APPLICATIONS OF THE MULTI-MAP ORBIT HOPPING MECHANISM IN STREAM CIPHER DESIGNS

by

XIAOWEN ZHANG

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2007

UMI Number: 3283617

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for the
Graduate Faculty in Computer Science in satisfaction of the
dissertation requirement for the degree of Doctor of Philosophy.

Date	Michael Anshel Chair of Examining Committee
Date	Theodore Brown Executive officer
	Theodore Brown
	Victor Pan
	Benjamin Fine
	Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

APPLICATIONS OF THE MULTI-MAP ORBIT HOPPING MECHANISM IN STREAM CIPHER DESIGNS

by

Xiaowen Zhang

Advisor: Professor Michael Anshel

The objective of this thesis is to design, implement, statistically test and cryptanalyze several stream ciphers that are based on the multi-map orbit hopping mechanism – Mmohom.

We propose Mmohom as a generic method to design stream ciphers. We discuss its cryptographic properties of period, density functions, and conjugate permutations. The applications of the Mmohom result in three kinds of stream ciphers.

The first kind is Mmohocc (Multi-map orbit hopping chaotic cipher). Since its inception, chaos has been connected with cryptography due to its confusion and diffusion, which are the two important properties of a secure cipher. On top of these properties, we apply Mmohom to multiple chaotic systems in order to build Mmohocc, which greatly accelerates the chaos behavior and makes chaos more suitable for cryptographic purposes. The implementation details of the cipher are given. For the generated keystreams we get satisfactory statistical results from two batteries of the most popular and stringent statistical tests. Cryptanalysis shows that the cipher is resistant against the most known attacks.

The second kind is the Mmohom-LFSR-based cipher. Mmohom is used in conjunction with multiple LFSRs (Linear Feedback Shift Register) to destroy the linearity of the LFSRs. Cryptanalysis against algebraic attack is discussed.

The third one is Mmohoct (Multi-map orbit hopping cipher by T-functions). The T-function has maximal single cycle property, meanwhile it has some undesired algebraic patterns. We introduce the Mmohom to destroy the unwanted properties and build two experimental Mmohoct ciphers. Cryptanalysis and performance are discussed, too.

Acknowledgements

I would like to express my gratitude to the following people and institutions for supporting me in completing this thesis:

Thanks first and foremost to my mentor Prof. Michael Anshel, who has been inspiring, guiding and supervising this work from the very beginning. He provided tremendous invaluable guideline, discussions and encouragements, meanwhile gave me a large degree of freedom in my research. He created and organized the Friday afternoon cryptography seminar at the Graduate Center, which keeps us to the frontier of the field as well as brings us new technologies and ideas.

A special thanks to Prof. Ted Brown, who as my committee member has been giving a great deal of helpful advice on my research and thesis; as Executive Officer in the PhD program he supported me with various fellowships during my study; as the former Chairperson of Computer Science Department at Queens College he offered my first adjunct position and made my joining to the PhD program possible. Without his generous support, I would not have had this unforgettable learning experience.

Many thanks to Prof. Jerry Waxman who is always there whenever I need help in both of my study and everyday life. Also, as my teaching supervisor, he taught me not only the best way to teach, but also the way to be friend with and to learn from students, which have been making my teaching experience so joyful and fruitful.

I greatly appreciate the supports and efforts of my committee members Prof. Victor Pan (Lehman College) and Prof. Benjamin Fine (Fairfield University) who have spent their precious time on reviewing this thesis and giving very important constructive suggestions.

Thanks to Computer Science Department of Queens College and Electrical Engineering Department of NYC College of Technology for offering me adjunct lecturer opportunities, which were the essential financial support of my study. I am grateful to the faculty and staff for their help and support.

Thanks to Umbanet Inc. for offering me the great opportunity to work on an NIST/ATP project to learn cryptography in practice. It was a great pleasure and experience to work for three years with Umbanet staff: Michelle Baker, Kent Boklan, Dmitry Bormotov, Amy Cai, Lin Chang, Yuri Fukuda, Jimmy Hoa, and Cliff Lee.

Thanks to Dr. Yiqun Lisa Yin (Independent Security Consultant, Connecticut) and Dr. Scott Contini (Macquarie University, Australia) for reviewing the initial Mmohocc cipher and providing valuable suggestions and references on eSTREAM and T-functions.

Thanks to my fellow classmates and friends: Qinghai Gao, Dammika Kahanda, Andis Kwan, Li Shu, Joseph Svitak, Ke Tang, Xinzhou Wei, Xiaowei Xu, whose accompany and sharing have been a constant encouragement and support towards the finish line of this thesis. Thanks to Dr. Xiangdong Li and Dr. Lin Leung for leading our team work to publication.

Jetta and Myron Gordon, who like my parents, check up on me and my progress all the time. Tamar Gordon and Scott Christianson, like my sister and brother, give generous encouragement and support. Tamar also read some chapters of the thesis to correct my writing.

To my wife Vicky Hong Wang and my parents for everything, including the tolerance on me for being a student for such a long time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter gives the research motivation and a brief overview of the thesis.

1.1 Research motivation

A stream cipher encrypts bits/bytes/words of plaintext one at a time by using a symmetric algorithm (a.k.a. secret-key algorithm), which generally requires a key to be shared and simultaneously kept secret among the legitimate communicating parties. The encryption transformation of a stream cipher varies with time according to message generating rate. In contrast, a block cipher is generally used to encrypt a block of characters of plaintext using a fixed encryption rate.

There are strong needs for stream ciphers in the following scenarios [7, 101]:

- Radio communications where zero error propagation is highly desirable.

- Very high speed multi-Gigabit-per-second communication links (such as routers, fiber-optics), where stream ciphers can be implemented completely in software, or combination of software and hardware.
- Efficient and constrained devices with very small battery powered e.g. RFID tags, smart cards (8-bit processors).
- Multimedia, Movie-on-Demand, and other real-time situations.
- Lightweight encryption applications (such as Bluetooth, sensor networks).

Before 2001, the only generic standards for stream ciphers were the block cipher modes of operation (OFB – output feedback mode, CFB – ciphertext feedback mode and CTR – counter mode) [76]. Today there are application-specific standards in use, such as A5/x for GSM (the Global System for Mobile communication), RC4 for IEEE 802.11 WEP and SSL, CTR-mode Kasumi block cipher for GSM/3GPP (the 3rd Generation Partnership Project), and E0 for Bluetooth. A5/1 is reverse engineered and broken, RC4 is vulnerable due to key scheduling issues, E0 is cryptanalyzed by fast correlation attack [65].

The ECRYPT Stream Cipher Project (abbreviated as eSTREAM) is a multi-year (from 2004 to 2008) effort to identify new stream ciphers potentially suitable for widespread adoption. Till the deadline of April 2005 more than 34 different stream cipher proposals were submitted in two different performance profiles (stream ciphers for software and hardware applications) [47]. However more than half of them have demonstrated weaknesses of various kinds.

By using the multi-map orbit hopping mechanism, this thesis attempts to design

stream ciphers that, hopefully, do not suffer the known weaknesses and are resistant to the known attacks.

1.2 Overview of the thesis

The rest of the thesis is organized in the following way:

In **Chapter 2** we present some basic notions of stream cipher design and cryptanalysis. We give informal definitions for both synchronous stream cipher and self-synchronous stream cipher. We briefly introduce various stream ciphers, including several LFSR-based stream ciphers, provably secure PRNGs (BBS, QUAD), OFB- and CTR-mode block ciphers (used as stream ciphers), as well as the current ongoing eSTREAM project which represents the state of the art of stream ciphers. At the end we review some of commonly known cryptanalysis techniques used to break stream ciphers, and performance comparison guideline for newly designed stream ciphers.

In **Chapter 3** we introduce the Mmohom (Multi-map orbit hopping mechanism), which is derived conceptually from frequency hopping mechanism widely used in wireless spread spectrum communication. We discuss its mathematical advantages in terms of period property, density functions, and conjugate permutations.

In **Chapter 4** we use chaotic maps in Mmohom to develop a stream cipher Mmohocc (Multi-map orbit hopping chaotic cipher). After the introduction of the properties of dynamical chaos systems and the relationship between chaos and cryptography, we give the specifications on the cipher design, the selection of chaotic map bank, key handling/subkey structure, and the implementation details. To demonstrate the

application we use the cipher to encrypt/decrypt a novel, subsequently the histogram analysis of the plaintext and ciphertext is given.

In **Chapter 5** we deal with the improvement of Mmohocc and the randomness evaluation of the keystreams. For better performance and higher security we enhance the cipher in three aspects: chaotic map bank, hopping patterns, and random number extraction methods. With respect to the keystream's randomness we conduct two batteries of the most stringent and well known statistical tests, namely the NIST Suite (which consists of 16 tests) and the DIEHARD Suite (which contains 18 tests). All test results are successful.

In **Chapter 6** for the improved Mmohocc we perform the security analysis in terms of period, auto-correlation and cross-correlation, and spatiotemporal mixture properties of the keystreams. We cryptanalyze the cipher and conclude that it is resistant against the related-key-IV, Time-Memory-Data tradeoff, and algebraic attacks. The performance comparison shows that the encryption and decryption speed of the Mmohocc cipher is as fast as RC4.

In **Chapter 7** we introduce the Mmohom-LFSR-based stream cipher that is intended to defeat the algebraic attacks, a well known fast attack against LFSR-based stream ciphers. We list features of the design and give a schematic diagram implementation of the cipher. Our cryptanalysis shows that it is strong against algebraic attacks.

In **Chapter 8** we address the Mmohoct (Multi-map orbit hopping cipher by T-functions) ciphers, which are the result of Mmohom applications in T-function-based stream ciphers. T-function, recently proposed by Klimov and Shamir as a new building block for cipher constructions, has nice cryptographic properties. Meanwhile it

has some unwanted algebraic structures. We use the Mmohom mechanism to destroy the structures. Two kinds of Mmohoct are implemented: (1) we design Mmohoct by directly replacing Mmohocc's chaotic maps with T-functions; (2) we design Mmohoct by using extended TSC-like and quadratic format T-functions as the map bank. The performance of the above two ciphers is evaluated and compared with RC4 and AES-CTR.

In **Chapter 9** we conclude the thesis, give open problems and future research direction.

Chapter 2

Stream Cipher Basics

This chapter covers some background material concerning stream ciphers. We review various stream cipher designs and some cryptanalysis methods.

2.1 Stream ciphers

Stream cipher is one of the workhorses for bulk data encryptions. It is a symmetric cipher in which plaintext units are encrypted one at a time with a time-varying transformation. In practice, a typical unit could be a single bit, byte, or word. By contrast, block ciphers (such as, DES, 3DES, AES, RC-5, IDEA, Blowfish) operate on large blocks of plaintext units with a fixed transformation. This distinction is not always clear-cut; for example, some modes of operation use a block cipher primitive in such a way that it then acts effectively as a stream cipher. Stream ciphers typically run at a higher speed than block ciphers and have lower hardware complexity [115].

2.1.1 Basic knowledge

Stream ciphers are a very important class of encryption algorithms. They encrypt individual characters (usually binary bits) of a plaintext message one at a time, using an encryption transformation which varies with time [74]. All those algorithms do is to "stretch" or "expand" a short random sequence, called the **secret key**, into a long pseudorandom sequence, called the **keystream**, then use the keystream to encrypt/decrypt plaintext/ciphertext message into ciphertext/plaintext message.

More precisely, in a stream cipher a sequence of plaintext units, $p_0p_1 \ldots$, is encrypted into a sequence of ciphertext units $c_0c_1 \ldots$ as follows: a pseudorandom keystream sequence $z_0z_1 \ldots$ is produced by a finite state machine whose initial state is determined by a **secret key**. The i-th keystream unit only depends on the secret key and on the (i-1) previous plaintext units. Then, the i-th ciphertext unit is obtained by combining the i-th plaintext unit with the i-th keystream unit [20, 114].

Steam ciphers are classified into two types: **synchronous stream ciphers** and **self-synchronizing stream ciphers**. A synchronous stream cipher is one in which the keystream is generated independently (depends on key alone) of the plaintext message and of the ciphertext message. More specifically, a synchronous stream cipher can be described by the following three equations [74]:

$$\sigma_{i+1} = f(\sigma_i, k), \quad z_i = g(\sigma_i, k), \quad c_i = h(z_i, p_i).$$

f is the **state transition function** (or **state update function**, or **iteration function**), which uses the current state σ_i and the key k to produce a new state σ_{i+1}. g is the **keystream output function** (usually called in short as *output function*),

which given the same two inputs produces the keystream z_i as output. And h is the **combining function** that combines the plaintext p_i with the keystream z_i to produce the ciphertext c_i. Usually the function h is the XOR (represented by $\oplus$) and this kind of ciphers is called binary additive stream ciphers, see Figure 2.1.

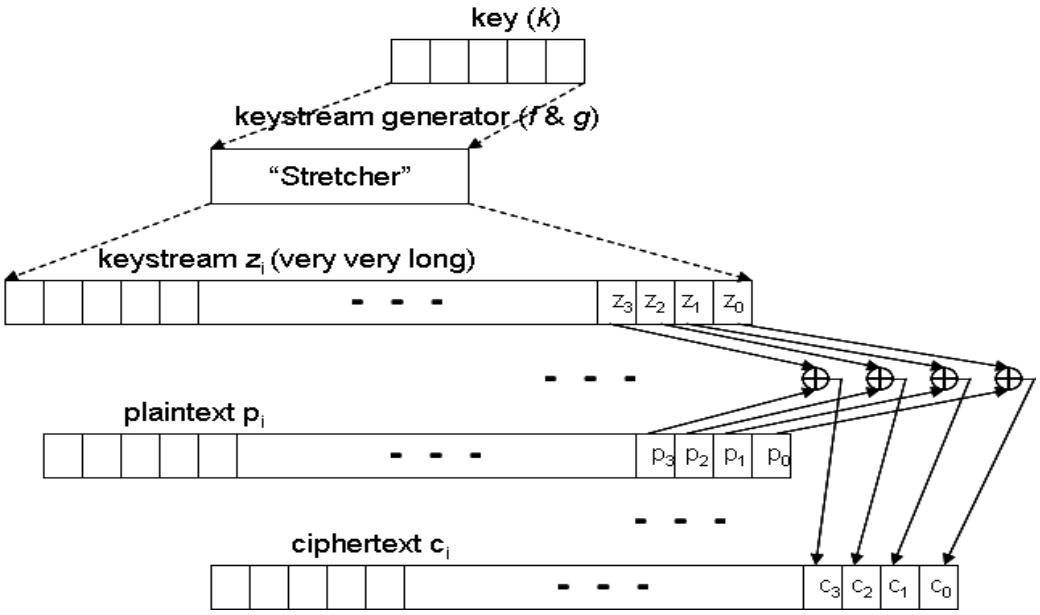


Figure 2.1: Diagram of synchronous stream cipher.

A self-synchronizing or asynchronous stream cipher is one in which the keystream is generated as a function of the key and a fixed number of previous ciphertext units [74]. The encryption can be expressed by equations $\sigma_i = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1})$, $z_i = g(\sigma_i, k)$, $c_i = h(z_i, p_i)$, where $\sigma_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$ is the initial state, and $\sigma_1 = (c_{1-t}, c_{1-t+1}, \ldots, c_0)$ is used for generating z_1, and so on so forth. This is the general t-digit cipher feedback mode. Again, if h is XOR ($\oplus$) the encryption procedure is depicted in Figure 2.2. The discussions at the ECRYPT State of the Art of Stream Ciphers workshop suggested that there was little demand for the self-synchronizing

stream cipher [22], therefore from hereafter, the term "stream cipher" is a synonym for "synchronous stream cipher", unless otherwise specified explicitly.

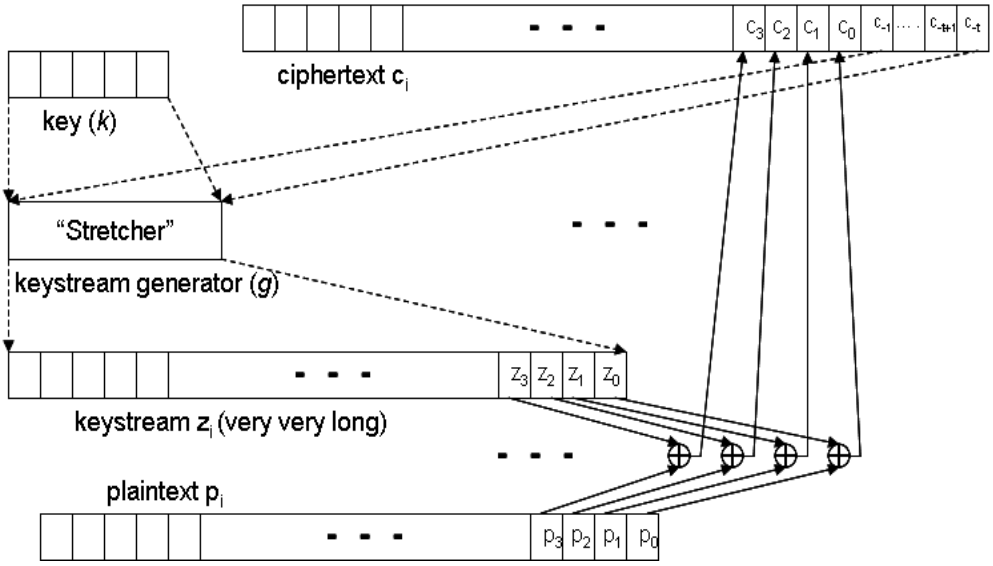


Figure 2.2: Diagram of self-synchronous stream cipher.

2.1.2 Stream cipher's yesterday, today and tomorrow

Cipher machines [115] used pre-WWII and during WWII, such as Hebern rotor machines, Typex (British), SIGABA (American), NEMA (Swiss), Purple (Japanese), Enigma (German), can be categorized to stream ciphers in the sense that they encrypt/decrypt one letter at a time. The name Enigma is made famous by its use in World War II and the successful break of the cipher by Allied codebreakers.

The Enigma was a portable cipher machine used to encrypt and decrypt secret messages. It was a family of related electro-mechanical rotor machines [115]. By continuously changing the substitution alphabet, it implemented a polyalphabetic

substitution cipher with a long period. With single-notched rotors, the period of the machine was 16,900 ($26 \times 25 \times 26$).

The perfectly secure and unbreakable stream cipher is the Vernam cipher, a.k.a. one-time pad (OTP) [48]. Its assumption is that the pad is never reused and unpredictable random numbers are generated from truly random physical sources, like fair coin-flips, radioactive decay, thermal noise, shot noise and photon's transmission upon a semi-transparent mirror. It was used by Russian spies and the Washington-Moscow "hot line" [100]. Although the original OTP is not practically useful today because of the difficulty of key distribution/management, it is seen as one of the cornerstones [103] of cryptography.

Linear feedback shift registers (LFSRs) are the basic components for many stream ciphers because they can produce sequences with large cycles and good statistical properties, have good mathematical structures and can be readily analyzed using algebraic techniques [74, 51]. A LFSR of length n consists of n bits (or n delay elements) $s_1, \ldots, s_n$ whose input bit is a linear feedback function exclusive-or (XOR) of the other bits in the register (see Figure 7.1).

The use of LFSR for a stream cipher design is regarded as a typical classical approach. However, you do not use LFSRs by themselves as stream ciphers since their linear properties make them insecure. Their good mathematical structure can often be used to cryptanalyze them. Ian Cassells said that [100] "cryptography is a mixture of mathematics and muddle, and without the muddle the mathematics can be used against you." In conjunction with nonlinear Boolean functions, there are three general methodologies for destroying the linearities of LFSRs, by using [74]:

1. a nonlinear combination function on outputs of several LFSRs (Fig. 2.3 (a));

2. a nonlinear filter function on the contents of a single LFSR (Fig. 2.3 (b));

3. the output of one/more LFSRs to control the clock of one/more other LFSRs (such as Beth-Piper generator).

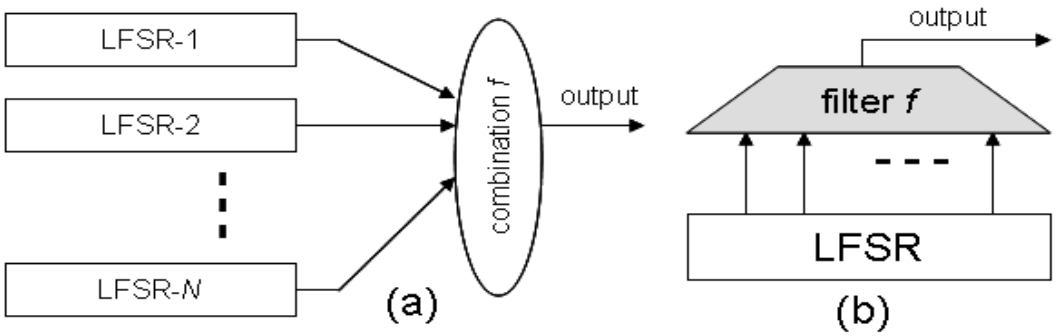


Figure 2.3: (a) Combination generator and (b) filter generator.

Here are several examples of the stream ciphers designed by using LFSRs.

The Geffe generator uses three LFSRs, which are combined in a nonlinear manner [100]: two LFSRs are inputs into a multiplexer, and the third LFSR controls the output of the multiplexer.

The Beth-Piper Stop-and-Go generator [11] uses the output of one LFSR to control the clock of another LFSR.

The Self-Decimated generators control their own clock. R. Rueppel [98] proposed one as follows: using only one LFSR, when its output is 0, the LFSR is clocked d times; and when its output is 1, the LFSR is clocked k times.

The Shrinking generator [24] uses two LFSRs: LFSR-1 and LFSR-2. Clock both of them. If the output of LFSR-1 is 1, then the output of the generator is LFSR-2. If 0 is the output of LFSR-1, discard the two bits, clock both LFSRs, and try again.

A5/1 [115] is the stream cipher used to encrypt the wireless link between the mobile phones and the base stations in GSM (Global System for Mobile Communications – a standard for mobile phones). It consists of three LFSRs of size 19, 22, and 23 with irregular clocking. The output is the XOR of the three LFSRs.

All of these LFSR-based stream ciphers have inherent weaknesses of LFSRs in the final output, and are broken [118, 119, 50, 9, 14, 38, 32].

LFSRs are extremely fast in hardware implementation but relatively slow in software implementation. This has led to several stream cipher proposals particularly for fast software implementation. From Fast Software Encryption (FSE) Workshop, SEAL and RC4 are two promising software oriented stream ciphers.

SEAL (Software-optimized Encryption Algorithm), a proprietary stream cipher by IBM [93, 94], is a length-increasing pseudorandom function which maps a 32-bit sequence number to an L-bit keystream, it is optimized for 32-bit processors. SEAL preprocesses the key into a set of tables which are used to speed up encryption and decryption.

RC4, a remarkably simple stream cipher by Rivest [92], is the most widely-used software stream cipher and is used in popular protocols such as SSL (Secure Sockets Layer) and WEP (Wired Equivalent Privacy – IEEE 802.11). But it falls short of the high standards of security, many papers [40, 34, 66, 86, 87] have shown its weakness and insecurity.

Bluetooth uses E0 stream cipher for encrypting packets. (Bluetooth is a radio standard and protocol for short-range wireless connectivity, specified by the Bluetooth Special Interest Group.)

Other well-known stream ciphers include word-oriented stream ciphers (suitable for software implementation, such as SNOW, SOBER, SCREAM), stream ciphers modes of operation of block ciphers (e.g., cipher feedback, output feedback mode of Triple DES or AES).

Provably secure pseudorandom number generators (PRNGs) have mathematically provable security, such as Blum-Micali [16] and Blum-Blum-Shub [15] (too slow for practical encryption use), hyper-encryption cryptosystem [90] (has not been popularly used), and recently QUAD [10] (too early to say anything).

The eSTREAM [33] is an ongoing ECRYPT Stream Cipher Project. This is a multi-year effort to identify new stream ciphers that might become suitable for widespread adoption. Now it is in the Phase–2 (Start from August 1st, 2006), there are more than 30 candidates in the list: DRAGON, ABC, HC-256, CryptMT, MICKEY, LEX, Rabbit, SNOW, SOBER-t, and more.

Stream ciphers have advantages which make them suitable for a hardware scheme with extremely small footprint and software scheme with extremely high speed applications [101]. Moreover, synchronous stream ciphers are not affected by error-propagation.

2.1.3 Stream ciphers out of block ciphers

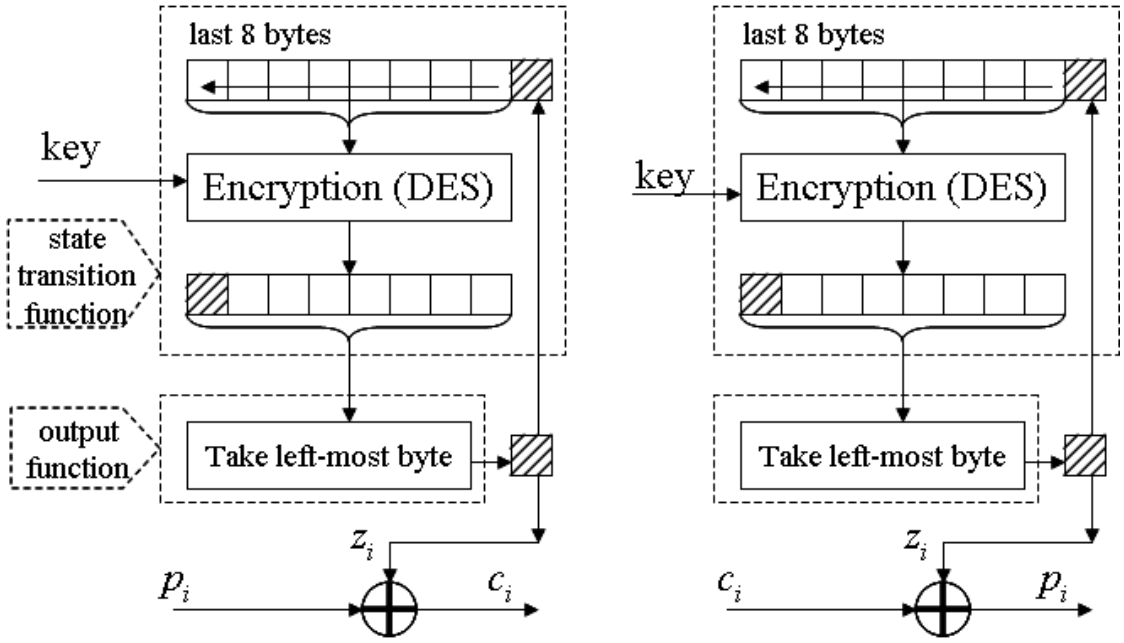
Output-feedback (OFB) and counter (CTR) modes of operation can make a block cipher become a stream cipher [100, 109]. Stream cipher made from block cipher in OFB mode uses complex state transition function and simple output function; in contrast, stream cipher made from block cipher in CTR mode uses a simple counter as state transition function and a complicated output function that is dependent on the key.

Figure 2.4 is an 8-bit OFB stream cipher, the plaintext is broken into 8-bit pieces $p = [p_0, p_1, p_2, \ldots]$, where each p_j has 8 bits, rather than the entire block size (64-bit for DES, 128-bit for AES). The encryption and decryption algorithms look like this:

$$c_i = p_i \oplus LM8(S_i), \quad S_i = E_K(S_{i-1}); \quad p_i = c_i \oplus LM8(S_i), \quad S_i = E_K(S_{i-1}).$$

S_i is the state, which is independent of either the plaintext or the ciphertext, i.e. OFB stream cipher is a synchronous stream cipher. Here $z_i = LM8(S_i)$, "LM8" means "leftmost 8-bit."

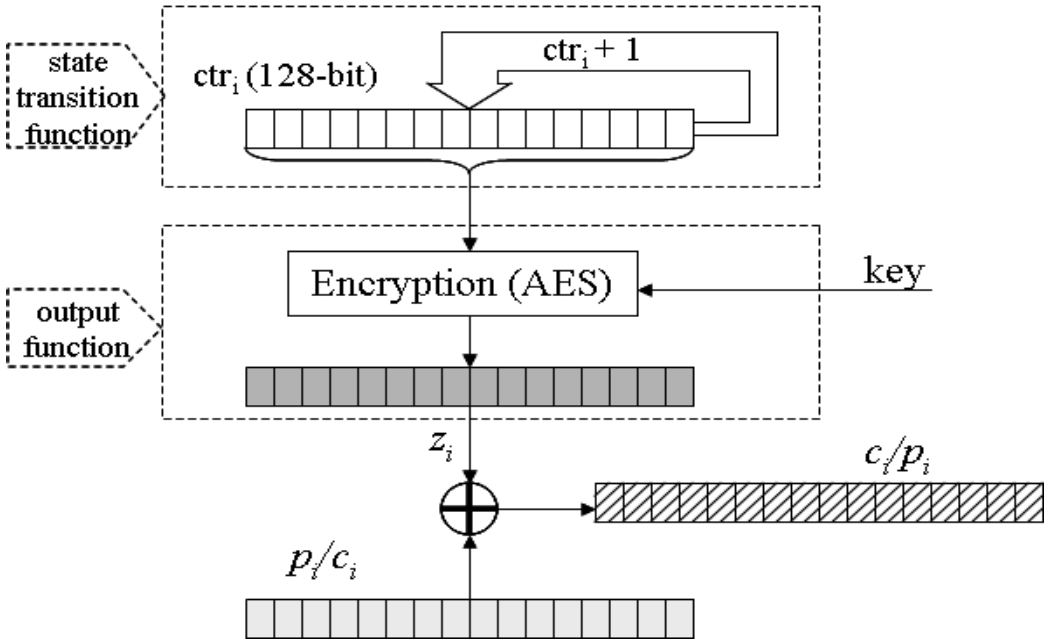
CTR mode stream cipher (see Figure 2.5) uses a counter $ctr_i = ctr_i + 1 \ (mod\ N)$ (N is the spate space) as the input to the algorithm. The plaintext is broken into 128-bit pieces $p = [p_0, p_1, p_2, \ldots]$, where each p_j has 128 bits. After each block encryption, the counter simply increments by one. To encrypt, you start with the plaintext p, a key and a counter ctr, ciphertext c is the XOR of p and the first $|p|$ bits of $AES_E(ctr)||AES_E(ctr+1)||AES_E(ctr+2)||\ldots$ (where $|p|$ gives the size of the plaintext, $AES_E(ctr+j)$ is AES Encryption algorithm, "||" stands for concatenation). To decrypt ciphertext c you compute the plaintext p by XOR'ing the c with the first $|c|$ bits of the pad $AES_E(ctr)||AES_E(ctr+1)||AES_E(ctr+2)||\ldots$ [64, 115].



Left side – encryption algorithm, right side – decryption algorithm.

Figure 2.4: 8-bit output feedback mode stream cipher.

In OFB mode a keystream XORs the text (plaintext or ciphertext). This keystream will eventually repeat. Also we should make sure that the cipher does not repeat with the same key. When the feedback size equals the block size n, the average cycle length is $2^n - 1$ (for DES $n = 64$); when the feedback size is less than the block size, the average cycle length drops to around $2^{n/2}$ [100]. In OFB and CTR mode stream ciphers, both encryption and decryption depend only on the block cipher encryption algorithm E (e.g. AES_E, DES_E) – neither depends on the block cipher decryption algorithm D. So D does not need to be implemented. This makes implementations simpler, and may result in a significant throughput increase in hardware. Other advantages (software/hardware efficiency, preprocessing, random-access) and disadvantages (no integrity, etc.) of CTR mode stream ciphers can be found in [64].



Input stream is chopped into 128-bit chunks in AES-CTR stream cipher. State transition function is a simple counter, but the output function of the stream cipher is AES encryption algorithm.

Figure 2.5: Counter mode stream cipher diagram.

2.2 Cryptanalysis of stream ciphers

In this section we review some of the commonly known cryptanalysis techniques used to break stream ciphers: key recovery attack, correlation attacks, distinguishing attacks, algebraic attacks. The assumption is that an attacker knows both the ciphertext and some of the plaintext (a.k.a. known text attack).

2.2.1 Correlation attacks

The correlation attack, proposed by Siegenthaler, is used to against combination (combining several output sequences in some nonlinear way) generators [104, 105]. The basic idea is to identify some correlation between the output of the generator and the output of one of its internal pieces. By observing the output sequence, you can obtain information about that internal output. Using that information and other correlations, collect information about the other internal outputs until the entire generator is broken. This is a divide-and-conquer attack [100]. The attack applies when a part of the internal state is updated independently from the other ones and has a considerable size. It aims at recovering one of the parts of the initial state (so called the target state).

This attack has been greatly improved [72, 73] when the target part of the internal state is updated linearly. In this case, efficient error-correcting decoding can be used in order to (partially) recover the initial state of the generator. Other fast correlation attacks have been successfully applied to a number of LFSR-based keystream generators [117, 42, 75, 120].

2.2.2 Algebraic attacks

Algebraic attacks are a new class of attacks against stream ciphers, they are known to be faster and more efficient against certain ciphers (especially LFSR-based) than all other kinds of attacks [5, 4, 25, 26, 27, 28, 115]. A stream cipher based on the multiple LFSRs with non-linear output function F is particularly vulnerable to this kind of attacks, i.e., the output keystream $z_t = F(s_t)$ (where s_t is the state in time

t). The basic idea is [27, 26, 21]: given everything about the combiner and some keystream bits z_t, an adversary recovers the initial state S_0 (the key) by solving an overdefined system of multivariate algebraic equations (see 7.3 for more details). The drawback is that it requires the knowledge of a huge amount of keystream bits z_t.

2.2.3 Other attacks

Key recovery attacks [115] are the strongest attack against stream ciphers, in which attacker attempts to retrieve the secret key given part of keystream z. Having the secret key in hand, the attacker can produce the unknown parts of keystream and decrypt the rest of the message.

Distinguishing attacks try to distinguish observed keystream from a pure random sequence and to make prediction about future portions of keystream out of the known keystream segment. Distinguishing attacks are often weaker than key recovery attacks. However, they may still be a threat if attackers can deduce information on unknown plaintext based on some known plaintext, e.g. if period of keystream sequence is small. They are applicable to general stream ciphers, such as linear attacks [39], low diffusion attacks [23].

Still there are other attacks, such as **Time-Memory-Data (TMD) tradeoff** attacks [12, 13], **inversion** attacks on nonlinear filter generators [41], **free binary decision diagram (FBDD)** attacks [59], and more.

2.3 Performance comparison

Performance of newly designed stream ciphers is usually compared with popular RC4 and AES-CTR stream ciphers under the guideline of eSTREAM software performance [84].

Chapter 3

The Multi-Map Orbit Hopping Mechanism

In this chapter we introduce a mixing method – multi-map orbit hopping mechanism (abbreviated as Mmohom) which can be used as a building block in cipher designs.

3.1 Frequency hopping

In spread spectrum communication the bandwidth W for the transmission of digital information is much bigger than the information rate R (both of them are represented in bits/second). The bandwidth expansion factor $Be = W/R$ is much bigger than one. There are two types of spread spectrum systems: direct sequence(DS) and frequency hopping (FH). A spread-spectrum transmission offers three major advantages over a fixed-frequency transmission [91, 115]:

- Suppressing narrowband interference due to jamming and self interference due to multipath propagation.
- Gaining message privacy since spread spectrum transmissions are difficult to intercept.
- Sharing a frequency band with many types of conventional transmissions with minimal interference, since spread spectrum signals are transmitted at lower power level of background noise.

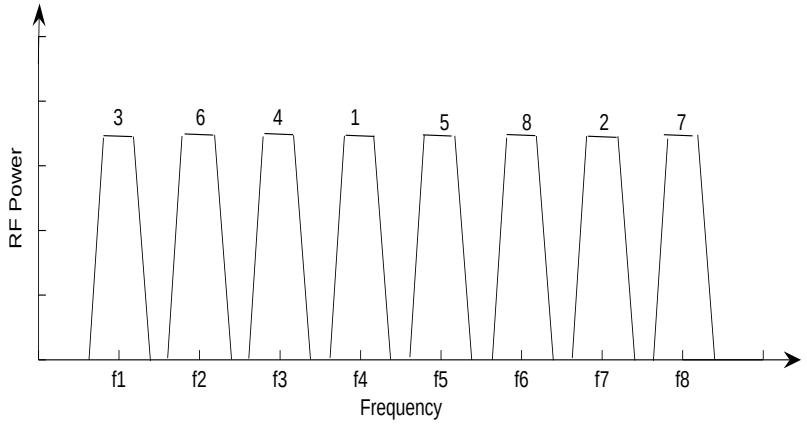


Figure 3.1: Frequency hopping pattern.

FH technique uses spread spectrum by rapidly changing a carrier among many frequencies, a carrier's frequency is selected by using a pseudorandom sequence (a.k.a. hopping pattern) known to both transmitter and receiver. Figure 3.1 shows an example of a random frequency hopping pattern for eight frequencies. The numbers on top of each waveform are the time serial, the set of corresponding frequencies to the

time serial from 1 to 8 on horizontal coordinate consists hopping pattern: {f4, f7, f1, f3, f5, f2, f8, f6}.

There are slow and fast frequency hopping systems depending on the rate at which hopping takes place. Most of FH systems are slow frequency hopping in which multiple data symbols are transmitted during one hop. In contrast, in fast frequency hopping one data symbol is transmitted through multiple hops.

3.2 The multi-map orbit hopping mechanism

After explaining some terms, we introduce the multi-map orbit hopping mechanism.

3.2.1 Terminology

The term **map** is a synonym for *function* and it pairs each of its inputs with exactly one output [115]. Let $\mathbb{A}$ be the input set (i.e. *domain*) and $\mathbb{B}$ be the output set (i.e. *range*). A map f is defined as $y = f(x)$, it maps each element $x \in \mathbb{A}$ to exactly one element $y \in \mathbb{B}$. We can denote the map as $f : \mathbb{A} \to \mathbb{B}$. For example, input and output $x, y \in \mathbb{N}$, $y = f_1(x) = x^3$ is a map that maps each natural number x to another natural number x^3, e.g., $1 \to 1$, $2 \to 8$, $3 \to 27$, ...

Continue with the above example, we view the map $f_1(x) = x^3$ as one of the transition maps in a state machine $\langle \mathbb{N}, \mathbb{F} \rangle$, here $\mathbb{N}$ is the set of all states – natural numbers, and $\mathbb{F}$ is the set of all state transition maps. A map $f_i(x) = x^2$ transits a state $x_i = 5$ to next state $x_{i+1} = 25$, and map $f_j(x) = x + x^2$ transits a state $x_j = 2$ to $x_{j+1} = 6$, see Figure 3.2.

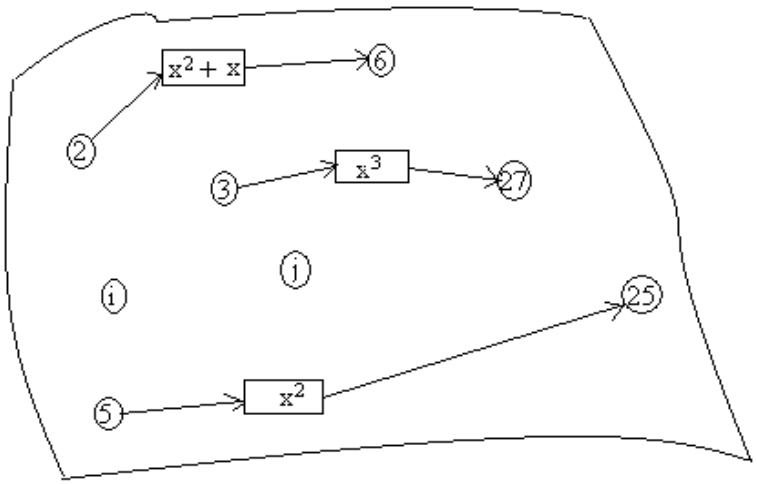


Figure 3.2: Diagram of state machine and map.

The term **Orbit** (a.k.a. trajectory) refers to the ordered set of intermediate states assumed by a dynamical system/map as a result of time evolution [115]. An orbit is generated by repeatedly evolving (iterating or calling) the dynamical system/map a certain number of times. An orbit is determined by a map, a ***seed*** (also called start point), and the number of times to evolve (you can get the end point). Here we give a definition of orbit in state machine $\langle \mathbb{N}, \mathbb{F} \rangle$ from [31], provided that all states are natural numbers.

Definition 3.1. ***Orbit****: Given $x_0 \in \mathbb{N}$ and a map f, we define the orbit of x_0 under f to be the sequence of points x_0, $x_1 = f(x_0)$, $x_2 = f^2(x_0)$, ..., $x_n = f^n(x_0)$. The point x_0 is the seed of the orbit, n is the number of times to iterate. Here f^n means iterating f n times.*

For example if the map is $f(x) = (x + x^2) mod\ 1023$, and the number of times to iterate $n = 10$, then we have an orbit of $x_0 = 1$ depicted in Figure 3.3.

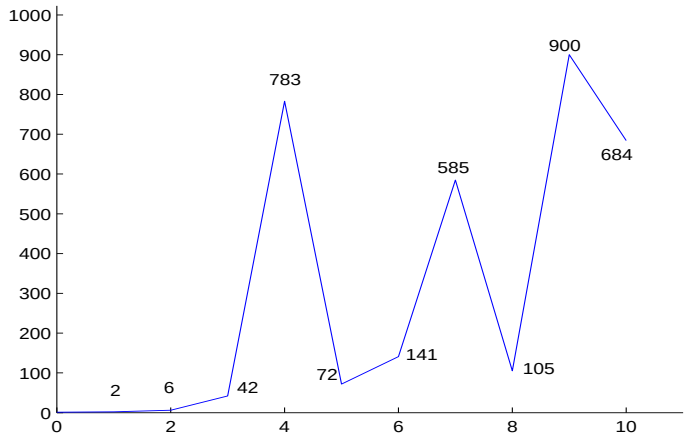


Figure 3.3: Diagram of an orbit from map $f(x) = (x + x^2) mod\ 1023$.

With the same map multiple orbits can be produced by using different seeds. We can obtain another orbit by adding an ***offset*** ϵ to the initial *seed* x_0. For the above example Figure 3.3, if an *offset* $\epsilon = 3$ we have the second orbit of $x_0 + \epsilon = 1 + 3 = 4$ by iterating the map 10 times: $4 \rightarrow 20 \rightarrow 420 \rightarrow 864 \rightarrow 570 \rightarrow 156 \rightarrow 963 \rightarrow 471 \rightarrow 321 \rightarrow 1023 \rightarrow 39$. The third orbit of $x_0 + 2 \times \epsilon = 1 + 2 \times 3 = 7$ is $7 \rightarrow 56 \rightarrow 123 \rightarrow 930 \rightarrow 372 \rightarrow 651 \rightarrow 930 \rightarrow 372 \rightarrow 651 \rightarrow 930 \rightarrow 372$, and the fourth orbit of $x_0 + 3 \times \epsilon = 1 + 3 \times 3 = 10$ is $10 \rightarrow 110 \rightarrow 957 \rightarrow 198 \rightarrow 528 \rightarrow 33 \rightarrow 99 \rightarrow 693 \rightarrow 132 \rightarrow 165 \rightarrow 792$, and so on so forth. Those orbits can be indexed as *Orbit*-0, *Orbit*-1, *Orbit*-2, *Orbit*-3, ... The number after the "dash" is an **orbit index**, it represents the number of offsets from the initial seed x_0 that orbit starts off. Those indices are used in orbit hopping patterns. The topic is elaborated in the next subsection.

3.2.2 The mechanism

Inspired by frequency hopping spread spectrum communication system we develop an orbit hopping mechanism (depicted in Figure 3.4) for further mixing and muddling the points sampled from multiple orbits of multiple maps. It consists of a hopping pattern generator, an orbit control switch, and multiple orbits. Instead of hopping around different frequencies in FH system, here the mechanism hops among multiple orbits.

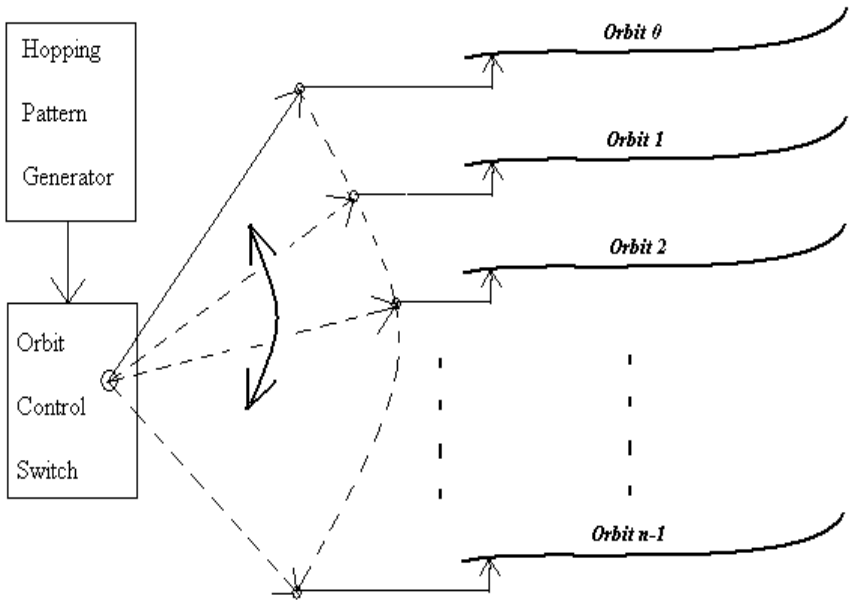


Figure 3.4: Diagram of orbit hopping mechanism.

Orbit hopping pattern, like the frequency hopping pattern in FH system, is a pseudorandom sequence generated by the hopping pattern generator. If the total number of orbits is n, then orbit hopping pattern contains n orbit indices from 0 to $n-1$. For instance if we have 10 orbits *orbit*-0 $\sim$ *orbit*-9 from a map f, a possible

hopping pattern could be a sequence {4, 7, 9, 1, 3, 5, 0, 2, 8, 6}.

The orbit hopping pattern is used by the orbit control switch to determine the order/way that the system uses multiple orbits. Use the above orbit hopping pattern for a 10-orbit mechanism we know that the switch will connect *orbit*-4 first, then *orbit*-7, *orbit*-9, ..., and *orbit*-6 in this order.

How long the switch stays connected with one orbit is determined by a factor called ***#samples*** – the number of points to be sampled from the orbit, which is an analogous concept to the signaling interval in FH system. In the orbit hopping mechanism we specify one *#samples* for each map, i.e., *#samples* is the same for any orbit coming from the same map. For another map the *#samples* may change.

The n orbits in Figure 3.4 are generated by multiple maps, say k maps: $M_0, M_1, \ldots, M_{k-1}$. Each map contributes different ***#orbits*** – number of orbits towards the n orbits. If map M_0 gives birth to n_0 orbits, map M_1 gives n_1 orbits, ... , map M_{k-1} gives n_{k-1} orbits, then $\sum_{i=0}^{k-1} n_i = n$. For demonstration purpose orbit hopping only happens among orbits which come from one map. Of course we could make the mechanism hop among the entire n orbits which come from k maps. Similarly we could make a map hopping mechanism hop among multiple maps in the similar way as in orbit hopping mechanism. We leave these topics for future investigation.

We will specify the parameters *seed*, *offset*, *#samples*, *#orbits* later.

3.2.3 The possible maps

All maps used in the Mmohom should have good cryptographic properties and be well-studied. Here we'll briefly introduce some of them.

Chaotic maps

A chaotic map is a discrete map (difference equation) which generates chaotic sequence. A one-dimensional map has the format of $x_{n+1} = f(x_n)$, and a two-dimensional map has the format of $(x_{n+1}, y_{n+1}) = f(x_n, y_n)$. Here are some chaotic maps which could be used in the Mmohom [115] [18, 113] [99]:

- Logistic: $x_{n+1} = \alpha x_n(1 - x_n),\ x_n \in [0, 1], \alpha \in [3.85, 4]$;
- Chebyshev of degree k: $x_{n+1} = cos(2^k cos^{-1} x_n),\ -1 < x_n \leq 1,\ k = 1, 2, 3, \ldots$;
- Kolmogorov[1]: $T_\pi(x, y) = (q_i(x - F_i), \frac{y}{q_i} + F_i), (x, y) \in [F_i, F_i + p_i) \times [0, 1)$.

Primitive polynomials modulo 2

In a broad sense we treat primitive polynomials as maps when the Mmohom is applied to the LFSR-based stream ciphers. A tap polynomial of length n: $p(x) = x^n + \ldots + 1$ over a finite field F_2^n is said to be primitive if it is irreducible and $p(x)$ divides $x^{2^n-1} + 1$ but not $x^d + 1$ for any d that divides $2^n - 1$ [109]. For example, $p(x) = x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$ is a primitive polynomial modulo 2.

T-functions

Klimov and Shamir [52, 53, 54, 55, 56] proposed the T-functions (Triangular-function) as a new class of invertible maps, with the property to be computable from the

[1]Given a unit square $\mathbb{E}$, a partition $\pi = (p_1, \ldots, p_k)$ $(0 < p_i < 1,\ \sum_{i=1}^{k} = 1)$ of a unit interval, let F_i defined by $F_1 = 0$, $F_i = F_{i-1} + p_{i-1}$, $q_i = \frac{1}{p_i}$

least significant bits (LSB) to the most significant bits (MSB) (i.e., information is propagated from right to left). A T-function is a mapping from n-bit to n-bit (a collection of memory words) with bitwise triangular structure such that bit i of the output depends only on bits 0-th through i-th of the input (for more details, see Chapter 8).

3.3 Analysis of the mechanism

3.3.1 Pseudo-cycle

If the state space is in $\mathbb{Z}^+$, the orbit of x_0 is a sequence of numbers by iterating the map $f(x)$. We know that there is a cycle t for any finite state machine. If we have n *seed*s $x_{0,0}, x_{0,1}, x_{0,2}, \ \ldots, \ x_{0,n-1}$, then we can have n sequences by iterating the map starting from those *seed*s.

How can we increase cycle by multiple orbits? Suppose we have a 2-bit finite state machine, then the total number of states is $2^2 = 4$. We have a PRNG (pseudo-random number generator) that can only generate 4 numbers {0, 1, 2, 3}, we start with an initial state, say 0, then the generated sequence of orbit-0 is $\{0, 1, 3, 2, \ldots, 0, 1, 3, 2\}$. If we start with another initial state say 1, then PRNG generates a sequence of orbit-1 $\{1, 2, 0, 3, \ldots, 1, 2, 0, 3\}$. They both have cycle length of 4. If we combine the two orbits into a long one by hopping, that is to pick one number at one orbit and then shift to the other orbit to pick another number, we'll get a sequence: $\{0, 1, 1, 2, 3, 0, 2, 3, \ldots, 0, 1, 1, 2, 3, 0, 2, 3\}$. We find out that the cycle doubled (see Figure 3.5).

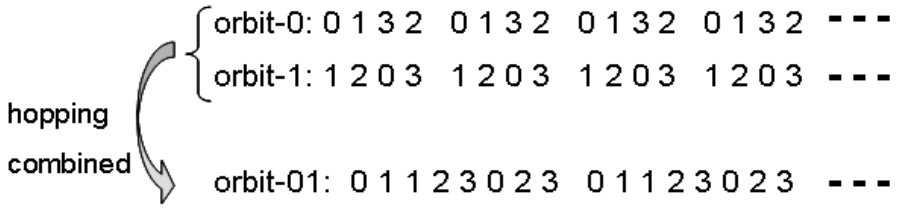


Figure 3.5: Combining two orbits into one, the "cycle" is doubled.

This does not contradict with the fact that the state size determines the cycle. The trick here is that we reuse each state twice in one "cycle." We call it a pseudo-cycle, and it does not mean that we expand the state space.

Definition 3.2. ***Pseudo-cycle****: a permutation of n elements in more than n positions. Some elements may take more than one position.*

One pseudo-cycle consists of multiple smaller cycles. In Figure 3.5 the pseudo-cycle is {0, 1, 1, 2, 3, 0, 2, 3}, it has smaller cycles {0, 1, 1, 2, 3}, {1, 1}, ... If we consider it in another way, making two adjacent states as one new "state", above sequence will become {01, 12, 30, 23}. It has cycle of 4 of distinct elements.

Theorem 3.3. *If n sequences generated by n orbits of* $x_{0,0}, x_{0,1}, x_{0,2}, \ldots, x_{0,n-1}$ *have cycles* $t_0, t_1, \ldots, t_{n-1}$*, then the new sequence generated by orbit hopping mechanism has a pseudo-cycle* $t_h \geq n \times min(t_0, t_1, \ldots, t_{n-1})$*. Here* $min(t_0, t_1, \ldots, t_{n-1})$ *takes the minimum of* $(t_0, t_1, \ldots, t_{n-1})$*.*

Proof. It can be proved by induction. □

3.3.2 Density functions

For the logistic map $S(x) = 4x(1-x)$ (used in Mmohocc cipher), we assume the density function $f_0(x) = 1$ for a large number N of initial states $(x_1^0, x_2^0, \ldots, x_N^0)$. When it is iterated infinite number of times, the density function becomes [61]

$$f_\infty(x) = \frac{1}{\pi\sqrt{x(1-x)}}.$$

In [61] it shows that the density of states along a trajectory approaches the same unique limiting density $f_\infty(x)$ as the iterates of densities approach. Therefore the states are mostly concentrated near 0 and 1 with a minimum at 0.5 for the logistic map.

For quadratic map $S(x) = x^2 + C$ (where $-2 \leq x \leq 2$, and $-1.9 \leq C \leq -2$ for the Mmohocc cipher) we obtain the limiting density function (for detail, see Appendix A)

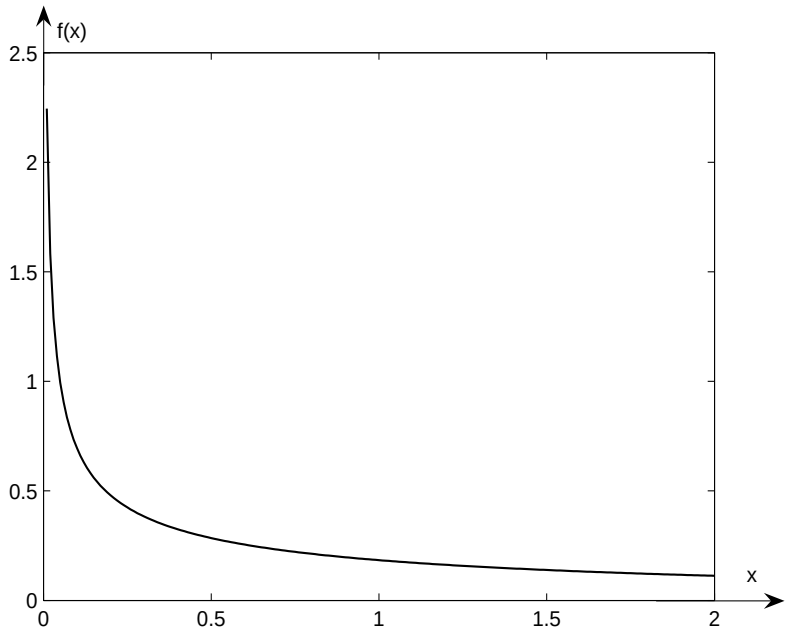
$$f_\infty(x) = \frac{1}{\pi\sqrt{x(x+2)}}.$$

It is depicted in Figure 3.6, and we can see that for quadratic map the states are concentrated near 0 with a minimum at 2.

3.3.3 Conjugate permutation

Let π and τ be any two permutations of a set S_n, i.e. $\pi, \tau \in S_n$ (where $S_n = (1, 2, \ldots, n)$), and $\tau = (i_1 i_2 \ldots i_m)$ is a cycle of length m $(1 \leq m \leq n)$, then

$$\pi\tau\pi^{-1} = (\pi(i_1)\pi(i_2)\ldots\pi(i_m)),$$



For quadratic map $S(x) = x^2 - 2$ the ultimate density function $f_\infty(x)$, which is derived from $f(x) = 1$ by calling Frobenius-Perron operator P infinite number of times.

Figure 3.6: Density function diagram for quadratic map.

i.e. the conjugate of a cycle τ of length m by an arbitrary permutation π is again a cycle of length m with entries obtained by applying π to all entries of the original cycle τ [96]. σ and π are called conjugate permutations by τ if $\sigma = \tau\pi\tau^{-1}$ has the same cycle structure as π.

Given π and σ in S_n, find a conjugate permutation τ in S_n such that $\sigma = \tau\pi\tau^{-1}$. For example, $\pi = (1, 2, 3)(4, 5, 6)$ and $\sigma = (7, 8, 9)(1, 3, 5)$ are permutations in S_9. We can find $\tau \in S_9$ such that

$$\tau(1, 2, 3)(4, 5, 6)\tau^{-1} = (7, 8, 9)(1, 3, 5).$$

It can be written as $\tau(1,2,3)\tau^{-1}\tau(4,5,6)\tau^{-1} = (7,8,9)(1,3,5)$.
So we assume that $\tau(1,2,3)\tau^{-1} = (7,8,9)$ and $\tau(4,5,6)\tau^{-1} = (1,3,5)$, which means that $\tau(1) = 7$, $\tau(2) = 8$, $\tau(3) = 9$, $\tau(4) = 1$, $\tau(5) = 3$, $\tau(6) = 5$. So we get $\tau = (6,5,3,9)(4,1,7)(2,8)$.

The Mmohom can be treated as a permutation τ on the state space. By applying it to a regular permutation π, we get another permutation σ which, we hope, has better diffusion property than π.

3.4 Conclusion

The Mmohom is a conceptual extension of the frequency hopping mechanism in spread spectrum communication, which offers three major advantages (i.e. anti-jamming, message privacy, and signal hiding) over the fixed-frequency transmission (see Section 3.1). Mmohom has a significant large pseudo-cycle for integer maps, better mixing, better confusion and diffusion, among other cryptographic desired properties. Our intention is to use this mechanism as a heuristic approach to build stream ciphers based on some well-studied maps.

Chapter 4

Multi-Map Orbit Hopping Chaotic Cipher

In this chapter we give a direct application for the multi-map orbit hopping mechanism in building a chaotic stream cipher which also utilizes fundamental chaos characteristics of mixing, unpredictability, and extremely sensitivity to initial conditions. The cipher design, key & subkeys handling as well as the cipher implementation are addressed. For illustration purposes an example is provided. This chapter is an extended version of [121].

4.1 Introduction to chaos

This section gives some basic knowledge about chaos, its properties, its measurement, and applications in cryptography.

4.1.1 What is chaos?

As a scientific branch, chaos theory has developed quickly since 1960s with efforts from many different research areas: meteorology, ecology, mathematics, physics, biology, chemistry, engineering, astronomy, economics, and more. There is no universally agreed common definition for chaos. But, most people would accept the following one[1]: "Chaos is aperiodic time-asymptotic behavior in a deterministic system which exhibits sensitive dependence on initial conditions." It contains three main elements:

1. **Aperiodic time-asymptotic behavior** – the phase-space orbits do not settle down to fixed points or periodic. We also require the orbits to be bounded: i.e., they should not go to infinity.

2. **Deterministic** – the equations of the system has no random inputs. I.e., the irregular behavior of the system arises from non-linear dynamics (system itself) and not from any random outside forces.

3. **Sensitive dependence on initial conditions** – nearby orbits in phase-space separate exponentially fast in time. I.e., the system has a positive Liapunov exponent[2].

In such a deterministic system how can two orbits never cross each other in a limited phase-space, and an orbit will never repeat itself no matter how long it runs?

[1] www.sewanee.edu/physics/PHYSICS123/chaos%20definitions.html

[2] G. Elert, Measuring Chaos: www.hypertextbook.com/chaos/43.shtml. Liapunov exponent λ is a quantitative measure of sensitive dependence on initial conditions. It is averaged rate of divergence (or convergence) of two neighboring orbits. Consider tow points X_0 and $X_0 + \Delta x_0$, then we have $\lambda = \lim_{\substack{t\to\infty \\ |\Delta x_0|\to 0}} \frac{1}{t} \ln \frac{|\Delta x_0(X_0,t)|}{|\Delta x_0|}$.

The technique behind the scene is the mechanism called **stretch-and-fold**. The "magic" is achieved by creating a very complex "interleaving" of orbits with a fractal structure, in such a way that there are always more orbits between any two you choose, no matter how much you "zoom in" [71].

In referring to chaos an important phenomenon is the butterfly-effect[3]. In fact, the butterfly-effect is a synonym of "sensitivity to the initial conditions." This makes the chaotic orbits generated by deterministic equations become entirely "unpredictable" as time elapses.

How do we characterize a dynamical system? People use equations (maps) to describe the behavior of a system. For instance, the famous logistic map is used to predict the behavior of population growth.

Here are other two well known maps that give rise to chaos [71] [19]: [4] [5] [6]

$$\text{Tent map: } x_{n+1} = \begin{cases} 2x_n & x_n \in [0, \frac{1}{2}) \\ 2(1 - x_n) & x_n \in (\frac{1}{2}, 1] \end{cases},$$

$$\text{Cat map: } \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} mod\ 1\ .$$

4.1.2 Chaos applications in cryptography

Shannon [103] indicated that the stretch-and-fold mechanism of mixing transformation is an important component of a cipher and it is also a way to confuse and diffuse

[3] www.cmp.caltech.edu/∼mcc/chaos_new/Lorenz.html

[4] classes.yale.edu/Fractals/Chaos/TestFcns/TentMap.html

[5] monet.physik.unibas.ch/∼elmer/pendulum/chaos.htm

[6] www-chaos.umd.edu/misc/catmap.html

messages. Chaotic dynamical systems, which exhibit unpredictable, mixing, ergodic and extreme sensitivity to initial conditions, have shown strong connection with cryptography since its inception. The essence of a chaotic dynamical 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. Therefore chaotic dynamical systems have attracted broad interest among cryptographic researchers investigating the cryptographic applications of chaotic properties [8, 1, 46, 36, 58, 3]. The links to conventional cryptography, in which there are a wide variety of appropriate design and analysis methods, have been established [19, 37, 29, 57].

However, regular chaos systems have proved to be difficult in research and application because of their certain instinct properties: they are based on continuous systems and they are long-term behaviors. Limited resources and computational power in digital systems can severely degrade or even extinct those properties [62]. Most of the chaotic cryptosystems are based on one particular map to iterate (especially for stream ciphers) [88, 124, 2, 68, 69, 85]. The major problem in using iterative number generators is that they can enter unexpectedly short cycles [102]. In order to create a long cycle and to make use of chaotic properties in cryptosystems, based on our Mmohom technique (described in Chapter 3), which accelerates chaotic behaviors more rapidly than a regular chaos system, we have developed a chaotic stream cipher, called Mmohocc (Multi-Map Orbit HOpping Chaotic Cipher).

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. Chaos itself can be used to generate frequency hopping sequences (patterns) [63].

Here we define a chaotic orbit [31]. Given $x_0 \in \mathcal{R}$ and a chaotic map $\mathbf{F}$, we define the orbit of x_0 under $\mathbf{F}$ to be the sequence of points x_0, $x_1 = \mathbf{F}(x_0)$, $x_2 = \mathbf{F}^2(x_0)$, ..., $x_n = \mathbf{F}^n(x_0)$, ... The point x_0 is called the seed of the orbit.

Rowlands [97] proposed an orbit shift logistic map chaotic encryption scheme in which 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 a particular orbit.

We now explain more about the settle and why we need it. In multiple orbit situations we want all orbits to be as dissimilar to each other 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 the 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 for a cryptographic application. In order to avoid this situation, 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 orbits are different enough from each other.

By using the Mmohom, 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.

4.2 Cipher design

Here we give the cipher design block diagram, and then verify that Chebyshev maps are chaotic maps.

4.2.1 Block diagram

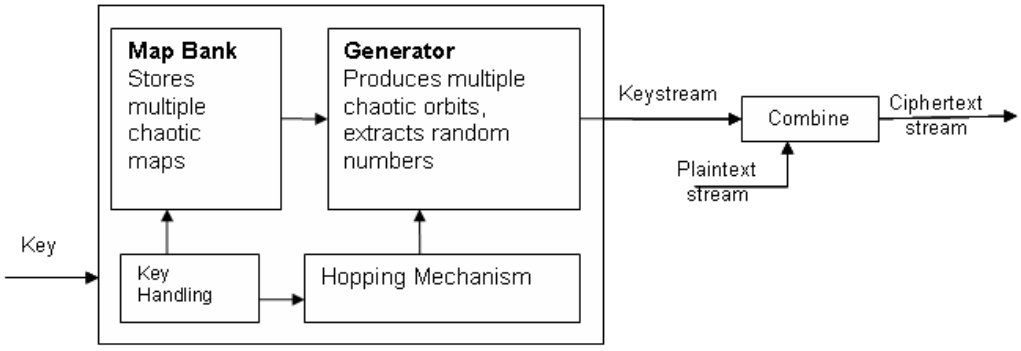


Figure 4.1: The block diagram of the chaotic cipher.

In Figure 4.1, given a key the hopping mechanism performs a key handling process, then the cipher chooses m maps $M_0, M_1, \ldots, M_{m-1}$ from a map bank and sets the order of the chosen maps to hop. (A key is turned into m subkeys after a key handling process.) For each individual map, s orbits $S_0, S_1, \ldots, S_{s-1}$ are generated.

And further, on each orbit, n points $N_0, N_1, \ldots, N_{n-1}$ are generated. Parameters s, n and hopping pattern are determined by the key.

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

4.2.2 Chaotic map bank

For the pseudo random number generator of the stream cipher we use a bank of chaotic maps whose parameters are properly tuned to make sure that all maps lead to chaotic. A map is chaotic [31] if it is sensitive to initial conditions, topologically mixing and its periodic orbits are dense.

In our current system, we tested a various number of logistic maps and Chebyshev maps which have been well studied and understood [31], and they both gave satisfied results. A logistic map is defined as

$$x_{n+1} = r x_n (1 - x_n),\ x_n \in (0, 1)$$

where $0 \leq r \leq 4$. Most values of r beyond 3.57 exhibit chaotic behavior (but there are still certain isolated values of r that appear to show non-chaotic behavior, e.g. $r = 3.82$) [115].

A Chebyshev map [18] of degree k is defined as

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

Chebyshev map of degree k is the kth iteration of the logistic map [113]. Thus

Chebyshev maps possess all chaotic features that logistic maps have, they are ergodic, mixing, unpredictable, and extremely sensitive to initial conditions.

4.3 Cipher implementation and key issues

4.3.1 Implementation flowchart

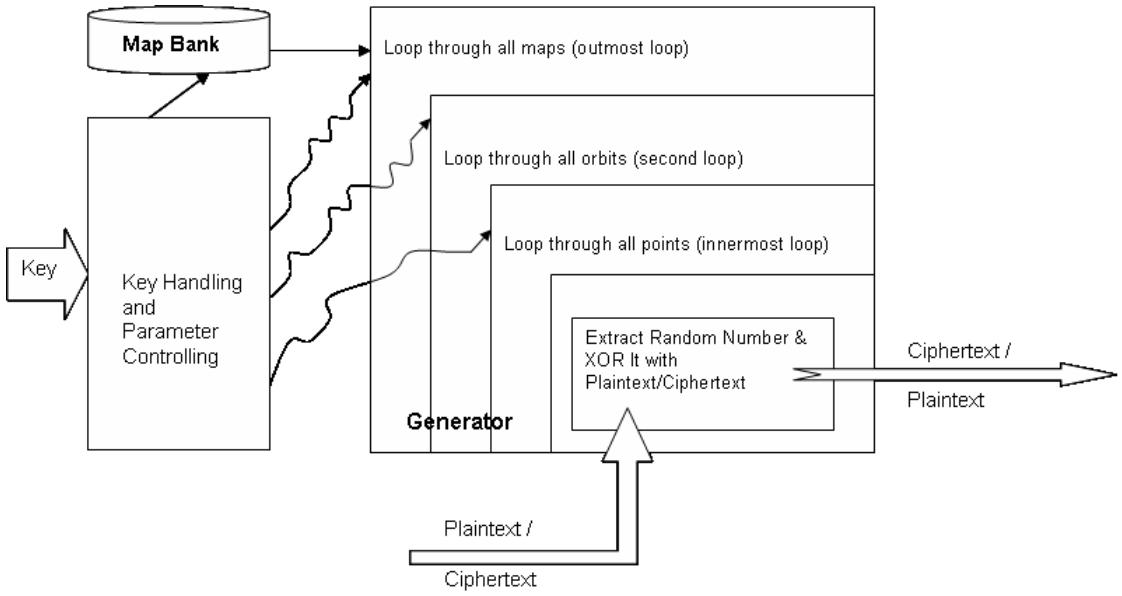


Figure 4.2: Detail implementation flowchart.

In Figure 4.2, the map bank stores all possible chaotic maps that could be used for the system. Hopping mechanism is in charge of picking up maps from the map bank, as well as key handling and parameter controlling for generator. The generator consists of three loops. The outmost loop steps through all maps. For each map, several orbits are produced, further for each orbit several points are sampled to generate

random numbers. The number of maps, orbits, and points are determined by the key.

Notice that orbits for each map-loop will not be repeated. When coming back to the first map-loop, orbits' offsets will be increased based on the offset that the same map-loop left off last time.

4.3.2 Key handling and subkey structure

The key handling is the means by which the key bits are turned into $\#maps$ (# represents "number of") subkeys that the cipher can use. In this initial version we simply provide a long key and sequentially split it into 8 (here $\#maps = 8$) subkeys. These subkeys are used to set up all control parameters for the 8 maps. Assume that we can use a slow coin-flipping method to generate keys (it's a one time work). In the later version, key handling will use a shorter key to generate all the subkeys needed for the system.

This is a variable-key-size stream cipher. The longer the key is, the more maps the cipher will use. The length of key is determined by number of maps used in the cipher. For the first version of the cipher, maps are picked up sequentially from the Map Bank without further scrambling. Orbits are generated by increasing a unit offset and sample points are picked sequentially, too.

Each map needs a subkey to control its behavior for generating suitable orbits/points. A subkey has the structure in Figure 4.3.

From Figure 4.3, a subkey has the fields *hpsn, seed, offset, #settles, #orbits, #samples* (the concatenated fields make the subkey.) The length of a subkey is 56-bit. Here we explain each field and its size.

Figure 4.3: Subkey structure.

***hpsn*:** hopping-pattern serial number (not considered in this version).

***seed*:** the initial value (state) of the chaotic map, represented in 6 hex-decimals, i.e. 24 bits. E.g., $25fbd8_h$ is the seed. Before being passed into map iteration, the hex will be converted into 2489304_d, then we add two zeros at left of decimal number, then add decimal point at the most left to make it as 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 value is represented in 8 bits.

***#orbits*:** the number of orbits generated during each use of a map. It is between 4 and 19, can be 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 too.

The key length is determined by the number of maps (8 maps here) and the length

of subkey. In our current settings the number of maps is set to 8, and subkey length is 56-bit. So the key will be (in bits) $8 \times 56 = 448$ bits and the key space is 2^{448}.

Subkeys are used to control the parameters of maps (*seed*, *offset*, *#settles*, *#orbits*, *#samples*, and *hpsn*). The *hpsn* is intended to control how the cipher hops between the orbits.

The random number extraction from a chaotic sample point is described in the section 4.3.3. After the random number is generated, we combine it with plaintext/ciphertext to get corresponding ciphertext/plaintext. Here, we simply use exclusive OR (XOR) as the combination function (it could be replaced by other alternatives.)

4.3.3 Chaotic random number extraction

We extract an 8-bit integer from a specific chaotic orbit point x_n by the following steps:

1. Remove the decimal point from x_n ($|x_n|$ is less than 1). E.g., $0.33461 \rightarrow 33461$, $0.9442345679457 \rightarrow 9442345679457$.

2. Make 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., $33461 \rightarrow 33461000$, $9442345679457 \rightarrow 45679457$.

3. Use the 8 digits number modulo 2^8 to generate a pseudo random number. E.g., $33461000 (mod\ 256) = 8$, $45679457 (mod\ 256) = 97$.

The generated random numbers are between 0 and 255, inclusively. That's the total number of ASCII character set (including extended ones). We combine (XOR here) the random numbers with plaintexts/ciphertexts for encryption/decryption.

4.3.4 State space/cycle length

The internal state space is $\#maps \times \#orbits \times 32$ bits. If $\#maps = 8$, $\#orbits = 4 \sim 9$, the average state space will be $8 \times 12 \times 32 = 3074$ bits. Therefore the cycle length for the generated numbers is about 2^{3074}.

4.4 An illustrating example

4.4.1 Key and subkeys

A key is given to generate the chaotic pseudo random numbers. According to the key, the cipher uses eight maps. The key handling breaks the key into eight subkeys and sets up all the control parameters of the system.

Here is the key, it consists of 112 random numbers (0 to 15, i.e., 0 to f in hexadecimal). We know that $\#maps = 8$. Each map needs a 56 bit subkey, so we need 56 x 8 = 448 bits, i.e., 112 hex numbers. Suppose those random numbers are generated by a coin-flipping method:

11 13 1 4 4 11 3 10 8 14 6 9 7 7 1 14 10 14 6 2 14 15 9 7 1 7 11 0 8 10 7 1 6 11 8 4 11 9 14 5 3 4 3 7 1 10 14 7 5 9 5 6 5 15 8 11 15 1 8 5 15 1 5 14 14 7 8 8 7 10 14 6 11 7 15 4 2 2 0 0 11 9 2 10 11 4 6 9 0 10 8 15 7 14 13 3 9 2 3 1 13 3 6

3 9 10 15 14 5 4 15 3.

The key handling process splits the bits, when represented in hexadecimal format the 8 subkeys are: bd144b3a8e6977 1eae62ef9717b0 8a716b84b9e534 371ae759565f8b f185f15ee7887a e6b7f42200b92a b4690a8f7ed392 31d3639afe54f3.

4.4.2 The chaotic maps and orbits

The eight chaotic maps are: $M_0 \sim M_3$ logistic, $M_4 \sim M_7$ Chebyshev, with corresponding coefficients r: $3.901, 3.931, 3.963, 4$, and k: $3, 4, 5, 6$. That is, the four logistic maps ($x_n \in (0, 1)$) are: $x_{n+1} = 3.901x_n(1 - x_n)$, $x_{n+1} = 3.931x_n(1 - x_n)$, $x_{n+1} = 3.963x_n(1 - x_n)$, $x_{n+1} = 4x_n(1 - x_n)$, and the four Chebyshev maps ($x_n \in (-1, 1)$) are: $x_{n+1} = cos(2^3 cos^{-1} x_n)$, $x_{n+1} = cos(2^4 cos^{-1} x_n)$, $x_{n+1} = cos(2^5 cos^{-1} x_n)$, $x_{n+1} = cos(2^6 cos^{-1} x_n)$.

According to the key, there are eight maps involved. Each map has its own *seed, offset, #settles, #orbits, #samples*. The details are in Table 4.1.

There are 45 orbits in the four logistic maps ($M_0 \sim M_3$), and 47 orbits from the four Chebyshev maps ($M_4 \sim M_7$). Different colors are used to distinguish orbits created from different maps, Red for M_0/M_4, Green for M_1/M_5, Blue for M_2/M_6, and Black for M_3/M_7. After their settle periods, the first four sample points of all those orbits are depicted in Figure 4.4. As shown here, those orbits are sufficiently different from each other. Furthermore, we use our chaotic random number extraction method described in the previous section to obtain random numbers from those sample points.

Table 4.1: Initial setups for eight maps

map	*seed*	*offset*	*#settles*	*#orbits*	*#samples*
M_0	0.0012391499	0.00001499	135	11	11
M_1	0.002010722	0.000061335	53	15	4
M_2	0.009073003	0.000033977	259	7	8
M_3	0.003611367	0.00002287	125	12	15
M_4	0.0015828465	0.000024295	166	11	14
M_5	0.0015120372	0.00008704	30	4	4
M_6	0.001182337	0.000036734	241	13	6
M_7	0.003265379	0.000039678	114	19	7

4.4.3 Histograms of plaintext and ciphertext

To compare with the result of [97] we use our cipher to encrypt the same novel "Huckleberry Finn" (total $592,299$ characters in pure text format) [111]. Figures 4.5(a) and (b) show histograms of plaintext and the corresponding ciphertext of the novel. In Figure 4.5(a) we see clustering around letters and peaking for space. Figure 4.5(b) shows fairly even distribution, i.e., in the ciphertext novel each of 256 ASCII characters occurs with nearly equal likelihood.

In MS Word format, the size of the novel is 1,404,928 characters long. The histograms of the plaintext (in MS Word format) and its corresponding ciphertext encrypted by using the same key are provided in Figure 4.6 for comparison.

In the histogram of the ciphertext in [97], there are four clusters.

4.5 Conclusion

In addition to chaotic features of mixing, unpredictability, and extreme sensitivity to initial seeds, by using multi-map orbit-hopping technique we implement a symmetric cryptosystem which spreads out pseudo random number base to a wide flat "spread spectrum" (similar to white noise) in terms of time and space. The distribution of those numbers is fairly even and flattened.

The stream cipher with variable key length makes use of chaotic system parameters *seed, offset, #settles, #orbits, #samples* plus *hpsn* as secret key. The chaotic maps we chosen are computationally fast, the cryptosystem is easily implemented in software.

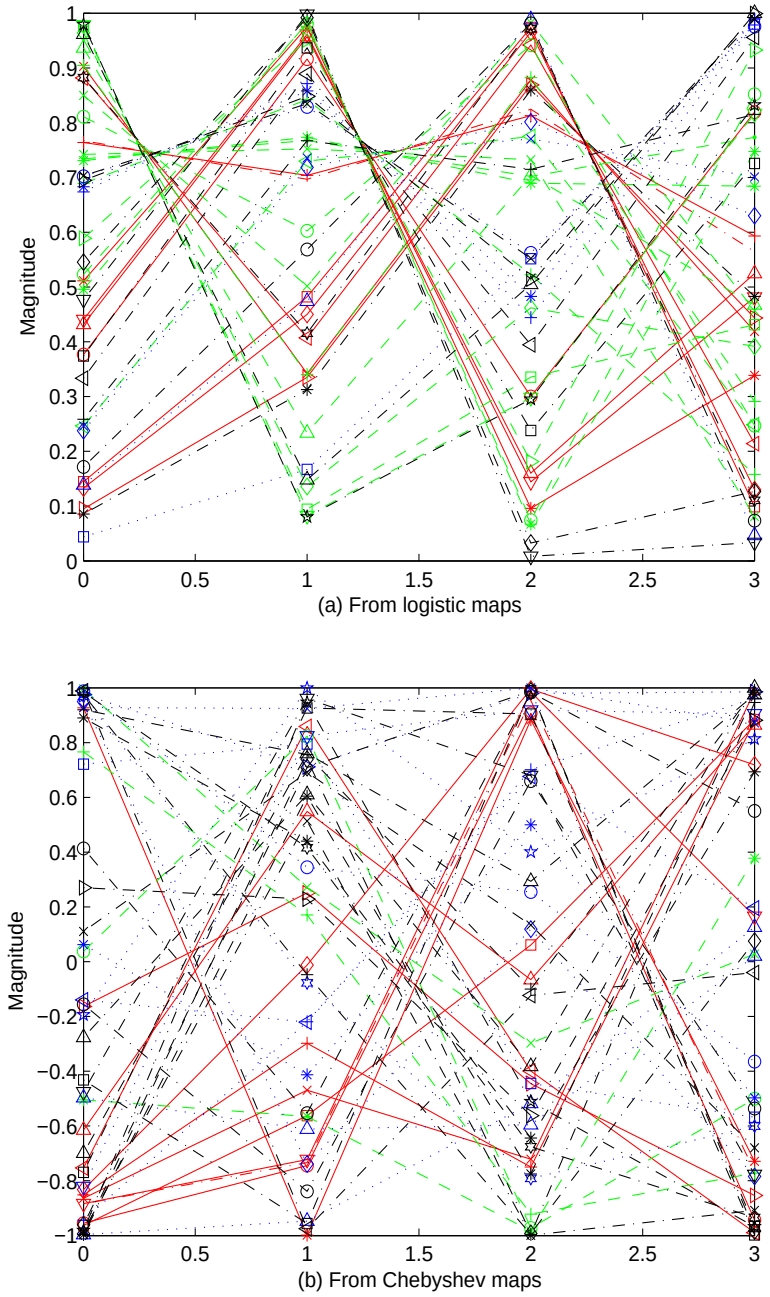
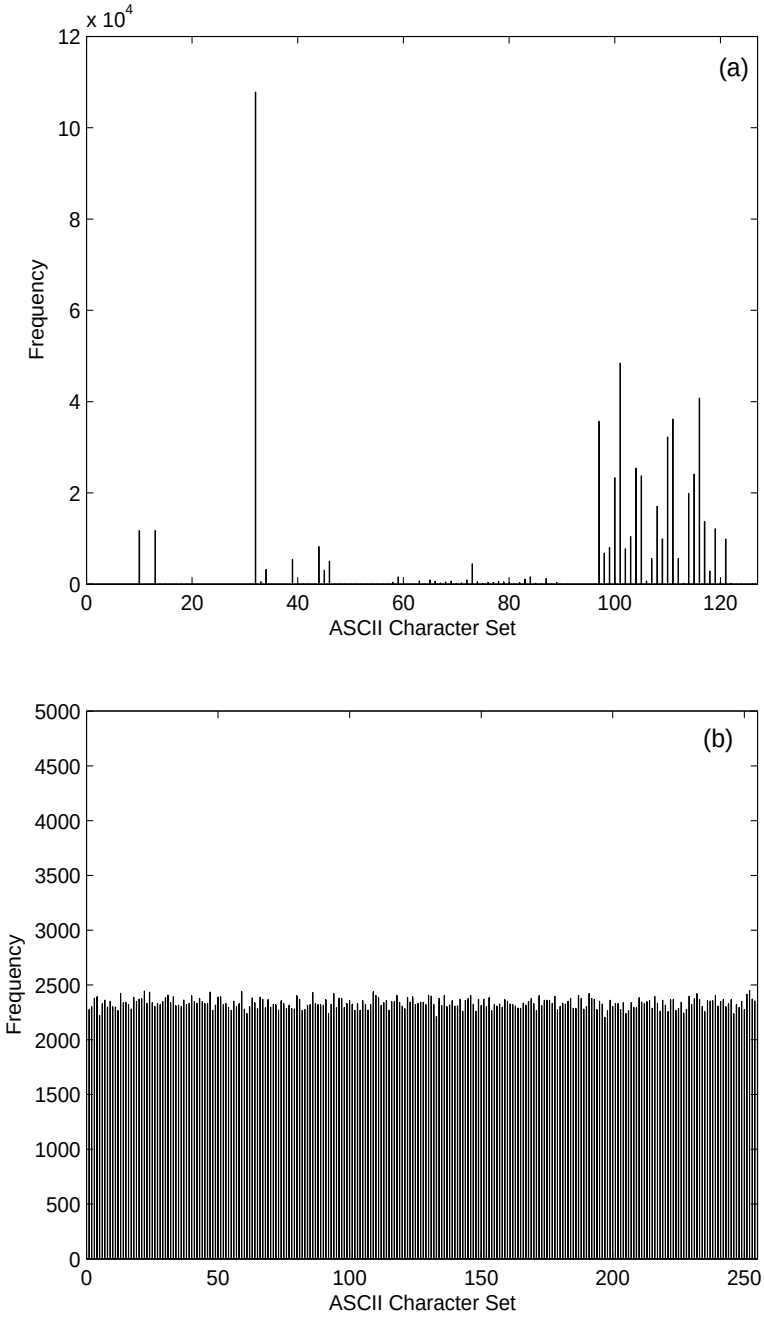
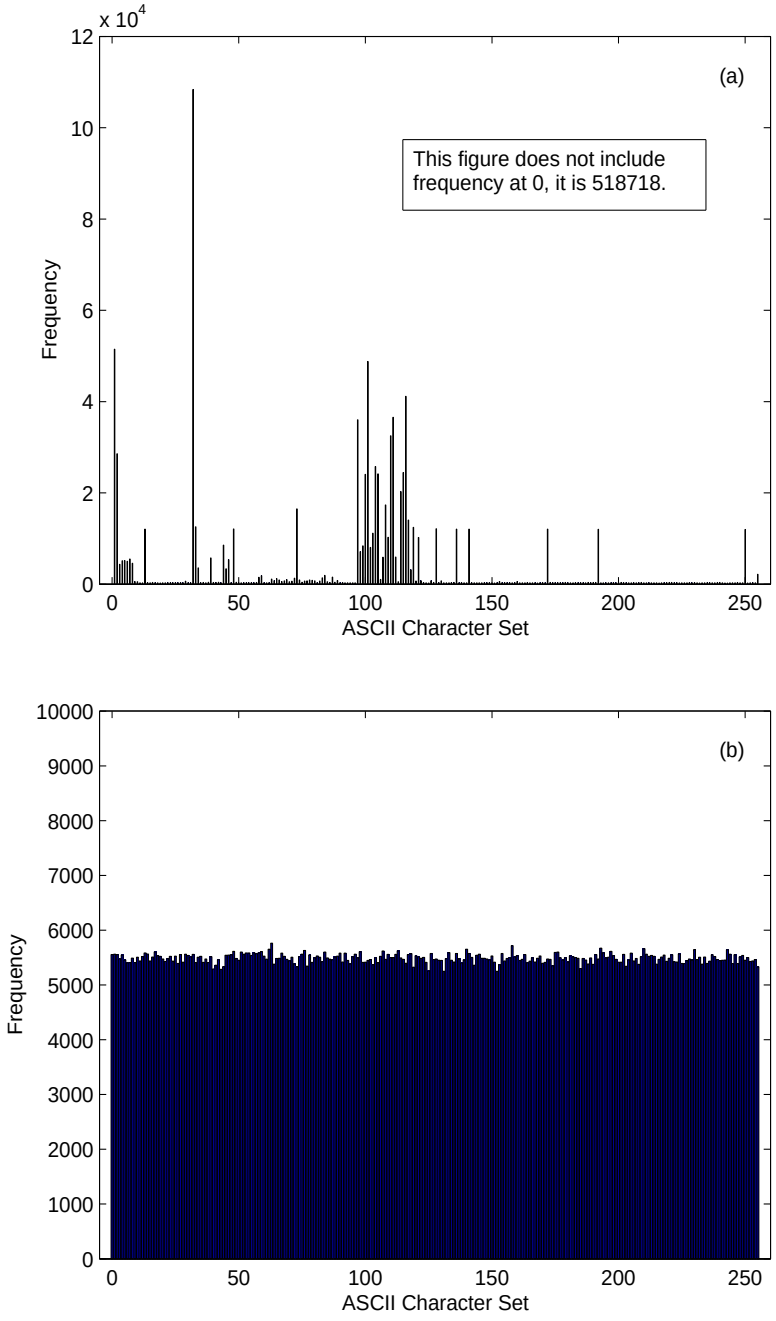


Figure 4.4: 92 chaotic orbits of the first four samples after settle periods.



(a) Plaintext histogram. (b) Ciphertext histogram.

Figure 4.5: Character frequency of "Huckleberry Finn" in pure text format.



(a) Plaintext histogram. (b) Ciphertext histogram.

Figure 4.6: Character frequency of "Huckleberry Finn" in MS Word format.

Chapter 5

Improved Mmohocc & Randomness Evaluation

In this chapter we improve the design of Mmohocc cipher and then conduct the randomness statistical tests against the keystreams generated by Mmohocc cipher. Two batteries of most stringent randomness tests, namely the NIST Suite and the Diehard Suite, were performed. The results showed that the keystreams successfully passed all the statistical tests. This chapter is an extended version of [123].

5.1 Improving Mmohocc cipher

The Mmohocc is a software based stream cipher. It "hops" among substantially many chaotic orbits to "pick up" its random sequence points. The base ground for random sequences is spread among many chaotic orbits, which are generated from multiple chaotic maps.

Improvements are achieved in the following aspects: selecting faster chaotic maps, using different key handling procedure (see Figure 4.1), adding *hpsn* (hopping-pattern serial number) field in subkeys, and changing the random number extraction method.

5.1.1 Chaotic maps

For the pseudo random number generator of the Mmohocc cipher, we use a set of chaotic maps from the Map Bank whose parameters are properly tuned to ensure that all the maps lead to chaotic. Besides the logistic maps $x_{n+1} = rx_n(1-x_n),\ x_n \in (0,1)$ used in previous chapter, we add quadratic maps into our map bank. Quadratic map is defined as [31]

$$x_{n+1} = x_n^2 + c,$$

and its behavior depends on parameter c. Most values of c beyond -1.45 left exhibit chaotic behaviors, see Figure 8.1 in [31]. When c is close to -2 the orbits x_n distribute within $(-2,+2)$, i.e. $x_n \in (-2,2)$. We choose c between -1.9 to -2 for our set of quadratic maps. In the implementation, for logistic and quadratic maps we chose r and c carefully to avoid the windows (openings).

5.1.2 Key handling and initialization vector

The key handling expands a 128-bit (or 256-bit, or 512-bit) key into many subkeys (SKs) for controlling the operations of multiple chaotic maps. Each subkey contains a certain number of fields.

Currently Mmohocc cipher has three versions, each with 128-, 256-, and 512-bit key; and we name them as Mmohocc-128, Mmohocc-256, and Mmohocc-512,

respectively.

The Mmohocc-128/256 uses 8 chaotic maps ($\#maps = 8$) and a 64 bits initialization vector (IV). This is as if it had 8 rounds in parallel (simultaneously). Each round uses a subkey to control the behavior of the corresponding map. Key handling has the following four steps:

1. Halve the key and IV. The key is written in the form $K_L K_R$, where K_L is the first 64/128 bits and K_R is the last 64/128 bits. IV is written in the form $V_L V_R$, where V_L is the first 32 bits and V_R is the last 32 bits.

2. Concatenate the halves as $V_L K_L$ and $V_R K_R$. They are now 96/160 bits each.

3. Expand the two concatenations. Hash $V_L K_L$ and $V_R K_R$ with SHA256, we get two 256-bit hashes $H_L = SHA256(V_L K_L)$ and $H_R = SHA256(V_R K_R)$.

4. Interweave and split. Interweave H_L and H_R bit-by-bit to get a 512-bit long string, then split it into 8 even substrings each with 64-bit $k_1, k_2, \ldots, k_8$, which are 8 subkeys for 8 chaotic maps, correspondingly.

For Mmohocc-512, the key handling is similar to Mmohocc-128/256, the difference is that it uses 16 chaotic maps and a 128-bit IV.

This key handling provides a fine chopping to the key. And it sufficiently uses the key and IV information. Expanding the key into subkeys does not reduce the key space. For our system IVs are multiple 64 bits. In Mmohocc-128/256 the IV is 64-bit, and in Mmohocc-512 the IV is 128-bit. Mmohocc-128/256 has subkey length of 64-bit, the length of each field in a subkey is as follows: $hpsn$ – 8, $seed$ – 24, $offset$ – 16, $\#settles$ – 8, $\#orbits$ – 4, $\#samples$ – 4.

5.1.3 Hopping patterns

The hopping pattern is a predefined pseudorandom sequence, it is represented by field *hpsn* in each subkey. Hopping mechanism uses *hpsn* to govern the jumping behavior between the chaotic orbits, i.e. how Mmohocc hops among orbits. In the current version the *hpsn* field of a subkey takes 8 bits long, therefore it can be changed between 0 and 255. Each *hpsn* corresponds to one of available hopping patterns, which are stored in a lookup table. For instance, if a chaotic map has 11 orbits and its *hpsn* is 141, then the hopping-pattern {2, 1, 9, 11, 10, 6, 5, 8, 7, 4, 3} is returned from the lookup table. When it comes to this map Mmohocc extracts random numbers on orbit 2 first and then orbit 1, orbit 9, ..., orbit 3 in this order. Table 5.1 is a partial hopping pattern lookup table, for more details please see Appendix B.

Table 5.1: A partial hopping pattern lookup table

hpsn	Sequential orbit permutation sequence	hpsn	Swapped orbit permutation sequence
20	(1,2), (9,10,11), (5,6), (3,4), (7,8)	140	(2,1), (9,11,10), (6,5), (4,3), (8,7)
21	(1,2), (9,10,11), (5,6), (7,8), (3,4)	141	(2,1), (9,11,10), (6,5), (8,7), (4,3)
22	(1,2), (9,10,11), (7,8), (3,4), (5,6)	142	(2,1), (9,11,10), (8,7), (4,3), (6,5)
23	(1,2), (9,10,11), (7,8), (5,6), (3,4)	143	(2,1), (9,11,10), (8,7), (6,5), (4,3)
24	(3,4), (1,2), (5,6), (7,8), (9,10,11)	144	(4,3), (2,1), (6,5), (8,7), (9,11,10)
25	(3,4), (1,2), (5,6), (9,10,11), (7,8)	145	(4,3), (2,1), (6,5), (9,11,10), (8,7)

5.1.4 Random number extraction

From the density functions in Chapter 3 we know that the trajectory points (states) are not distributed evenly: states are mostly concentrated near 0 and 1 with a minimum at 0.5 for the logistic map, and the states are concentrated near 0 with a minimum at 2 for quadratic map. Therefore we need a way to destroy the uneven distributions.

A chaotic orbit point x_n is a real number (in the case of logistic map, $x_n < 1.0$) and is represented in a data type of double in computer. We extract two smaller integers as random numbers from the point x_n in the following method, see Figure 5.1. By so doing we destroy the uneven density distributions, and further muddle the bits and add additional randomness to the keystream. This is the filtering function of the cipher, a multi-output Boolean function. It acts like an S-Box with 32-bit-in and 16-bit-out.

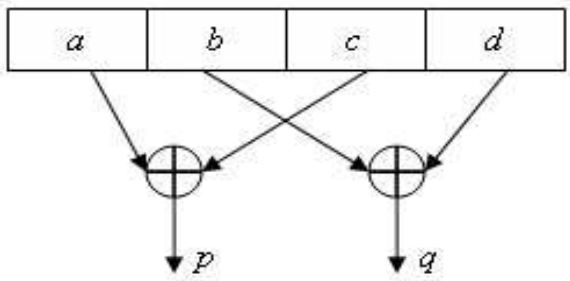


Figure 5.1: Random number extraction.

1. Convert double type x_n into a 32-bit integer by multiplying 2^{52} to take the least 32 significant bits (x_n is a 64-bit double type number).

2. Split the above integer into four 8-bit integers a, b, c, d. Here a and d are the most left and right 8-bit, b and c are in the middle. The two smaller output pseudorandom integers are: $p = a \oplus c$ and $q = b \oplus d$.

5.2 Statistical tests and results

There are batteries [82] of statistical tests available to analyze cryptographic random number generators (RNGs) and pseudorandom number generators (PRNGs): the pLab, the Crypt-X, the DIEHARD, the NIST, the ENT, the RIPE, and others. Among them the NIST and the DIEHARD are considered the most stringent randomness tests and are free of charge. To assess the Mmohocc cipher we conducted the two aforementioned batteries of tests. There are 16 statistical tests in the NIST Suite [83], and 18 in the DIEHARD Suite [67]. The names of these tests are listed in Table 5.4 and Table 5.5 with the test serial number (TSN) in the first column. For detailed description of these tests, see [82, 67].

The randomness of a bit sequence is characterized and described in terms of probability. Both NIST and DIEHARD Suites include dozens of independent and computationally intensive statistical tests. Most of these tests return a test statistic and its corresponding probability value (*p-value*) [106]. The *p-value* [115] is the probability of obtaining a test statistic as "impressive" as the one observed if the sequence is random, so that the statistic was the result of chance alone. In other words, the *p-value* summarizes the strength of the evidence against the perfect randomness hypothesis. Small values ($p\text{-}value < 0.05$ or $p\text{-}value < 0.01$) are interpreted as evidence that a sequence is unlikely to be random. Here 0.05 and 0.01 are significance level, usually

denoted as α.

The *p-value*'s are obtained by $p = F(z)$, where F is a special function such as the complementary error function $erfc$, the incomplete gamma function *gamminc*, the standard normal (cumulative distribution) function *normcdf*, or the gamma function *gamma*.

5.2.1 Results from NIST Suite

Parameter setups for the NIST Suite are in Table 5.2.

Table 5.2: Parameter setups for NIST Suite

Parameter Name	Value
Bit length	1000000
Block frequency block length	10
Non-overlapping template block length	9
Overlapping template block length	10
Universal block length	7
Universal number initialization steps	1280
Approximate entropy block length	14
Serial block length	16
Linear complexity sequence length	5000
Word Length	10
Number of streams	1000

Table 5.4 shows the result obtained on a series of data files of 1,192,755,216 (over 1 billion) bits each. The files are generated from the Mmohocc by using different keys. Each time the Suite takes one file as input data and executes all 16 tests. Set 1000 as the number of bit sequences, each containing 1 million random bits. The mean and variance of the *p-value*'s are displayed in the table. For more results, please see Appendix C.

For further interpreting empirical results NIST Suite has also adopted the following two approaches [83]:

Approach–1: The Examination of the Proportion of Passing Sequences

In the final analysis report file generated by the Suite, a value called the proportion is listed for each test. The proportion is the number of sequences having a *p-value* greater than the significance level α, divided by the total number of sequences tested, i.e., the percentage of passed tests.

NIST SP800-22 specifies a range of acceptable proportions. The range is determined by using the confidence interval defined as $\hat{p} \pm 3\sqrt{\hat{p}(1-\hat{p})/m}$, where $\hat{p} = 1-\alpha$, and m is the sample size, which tells how many bit sequences are tested.

In our case, 1000 sequences ($m = 1000$) with one million bits per sequence are used. By the formula above, the range of acceptable proportion is from 0.9805 to 0.9994, inclusively. Figure 5.2 shows the proportion for each of the 16 tests. Since the proportion for each test is within the range, we are confident to accept the sequence as random bit sequence.

Approach–2: The Examination of the Uniformity of *p-value*'s

To visually examine the uniformity of *p-value*'s we can make a histogram for each

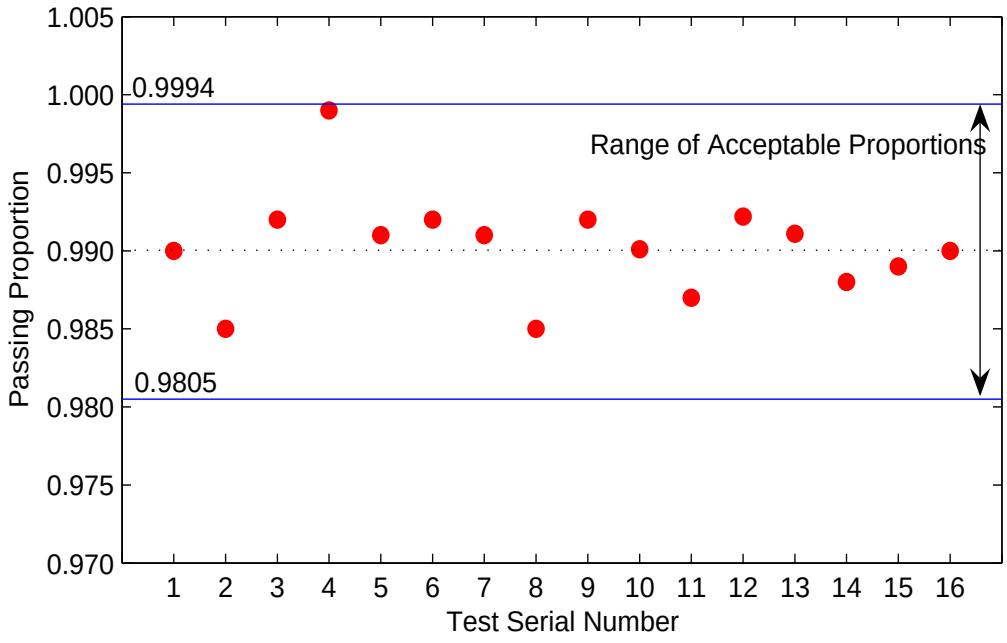


Figure 5.2: Passing proportions of 16 tests in NIST Suite.

test. The horizontal coordinate (the probabilities) interval between 0 and 1 is divided into 10 sub-intervals evenly, and the number of *p-value*'s that drops within each sub-interval is displayed. As an example, Figure 5.3 shows the histograms of *p-value*'s for longest runs and rank tests. We can see that the *p-value*'s are uniformly distributed.

Uniformity is also examined by computing the following χ^2 value for each test [83],

$$\chi^2 = \sum \frac{(C_i - m/10)^2}{m/10}$$

where C_i is the number of *p-value*'s in sub-interval $[(i-1)/10, i/10)$ $(i = 1 \sim 10)$, and m is the sample size (i.e., the number of bit sequences tested). We calculate a

p-value of the *p-value*'s obtained for a statistical test as

$$p_p\text{-}value = gammainc(\chi^2/2,\ 9/2,\ \text{`}upper\text{'}).$$

If p_p-*value* ≥ 0.0001 then those *p-value*'s can be considered uniformly distributed. Table 5.3 shows the p_p-*value*s of the 16 tests and all *p-value*'s are uniformly distributed.

Table 5.3: Uniformity distribution of *p-value*'s (NIST Suite)

TSN	1	2	3	4	5	6	7	8
p_p-*value*	0.546	0.432	0.035	0.057	0.197	0.047	0.693	0.392
Conclusion	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
TSN	9	10	11	12	13	14	15	16
p_p-*value*	0.302	0.473	0.347	0.395	0.380	0.868	0.107	0.508
Conclusion	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

5.2.2 Results from DIEHARD Suite

DIEHARD Suite does not ask for many parameter setups besides the size of the files being tested.

The Diehard Suite developed by Marsaglia [67] consists of 18 stringent statistical tests. A binary file must be provided – a file of 10 to 11 megabytes, i.e., at least 80 million bits. 100 binary files of 100 million bits each are generated by Mmohocc with

100 different keys. Table 5.5 shows the result produced from a single binary file. A *p-value* larger than 0.01 and smaller than 0.99 means that the sequence is random with a confidence of 99%. Note: most of the tests in DIEHARD return a *p-value*, which should be uniform on $[0, 1)$ if the input file contains truly independent random bits.

5.3 Conclusions

With the hopping mechanism the Mmohocc cipher greatly speeds up the chaotic mixing. This makes the Mmohocc a feasible stream cipher. Although evaluating statistical characteristics of keystreams can never replace security analysis for a stream cipher, in principle it is an inevitable and standard procedure to go through randomness tests. Successfully passing all two batteries of the stringent statistical tests confirms and supports our design. We conclude this chapter with the following summary:

- The Mmohocc uses a hopping mechanism to implement a chaotic cipher.
- The keystreams generated by the Mmohocc have passed two batteries of randomness tests with satisfactory results.
- All the testing results obtained on the Mmohocc confirmed that a high level of confidence in the randomness of the keystreams has been achieved.
- The Mmohocc generates high quality pseudorandom numbers.

- The randomness statistical tests ensure our technique and help to improve our adjustments for certain parameters/coefficients involved in the chaotic maps, orbit hopping offsets, and most importantly hopping patterns.

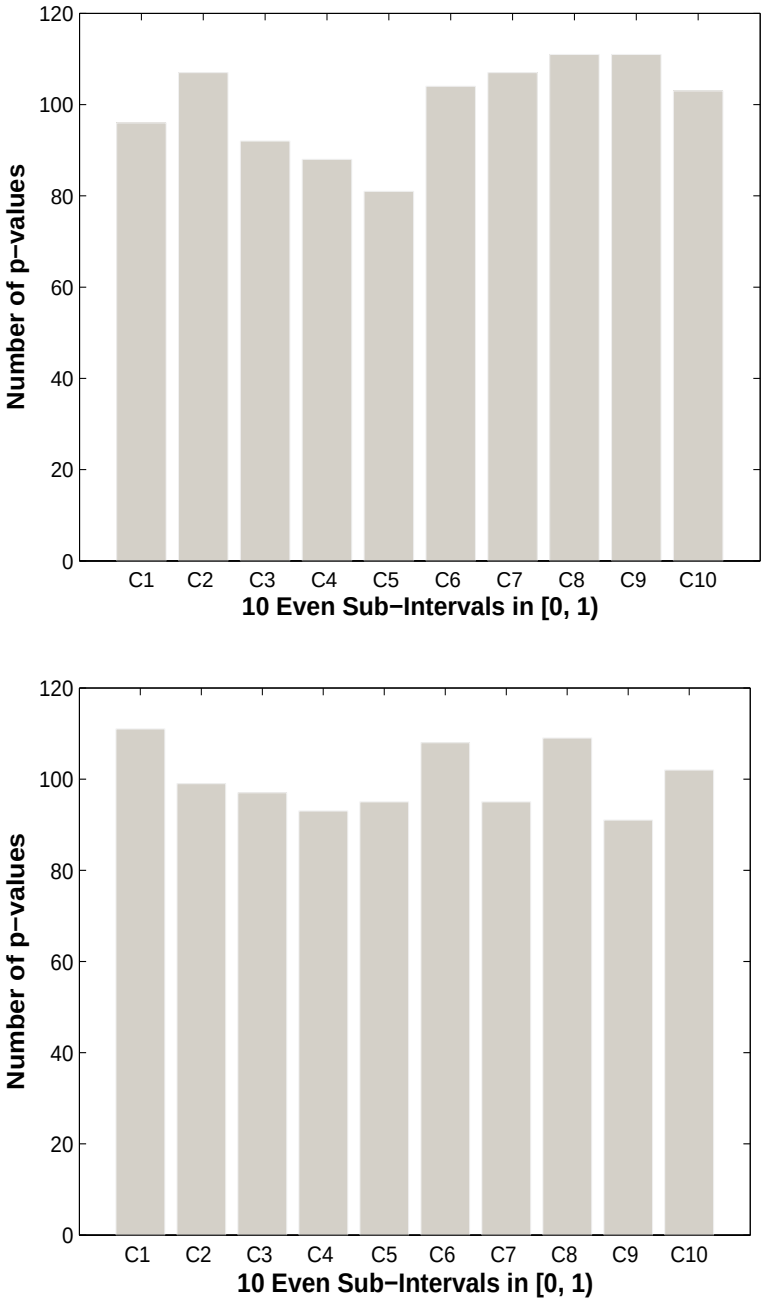


Figure 5.3: Histograms of *p-value*'s for longest runs and rank tests

Table 5.4: Mean and variance of *p-value*'s for a bit sequence (NIST Suite)

TSN	Test Name	Mean of *p-value*'s	Variance of *p-value*'s	Conclusion
1	Approximate Entropy	0.5146	0.0857	Success
2	Block Frequency	0.5040	0.0844	Success
3	Cumulative Sums (Forward)	0.4777	0.0799	Success
	Cumulative Sums (Reverse)	0.4824	0.0815	Success
4	Fast Fourier Transform (Spectral)	0.4799	0.0800	Success
5	Frequency (Monobit)	0.4894	0.0827	Success
6	Lempel-Ziv Compression	0.5024	0.0823	Success
7	Linear Complexity	0.5024	0.0823	Success
8	Longest Runs of Ones	0.5121	0.0849	Success
9	Maurer's Universal Statistical	0.5000	0.0889	Success
10	Non-Overlapping Template Matching	0.4997	0.0833	Success
11	Overlapping Template Matching	0.4970	0.0830	Success
12	Random Excursions	0.5087	0.0835	Success
13	Random Excursions Variant	0.5103	0.0813	Success
14	Rank	0.4963	0.0841	Success
15	Runs	0.4980	0.0841	Success
16	Serial	0.5024	0.0847	Success

Table 5.5: *p-value*'s and conclusion (DIEHARD Suite)

TSN	Test Name	*p-value*	Conclusion
1	Birthday Spacing	0.3213	Success
2	Overlapping 5-Permutation	0.1489	Success
3	Binary Rank (31×31 Matrices)	0.8363	Success
4	Binary Rank (32×32 Matrices)	0.3227	Success
5	Binary Rank (6×8 Matrices)	0.4506	Success
6	Bitstream	0.5638	Success
7	Overlapping-Pairs-Sparse-Occupancy	0.5704	Success
8	Overlapping-Quadruples-Sparse-Occupancy	0.5193	Success
9	DNA	0.4021	Success
10	Count-The-1's (on stream of bytes)	0.2279	Success
11	Count-The-1's (on specific bytes)	0.5343	Success
12	Parking Lot	0.9362	Success
13	Minimum Distance	0.2327	Success
14	3D Spheres	0.3366	Success
15	Squeeze	0.4369	Success
16	Overlapping Sums	0.6415	Success
17	Runs	0.5000	Success
18	Craps	0.5579	Success

Chapter 6

Security Analysis of Mmohocc Cipher

In this chapter we investigate some cryptographic properties of the Mmohocc cipher: period, auto- and cross-correlations, the mixture of Markov processes, and spatiotemporal effects. The cipher is resistant to the related-key-IV attacks, Time/Memory/Data tradeoff attacks, algebraic attacks, and chosen-text attacks. The encryption speed is comparable with RC4. This chapter is based on [122].

6.1 Cryptographic properties

6.1.1 Period

We all know that any chaotic orbit will eventually become periodic in computer realization with a finite precision. However, when we have many chaotic maps and

each map generates a sufficiently large number of chaotic orbits, the period of the mixture of the orbits will be extremely large.

State space S in bits is $\#maps \times \#orbits \times 32$. If $\#maps = 16$, $\#orbits = 20 \sim 35$, then in average the state space is $16 \times 27 \times 32 = 13824$-bit. This ensures that the period of the keystream is extremely big. But the exact period of Mmohocc is difficult to predict, it also depends on the selection of map coefficients and other coefficients in the implementation, for example, the "hopping skew" to $offset[i]$ (0.19, 0.29, or other small numbers bigger than 0.01). The average period of the keystream is approximately 2^{13824}.

6.1.2 Auto- and cross-correlations

The cross-correlation [115] is a measure of similarity between two bit sequences. It is a function of the relative time/lag between the sequences. Large cross-correlations between sequences mean that the sequences are very similar to each other and usually not statistically independent. The auto-correlation of a sequence s_n gives the amount of similarity between the sequence s_n and a shift of s_n by m positions. It is simply the cross-correlation of the sequence against a time-shifted version of itself. In general it is desirable to have as small values as possible for the correlation (except for $m = 0$ in the auto-correlation) to ensure the independence of sequences. I.e, the auto- and cross-correlations are δ-like function and close-to-zero, respectively.

If we treat the keystreams produced by the Mmohocc cipher as bit sequences we can examine the similarities between the sequences by calculating the auto- and cross-correlations. We take bit sequence length N, sliding from $-N/2$ to $+N/2$.

We use the following correlation definition formulas [115] to calculate the auto- and cross-correlations. The auto-correlation AC at lag m for the sequence $s_n(i)$ is (here $n+m$ is performed modulo N)

$$AC_i(m) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \left[s_n(i) - \mu(i)\right] \left[s_{n+m}(i) - \mu(i)\right] .$$

And the cross-correlation CC at lag m for the sequences $s_n(i)$ and $s_n(j)$ is

$$CC_{ij}(m) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \left[s_n(i) - \mu(i)\right] \left[s_{n+m}(j) - \mu(j)\right] .$$

where $\mu(i)$ and $\mu(j)$ are the averages (expected values) for $s_n(i)$ and $s_n(j)$, respectively. Here are two examples from the experiment and we find that the auto-correlation is δ-like function and cross-correlation is close-to-zero everywhere.

Figure 6.1 and Figure 6.2 are obtained by arbitrarily choosing two bit sequence segments for $N = 2048$ bits and $N = 8192$ bits from two long different keystreams. The average auto-correlation $AC(m)$ is very close to zero everywhere except at $m = 0$, i.e. a δ-like function. The cross-correlation $CC(m)$ is close-to-zero everywhere. Also we can see that the longer the sequences, the less correlations between the sequences.

6.1.3 Mixture of Markov processes and spatiotemporal effects

The cipher system is deterministic as long as the nonlinear equations are defined, however, it is a rather complicated mixture of multiple Markov processes. In the

Mmohocc-128 cipher there are eight chaotic maps

$$s_{i,n+1} = F_0(s_{i,n}), \quad t_{i,n+1} = F_1(t_{i,n}), \quad u_{i,n+1} = F_2(u_{i,n}), \quad v_{i,n+1} = F_3(v_{i,n}),$$
$$w_{i,n+1} = F_4(w_{i,n}), \quad x_{i,n+1} = F_5(x_{i,n}), \quad y_{i,n+1} = F_6(y_{i,n}), \quad z_{i,n+1} = F_7(z_{i,n}),$$

where $F_0 \sim F_7$ are the chaotic maps, n is time index and i is orbit index. The sequence of iterative equations produces a stochastic process, a mixture of eight Markov processes due to hopping and mixing process in the finite state space. This makes the cycle of the mixed Markov process even longer [35].

For instance, when the system iterates to chaotic map F_1, according to the subkey for the map $\#orbits = 11$, $\#samples = 17$, and its hopping-pattern is {7, 3, 9, 1, 6, 10, 4, 5, 2, 11, 8}. When it comes to this map the Mmohocc picks 17 points on orbit 7 first ($t_{7,0},\ t_{7,1},\ \ldots,\ t_{7,16}$), then another 17 points on orbit 3 ($t_{3,0},\ t_{3,1},\ \ldots,\ t_{3,16}$), on orbit 9 ($t_{9,0},\ t_{9,1},\ \ldots,\ t_{9,16}$), ..., on orbit 8 ($t_{8,0},\ t_{8,1},\ \ldots,\ t_{8,16}$) in this order. If we put them in a 2-D array, it will be a $\#orbits$ rows and $\#samples$ columns points array

$$\begin{pmatrix} t_{7,0} & t_{7,1} & \ldots & t_{7,16} \\ t_{3,0} & t_{3,1} & \ldots & t_{3,16} \\ \vdots & \vdots & \ddots & \vdots \\ t_{8,0} & t_{8,1} & \ldots & t_{8,16} \end{pmatrix}.$$

Or we can represent it in a 1-D array as

$$(t_{7,0}, t_{7,1}, \ldots, t_{7,16}, t_{3,0}, t_{3,1}, \ldots, t_{3,16}, \ldots, t_{8,0}, t_{8,1}, \ldots, t_{8,16}).$$

Then according to the aforementioned random number extraction method the Mmohocc generates ($\#orbits \times \#samples$) random numbers. They constitute a segment of the random numbers. All segments constitute a much complicated keystream

with very large period. On different map $\#orbits$, $\#samples$, and the corresponding hopping-pattern are different. The three factors ($\#maps$, $\#orbits$, and $\#samples$) determine the random number extraction base array.

If we treat each chaotic orbit as one dimension in space, then our Mmohocc cipher will be a spatiotemporal dynamical system. It exhibits chaotic properties in both space and time.

Multiple maps and multiple orbits spread chaotic properties to multi-dimensional directions. This property makes the Mmohocc cipher a collection of multiple spatiotemporal chaotic systems without coupling. In general, a spatiotemporal chaotic system is described by CML (Coupled Map Lattice) model [116]

$$x_{n+1}(i) = (1-\varepsilon)f(x_n(i)) + \varepsilon f(x_n(i-1))$$

where n is the time index, i =1, 2,..., N is the space/lattice index (N is the lattice size), and ε the coupling coefficient. The f is mapping function, i.e., the chaotic map. For $\varepsilon \to 0$, i.e.

$$x_{n+1}(i) = f(x_n(i))$$

there is no coupling. Therefore local neighborhoods have no influence on the behavior of the CML. This situation represents independently operating local orbits at each lattice site. In the multi-map orbit hopping situation there are multiple mapping functions F_0, F_1, ..., F_k. The size of the lattice N (number of orbits) is a variable and it depends on the key.

Besides the speed advantage by parallel computation the spatiotemporal chaotic system further enhances time and space mixing property.

6.2 Security against attacks

An IV allows a stream cipher to produce a unique keystream independent from others produced by the same encryption key, without having to go through a re-keying process [115]. In the key scheduling procedure the hash function guarantees that the key plus IV is chopped into subkeys in a complex manner. Even if a simple incremental IV is used, the cipher is still resistant to related-IV attacks. This cipher is also immune to the subkey guessing and related-key attacks, because the key is not simply applied to chaotic maps; instead it is expanded into subkeys by secure hash functions. Obtaining a few subkeys has no help on deducing the key unless the key scheduling procedure is totally insecure. The key scheduling may cost a little bit more time, however it is just a one-time initialization procedure.

The Mmohocc cipher is resistant to Time/Memory/Data (TMD) tradeoff attack. Such an attack has two phases: in the preprocessing phase an adversary explores the structure of the stream cipher and summarizes his findings in large tables. In the real attack phase the adversary uses the pre-computed tables and actual data produced from a particular unknown key to find the secret key quickly [13]. The TMD complexity of the cipher is much larger than 2^{128}. The large number of states (period 2^{13824} is large enough) eliminates the threat of the attack. Also from [6] if the state space is much larger than the number of secret keys, then this attack will provide no improvement. The extended Time/Memory/Key tradeoff attack [12] can be prevented by using keys longer than 128-bit.

Algebraic attacks on stream ciphers [26, 27, 4, 5] recover the key by solving an overdefined system of multivariate equations. Such attacks can break LFSR-based

stream ciphers with linear feedback when the output is obtained by a Boolean function. The Mmohocc is a chaotic cipher and the transition function has strong non-linear properties and it is immune to this kind of attacks by nature. Besides, the cipher is using an S-Box-like filtering function with 32-bit input and 16-bit output.

There are other attacks to be considered here. The ciphertext-only attack tries to use the output cipher to recover the key; it is in some sense equivalent to brute force attack. The key space for the cipher is large enough, at least 2^{128}. So it is not feasible to apply these attacks to the cipher.

A known-plaintext attack is one in which an adversary knows one or more plaintexts along with the corresponding ciphertexts. Chosen-plaintext/ciphertext attack implies that an adversary has access to the encryption/decryption equipment, therefore s/he can choose any plaintexts/ciphertexts and get the corresponding ciphertexts/plaintexts. The objective of those attacks is to deduce the key [74].

Binary representations of variables $a,\ b,\ c,\ d,\ p$, and q involved in filtering/output functions in Figure 5.2 (in Section 5.1.4) are $a_0a_1 \dots a_7,\ b_0b_1 \dots b_7,\ \dots,\ q_0q_1 \dots q_7$. Since $p = a \oplus c$ and $q = b \oplus d$, the following equations hold:

$$\begin{cases} p_0 = a_0 \oplus c_0, & q_0 = b_0 \oplus d_0, \\ p_1 = a_1 \oplus c_1, & q_1 = b_1 \oplus d_1, \\ \quad \vdots & \quad \vdots \\ p_7 = a_7 \oplus c_7, & q_7 = b_7 \oplus d_7. \end{cases}$$

In chosen-text attacks, the fact that an adversary knows a keystream number does not mean that s/he can solve the above equations to get the state, since 16 equations are not sufficient to obtain 32 unknowns. This S-box-like filtering function is

a compression function with 200% compress ratio (32-bit-in/16-bit-out). This satisfies that the filtering function must not leak too much information on the internal state [21]. Even an adversary can completely recover all the subkeys from these attacks, s/he still has no better way than brute force to obtain the key, since reversing the key scheduling procedure is hard assuming the hash is a good one-way function.

Often it is possible to analyze the behavior of a subsystem in dependence of a certain parameter. When key bits are directly used as parameter bits, the determination of this parameter reduces the key space by the corresponding number of bits [49]. The key scheduling (in Section 5.1.2) procedure turns the secret key into 8 (or 16) subkeys in a very complicated manner, so we avoid this problem.

In distinguishing attack to stream ciphers an adversary can distinguish between the keystream of a particular cipher and the output of a truly random number generator with a non-negligible probability. In practice distinguishing attack is not a security issue [95], therefore we do not consider it as a threat to the cipher.

6.2.1 Speed comparison with RC4

In comparing with stream cipher RC4, Table 6.1 shows that the Mmohocc cipher has similar encryption speed. The experiment is conducted on a Pentium-4 3.4GHz PC. The Mmohocc is using more complicated algorithm than RC4 and without key scheduling algorithm weaknesses found in RC4 [34], and without much software optimization effort. RC4 was implemented in Visual C++ 6.0.

Table 6.1: Speed comparison with RC4

	584KB	11,200KB	22,400KB	145,600KB
RC4	0.047s	1.093s	2.109s	11.032s
Mmohocc	0.078s	1.562s	3.11s	19.641s

6.3 Conclusion

From security analysis, we know that the Mmohocc cipher has extremely long period, δ-like auto-correlations, close-to-zero cross-correlations, and it is a stochastic process of mixture of multiple Markov processes. The cipher is resistant to related-key-IV attacks, Time/Memory/Data tradeoff attacks, algebraic attacks, and chosen-text attacks. In the respect of performance the encryption speed is similar to RC4's.

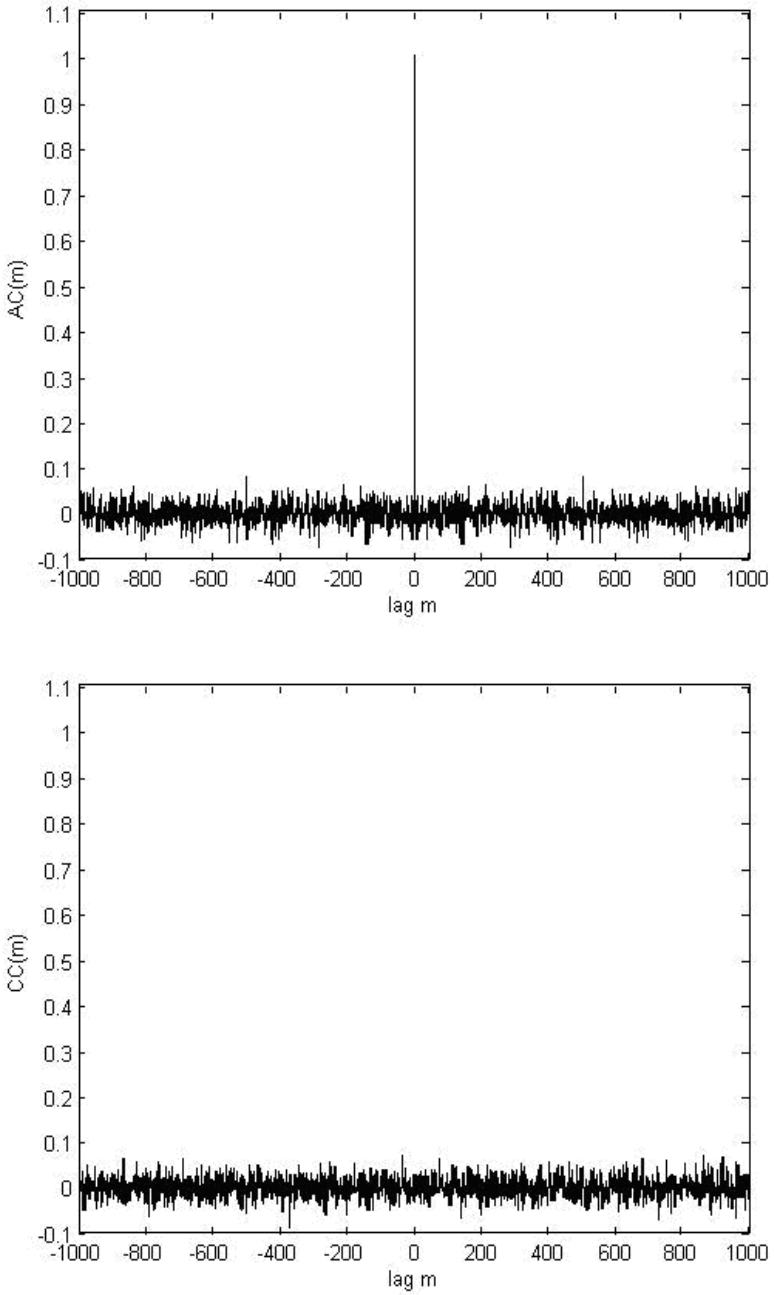


Figure 6.1: (T) Auto-correlation and (B) cross-correlation for $N = 2048$

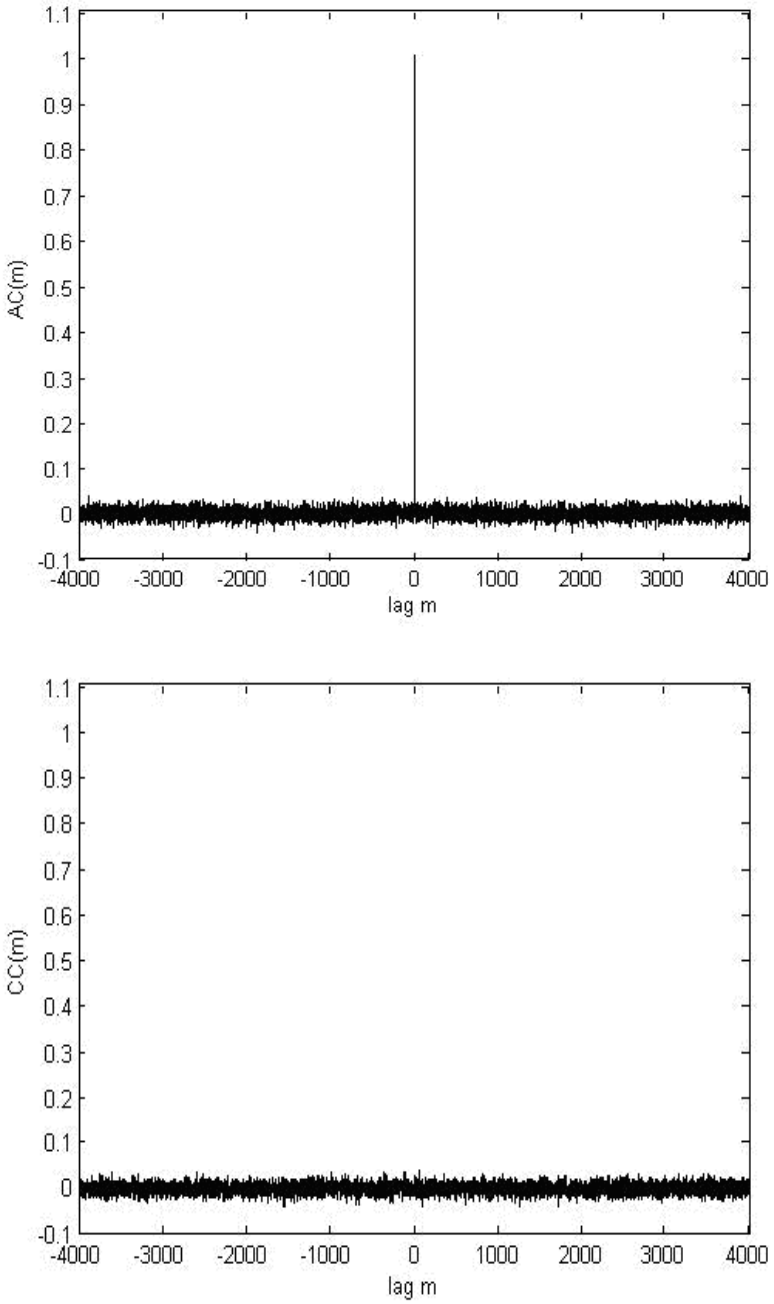


Figure 6.2: (T) Auto-correlation and (B) cross-correlation for $N = 8192$

Chapter 7

Mmohom in LFSR-Based Stream Ciphers

In this chapter we discuss Linear Feedback Shift Register's background, properties, and the reason why people like to use LFSRs in cipher designs. Then we introduce the Mmohom-LFSR-based stream cipher and its features. There are numerous known attacks, here we focus on a newly developed algebraic attack. We demonstrate that the Mmohom-LFSR-based cipher is resistant against this kind of attacks.

7.1 Linear feedback shift register basics

Linear feedback shift registers are widely used in pseudorandom sequence generators in both cryptography and coding theory. A LFSR is shown in Figure 7.1. It satisfies the following properties [51, 100, 109, 115]:

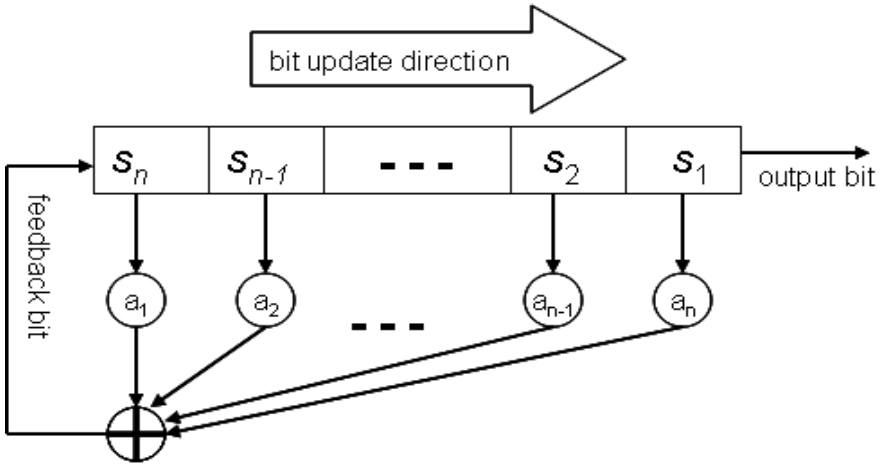


Figure 7.1: General linear feedback shift register.

- The n taps $a_1, a_2, \ldots, a_n$ on the cells of an n-state LFSR correspond to a tap polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + 1$ with coefficients $a_i \in \{0, 1\}$.

- The size of the smallest LFSR that generates a given periodic sequence is called the linear complexity – a measure of the cryptographic security of the sequence.

- A maximal length LFSR implies that all possible $2^n - 1$ states are obtained (excluding all-zero state), the sequence generated is also called m-sequence.

- A tap polynomial $p(x)$ of length n is primitive if (1) it is irreducible, and (2) $p(x)$ divides $x^{2^n-1} + 1$, but not $x^d + 1$ for any d that divides $2^n - 1$.

- The tap polynomial of a maximal period LFSR of length n is determined uniquely by any $2n$ consecutive terms of the output sequence.

- Using the output of a single LFSR as the keystream in a stream cipher is vulnerable to a known-plaintext attack [108].

Despite the last two properties, LFSRs are still widely used in cryptography. This is mainly because they can be easily implemented in hardware. Also they can be implemented relatively fast in software if we use Galois LFSRs (LFSR in Galois configuration) [89]. The Galois form is more efficient as the XOR operations can be implemented a word at a time [115]. By combining the output of several LFSRs it gives more secure nonlinear keystreams to be used in stream ciphers.

7.2 Combining LFSRs by Mmohom mechanism

We explain how to combine bunches of LFSRs into a stronger stream cipher by using Mmohom mechanism. Here the "maps" are the primitive polynomials of the LFSRs.

We use Mmohom mechanism in a slightly different way. We do not generate multiple orbits from one primitive polynomial (used by the LFSR) since resetting a LFSR is slow. Instead we use $\#orbits$ LFSRs in parallel for one primitive polynomial and set them to different initial states according to corresponding orbits. Given the $\#maps$, the total number of LFSRs $\#maps \times \#orbits$ is a variable of $\#orbits$ (which is determined by the key). There are $\#maps$ kinds of LFSRs, each kind with length n_i $(i = 1, 2, \ldots, \#maps)$ is configured with a primitive polynomial so that it generates a maximal cycle sequence of $2^{n_i} - 1$ by its own. And the overall state space is $n = \sum_{i=1}^{\#maps} n_i$.

Figure 7.3 is an example of Mmohom-LFSR-based stream cipher. There are four kinds of primitive polynomial maps, each can have up to i (j, u, v) orbits. As many as $i + j + u + v$ LFSRs are setup at the initialization. The number of LFSRs for each map and the total number of LFSRs for the cipher are determined by the $\#orbits$;

for an individual map *#orbits* in turn is determined by the key. The key handling & parameter control part is similar to Mmohocc cipher. According to hopping pattern Mmohom multiplexer (acting as a m-to-1 "multiplexer" located in the dashed circle) controls the way of hopping among those LFSRs and the duration of staying on a particular LFSR.

The features of the Mmohom-LFSR-based stream cipher are as follows:

- Due to the Mmohom mechanism one "LFSR" (a group of LFSRs implemented in parallel for the same map) can jump between non-contiguous states (without having to step through all the intermediate states).
- Mmohom mechanism adds another nonlinearity to the stream cipher. Together with nonlinear output function, the cipher has double nonlinear effect.
- The cipher can be implemented in hardware and software (if memory is not a concern) efficiently. The speed of the encryption/decryption should be fast.
- Long period: about $N = 2^n$, even longer pseudo-cycle: about $N \times (i+j+u+v)$.
- The double nonlinearity makes the cipher stronger against linear attacks, correlation attacks, and algebraic attacks.
- The cipher creates a thorough spatiotemporal mixing, which makes it almost impossible for an adversary to cryptanalyze the cipher.
- This stream cipher could be used in light-weight encryption, like RFID tags, PDAs, and other gadgets.

7.3 Against algebraic attacks

Algebraic attacks are a new class of attacks against stream ciphers, they are known to be faster and more efficient against certain ciphers (especially LFSR-based) than all other kinds of attacks [4, 5, 25, 26, 27, 28, 115]. We analyze the Mmohom-LFSR-based stream cipher to show that the cipher is resistant against this attack.

We use Figure 7.2 to demonstrate how algebraic attack works. It is a stream cipher implemented by combining k LFSRs by a non-linear combiner $\mathbf{F}$. Each LFSR has length n_i ($i = 1 \sim k$), the total state space for k LFSRs is $n = \sum_{i=1}^{k} n_i$. The combiner $\mathbf{F}$ generates a keystream bit $z_t = \mathbf{F}(x^{(t)})$ given the input state $x^{(t)}$, which consists of k parallel bits from k LFSRs.

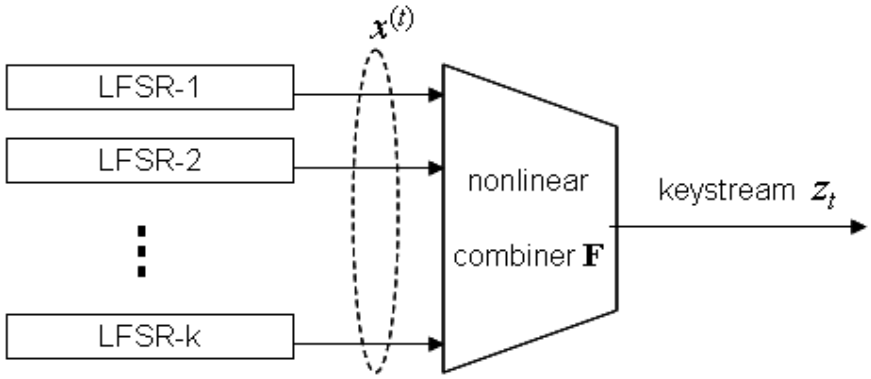


Figure 7.2: Algebraic attack analysis to a LFSR combiner.

Algebraic attack tries to use nonlinear equations to reconstruct the seed (i.e. the key) of a stream cipher. Each equation is built up by $\mathbf{F}(x^{(t)}) \oplus z_t = 0$ with degree d (here $d = deg(\mathbf{F})$).

The Mmohom-LFSR-based stream cipher proposed in the previous section has

double nonlinearity property, which makes the cipher resistant against the algebraic attacks. The first layer of the nonlinearity is due to the Mmohom mechanism which breaks the linearity of the individual LFSR (i.e. it makes LFSRs jumping among non-contiguous states). The second layer of nonlinearity comes from the nonlinear output function $\mathbf{F}$ itself, the degree of the nonlinearity is $d = deg(\mathbf{F})$.

We use the quadratic offset in the Mmohom mechanism (a.k.a. quadratic Mmohom), that is the next orbit is created by squaring the current state value (or the next orbit starts from cubical of the current state value). If the key length is $n = \sum_{i=1}^{\#maps} n_i$ (n_i is the length of LFSR i), then the initial seed has n unknowns. Without the Mmohom a nonlinear equation of degree d (due to $\mathbf{F}$) has up to [4]

$$N_d = \sum_{i=1}^{d} \binom{n}{i} \approx O(n^d)$$

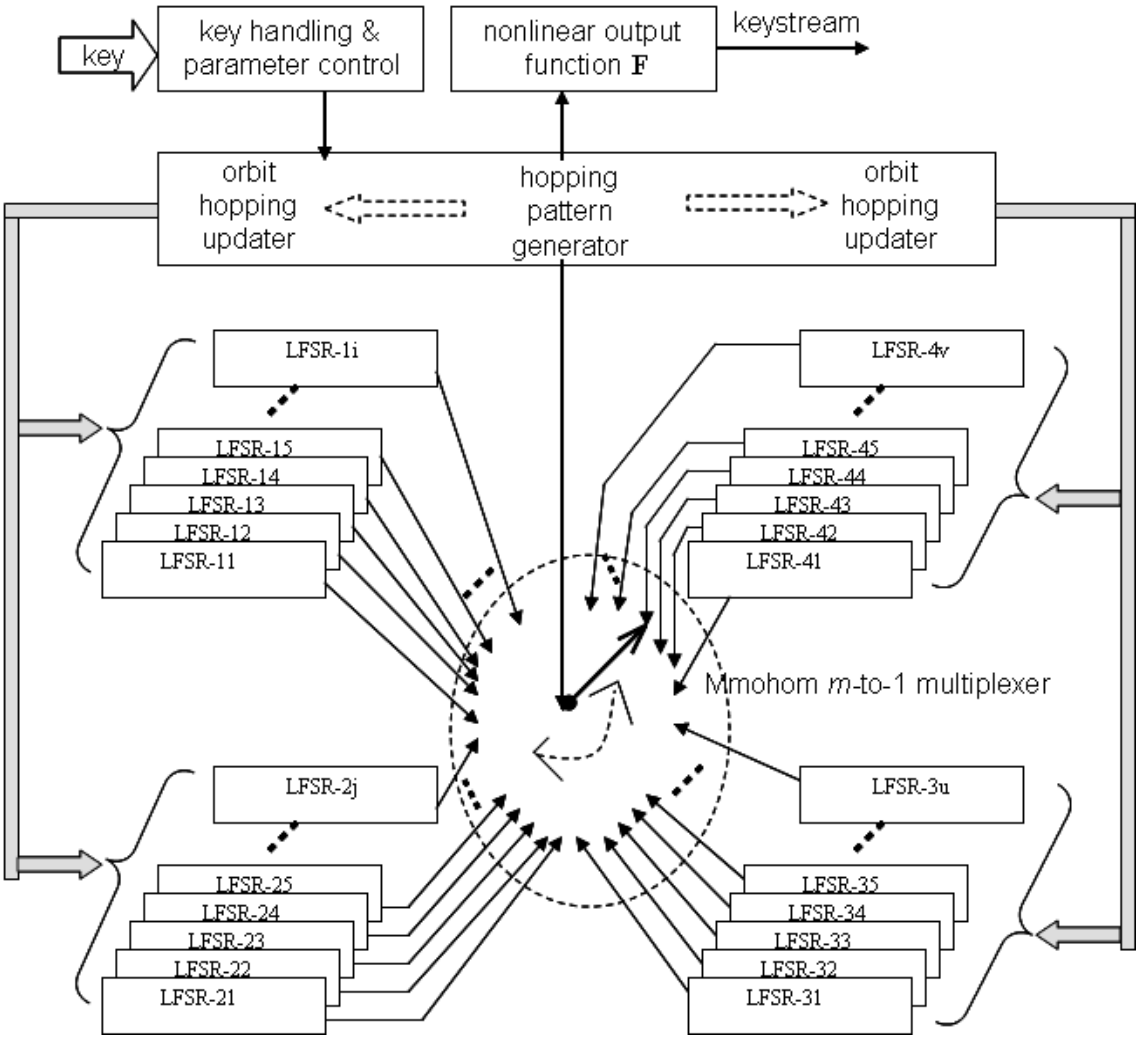
monomials. Now the quadratic Mmohom mechanism (*hpsn* changes by following a quadratic rule) is in tandem with the nonlinear output function $\mathbf{F}$, and the number of monomials of nonlinear equations can grow up to

$$N'_d = \sum_{i=1}^{2d} \binom{n}{i} \approx O(n^{2d}).$$

The exponential increase of the number of monomials makes the algebraic attack impractical.

7.4 Conclusion

We propose the Mmohom-LFSR-based stream ciphers and show that these ciphers are resistant against the algebraic attacks. Although we have not implemented the Mmohom-LFSR-based stream cipher, we give a design diagram which demonstrates the essential features of the system.



The 4 primitive polynomial maps have at most i-, j-, u-, v-orbits, respectively (depending on key). At initialization stage, the orbit hopping updater updates the states of LFSRs according to the instruction from the hopping pattern generator. The many-to-1 (m-to-1) multiplexer (the dashed circle in the center) controlled by the hopping pattern generator, switches to corresponding LFSR.

Figure 7.3: Schematic diagram of Mmohom-LFSR-based stream cipher.

Chapter 8

Mmohoct Stream Ciphers

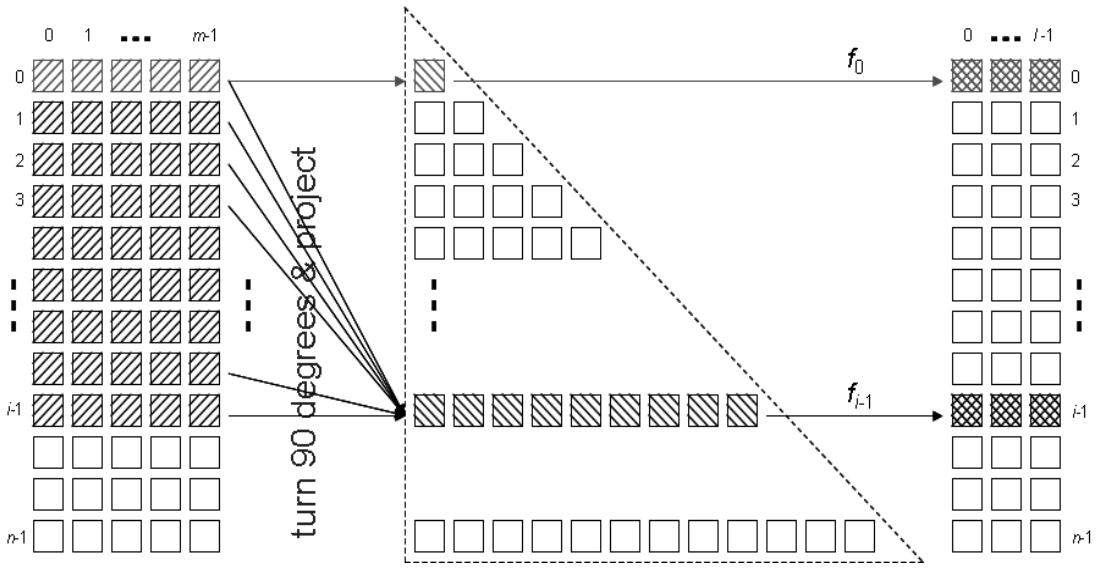
In this chapter we present T-functions and their properties. Some algebraic structures have been found in these functions. We apply our Mmohom mechanism to design the cipher Mmohoct: Multi-map orbit hopping cipher by T-functions. Two experimental Mmohoct ciphers are implemented. (Note: 'T-map' is a synonym for 'T-function' in this text.)

8.1 Introduction to T-functions

Klimov and Shamir [52, 53, 54, 55, 56] proposed the T-functions (Triangular-functions) as a new class of invertible maps. A T-function f is a mapping from m n-bit words to l n-bit words (a collection of memory words). Intuitively it can be illustrated in Figure 8.1, in which n-bit words are arranged vertically. Each i-th bit of any of the output words (for $0 \leqslant i < n$) depends only on bits $0, 1, \ldots, i-1$ of the input words. The f_i is the short format for $[f([x]_0, [x]_1, \ldots, [x]_i)]_i$. In other words, i-th output bit

depends only on the least significant i input bits. Each layer on the input side consists of m bits.

In computer processors most arithmetic operations ($+$, $-$, $\times$) and logic operations (OR, AND, XOR, NOT, ...) are T-functions, and hence so are their compositions. By using various combinations of these operations Klimov [52] designed a variety of T-functions with maximal single cycle property. Such T-functions are efficient in software implementation and can be used as alternatives for LFSRs in the stream ciphers.



The name of "T-function" is due to the "Triangular" shape. Vertical m n-bit words as input on left, T-function in the middle, vertical l n-bit words as output on right. The collection of words are chopped into layers, each layer consists of m bits on the input side, and l bits on the output side.

Figure 8.1: T-function diagram illustration.

Hong, Moon et al [43, 44, 45, 80] proposed a new construction of single-cycle T-function **T**, which uses the S-box property, for their TSC family stream ciphers as

$\mathbf{T}(\mathbf{x}) = \mathbf{x} \oplus \{\alpha(\mathbf{x}) \wedge (\mathbf{x} \oplus \mathbf{S}(\mathbf{x}))\}$. The 128-bit state space is represented by $\mathbf{x} = (x_k)_{k=0}^{3}$, x_k is a 32-bit word (integer). Here α is an odd parameter[1]: $\alpha(\mathbf{x}) = p \oplus (p+1) \oplus 2s$, $p = x_0 \vee x_1 \vee x_2 \vee x_3$, $s = x_0 + x_1 + x_2 + x_3$, $\mathbf{S}$ is single cycle S-box: $\mathbf{x} \rightarrow \mathbf{x} \oplus \mathbf{S}(\mathbf{x})$.

Two T-functions used for stream ciphers in [52], [70] and [110] are listed here:

$$f : x \rightarrow x \oplus (x^2 \vee 5)(mod\ 2^n),$$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 & \oplus & s & \oplus & 2x_1x_2 \\ x_1 & \oplus & (s \wedge \alpha_0) & \oplus & 2x_2x_3 \\ x_2 & \oplus & (s \wedge \alpha_1) & \oplus & 2x_3x_0 \\ x_3 & \oplus & (s \wedge \alpha_2) & \oplus & 2x_0x_1 \end{pmatrix},$$

where $\alpha_0 = x_0$, $\alpha_1 = \alpha_0 \wedge x_1$, $\alpha_2 = \alpha_1 \wedge x_2$, $\alpha_3 = \alpha_2 \wedge x_3$, $s = (\alpha_3 + C) \oplus \alpha_3$, C is any odd constant.

However, T-functions are not as "non-linear" or "non-algebraic" as they are claimed to be. Recently Molland [78, 79] found some algebraic structures in KS T-functions. TSC [45, 80] stream ciphers have been cryptanalyzed by using linear approximations of the algorithms in [81]. Although T-functions don't have as strong linearity as LFSRs, still we need some mixing mechanisms to destroy their algebraic structures. This is the initial motivation to introduce Mmohom mechanism into T-function based stream ciphers.

[1] a parameter $r(x)$ is a T-function that $[r(x)]_i$ does not depend on bit-slice $[x]_i$.

8.2 Multi-map orbit hopping ciphers by T-maps

In this section, we present two stream ciphers that are based on Mmohom mechanism and multiple invertible single cycle T-maps. We call this type of ciphers Mmohoct (short for Multi-Map Orbit HOpping Ciphers by T-maps). The first proposal is a direct transformation of Mmohocc cipher except that the maps are T-maps instead, the goal is to destroy the algebraic patterns in these quadratic T-maps by using Mmohom mechanism. The second proposal is a TSC-like multi-word stream cipher, we expand 4-word TSC family [80] of stream ciphers to 8-, 16-, and 32-word, the state space expanded tremendously. Since T-map itself can be viewed as a collection of multiple maps, in this sense there are already multiple maps involved in the system, we implement multiple orbits by changing initial state with an offset.

We use multiple T-maps, with each of them having a single cycle permutation on the state space, to construct a kind of one-way permutation to overcome the weakness of the single T-map permutation. The Mmohoct also offers countermeasures against "insufficient intermixing of higher and lower bits of stand-alone T-functions [77]".

8.2.1 Single-word Mmohoct

For demonstration purpose our experiments use maps up to quadratic terms with the least number of machine operations for efficiency. Here are some candidates of

quadratic maps with proved single cycles [55]:

$$f(\mathbf{x}) = 1 + \mathbf{x} + 4\mathbf{x}^2 \ mod\ 2^n, \quad f(\mathbf{x}) = 1 + 3\mathbf{x} + 2\mathbf{x}^2 \ mod\ 2^n,$$
$$f(\mathbf{x}) = 1 - \mathbf{x} + 6\mathbf{x}^2 \ mod\ 2^n, \quad f(\mathbf{x}) = \mathbf{x} \pm \mathbf{x}^2 \vee \mathbf{C} \ mod\ 2^n,$$
$$f(\mathbf{x}) = \mathbf{x} \pm (\mathbf{x} \vee \mathbf{D})^2 \ mod\ 2^n, \quad f(\mathbf{x}) = \mathbf{x} + r(\mathbf{x}) \ mod\ 2^n,$$

where $r(\mathbf{x})$ is an even parameter (see Theorem 2 in [54] for detail). $\mathbf{C}$ is a constant with the LSB and the third LSB are 1, i.e., $\mathbf{C} \equiv 5 \ or\ 7 \ mod\ 8$ (such as $\mathbf{C} = 5 = 00000101_2$, or $\mathbf{C} = 39 = 00100111_2$). And $\mathbf{D}$ is any odd constant.

In this design, multiple T-maps are used and the state space is 32-bit (64-bit, or 128-bit for future processors), a single word-size. Multiple T-maps and the Mmohom mechanism are used to destroy possible linear properties that a single T-map may have. The hopping among multiple orbits lets the information propagate from left to right. These T-map candidates are used to replace the chaotic maps in Mmohocc cipher (see Chapter 4). Here we use the following 4 T-maps to build a map bank:

$$f(\mathbf{x}) = 1 + \mathbf{x} + 4\mathbf{x}^2 \ mod\ 2^n,$$
$$f(\mathbf{x}) = 1 + \mathbf{x} + 8\mathbf{x}^2 \ mod\ 2^n,$$
$$f(\mathbf{x}) = \mathbf{x} + \mathbf{x}^2 \vee 5 \ mod\ 2^n,$$
$$f(\mathbf{x}) = \mathbf{x} + \mathbf{x}^2 \vee 21 \ mod\ 2^n.$$

For key handling and initialization vectors, we follow the same procedure as described in Chapter 5, except now we use 32-bit key and 32-bit IV since the state space is only 32-bit. We usually have key space much less than state space, here we just use this as a "toy" cipher to evaluate the effect of Mmohom mechanism in T-map based stream cipher.

Key handling procedure makes the key into four subkeys to control the four T-maps. Recall the subkey structure in Chapter 4, here the *#settles* field is eliminated because there is no need to settle the orbit. Each field of the subkey has different meaning from the chaotic ciphers. Figure 8.2 is the subkey structure for the Mmohoct cipher.

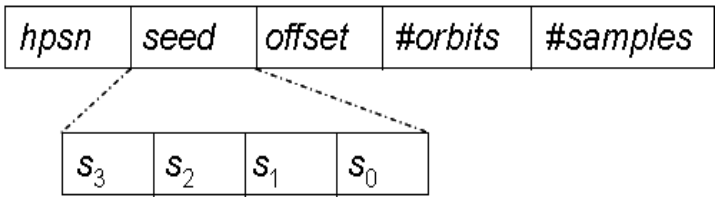


Figure 8.2: Subkey structure and subkey fields.

In Figure 8.2 a subkey has the fields *hpsn seed offset #orbits #samples* (the concatenation of the fields makes a subkey). The length of a subkey is 64-bit, and each field is an unsigned integer. Here we explain each field and its size.

***hpsn*:** hopping-pattern serial number, 8-bit: it has the same functionality as in Chapter 5, is used to locate a predefined pseudorandom hopping sequence.

***seed*:** the initial state value of a T-map, the length is the same as the word-size 32-bit: it consists of 4 bytes: s_3, s_2, s_1, s_0 with s_3 the most significant byte (MSB), and s_0 the least significant byte (LSB). Unlike in a chaotic map, we don't have to convert the field into a floating number.

***offset*:** the shift value to change the initial seed in order to generate the next orbit, 8-bit: for odd orbits, we make a new seed by adding the *offset* into the MSB s_3 of the seed (XOR indeed), then swap it with the LSB s_0; for even orbits, we

make a new seed by adding the *offset* into the s_2 of the seed (XOR indeed), then swap it with the s_1. By doing so, we move the MSB to LSB, it's equivalent to propagate the information from left to right of the state.

***#orbits*:** the number of orbits generated during each use of a map, 8-bit: it is used by plus 1 $mod\ 2^8$ to handle $\#orbits = 0$ situation.

***#samples*:** the number of sample points taken from an orbit, 8-bit: it decides how long to stay on a particular orbit. Again, it is used by plus 1 $mod\ 2^8$ to handle $\#samples = 0$ situation.

Output function should produce randomness, hide the internal state, and avoid known weaknesses [74]. We use four T-maps to update the internal states of the cipher. For each state a 16-bit keystream is generated by using the non-linear output function as the one in Chapter 5 (Figure 5.1). The squaring maps are biased [60], for example words $y = 2^{\frac{n-8}{2}} \cdot i^2$ (for $i \in [0, 3]$) occur two times more than uniform random words. Some linear properties and algebraic patterns have been found [78, 79]. For example for T-map $f(\mathbf{x}) = \mathbf{x} + \mathbf{x}^2 \vee \mathbf{C}\ mod\ 2^n$, the following linear equation holds: $x_{i+2^{j-1},j} = x_{i,j} + x_{i,j-1} + a_2 x_{i,1} + a_1 x_{i,0} + a_0\ mod\ 2$, here $x_{i,j}$ is the i-th iterate of the initial word $\mathbf{x}_0 = (x_{0,n-1}, x_{0,n-2}, \ldots, x_{0,1}, x_{0,0})$, j is the j-th bit of the word $\mathbf{x}_i$. And a_2, a_1, a_0 are determined by the constant $\mathbf{C}$.

Table 8.1 gives the encryption speeds for Mmohoct single-word version. The test is conducted on a Pentium-4 3.4GHz PC. The encryption speed of the Mmohoct is slower than RC4 (113 MB/s) and close to AES-CTR (58 MB/s) in Crypto++ 5.2.1 benchmarks [30]. There are reasons why Mmohoct is slower: it involves multiple nonlinear maps and multiplications (expensive operations). We can improve

the Mmohoct's speed by: doing code optimization, replacing file input/output with buffer operations.

Table 8.1: Encryption speed of Mmohoct single-word version

File Size (KB)	584	11,200	22,400	145,600
Time Taken (s)	0.014	0.125	1.105	5.327
Speed (MB/s)	43	92	21	28

8.2.2 Multi-word Mmohoct

Our second proposal is based on two types of T-maps with maximal single cycles. According to the key the maps are used alternatively and each of them generates many orbits. The Mmohoct cipher gives pseudorandom numbers from those orbits. In the following text, we'll give Mmohoct block diagram and the construction of the T-maps, nonlinear output function of the cipher, then details on the key and IV handling, performance comparison, and safety analysis at the end.

Construction of T-maps

Some notes about constructing T-maps: (1) We use the Mmohom mechanism with multiple T-maps, multiple orbits to destroy algebraic structures found in T-maps[78]. (2) Use 32 32-bit words to build a TSC-like T-map, the state space will be $32 \times 32 = 1024$, i.e., 2^{1024}. By doing so, every bit-slice itself is also a 32-bit word. (3) If expanding TSC to more words (bigger than four), then a larger S-box would be

needed (i.e., a larger lookup table). (4) Each individual T-function, indeed, has multiple "T-maps" involved already. For instance 4 32-bit word (x_3, x_2, x_1, x_0) based T-function, has 32 subfunctions involved in each bit iteration.

Hong et al [43, 45] proposed a new construction of single-cycle T-functions, which uses the S-box properties, for their 4-word state TSC family stream ciphers.

We use four T-functions in conjunction with a single cycle S-box (256 entries). The T-functions don't involve any nonlinear operation for better performance. We extend the state space into 8-word, therefore the cycle will be $2^{32\times 8} = 2^{256}$. In this proposal we construct two types of T-maps based on the eight word state space. The input 256-bit, denoted by $\mathbf{x}$, is a collection of eight words with 32-bit each, $\mathbf{x} = (x_j)_{j=0}^{7}$, where x_j $(j = 0, 1, \cdots, 7)$ denotes each word.

Experiments show that there is no significant speed penalty because of the expansion of the state space. The question is how to hop among multiple orbits generated by multiple T-maps.

The Mmohoct cipher (see Figure 8.3) uses four T-maps, $\mathbf{T}_i(\mathbf{x})$ $(i = 0, 1, 2, 3)$. $\mathbf{T}_0(\mathbf{x})$ and $\mathbf{T}_2(\mathbf{x})$ are TSC-like [43] T-maps, and $\mathbf{T}_1(\mathbf{x})$ and $\mathbf{T}_3(\mathbf{x})$ are derived T-maps from those used in [55].

Let's explain the TSC-like T-maps first. They do not have any multiplication and squaring, and use fast and simple logic operations and additions. Following the similar way in [80], $\alpha_0(\mathbf{x})$, $\alpha_2(\mathbf{x})$ (the two parameters), $\mathbf{T}_0(\mathbf{x})$, and $\mathbf{T}_2(\mathbf{x})$ are defined

as follows:

$$\pi(\mathbf{x}) = \bigwedge_{i=0}^{7} x_i = x_0 \wedge x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6 \wedge x_7,$$

$$\pi_0(\mathbf{x}) = \pi(\mathbf{x}) + 91212211_h \ mod\ 2^{32},$$

$$\pi_2(\mathbf{x}) = \pi(\mathbf{x}) + 49109815_h \ mod\ 2^{32},$$

$$e_0(\mathbf{x}) = x_0 + x_2 + x_4 + x_6 \ mod\ 2^{32},$$

$$e_2(\mathbf{x}) = x_1 + x_3 + x_5 + x_7 \ mod\ 2^{32},$$

$$\alpha_0(\mathbf{x}) = \pi_0(\mathbf{x}) \oplus e_0(\mathbf{x}),$$

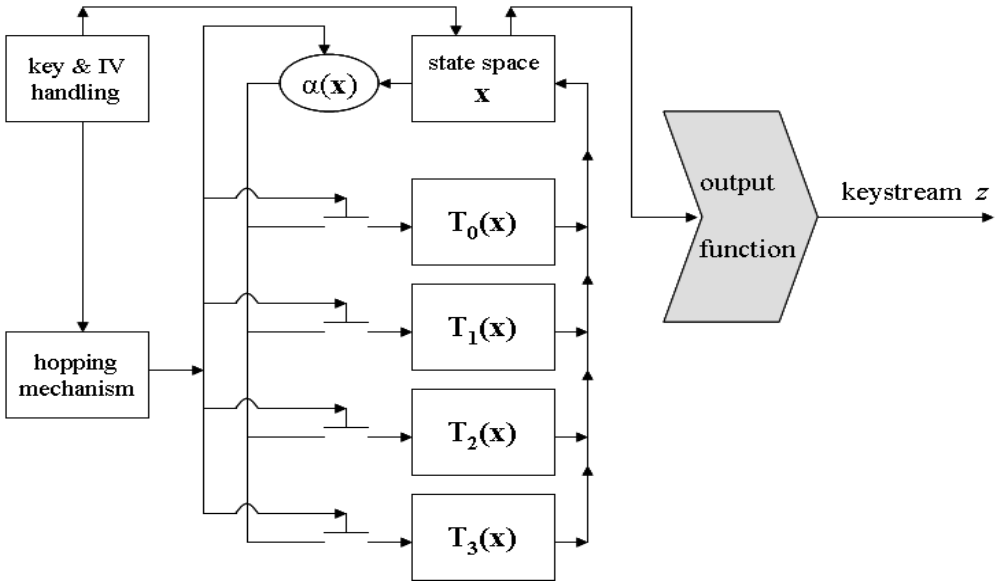
$$\alpha_2(\mathbf{x}) = \pi_2(\mathbf{x}) \oplus e_2(\mathbf{x}).$$

The above two hexadecimal constants 91212211_h and 49109815_h are chosen to help the T-map to quickly move away from the state with all columns identical [45].

$$\mathbf{T}_i(\mathbf{x}) = \mathbf{x} \oplus \{\alpha(\mathbf{x}) \wedge (\mathbf{x} \oplus S(\mathbf{x}))\},\ i = 0, 2,$$

where $S(\mathbf{x})$ is the so-called S-box. The following are the TSC-like T-maps (here $\overline{x_i}$ is the complementary of x_i, the last column of the right hand side is equivalent to the S-box):

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \oplus \alpha_{|i-2|}(\mathbf{x}) \\ x_1 \oplus \alpha_i(\mathbf{x}) \\ x_2 \oplus \alpha_{|i-2|}(\mathbf{x}) \wedge x_7 \oplus (x_1 \vee \overline{x_5}) \\ x_3 \oplus \alpha_i(\mathbf{x}) \wedge x_6 \oplus (x_0 \vee \overline{x_4}) \\ x_4 \oplus \alpha_{|i-2|}(\mathbf{x}) \wedge \overline{x_5} \wedge \overline{x_3} \oplus x_1 \oplus x_5 \wedge x_3 \wedge x_1 \oplus x_7 \wedge x_1 \\ x_5 \oplus \alpha_i(\mathbf{x}) \wedge \overline{x_4} \wedge \overline{x_2} \oplus x_0 \oplus x_4 \wedge x_1 \wedge x_0 \oplus x_6 \wedge x_0 \\ x_6 \oplus \alpha_{|i-2|}(\mathbf{x}) \wedge x_3 \oplus \overline{x_1} \wedge x_5 \\ x_7 \oplus \alpha_i(\mathbf{x}) \wedge x_2 \oplus \overline{x_0} \wedge x_4 \end{pmatrix}.$$



$\alpha(\mathbf{x})$ is the parameter generated from state space $\mathbf{x}$, and is adjusted according to the corresponding T-map $\mathbf{T}_i(\mathbf{x})$ $(i = 0, 1, 2, 3)$.

Figure 8.3: The Mmohoct diagram.

The T-maps $\mathbf{T}_1(\mathbf{x})$ and $\mathbf{T}_3(\mathbf{x})$, derived from T-maps used in [55], are:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \oplus s_j \oplus 2x_1x_2 \\ x_1 \oplus (s_j \wedge \alpha_0) \oplus 2x_2x_3 \\ x_2 \oplus (s_j \wedge \alpha_1) \oplus 2x_3x_4 \\ x_3 \oplus (s_j \wedge \alpha_2) \oplus 2x_4x_5 \\ x_4 \oplus (s_j \wedge \alpha_3) \oplus 2x_5x_6 \\ x_5 \oplus (s_j \wedge \alpha_4) \oplus 2x_6x_7 \\ x_6 \oplus (s_j \wedge \alpha_5) \oplus 2x_7x_0 \\ x_7 \oplus (s_j \wedge \alpha_6) \oplus 2x_0x_1 \end{pmatrix},$$

where parameters $\alpha_i = \bigwedge_{k=0}^{i} x_k$ $(i = 0, 1, \cdots, 7)$, $s_j = (\alpha_7 + C_j) \oplus \alpha_7 = \alpha_7 \oplus C_j$,

$C_j = 1, 3$ (where $j = 1, 3$) is an odd constant to determine T-map $\mathbf{T}_j(\mathbf{x})$.

The properties of the T-maps $\mathbf{T}_i(\mathbf{x})$ guarantee that the cycle of the Mmohoct cipher is 2^{256}. T-maps $\mathbf{T}_i(\mathbf{x})$ $(i = 0, 2)$ runs faster since they do not involve any multiplication; meanwhile $\mathbf{T}_j(\mathbf{x})$ $(j = 1, 3)$ have good diffusion property since they have multiplications.

Nonlinear output function

The output function (see Figure 8.3) produces the a 16-bit keystream z from the current state space $\mathbf{x}$. We introduce eight 8-bit temporary variables t_i $(i = 0, 1, \cdots, 7)$, four 16-bit temporary variables z_j $(j = 0, 1, 2, 3)$. Following the TSC-4 style [80], the t_i's can be defined as follows:

$$
\begin{aligned}
t_0 &= \{(x_0)_{>>8} \wedge ff_h \oplus (x_7)_{>>24} \wedge ff_h\} + x_3 \wedge ff_h, \\
t_1 &= (x_1)_{>>16} \wedge ff_h + (x_6)_{>>24} \wedge ff_h + \overline{(x_4)_{>>8} \wedge ff_h}, \\
t_2 &= (x_2)_{>>24} \wedge ff_h + (x_5)_{>>16} \wedge ff_h, \\
t_3 &= (x_3)_{>>8} \wedge ff_h + (x_4)_{>>16} \wedge ff_h, \\
t_4 &= \{(x_0)_{>>24} \wedge ff_h \wedge (x_2)_{>>16} \wedge ff_h\} + (x_5)_{>>8} \wedge ff_h, \\
t_5 &= (x_4)_{>>24} \wedge ff_h + (x_6)_{>>8} \wedge ff_h + \overline{(x_1)_{>>8} \wedge ff_h}, \\
t_6 &= (x_1)_{>>24} \wedge ff_h + (x_3)_{>>16} \wedge ff_h, \\
t_7 &= (x_5)_{>>24} \wedge ff_h + (x_7)_{>>16} \wedge ff_h,
\end{aligned}
$$

where $(x_i)_{>>k}$ is right shift x_i k-bit, ff_h is a hexadecimal number, additions are calculated modulo 128. Here are the z_j's:

$$z_0 = t_0|t_6, \quad z_1 = t_3|t_1,$$
$$z_2 = t_2|t_5, \quad z_3 = t_7|t_4,$$

where $t_i|t_j$ is the concatenation of t_i and t_j. And finally the 16-bit keystream z is:

$$z = (z_0)_{<<<6} \oplus (z_1)_{>>>7} \oplus (z_2)_{<<<5} \oplus (z_3)_{>>>2},$$

where $(z_i)_{>>>k}$ circularly right rotate z_i k-bit, and $(z_i)_{<<<k}$ circularly left rotate z_i k-bit.

State initialization, key & IV handling

Both key and IV are 128-bit long. State space $(x_i)_{i=0}^{7}$ is initialized with key $K = (K_{127}, \ \ldots, \ K_0)$ and initialization vector $IV = (IV_{127}, \ \ldots, \ IV_0)$ in the following interweaved way:

$$x_0 = (IV_{127}, \ \ldots, \ IV_{112}, \ K_{127}, \ \ldots, \ K_{112}),$$
$$x_1 = (IV_{111}, \ \ldots, \ IV_{96}, \ K_{111}, \ \ldots, \ K_{96}),$$
$$x_2 = (IV_{95}, \ \ldots, \ IV_{80}, \ K_{95}, \ \ldots, \ K_{80}),$$
$$x_3 = (IV_{79}, \ \ldots, \ IV_{64}, \ K_{79}, \ \ldots, \ K_{64}),$$
$$x_4 = (IV_{63}, \ \ldots, \ IV_{48}, \ K_{63}, \ \ldots, \ K_{48}),$$
$$x_5 = (IV_{47}, \ \ldots, \ IV_{32}, \ K_{47}, \ \ldots, \ K_{32}),$$
$$x_6 = (IV_{31}, \ \ldots, \ IV_{16}, \ K_{31}, \ \ldots, \ K_{16}),$$
$$x_7 = (IV_{15}, \ \ldots, \ IV_0, \ K_{15}, \ \ldots, \ K_0).$$

Before producing the keystream, a mixing procedure will be applied to "stir up" state bits: $x_i = x_i \oplus (x_i)_{>>>16}$ $(i = 0 \sim 3)$.

The 128-bit key will be expanded into four subkeys k_i for four corresponding T-maps $\mathbf{T}_i$ $(i = 0 \sim 3)$. The subkey structure (see Figure 8.2) is the same as in Section 8.2.1, the length is still 64-bit, and each field is an unsigned integer. However two fields *seed* and *offset* will be used differently due to the larger state space:

***seed*:** still 32-bit, split into four bytes s_3 (MSB), s_2, s_1 and s_0 (LSB). The state space $(x_i)_{i=0}^{7}$ has 256-bit, and the seed is only 32-bit, one-eighth of the state bits will be affected by the seed. For even T-maps (T_0, T_2) the four LSBs of the four odd state words (i.e., x_1, x_3, x_5 and x_7) will be updated by XORing s_3, s_2, s_1 and s_0, respectively. Likewise, for odd T-maps (T_1, T_3) the four 2nd LSBs of the four even state words (i.e., x_0, x_2, x_4 and x_6) will be updated by XORing s_3, s_2, s_1 and s_0, respectively.

***offset*:** it's still 8-bit. To generate a new odd orbit from the current state $(x_i)_{i=0}^{7}$, we add the *offset* into the MSBs of the even state words (XOR indeed), then swap these MSBs with their corresponding LSBs. For even orbits, we add the *offset* into the MSBs of the odd state words (XOR indeed), then swap these MSBs with their corresponding LSBs. By doing so, we move the MSB to LSB, it's equivalent to propagate the information from left to right of the state.

Performance and safety analysis

Table 8.2 gives the encryption speeds for the multi-word version Mmohoct. The test is conducted on a Pentium-4 3.4GHz PC. The encryption speed of the Mmohoct is slower

than RC4 (113 MB/s) and AES-CTR (58 MB/s) in Crypto++ 5.2.1 benchmarks [30]. Again, this is because the Mmohoct has multiple nonlinear maps and multiplications, however its state space is much larger (8×32 bits in current experiment). We can do better by: optimizing code, reducing file operations, implementing better algorithms for multiplication operations.

Table 8.2: Encryption speed of Mmohoct multi-word version

File Size (KB)	584	11,200	22,400	145,600
Time Taken (s)	0.031	0.625	1.868	8.007
Speed (MB/s)	20	19	13	19

By using Mmohom mechanism in designing T-function-based stream ciphers, we introduce another nonlinearity into the system. The algebraic structures of T-functions are destroyed. This makes the Mmohoct ciphers resistant against algebraic attack, which is a fast attack to stream ciphers. By carefully constructing T-functions, we know that they all have the maximal single cycles. In our multi-word Mmohoct cipher experiments the cycle length is 2^{256}, since the state space is 8 words with 32-bit each. This cipher uses two types of T-maps: TSC-like and quadratic format and achieves good diffusion spreading by using quadratic T-maps (multiplications).

8.3 Conclusion

The T-functions with maximal single cycles are used in our Mmohoct ciphers. The Mmohom mechanism destroys the algebraic structures (patterns) and certain degree

of linearity of the T-functions. This makes the Mmohoct resistant against algebraic attacks. The state transition orders are interrupted and mixed (there is state transition order for a given T-map). Two experimental Mmohoct ciphers have been implemented with comparable encryption speed with RC4 and AES-CTR in the Crypto++ 5.2.1 Benchmarks, and the keystreams have passed NIST and DIEHARD randomness statistical tests.

Chapter 9

Future Research

During this research we developed and analyzed a multi-map orbit hopping mechanism Mmohom and used it in stream cipher designs. Here we will discuss some of the open problems and our future research.

9.1 Open problems and future research

In mathematical analysis of the Mmohom mechanism we use conjugate permutation method, which is good for integer orbits situation, but can we use it to evaluate the cycle of the real number orbit case?

Chaotic cipher Mmohocc uses multiple chaotic maps, which are real functions. We show that the cipher is strong against most of common attacks. There are other ciphers in real-number-domain, such as the Littlewood cipher [17] (although it has been broken [107]). Can we find some real domain mathematical problems on which cryptographic primitives can be built?

In the integer domain we applied Mmohom into LFSR-based and T-function-based stream ciphers to destroy their undesired properties. Can we extend this design pattern to more words (say 32, or 64) state space without suffering significant slowdown of operation?

Here is a list of our future research topics:

- Further mathematical analysis of the Mmohom mechanism.
- Applications of the chaotic cipher Mmohocc into fiber-optics communication systems and the Internet/Web-based systems.
- Applications of the Mmohocc and Mmohoct ciphers into lightweight encryption scenarios, such as RFID tags, smart cards, and other ASICs.
- Apply Mmohom-based ciphers into Bluetooth systems, in which frequency hopping mechanism is used. The combination of both hopping's should result in better security and performance.
- Improving the throughput and security of Mmohom-based ciphers, meanwhile integrating authentication method according to eSTREAM profiles.
- Quantum stream cipher design.

Appendix A

Density Function

We give the proof of the density function for the quadratic chaotic map.

A.1 Density function for $S(x) = x^2 + C$

The Frobenius-Perron operator P [61, 112]: if $(X, \mathcal{A}, \mu)$ is a measure space, $S : X \to X$ is a nonsingular transformation, then the unique operator $P : L^1 \to L^1$ defined by the equation

$$\int_A Pf(x)\mu(dx) = \int_{S^{-1}(A)} f(x)\mu(dx), \quad \forall A \in \mathcal{A}.$$

is called the Frobenius-Perron operator corresponding to S.

For the quadratic chaotic map $S(x) = x^2 - 2$, the limiting density function is

$$f_\infty(x) = \frac{1}{\pi\sqrt{x(x-2)}}.$$

Proof. For the transformation $S : [-2, 2] \to [-2, 2]$, the Frobenius-Perron operator P and density function $f(x)$ has the following relation [61]

$$Pf(x) = \frac{d}{dx} \int_{S^{-1}([a,x])} f(u)du. \tag{A.1}$$

To apply the equation (A.1) we need an analytic formula for the counterimage of the interval $[0, x]$. From Figure A.1 we calculate the end points of the two intervals constituting $S^{-1}([0, 1])$ by solving a quadratic equation $x = y^2 - 2$. Thus for $x \leq 0$

$$S^{-1}([0, x]) = [-\sqrt{2}, -\sqrt{x+2}] \cup [\sqrt{x+2}, \sqrt{2}].$$

With this counterimage, equation (A.1) becomes

$$Pf(x) = \frac{d}{dx} \int_{-\sqrt{2}}^{-\sqrt{x+2}} f(u)du + \frac{d}{dx} \int_{\sqrt{x+2}}^{\sqrt{2}} f(u)du. \tag{A.2}$$

Since

$$\frac{d}{dx} \int_{\psi(x)}^{\varphi(x)} f(u)du = f(\varphi(x))\varphi'(x) - f(\psi(x))\psi'(x),$$

(A.2) becomes

$$Pf(x) = -\frac{1}{2\sqrt{x+2}}(f(-\sqrt{x+2}) + f(\sqrt{x+2})). \tag{A.3}$$

Pick an initial density $f(x) \equiv 1$, then both terms inside the braces in (A.3) are constant, therefore we have

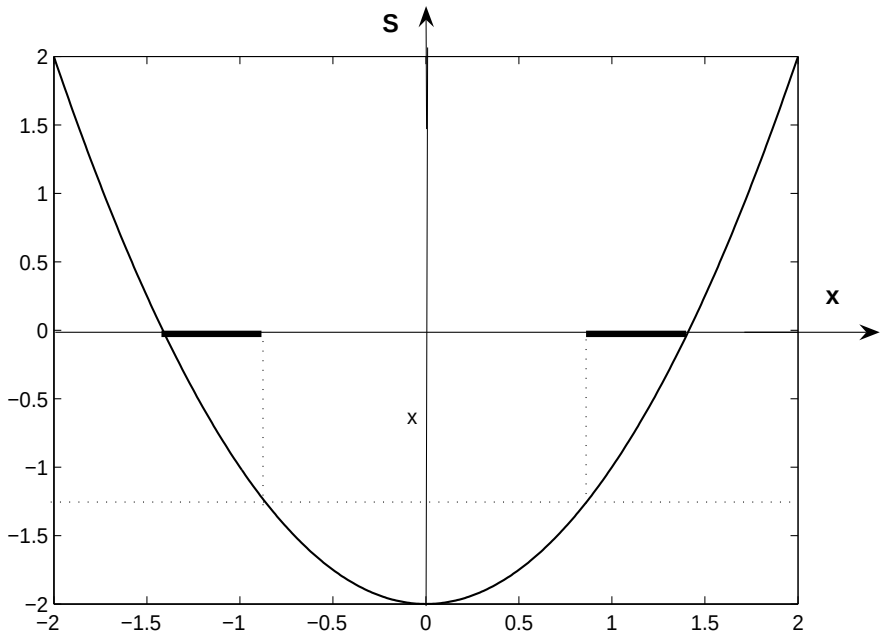
$$Pf(x) = -\frac{1}{\sqrt{x+2}}. \tag{A.4}$$

Now substitute this expression for $Pf(x)$ in place of $f(x)$ on the right hand side of (A.3) we have

$$\begin{aligned} P(Pf(x)) &= P^2 f(x) \\ &= \frac{1}{2\sqrt{x+2}} \left(\frac{1}{\sqrt{2-\sqrt{x+2}}} + \frac{1}{\sqrt{2+\sqrt{x+2}}} \right). \end{aligned} \tag{A.5}$$

As $n \to \infty$ we iterate $P^n f(x)$ to get the limiting density $f_\infty(x) = 1/\pi\sqrt{x(x+2)}$.

□



It consists of the union of the two sets denoted by the heavy lines on the x-axis.

Figure A.1: The counterimage of the set $[0, x]$ for a quadratic map.

Appendix B

Hopping Patterns

A hopping pattern lookup table is given in Table B.1. The numbers within each subgroup can be arbitrarily re-arranged, for example in pattern 141, the numbers in subgroup (9, 11, 10) could be rearranged as (10, 9, 11) or as (11, 9, 10). The "hpsn" is short for "hopping pattern serial number."

Table B.1: Partial hopping patterns for 11 orbits

hpsn	Orbit Permutation Sequence	hpsn	Orbit Permutation Sequence
0	(1,2), (3,4), (5,6), (7,8), (9,10,11)	120	(2,1), (4,3), (6,5), (8,7), (9,11,10)
1	(1,2), (3,4), (5,6), (9,10,11), (7,8)	121	(2,1), (4,3), (6,5), (9,11,10), (8,7)
2	(1,2), (3,4), (7,8), (5,6), (9,10,11)	122	(2,1), (4,3), (8,7), (6,5), (9,11,10)
3	(1,2), (3,4), (7,8), (9,10,11), (5,6)	123	(2,1), (4,3), (8,7), (9,11,10), (6,5)
4	(1,2), (3,4), (9,10,11), (5,6), (7,8)	124	(2,1), (4,3), (9,11,10), (6,5), (8,7)
5	(1,2), (3,4), (9,10,11), (7,8), (5,6)	125	(2,1), (4,3), (9,11,10), (8,7), (6,5)

Table B.1: Partial hopping patterns for 11 orbits

hpsn	Orbit Permutation Sequence	hpsn	Orbit Permutation Sequence
6	(1,2), (5,6), (3,4), (7,8), (9,10,11)	126	(2,1), (6,5), (4,3),(8,7), (9,11,10)
7	(1,2), (5,6), (3,4), (9,10,11), (7,8)	127	(2,1), (6,5), (4,3), (9,11,10), (8,7)
8	(1,2), (5,6), (7,8), (3,4), (9,10,11)	128	(2,1), (6,5), (8,7), (4,3), (9,11,10)
9	(1,2), (5,6), (7,8), (9,10,11), (3,4)	129	(2,1), (6,5), (8,7), (9,11,10), (4,3)
10	(1,2), (5,6), (9,10,11), (3,4), (7,8)	130	(2,1), (6,5), (9,11,10), (4,3), (8,7)
11	(1,2), (5,6), (9,10,11), (7,8), (3,4)	131	(2,1), (6,5), (9,11,10), (8,7), (4,3)
12	(1,2), (7,8), (3,4), (5,6), (9,10,11)	132	(2,1), (8,7), (4,3), (6,5), (9,11,10)
13	(1,2), (7,8), (3,4), (9,10,11), (5,6)	133	(2,1), (8,7), (4,3), (9,11,10), (6,5)
14	(1,2), (7,8), (5,6), (3,4), (9,10,11)	134	(2,1), (8,7), (6,5), (4,3), (9,11,10)
15	(1,2), (7,8), (5,6), (9,10,11), (3,4)	135	(2,1), (8,7), (6,5), (9,11,10), (4,3)
16	(1,2), (7,8), (9,10,11), (3,4), (5,6)	136	(2,1), (8,7), (9,11,10), (4,3), (6,5)
17	(1,2), (7,8), (9,10,11), (5,6), (3,4)	137	(2,1), (8,7), (9,11,10), (6,5), (4,3)
18	(1,2), (9,10,11), (3,4), (5,6), (7,8)	138	(2,1), (9,11,10), (4,3), (6,5), (8,7)
19	(1,2), (9,10,11), (3,4), (7,8), (5,6)	139	(2,1), (9,11,10), (4,3), (8,7), (6,5)
20	(1,2), (9,10,11), (5,6), (3,4), (7,8)	140	(2,1), (9,11,10), (6,5), (4,3), (8,7)
21	(1,2), (9,10,11), (5,6), (7,8), (3,4)	141	(2,1), (9,11,10), (6,5), (8,7), (4,3)
22	(1,2), (9,10,11), (7,8), (3,4), (5,6)	142	(2,1), (9,11,10), (8,7), (4,3), (6,5)
23	(1,2), (9,10,11), (7,8), (5,6), (3,4)	143	(2,1), (9,11,10), (8,7), (6,5), (4,3)
24	(3,4), (1,2), (5,6), (7,8), (9,10,11)	144	(4,3), (2,1), (6,5), (8,7), (9,11,10)
25	(3,4), (1,2), (5,6), (9,10,11), (7,8)	145	(4,3), (2,1), (6,5), (9,11,10), (8,7)

Appendix C

Statistical Test Results

Here we give some statistical test results from NIST and DIEHARD suites.

C.1 Results from the NIST Suite

The NIST statistical testing suite [83] has two versions, one is GUI version on Windows and the other is command-line based on UNIX/Linux (Note: GUI version does not have Lempel-Ziv Compression test). In the GUI version, we setup the parameters and tests to be performed in two GUI windows.

Here we give a set of results from testing 1000 streams, with 1000000 bit each. Those streams are the resulting keystreams from the improved Mmohocc cipher with 8 maps.

For demonstration we only give some of results in table format for approximate entropy, block frequency, fast Fourier transform, linear complexity. They are listed from Table C.1 to Table C.4, each of them gives two test results.

Table C.1: APPROXIMATE ENTROPY

(a) m	= 10	(a) m	= 10
(b) n	= 1000000	(b) n	= 1000000
(c) χ^2	= 1020.808952	(c) χ^2	= 1053.723883
(d) $\phi(m)$	= -6.930993	(d) $\phi(m)$	= -6.930917
(e) $\phi(m+1)$	= -7.623630	(e) $\phi(m+1)$	= -7.623537
(f) ApEn	= 0.692637	(f) ApEn	= 0.692620
SUCCESS	*p-value* = 0.522269	SUCCESS	*p-value* = 0.252943

Table C.2: BLOCK FREQUENCY

(a) χ^2	= 7627.218750	(a) χ^2	= 7807.656250
(b) # of substrings	= 7812	(b) # of substrings	= 7812
(c) block length	= 128	(c) block length	= 128
SUCCESS	*p-value* = 0.931198	SUCCESS	*p-value* = 0.511737

C.2 Results from the DIEHARD Suite

Each run of the DIEHARD [67] generates a text file, which contains the test results for all 18 statistical tests. We test the same keystream for the NIST suite, and give some of the results from Table C.5 to Table C.10, each with two test results.

Table C.3: FFT

(a) Percentile	= 94.974400	(a) Percentile	= 94.996000
(b) N_l	= 474872.000000	(b) N_l	= 474980.000000
(c) N_o	= 475000.000000	(c) N_o	= 475000.000000
(d) d	= -1.174609	(d) d	= -0.183533
SUCCESS	p-*value* = 0.240151	SUCCESS	p-*value* = 0.854380

Table C.4: LINEAR COMPLEXITY (frequency in bucket C1 $\sim$ C6)

M (substring length) = 500 N (number of substrings) = 2000								
C0	C1	C2	C3	C4	C5	C6	χ^2	p-*value*
20	68	257	966	504	135	50	4.377157	0.625778
20	70	262	1017	484	115	32	5.361590	0.498341

Table C.5: BIRTHDAY SPACINGS ($M = 512$, $N = 2^{24}$, $\lambda = 2.0$)

using bits 2 to 25 (mean = 2.044) and 5 to 28 (mean = 1.926)				
duplicate	observed	expected	observed	expected
0	62.	67.668	81.	67.668
1	133.	135.335	131.	135.335
2	136.	135.335	128.	135.335
3	100.	90.224	97.	90.224
4	41.	45.112	41.	45.112
5	17.	18.045	15.	18.045
$6 \sim \infty$	11.	8.282	7.	8.282
χ^2 w/6 d.o.f.	= 2.91	*p-value* = .179331	= 4.76	*p-value* = .424946

Table C.6: OVERLAPPING 5-PERMUTATION

for a sample of 1,000,000 consecutive 5-tuples	
χ^2 for 99 degrees of freedom= 71.500	p-value= .016905
χ^2 for 99 degrees of freedom= 77.685	p-value= .055763

Table C.7: BINARY RANK for 6×8 for bit 2 to 9 (L) and 3 to 10 (R)

rank	observed	expected	$\frac{(o-e)^2}{e}$	sum	rank	observed	expected	$\frac{(o-e)^2}{e}$	sum
≤ 4	918	944.3	.733	.733	≤ 4	911	944.3	1.174	1.174
5	21758	21743.9	.009	.742	5	21731	21743.9	.008	1.182
6	77324	77311.8	.002	.744	6	77358	77311.8	.028	1.210
$p\text{-}value = 1 - exp(-sum/2) = .31052$					$p\text{-}value = 1 - exp(-sum/2) = .45383$				

Table C.8: BITSTREAM for 20-BIT OVERLAPPING WORD

test no.	missing words	*p-value*	test no.	missing words	*p-value*
1	141456	.14476	2	142265	.79702
3	141697	.30991	4	141666	.28484
5	141395	.11474	6	142113	.68292
7	141723	.33166	8	142239	.77943
9	141199	.04849	10	142041	.62082
11	142004	.58753	12	141925	.51460
13	141709	.31987	14	141908	.49876
15	141935	.52391	16	141923	.51274
17	141571	.21462	18	141699	.31156
19	141641	.26535	20	142380	.49876

Table C.9: DNA for 10-LETTER WORD

using bits	missing words	*p-value*	using bits	missing words	*p-value*
31 to 32	142029	.6380	30 to 31	141895	.4831
29 to 30	142168	.7773	28 to 29	142029	.6380
27 to 28	141588	.1716	26 to 27	141634	.2083
25 to 26	141874	.4585	24 to 25	141439	.0827

Table C.10: OPSO for 2-LETTER WORD

using bits	missing words	*p-value*	using bits	missing words	*p-value*
23 to 32	141470	.0649	22 to 31	141342	.0252
21 to 30	141616	.1559	20 to 29	141883	.4638
19 to 28	141930	.5284	18 to 27	142448	.9684
17 to 26	141977	.5923	16 to 25	142325	.9241
15 to 24	141934	.5339	14 to 23	142070	.7102
13 to 22	141946	.5503	12 to 21	142111	.7566
11 to 20	142006	.6306	10 to 19	142234	.8685
9 to 18	141304	.0184	8 to 17	141845	.4122

Appendix D

Notation and Symbols

$\mathbb{N} = \{1, 2, \ldots\}$: the set of natural numbers.

$\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$: the set of integers.

$\mathbb{Z}^+ = \{0, 1, 2, \ldots\}$: the set of non-negative integers.

$\mathbb{Q}$: the set of rational numbers.

$\mathbb{R}$: the set of real numbers.

$\mathbb{R}^+$: the set of non-negative real numbers.

$\emptyset$: the empty set.

$\forall i \in \mathbb{N}$: for all i in $\mathbb{N}$.

$\mathbb{Z}[x_1, \ldots, x_n]$: the set of polynomials in n variables over $\mathbb{Z}$.

$\lceil x \rceil$: the smallest integer greater than or equal to x.

$\lfloor x \rfloor$: the greatest integer less than or equal to x.

$\log x$: the base two logarithm of x.

$\ln x$: the natural logarithm of x.

A^T: the transpose of the matrix A.

$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$: the residues *mod* n.

$\mathbb{Z}_n^+ = \{1, \ldots, n-1\}$: the non-zero residues *mod* n.

$\vee$: Boolean disjunction (OR).

$\wedge$: Boolean conjunction (AND).

$\neg$: Boolean negation (NOT).

$\mathbb{Z}_2$ or $\mathbb{B}$: set $\{0, 1\}$.

$\mathbb{B}^n$ or $\{0,1\}^n$: the set of zero-one n-tuples, or the set of zero-one strings of length n.

$\{0,1\}^*$: the set of all zero-one strings of finite length.

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$: the binomial coefficient 'n choose k'.

$Pr[E]$: the probability of the event E.

$E[X]$: the expectation of the random variable X.

$g = O(f)$: g is of order f.

$B \backslash A$: the difference set of sets A and B. If A, B are subsets of a set X, then $B \backslash A = \{x \in X | x \in B, x \notin A\}$.

2^X: the collection of all subsets of set X, i.e. the powerset.

Bibliography

[1] E. Alvarez, A. Fernandez, P. Garcia, J. Jimenez, and A. Marcano. New approach to chaotic encryption. *Physics Letters A*, 263: 373–375, 1999.

[2] F. Argenti, S. Benzi, E.D. Re, and R. Genesio. Stream cipher system based on chaotic maps. In *Proc. of SPIE 01 (The International Society for Optical Engineering)*, volume 4122, 2001.

[3] A. Argyris, D. Syvridis, L. Larger, V. Lodi, P. Colet, I. Fischer, J. Ojalvo, C. Mirasso, L. Pesquera, and K.A. Shore. Chaos-based communications at high bit rates using commercial fibre-optic links. *Nature*, 437(17), 2005.

[4] F. Armknecht. Algebraic attacks on stream ciphers. In *Proc. of ECCOMAS 04 (European Congress on Computational Methods in Applied Sciences and Engineering)*, 2004.

[5] F. Armknecht and M. Krause. Algebraic attacks on combiners with memory. In *Proc. of CRYPTO 03*, LNCS 2927 (Lecture Notes in Computer Science, Springer-Verlag), pages 162–175, 2003.

[6] S. Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. homes.esat.kuleuven.be/∼jlano/stream/papers/babbage96.pdf, 1996.

[7] S. Babbage. Stream ciphers – what does the industry want? In *Proc. of SASC 04 (The State of the Art of Stream Ciphers)*, 2004.

[8] M.S. Baptista. Cryptography with chaos. *Physics Letters A*, 240: 50–54, 1998.

[9] E. Barkan, E. Biham, and N. Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In *Proc. of CRYPTO 03*, LNCS 2729, pages 600–616, 2003.

[10] C. Berbain, H. Gilbert, and J. Patarin. QUAD: a practical stream cipher with provable security. In *Proc. of EUROCRYPT 06*, LNCS 4004, pages 109–128, 2006.

[11] T. Beth and F.C. Piper. The stop-and-go generator. In *Proc. of EUROCRYPT 84*, LNCS 209, pages 88–92, 1984.

[12] A. Biryukov. Some thoughts on Time-Memory-Data tradeoffs. Cryptology ePrint Archive, 2005. eprint.iacr.org/2005/207.pdf.

[13] A. Biryukov and A. Shamir. Cryptanalytic Time/Memory/Data tradeoffs for stream ciphers. In *Proc. of ASIACRYPT 00*, LNCS 1976, pages 1–13, 2000.

[14] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In *Proc. of FSE 00 (Fast Software Encryption)*, LNCS 1978, pages 1–18, 2000.

[15] L. Blum, M. Blum, and S. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Computing*, 15(2), 1986.

[16] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4), 1984.

[17] B. Bollabás. *Littlewood's Miscellany.* Cambridge University Press, 1986.

[18] A. Boyarsky and P. Góra. Invariant measures for Chebyshev maps. *J. Applied Math. Stoch. Analysis*, 14: 257–263, 2001.

[19] R. Brown and L.O. Chua. Clarifying chaos: Examples and counterexamples. *Int. J. Bifurcation and Chaos*, 6(2): 219–249, 1996.

[20] A. Canteaut. Articles on stream ciphers. *Homepage of A. Canteaut*, www-rocq.inria.fr/codes/Anne.Canteaut/encyclopedia.pdf, 2005.

[21] A. Canteaut. Open problems related to algebraic attacks on stream ciphers. In *Proc. of WCC 05 (International Workshop on Coding and Cryptography)*, 2005.

[22] A. Canteaut. Ongoing research areas in symmetric cryptography. ECRYPT – European Network of Excellence in Cryptology, 2006. www.ecrypt.eu.org/documents.html.

[23] D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. In *Proc. of CRYPTO 02*, LNCS 2442, pages 515–532, 2002.

[24] D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generators. In *Proc. of CRYPTO 93*, LNCS 773, pages 22–39, 1993.

[25] N. Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrpt. In *Proc. of ICISC 02 (International Conference on Information Security and Cryptology)*, LNCS 2587, 2002.

[26] N. Courtois. Algebraic attacks on combiners with memory and several outputs. In *Proc. of ICISC 04*, LNCS 3506, pages 3–20, 2004.

[27] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In *Proc. of EUROCRYPT 03*, LNCS 2656, pages 345–359, 2003.

[28] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Proc. of ASIACRYPT 02*, LNCS 2501, pages 267–287, 2002.

[29] F. Dachselt and W. Schwarz. Chaos and cryptography. *IEEE Trans. on Circuits and Systems - Part I*, 48: 1498–1509, 2001.

[30] W. Dai. Crypto++ 5.2.1 benchmarks. www.eskimo.com/∼weidai/benchmarks.html.

[31] R.L. Devaney. *A First Course in Chaotic Dynamical Systems*. Westview Press, Boulder Colorado, 1992.

[32] P. Ekdahl and T. Johansson. Another attack on A5/1. *IEEE Transactions on Information Theory*, 49(1), 2003.

[33] eSTREAM. eSTREAM – the ecrypt stream cipher project, www.ecrypt.eu.org/stream.

[34] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Proc. of SAC 01 (Selected Areas in Cryptography)*, LNCS 2259, pages 1–23, 2001.

[35] D.A. Freedman. Mixtures of markov processes. *Annals of Mathematical Statistics*, 33(1): 114–118, 1962.

[36] J. Fridrich. Symmetric ciphers based on two-dimensional chaotic maps. *Int. J. Bifurcation and Chaos*, 8(6): 1259–84, 1998.

[37] M. Gëtz, K. Kelber, and W. Schwarz. Discrete-time chaotic encryption systems part I: Statistical design approach. *IEEE Trans. Circuits and Systems-I*, 44(10): 963–970, 1997.

[38] J. Golić. Cryptanalysis of alleged A5 stream cipher. In *Proc. of EUROCRYPT 97*, LNCS 1233, pages 239–255, 1997.

[39] J. Golić. Linear cryptanalysis of stream ciphers. In *Proc. of FSE 94*, LNCS 1008, pages 154–169, 1997.

[40] J. Golić. Linear statistical weakness of alleged RC4 key stream generator. In *Proc. of EUROCRYPT 97*, LNCS 1233, pages 226–238, 1997.

[41] J. Golić, A. Clark, and E. Dawson. Generalized inversion attack on nonlinear filter generators. *IEEE Trans. on Computers*, 49(10): 1100–1109, 2000.

[42] J. Golić and M.J. Mihajlevič. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *J. of Cryptology*, 3(3): 201–212, 1991.

[43] J. Hong, D. Lee, Y. Yeom, and D. Han. New single cycle T-function and stream cipher proposal. eSTREAM, ECRYPT Stream Cipher Project, 2005.

[44] J. Hong, D. Lee, Y. Yeom, and D. Han. A new single cycle T-functions. In *Proc. of FSE 05*, LNCS 3557, pages 68–82, 2005.

[45] J. Hong, D. Lee, Y. Yeom, D. Han, and S. Chee. T-function based stream cipher TSC-3. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/031, 2005.

[46] G. Jakimoski and L. Kocarev. Chaos and cryptography: block encryption ciphers based on chaotic maps. *IEEE Trans. Circuits and Systems-I: Fundamental Theory and Applications*, 48(2): 163–169, 2001.

[47] T. Johansson. eSTREAM – stream cipher proposals for ongoing analysis. eSTREAM–ECRYPT Stream Cipher Project Report, 2006.

[48] D. Kahn. *The Codebreakers: The Story of Secret Writing.* The Macmillan Company, New York, 1967.

[49] K. Kelber and W. Schwarz. General design rules for chaos-based encryption systems. In *Proc. of NOLTA 05 (Int. Symposium on Nonlinear Theory & Its Applications)*, 2005.

[50] E.L. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Trans. Inform. Theory*, IT-22, 1976.

[51] A. Klapper and M. Goresky. Feedback shift registers, 2-adic span, and combiners with memory. *J. of Cryptology*, 10: 111–147, 1997.

[52] A. Klimov. *Applications of T-functions in Cryptography.* PhD thesis, Weizmann Institute of Science, 2005.

[53] A. Klimov and A. Shamir. A new class of invertible mappings. In *Proc. of CHES 02 (Cryptographic Hardware and Embedded Systems)*, LNCS 2523, pages 470–483, 2002.

[54] A. Klimov and A. Shamir. Cryptographic applications of T-functions. In *Proc. of SAC 03*, LNCS 3006, pages 248–261, 2003.

[55] A. Klimov and A. Shamir. New cryptographic primitives based on multiword T-functions. In *Proc. of FSE 04*, LNCS 3017, pages 1–15, 2004.

[56] A. Klimov and A. Shamir. New applications of T-functions in block ciphers and hash functions. In *Proc. of FSE 05*, LNCS 3557, pages 18–31, 2005.

[57] L. Kocarev. Chaos-based cryptography: A brief overview. *IEEE Circuits Syst. Mag.*, 1(3): 6–21, 2001.

[58] L. Kocarev, G. Jakimoski, T. Stojanouski, and U. Parlitz. From chaotic maps to encryption schemes. In *Proc. IEEE International Symposium on Circuits and Systems*, volume 4, pages 514–7, 1998.

[59] M. Krause. BDD-based cryptanalysis of keystream generators. In *Proc. of EUROCRYPT 02*, LNCS 1462, pages 222–237, 2002.

[60] S. Künzli, P. Junod, and W. Meier. Distinguishing attacks on T-functions. In *Proc. of MYCRYPT 05*, LNCS 3715, 2005.

[61] A. Lasota and M.C. Mackey. *Chaos, Fractals, and Noise – Stochastic Aspects of Dynamics*. Springer-Verlag, 1997 (2nd ed).

[62] S. Li. *Analysis and New Designs of Digital Chaotic Ciphers*. PhD thesis, Xi'an Jiaotong University, Dept. of Information & Communication Engineering, 2003.

[63] C. Ling and S. Sun. Chaotic frequency hopping sequences. *IEEE Trans. on Communications*, 46(11): 1433–37, 1998.

[64] H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST concerning AES modes of operations: CTR-Mode Encryption. NIST's First Modes of Operation Workshop, 2000. csrc.nist.gov/CryptoToolkit/modes/workshop1.

[65] Y. Lu and S. Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In *Proc. of CRYPTO 04*, LNCS 3152, pages 407–425, 2004.

[66] I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *Proc. of FSE 01*, LNCS 2355, pages 152–164, 2001.

[67] G. Marsaglia. Diehard battery of tests of randomness. *Florida State University*, stat.fsu.edu/∼geo/diehard.html, 1995.

[68] N. Masuda and K. Aihara. Cryptosystems based on space-discretization of chaotic maps. In *Proc. of IEEE Int. Symposium Circuits and Systems 01*, volume 3, pages 321–324, 2001.

[69] N. Masuda and K. Aihara. Cryptosystems with discretized chaotic maps. *IEEE Trans. of Circuits and Systems–I*, 49(1): 28–40, 2002.

[70] A. Maximov. A new stream cipher Mir-1. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/017, 2005.

[71] J.L. McCauley. *Chaos, Dynamics and Fractals*. Cambridge University Press, 1993.

[72] W. Meier and T. Siegenthaler. Fast correlation attacks on stream ciphers. In *Proc. of EUROCRYPT 88*, LNCS 330, pages 301–314, 1988.

[73] W. Meier and T. Siegenthaler. Correlation properties of combiners with memory in stream ciphers. *J. of Cryptology*, 5(1): 67–86, 1992.

[74] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001 (5th printing). www.cacr.math.uwaterloo.ca/hac.

[75] M.J. Mihajlevič. A correlation attack on the binary sequence generators with time-varying output function. In *Proc. ASIACRYPT 94*, LNCS, pages 67–79, 1994.

[76] C. Mitchell and A. Dent. International standards for stream ciphers: A progress report. In *Proc. of SASC 04*, 2004.

[77] J. Mitra and P. Sarkar. Time-memory trade-off attacks on multiplications and T-functions. In *Proc. of ASIACRYPT 04*, LNCS 3329, pages 468–482, 2004.

[78] H. Molland. *New Methods for Cryptanalysis of Stream Ciphers*. PhD thesis, University of Bergen, Dept. of Informatics, 2005.

[79] H. Molland and T. Helleseth. A linear weakness in the Klimov-Shamir T-function. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, pages 1106–10, 2005.

[80] D. Moon, D. Kwon, D. Han, J. Lee, G. Ryu, D. Lee, Y. Yeom, and S. Chee. T-function based stream cipher TSC-4. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/024, 2006.

[81] F. Muller and T. Peyrin. Linear cryptanalysis of TSC stream ciphers - applications to the ECRYPT proposal TSC-3. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/042, 2005.

[82] The NIST. Batteries of statistical tests for random number generators. *NIST Computer Security Division Website*, csrc.nist.gov/rng/rng6_3.html, 2001.

[83] The NIST. The NIST statistical test suite. *NIST Computer Security Division Website*, csrc.nist.gov/rng/rng2.html, 2005.

[84] Ecrypt NoE. eSTREAM optimized code HOWTO, www.ecrypt.eu.org/stream/perf.

[85] N.K. Pareek, V. Patidar, and K.K. Sud. Discrete chaotic cryptography using external key. *Physics Letters A*, 309(1-2): 75–82, 2003.

[86] S. Paul and B. Preneel. Analysis of non-fortuitous predictive states of the RC4 keystream generator. In *Proc. of INDOCRYPT 03*, LNCS 2904, pages 52–67, 2003.

[87] S. Paul and B. Preneel. A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In *Proc. of FSE 04*, LNCS 3017, pages 245–259, 2004.

[88] N.S. Philip and K.B. Joseph. Chaos for stream cipher. *arXiv:nLin.CD/0102012*, 2001.

[89] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge Univ. Press, 2002 (2nd ed).

[90] M.O. Rabin. Provably unbreakable hyper-encryption in the limited access model. In *Proc. of IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 34–37, 2005.

[91] S. Rao. Implementing a bidirectional frequency hopping application with TRF6903 and MSP430. *Texas Instruments Application Report*, September, 2004.

[92] R.L. Rivest. The RC4 encryption algorithm. *RSA Data Security Report*, 1992.

[93] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. In *Proc. of FSE 94*, LNCS 809, pages 56–63, 1994.

[94] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm (full version). CiteSeer.IST, 1997. citeseer.ifi.unizh.ch/article/rogaway97software-optimized.html.

[95] G. Rose and P. Hawkes. On the applicability of distinguishing attacks against stream ciphers. eprint.iacr.org, 2002. eprint.iacr.org/2002/142.pdf.

[96] J. Rotman. *The Theory of Groups: An Introduction*. Allyn and Bacon, Inc., Boston, 1973 (2nd ed).

[97] T. Rowlands and D. Rowlands. A more resilient approach to chaotic encryption. In *Proc. of ICITA 02*, pages 1–6, 2002.

[98] R.A. Rueppel. When shift registers clock themselves. In *Proc. of EUROCRYPT 87*, LNCS 304, pages 53–64, 1987.

[99] J. Scharinger. Fast encryption of image data using chaotic Kolmogorov flows. *J. of Electronic Imaging*, 7(2), 1997.

[100] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, New York, 1996 (2nd ed).

[101] A. Shamir. Stream ciphers: Dead or alive? In *Proc. of ASIACRYPT 04*, LNCS 3329, page 78, 2004.

[102] A. Shamir and B. Tsaban. Guaranteeing the diversity of number generators. *arXiv.org*, (CR/0112014), 2004.

[103] C.E. Shannon. Communication theory of secrecy systems. *Bell Technical Journal*, 28(4): 656–715, 1949.

[104] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. on Information Theory*, IT-30(5): 776–780, 1984.

[105] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. on Computing*, C-34: 81–85, 1985.

[106] J. Soto. Statistical testing of random number generators. In *Proc. of 22nd National Information Systems Security Conference*, 1999.

[107] D. Stehlè. Breaking Littlewood's cipher. Rapport de recherche, INRIA, 2003.

[108] J. Talbot and D. Welsh. *Complexity and Cryptography: An Introduction*. Cambridge University Press, 2006.

[109] W. Trappe and L.C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, 2002.

[110] Y. Tsunoo, T. Saito, H. Kubo, and M. Shigeri. Cryptanalysis of Mir-1, a T-function based stream cipher. eSTREAM, ECRYPT Stream Cipher Project, 020, 2006.

[111] M. Twain. The adventures of huckleberry finn. *Project Gutenberg Online Book*, www.gutenberg.org/etext/76.

[112] P. Walters. *An Introduction to Ergodic Theory*. Springer-Verlag, 1982.

[113] H. Wang. *Research on Chaotic Spread Spectrum CDMA Communication System*. PhD thesis, Beijing University of Posts and Telecommunications, 1997 (In Chinese).

[114] R. Wash. Lecture notes on stream ciphers and RC4. *Homepage of Rick Wash*, www-personal.si.umich.edu/∼rwash/pubs/stream.pdf, 2001.

[115] Wikipedia. *Wikipedia - the free encyclopedia*, en.wikipedia.org/wiki/Main_Page.

[116] F. Willeboordse. Encoding of traveling waves in a coupled map lattice. *Int. J. Bifurcation and Chaos*, 4(6): 1667–1673, 1994.

[117] K. Zeng and M. Huang. On the linear syndrome method in cryptanalysis. In *Proc. CRYPTO 88*, LNCS 403, pages 469–478, 1988.

[118] K. Zeng, C. Yang, D. Wei, and T. Rao. On the linear complexity test (LCT) in cryptanalysis with applications. In *Proc. CRYPTO 89*, LNCS 435, pages 164–174, 1989.

[119] K. Zeng, C. Yang, D. Wei, and T. Rao. Pseudorandom bit generators in stream-cipher cryptography. *IEEE Computer*, 24, 1991.

[120] M. Zhang and G. Xiao. A modified design criterion for stream ciphers. In *Proc. of CHINACRYPT 94*, pages 201–209, 1994 (In Chinese).

[121] X. Zhang, L. Shu, and K. Tang. Multi-map orbit hopping chaotic cipher. In *Presentation on ICGTMP26*, 2006.

[122] X. Zhang, L. Shu, and K. Tang. A chaotic cipher Mmohocc and its security analysis. In *Proc. of SPIE 07*, volume 6579, 2007.

[123] X. Zhang, K. Tang, and L. Shu. A chaotic cipher Mmohocc and its randomness evaluation. In *Proc. of ICCS 06*, 2006.

[124] H. Zhou, X. Ling, and J. Yu. Secure communication via one-dimensional chaotic inverse systems. In *Proc. of IEEE Int. Symposium Circuits and Systems 97*, volume 2, 1997.