Chaotic Cryptography: Applications of Chaos Theory to Cryptography

Nathan Holt

Rochester Institute of Technology
nxh7119@rit.edu

May 18, 2017

Abstract

In this paper, we discuss the history of Chaotic Cryptography, and main definitions surrounding chaos theory. We then attempt to implement an encryption scheme based upon Baptista's original construction in 1998. We then discuss the possibility of using chaotic ciphers as a pseudo-random number generator with validation being done by the National Institute of Standards and Technology's Statistical Test Suite.

I. Introduction and History

Chaotic Cryptography is the blending of mathematical chaos theory and cryptography. Chaos theory involves the study of dynamical systems that are highly sensitive to initial conditions. It stands to reason that this sensitive could be beneficial in cryptographic algorithms, such as encryption and pseudo-random number generation.

In 1996, Baptista published his paper entitled "Cryptography with Chaos" in which he proposed a function encryption scheme based upon a chaotic function [1]. This paper was novel in the sense that it was one of the first instances of concrete, functional examples of Chaotic Cryptography, and it also performed some rudimentary statistical analysis.

Since then, various papers have been published on the subject of Chaotic Cryptography. Researchers have made attempts to use chaos-based functions on everything from encryption to hash functions to pseudo-random number generators. In the recent years, there has been in increase in the amount of papers published on the subject, and it seems to be gaining popularity.

This increase of popularity does not imply that these cryptosystems and other cryptographic primitive developed are secure. On the contrary, many

proposed cryptosystems have been proven insecure or impractical for actual implementation.

Recent popularity has been garnered by the attempts to strengthen chaotic functions for cryptographic applications, and several papers which have published nice results about implementations of the algorithms.

II. Dynamical Systems and Chaos Theory

In order to understand the methods being Chaotic Cryptography, it's important to understand the underlying mathematics. Dynamical systems are simply equations that vary with respect to time. Thus, dynamical systems arise when studying natural phenomena, as everything in the universe has the potential to change over time.

Chaos theory is a subset of the study of dynamical systems. Chaos theory is concerned with dynamical systems that fulfill three conditions. I will explain each of the conditions for a dynamical system to be considered chaotic.

i. Sensitivity to Initial Conditions

The first requirement for a dynamical system to be considered chaotic is that it must be sensitive to the initial condition of the system. Loosely speaking, this means that arbitrarily close start states will eventually become drastically different in their behavior. We call the values that an initial condition takes on when iterated in a chaotic function a trajectory. Thus, we can say that given two arbitrarily close start states, they will have drastically different trajectories.

In the mainstream media, this phenomenon is referred to as "The Butterfly Effect." This was originally discussed by a pioneer in the field of Chaos Theory, Edward Lorenz, who published a paper in 1972 entitled "Predictability: Does the Flap of a Butterfly's Wings in Brazil set off a Tornado in Texas?" [4].

In mathematically-rigorous terms, we have ways to characterize the sensitivity of a function based upon its start state. The most prominent method in the study of dynamical systems is through the usage of the Lyapunov Exponent. The Lyapunov Exponent gives a numerical quantity to represent the long-term divergence of trajectories given to arbitrarily close start states. In fact, given a separation of δZ_0, it can be shown that

$$|\delta Z(t)| \approx e^{\lambda t} |\delta Z_0|$$

where λ is the Lyapunov Exponent [5]. For the sake of this paper, we will be dealing with the logistic map, which has a well-defined Lyapunov Exponent, and we will assume - without proof - that this function is chaotic.

In the world of cryptography, this is a potentially desirable feature, as we can consider the "start state" of a chaotic function as the key. Based upon this, using a map that's sensitive to initial conditions will lead to drastically

different outputs when iterated based upon the key. In this sense, obtaining the correct output is highly dependent on the key, and even a single bit off will look just as wrong as having all the bits incorrect, due to the diverging trajectories of arbitrarily close keys.

For the purposes of explanation, we can plot the trajectories two close start states take on to show their chaotic behavior.

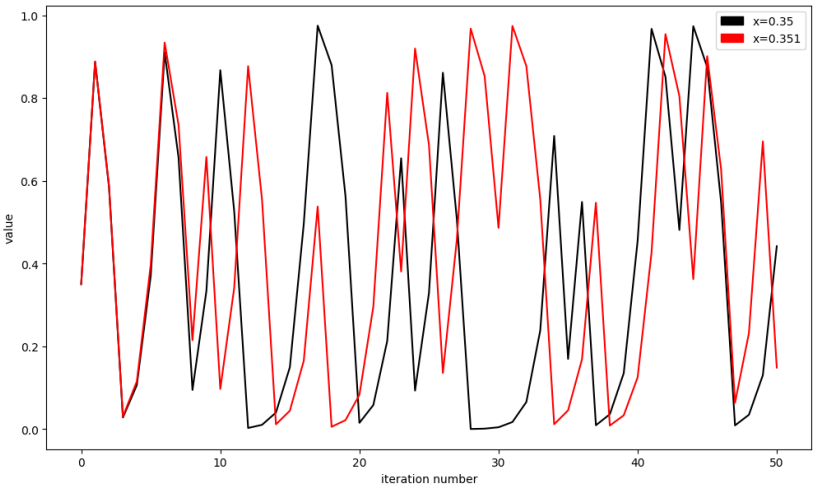

Figure 1: *A plot showing the divergence of trajectories from two close start states..*

Here the two initial values had a difference of 0.001, and before 10 iterations had occurred, they were completely different in trajectory.

ii. Topologically Mixing

The second criterion that a dynamical system must satisfy to be considered chaotic is that it must be topologically mixing. This means that any given open subset of the domain will eventully, intersect with any given open subset of the range of the function. In mathematically-rigorous notation, we can denote this as

$$B \cap f^n(A) \neq \emptyset$$

where A and B are open subsets of the domain and range respectively, and f^n is n iterations of the chaotic function f for some arbitrary n.

This is potentially desirable in the world of cryptography, as it means that given any key as a start state, the key will eventually be mapped to every possible output. This is near-required in cryptographic protocols, as gaps in the output of the function could lead to probabilistic attacks against the cipher.

iii. Density of Periodic Orbits

The final criterion to be met in order for a dynamical system to be considered chaotic is that it must have dense periodic orbits if it has periodic orbits. In mathematics, an orbit is simply a collection of points related by iterations of a given function. Thus, for the purposes of this paper, we use orbit and trajectory interchangeably. A periodic orbit is an orbit that oscillates between a finite set of values. Cycles of various lengths are possible in dynamical and chaotic systems. In fact, in the chaotic logistic map, for certain choices of parameters, it is possible to find a cycle of length k for any given k by utilizing the dyadic transformation.

In order for a dynamical system to have dense periodic orbits, any point in the range of the function must be approached arbitrarily close by periodic orbits.

This is potentially desirable in the world of cryptography, as having density of periodic orbits assures you that being in a cyclic orbit does not compromise your initial key, and also given any output of the function, your key is not compromised, as there are infinitely many other keys that can map to the same point via any cyclic cycle or through topological mixing.

iv. Logistic Map

The logistic map is one of the most basic examples of a chaotic function, and will be the basis for the rest of the paper. The equation for the logistic map is deceptively simple.

$$X_{n+1} = rX_n(1 - X_n)$$

This is an iterative map, meaning that you plug in the previous value of X_n in order to obtain the next value in the sequence, X_{n+1}. In the logistic map, r is a parameter of the system that is fixed before running the algorithm. The chaotic behavior of the map is entirely dependent upon the value of this parameter.

In the logistic map, values between 3.6 and 4.0 are considered chaotic, with the exception of a few islands of stability (points in this interval which do not produce chaotic behavior). For the purposes of this paper, we have selected to choose a parameter value of $r = 3.9$, as this produces chaotic behavior.

In a 2015 paper entitled "Modified Logistic Maps for Cryptographic Application", the following variant of the logistic map was proposed [2].

$$X_{n+1} = \begin{cases} g(x) = rX_n(1 - X_n), & X_n < 0.5 \\ h(x) = rX_n(X_n - 1) + \frac{r}{4}, & X_n \geq 0.5 \end{cases}$$

This modified logistic map increases the chaotic range for the parameter r to include values between 2 and 4, which is a five-fold expansion on the chaotic

range. This modified logistic map also increases the sensitivity to the initial conditions.

For the purposes of this paper, this is the logistic map function we will be using.

III. Encryption Scheme

In this section we will discuss the cryptosystem's algorithm. This cryptosystem currently takes a 64 bit key and runs as a stream cipher.

i. Start State Generation

Since we will be iterating the modified logistic map, we need a method to turn a 64-bit key into a number between 0 and 1. To do this, let us assume that they key k is represented in binary in the following way: $k = k_0k_1 \cdots k_{63}$. In order to convert this key into a start state, we will do the following:

$$X_0 = \sum_{i=0}^{63} \frac{k_i}{2^{i+1}}$$

This is guaranteed to generate a number between 0 and 1, as $\sum_{i=1}^{n} \frac{1}{2^i} < 1$ for any integer n.

It's worth noting that a start state of 0 will cause all future iterations in the modified logistic map to be 0 as well. Hence, this is not a valid start state for chaotic behavior, and the encryption scheme will not work. As such, a key containing 64 bits of 0's is not a valid key. Due to this, the key space of this encryption scheme is $2^{64} - 1$.

It is also important to note that X_0 is linked directly to your key. Thus, if an attacker knows your key, they know your start state, and vice versa. Thus, the start state must not be given out either, as there is a unique start state for all possible keys.

ii. Generation of the Lookup Table

The lookup table will be used to correlate given intervals in the range of the chaotic map with specific characters of plaintext that are supported in encryption. In our choice of the modified logistic map, we have that the range of the function is 0 to 1. Thus, we we call the range 1. Next, we must partition the range into k intervals, where k is the number of characters you wish be able to encrypt. For the purpose of this paper, we chose $k = 256$, as we wish to be able to encrypt all 256 extended ASCII characters.

In order to partition the range into 256 intervals, we create an ϵ length with is the range divided by the number of characters you wish to support encrypting.

$$\epsilon = \frac{1}{256}$$

Now, we partition the possible outputs of the modified logistic map into ϵ-intervals, and associate the given character with it's interval. For example, the character with an ASCII value of 0 will correspond with values of the modified logistic map between 0 and ϵ. The character with an ASCII value of 1 will correspond with values of the modified logistic map between ϵ and 2ϵ, and so on.

This can be precomputed to speed up encryption, assuming that the characters you wish to support encryption of stays constant.

One benefit of this encryption scheme is how easy it is to add new characters to support encryption of. To do this, you simply add this as a possible value, recalculate the lookup table, and the rest of the encryption algorithm works well. As such, this is a simple method for encrypting foreign languages, as the only change necessary is a quick recalculation of the lookup table.

iii. Encryption

Encryption is straightforward after the computation of the start state and the lookup table. To encrypt a message, we work as a stream cipher. We iterate the modified logistic map until the value of the modified logistic map lies in the ϵ-interval associated with the plaintext character you're trying to encrypt. For example, encryption of A (65 in ASCII) would stop iteration when $65\epsilon \leq X_n < 66\epsilon$. Upon finding a value in the associated ϵ-interval, we record the number of iterations it took to reach the value. This is recorded as our ciphertext for the character, and we move on to the next character, resetting the iteration counter, but not resetting the value of X_n. Keeping the value of X_n is good as it provides security based upon its sensitivity to initial conditions. Resetting back to the initial condition for every character would introduce a security flaw in our cryptosystem.

iv. Decryption

Decryption is straightforward. Given a ciphertext, we simply iterate the number of times specified in the ciphertext, and look at the value of X_n after that many iterations. We locate the corresponding ϵ-interval. The character associated with this interval is the plaintext character.

IV. Results of Cryptosystem

To test our encryption scheme, we decided to encrypt the opening to "Gravity's Rainbow" by Thomas Pynchon. This was a stream length of 35,557 characters to encrypt. For choice of 64 bit key, we chose to use 14484029183440969904234666382491833679978054820 11. This was a random 64-bit key we found when looking online for a random key generator.

First, we look at the distribution of the cipher text after encrypting.

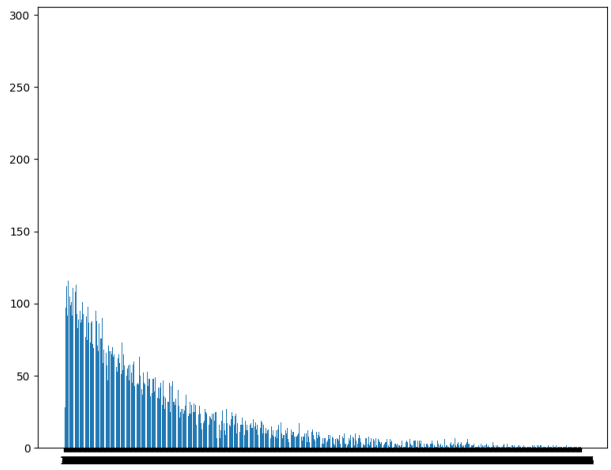

Figure 2: *A plot showing the frequencies of ciphertext occurring.*

In figure 1, we have the ciphertext values on the horizontal axis. The actual values are not possible to see as there are too many different possibilities a ciphertext character can take on to plot in a readable fashion. On the vertical axis, we have the number of occurrences.

We can see that lower values of ciphertext output are more likely than higher values. From a computational perspective, this is a good thing. Smaller values of ciphertext correspond with less iterations of the modified logistic map, which means the algorithm takes a shorter amount of time to run. From a probabilistic standpoint, the distribution is quite good. For encrypting over 35,500 characters, any given ciphertext output was repeated less than 300 times. This means that if an attacker were to choose a random ciphertext value to try and decrypt, there would be less than a $\frac{300}{35557}$ chance that it is the correct value.

V. Conversion to a PRNG

Much of the talk in the world of Chaotic Cryptography revolves around finding new ways to utilize chaotic maps as pseudo-random number generators. In this section, we explore the possibility of gathering pseudo-random numbers by looking at the ciphertext of the previous encryption.

i. Method

In order to gain random numbers, we loop through the ciphertext values. We look at each adjacent pair of cipher text values. If the first value is smaller than the second value, we consider this a 1. If the first value is bigger than the second value, we consider this a 0. It's important to note that a ciphertext length of $n + 1$ will correlate to n pseudo-random binary bits, as we're considering pairs of ciphertext values.

Due to the randomness inherent in chaotic maps, it stands to reason that this could produce a stream of pseudo-random bits.

To test this, we use the National Institute of Standards and Technology's Statistical Test Suite.

ii. Results

The PRNG behaves shockingly statistically randomly. When tested against all tests in the Statistical Test Suite, it passes nearly all of them. It does fail on two tests: the Runs Test and the Spectral Test.

The Runs Test measures for long repetitions of a single character. Long sequences of only 0's or only 1's could indicate that the sequence isn't random.

The Spectral Test measures periodicity in the output of the PRNG. Repetitive patterns could indicate that the sequence isn't random.

However, as a preliminary proof-of-concept, we are happy with the results of passing the majority of the Statistical Test Suite's tests.

VI. Discussion

This paper documents the successful implementation of the chaos-based cryptosystem based upon Baptista's original work in 1998. Furthermore, we show the potential that a simple encryption scheme has to be used as a powerful, quick way to obtain pseudo-random bits.

i. Future Work

There are several areas that need further research to be more thorough.

The first thing we wish to study further is the possibility of using other chaotic maps instead of the modified logistic map. The modified logistic map

has periodic orbits, which may be the cause of the pseudo-random number generator failing the Spectral Test (which test for periodicity in output). As such, using a chaotic function that doesn't have periodic orbits may allow us to pass the Spectral Test.

The second thing we wish to study further is extending the keysize, and thus the key space. We would like to implement a $2 - step$ logistic map, which takes a 128-bit key instead of a 1-step logistic map (as we used in this paper), which only takes a 64 bit key. The increase of key size also permits for more confusion and diffusion in the key through bit mixing, as is featured in other encryption algorithms, such as the Advanced Encryption Standard. The paper "2-Step Logistic Map Chaotic Cryptography Using Dynamic Look-up Table" presents a way to use a larger key along with subkeys, as is the case with the Advanced Encryption Standard which would provide further security from brute force attacks [3].

References

[1] M S Baptista. *Cryptography with Chaos*. Mar. 1998. URL: http://cmup.fc.up.pt/cmup/murilo.baptista/baptista_PLA1998.pdf.

[2] Shahram Etemadi Borujeni and Mohammad Saeed Ehsani. "Modified Logistic Maps for Cryptographic Application". In: *Applied Mathematics* 06.05 (May 2015), pp. 773–782. DOI: 10.4236/am.2015.65073.

[3] Yamini Goyal, Geet Kalani, and Shreya Sharma. *2-Step Logistic map Chaotic Cryptography using Dynamic Look-up Table*. Nov. 2015. URL: http://www.ijcaonline.org/research/volume130/number10/goyal-2015-ijca-907082.pdf.

[4] Edward Lorenz. *Predictability: Does the Flap of a Butterfly's Wings in Brazil set off a Tornado in Texas?* URL: http://eaps4.mit.edu/research/Lorenz/Butterfly_1972.pdf.

[5] Antonio Politi. *Lyapunov exponent*. URL: http://scholarpedia.org/article/Lyapunov_exponent.