Design of a secure symmetric key block encryption algorithm using reversible partitioning cellular automata

Federico Reyes Gómez

Abstract

Cryptography allows us to securely transmit terabytes of information on the web. Military secrets, corporate information, and most of our online presence is encrypted. Cellular automata are simulations with applications in fields like the simulation of biological processes, crystal lattices, and gas interactions. They exhibit high entropy and unpredictability which makes them good candidates for use in encryption. Cellular automata are usually hard to reverse, but partitioning cellular automata with a bijective rule are easily reversible. This study used a reversible partitioning cellular automata to design a block encryption algorithm using a variation of the cipher block chaining method of operation. A secret key is generated using user-inputted values such as the rule list and generations for the cellular automata and a text password. To decrypt the file, both the secret key and the exact cellular automaton parameters are required. The algorithm was tested for security by analyzing entropy, sensitivity to user input, satisfaction of the avalanche criterion, and by running the ENT, NIST, and DIEHARD statistical test suites. The study passed all the algorithm tests and most of the statistical tests, showing promise in the use of reversible partitioning cellular automata for encryption.

Introduction

As more and more information is transferred across the internet, there is a greater need to develop secure ways to transmit this information. The continuous study of cryptography is essential since huge corporations, governments, the military, and most of us rely on encryption to keep our information safe from attacks. Quantum computing is one technology that threatens to undermine current encryption mechanisms due to the massive increase in processing power it'll bring (Boutin, 2016). Encryption has recently gotten increased coverage in the media due to hacking attacks on major companies like Dropbox, Sony PSN, LinkedIn, and Ebay (McCandless). Encryption has also become a controversy in global politics with some proponents arguing that security agencies must have backdoors to protect national security, while others argue that personal privacy must be protected ("When back doors backfire").

Cellular Automata

Cellular automata (CA) were originally introduced to study self-replication by John Von Neumann (Wang et. al, 2013). They reached their highest level of popularity years later with John Conway's "Game of Life", a cellular automaton intended to model natural cell growth. Among their many applications lie the simulation of biological processes, crystal lattices, and gas interactions. They can be used as a cryptographic tool due to the fact that they can simulate complex behaviors (Sarkar 2000).

"Life-like" Cellular Automata

Conway's "Life" consists of a 2D grid where each cell is either in the "On" or "Off" state, indicating whether the cell is "Alive" or "Dead". The cellular automata is "evolved" using simple rules based on each cell's surrounding neighbors over a fixed number of generations. The state of each cell in the next generation is based on the number of live neighbors around it. Many neighborhoods exist, but the Game of Life uses the Moore Neighborhood, meaning that the rule is calculated using the eight neighbors around each cell (Toffoli and Margolus, 60).

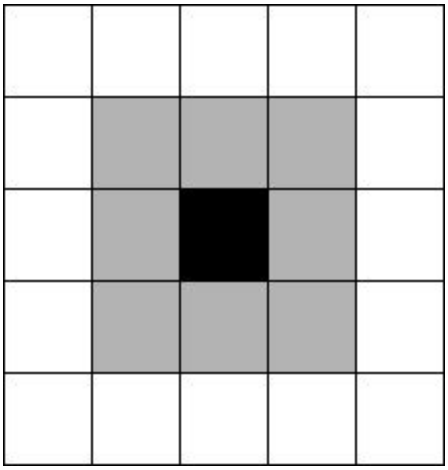

Figure 1: The Moore Neighborhood. The cell whose state will be calculated is in black, with the neighboring cells in gray. (Rocha 2015)

The rule for a "life-like" cellular automata is usually defined as Bx\Sy, where x is the number of neighbors that are alive that is needed for a dead cell to be "born", and y is the number of live neighbors needed for a live cell to remain alive, or "survive". The rule for Conway's Game of life is defined as B3\S23, meaning that exactly 3 neighbors are needed for a dead cell to be born and either exactly 2 or 3 live neighbors are needed for a live cell to remain alive. The new configuration of the CA is calculated by applying the rule to each cell individually (Machicao et. al, 2012). Sometimes, a problem arises when the neighbors of the outermost cells are calculated. To avoid any loss of information, an edge wrapping method was used, creating a 2D hypertorus.

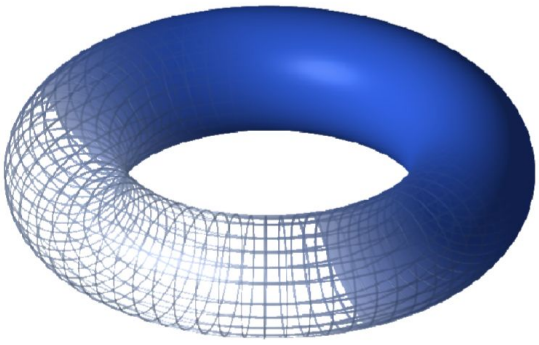

Figure 2: Hypertorus. It represents the physical representation of a wrapped two-dimensional grid. (Luckett, 2007)

Partitioning Cellular Automata

A partitioning cellular automata (PCA) uses a different neighborhood and rule. Designed by Norman Margolus, the Margolus neighborhood divides the grid into smaller 2 X 2 grids and applies a transformation to that grid as a whole. To avoid stagnation, however, the partitions alternate between generations as shown in Fig. 3 (Toffoli and Margolus, 119).

Odd x_{x-1}, y_{y-1}	Odd x_x, y_{y-1}	
Odd x_{x-1}, y_y	x_x, y_y	Even x_{x+1}, y_y
	Even x_x, y_{y+1}	Even x_{x+1}, y_{y+1}

Figure 3: The Margolus Neighborhood. The macro-cell used in the odd iteration is labeled as odd while the macro-cell used in the even iteration is labeled as even. (Luckett, 2007)

Since this automata operates on the smaller grids, the rule is defined differently as D = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p}. The 2X2 grids have 16 possible states, designated with numbers from 0 to 15. The value of the first number in the rule, *a*, defines what the next configuration is for a current configuration of 0. Considering a rule of D = {0, 8, 4, 3, 2, 5, 9, 7, 1, 6, 10, 11, 12, 13, 14, 15} , a grid with current configuration 0 stays at configuration 0, current configuration 1 changes to configuration 8, and so on.

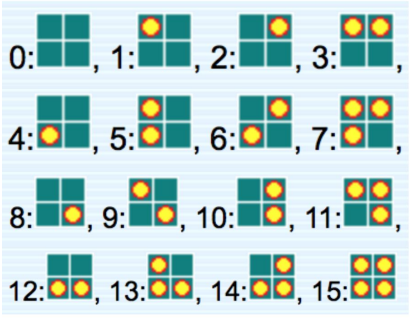

Figure 4: Margolus Configurations. With 4 cells, there are only 16 different configurations for the macro-cell, designated by numbers 0-15 (Wojtowicz, 2001)

Application to cryptography

The use of cellular automata in cryptography isn't new. Stephen Wolfram proposed using 1D CA's for secret keys (Wolfram, 1986). CA's most popular use is as a pseudo-random number generator (PRNG), as described in (Seredynski, Bouvry, Zomaya, 2004; Sen et. al, 2002; Hortensius, 1989; Tomassini et. al, 2000). 2D cellular automata have also been used, most commonly as life-like CA's (Seredynski, Pienkosz and Bouvry, 2000; Wang et. al, 2013; Machicao et. al, 2012; Faraoun "Design of a Fast One-pass Authenticated and Randomized Encryption Schema" , 2014; Faraoun "Fast Encryption of RGB Color Digital Images", 2014). Reversibility of the CA has been the main issue with these designs, however, with most of them having to use a 2^{nd} order technique to make them reversible. Reversibility is crucial in an encryption algorithm because no information can be lost in the decryption process. Partitioning CA's are unique in that certain rules make it simple to reverse as compared to implementing a second order technique on non-reversible life-like cellular automata. As long as the rule is bijective, having one and only one output for each input, then the CA is reversible (Luckett, 2007). Partitioning CA have been used in cryptography by Bouchkaren et. al (2014), but while cellular automata seem to exhibit random behaviors, it doesn't create an efficient encryption method on its own, even if the rule list and number of generations is kept secret (Luckett, 2007).

The purpose of this study was to create a private key encryption algorithm using a block chaining method and reversible partitioning cellular automata using the margolus neighborhood that would pass tests for security using entropy, the avalanche test, sensitivity tests, and statistical test suites.

Methods and Materials

Algorithm Design

Key Generation

To generate the key, the user specifies 4 pieces of information for the encryption: the rule list for the cellular automata, the number of generations to be evolved, a text password for user uniqueness, and the file to be encrypted. By having multiple aspects defined by the user, it makes it harder for a possible attacker to guess the correct values. The number of possible bijective rules for a PCA is $15!$ or 1.3076744×10^{12}. This means that an attacker must test more than

50% of the possible rules, over 653837184000, to correctly guess the user's rule. The password is made up of 32 8 bit unicode characters, giving 256 character possibilities for each of the 32 spaces. This means there are 256^{32} or 1.1579209×10^{77} possible passwords, and an attacker must try most of those in order to guess correctly. Finally, the set of possible generations is, in essence, infinite, although increasing the number of generations linearly increases the time it takes to process the file, thereby limiting the feasible number of generations that can be used. The result of the key processing as shown in Fig.6 is a 256 bit key that is used to both encrypt and decrypt the file. The process makes heavy use of the Exclusive OR (XOR) operator to combine blocks.

A	B	A ⊕ B
T	T	F
T	F	T
F	T	T
F	F	F

Figure 5: XOR Truth Table. To XOR two dishes, they're juxtaposed using the list indexes and the result is the value obtained by XORing the values in each dish (Luckett, 2007)

To process the key, a scrambled user block is created by first adding the number of generations to the rule list. Each number is then represented in binary and along with random padding as described in ISO 10126, made into a 256 bit block. To scramble it, this block is than evolved with the life-like "Fredkin" rule for the amount of user-specified generations. The Fredkin rule as defined by B1357\S02468 was found to exhibit high entropy and chaos, making it ideal for one-way scrambling of information (Machicao et. al, 2012). The user-supplied password is also used by duplicating it until it reaches a length of 32 characters, and then representing the unicode character in binary as a 256 bit block. This block is than XOR'ed with a block of random bits created by the python programming language's *os.urandom* to create a scrambled password block. The operating system's urandom pseudo-random number generator isn't the best source of randomness, being only a PRNG, but this ensures that encrypting the same file with the same rule list, number of generations, and user password twice will result in different encrypted blocks. This scrambled password block that is created is then further XOR'ed with a SHA256 checksum on the file, and finally that result is XOR'ed with the scrambled user

block. The checksum's original intent is to measure file integrity and authenticity, but the values it generates are high in entropy and thus useful for encryption ("How To SHA256SUM"). Even if an attacker were to have the processed key, they wouldn't be able to get the rule list and generations needed to successfully decrypt the file.

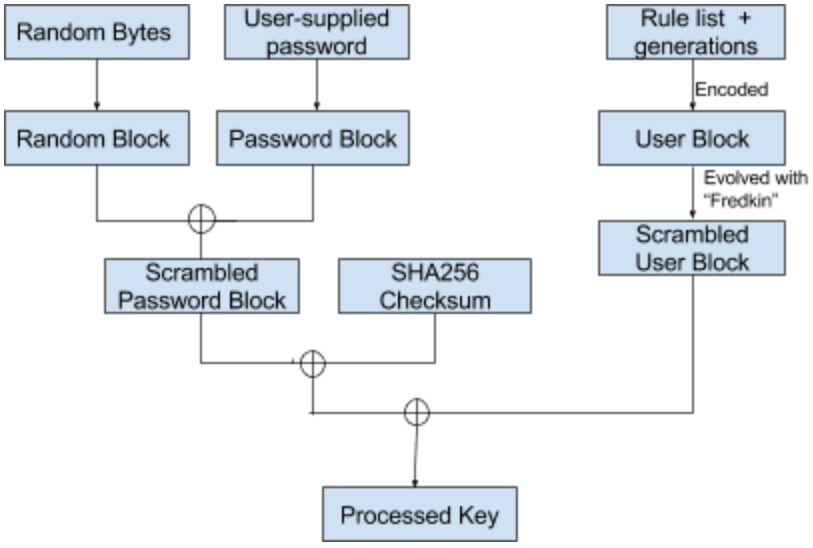

Figure 6: Key Processing. The user defines a password, rule list, and generations which are processed into dishes, XORed with a random block and a checksum to get a processed key

Encryption

The algorithm uses a variation of the Cipher Block Chaining mode of operation invented by Ehrsam, Meyer, Smith and Tuchman in 1976. The main advantage of this mode is that identical blocks are encrypted differently because each generated cipher block is XOR'ed with the plaintext before evolution. This helps propagate small changes and contributes to satisfying the avalanche effect. Instead of using an IV that is known to the attacker, this version uses the secret key that was previously generated as the IV. This adds an extra layer of security in the case that the rule list and generations is known by the attacker. The CA operates on binary on and off states so the contents of the file are read as binary to make the plaintext. Since the key is 256 bits, the plaintext is split up into 256 bit blocks. If the plaintext length is not divisible by 256 bits, the last block is padded as described in ISO-10126. The key is XOR'ed with the first block, or "dish", and then evolved with the PCA. That result is XOR'ed with the next dish and that is then evolved, and so on.

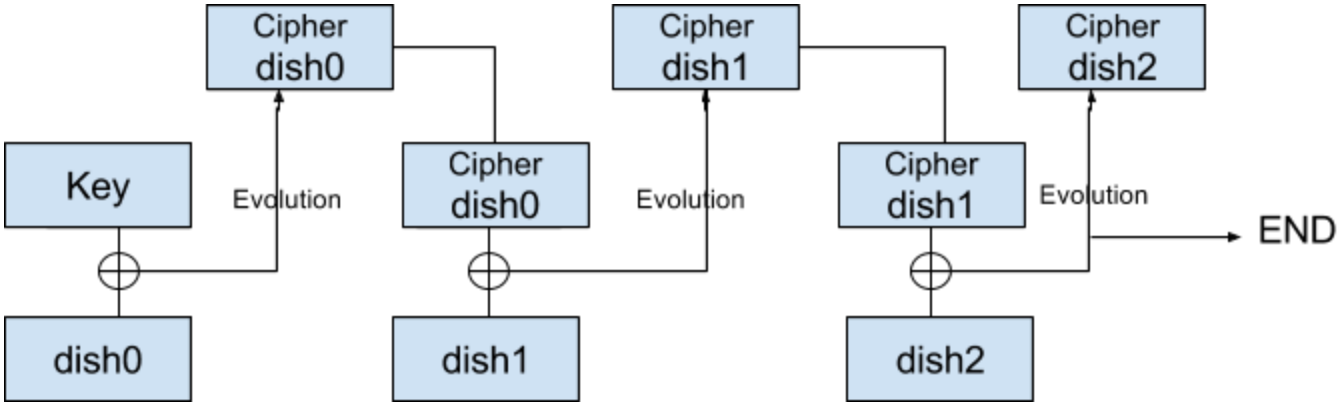

Figure 7: Encryption Process. The processed key is used in the CBC mode of operation, propagating through the whole chain. The previous cipher dish is XORed with the plaintext before evolution and in the end, all the cipher blocks are appended and written to a file.

Decryption

The decryption process is similar to encryption, but the order changes slightly. The last block in the ciphertext is evolved first, using the same number of generations and the inverse rule list, D'. Since D should be one-to-one, the inputs and the outputs must be switched. If D were {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0}, then D' would be {15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}. The decrypted dish is the result of XOR'ing the evolved dish and the second to last cipher dish. The second to last cipher dish is then evolved, XOR'ed with the preceding cipher dish, and so on. In the end, the padding is removed by examining the last byte and then removing that many bytes from the last dish. The blocks are then put together and the binary is written back to a file.

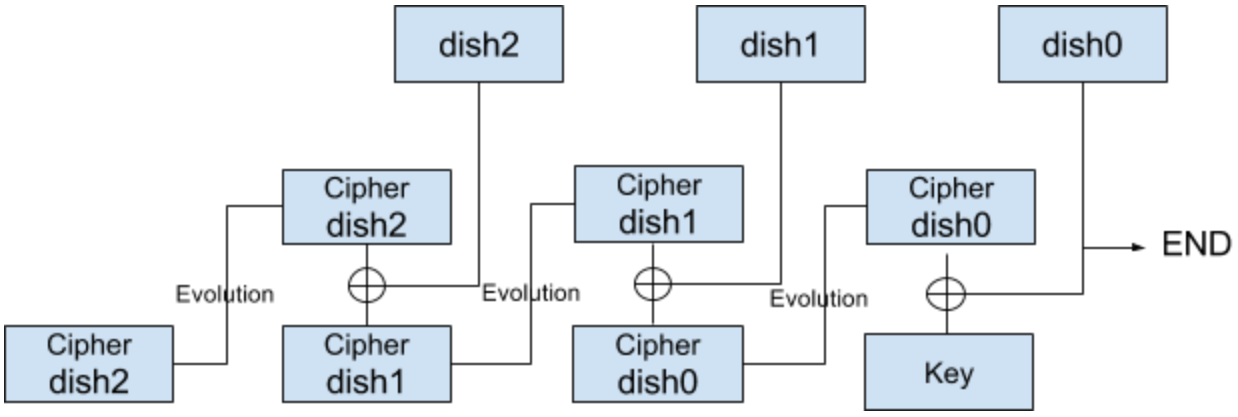

Figure 8: Decryption Process. Similarly to encryption, decryption uses the processed key, but does the evolution first and then the XORing. In the end, the processed plaintexts are appended and written to a file.

Testing the algorithm

Any encryption algorithm can be breached with enough computing power, but there exist some tests that help provide evidence of an algorithm's security. Security was tested by evaluating 1)The Shannon entropy of the ciphertext, 2) the satisfaction of the avalanche criterion, 3) sensitivity to user input changes, and 4) statistical analysis of randomness of the ciphertext using the ENT, DIEHARD, and NIST test suites for randomness. Tests were completed by myself using original code or command line tools on a Google Cloud Platform virtual machine.

Shannon Entropy

Shannon Entropy is the measure of randomness or disorder of information. Information like text that is easily predictable has low entropy. Information like a compressed file has fewer patterns and thus higher entropy. If the information is encoded in 8 bit binary, the highest possible entropy is a value of 8. (Machicao et. al, 2012). The equation for entropy is:

$$H(m) = -\sum_{i=0}^{2^n-1} p(m_i) log_2 \frac{1}{p(m_i)}$$

Equation 1: Shannon Entropy. Entropy is a measure of randomness that uses the probability that each byte will appear in the information. (Machicao et. al, 2012)

where m is the message to be encrypted and $p(m_i)$ is the probability that character m_i exists in the message. Entropy was tested via the ENT Test suite which is explained later in this section.

Avalanche Test

The avalanche test measures the percent change in ciphertexts when one bit in the plaintext is changed. Since a change of one bit should "avalanche" to larger change, the optimal value to satisfy the criterion is 50 percent. (Faraoun, "Design of a Fast One-pass Authenticated and Randomized Encryption Schema", 2014) If the avalanche criterion is met, that means that both "diffusion" and"confusion" are happening. Diffusion involves the scrambling of information and confusion involves substituting bytes (Luckett, 2007). If the criterion is met, that also means that the algorithm is safe from linear and differential attacks (Faraoun, "Design of a Fast One-pass Authenticated and Randomized Encryption Schema" , 2014). 62 files were tested, of 30 different types in categories like audio, video, text, compressed, pdfs, pkgs, documents,

and executables for all tests. This was done to ensure that file type or initial entropy were not factors in the results. The 256 bit block generated by urandom, the number of generations, the rule list, and password were all kept constant to make sure that they aren't the cause of the avalanche. One bit in the original file was randomly chosen using python's *math.randint* and flipped. The ciphertext of the unmodified file and the ciphertext of the flipped file were then compared and the percentage was recorded.

Sensitivity to user input

Similarly to the avalanche effect, a small change in any of the user input variables should create a big change when decrypting. When decrypting a ciphertext with the wrong key, the wrong password, the wrong number of generations, or the wrong rule list, the result should be significantly different from the correct plaintext, again around 50 percent. To measure sensitivity to the key, one bit in the processed key was flipped like before while keeping all other factors constant. To measure sensitivity to the password, one character was randomly switched to another random character while keeping all other factors constant, including the block generated by urandom. To measure sensitivity to the generations, a random number of generations from 15 to 30 was selected and then either increased or decreased by one while keeping all other factors constant. Finally, to measure sensitivity to the rule list, two values were randomly swapped in a way that still kept it bijective while keeping all other factors constant. Each test was conducted 100 times with a fixed plaintext and for all the tests, the result after decrypting was compared to the original plaintext and the percentage of different bits was recorded.

Statistical Test Suites

The ENT and DIEHARD tests are very commonly used to test statistical randomness, as seen in (Lafe, 1998; Faraoun, "Fast One-pass Authenticated and Randomized Encryption Schema", 2014; Faraoun "Fast Encryption of RGB Color Digital Images" , 2014). The ENT test suite consists of Entropy, Chi Square, Arithmetic Mean, Monte Carlo Value for Pi, and Serial Correlation Coefficient tests (Walker, 2008). The DIEHARD suite consists of 12 different statistical tests as described in (Marsaglia, 1995). The UNIX command line *ent* and *dieharder* tools were used to perform the tests. The dieharder test suite contains all the tests from the DIEHARD suite, but it also includes other tests by Robert G Brown, the author of the dieharder

tool, and other miscellaneous ones as well (Brown). The third test used was the statistical test suite from the National Institute of Standards and Technology, also known as NIST STS (Ruthin et. al, 2010). It's not as widely used as the ENT and DIEHARD suites, but it incorporates sound tests and has been recently updated. The input of these tests came from the ciphertexts of the 61 files that were tested.

Results and Discussion

Shannon Entropy

The average entropy of ciphertext achieved by the algorithm was 7.992023581 while the median value was 7.999841. This is within 0.01 of the optimal value, 8, and indicates that the ciphertext is unpredictable, a desired quality in encryption algorithms.

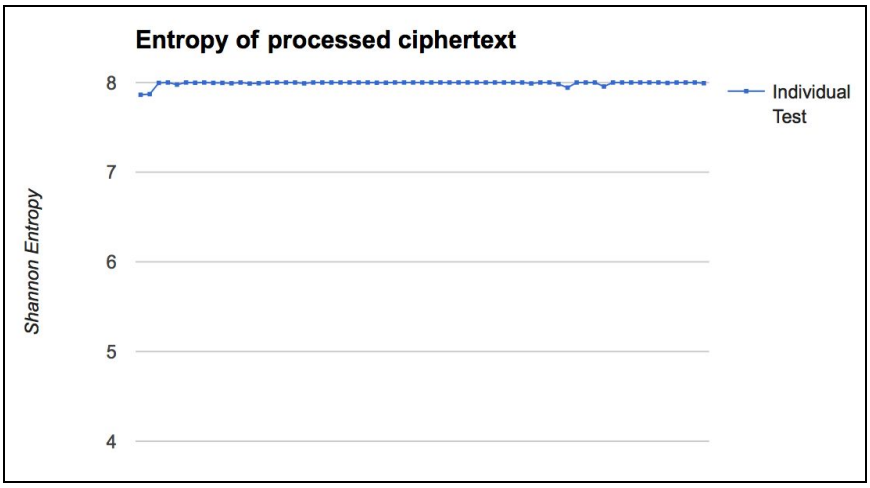

Figure 9: Shannon Entropy Results. Each individual point represents one of the processed ciphertexts. The ENT test was used to calculate Shannon Entropy

Avalanche Test

In testing the files to meet the avalanche criterion, the algorithm achieved an average of 50.14083846 and all results were around 50 ± 3.059439362. This satisfies the avalanche criteria well, showing the algorithm to be resistant to both linear and differential attacks.

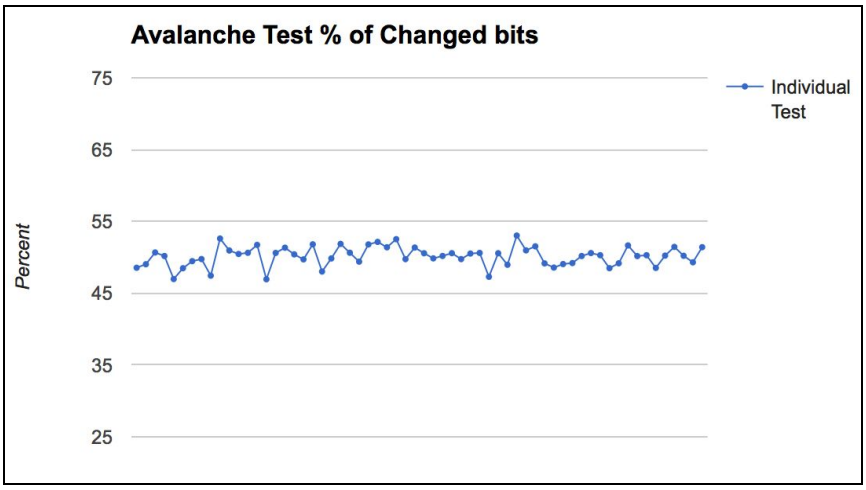

Figure 10: Avalanche Test Results. The avalanche percentage was measured as the percent of bits that change when encrypting an original plaintext and that same plaintext with one flipped bit.

Sensitivity to user input

The results for changes in user input were even stronger than the avalanche test, with the key, rule list, and password tests being around 50 ± 0.13055189 with averages of 50.12020737, 50.12079142, and 50.11961302, respectively. The generations test varied slightly more, but was still very strong at around 50 ± 0.98394899 with an average of 50.34484595. All of the tests indicate that the algorithm is very sensitive to user input and that it can take a potential attacker trillions of tries, if not more, due to the fact that the exact values must be input.

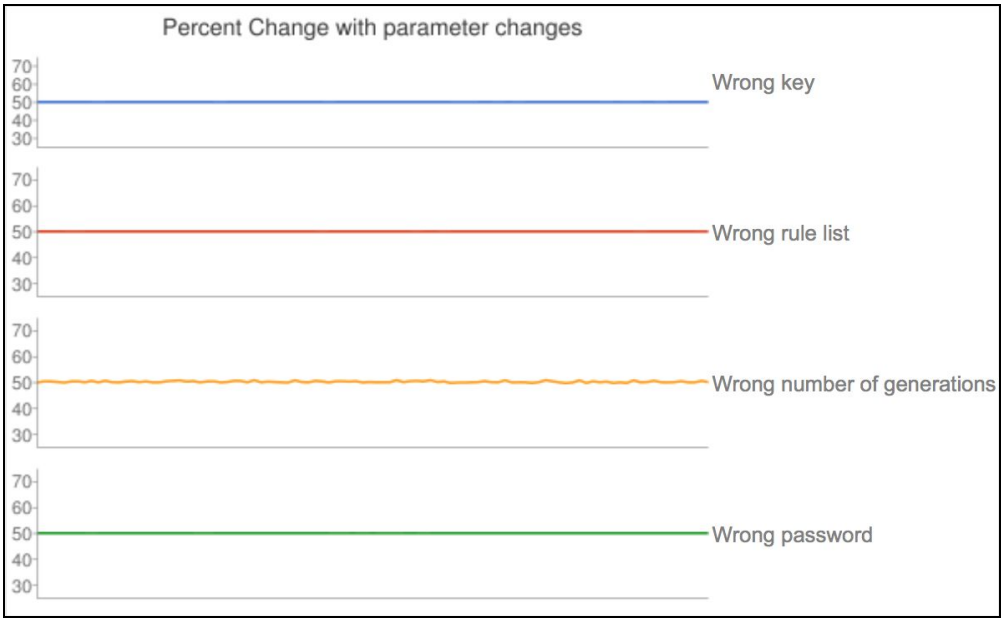

Figure 11: User-input Sensitivity Results. The calculation was done similarly to the avalanche test, but instead of flipping a bit, either the key, rule list, generations, or password was changed by the smallest permutation possible. Each test was performed 100 times, represented by the x axis with the y axis representing the % change of the plaintext.

Statistical Test Suites

ENT Test Suite

All of the results for the ENT tests were within acceptable limits. The monte carlo test only had a 0.148 % error, which is considerably small. The arithmetic mean was within .28 of the optimal value, entropy was within .01 of the optimal value, and the SCC was within 0.0015 of the optimal value. Additionally, the chi square test percentage is between the accepted values of 10% and 90% for a strong algorithm.

ENT Test (Averages)		Optimal Value
Monte Carlo Value for Pi	3.136950779	3.141592654
Monte Carlo Error %	0.1477554732	0
Arithmetic Mean	127.7702387	127.5
Chi Square Test %	23.28629032	10-90
Entropy	7.992023581	8
Serial Correlation Coefficient	-0.001474306452	0

Figure 12: ENT Test Results. The ENT test was performed using the UNIX command line utility.

DIEHARD Test Suite

The DIEHARD test calculates p values for each of the tests and a p value of 0 or 1 would indicate a failure. A p value lower than 0.005 or higher than 0.995 is considered a weak result, but still passes. All the results for the algorithm tested are well within those ranges, most of them being near 0.5. This indicates the algorithm passed all tests.

Dieharder Tests	p value	Dieharder Tests	p value
Diehard - Count the 1s	0.5566249079	Diehard - Oqso	0.5718344571
Diehard - Opso	0.5155458248	Diehard - cParking Lot	0.5008637016
Diehard - 2D Sphere	0.61204579	Diehard - Rank 6x8	0.5382818795
Diehard - Squeeze	0.5465959971	RGB - Lagged Sum (Avg.)	0.05683293172
Diehard - Runs	0.5516594535	RBG - Minimum Distance (Avg.)	0.4696530497
Diehard - DNA	0.4704665242	RGB - kstest	0.4623274777
Diehard - Craps	0.5656473653	RBG - Bitdist	0.5492723721
Diehard - Rank 32x32	0.5796215721	RGB - Permutations (Avg.)	0.5842067756
Diehard - Sums	0.20785863	Dab - Filtree (Avg.)	0.5225077191
Diehard - Bitstream	0.6105166098	Dab - Monobit (Avg.)	0.5757360779
Diehard - 3D Sphere	0.5742641984	Marsaglia - Tsang gcd	0.5912231198

Figure 13: DIEHARD Test Results. The DIEHARD test was performed using the UNIX dieharder command line utility. The average p value was calculated for each of the files for each of the dieharder tests with a passing value being between 0.005 and .995.

NIST Test Suite

Finally, the last test is the NIST Statistical Test Suite. NIST defines a sequence to be random if the proportion of passed tests to failed tests is over 0.8 when 10 bitstreams are used. The algorithm passed most of the tests, with the exceptions being the Rank, Approximate Entropy, Serial, Maurer's Universal Test, and Random Excursions. While not perfect, the fact that the algorithm passed so many tests is very promising. It isn't clear why the algorithm passed all its other tests but not these 5, so that is something to be studied further

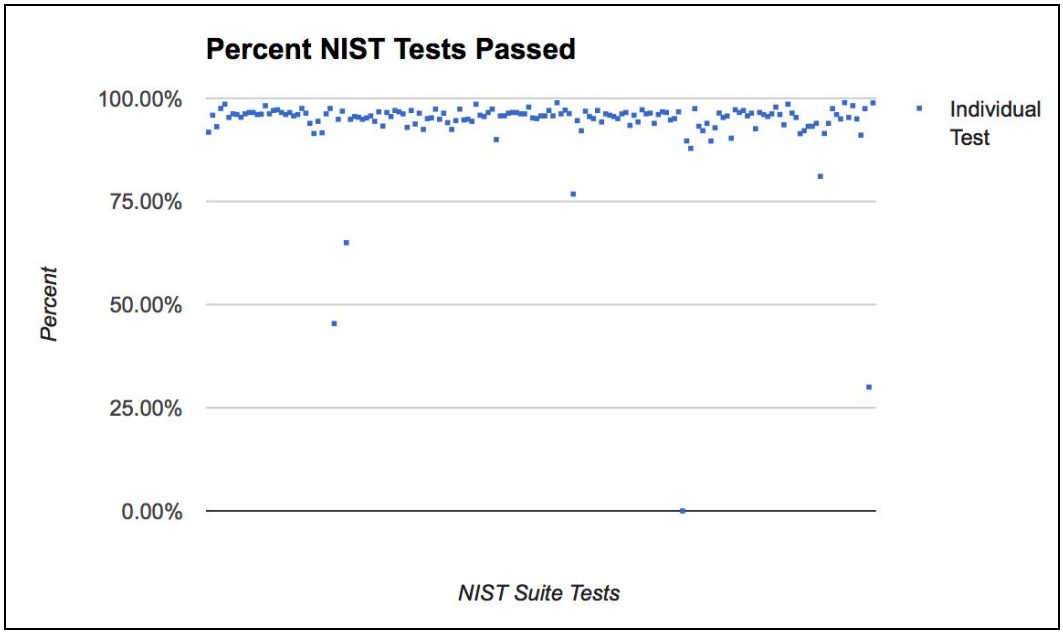

Figure 14: NIST STS Results. The NIST test was performed according to the instructions and guidelines in the original report. The test reports the number of tests that were passed, which was added for each of the NIST tests and reported as a percentage. Each dot represents an individual test in the NIST STS.

Implications

The purpose of the study was met for the most part. There have been studies in the past with cellular automata, as described in the introduction, that were proven to be secure against all tests measured, and some are also very fast and efficient. All of those, however, used life-like cellular automata with complicated reversibility methods (Seredynski, Pienkosz and Bouvry, 2000; Wang et. al, 2013; Faraoun "Design of a Fast One-pass Authenticated and Randomized

Encryption Schema", 2014; Faraoun "Fast Encryption of RGB Color Digital Images", 2014). The study by Bouchkaren et. al did use partitioning cellular automata, but that study wasn't as comprehensive and only used the rule list and generations as the secret key. This study also used the NIST test suite to provide a more accurate representation of the security of the algorithm. Like most other studies, the algorithm passed both the ENT and DIEHARD tests with flying colors, but failed a couple of the NIST tests. The algorithm design presented here isn't perfect, but it shows some promise as to the usefulness of partitioning cellular automata in encryption. One of the benefits of using a partitioning cellular automata is that the information can be made more secure simply by encrypting it with a higher number of generations. Assuming an attacker would start at 1 generation and move up from there to reduce processing time, information encrypted over lots of generations will take significantly more time to decrypt, making it theoretically infinitely secure if more time is spent on the encryption. This makes this algorithm ideal for encryption of information when time isn't a priority, such as when archiving information that may or may not be accessed later on, or for encryption of smaller files.

Limitations

One limitation to this project that is not immediately apparent is the speed of the algorithm. Due to the language used and the possible inefficiency of the code, the algorithm is slow, which prohibits the processing of files over 10 megabytes. The algorithm speed varies, averaging around 4 kilobits per second with 50 generations, meaning that for one generation, 80 bits are processed per second. This is something that can be further refined by increasing the efficiency of the code and/or writing it in a different language that is more appropriate for the project. Another limitation of this study is the small sample size. Due to a lack of resources, not very many tests could be performed, and the low sample size could have interfered with the results. Further research would definitely be necessary to confirm these results.

Conclusion

Overall, the algorithm performed quite well. It exhibited sufficiently high entropy, it met the avalanche criterion, it showed strong sensitivity to user input values, passed the ENT and DIEHARD tests, and passed most, although not all, of the NIST STS tests. The hypothesis was met overall, but the algorithm can still be improved and the results should be confirmed using a

larger sample size. With more time and resources, more trials can be done which would enrich the data. The use of partitioning cellular automata in encryption is quite promising and there is an infinite number of cryptography studies that can be done in the future using PCAs. With more time, the algorithm's shortfalls can be investigated and hopefully will pass all of the NIST tests and perform at faster speeds with some improvements. My algorithm design can be implemented with different processes that are faster and more efficient than partitioning cellular automata to secure sensitive information and protect our privacy.

References

"Banking -- Procedures for message encipherment (wholesale) -- Part 2": DEA algorithm. ISO 10126-2:1991. Geneva, Switzerland : ISO.

Bouchkaren, Said, and Saiida Lazaar. "A Fast Cryptosystem Using Reversible Cellular Automata." *International Journal of Advanced Computer Science and Applications IJACSA* 5.5 (2014): n. pag. Web.

Boutin, Charles. "NIST Kicks Off Effort to Defend Encrypted Data from Quantum Computer Threat." *NIST News*. NIST, 16 Aug. 2016. Web. 1 Sept. 2016.

Brown, Robert G. "Dieharder." *Robert G. Brown's General Tools Page*. Duke University Physics Department, n.d. Web. 11 Sept. 2016.

Faraoun, Kamel Mohamed. "Design of Fast One-pass Authenticated and Randomized Encryption Schema Using Reversible Cellular Automata." *Communications in Nonlinear Science and Numerical Simulation* 19.9 (2014): 3136-148. Web.

Faraoun, Kamel Mohamed. "Fast Encryption of RGB Color Digital Images Using a Tweakable Cellular Automaton Based Schema." *Optics & Laser Technology* 64 (2014): 145-55. Web.

Hortensius PD, McLeod RD, Card HC. "Parallel random number generation for VLSI systems using cellular automata". *IEEE Trans. Comput*. (1989);38:1466–73.

"How To SHA256SUM." *Ubuntu Documentation*. Web. 14 Sept. 2016.

Lafe, O. "Data Compression and Encryption Using Cellular Automata Transforms." *Proceedings IEEE International Joint Symposia on Intelligence and Systems* 10.6 (1998): 581-91. Web.

Luckett , William Matthew, "Cellular automata for dynamic S-boxes in cryptography." (2007). *Electronic Theses and Dissertations*. Paper 863.

M. Seredynski, K. Pienkosz, P. Bouvry. "Reversible cellular automata based encryption" *Netw Parallel Comput*, 3222 (2004), pp. 411–418

Machicao, Jeaneth, Anderson Marco G., and Odemir Bruno Martinez. "Chaotic Encryption Method Based on Life-like Cellular Automata." *Expert Systems with Applications* 39.16 (2012): 12626-2635. Web.

Marsaglia, George. "The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness." *DIEHARD*. National Science Foundation, 1995. Web. 11 Sept. 2016.

McCandless, David "World's Biggest Data Breaches & Hacks" *Information Is Beautiful*. Web. 14 Sept. 2016.

Rocha, Rodrigo Caetano. *Moore Neighborhood*. Digital image. *A Basic Example: Game of Life*. GitHub, 21 Aug. 2015. Web. 2 Sept. 2016.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S....Dray, J. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Publication no. 800-22. National Institute of Standards and Technology, 2010. Web. 11 Sept. 2016.

Sen S, Shaw C, Chowdhuri DR, Ganguly N, Pal Chaudhuri P. "Cellular automata based cryptosystem (CAC)." In: *ACRI-2002. LNCS*, 2513. Springer-Verlag; (2002). p. 303–14.

Seredynski F, Bouvry P, Zomaya AY. "Cellular automata computation and secret key cryptography." *Parallel Comput*. (2004) ;30:753–66.

Toffoli, Tommaso, and Norman Margolus. *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA: MIT, 1987. Print.

Tomassini M, Perrenoud M. "Stream ciphers with one- and two-dimensional cellular automata". In: Schoenauer M et al., editors. *Parallel Problem Solving from Nature* – PPSN VI. LNCS, 1917. Springer; (2000). p. 722–31.

Walker, John. "Pseudorandom Number Sequence Test Program." *ENT*. Fourmilab, 28 Jan. 2008. Web. 11 Sept. 2016.

"When Back Doors Backfire." *The Economist*. The Economist Newspaper, 02 Jan. 2016. Web. 18 Sept. 2016.

William F. Ehrsam, Carl H. W. Meyer, John L. Smith, Walter L. Tuchman, "Message verification and transmission error detection by block chaining", US Patent 4074066, 1976

Wojtowicz, Mirek. Margolus Transition Rules. Digital image. *Cellular Automata Rules Lexicon*. N.p., 5 Sept. 2001. Web. 1 Sept. 2016.

Wolfram S. "Random sequence generation by cellular automata." *Adv. Appl.* Math. (1986);7(2):123–69.

X.Y. Wang, D.P. Luan "A novel image encryption algorithm using chaos and reversible cellular automata" *Commun Nonlinear Sci Numer Simul*, 18 (2013), pp. 3075–3085

Executive Summary

The topic of encryption deals with making information secure by manipulating it in a way that only the intended recipient can make sense of it. Cryptography has recently gotten increased attention in popular media due to emerging technologies like quantum computing that threaten to undermine current encryption mechanisms. Almost every computer in the world encrypts sensitive information, and encryption is crucial for entities like the government and large corporations to keep their secrets safe from prying eyes. This study wanted to test whether if cellular automata, computer simulations of biological cells, could be used to design an encryption algorithm that is secure to various types of attacks. The algorithm is able to use a type of cellular automata that is easily reversible but also secure along with user-specified information like a password to encrypt a file well enough to resist infiltration by a potential malicious attacker. The user-defined information plus the cellular automata is used to scramble the information contained in a file. To turn the encrypted information back into readable information, the recipient needs both the "secret key" generated in the encryption step, and the exact cellular automaton parameters. The algorithm passed most of the tests and shows that reversible cellular automata are good candidates for use in encryption algorithms.