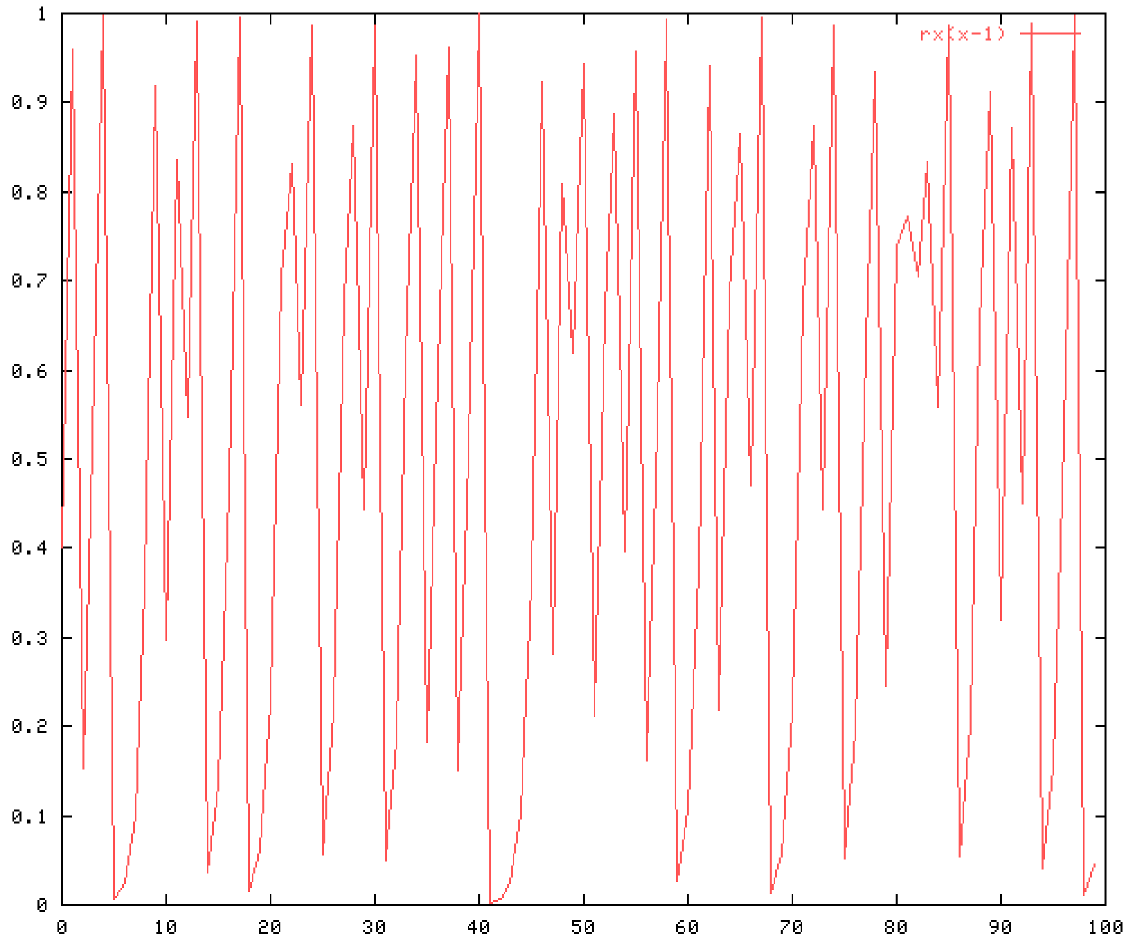
<u>Chaos Cryptography:</u>

Protecting Data Using Chaos



By James Lampton (jhl1@msms.doe.k12.ms.us)

Mississippi School for Mathematics and Science

Advisor: Dr. Wenstrom

Abstract: The purpose of this paper is to investigate and merge the two worlds of cryptography and chaos theory by creating a secure ciphering algorithm based on known Chaos equations. What follows is a description of a ciphering method based on a chaotic system that was developed to protect information from detection even if one was able to find some information on the system used in encryption.

1.0 Introduction:

In today's society, where information flows so freely all over the world, there is a need to regulate and protect sensitive data. The way we currently do this without stopping the information from being transmitted is by using cryptography. Cryptography has been important even before the development of computers and the Internet. Before computers, countries would have "black rooms" filled with mathematicians who opened and read incoming and out going mail. There were many methods of encryption, ranging from the Caesarian letter replacement to the notorious Enigma machine used by German U–Boats in World War II.

There is one encryption algorithm that has survived since the days of the Renaissance. This would be PAD encryption. This encryption method works by taking a message and splitting it down to its given letters and assigning them numbers. This list of numbers would then have an equally long list of numbers, the PAD list, added to it. Only the sender and receiver would have a copy of this PAD list. The only problem with this form of encryption is that you must have a list as long as your message, which in today's world would be a great loss because for every message transmitted, twice the amount of data must be sent. However, using chaos, one could generate a set of pseudo–random numbers, and hence create arbitrarily long PAD lists, with relatively little information originally held by the sender and receiver. This would be an improvement, because it has all the security of the PAD method minus the large and hard to transmit key.

One still must worry of detection of the initial value or even a value along the encryption process. That is why some measures must be added to further protect the key while encrypting and still allow for reproducible results when decrypting.

2.0 Basics of Chaos Theory:

For many years scientists and mathematicians have been astounded and perplexed by the complexity generated by real world systems. Due to relatively new math breakthroughs in such fields as differential equations and non–linear systems we find that many simple looking equations can generate chaotic behavior. Chaotic behavior can be defined as long term unpredictability. In some equations, starting with a generically defined initial value can render a specific and predictable outcome. Take for example, the function $f(x)=x^2$. Given a starting value of $x_0=2$ and having $x_n=f(x_{n-1})$, the values x_n race to infinity. In turn, if you start with an x, where $0<x_0<1$, the iterates limit to zero. This is a predictable system not exhibiting chaos.

One simple function used to generate chaos, which is also the equation I used in my ciphering system, is the well known logistic equation to simulate the population of a species given a certain growth rate. This function is defined as follows: [1]

$$(1) \quad x_{n+1} = r(x_n)(1 - x_n)$$

Where x is the population of the species at a given time interval. The domain of x ranges from 0 (extinction) to 1 (largest population possible). The constant r represents some growth constant of a given species. This constant also happens to represent the non−linear part of this linear function. Depending on the value of r, this equation has areas where its results are predictable. As r increases so does the non−linearity of the system. Figure 1 will help in the explanation of the dynamics of this function. This image illustrates results of this function given different values of r. The x−axis represents the r value while the y−axis represents the resting state of the system. As you can see, when $0 < r < 1$, any initial population value will eventually become extinct. When r lies between 1 and ~3 the system comes to a single resting place, this is an r value that results in the system having a period one cycle. At approximately r=3 you can see the resting point split into two. This means that the value of x periodically switches from a high to a low state. This value has a period of 2. Then the values break apart again, producing period a 4. After this the well defined resting points begin to disappear. Around the r value of 3.75 you can see an area where the system has a period of 3, this is where the system races away to chaos ("Period 3 implies Chaos" − James Yorke [2]). The values don't begin to fluctuate to their full capacity until r equals 4, where the resulting values will fully range between 0 and 1.

The overall strength of using chaos as a backbone to a ciphering algorithm is that it has sensitive dependence on initial conditions. Therefore, small deviations become larger as the iteration of the system continues. (Ex: the iterates of an initial value of 0.4 will eventually deviate greatly from the iterates of 0.4001.)

2.1 Explanation of Components of a Chaos Ciphering System:

In my system of ciphering I use basic aspects of chaos theory to encrypt the data. I also have added a few new techniques in order to protect the key from discovery and thus to protect the data. These are additions that I added to further ensure the mathematic improbability of discovering the key. These methods include chaos "priming" and a truncating function. In a nutshell, the priming function is used to hide the attractor (which is a part of a chaotic system that seems to pull the points to it), and the truncating function is used to remove leading decimal points to make a usable integer from the given state of the system.

2.2 Truncating Function:

Because the ASCII character set can only hold integers with values between 0 to 255, values that range between 0 and 1 wouldn't work well for encryption due to the small impact they would have on a PAD ciphering standpoint. So before we can use the current state of the chaos system to encrypt data we must first run it through the truncating function. This is just a simple function that takes two parameters, a large decimal value and an integer representing the number of digits the function is to return. The equation simply multiplies the decimal by ten until it reaches the desired number of digits then truncates the remaining decimal places using the greatest integer function. The mathematical format of this equation can be seen as follows:

$$(2)\quad T(n, size) = \| n * 10^{count} \|, n \neq 0$$

Where *count* begins at 1 and is incremented until $n * 10^{count} > 10^{size-1}$. The result is then passed through the greatest integer function, denoted by the double lines.

The overall use of the truncating function is to make the numbers useable by the ciphering system as well as to chop off significant digits that would make it impossible to discover the current exact condition of a system. This would cause any attempt of key recovery to fail.

2.3 System "Priming":

The part of this system seen in equation 1 that is essential to hide is its current state. The current state of the system, namely x_n, can be used to decode every number from the n–th byte to the end of the stream. This means that if a person was able to obtain a value close to the current state, he may be able to generate enough PAD numbers up to a point and decrypt more information. This value that is relatively close to the current value would work for a few iterations until the state of the actual current value and the phantom current values become unrelated. For ciphering, where you want to avoid as much data leakage as possible, I found it necessary to add another value to the current state. This value is called the initial primer (*pr*). The idea is to take the current value and iterate by the number of iterations represented by the initial primer. This may be any value, but a small number will suffice. Once the system is primed, the value at *pr* iterations is truncated by $pr_{next} = T(x_n, 2)$ generating the number to be used as the next primer. The PAD number is generated before priming by iterating the system one time past the current primer to *pr*+1 where it is fed through the truncating function and the next PAD number is generated.

This adds strength to the system because if a person were able to find a possible value for the current state of the system, they would have to then use the quadratic function to reverse the system by one iteration rendering two false values to be used in creating the next primer. Even if the phantom current value is remotely close it would be broken by the time the system is primed again. This also has a role in hiding the attractor of the system. This will be demonstrated in section 3.2.

2.4 Ciphering Method:

Using chaos one could just transmit the initial value of the system and the initial primer to the receiving end over a safe channel and that would allow the receiver to generate the same PAD list used to encrypt the message. In my application you can add numbers greater that 255 to the plain text which is then modulated by 255. In the decryption process, you just have to add 255 to the cipher value until you find that a valid character will be generated and then subtract. This means that a 3rd party would have no idea if the numbers generated by his phantom key are the correct numbers to decrypt with.

3.0 Application:

An application of this system would be in a simple encryption of the world renowned string "Hello World" which can be seen in the following example. I wrote this application in C++, where I was able to encapsulate all the key manipulation functions into a Key object, and all the ciphering functions into a Crypt object.

I used many different types of graphs to view the distribution of the PAD numbers generated from the chaotic system which can be seen in the section 3.2. I went through many different phases of the ciphering system. For instance, at first I only used PAD numbers modulated by 255. Next I moved to the current system which can use any size, and hence increases the strength of the ciphering system because it allows for a greater possibility of numbers.

3.1 Simple Encryption/Decryption:

I first setup the key and cipher systems. I set the key information manually to a simple value to make my explanation easier. A real key would be set using a random number generator and would have more significant figures. The initial value in this situation was ".4" with an initial primer of "6".

I first made a key object named *testkey* and set its initial values. I then made a Crypt object called *ctest* and set it up to use *testkey* as its key. I then setup a buffer with the string "Hello World" contained within it. Here are the ascii values of that buffer:

{ 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100 }

Each number representing a specific character in the string. (72 for 'H', 101 for 'e', etc.) I then encrypted the buffer. This is done by generating 3 digit numbers from the chaos system and adding them to the plain−text values and modulating them by 255. This results in the following:

{ 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100 }

+{ 876, 840, 473, 144, 498, 999, 243, 325, 440, 683, 269 }

−−−−−−−−−−−−−−−−−− mod 255 −−−−−−−−−−−−−−−−−−

{ 183, 176, 71, 252, 99, 11, 75, 181, 44, 26, 114 }

It can be seen that they are indeed different (as they should be). Also be aware that it doesn’t have the same cipher values for all of any given characters (thus, ’l’ doesn’t always have the value of 71, which can be seen with the 252 following it.) The receiving end of this transmission would have exactly the same key and would be able to regenerate the PAD numbers and subtract them from the values (after they have been un−modulated by adding 255 until they are larger than the current pad number) rendering the original plain−text message:

$$
\begin{array}{r}
\{ 948, 941, 581, 252, 609, 1031, 330, 436, 554, 791, 369 \} \\
-\{ 876, 840, 473, 144, 498, 999, 243, 325, 440, 683, 269 \} \\
\hline
\{ 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100 \}
\end{array}
$$

This has the same values as the original buffer (again as to be expected), "Hello World"!

3.2 Details of Security:

To further check out the results and make sure I didn’t have a weak ciphering algorithm I examined many different graphs of the PAD number output to make sure it didn’t have an attractor.

I did multiple tests on my PAD number generator to see if it showed the characteristic parabola that is seen when you graph the results using the last value on the x−axis and using the current value on the y−axis. I also made a 3D graph of the output to see if their was a 3 dimensional attractor (using (x_{n-2}, x_{n-1}, x_n) as coordinates).

Figures 2−4 show the distribution of PAD numbers. As you can see there is a heavy density of points around the corners of the graph. This shows that there is a better chance that you will get a high pad right after another high PAD number. Figure 2 is a distribution with 100000 samples, and Figure 3 is a distribution with 500000 samples. On Figure 3 one sees that there is an even distribution of samples. This can be seen further in the 3D graph (Fig. 4).

Graphs:

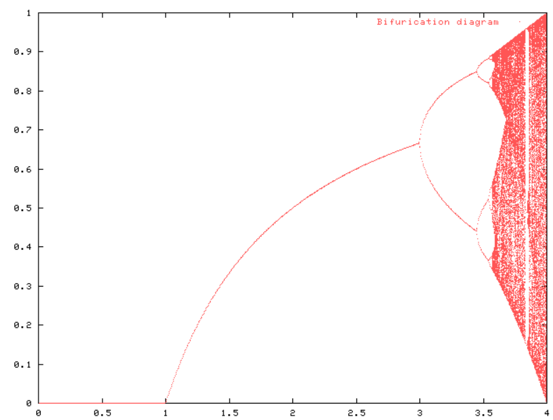


Figure 1

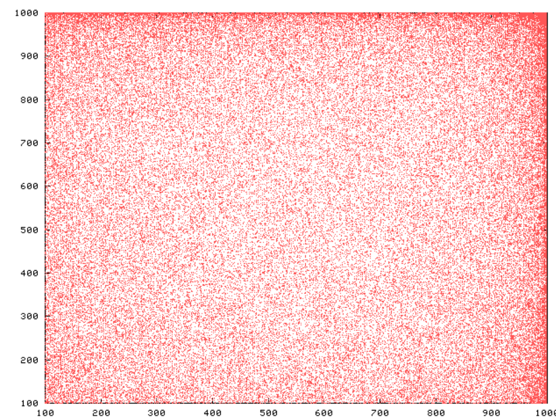


Figure 2

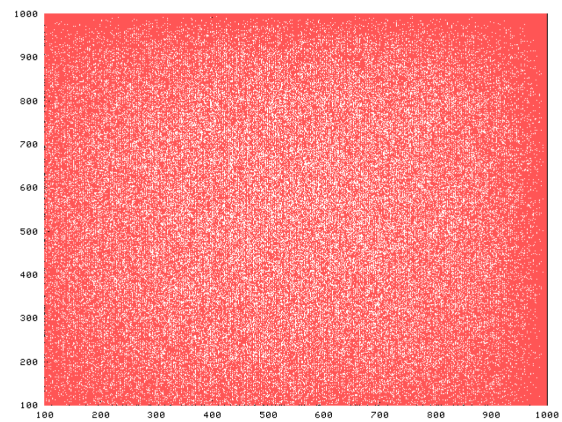


Figure 3

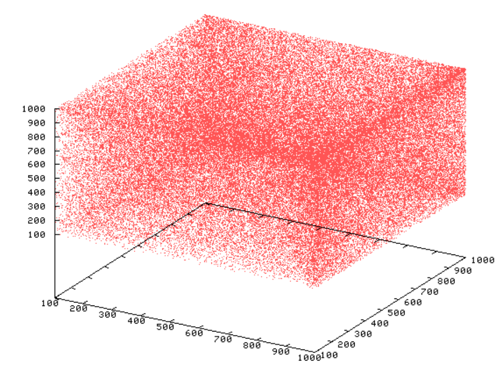


Figure 4

4.0 Acknowledgments:

I would like to thank Dr. Wenstrom for proof reading this paper and helping me understand some of the concepts of Chaos theory; Mr. Knight, my CS teacher for answering some of my C++ questions; Mr. Bradley for small optimizations of my code; and my classmates for putting up with my ravings about my newly found algorithm.

5.0 Works Cited:

[1] Barnsley, Michael F., Robert L. Devany, Benoit B Mandelbrot, Heinz–Otto Peiten Dietmar Saupe Richard F. Voss. The Science of Fractal Images. New York: Springer–Verlag, 1988

[2] Gleick, James. Chaos: Making a New Science. New York: Penguin, 1987