	38293	64515	43433	16320	27242
2040	22118	57375	39317	63495	43417	8160	26218
2550	25942	36975	38233	62985	39593	28560	27302
3060	26198	53295	38297	62475	39337	12240	27238
3315	25957	12495	22873	62220	39578	53040	42662
4590	21850	34935	42325	60945	43685	30600	23210
5100	22106	51255	42389	60435	43429	14280	23146
6885	26201	22695	26009	58650	39334	42840	39526
7140	26202	55335	42393	58395	39333	10200	23142
8670	22870	33915	38245	56865	42665	31620	27290
9180	23126	50235	38309	56355	42409	15300	27226
9945	23141	25755	22949	55590	42394	39780	42586
12750	22874	35955	42341	52785	42661	29580	23194
13005	23129	19635	26021	52530	42406	45900	39514
13116	7710	49971	34695	52419	57825	15564	30840
14025	23145	27795	27045	51510	42390	37740	38490
14028	23160	52371	57765	51507	42375	13164	7770
15043	27465	15523	27945	50492	38070	50012	37590
15045	27225	23715	26025	50490	38310	41820	39510
16065	27241	31875	27049	49470	38294	33660	38486
17340	22166	49725	38549	48195	43369	15810	26986
17595	21925	8925	23125	47940	43610	56610	42410
17850	21926	41565	39509	47685	43609	23970	26026
18105	22181	25245	23189	47430	43354	40290	42346
18795	13763	10605	15443	46740	51772	54930	50092
19125	26261	21165	22169	46410	39274	44370	43366
19275	13251	11565	15411	46260	52284	53970	50124
20655	21913	2805	26197	44880	43622	62730	39338
21675	21929	10965	27221	43860	43606	54570	38314
23070	13212	34725	50739	42465	52323	30810	14796
24225	26281	31365	27289	41310	39254	34170	38246
24735	22933	1785	22117	40800	42602	63750	43418
26005	18615	22105	4845	39530	46920	43430	60690
27029	26775	22121	5865	38506	38760	43414	59670
28050	27046	46665	39529	37485	38489	18870	26006
29835	22953	11985	27237	35700	42582	53550	38298
30090	22954	44625	43621	35445	42581	20910	21914
31110	27034	40545	42601	34425	38501	24990	22934
33405	38485	16830	21910	32130	27050	48705	43625
35190	42326	37230	38234	30345	23209	28305	27301
36210	42342	45390	39258	29325	23193	20145	26277
39015	42329	6630	25946	26520	23206	58905	39589
40035	42345	14790	26970	25500	23190	50745	38565
41055	39253	1530	21862	24480	26282	64005	43673
42075	39269	9690	22886	23460	26266	55845	42649
43095	43349	5610	21866	22440	22186	59925	43669
44115	43365	13770	22890	21420	22170	51765	42645
46155	39273	11730	26982	19380	26262	53805	38553
47175	43353	7650	25962	18360	22182	57885	39573
50003	33735	13628	7230	15532	31800	51907	58305
51000	38566	58140	39574	14535	26969	7395	25961
51075	50083	15900	14908	14460	15452	49635	50627
52020	42646	54060	38554	13515	22889	11475	26981
54825	38569	27540	27286	10710	26966	37995	38249
55080	38570	60180	43670	10455	26965	5355	21865
57630	39318	34680	38502	7905	26217	30855	27033
61710	39322	36720	42598	3825	26213	28815	22937

CHAPTER SIX: CONCLUSIONS

3-bit based CA has proven to have use in random number generation, and 4-bit shows even more potential with its 378 rules as opposed to 16. While 3-bit's complexity is limited, there are many complex variations of its application and most if not all are applicable to 4-bit. With 4-bit's increased rule space, it could mean that applications that showed extremely high potential such as 3D and 2D CA could have even greater complexity. More testing will always be required to verify results, but the potential to use 4-bit in place of 3-bit CA could make a significant impact on data encryption as the increase in rules will makes reversing the encryption exponentially more complex.

Even though 4-bit does add many new rules that can be used for encryption, there is little reason to stop at 4-bit. As technology advances and more efficient methods of testing rules become available, the potential to develop more complex CA will always grow. There is no limit to seed length of a neighborhood that can be used outside of technical limitations. This means that larger bit neighborhood can scale with technology as long for the foreseeable future.

REFERENCES

[1] J. V. Neumann, “The Theory of Self-Reproducing Automata“, Urbana and London: University of Illinois Press, 1966.

[2] S. Faulkenberry, “Optimal Analysis, Coding and Testing for Encryption” master’s thesis, Middle Tennessee State University, 2016, http://jewlscholar.mtsu.edu/handle/mtsu/4888

[3] K. Salman, "Elementary Cellular Automata (ECA) Research platform," *Journal of Selected Areas in Software Engineering (JSSE),* vol. 3, no. 6, 2013.

[4] D. Nichols, “Optimum Cellular Automata Configurations for Encryption” master’s thesis, Middle Tennessee State University, 2015, http://jewlscholar.mtsu.edu/handle/mtsu/4527