A General Framework for a Deniable Private Key Chaotic Cryptosystem

Chris L. Bresten

May 14, 2009

Abstract

We propose a generalized framework for the construction of deniable private key chaotic cryptosystems. This idea relies on the ergodic and diffusive properties of 1-D chaotic maps, allowing us to construct nice trapdoor functions. We provide a proof of concept based on the well studied Logistic map. The resulting cryptosystem allows us to encrypt one, two, or more messages into the same ciphertext with separate keys. Each of these messages can be retrieved with their respective keys, whilst yielding no information about the presence of alternative messages. This is the heart of deniability in its true sense, the only defense against the feared rubber hose attack.

1 Introduction

With the growing popularity of rubber-hose cryptanalysis techniques, it has been called upon the community to respond with cryptosystems resistant to these kinds of cryptanalysis. The holy grail of such solutions, though poorly realized, is the deniable private key system. The ideal deniable private key system would allow someone to encrypt multiple plaintexts into a single ciphertext, with separate keys. In the event of a rubber-hose attack, the holder of the keys can fork over an alternative key, yielding a plaintext that has been engineered to divert or confuse the attacker from the real secrets.

1.1 Rubber-hose Cryptanalysis

Rubber-hose cryptanalysis refers to the use of physical or psychological violence as a coercive force to extract a private key from the holder. As with many non-classical attacks on information security, this one relies on what is often the weakest link: users. Deniability tries to empower the user with the ability to provide bogus, but plausible alternatives to the real secrets the attackers are after.

1.2 Deniability - The ideal

The ideal deniable cryptosystem will allow one to seamlessly encode n plaintexts into a single ciphertext with n separate, distinct keys. These plaintexts must be retrievable with their respective keys, yielding absolutely no information about the existence of other plaintexts. This ideal deniable cryptosystem must also not guarantee that any such alternative messages exist. Given full knowledge of the system, it must be impossible for an attacker to determine if he has been given the only key, or one of many keys. Such a system should also be held to the same standards all cryptosystems are held, or at least have the ability to be streamlined with classical systems to ensure security. There is a difference between overall security and security of the deniability. If the deniability is secure then we know that one can not retrieve the alternative messages easier once one has the key to one message, and that one is never given any clue as to how many messages may be encoded into one ciphertext.

1.3 Chaotic Cryptosystems

In relatively recent years, many chaotic cryptosystems have been proposed. Many of them insecure, and some truly spurious. Many were not developed by cryptographers, but by physicists and signal processing people. Many make use of a chaotic system as a synchronized entropy source. This, while intriguing, is a naive idea, for many chaotic systems are not nearly as good at psudorandom number generation as known seeded algorithms, there is often no reason to choose a chaotic sequence over a standard seeded psudorandom number generator in this case.

Other systems make use of chaotic sequences as one way functions, somehow incorporating the plaintext into initial conditions, encoding the message by passing it through some chaotic system or another. [1]

Our system uses both of these ideas, but is at core mostly of the one-way function variety. It may be useful to look at it as an improvement upon the Baptista ergodic cipher [1].

There are even key exchange algorithms based on chaotic dynamics, hash algorithms, all sorts of things. Overall as a field chaotic cryptography seems very young. There are many issues with using chaos to fill the needs of cryptosystems, such as: computational expense, floating point operations and ciphertexts(expensive and large), inability to hold up to known plaintext and known ciphertext attacks, many have been created with a general lack of foresight about typical cryptanalysis(like one time pad attacks [3]).

We aim to use the full potential of chaotic systems to do what seemingly can not be done with classical integer based systems: Deniability.

2 A generalized framework for a Deniable Chaotic Cryptosystem

The system proposed makes use of chaotic sequences as one way functions. Though it is worth noting that synchronized entropy sources are probably necessary in any secure implementation. To construct our system we require a 1-D chaotic sequence with the following properties:

- Invertible in the sense that we can create a tree of possible trajectories to a given state with relative ease
- Includes a parameter that continuously affects it's behavior, but does not significantly affect the distribution of the attractor on an interval of useful size, so that the parameter can not be intelligently guessed by statistical analysis.
- ergodicity
- Maps itself on interval of the real or complex numbers.

2.1 Encryption

In the simplest case, pre-encoding is done by mapping our plaintext to a random value of the system's attractor, we will call this x_n. The system is reversed, yielding a tree with k^n leaves for n iterations on a system with k possible x_{n-1} for a given x_n. A random x_0 is chosen from the tree, making our x_n the plaintext and x_0 the ciphertext. The private key consists of n and the parameters necessary to reconstruct the system.

For a a deniable system we take the x_0 found above and make sure that with a separate set of parameters it will step forward to values that encode a separate message. We have to solve for a congruency like, so that this common x_0 will bring us to specified intervals of the domain for various secret keys, the interval where the system lands will decode to the original data. This will be expanded upon later.

Some important things must be taken into consideration if one wants to even begin talking about security of such a system:

- The plaintext must be mapped randomly to the attractor of the system, making the plaintext(if one discovers the parameters to the sequence) indistinguishable from it's surroundings. This is camouflage/steganography.
- Choices on the tree must be random.
- n must not be constant, it must change randomly from bit to bit. Exchanged not as an integer, but as the seeds to a psudorandom number generator.

- Increasing the number of plaintexts should not compromise the randomness of selection for ciphertexts. It should not significantly decrease the size of the ciphertext space.
- a good source of entropy is always necessary
- High digit precision arithmetic is a must
- the key, of course, must be exchanged securely
- The source of chaos must have parameters, preferably a few, to which it's behavior is very sensitive to, but does not affect the attractors statistical properties. This can prevent intelligent guessing of the parameter by statistical analysis of the ciphertext.

2.2 Decryption

Decryption consists of taking the ciphertext and using it as initial conditions for the system declared in the private key, choosing the values it takes at the discrete time steps specified in the private key. These values are then decoded back into binary with a common padding scheme.

3 Our proof of concept

Now to show you how this can be put together into something that works. Here we present a proof of concept architecture that follows the previously described framework, creating a truly deniable cryptosystem. We choose the logistic map as our chaos source, this choice is based largely on ease, it is the simplest case. The logistic map is well studied and understood by the community, so it is the easiest to work with. It is by no means ideal, but it fits the bill. This implementation is functional but sloppy. It employs several techniques for confounding cryptanalysis, but it is inherently weakened by it's sub-prime choice of a chaos source.

3.1 The Logistic Map

The discrete logistic map takes on the form of the quadratic:

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

r is the parameter, in our case it's interval will be $[3.9, 4]$. This region lacks bands of stability, so it is thoroughly ergodic.

3.2 Inverse logistic map

Say if we want to construct a logistic map sequence that will land on c after n iterations. To do this we have to invert the logistic map. This can be done by solving for the roots of:

$$0 = x_{n+1} - x_n * (1 - x_n) * r$$

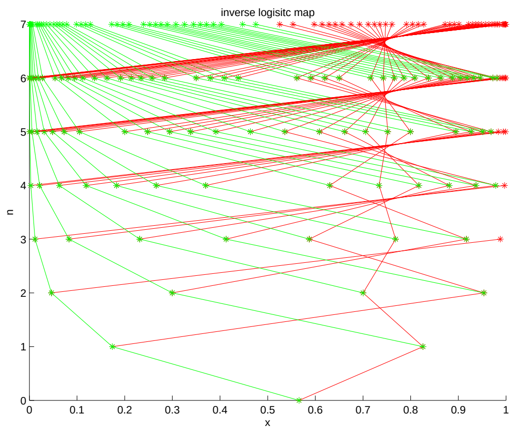
the closed form solutions have been determined(in previous research[4]) to be:

$$x^l_{n-1} = \frac{\frac{1}{2}}{r * (r + \sqrt{r^2 - 4 * r * x_n}} \tag{1}$$

$$x^r_{n-1} = \frac{\frac{1}{2}}{r * (r - \sqrt{r^2 - 4 * r * x_n}} \tag{2}$$

The left and right roots, respectively. Some roots are complex. For our purposes these are ignored. Having to ignore these is not ideal, we would prefer a system that always inverted to a predictable number of values, but this is a proof of concept.

Here is a nice image of the first seven iterations of the inverse logistic map for some arbitrary initial conditions for $r = 3.934233$:



The left roots are red and the right roots are green. after 7 iterations we have 82 possible trajectories out of a possible $2^7 = 128$, this is because there are some complex roots.

Now that we can invert the logistic map, we have the basic tools needed to construct a working deniable cryptosystem.

3.3 Pre-encoding

The validity of this proof of concept is very dependent on the way we pre-encode the plaintext. To convert our raw data to something we can use we do a simple mapping of binary to $[0, 1]$, like:

$$1 \rightarrow x_n \in [.5, 1] \tag{3}$$

$$0 \rightarrow x_n \in [0, .5] \tag{4}$$

In order to randomly choose a proper x_n, we must use a random value from the attractor of the system we are using for our chaos source. In our PoC case we generate entropy by recording many values the logistic map takes for various initial conditions with our private r, and choose one in the proper interval randomly. This way the pre-encoded data blends right in with the natural values of the system, because it is a natural value the attractor takes. This helps confound cryptanalysis, making it harder to spot the message when its in front of your face. If we simply used uniformly distributed random x_n we would be compromising the security of the system in a very big way, these values would be easy to spot statistically.

It is important that we use such a raw binary encoding. It allows for us to have a the largest number of pre-encoded plaintexts to work with for a given plaintext. The fact that the attractor we are working with spends about an equal amount of time in both of these intervals will become important for constructing our deniable PoC.

3.4 Putting it together: Encryption

For simplicity, we are going to break this down so at first we will go through the process of encrypting only one bit. This will make it much easier to understand the overall process, by separating some messiness we incorporated to confound cryptanalysis. This basic example covers the encoding of two one bit messages but could easily extend to k messages with some strings attached.

- Select secret keys r_1 and r_2, as well as secret seeds to our random number generator to generate our n values
- Select our n_1 and n_2, each is chosen separately using separate psudorandom number generators. We will call our corresponding binary plaintext m_1 and m_2
- We pre-encode the m_1 binary message to x_{n_1} as above. m_1 is called the "king bit"
- x_{n_1} is provided as the initial conditions to the inverse logistic map and the tree is constructed, giving us many x_0, we choose a random one.
- if x_0 steps forward to n_2 with r_2 to the interval corresponding to m_2 ($[.5, 1]$ for 1 or $[0, .5]$ for 0)then we are done for this bit. If it does not, we try another random x_0 from the previous step.
- now we have x_0, which is our ciphertext.

the secret key consists of our n values and r. To decrypt we simply step from x_0 to x_{n_1} or x_{n_2} with r_1 or r_2 depending on which message we want.

3.5 Commentary

The values for our n_1 and n_2 are chosen randomly by separate seeded psudo-random number generators. The initial conditions to these are included in their respective secret keys so the receiver can reproduce the sequence to find the n values needed for decryption. The effect of randomizing the n values confounds an attacker who knows the proper r, or is simply looking for the proper r. This effectively weaves the message into the sea of chaos, retrievable if you know exactly where to look, but if you don't its like brute forcing a one time pad, all possible combinations are equally valid. There is no real limit to how many messages we can encrypt at once, it doesn't effect the running time considerably since it is easy to solve for this congruency. In our implementation the message for which we solve the tree is called the king bit, for security reasons this is chosen randomly bit by bit.

In order to retrieve the plaintext without the key one will have to guess the proper r and the proper seed for the random number generator. If we have double precision there is 10^{13} or so valid values for r. For each of these, there would be another 10^{14} or so valid values for the n generator seed. This translates to 10^{27} possible keys to retrieve both messages, if you can spot them when you see them. the equivalent key size in bits would be $log_2(10^{27}) \approx 89.6$ with quad precision we would be dealing with something more like a 200 bit key.

High digit precision is necessary to use values of n of reasonable size. And n of large size are necessary for small perturbations in r to manifest properly, assuring key space size.

It is also worth noting that with a little more trickery the secondary plaintexts could use a separate r. If we use the same r, then an attacker who knows one key has greatly narrowed his search for secondary messages. Though the attacker may now know proper r, he still has to guess the parameters for the n generating sequence. If pre-encoding is done properly, and we use the above method for choosing our n, it should be nearly impossible to pick the message out without exhausting the parameter space for the n generating sequence. By ergodicity, any arbitrary message could be picked out of the static, similar to brute forcing of a one-time pad.

It would require just a little extra computational time to use separate r for our deniable messages. This would greatly improve the security of this as a deniable scheme.

3.6 Security

The security of this system relies largely on the secrecy of r and the steganographic technique of varried n. Any real implementation should stack this system with a classical private key system like AES to ensure security. It is worth noting that the ciphertext will always appear as randomly selected values from the chaotic attractor. For a given cleartext and key there is an infinite number of corresponding ciphertexts. It is also worth noting that varrying the king bit, bit by bit, is a good thing. It makes it much less possible to distinguish one message as primary or secondary, etc. If one was to know the entrophy by which we choose our x_n they could possibly know if the message they have been given is one of many or alone.

3.7 Drawbacks

This algorithm is overall very slow and inefficient, ciphertext will take up a lot of space compared to the raw plaintext. Compression algorithms should be used on the ciphertext to help this problem. Parallel computing and high performance compiled languages should make it of useful speed for encrypting files and the like. It also requires high digit computation to use n values of a more reasonable range.

3.8 The Future

An implementation with a fast compiled language and high digit computation is in order. Such an implemention should include a classical system first, maybe another bitwise stream cipher depending on the application. It is also worth noting that this stream cipher would be very well suited for intense parrellel computation using GPUs. A scheme using a GPU to encrypt/decrypt a hard drive with this may be of useable speed.

An implementation to encrypt files will require a random padding scheme to make the pre-encoded cleartexts to be of equal length. Then it will be hit with a classical private key cipher and subsequently the deniable cipher and last a compression algorithm. Overall this would make for a block cipher.

4 Code Documentation

The PoC code included can be invoked like:

```
ciphertext = encrypt6('message1','key1','message2','key2',....,'messageN','keyN')
messageN = decrypt3(ciphertext,'keyN')
```

The keys must be ascii of length 3 or greater. This is hashed to the parameters r and the parameters for the n generating sequence. messages MUST be of the same length. or, at least the first message must be the shortest for the code to run at all.

It is very slow on matlab.

5 Acknowledgements

We hope this paper serves to help others produce deniable cryptosystems of viable security and computational cost.

Thank you to Jae-Hun Jung, and Gary Davis, Saeja Kim, and Daniel Higgs for motivation and helpful discussion that helped make this possible.

References

[1] M.S. Baptista, Cryptography with chaos. Physics letters A

[2] J.M. Amig a, , L. Kocarev b , J. Szczepanski. Theory and Practice of chaotic cryptography. Physics Letters A

[3] G. lvarez , F. Montoya, M. Romera, G. Pastor. Cryptanalysis of an ergodic chaotic cipher. Physics Letters A

[4] C. Bresten, Jae-Hun Jung A study on the numerical convergence of the discrete logistic map. Commun Nonlinear Sci Numer Simulat