Chaotic Image Encryption

Niklas Fejes

May 1, 2015

1 Introduction

The purpose of this paper is to investigate how the topic of image encryption merges in the fields of chaos theory, cryptography and computer science. Over the last 10 years there has been many attempts to use chaotic systems to achieve encryption of images, and searching Google Scholar for the terms "chaotic image encryption" in publications only in the last five years give over 6,000 results. However, so far no method appears to have managed to get any major acceptance in the cryptography community which is interesting considering the amount of research in the field.

The main motivation behind the introduction of image specific encryption algorithms is that images have big data volumes and high correlation between adjacent pixels, which has been shown to reduce the security and performance in traditional encryption algorithms. The project has mainly been focused on investigating three papers proposing methods for chaos-based image encryption and some of the response they have received from the chaos and cryptography community.

2 Background

The first paper by A. N. Pisarchik et al. [5] suggests a method based on chained iterations of the logistic map to scramble the pixel intensity values of an image. This paper was published in *Chaos: An Interdisciplinary Journal of Nonlinear Science* in 2006, and in 2008 in a comment paper by Ercan Solak and Cahit Çokal [6] was published that exposed three major problems in the algorithm, related to implementation, reversibility and error propagation in finite precision arithmetics. Their conclusions were that two of the problems were easy to account for, while the third related to determinism is a much harder problem to solve.

The second paper by Xiaoling Huang [3] uses a chaotic Chebyshev map to generate a sequence of numbers that is then used to scramble the positions of the pixels in the image. A paper where the method is analyzed with standard tools from cryptoanalysis was published by Xingyuan Wang [7], which suggests that the method is vulnerable to cryptoanalytic methods, and a few improvements to overcome these vulnerabilities were suggested.

The third paper by Guanrong Chen et al.[2] was published in 2004 and is one of the most referenced on the topic of chaos-based image encryption. It proposes an algorithm based on chaotic cat maps to transform an image in a reversible way. The biggest difference from the other two methods is that this technique does not rely on finite precision floating-point arithmetics which is a big

advantage from a computer science perspective.

The three proposed algorithms that are examined in this paper have been implemented in Python, and the source is publicly available online[1]. The reason behind the choice of language is that Python is well suited for algorithmical coding, and that the standard integer type can be of arbitrary size which means that it does not overflow. What is limiting is that iterative algorithms in general are comparatively slow in Python, which means that timing comparisons to the results in the papers can not be made.

3 Theory

In computers images are generally represented as 1D arrays of color intensities, and for an image X of size $M \times N$ this sequence is

$$X = (r[1], g[1], b[1],\ r[2], g[2], b[2],\ \ldots, b[MN])$$

where $r[i], g[i], b[i] \in \{0, \ldots, 255\}$. In all three papers images are considered to be grayscale, and a color image should thus be encrypted as three separate images for each color layer.

3.1 Symmetric-key encryption

The basic idea behind encryption with symmetric keys are as follows: Let $K \in \mathcal{K}$ be a key, often an integer or a number of bits, where $\mathcal{K}$ is the set of possible keys. Let $M \in \mathcal{M}$ be a messages, e.g. a piece of text/image/data represented as a sequence of bytes, where $\mathcal{M}$ is the set of possible messages. We then want the encryption $E_K : \mathcal{M} \to \mathcal{M}$ and decryption $D_K : \mathcal{M} \to \mathcal{M}$ be such that $D_K(E_K(M)) = M$ only if $K_1 = K_2$.

Some properties that a good encryption algorithm should have are:

1. Hard to guess M given $E_K(M)$
2. If $K_1 \approx K_2$, then $D_{K1}(M) \not\approx D_{K2}(M)$
3. Knowing M and $E_K(M)$ should not reveal K
4. Small change in $M \iff$ small change in $E_K(M)$
5. Well-defined in terms of input-output

The second point is one of the reasons why chaos theory has been proposed for encryption applications; chaotic systems are by definition sensitive to initial conditions, small perturbations and small changes in system parameters. If chaotic systems are used in the encryption scheme, where the key affects the system and the system affects the message, we can get that the message will be greatly distorted by the slightest change in key value.

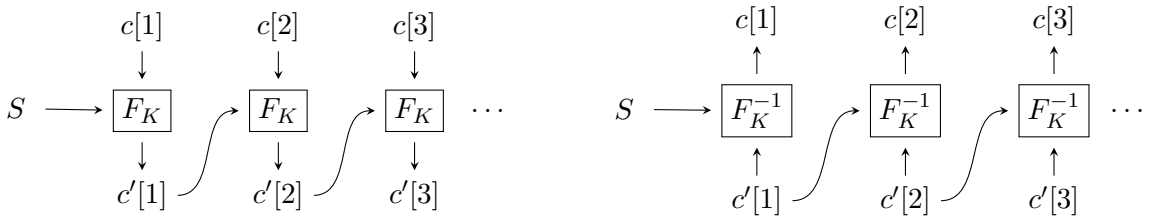
3.2 Generalizing the algorithms

As a step to analyze the three algorithms, they were broken down into high-level parts to get a common algorithmical structure. All proposed algorithms have some kind of *diffusion* technique

[1] `https://github.com/nfejes/chaotic-image-encryption`

which is illustrated in Figure 1. This is a common structure in encryption which ensures that each letter in the output message is affected by each letter in the input message, which helps in achieving point 1 and 4 in the list in Section 3.1. It is important to note that the function F_K (see Figure 1) must have an inverse, since it otherwise would be possible to get the same encrypted message from two different messages. Generally in encryption algorithms, this function is a combination of the nonlinear functions: bitwise xor, bitwise circular shifts, and addition modulo 2^N. These functions are all invertible, acts on standard computer data types and can usually be implemented with single CPU instructions.

The second common structure used in both [3] and [2] is *shuffling*. For an image, this is essentially a parameterized way of scrambling the location of the pixels. Compared to 1D encryption, this allows use of the 2D structure of images which is what separates the topic of image encryption from general encryption. Interestingly, the algorithm proposed by Pisarchik treats the image as a 1D sequence of pixels, so the 2D structure of the image is never actually used.



(a) Diffusion. **(b)** Reverse diffusion.

Figure 1: The process of diffusion. The message is c and the encrypted message is c'. Here F_K is an invertible function depending on the key K and the previous output value, and S is an initialization value that can be either a value depending on the key, or the last entry in the sequence c.

These two methods, diffusion and shuffling, are then repeated a number of times to further confuse the message, possibly with different keys in each layer. On this level of abstraction there is no chaos in the encryption system since we are mapping discrete values to discrete values. This part part of the encryption system must be implemented with logical operations, since it is not possible to implement floating-point functions that have inverses that gives back the exact same floating-point value. Instead, the chaos lies in the implementation of F_K in the diffusion process, and in the way the pixels are moved around in the shuffling.

Algorithm	Definition
Pisarchik[2]	$F_K(c[i]) = f_K(c'[i-1]) + c[i] + A_K \mod 2^N$
Huang	$F_K(c[i]) = \phi_K[i] \oplus \big(c[i] + A_K\, \phi_K[i] + c'[i-1] \mod 2^N\big)$
Chen	$F_K(c[i]) = c'[i-1] \oplus \phi_K[i] \oplus \big(c[i] + \phi_K[i] \mod 2^N\big)$

Table 1: The diffusion functions used in the three papers. The functions are invertible and maps the set $\{0, \ldots, 2^N-1\}$ to itself. The $\oplus$ operator is bitwise xor, N is the number of bits in the data representation, A_K is a constant and ϕ_K is a sequence. The subscript K indicates that the variable depends on the key used. The variables in Pisarchik comes from the discretization of the model, which is explained in Section 4.1.

[2]This definition is the discretization of Pisarchik's definition, where the logistic map's phase space is mapped to $\{0, \ldots, 2^N-1\}$.

3.3 Chaotic systems

The diffusions can be further generalized as can be seen in Table 1. While no motivation is given for why the functions are constructed as they are, the interesting part is how the key-dependent variables are generated by chaos. To generate the key-dependent variables for the diffusion functions, the three algorithms use different ways of generating numbers and sequences based on chaos.

3.3.1 The logistic map

The logistic map $L_a(x) = a\,x\,(1-x)$ is used in both Pisarchik's and Chen's diffusion processes but for different purposes. Pisarchik proposes that the map should be used directly in the diffusion of each pixel (see Table 1), where $f_K(x) = L_a^n(x)$ and the key is $K = (a, n)$. Chen instead uses the map to generate the chaotic sequence ϕ with $a = 4$ and the initial value x_0 derived from the key.

3.3.2 Chaotic Chebyshev maps

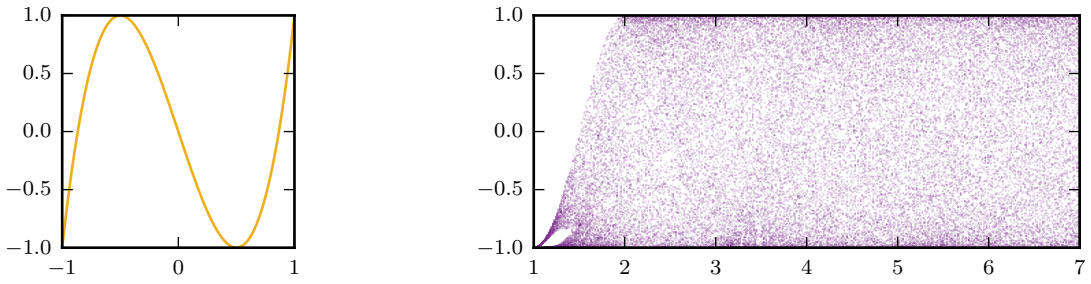
The paper by Huang suggests the use of chaotic Chebyshev maps both to generate the variables for the diffusion function and for shuffling the pixels in the image. The map is defined as

$$T_k(x) = \cos(k\,\arccos(x))$$

and is chaotic for $k \geq 2$ with $x \in (-1, 1)$. In addition to this, if k is integer the map $T_k(x)$ can be expressed as a polynomial of order k with integer coefficients. The paper suggests using $k = 4$, and $k = 6$ with a 2D generalization of the polynomial, such that the two maps used are

$$x_i = 8\,x_{i-1}^4 - 8\,x_{i-1}^2 + 1 \;\big(= T_4(x_{i-1})\big) \qquad \text{and} \qquad \begin{cases} x_i = 1 - 2\,y_{i-1}^2 \\ y_i = 32\,x_{i-1}^6 - 48\,x_{i-1}^4 + 18\,x_{i-1}^2 - 1 \;\big(= T_6(x_{i-1})\big) \end{cases}$$

for the shuffling and the diffusion respectively. The map and the bifurcation plot (1D) are shown in Figure 2.



(a) The Chebyshev map, with $k = 3$.

(b) Bifurcation plot with $k \in [1, 7]$.

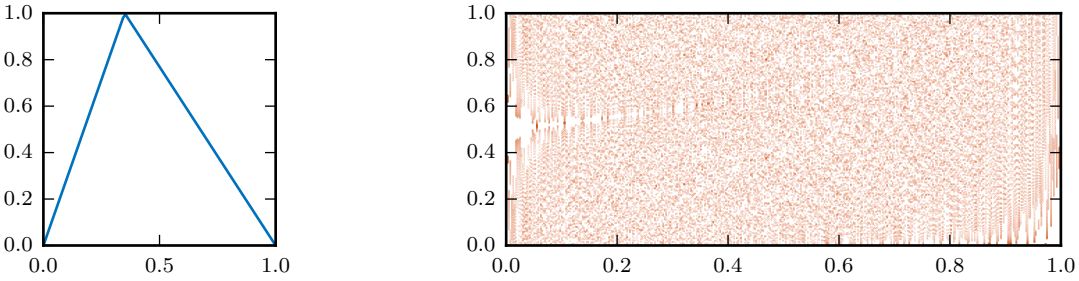
Figure 2: The chaotic Chebyshev map and its bifurcation plot, showing the chaotic behavior for $k \geq 2$. From the plot it is clear that the map is chaotic with the phase space being the interval $(-1, 1)$. As discussed in Section 4.3.1, these properties are good for fixed-point implementations of the algorithm.

3.3.3 Piecewise linear chaotic maps (PWLCM)

Due to the security flaws related to the use of the logistic map as discussed in Section 4.1, the comment paper [6] on Pisarchik's algorithm suggests using a class of chaotic maps known as *piecewise linear chaotic maps* [4] instead of the logistic map. The simplest map of this class is defined as

$$W_p(x) = \begin{cases} \frac{x}{p} & \text{if } x < p \\ \frac{1-x}{1-p} & \text{if } x \geq p \end{cases}$$

and its shape and bifurcation plot is shown in Figure 3.



(a) PWLCM with $p = 0.4$.

(b) Bifurcation plot with $k \in (0, 1)$.

Figure 3: A Piecewise linear chaotic map and its bifurcation plot, showing that it is chaotic for all $p \in (0, 1)$. Note the patterns that occur close to the edges of the bifurcation plot that occur due to the map being close to a single line at those locations.

3.3.4 Chen's chaotic system

Chen's paper also suggests using a 3D continuous chaotic system similar to the Lorenz system, defined as

$$\begin{cases} \dot{x} = a(y - x) \\ \dot{y} = (c - a)x - xz + cy \\ \dot{z} = xy - bz \end{cases}$$

to confuse the user key (known as *key scheduling*), which is illustrated in Figure 4.

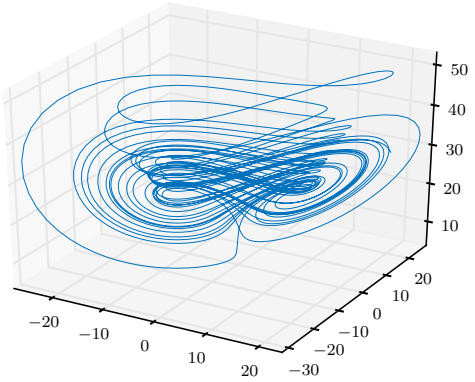


Figure 4: Trajectory in Chen's chaotic system, plotted with $a = 35$, $b = 3$, $c = 28$ and $x_0 = (20, 20, 30)$. By initializing the system and the parameter c from the user key, the system is iterated with a numeric solver and after a number of time steps the position is used to initialize the other systems in the algorithm.

3.3.5 Arnold's cat map and its 3D generalization

Chen's algorithm uses a system known as *Arnold's cat map* to shuffle the pixels, and the definition of the system is

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & a \\ b & 1+ab \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \mod 1$$

where a and b are integer parameters that determine how much the system is skewed. The map can be seen as a skewing the unit square, cutting straight edges, and putting it back together such that it fits in the unit square again.

This map can be discretized such that it maps the set $\mathbb{Z}_\mathbb{N} \times \mathbb{Z}_\mathbb{N}$ to itself (where $\mathbb{Z}_\mathbb{N} = \{1, \ldots, N-1\}$), simply by replacing the mod 1 with mod N in the equation. This is equivalent to using rational number on the form $\frac{a}{N}$. Since the discretized version of the map must be periodic, this means that all points in the discrete system travel along the unstable periodic orbits of the real-valued system.

To further expand the parameter space, Chen's paper suggests the use of a map generalized to three dimensions. While the equations are lengthy and best explained in Section 2 of Chen's paper[2], the general concept is that the unit cube is skewed with three different cat maps along each of the three sides of the cube, and then cut and stitched together back to a unit cube. Since each side is parameterized by two integers, we get that the 3D map has six parameters. To be able to apply the 3D map to a 2D image, the image is rearranged into a number of cubes which are all shuffled independently by the 3D cat map.

3.4 Chaotic sequences

Both Chen's and Huang's methods generate chaotic sequences by iterating maps, which are then used in either diffusion or shuffling. This is essentially a process of generating pseudo-random numbers using chaotic maps. When using the (floating point) sequence generated by iteration in diffusion, Chen's method simply maps the logistic map's state space $[0.2, 0.8]$ (since $a = 4$) to the set $\{0, \ldots, 2^N - 1\}$ and rounds it to integer values. Huang's method is slightly more complicated since the 2D Chebyshev map is used. The 1D sequence is created by alternating the x and y values (e.g. $\{x_1, y_1, x_2, y_2, \ldots\}$), and the mapping from floating point space to discrete space is through the formula

$$\mu(x_i) = \text{floor}(x_i \times 10^{14}) \mod 256.$$

In Huang's algorithm a chaotic sequence is also used to shuffle the pixels. Somewhat simplified, the idea is to generate two sequence of length H and W which are the dimensions of the image. The sequences are then sorted, and the rows and columns of the image are permuted in the same order as the sorting, a process which is illustrated in Figure 5.

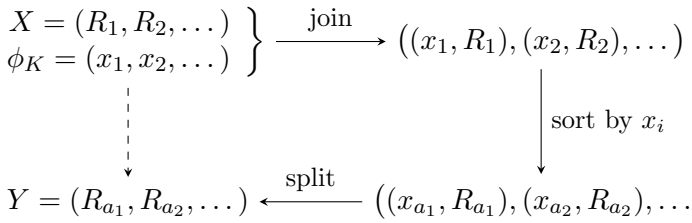


Figure 5: The idea behind a sequence based shuffle. Here R_i represents the i:th row of pixels in image X, and ϕ_K is a chaotic sequence of numbers. The same scheme is applied to the image's columns together with another chaotic sequence.

4 Analysis

4.1 Pisarchik's algorithm

One of the reasons why Pisarchik's proposed algorithm is interesting is that it contains many problems which seem to be common among the encryption algorithms based in chaos theory. While the basic idea illustrated in Figure 6 is theoretically feasible it has some major algorithmical flaws pointed out in [6], and some security flaws pointed out in [1].

The first problem is that the D/A stage in the decryption does not give back the same values as the A/D in the encryption since there is rounding in the process. If floating-point values are used to represent the real numbers, the A/D stage could be skipped and the encrypted message would be stored as floating-point values. This would however cause the encrypted message to be four or eight times the size of the original message, which is usually considered a bad property. Using floating point numbers also implies another problem as pointed out in both [6] and [1]; it is not generally true that $(a + b) - b = a$ which is required to get reversibility in the process.

To get around this, the algorithm can be discretized to resemble the diffusion in Chen's and Huang's algorithm which is essentially the same thing as using fixed-point arithmetics. We get that

$$F(x[i]) = B\big(f^n(x'[i-1]) + x[i]\big)$$

where $B(x)$ truncates the floating point message x such that it lies inside the phase space becomes

$$F(c[i]) = f^n(c'[i-1]) + c[i] + A_K \mod 256.$$

Here, c is the discrete message $c \in \{0, \dots, 255\}$, and $f^n : \{0, \dots, 255\} \to \{0, \dots, 255\}$ which includes the D/A, logistic map interation, and A/D. In the original method the the state space of the encrypted pixels are $x \in [(4\,a^2 - a^3)/16, a/4]$ which is now mapped to the discrete interval $\{0, \dots, 255\}$. Implementation-wise this resembles a lookup table based encryption. For speed the output values of the function f^n should be precomputed since for each input $c \in \{0, \dots, 255\}$ you expect to get the same integer output. The variable A_K comes from the way Pisarchik maps the addition into the phase space, and from the discretization we get that $A_K = 255\, x_{min}/(x_{max} - x_{min})$.

While this generalization makes the algorithm similar to the other two and solves the problem of invertibility, it will likely also induce other statistical vulnerabilities, especially since there is no pixel shuffling in the method.

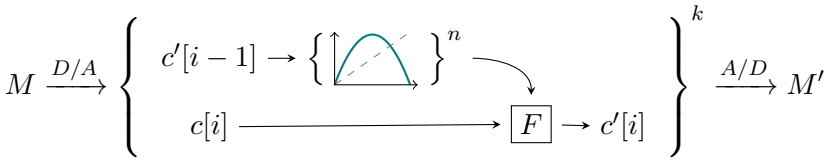


Figure 6: The basic concept behind Pisarchik's algorithm. Here M is the message, M' the encrypted message, a (in the logistic map) and n are keys, and k is a system parameter. In the D/A stage the message M is converted from a sequence of integers to a sequence of real values in the interval $[0, 1]$, denoted $c[i]$, and reverse in the A/D. The initial $c'[i-1]$ is defined as the last entry in c, i.e. $c[m]$ if there are m pixels. F is an invertible function mapping the state space of the logistic map to itself. In the paper it is suggested to use $a \in (3.57, 4)$ and $n \in \mathbb{Z}$ as the key, however a must be chosen such that the system is chaotic which is not generally true for $a \in (3.57, 4)$.

4.2 Shuffling and key space

The process of shuffling pixels is essentially the same thing as permuting the elements in a 1D sequence, where the permutation is parameterized by the key. In theory, an image of size 512×512 can be permuted in 262144! ways and the 3D cat map is a way of parameterizing 64^6 of those. To further increase the key space one can add translative terms to the equation, which has the advantage that the pixel at $(0, 0)$ will not stay at the same point after the shuffle while the properties of the cat map are preserved.

The shuffling based on sorting a chaotic sequence and permuting rows and columns is in theory limited by the number of possible initial values for the sequence. Unlike the cat map however, it is possible for two different initial conditions to generate the same shuffle since the order determines the permutation. It should also be noted that its computational complexity is somewhat higher than the cat map based shuffle since the method requires sorting lists.

4.3 Floating-point arithmetics

Another difference between the shuffles is that the cat map based shuffle is completely deterministic, while the sequence based shuffle depends on the underlying finite precision implementation. If the sequence generating algorithm is based on floating-point values and is not strictly designed in terms of floating-point operation you might get different results with different implementations or on different platforms. Some things that must be considered are:

- Floating-point standard (e.g. IEEE 754) and bit-size
- Operation order, since $(a + b) + c \neq a + (b + c)$
- Rounding error modes and type-cast roundings
- Compiler optimizations and intermediate value precision
- If numerically integrated continuous system; solver and time-step.

The problem with the first of these points is demonstrated in Figure 7, and the other items could be a source of similar results. While it is certainly not impossible to implement two identical floating-point based encryption systems on different platforms, it is not a trivial task.

Another problem is that it is hard to estimate the actual size of the key space. If a single 64-bit floating-point value is used as the key, the size of the key space is far less than 2^{64} since we limit the key to a certain interval. If the key is an initial value for the map, it is possible that two almost equal floating-point number will fall onto the same trajectory which make the keys practically identical. This effect can be seen in both Pisarchik's and Huang's algorithm, where NumPy's `nextafter(x,d)` which gives the next representable floating-point value after `x` in the direction of `d` can be used to create different keys with the same encryption. To prevent this, one can use some key scheming technique like the one mentioned in Section 3.3.4.

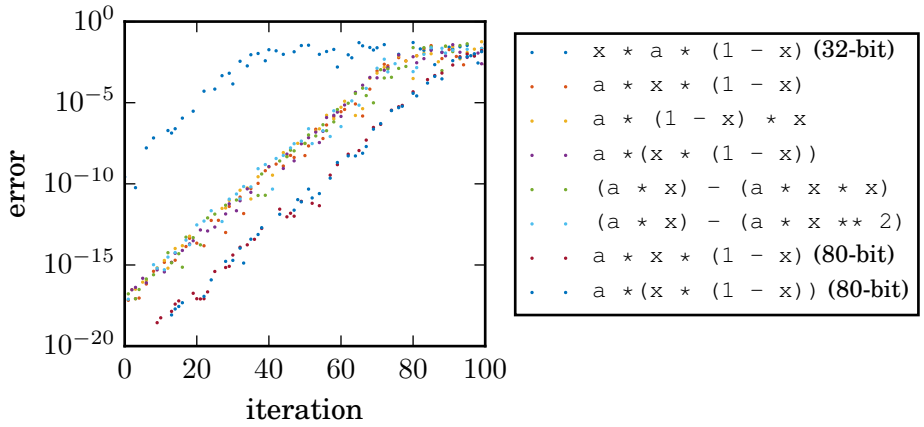


Figure 7: Error propagation in the logistic map for different implementations of the logistic map in Python. The three different precisions (default is 64-bit) form bands, and as expected the error decreases with the number of bits. Note that all five 64-bit implementations are different which shows that the order of multiplication is important. The error is evaluated from a ground truth computed with SymPy's `Rational` class, which is slow to use but has rational precision.

4.3.1 Fixed-point arithmetics

One resolution to the floating-point problem is to use *fixed-point* arithmetics, which is implemented by mapping all real numbers in the system's phase space to some discrete set. An example with the logistic map would be to map $[0, 1] \to \{0, \ldots, 1000\}$, so that the map becomes

$$X_{i+1} = \text{round}\big((A\, X_i\, (1000 - X_i))/1000^3\big)$$

where $A, X_{i+1}, X_i \in \{0, \ldots, 1000\}$. Since it uses integer arithmetics the problems with floating-point implementation can be completely avoided. To maximize the size of the fixed-point phase space the real phase space should be mapped the full integer space used, e.g. $\{0, \ldots, 2^N - 1\}$, which is trivial with Chebyshev polynomials since the phase space is the interval $(-1, 1)$. Another good property is that the map can be expressed as polynomials with integer coefficients, which removes the need of a fixed-point cos/arccos implementations and dealing with approximations of rational numbers.

5 Discussion & conclusions

5.1 Common problems

One of the problems brought up in [1] is that the output values of the logistic map are limited to an interval defined by the key a, so that if you know the interval you can guess the value of a. Another vulnerability is that if it is possible to statistically measure the distribution of outputs from the logistic map in the system, it is possible to guess the key by comparing the distribution to the patterns that occur in the chaotic regions of the logistic map's bifurcation plot. Such statistical weaknesses are the reason why PWLCM's are suggested to be used instead of the logistic map. A similar method is used in [7], where a vulnerability is revealed which allows an attacker to break the encryption.

Another problem is that the number of iterations n is part of the key. This is a vulnerability if the attacker can measure the time it takes to encrypt the message, since the time will depend linearly on n.

5.2 Conclusions

One of the main conclusions that can be drawn from these attempts to achieve encryption from chaos is that it is hard to balance determinism and chaotic behavior in encryption systems. Overly simplified, the process of encryption can be seen as mapping a discrete set of messages to itself, and while chaos can likely find its place in this field, it is difficult to merge the two due to the nature of chaos. The underlying problem comes from the difficulty to combine the approximations of infinite precision systems in chaos with the necessity of discrete mappings in encryption.

Many proposals seem to come from the chaos community without taking into account the aspects of deterministic, well-defined implementations and recent research on cryptoanalysis. While chaos theory can be used to confuse the system and generate pseudo-random sequences, a seemingly better approach would be to use chaos in an attempt to modify and improve the current cryptosystems, instead of coming up with new naive techniques.

References

[1] David Arroyo, Rhouma Rhouma, Gonzalo Alvarez, Shujun Li, and Veronica Fernandez. On the security of a new image encryption scheme based on chaotic map lattices. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):033112, 2008.

[2] Guanrong Chen, Yaobin Mao, and Charles K Chui. A symmetric image encryption scheme based on 3d chaotic cat maps. *Chaos, Solitons & Fractals*, 21(3):749–761, 2004.

[3] Xiaoling Huang. Image encryption algorithm using chaotic chebyshev generator. *Nonlinear Dynamics*, 67(4):2411–2417, 2012.

[4] Shujun Li, Guanrong Chen, and Xuanqin Mou. On the dynamical degradation of digital piecewise linear chaotic maps. *International Journal of Bifurcation and Chaos*, 15(10):3119–3151, 2005.

[5] AN Pisarchik, NJ Flores-Carmona, and M Carpio-Valadez. Encryption and decryption of images with chaotic map lattices. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(3):033118, 2006.

[6] Ercan Solak and Cahit Çokal. Comment on "encryption and decryption of images with chaotic map lattices"[chaos16, 033118 (2006)]. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):038101, 2008.

[7] Xingyuan Wang, Dapeng Luan, and Xuemei Bao. Cryptanalysis of an image encryption algorithm using chebyshev generator. *Digital Signal Processing*, 25:244–247, 2014.