On the Cryptanalysis of two Cryptographic Algorithms that Utilize Chaotic Neural Networks

Ke Qin[*] and **B. John Oommen**[†]

Abstract

This paper deals with the security and efficiency issues of two cipher algorithms which utilize the principles of Chaotic Neural Networks (CNNs). The two algorithms that we consider are: (1) The CNN-Hash, which is a one-way hash function based on the Piece-Wise Linear Chaotic Map (PWLCM) and the One-way Coupled Map Lattice (OCML), and (2) The Delayed CNN-Based Encryption (DCBE), which is an encryption algorithm based on the Delayed CNN. Although both these cipher algorithms have their own salient characteristics, our analysis shows that, unfortunately, the CNN-Hash is not secure because it is neither Second-Preimage resistant nor collision resistant. Indeed, one can find a collision with relative ease, demonstrating that its potential as a hash function is flawed. Similarly, we show that the DCBE is also not secure since it is not capable of resisting known-plaintext, chosen-plaintext and chosen-ciphertext attacks. Furthermore, unfortunately, both the schemes are not efficient either, because of the large number of iteration steps involved in their respective implementations.

1 Introduction

Over the last few decades, the phenomenon of chaos has been widely investigated and applied in a variety of domains including social networks, control systems, and prediction etc. A chaotic system is characterized by salient phenomena such as its sensitivity to initial values, its pseudo-randomness and ergodicity, rendering it to be quite similar to a cryptographic system. The characteristics that render chaotic systems to be akin to cryptographic algorithms are listed below:

1. Chaotic maps vs. Encryption/Decryption algorithms.

   The form of a chaotic system is usually iterative, when the system is discrete, or it involves differential equations when it is continuous. As opposed to this, an encryption/decryption algorithm is usually a nonlinear mapping from the plaintext space to the ciphertext space, and this mapping is, often, not

[*]School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu, China, 611731. E-mail: qinke@uestc.edu.cn. The work of this author is supported by the National Natural Science Foundation of China (Grant NO. 61300093) and Fundamental Research Funds for the Central Universities in China (Grant NO. ZYGX2013J071)..

[†]*Chancellor's Professor* ; *Fellow: IEEE* and *Fellow: IAPR*. School of Computer Science, Carleton University, 1125 Colonel By Dr., Ottawa, ON, K1S 5B6, Canada. E-mail: oommen@scs.carleton.ca. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway. The work of this author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada. The authors are also extremely grateful to the anonymous referees for their input and feedback on the earlier version of the paper. their comments, certainly, improved the quality of the final version.

complex. The similarity between the two is that both of them can yield, as their outputs, results that appear to be random – by virtue of the underlying algorithm repeating certain steps.

2. Iterations vs. Rounds.

   For a chaotic system, each of the steps mentioned above that are "repeated" constitute a so-called "*iteration*". As opposed to this, a cryptographic system involves a sequence of "*rounds*". Only long-term chaotic iterations can yield sequences that appear to be random [1].

3. Controlling parameters vs. Keys.

   If a chaotic system starts from a given initial value, different control parameters can yield different output sequences at each iteration. This, in turn, is analogous to the role of keys in a cryptographic system. The similarity between the two lies in the fact that it is computationally infeasible to deduce the initial input without knowing the controlling parameters or the keys respectively.

4. Sensitive to initial values vs. Diffusion and Confusion.

   When it concerns a chaotic system, a slightly different initial value may result in a significant difference in the output generated after a sufficiently large number of iterations. Analogously, in a cryptographic system, the change of even a single bit (whether it be in the key or the plaintext) should affect most of the ciphertext bits. Furthermore, the statistics relating the plaintext and the key should be "as complicated as possible". Thus, if we regard the plaintext or the key as the initial value, the ciphertext should be highly sensitive to these.

5. Pseudo-random and ergodic.

   The sequence of outputs generated by a chaotic system should be able to fill the entire range in a random-like manner. Analogously, a good encryption algorithm requires that the ciphertexts are randomly distributed in the cipher space.

1.1 Brief Survey of the Field

As a result of the above observations, chaos has also been widely applied in the field of information security since Matthews proposed the first chaotic encryption algorithm [2] in 1984. Later, Baptista and Alvarez reported two cryptographic algorithms based on the phenomenon of chaotic searching in [3], [4] and [5] respectively. While Erdmann *et al* described a stream cipher based on the so-called Henon maps [6], Kanso and his co-authors illustrated a novel hash function [7] and showed how one could achieve digital image encryption based on chaotic maps [8]. Kocarev and his coauthors presented a public-key encryption [9] and random number generators [10] based on chaotic maps. A detailed list of articles that advocate the use of chaotic principles in cryptographic systems can also be found in [11] and [12], and systematic reviews about chaos-based ciphers are found in [13] and [14].

Now that chaotic *maps* have been proven to be useful in encryption, researchers have attempted to use Chaotic Neural Networks (CNNs), which are characterized by much more complicated dynamics than chaotic maps, to develop crypto-systems. The authors of [15–17] proposed different one-way hash functions based on different CNNs. Similarly, Cao *et al* proposed an encryption algorithm based on delayed CNNs [18]. Our present paper concerns some of these results.

1.2 Motivation of this paper

Although the latter above-mentioned authors have affirmed that their schemes are secure and efficient, in this paper, we shall demonstrate that the security levels guaranteed by them are weak, and that they are inefficient. For example, most chaos-based ciphers require an excessive number of iterations, without which the ciphertexts are not sensitive to plaintexts. As opposed to these, traditional ciphers, e.g., the AES, only require a 10-round calculation if one utilizes a key of 128-bits. Further, since chaotic equations are typically specified on the set of real numbers, the associated accuracy of implementing these schemes using digital computations is also problematic. Indeed, when we implement the associated computations numerically, we observe that some of the significant digits will be automatically truncated, and the consequence of this is that the original system which was chaotic within the domain of "real" numbers, is no longer chaotic [13]! Also, the improvement brought about by increasing the accuracy using higher-precision software entails a larger computational cost.

In this paper, we analyze two typical CNN-based cipher systems, the first of which is a one-way hash function, and the second is an encryption method. However, we believe that our analysis is also valid for other CNN-based schemes.

2 The CNN-based Hash Function

2.1 The Description of the CNN-based Hash Function

The authors of [15] proposed a novel one-way hash function based on a special CNN. The structure of the network[1] is shown in Fig. 1.

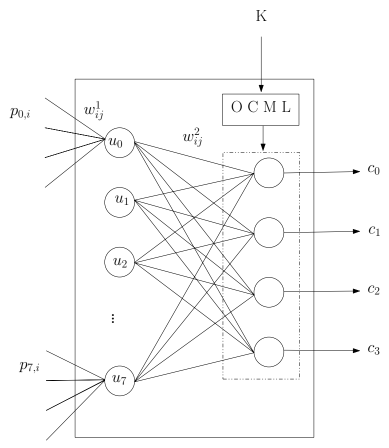


Figure 1: The structure of the network used for the CNN-Hash.

[1]More details about PWLCM's dynamics and analysis can be found in [19] and omitted here to avoid repetition.

More specifically, they used two chaotic maps, namely, the Piece-Wise Linear Chaotic Map (PWLCM, see Eq. (1)) and the Logistic map:

$$f(x) = \begin{cases} \frac{x}{Q}, & 0 \leq x < Q \\ \frac{x-Q}{0.5-Q}, & Q \leq x < 0.5 \\ \frac{1-Q-x}{0.5-Q}, & 0.5 \leq x < 1-Q \\ \frac{1-x}{Q} & 1-Q \leq x \leq 1, \end{cases} \tag{1}$$

where Q is a control parameter, which is a real number between 0 and 0.5.

The network has a single input layer with 8 neurons, and a single output layer with 4 neurons. Each of the input neurons can receive 4 external inputs $p_{i,j}, i = 0, 1, \cdots, 7;\ j = 0, 1, 2, 3$, where each $p_{i,j}$ consists of 8 bits. If $P_i = [p_{i,0}, p_{i,1}, p_{i,2}, p_{i,3}]^T$, we see that the CNN can receive a 256-bit external input sequence. Each of the output neurons can now generate a 32-bit output sequence, where the One-way Coupled Map Lattice (OCML), specified by Eq. (2) - (5) is used to control the output neurons. The associated weights $\{w_{ij}\}, i = 0, 1, \cdots, 7;\ j = 0, 1, 2, 3$ for each connection is a constant, $W^1 = [1/2^8, 1/2^{16}, 1/2^{24}, 1/2^{32}]$. Further, the internal state of the input neuron u_i is given by $W^1 P_i$. Let $U = [u_1, u_2, \cdots, u_7]^T$ be the internal state vector.

In all brevity, we remark that the CNN compresses a 256-bit sequence to yield a 128-bit sequence.

$$\begin{aligned} x_0(t+1) &= (1-\varepsilon)g(x_0(t)) + \varepsilon g(x_3(t+1)); & (2) \\ x_1(t+1) &= (1-\varepsilon)g(x_1(t)) + \varepsilon g(x_0(t+1)); & (3) \\ x_2(t+1) &= (1-\varepsilon)g(x_2(t)) + \varepsilon g(x_1(t+1)); & (4) \\ x_3(t+1) &= (1-\varepsilon)g(x_3(t)) + \varepsilon g(x_2(t+1)). & (5) \end{aligned}$$

where $g(x)$ is the Logistic map and ε is a coupling factor between 0 and 1.

We now present the process involved in the hash function:

1. **Data Preparation:** Divide the given plaintext into small blocks P_i, where each block is 4×8 bits long. All together, there are 8 such blocks. Thus, the network is able to accept a 256-bit length input sequence at a time.

2. **Data Formatting:** Format the input integer numbers to be real number between [0, 1] by means of the PWLCM. To be specific, this is achieved by using $u_i = f^\tau(W^1 P_i, Q)$, where τ is the number of iterations that is enforced so as to yield the required "diffusion" and "confusion", and $Q \in (0, 0.5)$ is the control parameter. The authors of [15] have suggested to set $\tau = 40$ and $Q = 1/3$.

3. **Key Preparation:** For the given 128-bit key K, divide it into 4 32-bit sequences K_0, K_1, K_2, K_3. Using these, compute $k_i = K_i/2^{32}, i = 0, 1, 2, 3$. The four values of $\{k_i\}$ are used as the initial values of the OCML. The authors suggested to set the value of ε as $\varepsilon = 1/3$.

4. **Hash Computing:** For every 30 iterations, record a vector $X^0 = [x_0^0, x_1^0, x_2^0, x_3^0]$, and repeat this until we have gathered 10 vectors. The vectors $X^0, X^1, \cdots, X^7$ are used as the connection weights W^2

between the input and output neurons, $W = [X_0^T, X_1^T, \cdots, X_7^T]_{4\times 7}$. $\Theta = [x_0^8, x_1^8, x_2^8, x_3^8]$ is set as the threshold, and $Q = [x_0^9, x_1^9, x_2^9, x_3^9]$ is used as the PWLCM's control parameter.

5. **Output Preparation:** The output of each neuron is given by:

$$c_i = f^\tau(\mathbf{mod}(W_i^2 U + \Theta_i, 1), Q_i), \tag{6}$$

where W_i^2 means the i^{th} row of W^2.

6. **Loop:** Repeat the above steps until all message blocks have been processed.

7. **Assembling:** Transform the output of each neuron of the last CNN to be a 32-bit sequence, and then combine the four 32-bit sequences to be be the final 128-bit hash value, as shown in Fig. 2.

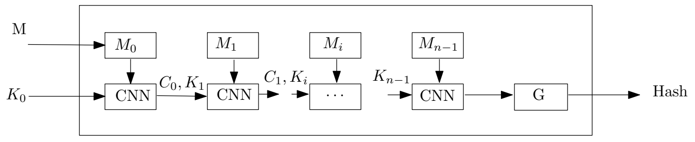


Figure 2: The CBC mode hashing process.

Summary: The entire process of the CNN-Hash can be summarized by the following equations:

$$\begin{aligned} u_i &= f^\tau(W^1 P_i, Q), & (7)\\ c_i &= f^\tau(\mathbf{mod}(W^2 U + \Theta_i, 1), Q_i), & (8)\\ H &= G(C). & (9) \end{aligned}$$

where W^2, Θ, Q are computed according to the CNN, W^1, τ are given constants, P_i is transformed from the plaintext.

2.2 The Analysis of the CNN-based Hash Function

Although the authors of [15] claimed that this CNN-Hash has good properties such as its sensitivity to the plaintext and the key, its one-way computation power, its anti-birthday attack etc., our analysis below proves that it is not secure.

As is well known, a good one-way hash function (both keyed or unkeyed) must satisfy the following conditions [20]:

1. Efficiency: For a given key k and message m, it must be easy to compute the Message Authentication Code (MAC): $H(m, k)$.

2. Preimage Resistance: For a given value H^*, it must be computationally infeasible to find x such that $H(x, k) = H^*$

3. Second-Preimage Resistance: For a given message x, it must be computationally infeasible to find a different message y such that $H(x, k) = H(y, k)$.

4. Collision Resistance: It must be computationally infeasible to find two different messages x and y such that $H(x, k) = H(y, k)$, where the two inputs x and y can be freely chosen.

We now evaluate the properties of the CNN-Hash by using the above metrics.

1. **Analysis on Efficiency**:

   As explained above, the computations needed for the CNN-Hash are done on the elements of the real numbers in [0, 1], which is, unarguably, much slower than the corresponding computations executed on the set of integers. Besides, according to Step 4), we have to do at least 300 iterations to compute the first output C_0, which is thereafter used as the input for hashing the second block. Therefore, for hashing a message of 1MB, we need at least $1024 \times 1024 \times 8 \times 300/256 = 9,830,400$ iterations, which is a computationally intensive task. The authors of [15] have stated that their algorithm is not competitive against MD5 or SHA, and said that it requires almost twice as much computation as both of them. Our analysis and experiments, however, show that the performance is even worse than they claimed. To confirm this, we mention that we conducted a simulation on an Intel Celeron CPU E1500 (2.20GHz) with 4G main memory, and the time involved for the CNN-Hash for a 1MB input of text was almost 59.83s – which is much more expensive than the cost of both the MD5 and the SHA.

2. **Analysis on Preimage Resistance**:

   Because chaotic maps have ergodic and stochastic properties, it is, indeed, not possible to find the inverse of a given value. This is especially true of the CNN-Hash which uses two different chaotic systems. From this perspective, we agree with that the CNN-Hash is preimage resistant even when the key K is known.

3. **Analysis on Second-Preimage Resistance**:

   Although the CNN-Hash is preimage resistant, it is *not* Second-Preimage Resistant. The reason for this is quite straightforward. Consider Eq. (7) – (9) from which we see that the final hash value only depends on the initial value P_i and the key K. Thus, if we are able to find another different P_i^* such that $f^\tau(W^1 P_i^*, Q) = f^\tau(W^1 P_i, Q)$, we can conclude that the subsequent intermediate/final results are exactly the same if the system uses the same key. For example, consider Eq. (1) and the iteration trajectories of the PWLCM as shown in Fig. 3. From examining these, we see that we can determine four different values:

   $$x_1 = 0.3, x_2 = 0.475, x_3 = 0.525, x_4 = 0.7,$$

   sharing the same iteration trajectories yielding the final result $f^\tau(x) = 0.39887$. Thus, if we let $v = W^1 P_i$ and $f(v, Q) = F$ (where F is some specified value), by examining Eq. (1), we see that we can have at least four solutions for $\{v\}$:

   $$v_1 = FQ, v_2 = F(0.5 - Q) + Q, v_3 = (1 - Q) - F(0.5 - Q), v_4 = 1 - FQ.$$

We can thus have four different $\{P_i^*\}$ each of which is the solution of $W^1 P_i^* = v_i$, whence we see that the CNN-Hash is *not* Second-Preimage resistant.

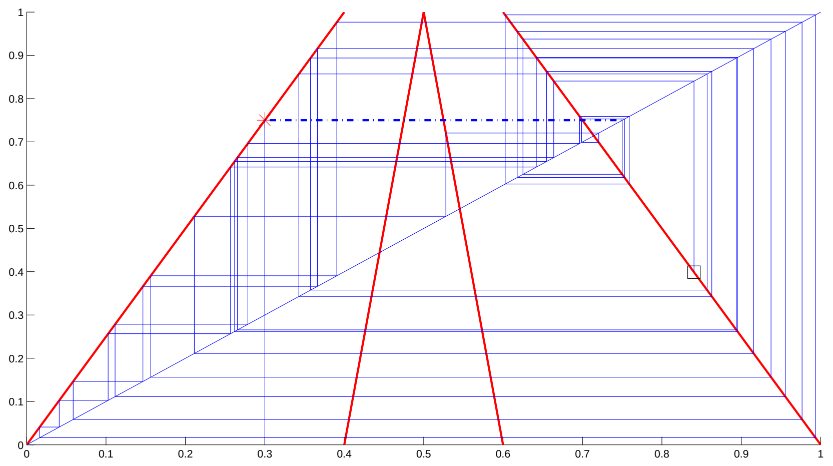


Figure 3: An example of the PWLCM's iteration trajectories. The four red bold lines make up the image of the PWLCM. The line $y = x$, vertical and horizonal lines indicate the iteration process. In this figure, an iteration begins at starting point $(0.3, 0.75)$ (marked with $*$) and ends at $(0.84, 0.40)$ (marked with a "square"). The reader should note that associated with the line $y = 0.75$ (marked with dash-dot line), there are at least four starting points that share the same trajectory. These are, namely, the points $(0.3, 0.75)$, $(0.475, 0.75)$, $(0.525, 0.75)$ and $(0.7, 0.75)$, which, in turn, implies that there is a collision for at least four different initial inputs.

4. **Analysis on Collision Resistance**:

   The analysis on collision resistance is quite similar to the analysis on Second-Preimage resistance, and is omitted here in the interest of brevity.

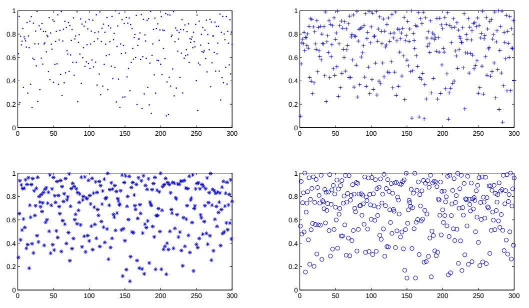
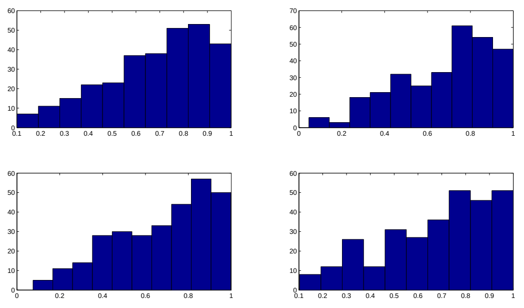
Besides the above four conclusions, we can also claim:

1. **The OCML component has many "*weak keys*"**.

   According to Step 3), the initial values of the OCML come from the initial key K. Based on the above, one can see that *those* keys which lead to the four equal parts are necessarily weak keys. Further, the reader should observe that since the CNN is a fully-connected network, if $k_0 = k_1 = k_2 = k_3$, we can conclude that no matter how many iterations have been done, the condition $x_0(t) = x_1(t) = x_2(t) = x_3(t)$ always holds, which implies that a message of length 256-bits compresses to be 32 bits long instead of being 128 bits long. Thus, in this case, we see that it is feasible to find a collision since the ciphertext space is contracted.

2. **Hash values do not obey a uniform distribution**.

   The OCML employs the Logistic chaotic map, whose values are not uniformly distributed in [0, 1]. To demonstrate this, we have computed the statistics of the distribution, and these are shown below

(*a*) (*b*)

Figure 4: The distribution of the values of the OCML. (*a*): Trajectories of 300 points. (*b*): The distributions of the 300 points in ten intervals from 0 to 1. The four figures in (*a*) and (*b*) are for $x_0(t)$, $x_1(t)$, $x_2(t)$, $x_3(t)$ respectively.

in Fig. 4 (*a*) and (*b*). We can clearly see from the two figures that most of the values fall into the intervals close to unity. This will cause the distribution of the hash values to also be non-uniform, further implying that the probability of collision is high in certain parts of the interval [21].

3 The Delayed CNN-based Cryptography

3.1 The Description the Delayed CNN-based Cryptography

Delayed CNNs have been widely investigated in the past decades. The authors of [18] proposed a cryptographic system based on a special type of the delayed CNN. The model used in [18] is also a Hopfield-like NN which exhibits chaotic phenomenon and which obeys Eq. (10):

$$\frac{dx_i(t)}{dt} = -c_i x_i(t) + \sum_{j=1}^{n} a_{ij} f(x_j(t)) + \sum_{j=1}^{n} b_{ij} f(x_j(t - \tau_{ij}(t))) + I_i(t), \text{ where} \tag{10}$$

1. n denotes the number of units in the CNN,
2. $x(t) = \{x_1(t), x_2(t), \cdots, x_n(t)\} \in R_n$ is the state vector associated with the neurons,
3. $I = \{I_1, I_2, \cdots, I_n\} \in R_n$ is the external input vector,
4. $f(x(t)) = \{f_1(x_1(t)), f_2(x_2(t)), \cdots, f_n(x_n(t))\} \in R_n$ are the neurons' activation functions,
5. $\tau(t) = \tau_{ij}(t)(i, j = 1, 2, \cdots, n)$ are the time delays,
6. $C = diag(c_1, c_2, \cdots, c_n)$ is a diagonal matrix, and
7. $A = (a_{ij})_{n\times n}$ and $B = (b_{ij})_{n\times n}$ are the connection weight matrix and the delayed connection weight matrix, respectively.

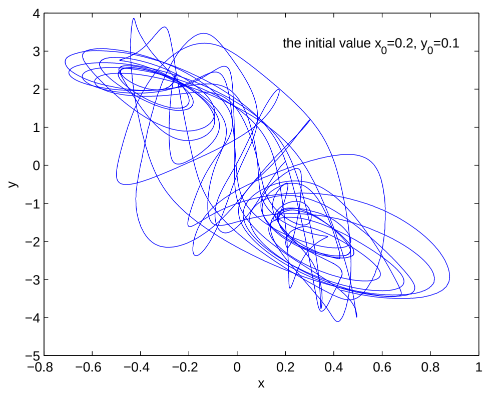


Figure 5: The trajectories of Eq. (10). In this figure, the values of $x(t)$ and $y(t)$ are calculated by means of the fourth-order Runge-Kutta method. The time span is from 0 to 200 with a total of 30,000 steps.

The dynamics of Eq. (10) have been well studied and it is reported that it can exhibit rich chaotic phenomenons [22, 23]. As demonstrated in [18, 23], if the parameters are:

$$A = \begin{pmatrix} 2.0 & -0.1 \\ -5.0 & 3.0 \end{pmatrix}, B = \begin{pmatrix} -1.5 & -0.1 \\ -0.5 & -2.5 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

and if
$f_i(x_i(t)) = tanh(x_i(t))$,
$\tau(t) = 1 + 0.1sin(t)$, and
$I = 0$,
the trajectories of Eq. (10) are shown in Fig.5

The encryption and decryption schemes proposed in [18] are based on the above Eq. (10) and can be summarized as following:

- **Initialization:** Obtain the starting point x_0 from the last N_0 transient time iterations as $x_0 = x_1(N_0 h)$ where h is the discretized time step.

- **Data Preparation:** Divide the plaintext m into subsequences m_j of length l bytes, e.g., $l = 4$. That is, any message m can be digitized as:

  $$m = \underbrace{p_0, p_1, \cdots, p_{l-1}}_{m_0} \underbrace{p_l, p_{l+1}, \cdots, p_{2l-1}}_{m_1} \underset{\cdots}{\cdots}$$

  where p_i is an 8-bit binary string. Then combine four p_i to form a 32-bit binary block, implying that $P_j = p_j, p_{j+1}, p_{j+2}, p_{j+3}$.

The following steps constitute the core process of encryption:

1. **Dynamic Parameter Computing:** Iterate the initial value x_k 38 times and to yield $x_{k+1}, x_{k+2}, \cdots, x_{k+38}$. Extract *one* bit from the 38 numbers and to obtain a 38-bit random binary sequence,

   $B_i = B_i^{k+1} B_i^{k+2} \cdots B_i^{k+38}$,

   where $B_i^k = b_i(x_k)$, is computed as per:

$$b_i(x_k) = \sum_{r=1}^{2^i-1} (-1)^{r-1} \Theta_{(e-d)(r/2^i)+d}(x_k), \tag{11}$$

   and where e and d are the upper and lower bounds of x_k respectively.

$$\Theta_{threshold}(x_k) = \begin{cases} 0, & x_k < threshold \\ 1, & x_k \geq threshold. \end{cases} \tag{12}$$

   Denote:

   $A_j = B_i^1 B_i^2 \cdots B_i^{32}$,

   $A_j^1 = B_i^{33} B_i^{34} \cdots B_i^{37}$, and

   $A_j^2 = B_i^{38}$.

   Also, let D_j denote the decimal value of A_j^1.

2. **Permutation:** Permute the message block P_j with a left cyclic shift D_j bits and the message block A_j with right cyclic shift D_j bits, to obtain P_j^* and A_j^*. If $A_j^2 = 0$, the $x(t)$ is used for the successive block iteration illustrated in Step 1). Otherwise, $y(t)$ is used as the initial value of the next iteration.

3. **Encryption by XOR:** Encrypt the message block P_j by XOR operations (represented by $\oplus$) to yield:

$$C_j = P_j^* \quad \oplus \quad A_j^*. \tag{13}$$

4. **Loop:** Reset the initial value by $x(0) = x(38 + D_j)$ (or $x(0) = y(38 + D_j)$, this depends on the value of A_j^2.) and repeat the above steps till all blocks are encrypted.

As for the decryption, the steps are very similar to the encryption process except in the case of Step 3) where:

$$P_j^* = C_j \quad \oplus \quad A_j^*. \tag{14}$$

The plaintext P_j can be recovered by performing inverse permutations with right cyclic shifts of D_j bits.

3.2 The Analysis of the Delayed CNN-based Cryptography

We now proceed to analyze the security and performance of the delayed CNN-based cryptography. Our goal is to demonstrate that this cryptography has several weaknesses:

1. **Non-randomness**:

   x and y are not uniformly distributed, which causes the "*random*" bits generated in Step 1) to be non-random. To illustrate this, we present the frequency statistics of the value of $x(t)$ and $y(t)$. The parameters used here are exactly the same as those used in Fig. 5. We categorize the combination of $x(t)$ and $y(t)$ into 4 classes:

   (a) $x \geq 0$ AND $y \geq 0$: 1801

   (b) $x \geq 0$ AND $y < 0$: 15618

   (c) $x < 0$ AND $y \geq 0$: 10781

   (d) $x < 0$ AND $y < 0$: 1800

   We can clearly see from the statistics that more than a half (52.06%) of the $x(t)$ and $y(t)$ gather in the first quadrant, while only 48.94% distribute in the other three quadrants. This phenomenon is confirmed from Fig. 5. Furthermore, as demonstrated in Step 1), we can normalize $x(t)$ and $y(t)$ into $[0, 1]$ by:

   $$g(x) = \frac{x-d}{e-d} = 0.b_1(x)b_2(x)\cdots b_i(x)\cdots b_n(x) \tag{15}$$

   where e and d are the upper and lower bounds of x respectively. We can thus generate the "random" binary bits according to $g(x)$. Indeed, the new counts are:

   (a) $b(x) = 0$ AND $b(y) = 0$: 2769

   (b) $b(x) = 0$ AND $b(y) = 1$: 11573

   (c) $b(x) = 1$ AND $b(y) = 0$: 14379

   (d) $b(x) = 1$ AND $b(y) = 1$: 1279

   Clearly, the bits generated by Eq. (12) are not "random".

2. **Trajectory behavior**:

   The authors of [18] did not use the trajectories as shown in Fig. 5 directly. Instead, the random bits were generated according to the 38 successively iterations, as demonstrated in Step 1). We should thus carefully check the randomness of the corresponding sequences. According to Step 2) in Section 3.1, if $A_j^2 = 0$, $x(t)$ is used for the successive iteration, otherwise, it is $y(t)$. In this case, we swap the value of $x(t)$ and $y(t)$ every 38 iterations. As shown in Fig. 6 we can see that the value of $x(t)$ and $y(t)$ are very close during the 38 iterations, which means the random bits $B_i^1 B_i^2 \cdots B_i^{38}$ are almost identical.

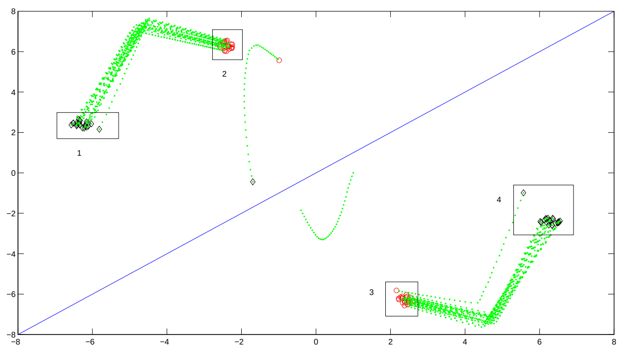


Figure 6: The controlled trajectories of Eq. (10). For a better view, we have used a larger step 0.05 yielding a lesser number of points. The points in contained in rectangles marked as 1 and 3, 2 and 4 are symmetric pairs along the axis given by the line $y = x$.

In spite of the above, the authors of [18] attempted to use this sequence to achieve the goals of "*diffusion*" and "*confusion*". It is well known that a sequence possessing poor randomness properties cannot be used in any cryptographic algorithm [21], because it would otherwise lead to a more predictable ciphertext. Consequently, we argue that this algorithm is not secure.

3. **Resistance to attacks**:

This cryptographic system cannot resist known plaintext attacks, chosen plaintext attacks and chosen ciphertext attacks. To demonstrate this, assume that an attacker has some plaintext-ciphertext pairs (M_1, C_1), (M_2, C_2) and (M_3, C_3), where $\{M_i\}$ are the first 4 bytes of different plaintexts. If they are all encrypted by the same key, according to the algorithm, then A_j^*, D_j and some other intermediate iteration results should be the same. Thus:

$$\begin{aligned} C_1 &= (M_1 << D_j) \oplus A_j^* \\ C_2 &= (M_2 << D_j) \oplus A_j^*. \end{aligned}$$

where $<<$ denotes the cyclic left shift operation. Thus,

$$\begin{aligned} C_1 \oplus C_2 &= (M_1 << D_j) \oplus (M_2 << D_j) \\ &= (M_2 \oplus M_2) << D_j. \end{aligned}$$

Since (M_1, C_1) and (M_2, C_2) are known, it is quite easy to find the value of D_j. After that, we can solve the equation $C_1 \oplus C_3 = (M_1 \oplus M_3) << D_j$ and thereafter determine M_3 successfully. Observe that during the whole process, we did not need any knowledge about the delayed CNN. The reason why we are able to proceed with such attacks is that the authors did not introduce the concept of the *Initial Vector* to the scheme.

4. **Efficiency**:

   Although the authors of [18] claimed that the algorithm is efficient, this is not really the case. Actually, this conclusion is also true for many other crypto-systems such as those algorithms presented in [8,24], which involve time delays in their equations. It is well known that the Runge-Kutta method is one of the best ways to solve differential equations where the initial values are provided. However, this method is still far too expensive when compared to traditional block ciphers such the DES or AES. Indeed, the computation of these traditional ciphers involves a finite field and only makes use of simple operations such as permutation. As opposed to this, solving differential equations involves the set of real numbers. For example, to encrypt a plaintext with size 1M bytes, we have to divide the message into $1024 \times 1024/4 = 262,144$ blocks, where each block is of length 4 bytes. According to the encryption phase, at least $N_0 + 38$ iterations are involved to encrypt a single block. If we assume that $N_0 = 62$, we see that we have to thus do approximately $262,144 \times 100 = 26,214,400$ iterations to encrypt the entire file, which is, really, prohibitively large. More specifically, on an Intel Celeron CPU E1500 (2.20GHz) with 4G main memory, this encryption time using Matlab was about 7 minutes, which is unacceptable when compared to the "real time" operation of traditional block ciphers.

5. **Statistical Attacks**[2]:

   The reader should take note of the fact that the block size was increased from 64 bits in DES to 128 bits in AES in order to avoid statistical attacks. Thus, it is not recommended that one uses blocks whose sizes are less than 128 bits in modern block ciphers [25]. Consequently, the fact that the Delayed CNN-based Cryptography still relies on Exclusion OR operations involving strings of length 32-bits, renders it more susceptible to statistical attacks.

4 Conclusion

Chaotic Neural Networks have been widely used in various fields such as pattern recognition, dynamic associate memory and optimization. Recently, cryptography based on chaos or CNNs has drawn great attention. In this paper, we present a detailed analysis of two typical cipher schemes: The CNN-Hash and Delayed CNN-Based Encryption. The former compresses a plaintext onto a 128-bit sequence, which is similar to MAC. The latter encrypts plaintext so that an eavesdropper will not be able to decrypt the message without the key, which is analogous to common cipher algorithms. Although the authors have affirmed that their schemes are secure and efficient, our investigation proves that these claims are not valid. We have proven that the CNN-Hash is not Second-Preimage resistant and collision resistant. The DCBE has also been shown to not be secure since an attacker can partially recover the plaintext by using a known plaintext attack, a chosen-plaintext attack or chosen-ciphertext attack. We have also concluded that the two schemes are not computationally efficient.

[2]We sincerely thank an the anonymous Referee who provided this insight.

References

[1] F. James, “Chaos and randomness,” *Chaos, Solitons & Fractals*, vol. 6, pp. 221–226, 1995.

[2] R. Matthews, “On the derivation of a “chaotic” encryption algorithm,” *Cryptologia*, vol. 8, no. 1, pp. 29–41, 1984.

[3] M. S. Baptista, “Cryptography with chaos,” *Physics Letters A*, vol. 240, no. 1-2, pp. 50–54, 1998.

[4] E. Alvarez, A. Fernandez, P. Garcia, J. Jimenez, and A. Marcano, “New approach to chaotic encryption,” *Physics Letters A*, no. 266, pp. 373–375, 1999.

[5] G. Alvarez, F. Montoya, M. Romera, and G. Pastor, “Cryptanalysis of a chaotic encryption system,” *Physics Letters A*, vol. 276, no. 1-4, pp. 191–196, 2000.

[6] D. Erdmann and S. Murphy, “Henon stream cipher,” *Electronics Letters*, vol. 28, no. 9, pp. 893–895, 1992.

[7] A. Kanso, H. Yahyaoui, and M. Almulla, “Keyed hash function based on a chaotic map,” *Information Sciences*, vol. 186, no. 1, pp. 249–264, 2012.

[8] A. Kanso and M. Ghebleh, “A novel image encryption algorithm based on a 3d chaotic map,” *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 7, pp. 2943–2959, 2012.

[9] L. Kocarev and Z. Tasev, “Public-key encryption based on chebyshev maps,” in *Proceedings of the 2003 International Symposium on Circuits and Systems*, ser. Proceedings of the 2003 International Symposium on Circuits and Systems, vol. 3, 2003, pp. 28–31.

[10] T. Stojanovski and L. Kocarev, “Chaos-based random number generators - part i: Analysis,” *Ieee Transactions on Circuits and Systems I-Fundamental Theory and Applications*, vol. 48, no. 3, pp. 281–288, 2001.

[11] K. Qin, M. T. Zhou, and N. Q. Liu, “A novel group key management based on jacobian elliptic chebyshev rational map,” *Lecture Notes in Computer Science*, vol. 4672, pp. 287–295, 2007.

[12] A. A. Zaher and A. Abu-Rezq, “On the design of chaos-based secure communication systems,” *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 9, pp. 3721–3737, 2011.

[13] S. J. Li, “Analysis and design of digital chaotic ciphers.” Ph.D. dissertation, Xi’an Jiaotong University, China, 2003.

[14] L. Kocarev, “Chaos-based cryptography: a brief overview,” *IEEE transactions on Circuits and Systems Magazine*, vol. 1, no. 3, pp. 6–21, 2001.

[15] G. J. Liu, L. Shan, Y. W. Dai, J. S. Sun, and Z. Q. Wang, “One-way hash function based on chaotic neural network,” *Acta Physica Sinica*, vol. 55, no. 11, pp. 5688–5693, 2006.

[16] Y. T. Li, S. J. Deng, and D. Xiao, “A novel hash algorithm construction based on chaotic neural network,” *Neural Computing & Applications*, vol. 20, no. 1, pp. 133–141, 2011.

[17] Q. T. Yang and T. G. Gao, “One-way hash function based on hyper-chaotic cellular neural network,” *Chinese Physics B*, vol. 17, no. 7, pp. 2388–2393, 2008.

[18] W. W. Yu and J. D. Cao, “Cryptography based on delayed chaotic neural networks,” *Physics Letters A*, vol. 356, no. 4-5, pp. 333–338, 2006.

[19] D. Hinrichsen and A. J. Pritchard, *Mathematical Systems Theory I - Modelling, State Space Analysis, Stability and Robustness*. Berlin, Germany: Springer Verlag, 2005.

[20] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. FL, USA: CRC Press, 2001.

[21] D. Eastlake and J. Schiller, “Randomness requirements for security,” *IETF RFC 4086*, 2005.

[22] H. T. Lu, “Chaotic attractors in delayed neural networks,” *Physics Letters A*, vol. 298, no. 2-3, pp. 109–116, 2002.

[23] D. Zhang and J. Xu, “Chaotic and hyperchaotic attractors in time-delayed neural networks,” *Complex Sciences*, vol. 5, pp. 1193–1202, 2009.

[24] E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, “Public-channel cryptography using chaos synchronization,” *Physical Review E*, vol. 72, no. 1, p. 016214, 2005.

[25] W. B. Mao, *Modern Cryptography: Theory and Practice*. NJ, USA: Prentice Hall, 2003.