Multi-Map Orbit Hopping Chaotic Cipher

Xiaowen Zhang¹, Li Shu², and Ke Tang³

¹Computer Science Dept., College of Staten Island / CUNY, 2800 Victory Blvd, Staten Island, NY 10314, USA
²College of Computer Science, Sichuan University, Chengdu, Sichuan 610065, P.R. China
³Computer Science Dept., Graduate Center / CUNY, 365 5th Ave., New York, NY 10016, USA

Abstract - In this paper we propose a multi-map orbit hopping chaotic cipher that utilizes the idea of spread spectrum mechanism for secure digital communications and takes advantage of fundamental chaos characteristics of mixing, unpredictability, and extreme sensitivity to initial conditions. The design, key and subkeys, and detailed implementation of the system are addressed. A variable number of well studied chaotic maps form a map bank. The key determines how the system hops between multiple orbits, the number of orbits for each map, and the number of sample points for each orbit. For illustration purpose an example is provided.

Keywords: chaos, chaotic map, stream cipher, security.

1 Introduction

Chaotic cryptography has been attracting more and more attention from nonlinear system researchers because the essence of a chaos dynamic system matches the very basic criteria of cryptography. The mixing in a chaos system matches the confusion of a cryptosystem and sensitivity to initial conditions corresponds to the diffusion of a cryptosystem. Moreover the links to conventional cryptography that has a wide variety of appropriate design and analysis methods have been established [1, 2].

In the frequency hopping spread spectrum system, by following a specific hopping pattern the carrier of information "hops" from frequency to frequency over a wide band frequency spectrum. By doing so, noise-like signals are transmitted, and they are hard to be detected, intercepted, or demodulated.

Rowlands [3] proposed an orbit shift logistic map chaotic encryption scheme. In their algorithm, the key consists of three parameters: seed, settle, and offset. The seed is the initial value of the logistic map iteration sequence. The offset is the shift of the seed that will generate a new orbit, i.e., the next orbit starts its map iteration from initial value of seed plus offset. The settle is some number of iteration times of the map.

Now let’s explain a bit more about the settle and why we need it. In multiple orbit situations we want all orbits to be as dissimilar as possible. For one specific chaotic map, in order to have many orbits, a new orbit starts with an initial value that deviates from the preceding one by a very small offset. A chaotic system is very sensitive to its initial value (it is well known butterfly effect), but it is a long term behavior. So the two orbits almost overlap each other in the beginning of the sequence. This is undesirable behavior for a cryptographic application. In order to avoid this situation from happening, before we use chaotic orbits / sample points for cryptographic purpose we will let adjacent orbits iterate some number of times (a smaller offset may need more iterations) and make sure that they are different from each other.

Inspired by frequency hopping, in our stream cipher the pseudo random numbers – the carriers in the sense of communication – are generated by hopping through many chaotic orbits using a hopping pattern. Those orbits are generated from multiple chaotic systems / maps. The multi-map orbit hopping chaotic stream cipher is a symmetric algorithm, it is fast, strong, computationally efficient, and applicable to most stream cipher applications, including wireless transmissions.

The organization of the paper is as follows. In section 2 we explain the design diagram of the cipher and the chaotic map bank. Section 3 details the cipher implementation, key handling, subkey structure, and random number extraction method. In section 4 we give a concrete example of the cipher. Section 5 concludes the paper.

2 Cipher design

2.1 Block diagram

The system chooses $m$ maps $M_0, M_1, ..., M_{m-1}$ from the chaotic map bank and sets the order of the chosen maps to hop. A key is turned into $m$ subkeys after going through key handling process. For each individual map, $s$ orbits $S_0, S_1, ..., S_{s-1}$ are generated. And further, on each orbit, $n$
points \( N_0, N_1, ..., N_n \) are generated. Parameters \( s, n, \) and the hopping pattern are determined by the key.

For a given chaotic map, the second orbit is generated by increasing the initial seed of the first orbit by an offset. The third orbit is generated by increasing the initial seed of the second orbit by the same offset, and so on for additional orbits as needed.

### 2.2 Chaotic map bank

For the stream cipher, we are using a bank of chaotic maps whose parameters are properly tuned to make sure that all maps lead to chaotic. A map is chaotic [5] if it is sensitive to initial conditions, topologically mixing, and its periodic orbits are dense.

In our current system, we are using a group of logistic maps and Chebyshev maps. Both types have been studied and are well understood [4]. The logistic map is defined as

\[
x_{n+1} = rx_n(1 - x_n), \quad x_n \in (0, 1)
\]

where \( 0 \leq r \leq 4 \) [5]. And the Chebyshev map of degree \( k \) is

\[
x_{n+1} = \cos(2^k \cos^{-1} x_n), \quad -1 < x_n \leq 1, n = 1, 2, 3, ...
\]

### 3 Cipher implementation and key issues

#### 3.1 Implementation

The Key Handling turns a key into subkeys, the subkeys then control parameters of chaotic maps (seeds, offsets, #orbits, #samples, and HPSN). According to HPSN the Hopping Mechanism controls how the cipher hops between the orbits. The Generator consists of three nested loops. The outermost loop steps through all maps involved. For each map in the second loop, several orbits are produced. For each orbit in the innermost loop several points are sampled for producing random numbers.

#### 3.2 Key handling

The key handling is the means by which the key bits are turned into subkeys that the cipher can use. In this initial version we simply provide a much longer key and sequentially split it into #maps subkeys. In the later version, key handling will use a shorter key to generate all the subkeys needed for the system.

After a random-bit key is fed into the system, the key handling procedure splits it into #maps (8 for the experimental version) subkeys. These subkeys are used to set up all control parameters. We assume that, since it is a one time process, a slow coin-flipping method can be used to generate keys.
3.3 Subkey structure and key length

Each map needs a subkey to control its behavior for generating suitable orbits. A subkey has the structure, see Fig. 2.

The length of subkey is 56-bit (without HPSN). Here we explain the size of each component of the subkey: Seed, Offset, #Settles, #Orbits, and #Samples.

- Seed: the initial value (state) of the chaotic map, represented in 6 hex-decimals, i.e. 24 bits. E.g., (25fd08)$_h$ is the seed. It is (2489304)$_d$. Make it as decimal 0.002489304.

- Offset: the shift value added to the Seed in order to generate the next orbit. It is represented in 4 hex-decimals, i.e. 16 bits. The Offset is far less than the Seed, at least 100 times smaller.

- #Settles: the number of iterations in settling period for an orbit before it can be used to generate sample points. It varies from 30 to 285. The variation is represented in 8 bits.

- #Orbits: the number of orbits generated during each use of a map. It is between 4 and 19. It is represented in 1 hex-decimal, i.e. 4 bits.

- #Samples: the number of sample points taken from an orbit, i.e. how long to stay on a particular orbit. It varies from 4 to 19 and is represented in 1 hex-decimal, i.e. 4 bits.

- HPSN: hopping-pattern serial number is ignored in this version.

The total key length is determined by the number of subkeys, because each map needs one subkey, i.e. the total key length is determined by the number of maps involved. In our current settings the number of maps is 8. Then the total key length will be (in bits) 8 x 56 = 448 bits. So the key space for the 8-map system is 2^448.

The random number extraction from a chaotic sample point is described in section 3.4. After the random number is generated, we combine it with plaintext/ciphertext to get corresponding ciphertext/plaintext. For this version, we use exclusive OR (XOR) as the combination function.

3.4 Chaotic random number extraction

We extract the lowest digits from a specific chaotic orbit point $x_n$ by the following steps.

Step 1: Removes the decimal point from $x_n$ (i.e. $x_n$ is less than 1). E.g., 0.33461 → 33461, 0.9442345679457 → 9442345679457

Step 2: Makes the number 8 digits. If the original number is shorter than 8 digits, we add zeros at the end. If the original number is longer than 8 digits, we chop off the extra digits from the left side. E.g., 33461→33461000, 9442345679457 → 45679457.

Step 3: Gets the remainder from the 8 digits number by mod 256, it will be the generated pseudo random number. E.g., 33461000(mod 256) = 8, 45679457(mod 256) = 97.

The generated random numbers are between 0 and 255, inclusive. That’s the total number of the extended ASCII character set. This simplifies the process of combining (XOR) random numbers with plaintext/ciphertext for encryption/decryption.

3.5 Statistical test results

The pseudo random numbers generated by our stream cipher passed all 16/18 tests in NIST/DIEHARD Suite of statistical tests [7, 8] with satisfactory results.

4 An illustrating example

The system setup is as follows. The #maps = 8, and each map needs a 56 bit subkey, so we need 56 x 8 = 448 bits. That is it consists of 112 random numbers (0 to F in hexadecimal). Suppose those random numbers are generated by coin-flipping method. Here is a sample key: "bd144bf3a8e6977 ae62ef9717b0 8a716b4b9e534 371abc759565f8b fl85f15ee7887a e667f42200b92a b4690a87ed392 31d3639afe54f3".

Here are the eight chaotic maps. The coefficients of these dynamical system difference equations are well tuned so that the equations all lead to chaotic. For Logistic maps $0 \sim 4$, $x_n \in (0, 1)$, and for Chebyshev maps $4 \sim 7$, $x_n \in (-1, 1)$.
Table 1  Setup parameters for the eight maps

<table>
<thead>
<tr>
<th></th>
<th>Seed</th>
<th>Offset</th>
<th>#Settles</th>
<th>#Orbits</th>
<th>#Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map #0</td>
<td>0.0012391499</td>
<td>0.00001499</td>
<td>135</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Map #1</td>
<td>0.002010722</td>
<td>0.000061335</td>
<td>53</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>Map #2</td>
<td>0.009073003</td>
<td>0.000033977</td>
<td>259</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Map #3</td>
<td>0.003611367</td>
<td>0.00002287</td>
<td>125</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Map #4</td>
<td>0.0015828465</td>
<td>0.000024295</td>
<td>166</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Map #5</td>
<td>0.0015120372</td>
<td>0.00008704</td>
<td>30</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Map #6</td>
<td>0.001182337</td>
<td>0.000036734</td>
<td>241</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Map #7</td>
<td>0.003265379</td>
<td>0.000039678</td>
<td>114</td>
<td>19</td>
<td>7</td>
</tr>
</tbody>
</table>

0) \( x_{n+1} = 3.901x_n(1-x_n) \), 1) \( x_{n+1} = 3.931x_n(1-x_n) \),
2) \( x_{n+1} = 3.963x_n(1-x_n) \), 3) \( x_{n+1} = 4x_n(1-x_n) \),
4) \( x_{n+1} = \cos(2^1 \cos^{-1} x_n) \), 5) \( x_{n+1} = \cos(2^2 \cos^{-1} x_n) \),
6) \( x_{n+1} = \cos(2^3 \cos^{-1} x_n) \), 7) \( x_{n+1} = \cos(2^6 \cos^{-1} x_n) \).

After the key scheduling, the eight maps are setup with parameters shown in Table 1 above.

There are 45 orbits that come from four logistic maps (Map #0 – #3), and 47 orbits from the four Chebyshev maps (Map #4 – #7). We use different colors to distinguish orbits from different maps, Red for Map #0 / #4, Green for #1 / #5, Blue for #2 / #6, and Black for #3 / #7. After their settle periods, the first four samples of each orbit are depicted in Fig. 3. As shown here, those orbits are sufficiently different from each other. Furthermore we use our chaotic random number extraction method to obtain random numbers from those sample points.

![Figure 3](image)
(b) Chebyshev Maps

Fig. 4 shows frequencies of ASCII characters in plaintext (a) and ciphertext (b) for Mark Twain's novel "Huckleberry Finn" [6] of total length 597,299 characters (for comparison with [3], we chose the same novel). From Fig. 4(a), we see clustering around letters and peaking for space. In Fig. 4(b) we see fairly even flat distribution, i.e., in the ciphertext novel each of 256 ASCII characters occurs with nearly equal likelihood. However, in the histogram of the ciphertext in [3], there are four obvious clusters near ASCII values of 45, 100, 150 and 220. We see the significant improvement in terms of frequency distribution for every ASCII character.
5 Conclusions

In addition to chaotic features of mixing, unpredictability, and extreme sensitivity to initial seeds, using the multiple chaotic maps/orbits hopping mechanism, we spread out the pseudo random number base to a wide flat spread spectrum in terms of time and space. Our pseudo random numbers are similar to white noise. Therefore the distribution of those numbers is quite even, flattened, and further results have no clustering in the ciphertext as shown in Fig. 4(b).

The variable key length stream cipher that we have designed is a symmetric cryptosystem that makes use of a secret key to determine chaotic system parameters: number of maps, seeds, offsets, settles, orbits, and sample points. The chaotic maps we chose are computationally economic and fast. The current cryptosystem is easily implemented in software. We expect this cipher will be suitable for applications like wireless, cable, and optical fiber communications.

![Histograms for Character Frequency of “Huckleberry Finn”](image)

6 Acknowledgements

The authors would like to thank Prof. Michael Anshel for his encouragement and helpful comments. We also want to say special thanks to Prof. Kent D. Boklan for his review and corrections to the manuscript. Our thanks also give to Dr. Xiaowei Xu for valuable discussions.

7 References


