Chaotic Cryptography: Applications of Chaos Theory to Cryptography

Nathan Holt

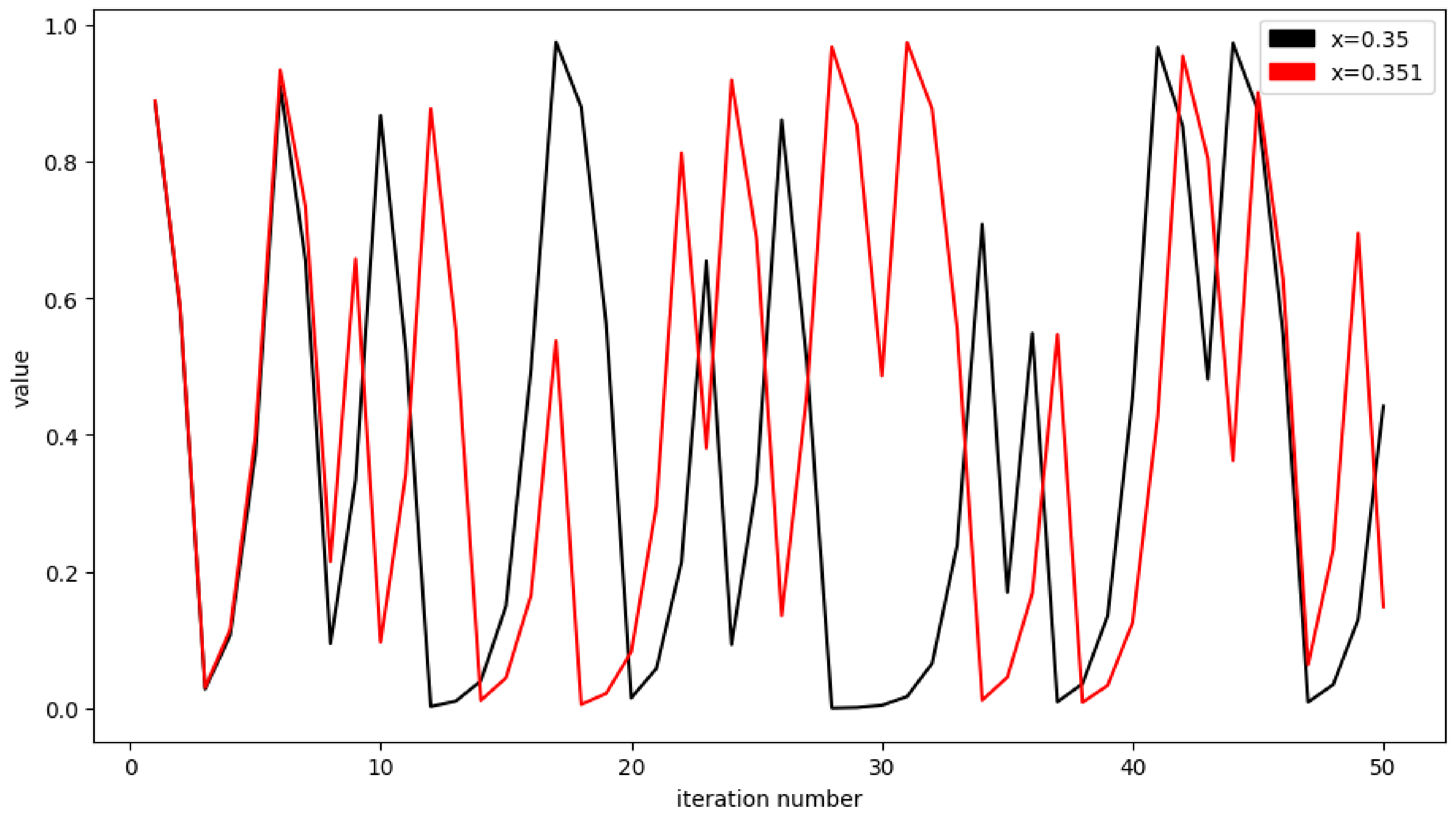
What is Chaos Theory?

- Loosely speaking, the study of dynamical systems that are sensitive to initial conditions.

- In mathematically rigorous terms:
  - Sensitive to initial conditions
  - Topologically mixing
  - Dense periodic orbits

- In the words of Lorenz, “Chaos: When the present determines the future, but the approximate present does not approximately determine the future.”

Visual Chaotic Dynamics



What Are the Proposed Cryptographic Processes?

Based on the Logistic Map:

$$X_{n+1} = r \cdot X_n \cdot (1 - X_n)$$

- Most cryptosystems are based on the Logistic Map, or variants thereof.
- In “Modified Maps for Cryptographic Application”, a modified Logistic Map is proposed which increases sensitivity to the initial condition and the chaotic range of the map, meaning an increase in the keyspace.
- In “2-Step Logistic Map Chaotic Cryptographic Using Dynamic Look-up Table,” an encryption scheme is proposed that uses subkeys and a dynamic lookup table.

Modified Logistic Map

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

- This increases the chaotic range of the parameter r to [2,4], whereas in the original Logistic Map, the chaotic range is [3.56995,4] (excluding islands of stability).

- This alteration increases the chaotic range of the logistic map fivefold times, increasing the potential keyspace for cryptographic protocols that implement this.

Cryptosystem Overview

- Currently using a 64-bit key with only one-step logistic map.
- Using a modified logistic map, as presented by “Modified Maps for Cryptographic Application”
- Uses a lookup table to figure out how many iterations to run logistic map before encrypting.

Cryptosystem
Step One: Start State Generation

- I'm currently only using a 64-bit key. In order to get a stable orbit in the logistic map, I need an initial value between 0 and 1.
- Assume key = $k_0 k_1 \cdots k_{63}$

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

- This gives a start state between the values of 0 and 1.
- Note: requires 22 decimals points of precision, which is available in Python through the Decimal library (otherwise, Python defaults to 16 digits)

Cryptosystem
Step Two: Creation of Lookup Table

- Divide all possible values of x in the Chaotic Logistic Map (0-1) by however many characters you want to be able to encrypt. I chose to encrypt all ASCII characters, so I divide by 256.

$$\varepsilon = 1/256$$

- Now associate each ASCII character with it's associated interval. For example, A is 65 in ASCII, so we associate values of x in the following interval as A:

$$65\varepsilon \leq x < 66\varepsilon$$

Cryptosystem
Step Three: Encryption

- For each character, iterate the modified logistic map until the value of x is in the associated ε-interval for the ASCII character it's encrypting.
- For example, encrypting 'A' means that we iterate the logistic map until $65\varepsilon \leq x < 66\varepsilon$.
- We record the number of iterations of the logistic map as the ciphertext for that character.
- Moving on, we maintain the value of x as the new initial start state for encrypting the next character.

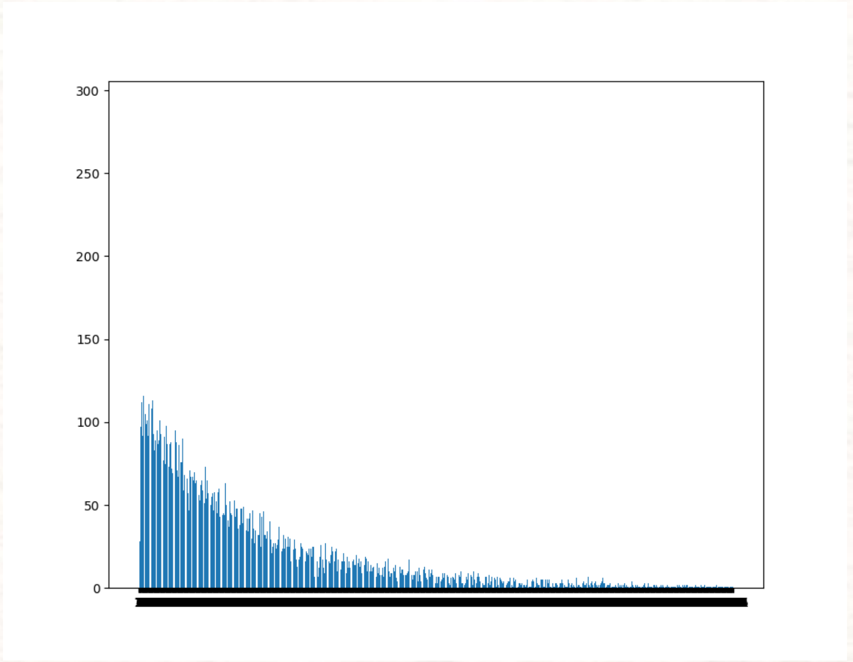
Cryptosystem
Step Four: Decryption

- Ciphertext is a list of integers, so using the same key, we simply iterate the logistic map the required number of times and look at the associated interval.

- For example, if the first encrypted character had a ciphertext output of 43, we'd iterate the logistic map 43 times, and then look at the value of x. We'd find which corresponding ε-interval this corresponded to in our lookup table, and that's our plaintext character.

Cryptosystem Results

- I encrypted the introduction to my favorite book with the key 18281414996634873555. The text to encrypt is 35,557 characters long.
- Encryption took 8307.846 ms.
- Here is a plot of the cipher text value frequencies.
- It has a minimum value of 1, and a maximum value of 5201.



Cryptosystem
As a Pseudorandom Number Generator

- I attempted to turn this into a pseudorandom number generator.

- My method: take the ciphertext from encryption and look at the difference between the subsequent values. If the second one is greater than the first, put a 1. If not, put a 0.

- Since this looks at the difference between encrypted characters, to get n bits of pseudorandom goodness, you require $n + 1$ characters to encrypt.

Cryptosystem Testing as PRNG

- I use the NIST Statistical Test Suite to test output of the function as a PRNG.
- I test against all 15 types of tests, and all 188 variants.
- I encrypted the book introduction as before, and looked at the pseudorandom number stream it outputted to analyze.

Cryptosystem PRNG Results

- It works surprisingly well.

- It passes all but two statistical tests, the runs test, and the Fourier transform test. (186 out of 188 is pretty good, though, right?)

- Frequency results for 10000 bit blocks:
  - 4975 0's, 5025 1's
  - 5020 0's, 4980 1's
  - 4930 0's, 5070 1's

Encrypting book introduction

- Frequency results for 10000 bit blocks:
  - 5039 0's, 4961 1's
  - 5000 0's, 5000 1's
  - 4981 0's, 5019 1's

Encrypting random text (took 8074.857 ms)

Runs Test

- Tests for long runs of the same character. (0000100000000000 would fail the runs test, for example)

- Is this characteristic of the test?

Fourier Transform Test

- Analyzes the periodicity of the sequence.

- Periodicity is a rigidly defined thing in chaotic dynamics. One of the three properties of a chaotic map is that if it has periodic orbits, they must be dense. Thus, we can conclude an attacker would not be able to gain anything, even if some periodic behavior was detected.

- Chaotic maps need not have periodic orbits at all. To pass this test, another chaotic map could be used instead of the logistic map to prevent periodicity.

Conclusions

- Chaotic cryptography is a new field of academic interest, and is only starting to be rigorously studied.
- Our proposed cryptosystem works well, although encryption is slower than desired for larger text.
- My proposed PRNG seems to pass NIST’s Statistical Test Suite well, with the exception of the runs test and the spectral test. It’s worth noting that even SHA-1, and BBS can fail specific tests for smaller sequences. (So failing two is actually a very good sign.)
- Future work should investigate expanding the key length, and using other chaotic maps (Lorenz System?) to remove any dense periodic orbits to pass the spectral test.

References

- Gutowitz, H. (1993). Cryptography with Dynamical Systems. *Cellular Automata and Cooperative Systems*, 237-274. doi:10.1007/978-94-011-1691-6_21
- Baptista, M.S. “Cryptography with Chaos”, *Physics Letters A*, vol. 240, pp. 50 –54, 1998.
- Arroyo, David, Gonzalo Alvarez, and Veronica Fernandez. "On the Inadequacy of the Logistic Map for Cryptographic Applications." *Internet Archive*. Universidad De Salamanca, 28 May 2008. Web.
- Borujeni, Shahram Etemadi, and Mohammad Saeed Ehsani. "Modified Logistic Maps for Cryptographic Application." *Applied Mathematics* 06.05 (2015): 773-82. Web.
- Goyal, Yamini, Geet Kalani, and Shreya Sharma. "2-Step Logistic Map Chaotic Cryptography Using Dynamic Look-up Table." *International Journal of Computer Applications*, 10 Nov. 2015. Web.