Implementation of a Cryptography Algorithm Based on a Cellular Automaton

Andrew Chronister and Nikko Rush

Introduction

This paper presents an implementation of a cryptographic technique based on simple 2-state reversible cellular automata, and thorough analysis of the benefits and tradeoffs of this technique. Historically, most cryptographic methods have been vulnerable to attack either by analyzing frequencies of certain characters or bits in the output, called the ciphertext, or by exploiting features of the underlying mathematics that can be sidestepped through various techniques. Most modern cryptography systems use multiple techniques in sequence to reduce the feasibility of using some of the more prosaic methods of analysis (Swenson, 4.2). This paper will be focusing on an individual method, and as such, will not be drawing direct comparisons to most modern techniques. The technique discussed herein is almost three decades old by now, but has not achieved widespread usage or even much in the way of discussion ("Cryptography with Cellular Automata", 429). This paper will offer a concrete, proof-of-concept implementation of the technique, using the Python programming language. It will also evaluate the efficacy of the technique by several standard methods of cryptanalysis.

Background Research

The cryptographic technique described here primarily rests on particular irregularities of a group of simple rule systems called cellular automata. The implementation described is primarily based off of the work of Stephen Wolfram, in his 1986 paper suggesting the utility of such rule systems in a cryptographic application ("Cryptography…", 430-431). The technique uses a (1-dimensional) row of cells which can be one of two states, on or off. It should be noted that cellular automata work in any number of dimensions, but that 1-dimensional implementations are least computationally intensive and easiest to explore for the kinds of behavior that need to be targeted. The states of the cells in each consecutive row are determined by the states in the previous row according to rules that involve the state of the cell in the prior row, the states of the neighboring cells in the prior row, and the state of the cell two iterations back ("A New Kind of Science", 437). This can be viewed as a kind of evolution over time, but is usually depicted as a series of rows on a two-dimensional grid with the y axis representing time. The dependence on a cell two iterations back is not strictly necessary, and is only included to allow the automaton to be inherently reversible, a property which implies it can be run forwards or backwards from any given pair of rows of cells.

The specific construction used is of the kind proposed by Wolfram when discussing reversibility in *A New Kind of Science* (437). Despite being formed from simple rules, these constructions have been shown to result in several different kinds of behavior, ranging from nested patterns to random noise to lattices with "localized structures…some propagating" ("Computation Theory of Cellular Automata", 17). The kind used in this implementation is of Wolfram's third or "Class III" variety that generates essentially random patternless noise. Wolfram demonstrates that these seem to "never become simpler with time", suggesting that continuous evolution would result in continuous noise of unchanging type ("Computation Theory…," 43). Several methods of cryptanalysis were applied to sets of generated output from the algorithm to determine if any of the methods that broke other modern symmetric ciphers could be used on our algorithm. Pure brute force was the first method considered, as it would not be very effective encryption if it could be easily brute forced. Ideally, this would require exponential time with increasing key size, as it does for most modern encryption techniques (Swenson, 5.1). Frequency testing was used to ensure that the cipher was not vulnerable to the most basic method of cryptanalysis, figuring out the frequency of characters in the output based on known frequencies of characters in the input language (Swenson, 1.5.1). Linear cryptanalysis, an attack that targets imperfect diffusion of information, was another method applied. This attack uses systems of linear equations to relate inputs and outputs together. This method was successfully used against FEAL and DES in the early 1990s (Swenson, 6.0). Integral cryptanalysis, which uses a group of plaintexts with a certain block permutated through all combinations, was also used. The block could be, for instance, the last 8 bits of the message text in all possible values. Each output is examined with the other outputs to determine patterns of bit diffusion in the encryption process. The final method examined is related key cryptanalysis, which uses a group of similar keys with known relationships, such as having a string of bits which remain constant, to examine for patterns in the output.

Methods

The algorithm developed here simulates a 1D cellular automaton over a predetermined number of steps to form a stream of bit input for encrypting a message string. The cellular automaton (CA) is a 2-state, finite register implementation using rules specifically constructed to be reversible.The replacement patterns of a simple CA rule are typically enumerated according to a standard scheme in which each digit of the binary representation of the rule number corresponds to the output of a given pattern. For reversible rules, an identical scheme is used, save for the state of the cell two iterations prior: if this cell is active, it inverts the normal output of the replacement rule. The particular rule used is 214R, as specified by the following pattern in **Figure 1** (see Appendix). This method is one of several reversible rules of this format identified by Wolfram to be Class III ("A New Kind of Science," 438-439). It is also the most monotonic of the Class III rules, with very few readily identifiable structures or patterns in its noise patterns.

The register is confined to the size of the key and is not cyclic, meaning that cells on the far left side of the register are only affected by cells on the far right side of the register if evolution is performed for a sufficient number of steps. This is a possible weakness of the algorithm, as these bits may undergo less significant change over the course of the CA's evolution than those in the center of the register. The Python implementation uses lists of integers (assumed to be either 0 or 1) to internally store each iteration of the CA evolution. At any given time, the latest two rows of the evolution are accessible; to reduce memory usage, only the rows that are necessary for further evolution are kept. Python was used as it is quick to write and easy to debug. Real-world applications would more likely use C or a similar low-level, fast language for implementation.

The encoding process treats the CA as a stream of bits for message obfuscation, performing consecutive bitwise XOR operations between CA bits and message bits. Messages are passed into the function as a list of characters, a format that is convenient for testing. This effectively equates to a simple stream of input bits of block size 8; however, any block size could be used. The characters are converted into their ASCII integral representations so that they can be operated upon. The encoding function also accepts a seed, and an integral number of iterations to perform. The seed, or the key, is used as the initial state for the CA. Because a reversible CA is used, it requires two seed states; this implementation simply uses the provided seed twice. The number of steps should be kept constant in a real world application, or integrated into whatever form the by which key is communicated.

The encoding process keeps track of three indices: A message write head, indicating which position on the list of message blocks should currently be manipulated. A CA message read head, indicating which cell of the current row of CA evolution should be read to manipulate the position of the message write head. If the value of this cell in the CA is 0, the message write head is kept in place. If the value of this cell in the CA is 1, the message write head is incremented by one. If the message write head reaches the end of the string, it is looped back around to the beginning (index 0). A CA bit-block read head, indicating the position in the CA from which the encoder should begin reading a block of bits. The CA then iterates through the specified number of generations. Each iteration, the new row of CA cells is read by the corresponding read heads. The CA message read head is examined first and the message write head is translated accordingly.

After this, an 8-bit binary string is constructed from the bit-block read head and the next 7 cells. A bitwise XOR operation is then performed between the current index of the message, and the constructed binary number. The XOR operation is used because of the simplicity of decoding: a subsequent XOR operation with one of the same operands will yield the other operand. The encoded message is output as a colon-delimited hexadecimal string. For further obfuscation, other cells of the CA could be referenced for manipulation of string size, to avoid the ciphertext being the exact same length as the message text.

The decoding process simply follows the encoding process, but in reverse. First, the CA iterates through the specified number of generations from the seed. If an implementation with a non-reversible CA were used, these iterations would be stored and then referenced in reverse to decode the message. However, in this implementation, these iterations are simply to align the CA to the final state reached during encoding. The CA is then run with the final two states used in reverse order as the two seed states. Because of the nature of the bitwise XOR operator, the decoding process works in exactly the same way. The only difference between encoding and decoding here is that the message write head is decremented instead of incremented if the CA message read head's value is 1. The output can then be converted back into an ASCII string or the original sequence of bits.

Data was collected by storing a large number of message/ciphertext pairs and performing various methods of analysis on these. Brute force was not tested as it was determined that it would take far too long to actually run. For a seed of length 256, there are 2^{256} possible combinations of bits in the cellular automaton seed. Any optimizations to this number would be specific to the algorithm and, hence, not a brute-force attack. This means that as a baseline for the rest of the analysis, if it would take fewer than 2^n steps to run (where n is the length of the seed), then that particular method of analysis could be considered a crack or a shortcut.

Frequency analysis was performed by storing the results of 15000 different message/ciphertext pairs and scanning for frequencies of characters in various positions. Messages were generated by selecting 75-125 random english keyboard characters (commonly used ASCII values) and concatenating them. As encrypted messages are more realistically represented as blocks of 8 bits, this was not strictly for character frequency as in a traditional cipher but was instead for ensuring that certain combinations of bits were not more likely to appear in the ciphertext. These message-ciphertext pairs were then analyzed for character frequency by creating a list of the 256 possible values and incrementing a value when it appeared in a ciphertext string. Character displacement was also analyzed by comparing each character value in the original string

Results

Results seem to indicate that the algorithm, as it was implemented here, has some fundamental flaws. Information seems to be at least partially preserved in many cases, and messages can theoretically be reconstructed in far less than 2^n time.

Frequency analysis shows that bits in certain ranges were indeed more likely to appear (**Figure 2**). The mean frequency for any given byte value is 2952.61, and the median is 2199. The most obvious deviation is in the range from 98 to 122, where the frequencies spike to around 8000 - meaning they appeared in more than half of the ciphertexts generated, as opposed to the 2200 occurrences of much of the rest of the byte range, which represents a mere 15% or so of the messages. There are also outlying spikes at 39, 44, and 46, with other values deviating from significantly from the median surrounding these points. These deviations suggest one of two things. First, that characters are being missed by the message write head. The fact that the outlying values are mostly in the range for ASCII alphanumeric characters supports this suggestion. The standard ASCII character 39 is the single quote (‘), 44 is the comma (,), 46 is the period (.), and 98-122 are the lowercase letters b through z. All of these were characters used in the randomly generated messages created for testing the algorithm. The second possibility is that character displacement is occurring in a nonrandom manner. **Figure 3** indicates that significantly many characters were displaced forwards by 50 to 100, and an alarming number (6966 out of 15243) were not displaced at all. The mean displacement was 68.74, and the median was 67.0. The presence of these more frequent displacement values indicates that

The significance of this is that information is preserved in a predictable way through encryption. For a bitstream, this would allow an attacker to, at minimum, determine what type of message is being sent. The data also suggest that information tends not to diffuse into other parts of the message, as displacement is nonrandom. With knowledge of the format of the message, this may allow an attacker to determine specific characters and message content, as a matter of simple (very fast on a computer) analysis. Because of this very basic lack of security, it was not deemed necessary to evaluate the security of the cipher through any more complex methods. These attacks would be pointless against the algorithm, as there is a much faster and less computationally intensive way to break any given message.

Conclusions

The relative ease by which this implementation of a cryptography algorithm based on a cellular automaton can be broken suggests that this would not be a viable method to use in the real world. It is far too simple to determine the type of information being sent, as well as, in many cases, particular pieces of information in the message because of the frequency of characters in the message and the nonrandom displacement of characters. These shortcomings may be due in part to specifics of the implementation, particularly the choice to encrypt messages composed of ASCII characters, the use of only 8 bits of information at a time from the CA, and the use of a single write head for message obfuscation. Further investigation is needed to see if a more complex implementation would successfully provide a hard (difficult computationally) cipher for real-world cryptographic use. Specifically, further research should explore the use of larger blocks that have a bitwise XOR applied to them during encryption, with the second operand sampled from every third or fourth bit of the CA to limit the amount of information that can be determined about a given row of evolution. Additionally, it should be determined whether limiting the CA to a finite register negatively impacts the amount of obfuscation that can be performed, and whether the reversibility of the CA rule affects the ability of an attacker to analyze the cipher.

Works Cited

Biham, Eli, Orr Dunkelman, and Nathan Keller. *New Results on Boomerang and Rectangle Attacks*. Publication. Belgium: IACR, 2002. *New Results on Boomerang and Rectangle Attacks*. IACR, 2002. Web. 22 Feb. 2014.

Joux, Antoine. *Algorithmic Cryptanalysis*. Boca Raton: CRC, 2009. Print.

Junod, Pascal, and Anne Canteaut. *Advanced Linear Cryptanalysis of Block and Stream Ciphers*. Amsterdam: IOS, 2011. Print.

Knudsen, Lars, and David Wagner. Integral Cryptanalysis. Publication. Belgium: IACR, 2002. Integral Cryptanalysis. IACR, 2002. Web. 22 Feb. 2014.

Swenson, Christopher. *Modern Cryptanalysis: Techniques for Advanced Code Breaking*. Indianapolis, IN: Wiley Pub., 2008. Print.

Wolfram, Stephen. "Computation Theory of Cellular Automata." *Communications in Mathematical Physics* 96 (1984): 15-57. Web. 12 Dec. 2013. <http://www.stephenwolfram.com/publications/academic/computation-theory-cellular-automata.pdf>.

Wolfram, Stephen. "Cryptography with Cellular Automata." *Lecture Notes in Computer Science* 218 (1986): 429-32. Web. 12 Dec. 2013. <http://www.stephenwolfram.com/publications/academic/cryptography-cellular-automata.pdf>.

Wolfram, Stephen. *A New Kind of Science*. 1st ed. Champaign, IL: Wolfram Media, 2002. 598-606. Print.

Appendix A

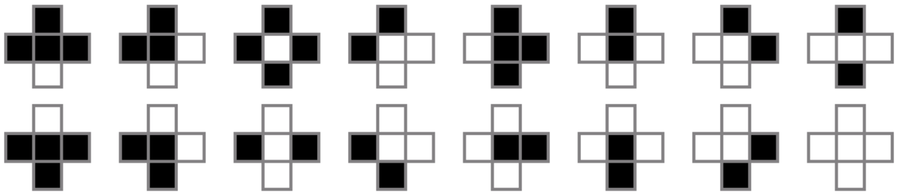
Figure 1

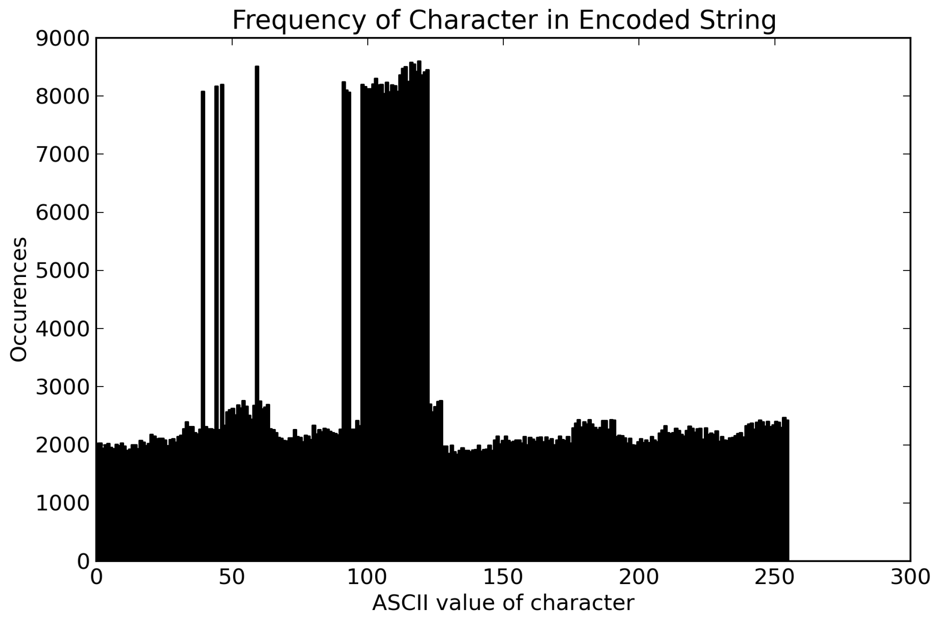

Figure 2

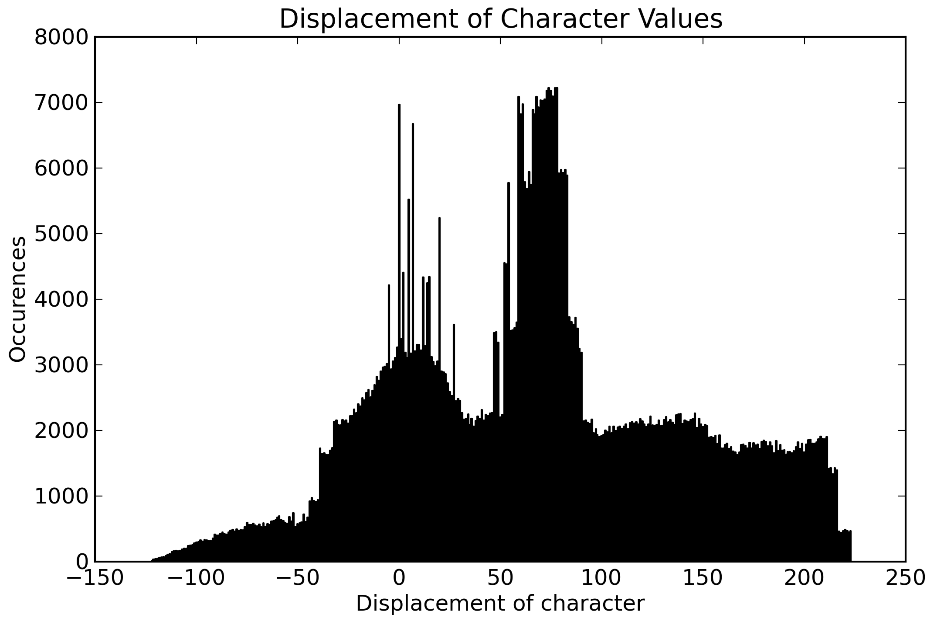

Figure 3