THE DESIGN PRINCIPLES OF LINEAR FEEDBACK SHIFT REGISTER BASED CLOCK CONTROLLED STREAM CIPHERS

by

İmran Ergüler

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2005

THE DESIGN PRINCIPLES OF LINEAR FEEDBACK SHIFT REGISTER BASED CLOCK CONTROLLED STREAM CIPHERS

APPROVED BY:

Prof. Dr. Emin Anarım ……
(Thesis Supervisor)

Prof. Dr. Kemal Cılız ……

Prof. Dr. Ufuk Çağlayan ……

DATE OF APPROVAL: 02.08.2005

ACKNOWLEDGEMENTS

This work has been conducted at BUICS Laboratory of Electrical-Electronic Engineering Department of Bogazici University. Many people have contributed and helped me during my graduate study. Firstly, I would like to thank to my supervisor Prof. Emin Anarım for his inspiring ideas, technical advice, guidance and help during my thesis. I am grateful for the freedom he gave me in choosing my research area in the field of cryptography. BUICS Lab. deserves a special mention for its comfortable working environment.

Also, I wish to thank all the faculty of the Department of Electronics & Communications Engineering of Dogus University for the friendly atmosphere they provided to work in and their morale support in the course of this thesis. I would also like to thank my colleagues at the University that are excellent companions at social events, and just good friends of mine. Special thanks to Mete Akgün for his effort and help in my study. Also, many thanks to committee for their kindness and participation in my thesis.

The last but not least I would like to thank my parents, my sister and my brother, Bahadır Ergüler, for their enthusiastic support, endless love, patience and encouraging me in my choices in life.

ABSTRACT

THE DESIGN PRINCIPLES OF LINEAR FEEDBACK SHIFT REGISTER BASED CLOCK CONTROLLED STREAM CIPHERS

Stream ciphers are one of the most important classes of encryption algorithms used to ensure security in digital communication. The design of many stream ciphers is based on use of Linear Feedback Shift Registers (LFSRs), due to their simplicity, speed of implementation in hardware and providing sequences with good statistical properties. A stream cipher can not be considered suitable for cryptographic applications unless its output sequences have large periods, large linear complexities and possess certain randomness properties. Moreover a stream cipher must provide high resistance against well known cryptanalytic attacks such as time-memory trade-off attacks, divide-conquer attacks and correlation attacks. The use of clock-controlled shift registers in key stream generators can be a good alternative for achieving these properties.

In this thesis, the design principles of a cryptographically secure LFSR based clock controlled stream ciphers are described and two new stream cipher algorithms ANER-H and CSDS are presented. In addition, key stream properties of these algorithms and their resistance with respect to some well known cryptographic attacks are investigated. From the mathematical expressions and simulation results, it is shown that the two algorithms produce key stream sequences with satisfying basic security requirements and provide high resistance against currently known styles of attacks. Within their security powers, the CSDS is also designed to be very fast, especially for software usage. On the other hand, the stream cipher ANER-H can be applicable for both in hardware and software, due to its conceptually simple design.

ÖZET

DOĞRUSAL GERİ BESLEMELİ KAYAN SAKLAÇ TABANLI SAAT KONTROLLÜ DİZİ TİP ŞİFRELERİN TASARIM İLKELERİ

Dizi tip şifreleme algoritmaları güvenli sayısal haberleşme uygulamalarında kullanılan en yaygın şifreleme metotlarındandır. Bu tip şifreleme algoritmaların çoğunluğu basitliğinden, donanımdaki hızından ve iyi istatistiksel özelliklere sahip olduğundan Doğrusal Geri Beslemeli Kayan Saklaçları (LFSRs) tasarımlarında kullanmaktadır. Yüksek periyotlu, yüksek doğrusal karmaşıklığa ve belirli rasgelelik özelliklerine sahip olmayan çıktı dizileri üreten dizi tip şifreleme algoritmaları güvenlik uygulamalarına uygun değildir. Bununla birlikte, bir dizi tip şifreleme algoritması bilinen saldırılara karşı örneğin; zaman-bellek ödünleşimi saldırıları, böl-fethet saldırıları ve ilinti saldırıları, yüksek direnç göstermelidir. Saat kontrollü kayan saklaçları bu tip şifreleme algoritmalarında kullanmak istenen özellikleri gerçekleştirmesi açısından iyi bir alternatiftir.

Bu tezde, kriptografik olarak güvenli LFSR tabanlı saat kontrollü dizi tip şifrelerin tasarım ilkeleri anlatılmakta ve isimleri ANER-H ve CSDS olan iki tane yeni dizi tip şifreleme algoritması önerilmektedir. Ayrıca bu algoritmaların ürettikleri çıktı dizilerinin özellikleri ve algoritmaların bilinen bazı saldırılara karşı dirençleri çalışmada verilmektedir. Matematiksel açılımlar ve simülasyon sonuçları ışığında iki algoritmanın da istenen minimum çıktı özellikleri gereksinimleri yerine getirdiği ve bilinen bazı saldırılara karşı yüksek dirence sahip olduğu gösterilmektedir. Yüksek güvenlik sağlamalarının yanı sıra, CSDS özellikle yazılımda çok hızlı olacak şekilde tasarlanmıştır. Diğer yandan ANER-H basit eleman tasarımlarından dolayı hem donanım hem de yazılım ortamlarındaki uygulamalara imkan vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

C	Ciphertext
C_i	Individual character of ciphertext
$C_i(t)$	The number of clocking for i'th register according to $f_{Ci}(t)$
D	Decryption
$d_{\max}$	Maximum number of clocking of generator register
E	Encryption
f	Next state function
F_q	The field
$f(x)$	The polynomial for the feedback function
g	The function which produces key stream bits
K	Secret key
L	The length of an LFSR
LC	Linear complexity
L(s)	Linear complexity of sequence s
M	Message
m	The order of the Boolean function
P	Period
P_d	Deletion rate
P_i	Individual character of plaintext
P_z	Period of the key stream sequence z
S	The summation of clockings of GR in CR's period
S_i	The present state of the generator
S_j_column	Computed results for column decision for S-box j of CSDS
S_j_output	Output of S-box j of CSDS
S_j_row	Computed results for row decision for S-box j of CSDS
S_k_column	Computed results for column decision for S-box k of CSDS
S_k_output	Output of S-box k of CSDS

S_k_row	Computed results for row decision for S-box *k* of CSDS
v	A fixed number of ciphertext symbols that key stream depends
Z_i	Key stream character
z(t)	The key stream bit at time *t*
α	Significance level
λ	Period of the CR register
ANER-H	The name of the first proposed algorithm
CBC	Cipher block chaining
CFB	Cipher feedback
CSDS	Clock-controlled Stream cipher with Dynamic S-box Selective
CTR	Counter
FIPS	Federal Information Processing Standards
FSR	Feedback Shift Register
gcd	Greatest common divisor
KFB	Key feedback
lcm	Least common multiplier
LFSR	Linear Feedback Shift Register
lsb	Least significant bit
NIST	National Institute of Standards and Technology
OFB	Output feedback

1. INTRODUCTION

Cryptosystems are mainly categorized into two groups according to the used key type, symmetric-key or public-key. In case of public-key cryptography, the sender uses publicly known information (public-key) in encryption process to send a message to the receiver and the receiver uses his secret information (private-key) to recover the message. On the other hand in the symmetric-key encryption systems, the sender and receiver have previously agreed on use of a secret key for both encryption and decryption. This key must be kept secret to avoid revealing of the secret information by the potential eavesdroppers.

The symmetric-key crypto systems are also classified into two major subgroups that are block ciphers and stream ciphers. Block ciphers tend to simultaneously encrypt groups of characters of a plaintext message with a fixed transformation, whereas stream ciphers operate on individual plaintext digits (usually bits or sometimes byte) at a time with a time-varying transformation [1]. In other words, block ciphers operate on large blocks of data, while stream ciphers typically operate on smaller units of plaintext. Stream ciphers seem to be one of the best alternatives to high-speed communications, since they offer required security for high data rate applications and have algorithms that are popular in fast implementations. They are generally faster than block ciphers in hardware and have less complex hardware circuitry. Furthermore, stream ciphers can be more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Since stream ciphers have limited or no error propagation, they can be advantageous in situations where transmission errors are highly probable [2].

The rest of the chapter gives a general introduction to the topic. A review of the cryptography from a stream ciphers point of view is presented in Section 1.1. Section 1.2 gives an explanation about symmetric key cryptography and a comparison between stream ciphers and block ciphers. In Section 1.3, the general structure of stream ciphers is introduced. Some different types of cryptanalytic attacks are discussed in Section 1.4. Finally, Section 1.5 gives the outline of the thesis.

1.1. A Review of the Cryptography from a Stream Ciphers Point of View

Cryptology is the branch of mathematics that uses mathematical techniques for designing, attacking and analyzing information security services. It is consisted of two sub study fields; named as cryptography and cryptanalysis. Cryptography is the science that studies mathematical techniques keeping messages or information secure and providing security services. On the other hand, cryptanalysis is the study of how to attack cryptographic mechanisms to recover secret information or to defeat the security services [1]. Modern cryptography mostly concerns with the concepts; confidentiality, authenticity, data integrity and non-repudiation. In Table 1.1, some cryptographic terms and their short explanations are given to make the remained sections clear for the non-technical readers.

For the ancient times, the application of cryptography is something transforming the letters or the symbols into different symbols or representations according to a simple rule to provide secrecy of the messages. Substitution ciphers and Caesar cipher can be nice examples to these approaches. A classic example to these systems is Vigenère cipher which operates on the following formula:

$$C_i = P_i + K_i \bmod 26 \tag{1.1}$$

According to this cipher model, each letter in the alphabet can be thought as a number ranging from 0 to 25. Then the plaintext letter *P* is added to key letter *K* in mod 26 to obtain corresponding ciphertext letter *C*. The reason why mod 26 operation is used is obvious; to keep the produced ciphertext letter in the alphabet set. A message encrypted by such an algorithm can be easily cryptanalyzed, due to its simple rule and risk in reuse of the key. However, the importance of the algorithm comes from the fact that, this algorithm can be seen as the first algorithm that has some common points with today's stream cipher structure. At the beginning of the 20^{th} century, Gilbert Vernam proposed a new type of cipher known as Vernam cipher which uses a secret key as long as the plaintext [3]. The cipher relies on the algorithm that the plaintext is XOR'ed with a random or pseudorandom

Table 1.1. Explanations of some cryptographic terms

Term	Explanation
Authentication	The process for identification of the data origin or destination
Cipher	A cryptographic algorithm used to encrypt and decrypt messages.
Ciphertext	Encrypted message of information
Confidentiality	Ensuring that the information is only available to authorized users
Cryptosystem	Mechanism for providing a secure means of information exchange.
Data Integrity	Ensuring that no one without any authorization can alter the message
Decryption	Recovering the actual message from the cipher
Encryption	Process of transforming the message into cipher
Non-repudiation	Ensuring that someone cannot deny a previous commitment or action.
Plaintext	The original message or information

stream of data the same length to generate the ciphertext as shown in the equation below; where $\oplus$ denotes the XOR operation and *M, K* and *C* represent the message (plaintext), stream sequence and ciphertext respectively .

$$C = M \oplus K \tag{1.2}$$

If the data stream is truly random and used only once, this is called the one-time pad. Vigenère cipher and the Vernam cipher have two important common points; first- both of the algorithms use the symmetric key encryption, and second- both ciphers operate on plaintext with symbol by symbol which is an important feature of the stream ciphers.

1.2. Symmetric Key Cryptography

There are two general types of key-based encryption techniques which are symmetric and public-key. Symmetric-key algorithms, sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa [1]. For most of the symmetric algorithms, sender and receiver have the same key to

realize encryption and decryption. Since these systems require that both of the sender and receiver previously have agreed on a key to communicate securely and this key must be kept secret from any eavesdropper, these algorithms are also called secret-key algorithms, single-key algorithms, or one-key algorithms. The security power of a symmetric-key algorithm can depend on secrecy of the key, because anyone who obtains the key can encrypt and decrypt messages.

In symmetric-key systems, if E denotes the encryption and D denotes decryption operations, then encryption and decryption can be formulated respectively by:

$$E_K(M) = C \tag{1.3}$$

$$D_K(C) = M \tag{1.4}$$

As it is mentioned in previous sections, symmetric-key algorithms can be divided into two main categories as stream ciphers and block ciphers. Block ciphers operate on large blocks of plaintext data, while stream ciphers operate on individual plaintext characters or bits (or bytes). Stream ciphers use a simple but dynamic transformation to one individual symbol or character at a time whereas block ciphers apply a more complex, but static transformation to a block of symbols at once. Stream ciphers can be designed to be faster than block ciphers, especially by using LFSRs (Linear Feedback Shift Registers) in hardware. Although the main difference between stream ciphers and block ciphers seems as the operational characteristics, a block cipher can operate as a stream cipher by using the CBC (cipher block chaining) mode. Moreover, by using different modes of block ciphers such as the CFB (cipher feedback) mode, the OFB (output feedback) mode, the KFB (key feedback) mode and the CTR (counter) mode, block ciphers can provide stream cipher like characteristics. The inverse is also true; one can obtain a block cipher operation from a stream cipher, though it is less efficient.

The general structure of a simple block cipher can be summarized as shown in Figure 1.1. This cipher takes n bit block of plaintext and secret key as its inputs, then transforms this block to n bit block of ciphertext, where n is called block size. To realize unique decryption, this transformation must be chosen as one to one. Most of the block ciphers

define a permutation on the set of *n* bit blocks. Some well known block ciphers are DES, FEAL, IDEA, RC5 and AES [1, 2].

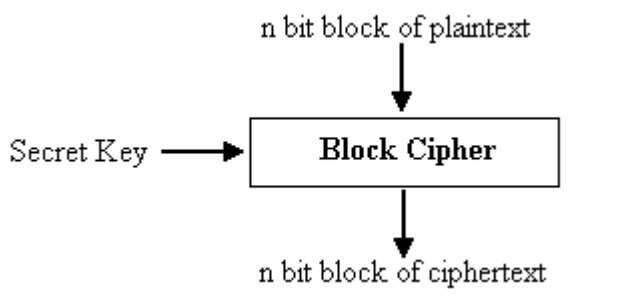

Figure 1.1. General structure of a simple block cipher

1.3. General Structure of Stream Ciphers

The Vernam cipher mentioned in Section 1.1. provides enough security, but it is not very practical considering its key length requirement which must be as long as plaintext. Also, key distribution and management for Vernam cipher is not an easy matter. Therefore cryptographers proposed to use a reduced key size but still providing a reasonable level of security. To achieve this process; the key in Vernam cipher is replaced by a pseudo random sequence of bits produced by a generator which is initialized with a shorter key. Although the proposed method is not as secure as Vernam cipher model, it is very convenient considering practical requirements and still maintains a reasonable level of security. In fact, this solution is not different from the notion of a basic stream cipher. Figure 1.2 shows the block diagram of general structure of a stream cipher. When we look at (1.2), it is seen that ciphertext is generated in a similar manner; instead of using the key directly, now key stream symbols Z_i produced by the key stream generator are used in XOR operation with message symbols M_i to produce ciphertext symbols C_i.

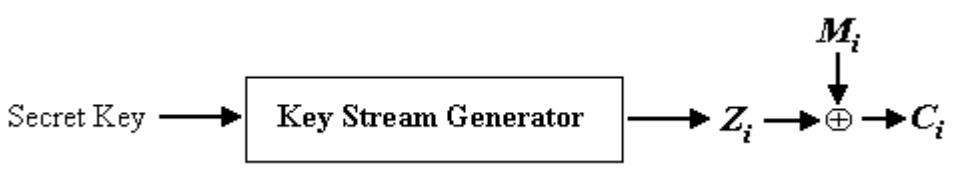

Figure 1.2. General structure of a simple stream cipher

1.4. Different Types of Cryptanalytic Attacks

As stated before, cryptanalysis is the study that uses mathematical techniques for breaking a cryptosystem and an attempted cryptanalysis is called an attack [1]. The aim of the attack can be recovering original message from the cipher, obtaining the secret key or revealing any secret information relating the message or cryptosystem. The Dutch linguist Auguste Kerckhoffs stated in his book that 'The security of the encryption scheme must depend only on the secrecy of the key, and not on the secrecy of the algorithms' which is known as Kerckhoffs' principal [4]. In fact, this is a necessary rule for a secure cryptosystem. To understand security strength of a cipher, it is investigated against some well known attacks. So, by comparing its resistance against these attacks, one can conclude that whether or not the cipher is secure. Of course, providing high resistance to the currently known styles of attacks does not guarantee the security of the cipher, because it is always possible to break any cipher with an unknown attack. However, any proposed cipher design must provide high resistance against known attacks. These attacks can be classified into two main groups; generic attacks and specific attacks. Generic attacks are applicable even if the attacker does not know the design of the cryptosystem [5]. On the other hand, in order to apply specific attacks, whose examples will be given in following chapters, to a cryptosystem, its inner system and how it works have to be known the by the cryptanalyst. In this section, we explain five general attack types.

- **Ciphertext-only attack:** For this type of attack, the attacker has several ciphertexts corresponding to several messages all of those have been encrypted using the same encryption algorithm. The cryptanalyst tries to recover the plaintext of as many messages as possible, or for some cases he tries to get the secret key, in order to decrypt other messages encrypted with the same key. Within these processes, this attack is the most difficult type of attack, since the attacker does not have enough information compared to other attacks that will be discussed below.

- **Known plaintext attack:** This is the most probable type of attack for stream ciphers. In known plaintext attack, the receiver knows a quantity of plaintext and the corresponding ciphertext. The goal of the attacker is to recover the key used in encrypting the messages or by using this information, an amount of known

plaintext-ciphertext pairs, developing algorithm which deduces unknown parts of the plaintext from the ciphertext. For example let the cryptanalyst has the following plaintext-ciphertext pairs: (P_1, C_1), (P_2, C_2),…(P_i, C_i) where $C_1=E_K(P_1)$, $C_2=E_K(P_2)$, $C_i=E_K(P_i)$ and K is the secret key. Then the attacker within this information, can deduce the key K, or get an algorithm to find remained unknown parts of the plaintext such as P_{i+1} from C_{i+1}.

- **Chosen plaintext attack:** In this type of attack, the attacker has access to encrypting a chosen plaintext, so he can obtain the ciphertext for a specific plaintext. Due to this property, chosen plaintext attack is a more powerful attack type than known plaintext attack. The aim of the attacker is the same as for case of known plaintext attack that is deducing the key or revealing the unknown parts of the plaintext.

- **Chosen ciphertext attack:** Compared the other three types of attacks, this method is the most powerful one. The strength of the method stems from the fact that; attacker can choose specific ciphertext samples and can decrypt these ciphertexts to get corresponding plaintexts. The attacker's job is deducing the secret key. For example let the cryptanalyst chose the following ciphertext samples: C_1, C_2,…C_i and decrypt these ciphertext samples to obtain corresponding plaintext pairs P_1, P_2,…P_i where $P_1=D_K(C_1)$, $P_2=D_K(C_2)$, $P_i=D_K(C_i)$ and K is the secret key.

- **Adaptive chosen-plaintext attack:** This attack is a chosen-plaintext attack wherein the choice of plaintext may depend on the ciphertext received from previous requests [2]. In other words, the cryptanalyst can also modify his choice based on the results of previous encryption.

The main goal of the attacks as described above is to recover the original message from given ciphertexts or deduce the secret key.

1.5. Thesis Outline

The thesis is consisted of six chapters. Chapter 1 gives an introduction about main aspects of cryptography. Chapter 2 discusses characteristics of LFSR based stream cipher, different applications of this type stream ciphers and presents detailed information about clock-controlled stream ciphers. In Chapter 3, the minimum required features of a well designed and secure stream cipher are given. Chapter 4 is devoted to description of the proposed cipher models ANER-H and CSDS. In Chapter 5, the security analysis of the proposed algorithms and their key stream properties such as statistical test results, period and linear complexity of the sequences are given. Finally, Chapter 6 gives the conclusions of the study.

2. STREAM CIPHERS

In this thesis, we will deal with LFSR based stream ciphers, only and in particular with stream ciphers in which the plaintext, the ciphertext and the key stream sequence are all binary sequences, and in which the mixing operation, for both encryption and decryption is XOR operation.

As it is given in the previous chapter, a stream cipher inspires the spirit of the one-time pad by using a short key to produce the key stream which appears to be random. Such a key stream sequence is often described as pseudo-random generation of which can be thought as in the field of stream ciphers. Therefore key stream generator can also be known as pseudo-random sequence generator or running key generator. Actually, producing random look like sequences is necessary condition for a secure stream cipher design, because the closer the key stream generator's output is to random, the harder time a cryptanalyst will have breaking it [1].

The stream cipher encryption and decryption can be formulated as follows: Let k_1, k_2, k_3,..., k_i denote the sequence that key stream generator outputs and p_1, p_2, p_3,..., p_i denote the plaintext bits. Then, if c_1, c_2, c_3,..., c_i represents the corresponding ciphertext bits, encryption and decryption are realized according to the equations below respectively.

$$c_i = p_i \oplus k_i \tag{2.1}$$

$$c_i \oplus k_i = p_i \oplus k_i \oplus k_i = p_i \tag{2.2}$$

Stream ciphers can be classified as synchronous or self-synchronizing stream ciphers according to the relation between key stream generation and plaintext.

- **Synchronous stream ciphers:** A synchronous stream cipher is one in which the key stream is generated independently of the plaintext message and of the ciphertext [2]. In the encryption side, a key stream generator outputs the key stream

bits, one after the other. On the decryption side, another key stream generator produces the identical key stream bits, one after the other. To avoid false decryption so error in communication, the two key stream generators must be synchronized. In case of losing synchronization during transmission, every ciphertext character after the error will be decrypted incorrectly. To solve this problem, the sender and receiver must resynchronize their key stream generators before continue their communication. Techniques for re-synchronization can be re-initialization or placing special markers at regular intervals in the ciphertext. An advantage of the synchronous stream cipher can be seen as; synchronous ciphers do not propagate transmission errors. If a bit or bits are changed during transmission so error occurs, then only error bits will be decrypted incorrectly, all preceding and subsequent bits will be unaffected. The encryption and decryption of a synchronous stream cipher are depicted in Figure 2.1 and Figure 2.2; where f denotes the next state function, g is the function which produces key stream bits, S_i is the present state of the generator, K is the secret key, p_i, k_i and c_i represent plaintext, key stream and ciphertext bits respectively. Most of the stream ciphers are binary additive stream ciphers that are synchronous stream ciphers in which the key stream, plaintext, and ciphertext digits are binary digits, and the output function is the XOR of plaintext and key stream sequence.

- **Self-synchronizing stream ciphers:** A self-synchronizing stream cipher is a finite state machine for which the key stream is generated as a function of the key and a fixed number of the previous ciphertext symbols [6]. In other words, for this type of stream ciphers each key stream bit is produced within a function of a fixed number of previous ciphertext bits. Since the key stream depends on a fixed number of the previous ciphertext symbols say v, the cipher will resynchronize after v symbols if there is a transmission error. In case of this, the next v symbols will be erroneous and the error propagation is thus worse than for a synchronous stream cipher. However, if some ciphertext symbols are deleted or inserted during transmission, the self-synchronizing cipher will recover after v correct ciphertext symbols, whereas the synchronous ciphers will never regain synchronization [6]. The encryption and decryption of a self-synchronizing stream cipher is shown in Figure

2.3 and Figure 2.4 respectively. As can be seen the ciphertext bits are given as input to determine next state of the key stream generator.

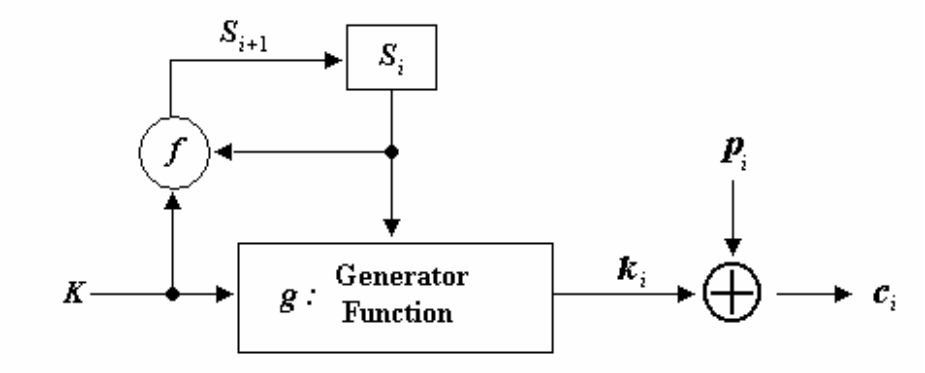

Figure 2.1. A general model for synchronous stream cipher encryption with XOR operation

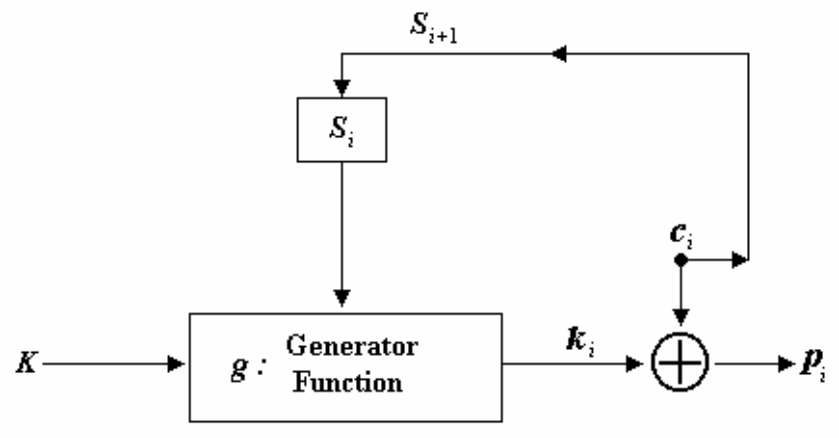

Figure 2.2. A general model for synchronous stream cipher decryption with XOR operation

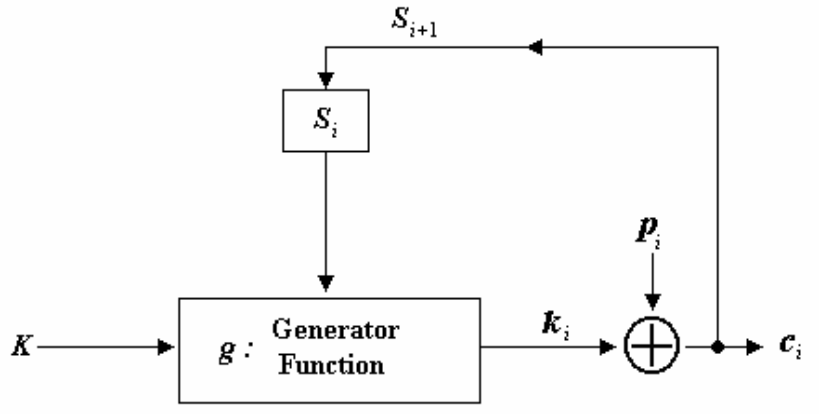

Figure 2.3. A general model for self-synchronizing stream cipher encryption, with XOR

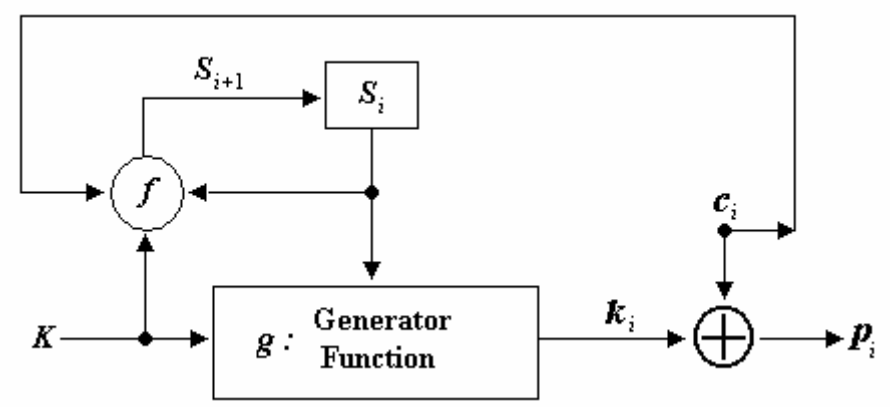

Figure 2.4. A general model for self-synchronizing stream cipher decryption, with XOR

For most of the stream ciphers, key stream sequence is generated independently from plaintext; so in some applications key stream sequence can be produced prior to encryption or decryption to speed up the process. Due their simple designs, low hardware complexity, high speed encryption characteristic and having low error propagation rate, stream ciphers are dominantly preferred in wireless communications such as in the applications of GSM, US Cellular Systems, WLAN, Bluetooth [7-10]. Also, majority of the stream ciphers relies on the use of LFSRs in their design. Therefore before going on different stream cipher types, in Section 2.1 LFSRs (Linear Feedback Shift Registers) and the reasons why they are used will be discussed. Next, some important variants of LFSR based stream ciphers will be presented in Section 2.2. The last section, Section 2.3, will give detailed information about clock-controlled stream ciphers that are the main skeleton models for the proposed stream ciphers ANER-H and CSDS.

2.1. Linear Feedback Shift Registers

An FSR (Feedback Shift Register) is a device made up by registers that produces binary sequences or symbols from a field F_q where $q=2^k$ and k is the symbol size (for our case and most of the stream ciphers q is chosen as 2). These registers are the main components of many key stream generators and they are used both in coding and cryptography. A feedback shift register is made up of two parts; a shift register *s* and a feedback function *f*. If the shift register *s* has a length of *n* bits or consists of *n* stages as s_1, s_2, ..., s_n which contains one bit 0 or 1, it is called an *n*-bit shift register. The feedback

function maps the state of the shift register according to its bits content. When the register is clocked at a time interval, all of the bits in the shift register are shifted one bit to the right. The new value of the left-most bit is computed by applying the feedback function to the contents of the register before clocking. At each clock, the right most bit of the register can be concerned as its output. The period of a shift register is the length of the output sequence before it starts repeating [1]. A general structure of a feedback shift register is depicted in Figure 2.5.

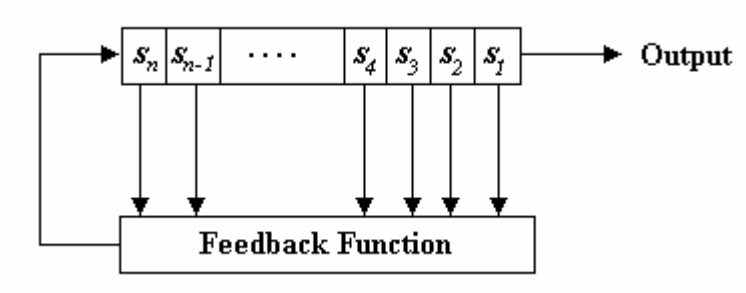

Figure 2.5. General structure of feedback shift register of length *n*

The simplest kind of feedback shift register is a linear feedback shift register. In that case, the feedback function can be written as $c_1 s_1 \oplus c_2 s_2 \oplus c_3 s_3 \ldots \oplus c_n s_n$, where *s* values are the contents of the register at time *t* and *c* values are the feedback coefficients. As can be seen the feedback function is linear and simply the XOR of the appropriate bits in the register according to whether or not c_i is equal to 1 or not; the list of the bits that have feedback coefficient value as 1 is called a tap sequence. An example of an LFSR is shown in Figure 2.6. Since the feedback function is linear and simple, many mathematical theories have been applied to analyzing LFSRs. The mathematical expression for the period of the shift register depends on its characteristic feedback function. If the feedback function is a primitive polynomial, then the period of the register becomes 2^n-1, where *n* is the length of the register. An irreducible polynomial $f(x) \in F_q[x]$ of degree l is said do be primitive if the root of $f(x)$ in the splitting field F_{q^l} is a generator of multiplicative group $F^*_{q^l}$; where a polynomial $g(x) \in F_q[x]$ is defined as irreducible polynomial over F_q, if it can not be factored into polynomials of smaller positive degrees in the ring of polynomials $F_q[x]$ [6].

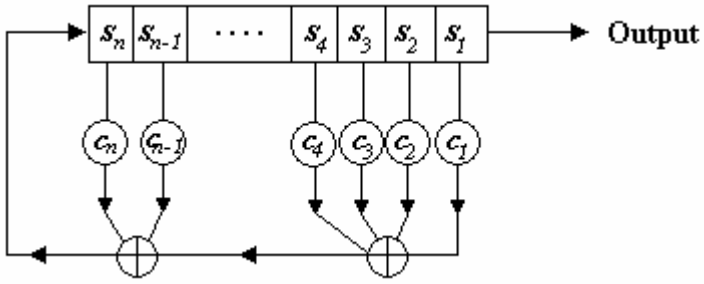

Figure 2.6. General structure of linear feedback shift register of length *n*

For our binary case, we can restrict the definition of irreducible polynomial and primitive polynomial as: A polynomial *f(x)* over GF(2) is said to be an irreducible polynomial over GF(2) if the only polynomials over GF(2) which divide *f(x)* are 1 and itself. An irreducible polynomial *f(x)* of degree *n,* which is also the length of the shift register, over GF(2) is said to be a primitive polynomial, if $(2^n - 1)$ is the least positive integer p such that *f(x)* divides $(1 + x^p)$ over GF(2).

If we start with a non-zero state as the initial state of the LFSR and the register has a primitive feedback polynomial, then all possible states except the all-zero state will appear during a period and the length of the period will be 2^n-1 as stated before. An LFSR with a primitive feedback polynomial is also called a maximum-length LFSR, and the sequence generated is called a maximum-length sequence. Notice that to say the sequence is maximum length, the initial state of the register must be non-zero and hereafter it is assumed that the starting state is as such. For example if the register has a length of 3 bits and a primitive feedback polynomial then the period of the register will be 2^3-1=7. To realize this example, let the register have a primitive feedback polynomial as x^2+x+1 in GF(2), tapped at the second and third bit; the state of the LFSR begins with '101'and changes as shown in Table 2.1. As can be seen, after 7 clockings the register repeats itself.

Most of the practical stream ciphers use LFSRs in their designs. There can be several reasons for this: Firstly, LFSRs are well suited for hardware implementation, because an LFSR is nothing more than an array of bit memories and its feedback function is just use of a series of XOR gates. Therefore, within a few logic gates an LFSR based stream cipher can be realized. Second reason is LFSRs can generate sequences with large period. An L-bit maximal length LFSR can produce a sequence of 2^L-1, so as L increases the length of

Table 2.1. State transition of the 3-bit LFSR

State of the LFSR
101
110
111
011
001
100
010

the period becomes incredibly large. The last reason why cryptographers use LFSRs in their stream generator models can be the fact that LFSRs produce sequences with good statistical properties. That is, they can produce random-looking key stream sequences. However they can be easily analyzed using algebraic techniques, due to their linear structure. The Berlekamp-Massey algorithm can generate sequence of an n-bit LFSR after using only $2n$ bits of the key stream [11]. Thus, if an attacker gets $2n$ bits of key stream he can break the stream cipher which is based on a pure single n bit LFSR. Considering Berlekamp-Massey algorithm, the strength of an LFSR stream cipher against such an attack can be evaluated by using the metric linear complexity or linear span. The linear complexity of a sequence say s, denoted by $L(s)$, is the length of the shortest LFSR that generates the same sequence. Linear complexity is very important, since the Berlekamp-Massey algorithm, can generate the sequence of a stream cipher with a linear complexity n, after examining only $2n$ bits of the key stream. Note that a high linear complexity value does not indicate that the stream cipher is secure, while the lower one means that the cipher is weak and insecure. So, pure LFSR can not be used as a secure stream cipher, although it has nice properties. To eradicate linear complexity problems of the LFSRs and keeping their good characteristics, different approaches that will be discussed in the following section have been proposed.

2.2. Some Stream Cipher Designs

An LFSR should never be used by itself as a key stream generator, since the output sequences of LFSRs are also easily predictable. Therefore for LFSR based stream ciphers

different techniques that can be can be divided into three general categories; nonlinear combination generators, nonlinear filter generators and clock-controlled generators have been presented to solve weaknesses of LFSRs. In first two of these techniques, stream generator design is simple; one or more LFSRs, generally of different lengths and with different feedback functions are used and their outputs or appropriate bits of the whole generator are taken by a nonlinear Boolean function to produce key stream sequence. Then the registers are regularly clocked and system works in this fashion. In case of the last category, clock-controlled generators, some LFSRs are clocked at different rates according to a rule or depending on the output of other LFSR; so they can be clocked irregularly. This property increases the linearity complexity of the system. The nonlinear combination generators and nonlinear filter generators will be explained in the following subsections. Since clock-controlled generators will be discussed in Section 2.3, there is no need to give brief information about it in this section.

2.2.1. Nonlinear Combination Generators

Nonlinear combination generators use several LFSRs in parallel to solve the linearity problem of LFSRs. They do this job by combining LFSR outputs with a nonlinear Boolean function *f,* which is also called combining function, as depicted in Figure 2.7. Before proceeding to an example of nonlinear combiner generator, it will be convenient to give some information about the Boolean functions. A product of *m* distinct variables is called an m^{th} order product of the variables. Every Boolean function $f(x_1, x_2, \ldots, x_n)$ can be given as a modulo 2 sum of distinct m^{th} order products of its variables, $0 \le m \le n$; which is called the algebraic normal form of *f*. The nonlinear order of *f* is the maximum of the order of the terms appearing in its algebraic normal form [2]. For instance, $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_3 \oplus x_2x_3x_4$ has a nonlinear order 3. Therefore a nonlinear combination generator has a high linear complexity, if its nonlinear Boolean function has a high order nonlinear order. By using nonlinear combination generator, increase in linear complexity is achieved and it seems there is no problem. However, using output of different LFSRs into a nonlinear Boolean function also increases the possibility that one or more of the internal output sequences or just outputs of individual LFSRs can be correlated with the produced key stream and by means of this correlation the generator can be attacked which is often called a correlation attack. The metric indicating the strength of the

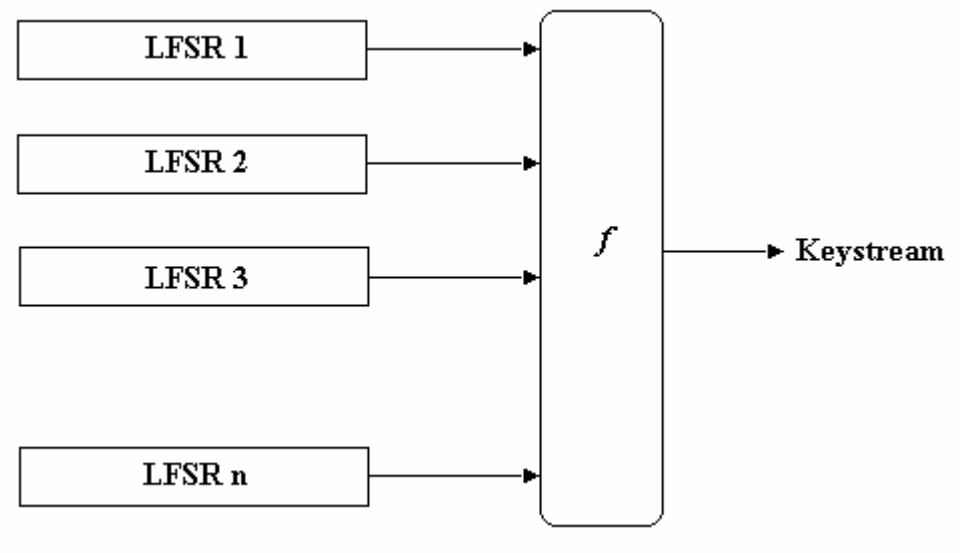

Figure 2.7. A nonlinear combination generator, where n LFSR outputs are combined with a nonlinear Boolean function *f* to produce key stream sequence for destroying linearity

generator to the correlation attack can be defined as the correlation immunity whose details have been shown in [12]. Thus, we can say that there is a trade-off between high correlation immunity and high linear complexity. To understand the importance of the correlation immunity, let us give the description of a popular example of nonlinear combination generator as the Geffe generator [13]. The Geffe generator is consisted of three maximal length LFSRs of L1, L2 and L3 respectively as shown in Figure 2.8. The outputs of LFSRs are combined within the function $f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$. The key stream generator uses three LFSRs, combined in a nonlinear manner. If L1, L2 and L3 are pairwise relatively prime, then the period of the generator is $(2^{L1}-1)$ $(2^{L2}-1)$ $(2^{L3}-1)$ and the linear complexity of the key stream sequence becomes L1L2 + L2L3 + L3. For the appropriate values of L1, L2 and L3 large period and high linear complexity is achieved, however when we look at the combining function *f*, if *z*(t) represents key stream bit at time *t*, one can realize the probabilistic relation between output of first LFSR and key stream bit as: $P(z(t) = x_1(t)) = P(x_2(t) = 1) + P(x_2(t) = 0)P(x_3(t) = x_1(t)) = \frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{3}{4}$.

The output of first LFSR is equal to key stream bit at any time with a probability of 3/4. Thus, one can see that Geffe generator has weaknesses considering correlation attack.

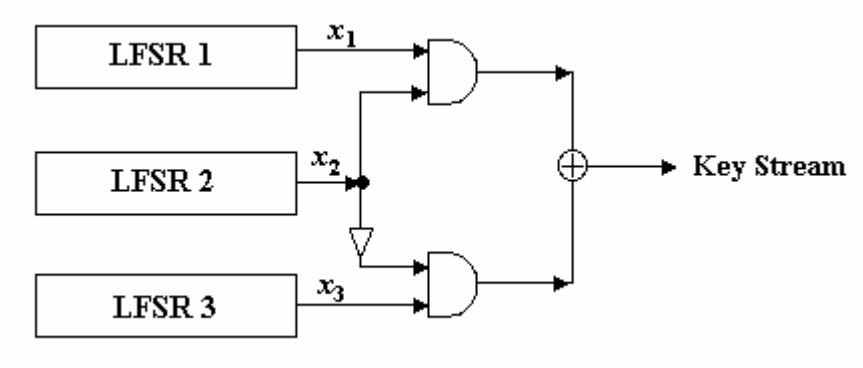

Figure 2.8. The Geffe generator

Therefore, to have a secure nonlinear combination generator, the combining function *f* must have high algebraic degree, high nonlinearity and a high order of correlation immunity. Also *f* must be a balanced function, which has equal number of ones and zeros in the output column of its truth table, to provide key stream sequences with good statistical properties.

2.2.2. Nonlinear Filter Generators

This type of generator is not so different from nonlinear combining generators. In this case, instead of giving outputs of several LFSRs to nonlinear function *f*, appropriate bits of a single LFSR are given. A simple example of nonlinear filter generator is depicted in Figure 2.9, now the function *f* is called as the filter function. Actually, not all elements of the LFSR need to be taken as inputs to the filtering function.

The period of the key stream sequence is 2^n-1, if the LFSR is maximal length register and has a length of *n* bits. The maximum value for the linear complexity of the output sequence is computed as $LC = \sum_{i=1}^{m} \binom{n}{i}$, where *LC* and *m* denote the linear complexity and nonlinear order of the function. The same danger as low correlation immunity can be also valid for the nonlinear filter generators. Also, the same criteria must be concerned for the filter function as in the case of nonlinear combining function.

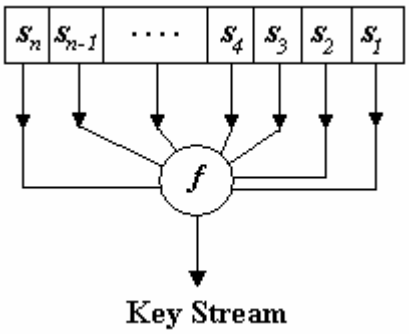

Figure 2.9. A nonlinear filter generator, where a single *n* bit-LFSR's bits are combined with a nonlinear Boolean function *f* to produce key stream sequence

2.3. Clock-controlled Stream Ciphers

The main idea behind an LFSR based clock-controlled stream cipher, is to control the number and time of clockings of the LFSRs using some irregular mechanism. This mechanism can depend on the output of another LFSR or some other internal variables of the cipher. By means of clocking the LFSRs at different rates, the linearity of the system is destroyed and attacks based on a regular clocking of the LFSR become harder. Many stream ciphers using non-linear combining functions are susceptible to the correlation attacks such as fast correlation attacks firstly described in [14]. On the other hand, using irregular clocking reduces the power of correlation attacks and provides practical resistance to the fast correlation attacks. To understand the properties of clock-controlled ciphers, let us give descriptions of some its popular applications.

2.3.1. Alternating Step Generator

Alternating step generator is consisted of three LFSRs denoted as R1, R2 and R3 respectively [15]. The LFSR R1 controls the clocking of the other two, R2 and R3. In each clock cycle 1 bit key stream is generated by XOR'ing the outputs of R2 and R3. The clocking mechanism works as follows: Firstly, R1 is clocked, if its output is 1, then R2 is clocked and R3 is not clocked. On the other hand, if the output of R1 is 0, then R3 is clocked and R2 is not clocked. Whether or not a generator register (R2, R3) is clocked, it gives its output to key stream generation process; if it is not clocked, it repeats its output. The period of the produced key stream sequence can be expressed as: Let length of R1, R2

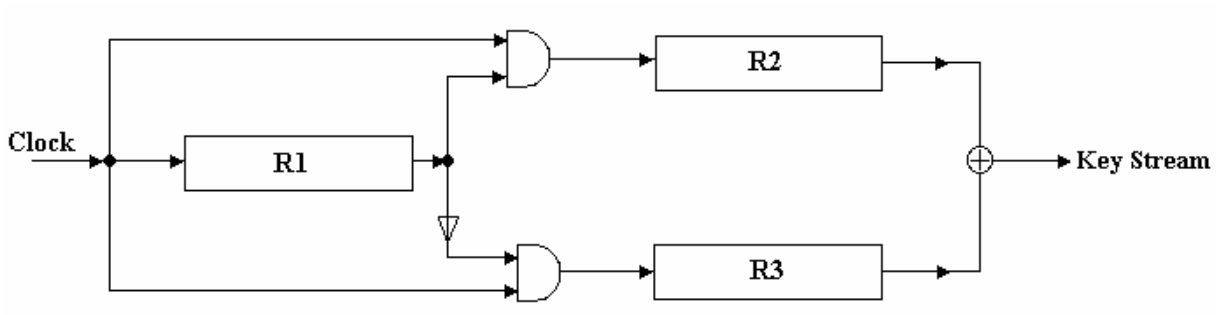

Figure 2.10. The alternating step generator

and R3 be denoted by L1, L2 and L3 respectively. If the period of the output sequence (for any initial state) of a non-singular FSR of length L is 2^L, then this output sequence is called a de Bruijn sequence [2]. If R1 produces a de Bruijn sequence with a period 2^{L1}, R2 and R3 are maximal length LFSRs, and gcd(L2,L3)=1, then period of the key stream sequence becomes 2^{L1} (2^{L2}-1) (2^{L3}-1), where gcd(a,b) stands for greatest common divisor of a and b. For its linear complexity there are lower and upper bound within the same conditions. Let *LC* denote the linear complexity of the key stream sequence; then linear complexity becomes: $2^{L1-1}(L2+L3) < LC \leq (L2+L3)2^{L1}$. Thus if L1 is chosen enough high, the generator produces key stream sequences with large period and high linear complexity. From the point of a cryptanalyst; since the first register R1 controls the clocking of the other two, if the content of the register R1 is guessed, then an attacker can have information about future state transitions of R2 and R3. As a result, he can obtain the internal bits of R2 and R3 from just guessing the bits of R1 which costs approximately 2^{L1} operations. However, if one chooses length of R1 enough high such as 128 bit, such an attack, which can be classified a guess-determine type attack, requires about 2^{128} steps and becomes infeasible.

2.3.2. Shrinking Generator

The shrinking generator is a relatively new stream generator [16]. It uses two LFSRs denoted as R1, R2 which have lengths of L1 and L2 respectively as depicted in Figure 2.11. The generator produces key stream bits as follows: Firstly, both of the LFSRs are clocked. If the output of the first register is 1, then key stream bit takes the value of the output of the R2. If it is zero, no key stream bit is generated and output of second register

is discarded. The main principle of the generator is simple, but very effective, and looks secure. The period of the sequence produced by the shrinking generator is: 2^{L1-1} (2^{L2}-1) if the gcd(L1,L2)=1. This stems from the fact that, R1 gives output 1 exactly 2^{L1-1} times out of its period 2^{L1}-1, incase of it is a maximal length register. The proof of this theorem can be found in [16]. Also, within the conditions the lower and upper bounds for linear complexity of the key stream sequence generated by the shrinking generator is $2^{L1-2}L2 < LC \leq 2^{L1-1}L2$. As can be seen by selecting the length of registers properly high period and high linear complexity can be realized. The generator also seems secure considering different type attacks. As in the case of alternating step generator, a guess-determine type attack can be applied; firstly bits of R1 is guessed, then contents of R2 deduced within known generated key stream sequence. By increasing the length of register the resistance of the cipher against such an attack is increased. Another important point for shrinking generator is its feedback polynomials must be dense, to avoid any vulnerability. Moreover, though it is not related security of the generator, one implementation problem is that the output rate is not regular, because generation of key stream bit depends on whether R1 outputs 1; if first LFSR has a long string of zeros then the generator outputs nothing. In [16], the use of buffering to solve this problem has been suggested.

Actually, there is a variant of shrinking generator called the self-shrinking generator. In this case, instead of using two LFSRs, a single LFSR is used and pairs of bits are taken from it. System operates as follows: Clock the LFSR twice; if the first bit in the pair is 1, the output of the generator is the second bit. If the first bit is 0, discard both bits and try again. While the self-shrinking generator requires about half the memory space as the shrinking generator, it is also half the speed [1].

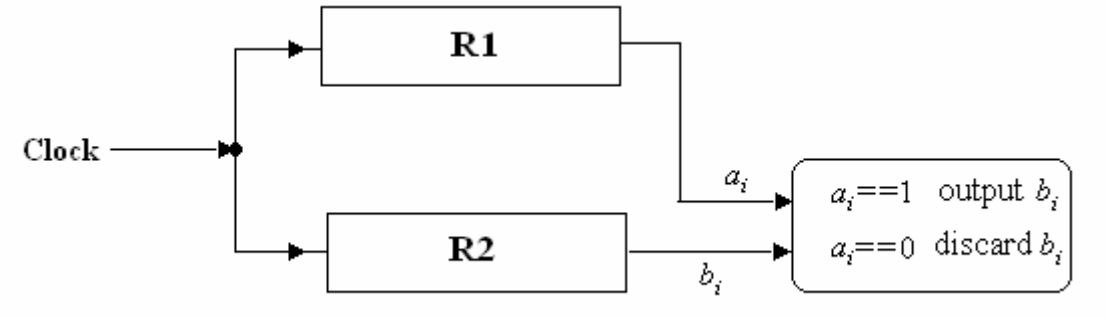

Figure 2.11. The shrinking generator

2.3.3. A5/1

GSM uses A5 stream generator to encrypt digital user data transmitted from mobile station to the base station and base station to the mobile station. A5 stream cipher has two major variants: A5/1 is the stronger version used in western European countries and A5/2 is the weaker version used in the other countries. A5/1 stream cipher is a binary linear feedback shift register based key stream generator. It combines three LFSRs of lengths 19, 22 and 23 bits which are denoted by R1, R2 and R3 respectively [8]. All of these registers have primitive feedback polynomials and each register is updated according to its own feedback polynomial. The taps of R1 are at bit positions 13, 16, 17, 18; the taps of R2 are at bit positions 20, 21; and the taps of R3 are at bit positions 7, 20, 21, 22. The three registers are maximal length LFSRs with periods 2^{19} -1, 2^{22} - 1, and 2^{23} -1, respectively. The output of A5/1 is produced by XOR'ing the most significant bit of each register as shown in Figure 2.12.

Each LFSR has a single clocking tap in bit 8 for R1, bit 10 for R2 and bit 10 for R3; denoted as C1, C2 and C3 respectively. Clocking mechanism of each LFSR is determined according to the majority rule: In each clock cycle majority of C1, C2, and C3 is calculated and two or three registers whose clocking tap value is the same as majority bit are clocked [8]. Since at each clock cycle at least two LFSRs are clocked, an individual LFSR moves with probability 3/4 and stops with probability 1/4.

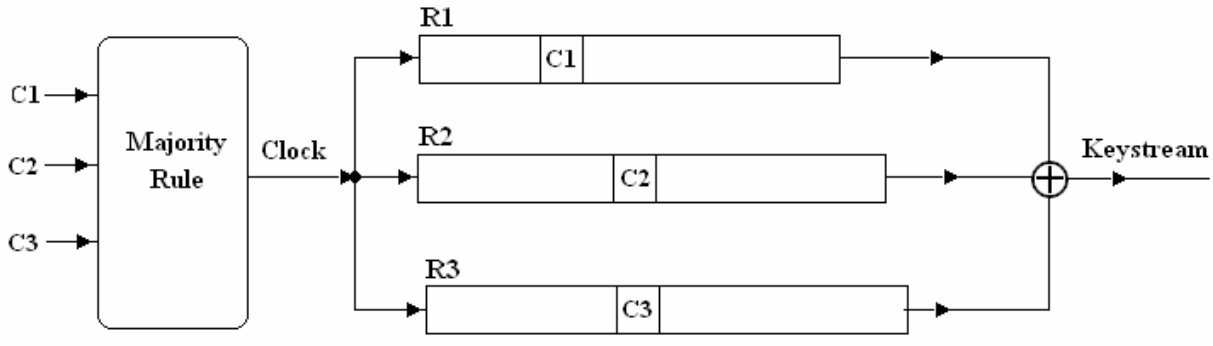

Figure 2.12. The A5/1 stream cipher

The initial state of A5/1 is carried out as follows: All of the registers are first zeroed and then 64 bit secret session key K and 22 bit frame number F_n XOR'ed (ignoring

majority rule) in parallel into the least significant bits (lsb) of the three registers. In the next step, all LFSRs are clocked for 100 clock cycles according to majority rule, however no output is produced. Finally, three LFSRs are clocked according to majority rule to generate 228 bits of key stream sequence.

Attacks against A5 algorithms have been presented in different papers [8, 17, 18, 19, 20]. In [8], it is shown that cryptanalysis of A5/1 can be performed on a single PC with a few minutes of computational time and about 150-300 Gbytes of memory. Most of attacks against A5/1 make use of the security flaws in clocking mechanisms of the algorithm. In [17] and [18] divide & conquer attacks have been applied. According to these studies: Since the clocking tap positions of R1, R2 and R3 are known, linear equations about the LFSRs content can be obtained by guessing the some bits before the clocking tap bits. It is shown that by using these linear equations A5/1 can be cryptanalized. Considering these attacks, some designs have been proposed in different studies to overcome the weaknesses of A5/1 such as in [21] and [22].

2.3.4. ORYX

The ORYX cipher is a stream cipher that is used to encrypt wireless digital data as a key stream generator. The output of the generator is a pseudo-random stream of bytes. The generated key stream is XORed with the plaintext to get the ciphertext. As in case of the most stream ciphers, to recover the plaintext from the ciphertext, same key stream sequence is XOR'ed with the ciphertext at the receiver side.

The ORYX cipher is consisted of three 32-bit LFSRs denoted as $LFSR_A$, $LFSR_B$, and $LFSR_K$, and uses an S-box [10]. The S-box is used for a permutation operation of the numbers between 0 – 255. The block diagram of ORYX is shown Figure 2.13, where P_K, P_B, P_{A1} and P_{A2} represent the feedback functions of $LFSR_K$, $LFSR_B$, and $LFSR_A$ respectively.

The primitive feedback polynomial of $LFSR_K$ is as follows:

$$x^{32} + x^{28} + x^{19} + x^{18} + x^{16} + x^{14} + x^{11} + x^{10} + x^9 + x^6 + x^5 + x + 1$$

$LFSR_A$ uses two different primitive feed back functions which are:

$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

and

$x^{32} + x^{27} + x^{26} + x^{25} + x^{24} + x^{23} + x^{22} + x^{17} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^2 + x + 1$

Finally,the $LFSR_B$ has the following feedback function:

$x^{32} + x^{31} + x^{21} + x^{20} + x^{16} + x^{15} + x^6 + x^3 + x + 1$

The algorithm works in the following manner: Firstly, $LFSR_K$ is clocked once with its feedback function. $LFSR_A$ is stepped once using either one of its two feedback polynomials. The decision of which polynomial depends on one of the high eight bits of $LFSR_K$. Also, $LFSR_B$ is clocked either once or twice depending on another one of the high eight bits of $LFSR_K$. Then, the last eight bits of $LFSR_K$ is added to the last eight bits of $LFSR_A$ after being permuted with S-box and the last eight bits of $LFSR_B$ after being permuted with S-box, with modulus 256 to create 8 bits of key stream.

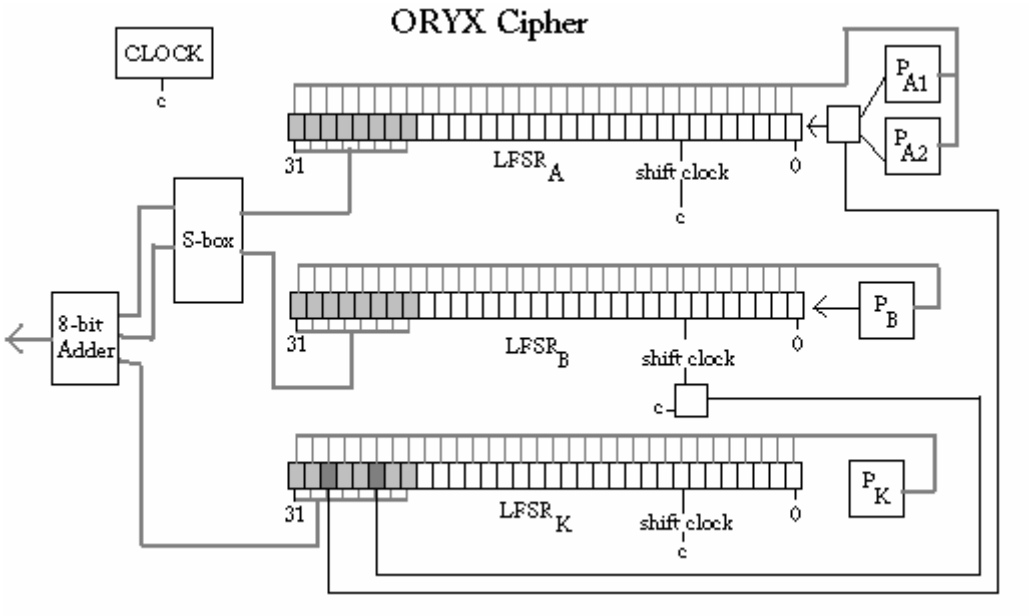

Figure 2.13. The ORYX stream cipher [23]

ORYX was firstly cryptanalyzed by D. Wagner et. al. in [10]. It is shown that by using a divide and conquer attack with an amount of 25-27 byte known plaintext; the

stream cipher can be easily cryptanalyzed in 2^{16} time complexity. Thus, one can say that ORYX is not a secure stream cipher.

3. REQUIRED KEY STREAM PROPERTIES OF A STREAM CIPHER

A stream cipher can not be considered as secure and convenient for stream cipher applications unless its produced key stream sequences have large periods, high linear complexities and provides certain good statistical properties. In other words, the produced key stream sequence must have a guaranteed minimum period which is longer than the length of the message. Also the ciphertext must appear to be random, so that an attacker can not reveal any information about the plaintext from it.

In this chapter, we will focus on these properties from a clock-controlled stream ciphers point of view. In Section 3.1, the period concept and its mathematical expressions for some clock-controlled generators will be given. The linear complexity and linear complexity profile will be discussed with its upper and lower bounds for some clock-controlled generators in Section 3.2. Finally, Section 3.3 will present the details of some important statistical test suits such as FIPS 140-2, NIST that are used to determine whether or not the generated key stream sequences posses some randomness properties [24, 25].

3.1. Period of Generated Key Stream Sequences

The LFSR based stream generators can be thought as a type of finite state machines and since any finite-state machine produces a sequence that is eventually periodic, we can from now on assume that the binary key stream sequence has a period P. For example, if we have a periodic sequence with period P like s_1, s_2, ..., s_P then each element of the sequence repeats itself after P elements as: $s_i = s_{i+P}$ for $1 \leq i \leq P$.

In previous chapter, it is mentioned that a maximal length LFSR with n bits length has a period of 2^n-1. Also, an LFSR can obtain maximal period if it feedback function is a primitive polynomial. Although algebraically analyzing of LFSRs is simple, mathematical modeling and expression for some LFSR based clock-controlled generators is not an easy matter such as mutual clock-controlled stream ciphers. In the following subsections, we

will give period expressions for some clock-controlled generators and these expressions will be used in period evaluation of ANER-H and CSDS in Chapter 5.

3.1.1. Period of a Basic Clock-controlled Generator

In this part, our basic clock controlled generator is consisted of two LFSRs denoted as CR and GR. The CR is the control register which controls the clocking of the second register and the GR is the generator register which is responsible for generating key stream bits. Let the period of the CR be λ, and GR be a maximal length register with length *m* and period P_{GR}. Also, the GR generates sequence of $b = \{b_i\}_{i\geq 0}$ and the CR produces sequence of integers $a = \{a_i\}_{i\geq 0}$, not necessarily to be binary . The stream cipher works as follows: In each clock cycle, the GR is clocked according to the value of a_i. For example if $a = \{2,3,1....\}$, then GR is clocked twice in the first cycle, three times in the following, once in the next and so on. After each clocking of GR, the output of GR becomes the key stream bit as shown in Figure 3.1. Let us continue on the example: If the GR outputs the sequence $b = \{1,0,0,1,1,0,1,1,0...\}$, the key stream sequence, *z*, becomes as z={0,0,1…}, since it is clocked 2, 3, 1 and so on according to the sequence $a = \{2,3,1....\}$. So, the key stream is produced as: Firstly, the initial key stream is $z(0) = b(a_0)$. After output z(*t*) has been produced, the CR determines the nonnegative integer *a(t)* and the GR is clocked *a(t)*

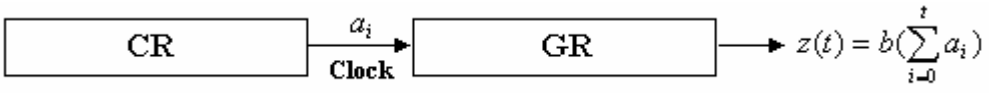

Figure 3.1. The basic clock-controlled generator

times. Next, it produces the next output $z(t+1)$. Lastly, the CR is clocked once and become ready for the next iteration. This operation can be formulated as below:

$$z(t) = b(\sum_{i=0}^{t} a_i), \text{ for } t \geq 0 \qquad (3.1)$$

In fact, z is the '*'marked bits of the sequence b as shown in Figure 3.2. Thus the key stream sequence is the decimated sequence of GR output b with depending on CR sequence a.

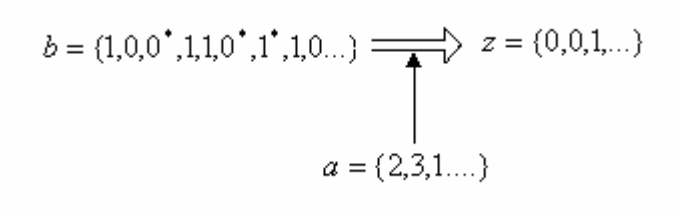

Figure 3.2. Decimated sequence of GR output b

The period of such a system can be shown in terms of GR and CR parameters. Let S denote the summation of clockings of GR in CR's period duration which is λ. Then S can be expressed as:

$$S = \sum_{k=0}^{\lambda-1} a_k \tag{3.2}$$

In [26], it is shown that the key stream sequence produced by such a generator can have a maximum period as P_z:

$$P_z = \frac{\lambda P_{GR}}{\gcd(S, P_{GR})} \tag{3.3}$$

According to [26], the generator can reach this limit, if one of two conditions are satisfied:

- Degree m of $f(x)$ is prime and S is not a multiple of $\frac{P_{GR}}{\gcd(P_{GR}, q-1)}$, where $f(x)$ is the feedback function of GR over GF(q).

- $f(x)$ is a primitive polynomial and $\gcd(S, P_{GR}) \le q^{m/2}$.

For our case, the feedback function is defined over GF(2), so q = 2. To make the subject clearer, let us give two examples, one violates the conditions and the other satisfies.

3.1.1.1. Example 1. Let the GR have a primitive feedback polynomial over GF(2) as $f(x) = x^4 + x + 1$. Also, the CR produces the sequence $a = \{2,3\}^{\infty} = \{2,3,2,3,...\}$ with a period of two, so λ=2. Since the GR has a primitive feedback polynomial and $\{1,1,1,1\}$ as its initial state, it outputs the sequence $b = \{1,1,1,1,0,0,0,1,0,0,1,1,0,1,0\}^{\infty}$ with a period of 15. When we decimate sequence *b* according to a, we obtain the key stream sequence as $z = \{1,0,1\}^{\infty}$. As can be seen the period of the sequence is 3. If it satisfied one of the two conditions given above, it would have a period according to (3.3). For this example, *S* is 5 and P_{GR} is 15, *m*, the degree of *f(x)*, is 4 and *q* is 2 due to *f(x)* is defined over GF(2) . From here, $\gcd(S, P_{GR}) = \gcd(5,15) = 5$ which exceeds $q^{m/2}$=4. Thus, the second condition is not satisfied, although *f(x)* is a primitive polynomial. When we look at whether or not first condition is met, we can see that the degree *m* of *f(x)* is not a prime number, though *S* is not a multiple of $\frac{P_{GR}}{\gcd(P_{GR}, q-1)} = 15$. So, first condition is also not satisfied. If one of these two conditions were satisfied, the period would be $P_z = \frac{\lambda P_{GR}}{\gcd(S, P_{GR})} = \frac{2*15}{5} = 6$. However, a sequence can have the maximum period, although neither of the two conditions is met. Therefore we can say that, satisfaction of one of the two conditions guarantees the maximum period given in (3.3), but they are not the necessary conditions. For example, let the CR sequence be $a = \{3,2\}^{\infty} = \{3,2,3,2,...\}$, no other thing changes. Now the key stream sequence becomes as $z = \{1,0,0,1,1,1\}^{\infty}$ which has period of 6. As can be seen the *S* value and the feedback polynomial are not changed, so still two conditions are not met; but now the sequence has the maximum period as 6.

3.1.1.2. Example 2. For this example, the GR have a feedback polynomial over GF(2) as $f(x) = x^3 + x^2 + x + 1$. The clock control register CR generates the sequence $a = \{2,1,3,1\}^{\infty} = \{2,1,3,1,2,1,3,1...\}$ with a period of 4, so for this case λ=4. The GR begins with the state $\{1,1,1,1\}$ and produces the sequence $b = \{0,0,1,1\}^{\infty}$ with a period of 4, so P_{GR} is 4. When we decimate sequence *b* according to a, the key stream sequence becomes as $z = \{1,1,1,1,0,1,0,1,0,0,0,0,1,0,1,0\}^{\infty}$. As can be seen the period of the sequence is 16. For this example; *S* value is 7, *m* the degree of *f(x)* is 3, and $\gcd(S, P_{GR}) = \gcd(7,4) = 1$. The

feedback polynomial of the GR *f(x)* is not a primitive polynomial, so the second condition does not hold. On the other hand, the degree *m* of *f(x)* is prime and *S* is not a multiple of $\frac{P_{GR}}{\gcd(P_{GR}, q-1)} = 4$. Thus, the first condition holds and by using (3.3) we get the maximum period of the sequence as 16 which agrees with observed period.

3.1.2. Period of Alternating Step(r,s) Generator

In [27], a new stream generator named as Alternating Step(*r,s*) Generator has been introduced. This generator is consisted of three FSRs denoted as A, B, C. The first register has a length of L1 and produces a de Bruijn sequence, which is defined in Section 2.3.1, of period 2^{L1}. The registers B and C are maximal length-LFSRs with length of L2 and L3; so with periods of 2^{L2}-1 and 2^{L3}-1 respectively. The generator works as follows: If the output of register A is 1, then B is clocked *r* times and C is not clocked. However, if the output of register A is 0, then C is clocked *s* times and B is not clocked. The XOR of outputs of B and C sets the key stream bit as shown in Figure 3.3.

The register A generates exactly 2^{L1-1} ones and zeros in its period 2^{L1} which is denoted as P_A. That means, B is clocked $r\ 2^{L1-1}$ times and C is clocked $s\ 2^{L1-1}$ times. If S_B represents the summation of clockings of B in P_A duration and S_C represents the summation of clockings of C, then we obtain the following equations:

$$S_B = 2^{L1-1} r \tag{3.4}$$

$$S_C = 2^{L1-1} s \tag{3.5}$$

Let P_B and P_C denote the periods of B and C respectively. Also it can be easily computed that $\gcd(S_B, P_B)=\gcd(r\ 2^{L1-1}, P_B)= \gcd(r, P_B)$ and $\gcd(S_C, P_C)=\gcd(s\ 2^{L1-1}, P_C)= \gcd(s, P_C)$. Let us firstly find the period of decimated version of output of B, defined as P_{DB}, ignoring existence of C. If $\gcd(r, P_B)=1$, by using (3.3), one can obtain the value for the period as:

$$P_{DB} = P_A P_B = 2^{L1}(2^{L2} - 1) \tag{3.6}$$

By making the same assumptions for C, assuming gcd(s, P_C)=1, and ignoring existence of B, it can be obtained that the period of the decimated version of sequence of C defined as P_{DC} is:

$$P_{DC} = P_A P_C = 2^{L1}(2^{L3} - 1) \tag{3.7}$$

If gcd(L2,L3)=1, gcd(r, P_B)=1 and gcd(s, P_C)=1, the period of the key stream sequence represented as P_z will be lcm(P_{DB},P_{DC}), where lcm(a,b) stands for the least common multiplier of a and b. By using (3.6) and (3.7), the period of the key stream sequence is found as:

$$P_z = \text{lcm}(P_{DB}, P_{DC}) = P_A P_B P_C = 2^{L1}(2^{L2} - 1)(2^{L3} - 1) \tag{3.8}$$

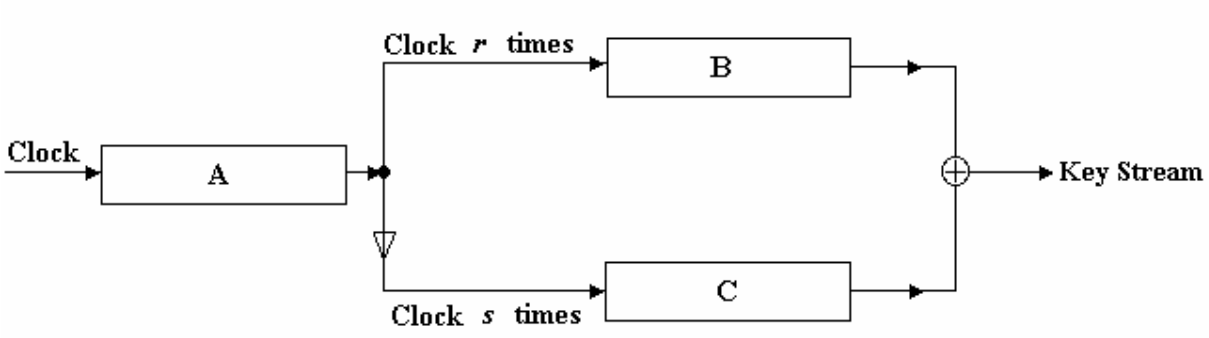

Figure 3.3. The alternating step(r,s) generator

3.1.3. Period of LILI Key Stream Generator

The LILI key stream generator is a simple and fast key stream generator that uses two binary LFSRs and two functions to produce a pseudorandom binary key stream sequence as shown in Figure 3.4 [28]. The first register represented as $LFSR_c$ controls the clocking of the second register which is $LFSR_d$. The second register gives input bits to nonlinear function f_d that generates key stream bits. $LFSR_c$ is a regulary clocked register that gives k bits of inputs to the function f_c which determines how many times the $LFSR_d$ is

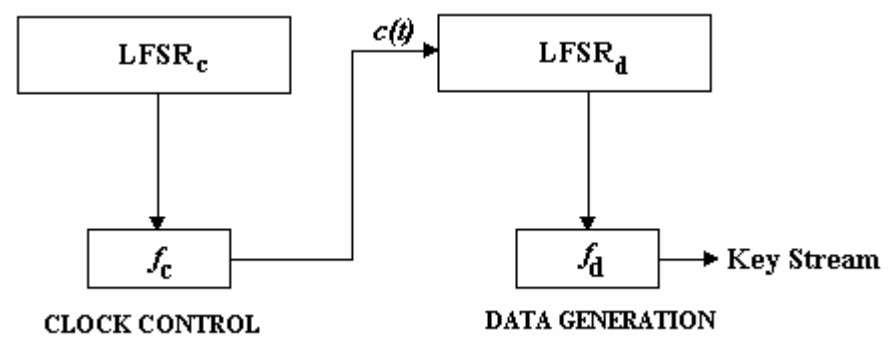

Figure 3.4. The LILI key stream generator

clocked. This function outputs *c(t)* which is an integer such that $c(t) \in \{1,2,\cdots,2^k\}$. The function f_c is a bijective mapping $\{0,1\}^k \rightarrow \{1,\cdots,2^k\}$, thus the distribution of integers *c(t)* is close to uniform. f_c can be expressed as: $f_c(x_1, x_2, \cdots, x_k) = 1 + x_1 + 2x_2 + \ldots + 2^{k-1}x_k$, where 'x's are the input bits that come from $LFSR_c$. At each instant firstly *c(t)* is computed by f_c and then $LFSR_d$ is clocked *c(t)* times. Key stream generation is realized by the function f_d which takes *n* bits of inputs from the $LFSR_d$ and using its nonlinear characteristics produces one key stream bit.

Let L_c and L_d denote the length of the registers $LFSR_c$ and $LFSR_d$ respectively. If *g(t)* represents the output of regularly clocked $LFSR_d$ with nonlinear function f_d and *z(t)* stands for the key stream sequence of LILI system, then *z(t)* can be expressed as: $z(t) = \sum_{i=1}^{t} g(c(i))$, where *g()* is the $LFSR_d$ sequence. Moreover, let P_c, P_d, and P_z denote the periods of $LFSR_c$, $LFSR_d$ and *z(t)* respectively. In [28], it is assumed that $LFSR_c$ and $LFSR_d$ have primitive feedback polynomials, so $P_c = 2^{L_c} - 1$, $P_d = 2^{L_d} - 1$. If S_d represents the summation of clockings of $LFSR_d$ in P_c duration, one can evaluate that $S_d = 2^{L_c - 1}(2^k + 1) - 1$. According to [28]; if P_d is a prime and f_d is not a constant function or if f_d is balanced and S_d is relatively prime to P_d, then the period of the output sequence *z* is given by:

$$P_z = P_c P_d = (2^{L_c} - 1)(2^{L_d} - 1) \tag{3.9}$$

3.2. Linear Complexity of Generated Key Stream Sequences

The linear complexity of a sequence $s = s_1, s_2, \cdots s_k \in F_q$, represented L($s$) as given in Section 2.1, is the length of shortest LFSR, defined over $\in F_q$, that generates the same sequence. If s is the zero sequence s=0,0,…, then L(s)=0 and if no LFSR generates s, then L(s)= ∞. The linear complexity of a sequence can be determined with the Berlekamp-Massey algorithm [11] which finds the feedback function of the shortest LFSR given at least 2L(s) symbols. Since a pure LFSR has low linear complexity as stated in Section 2.1, it can not be a suitable stream generator design. For example, a 100 bit maximal length LFSR has a high period of 2^{100}-1, however by using the Berlekamp-Massey attack, its known 200 bits will be enough to find the coefficients of feedback polynomial that produces the same sequence.

Considering the sequence s given above, let L_N denote the linear complexity of the subsequence $s^N = s_0, s_1,\ldots,s_{N-1}$, for $N \geq 0$. The sequence L_1, L_2, is called the linear complexity profile of s [2]. The linear complexity profile of a sequence can be evaluated by using the Berlekamp-Massey algorithm. The expected linear complexity of a random sequence should closely follow the line L = *N/2*, where L is the linear complexity profile. Of course, that does not mean that a sequence is random if its linear complexity profile follows the line L = *N/2*. The linear complexity profile indicates the manner in which the complexity changes while the bits are being processed.

In the following subsections, we will give linear complexity bounds and expressions for some clock-controlled generators.

3.2.1. Linear Complexity of the Stop and Go Generator

The basic clock-controlled generator is described in Section 3.1.1. For this type, if a_i takes on the values of 1 and 0, then this generator becomes the arrangement of a stop and go generator described in [29]. Let us assume the length of the register CR is n and it has a primitive feedback polynomial. Since a_i takes the value of 1 2^{n-1} times in period of the CR as λ, S denoting the summation of clockings of GR in CR's period becomes $S=2^{n-1}$. The period of the GR is given Section 3.1.1 as $P_{GR}=2^m$ -1, where m is the length of GR. Now

the period of the sequence from (3.3) becomes multiplication of the two individual periods as $P_z = \lambda P_{GR}$, since gcd(S, P_{GR})=gcd(2^{n-1}, 2^m -1) = 1. For the particular case when n=m, $\lambda = P_{GR}$ and the linear complexity of the key stream sequence has its largest possible value which is $n(2^n - 1)$.

3.2.2. Linear Complexity of the Alternating Step(r,s) Generator

The alternating step(r,s) generator is described in Section 3.1.2. The linear complexity of a purely periodic sequence is equal to the degree of its minimal polynomial. The minimal polynomial is the feedback polynomial of the shortest LFSR that can generate the given sequence. Using the same notations of Section 3.1.2, we can give the following the notations: Firstly let us assume that the register C is ignored and key stream is the decimated version of output of B. If gcd(r, P_B)=1, then the minimal polynomial of the sequence is of the form $I(x)^{\beta}$, where $2^{L1-1} < \beta \le 2^{L1}$ and $I(x)$ is an irreducible polynomial of degree L2. In particular, the linear complexity of the sequence, LC_B, becomes as $(2^{L1-1})L2 < LC_B \le 2^{L1}L2$. The same assumptions and ignorance of B can be made for the linear complexity expression of the sequence of register C. For that case it can be written that $(2^{L1-1})L3 < LC_C \le 2^{L1}L3$, if gcd($s$, P_C)=1. Finally, if gcd(L2,L3)=1, gcd(r, P_B)=1 and gcd(s, P_C)=1, the linear complexity of the key stream sequence represented as LC will be:

$$(2^{L1-1})(L2 + L3) < LC \le 2^{L1}(L2 + L3) \tag{3.10}$$

The proofs of the statements mentioned above have been given in [27].

3.2.3. Linear Complexity of the LILI Stream Generator

In Section 3.1.3, the description of LILI stream generator has been given, and for this section also the same notations are valid. In [30], the upper bound on the linear complexity of irregularly decimated maximum-length sequences is given. The lengths of the LFSRs are represented as L_c and L_d and their periods are denoted as P_c, P_d. When a maximum-length sequence of period P_d is non-uniformly decimated by means of a decimating sequence of period P_c, if S_d represents the summation of clockings of $LFSR_d$ in P_c duration,

then the decimated sequence has a maximum linear complexity of $L_d\ P_c$, only if the multiplicative order of 2 modulo $P_d / \gcd(P_d, S)$ is equal to L_d. Notice that if $\gcd(P_d, S) = 1$, this condition will be satisfied. According to [30]; it has been shown that if the decimating sequence is randomly chosen, then the probability that maximum linear complexity is obtained can be made arbitrarily close to one for appropriately chosen L_d and P_c. For the LILI stream generator, the key stream bit is not exactly a pure decimation of output of $LFSR_d$; a nonlinear filter is also used to generate key stream bits. Thus, this effect has to be considered in linear complexity computation. For a non-uniformly decimated nonlinearly filtered sequence can have a maximum linear complexity of $L_d' P_c$ where L_d' is related to maximum linear complexity of a regularly clocked nonlinear filtered sequence. This term depends on the order of nonlinear function and the bit positions of the register which are taken as inputs to the nonlinear function. This relation can be expressed as $\binom{L_d}{m}$, where m is the nonlinear algebraic order of the nonlinear function, as the lower bound of the linear complexity. For the LILI key stream generator, it is presented that the linear complexity of the sequence is lower bounded by $\binom{L_d}{m} P_c$, also lower bounded by $L_d\ P_c$.

3.3. Statistical Properties of Generated Key Stream Sequences

A stream generator can not be regarded as a suitable stream cipher for security applications unless its generated key stream sequences possess certain randomness properties. A binary key stream sequence should be a realization of independent identically distributed random variables with the parameter equal to 0.5. As the key stream deviates from these distributions, it does not seem as a random look like sequence and an attacker can use this weakness to obtain future key stream bits.

Golomb defined a PN-sequence (pseudo-noise sequence) to be a binary sequence of period P that satisfies the three randomness postulates [31]. Actually, they were one of the first attempts to show some necessary conditions for a periodic pseudorandom sequence to look random. It is emphasized that these conditions are far from being sufficient for such sequences to be considered random [2]. Before giving the three postulates, it seems more sensible to give the definitions of run, gap and block. A run of a sequence is a subsequence

of the sequence consisting of consecutive 0's or consecutive 1's which is neither preceded nor succeeded by the same symbol. A run of 0's is called a gap and a run of 1's is called a block. The three postulates are given below for:

- The number of 1's differs from the number of 0's by at most 1 in a cycle.
- At least half the runs have length 1, at least one-fourth have length 2, at least one eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks [2].
- The out-of-phase of the autocorrelation should be constant.

These randomness postulates apply to a complete cycle of a key stream sequence. However, in most stream ciphers systems a complete cycle of the enciphering sequence may never be used. Thus; although the global properties, properties in a complete cycle, should not be discarded, local randomness properties, which are the properties of subsequences shorter length than the whole period, can be more important. Usually, testing the local randomness of a key stream sequence can be realized by using some major statistical tests.

In this thesis, two important test suites which are FIPS 140-2 and NIST Statistical Test Suite [24, 25], are applied to key stream sequences generated by the proposed stream ciphers ANER-H and CSDS. In the following subsections the details of these tests will be given.

3.3.1. FIPS 140-2

FIPS 140-2 test suite is consisted of four statistical tests that are monobit test, poker test, runs test and long run test for randomness. For each trial of the test, a sequence of length 20000 bits are produced from the stream generator. Then the four tests mentioned above are applied to this sequence. If any of the tests fail, then the sequence of the generator fails the test. The descriptions of the tests are given below for a sequence of length 20000 bits.

3.3.1.1. Monobit Test. By means of this test, it is determined that whether or not the number of 0's and 1's in the sequence are approximately equal as would be expected for a random sequence. Let X_1 denote the number of 1's in the sequence, if the inequality $9725 < X_1 < 10275$ is satisfied, the sequence passes this test; otherwise it fails.

3.3.1.2. Poker Test. The purpose of poker test is determining whether the sequences of length 4 bits each appear approximately the same number of times in the stream, as would be expected for a random sequence. It can be done as follows: Firstly the 20000 bit stream is divided into 5000 contiguous 4 bit segments. Then compute the number of each 4-bit segment in the sequence denoting as n_i for $0 \le i \le 15$. At the last step, calculate the term $X_2 = \frac{2^4}{k}\left(\sum_{i=0}^{15} n_i^2\right) - k$, where k=20000/4=5000. If $2.16 < X_2 < 46.17$ is satisfied, sequence passes this test.

3.3.1.3. Runs Test. This test is used to determine whether the number of runs (of either zeroes or ones) of different lengths in the sequence as expected for a random sequence. It is done as: A run is defined as a maximal sequence of consecutive bits of either all ones or all zeros that is part of the 20,000 bit stream. The occurrences of runs of all lengths in the stream sequence should be counted and stored. If the number of occurrences for each length is in the interval given in Table 3.1, then the sequence passes the test, otherwise it fails.

Table 3.1. Passing intervals for the runs test

Length of run	**Required interval**
1	2343-2657
2	1135-1365
3	542-708
4	251-373
5	111-201
6	111-201

3.3.1.4. Long Run Test. A long run is defined to be a run of length 26 or more. So, the long run test is passed if there are no runs of length 26 or more.

3.3.2. NIST Statistical Test Suite

The NIST Test Suite is a statistical package consisting of 16 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators [25]. These tests determine whether or not a variety of different types of non-randomness exists in a sequence. The 16 tests are; The Frequency (Monobit) Test, Frequency Test within a Block, The Runs Test, Test for the Longest-Run-of-Ones in a Block, The Binary Matrix Rank Test, The Discrete Fourier Transform (Spectral) Test, The Non-overlapping Template Matching Test, The Overlapping Template Matching Test, Maurer's "Universal Statistical" Test, The Lempel-Ziv Compression Test, The Linear Complexity Test, The Serial Test, The Approximate Entropy Test, The Cumulative Sums (Cusums) Test, The Random Excursions Test and The Random Excursions Variant Test. For these tests, firstly a P-value, which is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test, is computed. Then according the specified bound for P-value by the user, the sequence passes the test or fails. If a P-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. On the other hand, a P-value of zero means that the sequence seems to be completely nonrandom. A significance level denoting as α can be chosen for the tests. If P-value $< \alpha$ then the sequence fails the test. For example if α chosen as 0.001, this means P-value < 0.001 is required for a failure; thus the sequence would be considered to be non-random with a confidence of 99.9 %. Also the test with a P-value < 0.01 indicates that the sequence is non-random with a confidence of 99%. In this thesis, α is chosen as 0.01 for the tests applied on the stream sequences generated by ANER-H and CSDS. The descriptions and their use of purposes but not their computations are explained in the following parts.

3.3.2.1. The Frequency (Monobit) Test. The purpose of this test is to determine whether the number of ones and zeros in a given sequence are approximately the same as would be

expected for a random sequence. For a random look like sequence it is expected that the ratio of number of ones to number of zeros should be close to 1/2.

3.3.2.2. Frequency Test within a Block. By means of this test, it can be evaluated whether the frequency of ones in an M-bit block is approximately $M/2$, as would be expected for a random sequence. For a sequence of length n and $N=n/M$, where N is number of non-overlapping M blocks, it is recommended that to choose M and N values as $M \geq 20$, $M > 0.01n$ and $N < 100$.

3.3.2.3. Runs Test. The purpose of the runs test is to determine whether the number of runs, whose definition has been given in Section 3.3, of ones and zeros of various lengths is as expected under an assumption of randomness. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

3.3.2.4. Test for the Longest-Run-of-Ones in a Block. This test focuses on the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is approximately the same as with the length of the longest run of ones that would be expected in a random sequence. Notice that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Therefore, only a test for ones is necessary.

3.3.2.5. The Binary Matrix Rank Test. The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence. It focuses on the rank of disjoint sub-matrices of the entire sequence.

3.3.2.6. The Discrete Fourier Transform (Spectral) Test. The purpose of this test is to detect periodic features, i.e., repetitive patterns that are near each other, in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 % [25].

3.3.2.7. The Non-overlapping Template Matching Test. The purpose of this test is to detect generators that produce too many occurrences of a given non-periodic pattern. For this test an *m*-bit window block is used to search for a specific *m*-bit pattern. If the pattern is not found, the window slides one bit position. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

3.3.2.8. The Overlapping Template Matching Test. This test focuses on determining the number of occurrences of specified target strings. Both this test and the Non-overlapping Template Matching test of Section 3.3.2.7 use an *m*-bit window to search for a specific *m*-bit pattern. As with the test in Section 3.3.2.7, if the pattern is not matched in the sequence, the window slides one bit position. The difference between this test and the non-overlapping template matching test is that when the pattern is obtained, the window slides only one bit before resuming the search.

3.3.2.9. Maurer's "Universal Statistical" Test. The main idea behind Maurer's universal statistical test is that it should not be possible to significantly compress the output sequence of a random generator, without loss of information. So it tests whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is evaluated as a non-random sequence.

3.3.2.10. The Lempel-Ziv Compression Test. The purpose of the test is to determine how far the tested sequence can be compressed. This test compresses the candidate random sequence using the Lempel-Ziv algorithm presented in [32]. If the reduction is statistically significant when compared to a theoretically expected result for a random sequence, then the sequence is evaluated as a non-random sequence.

3.3.2.11. The Linear Complexity Test. This test uses linear complexity, which is defined in Section 2.1, to test for randomness. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by longer LFSRs. An LFSR that is too short means that the sequence is non-random.

3.3.2.12. The Serial Test. This test is based on testing the uniformity of distributions of patterns of given lengths. In other words, it tests to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. Random sequences have uniformity; that is, every m-bit pattern has equal probability for appearing in the sequence.

3.3.2.13. The Approximate Entropy Test. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

3.3.2.14. The Cumulative Sums Test. The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the excursions of the random walk should be near zero. For certain types of non-random sequences, the excursions of this random walk from zero will be large [25].

3.3.2.15. The Random Excursions Test. The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. It is based on considering successive sums of the binary bits (plus or minus simple ones) as a one-dimensional random walk. The test detects deviations from the distribution of the number of visits of the random walk to a certain \state," i.e., any integer value. So the purpose of this test is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence.

3.3.2.16. The Random Excursions Variant Test. The focus of this test is the total number of times that a particular state is visited (i.e., occurs) in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk.

4. DESCRIPTION OF PROPOSED STREAM CIPHERS

In this section, the designs of two new binary stream cipher algorithms suitable for high speed communication applications, referred to as ANER-H and CSDS (Clock-controlled Stream cipher with Dynamic S-box Selective) are described. ANER-H stream cipher is a simple and fast stream cipher that consists of four binary LFSRs and uses a 128 bit secret key with a 208-bit initial vector which can be public. The CSDS cipher consists of four LFSRs and uses a 128 bit secret key *K* with a 136 bit initialization vector (IV). The CSDS generates four bits about 2.5 clocking of each shift register, where as ANER-H one bit. Both of the designs satisfy minimal security requirements (long period, high linear complexity, good statistical properties) and provide high resistance to currently known attacks. The main idea behind ANER-H is its characteristic mutual clock control mechanism, while the core of CSDS is the clock-controlling mechanism for the shift registers and the S-boxes that are selected dynamically.

In Section 4.1, a detailed description of ANER-H is given and Section 4.2 discusses the design of CSDS stream cipher.

4.1. The Stream Cipher ANER-H

The ANER-H stream cipher is a simple key stream generator that uses four binary LFSRs to produce a pseudorandom binary key stream sequence as shown in Figure 4.1. This stream cipher has been designed from ideas around the clock controlled generator [2]. Key stream sequence is generated according to the bits come from the three LFSRs (generator LFSRs) of lengths 61, 127, 89 bits denoted by R2, R3 and R4 respectively. The job of the remained LFSR, denoted as R1, is to control the clocking of R2, R3 and R4. Length of the R1 is 59 bits, so LFSRs of ANER-H is totally 336 bits length.

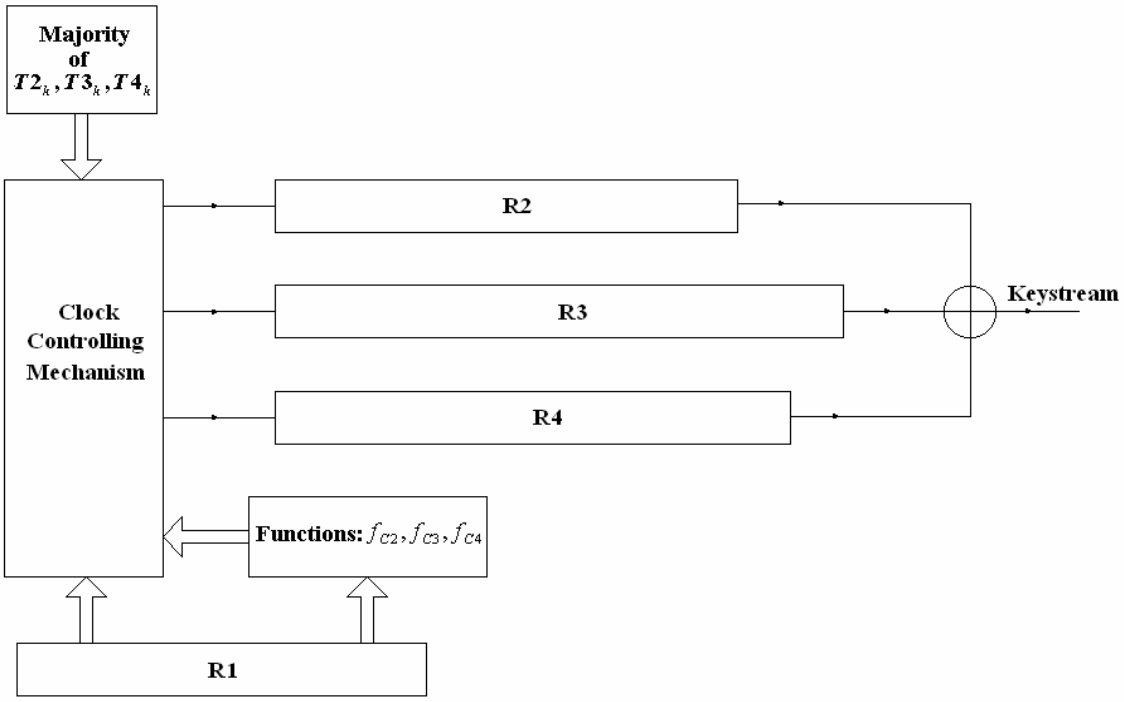

Figure 4.1. The ANER-H stream cipher; R1, R2, R3 and R4 give input to the Clock Controlling Mechanism and R2, R3 and R4 generate the key stream bits

R1 is a regularly clocked LFSR and it has a primitive feedback polynomial as:

$$g(x) = x^{59} + x^{52} + x^{44} + x^{36} + x^{29} + x^{22} + x^{14} + x^{7} + 1 \tag{4.1}$$

R1 gives two bit-input to the each of three clock control functions which are denoted as f_{C2}, f_{C3} and f_{C4}. These functions are given by:

$$f_{C2}(R1(19), R1(37)) = 2R1(19) + R1(37) + 2 \tag{4.2}$$

$$f_{C3}(R1(27), R1(49)) = 2R1(27) + R1(49) + 2 \tag{4.3}$$

$$f_{C4}(R1(32), R1(56)) = 2R1(32) + R1(56) + 2 \tag{4.4}$$

where R1(i) represents the i'th tap bit of R1 at time instant t. f_{C2}, f_{C3} and f_{C4} give integer numbers that are the numbers of clocking of R2, R3 and R4 at time t respectively. If $C_i(t)$

represents the number of clocking for *i*'th register according to $f_{Ci}(t)$ at time *t*, then $C_i(t) \in \{2,3,4,5\}$. As can be seen from (4.2), (4.3) and (4.4), the distribution of the integers $C_i(t)$ is close to uniform. If the clocking mechanism of the generator were like that, each of R2, R3 and R4 would be clocked at least twice and at most five times between the generations of two consecutive key stream bits according to specified bit values of R1. However for ANER-H, clocking mechanism also depends on R2, R3 and R4 as follows: For each key stream bit generation majority of *k*'th clocking tap bit of R2, R3 and R4, $T2_k$, $T3_k$ and $T4_k$ respectively, is calculated and only those registers whose clocking tap value is the same as majority result are clocked $C_i(t)$ times. If there exists a register whose clocking tap value is not equal to majority, it is clocked once. So each of R2, R3 and R4 is clocked at least once and at most five times before each key stream bit is produced. The tap bit locations of the registers are shown in Fig. 4.2. The value of '*k*' is determined according the result of 4R1(23)+2R1(34)+R1(52). For example, let the R1(23), R1(34) and R1(52) be 0, 1 and 1 respectively. Then '*k*' is evaluated as 3. Thus, for this case clocking tap bits of R2, R3 and R4 become $T2_3$, $T3_3$ and $T4_3$. Majority of these clocking tap bits is calculated and two or three registers whose clocking tap value is the same as majority result are clocked $C_i(t)$ times. By using this clocking mechanism non-linearity of the system is increased and stop and go clocking mechanism which may permit attacks is avoided.

Key stream generation of ANER-H is simple; each key stream bit $z(t)$ is generated by XOR'ing the last bits of R2, R3 and R4; 60'th, 126'th and 88'th bits respectively. Following the bit generation, clockings of R1, R2, R3 and R4 are done. The operation of the stream cipher is as follows: Firstly in the initialization part, 128 bit secret key *K* is mixed with 208 bit initial vector to form the initial states of LFSRs. Next, one key stream bit is produced by XOR'ing last bits of R2, R3 and R4. After that, $C_2(t)$, $C_3(t)$ and $C_4(t)$ values (how many times each register is clocked) are obtained according to appropriate tap values of R1. Then, which clocking tap bits of R2, R3 and R4, used in majority rule, are determined with depending on specified bits of R1. After majority of the clocking tap bits $T2_k$, $T3_k$ and $T4_k$ is calculated and the registers whose clocking tap value agrees with majority result are clocked $C_i(t)$ times. If clocking tap value of a register is not same as majority result, it is clocked once. Finally R1 is clocked once. Next key stream bit is generated and the operation continues in this fashion. The maximum allowed length of the

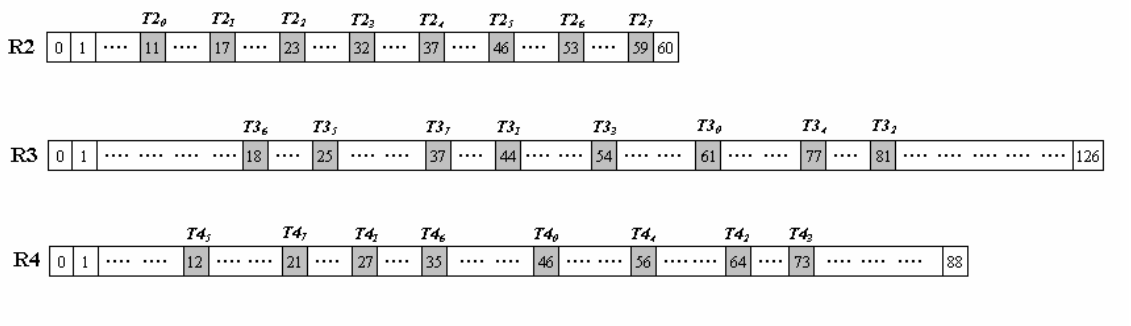

Figure 4.2. Clocking tap bit locations of the generator registers R2, R3 and R4

running key stream sequence set to 2^{54} bits, and then the cipher must be rekeyed (reinitialized). Producing a key stream sequence of length greater than 2^{54} bits in practice is quite unlikely to happen.

R2, R3 and R4 have primitive feedback polynomials as:

$$\text{LFSR R2: } g(x) = x^{61} + x^{53} + x^{45} + x^{38} + x^{30} + x^{23} + x^{15} + x^{7} + 1 \tag{4.5}$$

$$\text{LFSR R3: } g(x) = x^{127} + x^{103} + x^{96} + x^{87} + x^{66} + x^{51} + x^{41} + x^{35} + x^{23} + x^{3} + 1 \tag{4.6}$$

$$\text{LFSR R4: } g(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1 \tag{4.7}$$

4.2. The Stream Cipher CSDS

CSDS stream cipher is a simple and fast key stream generator which has four binary LFSRs denoted by R1, R2, R3 and R4 respectively as shown in Fig. 4.3. According to their functions in the algorithm, these four LFSRs can be categorized into three classes; clock-controlling, S-box selection and key stream generation. R1 has a length of 61 bits and controls the clocking of the other three registers. In each clock cycle, it computes the functions f_{C2}, f_{C3} and f_{C4}, each of those determines how many times R2, R3 and R4 are clocked respectively as given below:

$$f_{C2}(R1(8), R1(38)) = 2R1(8) + R1(38) + 1 \tag{4.8}$$

$$f_{C3}(R1(22), R1(56)) = 2R1(22) + R1(56) + 1 \tag{4.9}$$

$$f_{C4}(R1(16), R1(42)) = 2R1(16) + R1(42) + 1 \tag{4.10}$$

Let $C_i(t)$ represent the number of clocking of R_i according to $f_{Ci}(t)$ at time t for $i \in \{2,3,4\}$. As it can be seen from (4.8), (4.9) and (4.10), $C_i(t) \in \{1,2,3,4\}$ whose distribution of elements is close to uniform. So, each of R2, R3 and R4 is clocked at least once and at most four times in each clock cycle. Following the clocking of the R2, R3 and

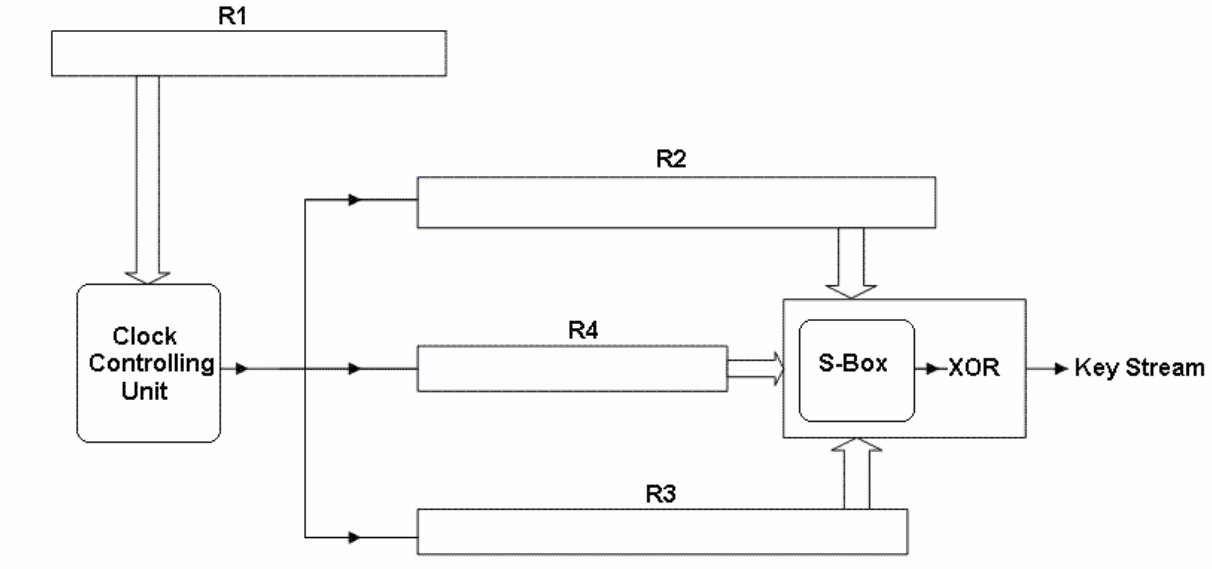

Figure 4.3. The CSDS stream cipher; R1 gives input to the Clock Controlling Unit and R2, R3 and R4 generate the key stream bits within the chosen S-boxes.

R4, R1 is clocked once. In key stream generation CSDS uses 4x16 S-boxes of s^5DES that are given in Table A.1 of Appendix A [33]. In each clock cycle, R4 decides which S-boxes are used by R2 and R3, according to the values of the functions f_{S2} and f_{S3} :

$$f_{S2} = 4\mathrm{R4}(6) + 2\mathrm{R4}(12) + \mathrm{R4}(24) + 1 \tag{4.11}$$

$$f_{S3} = 4\mathrm{R4}(10) + 2\mathrm{R4}(17) + \mathrm{R4}(29) + 1 \tag{4.12}$$

(4.11) and (4.12) determines the orders of S-boxes to use among eight S-boxes for R2 and R3 respectively. For example; if f_{S2} is 4 and f_{S3} is 7, then R2 uses S-box S4 and R3 uses S-box S7 in producing the key stream. R4 has a length of 31 bits and it is irregularly clocked depending on the bits of R1.

The remaining two LFSRs R2 and R3 whose lengths are 89 and 83 bits respectively generate the key stream according to the S-boxes decided by R4 as follows: Let S_j represent the selected S-box for R2 and S_k represent the S-box for R3 where j and k denote the order of selected S-box for R2 and R3, so $1 \leq j \leq 8$ and $1 \leq k \leq 8$. S_j uses six bits of R2 and S_k uses six bits of R3 as input bits with respect to the row-column method shown in Table 4.1. In this table, variables S_j_row and S_k_row save the computed results for row decision of S_j and S_k respectively. In a similar fashion, S_j_column and S_k_column save the computed values for column decision of S_j and S_k. Each of S_j_output and S_k_output keeps the appropriate four bits output. The output of the S-boxes is XOR'ed and constructs four bits of string. Another four bits of sequence is produced by combining the last two bits of R2 and R3. The first two bits of the sequence come from last two bits of R3 and the remained two bits of the sequence come from last two bits of R2. Then by XOR'ing the two 4-bit sequences that are produced by the S-boxes and combination of last bits of the two LFSRs, four bits of key stream is generated. In the following part, whole algorithm is summarized:

- The functions f_{C2}, f_{C3} and f_{C4} are computed according to the specified bits of R1.
- R2, R3 and R4 are clocked with respect to the results of f_{C2}, f_{C3} and f_{C4}.
- R1 is clocked once.
- f_{S2} and f_{S3} are evaluated according to the specified bits of R4; S_j and S_k are determined.
- Each of S_j and S_k contributes four bits output by using appropriate bits of R2 and R3; by XOR'ing them four bits of sequence is constructed.
- Another four-bit sequence is generated by combining the last two bits of R2 and R3
- By XOR'ing these two 4-bit sequences, 4 bits of key stream is generated.

Table 4.1. S-Box row-column method

S_j_row = 2R2(26)+ R2(70)
S_k_row = 2R3(12)+ R3(32)
S_j_column = 8R2(6)+ 4R2(22)+2R2(46)+R2(64)
S_k_column = 8R3(21)+ 4R3(40)+2R3(3)+R3(61)
$S_j\_output = S_j(S_j\_row)(S_j\_column)$
$S_k\_output = S_k(S_k\_row)(S_k\_column)$

The maximum allowed length of the running key stream sequence is set to 2^{58} bits. Then the stream cipher must be rekeyed. In practice, generating a key stream sequence of length greater than 2^{58} bits is quite unlikely to happen. In the initialization process, CSDS uses a 128 bit secret key *K* with a 136 bit initialization vector (IV). The contents of the S-boxes are mixed by adding an integer to all of the elements of S-boxes in mod 16. The value of this integer depends on the secret key *K* and the initialization vector (IV).

The primitive characteristics polynomials of the LFSRs used in the proposed CSDS system are as follows:

$$\text{LFSR R1: } g_1(x) = x^{61} + x^{53} + x^{45} + x^{38} + x^{30} + x^{23} + x^{15} + x^7 + 1 \quad (4.13)$$

$$\text{LFSR R2: } g_2(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1 \quad (4.14)$$

$$\text{LFSR R3: } g_3(x) = x^{83} + x^{72} + x^{61} + x^{51} + x^{41} + x^{30} + x^{20} + x^{10} + 1 \quad (4.15)$$

$$\text{LFSR R4: } g_4(x) = x^{31} + x^{27} + x^{23} + x^{19} + x^{15} + x^{11} + x^7 + x^3 + 1 \quad (4.16)$$

5. SECURITY OF THE CIPHERS

A suitable stream cipher should be resistant against different known-plaintext attacks. In a known-plaintext attack, the cryptanalyst attempts to reproduce the whole key stream or deduce the secret key somehow, from given samples of plaintext and the corresponding ciphertext. Also for cryptographic applications, generated key streams must provide some basic security requirements: large period, high linear complexity and good statistics regarding the distribution of ones and zeroes in the key stream sequence. Therefore in Section 5.1 and Section 5.2, we analyze the key stream properties of ANER-H and CSDS respectively, considering the security requirements. Moreover, we attempt to justify the security of the ANER-H and CSDS by investigating some known-plaintext attacks on the stream ciphers under the assumption that the cryptanalyst knows the whole internal structure of stream cipher in Section 5.3 and Section 5.4 respectively.

5.1. Key Stream Properties of ANER-H

5.1.1. Period and Linear Complexity

All LFSRs of the ANER-H stream cipher have primitive feedback polynomials, so periods of R1, R2, R3 and R4 which are represented as P_1, P_2, P_3 and P_4 are 2^{59}-1, 2^{61}-1, 2^{127}-1 and 2^{89}-1 respectively. Due to mutual clock control, it does not seem possible to establish mathematical results about the period and linear complexity of the cipher. However, we can give upper bounds for the period and linear complexity of the algorithm with ignoring the mutual clock controlling effect. One can see that P_2, P_3 and P_4 are Mersenne Prime. Let S_i represent sum of clockings of i'th register in P_1 duration for $i \in \{2,3,4\}$. Since S_i can not be a multiple of P_i and degree of the feedback polynomials of R2, R3 and R4 are prime, we can make an analogy with alternating step (r,s) generator and can use the equations from (3.4) to (3.8). Then, the period of the generated key stream sequence z can be written as $P_z = \dfrac{P_1 P_2 P_3 P_4}{\gcd(S_2 P_2)\gcd(S_3 P_3)\gcd(S_4 P_4)}$. Notice that, P_2, P_3 and P_4 are prime and S_i can not be a multiple of P_i, so $\gcd(S_i,P_i) = 1$. Therefore P_z becomes:

$$P_z = P_1 P_2 P_3 P_4 \tag{5.1}$$

If we set the values of P_1, P_2, P_3 and P_4 in (5.1), period of the sequence z becomes about 2^{336}. It can be seen that, the period of the sequence is enough high by considering the security requirements.

According [30]; since $\gcd(P_i, S_i) = 1$ for $i \in \{2,3,4\}$, the period of the clock control register becomes a multiplier in the upper bound on the linear complexity of the non-uniformly decimated sequence. Also in [28], it is given that if the decimating sequence is randomly chosen, then the probability that maximum linear complexity is obtained can be made arbitrarily close to one for appropriately chosen generator register lengths and the period of the clock control register. For ANER-H, the clock control register is R1 whose period is 2^{59}-1. Thus; if *LC* denotes the linear complexity of the key stream sequence generated by ANER-H, *LC* is very likely to lower bounded by 2^{59}-1. It is obvious that linear complexity of the sequence is high enough considering the fact that about 2^{60} known plaintext bits must be intercepted in order to perform the Berlekamp-Massey attack [11]. Since the registers will be reinitialized with a different initial vector well before this amount of data is generated, ANER-H is considered to be secure from such an attack.

5.1.2. Statistical Properties

Key stream sequence of the ANER-H stream cipher is investigated by using the statistical tests of FIPS 140-2 and NIST Statistical Test Suite which are explained in Section 3.3, to determine its randomness properties [24, 25]. The FIPS tests are based on performing a pass/fail statistical test on 10000 sequences of 20000 bits each produced by our proposed stream cipher. Also, autocorrelation test (Golomb's 3rd postulate) is applied to the stream sequences of ANER-H and result of one sequence is depicted in Figure 5.1. As can be seen, there is no significant peak compared to the value at the origin. For the NIST tests, 1000 keys for ANER-H cipher are randomly chosen to produce key streams of length 10^6 bits. ANER-H passes FIPS 140-2 in proportion of 99.52%. For the ANER-H, the generated sequences pass the NIST Tests with a significance level of α=0.01. The spectral test result of a sequence is shown in Figure 5.2; as can be seen no more than 5% of

the peaks surpass the 95% cutoff value, so the sequence passes the test. The statistical test results of ANER-H are summarized in Table 5.1.

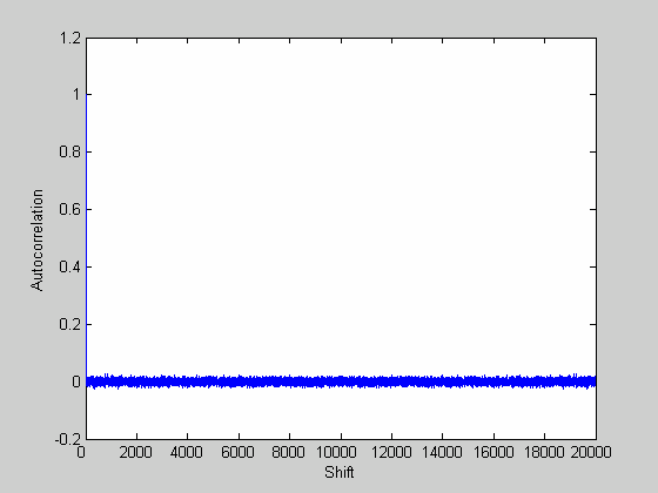

Figure 5.1. Autocorrelation test result of ANER key stream sequence

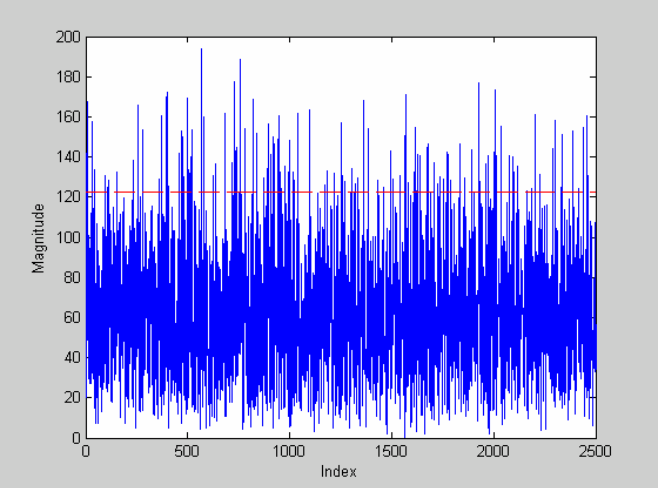

Figure 5.2. Spectral test result of ANER; dashed line represents cutoff value as 122.47

Table 5.1. The statistical test results of ANER-H

The Test Suite	**Success Rate**
FIPS 140-2	99.52 %
NIST	99 %

5.1.3. Throughput Rate

Each of R2, R3 and R4 does not agree with the majority result with a probability of 1/4 for a single key stream bit generation. Therefore, each of R2, R3 and R4 is clocked once with a probability of 1/4. In other words, each of R2, R3 and R4 agrees with the majority with a probability of 3/4. In this case, registers that agree with the majority are clocked $C_i(t)$ times where $C_i(t) \in \{2,3,4,5\}$. Since the distribution of the integers $C_i(t)$ is close to uniform, $P(C_i(t) = j) \approx 1/4$ for $j \in \{2,3,4,5\}$. So each of R2, R3 and R4 is clocked with one of $C_i(t)$ values with a probability of 3/4x1/4 =3/16. If $d_i(t)$ represents the number of clockings of i'th generator register at time t, $d_i(t) \in \{1,2,3,4,5\}$, then expected value of $d_i(t)$ becomes:

$$E\{d_i(t)\} = \frac{1}{4}1 + \frac{3}{16}2 + \frac{3}{16}3 + \frac{3}{16}4 + \frac{3}{16}5 = \frac{46}{16} = 2.875 \qquad (5.2)$$

So, on average each of generator LFSRs is clocked about 2.87 times per key stream bit produced. The throughput rate is approximately $1/2.875 \approx 0.348$ of the rate at which each generator LFSR is clocked. Reduced throughput relative to the LFSRs speed can be a problem when fast data encryption is used and high throughput is required from the cipher. For this problem, using multiple copies of the feedback function of each LFSR for each possible value of $d_i(t)$ can be a solution which is described in [34]. By means of this method, 1 output bit per clock pulse can be achieved; however it results in a moderate cost in hardware. Notice that if the clock is fast enough considering the required application, the offered method may not necessary at all.

5.2. Key Stream Properties of CSDS

5.2.1 Period and Linear Complexity

All LFSRs of the CSDS have primitive feedback polynomials as given in (4.13), (4.14), (4.15), (4.16) and the lengths of those registers denoted as L1, L2, L3 and L4 (61, 89, 83 and 31 bits) are prime numbers. If P_1, P_2, P_3 and P_4 represent the periods of the registers R1, R2, R3 and R4, then they take the values 2^{61}-1, 2^{89}-1, 2^{83}-1 and 2^{31}-1

respectively. Notice that, P_1, P_2 and P_4 are Mersenne prime and P_3 is relatively prime to them. Let S represent sum of clocking of each register R2, R3 and R4 in P_1 duration. R1 has primitive feedback polynomial, so any given stage of R1 takes the value 1 for $2^{61}/2$ times and the value 0 for $2^{61}/2$ -1 times over period P_1. Therefore considering (4.8), (4.9) and (4.10); each of R2, R3 and R4 is clocked once for $2^{61}/4$-1 cases, clocked twice for $2^{61}/4$ cases, three times for $2^{61}/4$ cases and four times for $2^{61}/4$ cases in P_1 duration. Thus, the value of S becomes: $S = (2^{59} - 1) + 2^{59}(2 + 3 + 4) = 5\,2^{60} - 1$. According to (3.4) and (3.8); since S is not a multiple of P_2 , P_3 and P_4 and the degrees of the primitive feedback polynomials of R2, R3 and R4 are prime, the period of the key stream generator P_g can be written as:

$$P_g = \frac{P_1 P_2 P_3 P_4}{\gcd(S, P_2)\gcd(S, P_3)\gcd(S, P_4)} \tag{5.3}$$

After P_g duration, the key stream generator repeats itself. One can see that $\gcd(S, P_2) = 1$ and $\gcd(S, P_4) = 1$, because P_2 and P_4 are Mersenne prime and S is not a multiple of P_2 or P_4. Notice that P_3 is not a prime number, however we know that $\gcd(S, P_3) = \gcd(5\,2^{60} - 1, 2^{83} - 1) = \gcd(2^{60} - 1, 2^{83} - 1)$. This term can be written as $\gcd(2^{60} - 1, 2^{83} - 1) = 2^{\gcd(60,83)} - 1 = 1$. So period of the key stream generator reduces to:

$$P_g = P_1 P_2 P_3 P_4 \tag{5.4}$$

If we set the values of P_1, P_2, P_3 and P_4 in (5.4), the period of the generator becomes $(2^{61} - 1)(2^{89} - 1)(2^{83} - 1)(2^{31} - 1)$, which is about 2^{264}. In fact, we know that in each clock cycle 4 bits are produced; so period of the key stream sequence is a multiple of P_g. Therefore, the period of the sequence is enough high by considering the security requirements.

According [30]; since $\gcd(S, P_i) = 1$ for $i \in \{2,3,4\}$, the period of the clock control register become a multiplier in the upper bound on the linear complexity of the nonuniformly decimated sequence. Also in [28], it is mentioned that if the decimating

sequence is randomly chosen, then the probability that maximum linear complexity is obtained can be made arbitrarily close to one for appropriately chosen generator register lengths and the period of the clock control register. For CSDS, the clock control register is R1 whose period is 2^{61}-1. Thus; if *LC* represents the linear complexity of the key stream sequence generated by CSDS, *LC* is very likely to lower bounded by (2^{61} -1) *A*, where *A* denotes the effect of register lengths L2, L3, L4 and the substitution boxes in the linear complexity *LC*. So if one considers Berlekamp-Massey attack [11] for CSDS, he needs to intercept at least 2^{62} known plaintext bits to realize the attack. Since registers of CSDS will be reinitialized with a different initial vector, before this amount of key stream is generated, CSDS is secure enough for such an attack.

5.2.2 Statistical Properties

In this subsection, key stream sequence of the CSDS stream cipher is investigated by using the statistical tests of FIPS 140-2, NIST Statistical Test Suite and autocorrelation test to determine its randomness properties. For FIPS 140-2, the test is based on performing a pass/fail statistical test on 10000 sequences of 20000 bits each produced by CSDS stream cipher (software implementation in C++ and MATLAB). CSDS passes FIPS 140-2 in proportion of 99.62 %. For the NIST tests, 1000 keys for CSDS cipher are randomly chosen to produce key streams of length 10^6 bits. Also the generated sequences pass the NIST Tests with a significance level of α=0.01. The statistical test results of ANER-H are summarized in Table 5.2. Such a result provides evidence that the tested sequences of CSDS have certain characteristics of randomness. In addition, autocorrelation test and spectral test results of one sequence of CSDS are depicted in Figure 5.3 and Figure 5.4 respectively.

Table 5.2. The statistical test results of CSDS

The Test Suite	Success Rate
FIPS 140-2	99.62 %
NIST	99 %

As can be seen, the autocorrelation is normalized to one at the origin and there is no significant peak compared to the value at the origin and for the spectral test no more than 5% of the peaks surpass the 95% cutoff value. Thus, these tests support the hypothesis that the CSDS stream cipher produces random-looking key stream sequences.

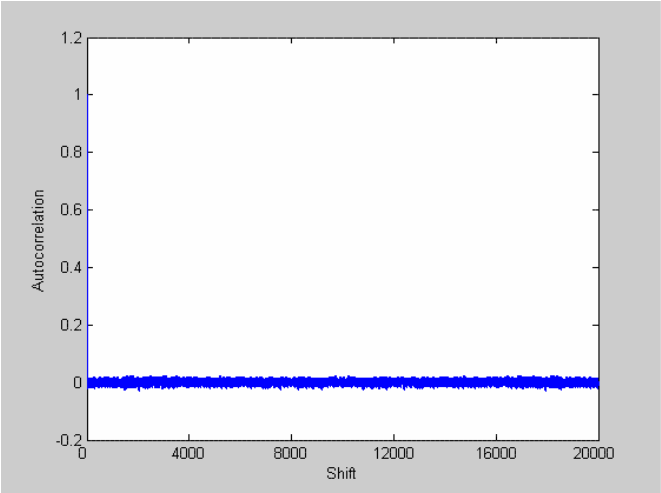

Figure 5.3. Autocorrelation test result of CSDS key stream sequence

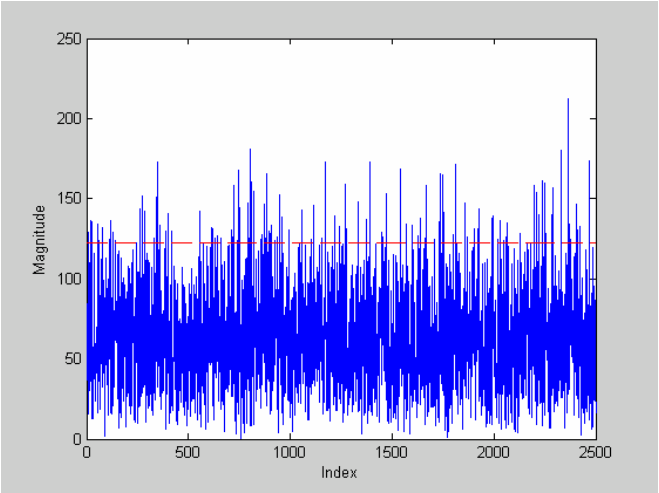

Figure 5.4. Spectral test result of CSDS; dashed line represents cutoff value as 122.47

5.2.3 Throughput Rate

Let $d_i(t)$ represent the number of clocking of i'th generator register of CSDS at time t, $d_i(t) \in \{1,2,3,4\}$ for $i \in \{2,3\}$, then expected value of $d_i(t)$ becomes:

$$E\{d_i(t)\} = \frac{1}{4}1 + \frac{1}{4}2 + \frac{1}{4}3 + \frac{1}{4}4 = \frac{10}{4} = 2.5 \qquad (5.5)$$

So on average each of generator LFSRs is clocked about 2.5 times and produces 4 bits in each cycle. The output rate of the stream cipher becomes 4/2.5=1.6 of the rate at which each generator LFSR is clocked. Therefore CSDS can be considered as a fast key stream generator. Alternatively, using multiple copies of the feedback function of each LFSR for each possible value of $d_i(t)$ can increase the output rate as described in [34].

5.3. Security of ANER-H

In cipher design, the most crucial point that a designer must consider is its resistance against different attacks. Therefore, in this sub-section we consider a number of attacks with respect to ANER-H stream cipher. These attacks are known-plaintext attacks conducted under the assumption that the cryptanalyst knows the whole internal structure of the generator.

5.3.1 Exhaustive Key Search

ANER-H has a key length of 128 bit, so there are 2^{128} possible keys which is approximately $3\ 10^{38}$ keys. Therefore, such an attack appears impractical.

5.3.2 Time-Memory Trade-off Attacks

Golic [17] and Babbage [35] independently described a time-memory trade-off attack to stream cipher. This attack has complexity $T = D = N/M$ and $P = M = N/D$, where T is the time for the actual attack stage, D is the amount of observed key stream, N is total number of solution space for the LFSRs' internal state, M is the amount of required

memory, and P is the time for the preprocessing stage of the attack. Also a new type of time-memory trade-off attack which has the trade-off formula $M^2TD = N^2/4$, has been described by I. Erguler and E. Anarim [36]. In [37], A. Biryukov and A. Shamir presented an improved time-memory trade-off attack as biased birthday attack. They presented techniques for saving memory, offering a more flexible tradeoff $M^2T^2D^2 = N^2$ for any $D^2 \leq T \leq N$. These kinds of attacks can be practical if the state space of the stream cipher is small, for example A5/1 GSM stream cipher [8]. We note that these kinds of attacks do not seem to be applicable to ANER-H, because the solution space N is enough large (2^{336}). Let us consider Biryukov's trade-off: if we choose $M=2^{40}$ (about terabytes), equation $M^2T^2D^2 = N^2$ becomes $TD^2 = 2^{592}$. So time-memory trade-off attacks not seem practical for the ANER-H stream cipher, although an attacker may have large amount of known key stream sequences with $D^2 \leq T \leq N$.

5.3.3 Divide and Conquer Attacks

Divide and conquer attacks are based on "guess some bits and determine the others" principle. The procedure of this type of attack can change according to internal structures of stream ciphers, variations of this technique are presented in different studies such as: [10], [17] and [18]. For clock-controlled registers, attacker usually guesses the initial state of the clock control register, and then deduces the unknown bits of generator register. If attacker applies a similar method on ANER-H, firstly he has to guess all bits of R1 to determine clocking tap bit location and $C_i(t)$ value of each generator register. Since length of R1 is 59 bits, this operation requires 2^{59} workload. Although required complexity is enough high for guessing part, this step does not provide recovering the states of R2, R3 and R4. The reason is obvious; clocking of the generator registers is not only controlled by R1 but also by themselves with the majority rule. Attacker must also make guesses from the generator registers and when total length of three registers, 277-bit, is considered, the difficulty of the attack can be realized. Moreover, since each register is clocked about 2.87 times for each key stream bit, some guessed bits from generator registers may not be useful as expected. So, we can say that divide and conquer attacks do not seem feasible for the ANER-H stream cipher, due to long internal sizes of the registers, and characteristic clocking mechanism.

5.3.4 Correlation Attacks

The most important general attacks on LFSR-based stream ciphers are correlation attacks. Basically, the principle of the attack is based on detecting in some way a correlation between the known output sequence and the output of one individual LFSR.

Key stream generators that are based on regularly clocked LFSRs are susceptible to correlation attacks, including fast correlation attacks, firstly described in [14]. On the other hand, key stream generators consisting of irregularly clocked LFSRs have resistance to this type of correlation attack. There are different types of correlation attacks that can be effective on irregular clocking LFSRs based key stream generators such as unconstrained embedding attack, constrained embedding attack, edit distance attack and edit probability attack. Before proceeding to correlation attack resistance of ANER-H, let's define some parameters that will be used in the remainder of this section. Expectation number of clockings of i'th generator register at time t, $E\{d_i(t)\}$, is given in (5.2). Then deletion rate, P_d becomes:

$$P_d = 1 - \frac{1}{E\{d_i(t)\}} = 1 - \frac{1}{2.875} \approx 0.652 \tag{5.6}$$

Also let $d_{\max} = \max(d_i(t)) = 5$. Considering the unconstrained embedding attack described in [38], the cryptanalyst tries to embed the known key stream Y of length n into a string of length m of X which is the sequence generated by the initial state of the generator register for $m \geq n$. The attack can be successful, if the deletion rate is smaller than 1/2. Since deletion rate, P_d, of ANER-H is about 0.652 which is greater than 1/2, such an attack can not be successful on the ANER-H stream cipher.

In case of constrained embedding attack, the attacker considers information of d_{max} as opposed the idea behind the unconstrained embedding attack. In [38], it is shown that, the constrained embedding attack is successful if the length of the observed output sequence is greater than a value linear in the generator length and super exponential in d_{max}. The attack can not be successful, if the minimum required key stream length is smaller than a value linear in the in the length of the generator register and exponential in d_{max}. Therefore by

making d_{max} sufficiently large, practical security can be improved significantly. For ANER-H stream cipher d_{max} is 5 which requires prohibitively large amount of known key stream. So we can say that constrained embedding attack on ANER-H stream cipher does not seem practical, although it is theoretically possible.

Edit distance and edit probability correlation attacks proposed in [39] and [40] can be considered for ANER-H stream ciphers. However, these attacks are correlation attacks on the initial state of the generator registers implying an exhaustive search over all possible initial states. So, their computational complexity remains exponential and applications of these attacks on the ANER-H stream cipher are not practical either.

5.4. Security of CSDS

A suitable stream cipher should be resistant against different known-plaintext attacks. In a known-plaintext attack, the cryptanalyst attempts to reproduce the whole key stream or deduce the secret key somehow, from given samples of plaintext and the corresponding ciphertext. In this subsection we attempt to justify the security of the CSDS by investigating some known-plaintext attacks on the stream cipher under the assumption that the cryptanalyst knows the whole internal structure of stream cipher. There is no need to state that the cipher is secure against an exhaustive search attack, since its key length, 128 bit, is same those of ANER-H whose resistance is given before.

5.4.1. Time-Memory Trade-off Attacks

Generally in time-memory trade-off attacks, cryptanalyst generates a number of output bits from certain states of the cipher and then keeps these cipher states and their corresponding outputs in pairs in a sorted list. Then he scans a received output sequence to find one of the stored output sequences in the received output sequence. If this occurs, the corresponding cipher state is obtained and from this state the key can be successfully recovered. Time memory trade off attacks can be practical if the state space of the stream cipher is too small. However state space of the CSDS is 2^{264} which is simply too large compared to the key size (2^{128}). The improved time memory trade-off attack presented in [37] can be considered. According to this study, state space N can be distributed between

memory M, computational time T and known amount of data D with respect to the equation $M^2TD^2 = N^2$ for $D^2 \leq T \leq N$. This trade-off equation for CSDS may become as follows; Let available memory to the cryptanalyst be 2^{43} (about terabytes), also state space of CSDS is 2^{264}, N^2 is 2^{528}. The trade-off equation becomes $TD^2 = 2^{442}$. So considering the required time and known data, the time-memory trade-off attacks do not seem to be applicable to CSDS.

5.4.2. Divide and Conquer Attacks

The main idea behind divide-and-conquer attacks is to guess the value of some unknown parts of the stream cipher and from the guessed parts deduce the value of other unknown parts. If the stream cipher has a clock-controlled mechanism, cryptanalyst usually guesses the state of the register which controls clocking of the others. For the CSDS system, the register that concerns with the clocking mechanism is R1 has a length of 61 bits. So if an attacker guesses all bits of the R1 to determine clock controlling functions f_{C2}, f_{C3} and f_{C4}, this process requires 2^{61} workload. Of course this process is not enough to determine the initial state of the stream generator. Attacker also has to know the content of the R4, to obtain the information about which S-boxes are used in the generator registers R2 and R3. This part results in extra 2^{31} workload, due to length of R4. Morover; when we consider the total length of generator registers as 172 bits, the irregular clockings of the three registers and effect of the S-boxes, divide-and-conquer attacks seem impractical for the proposed CSDS system.

5.4.3. Correlation Attacks

The expected value of $d_i(t)$ which is the number of clocking of i'th generator register of CSDS at time t has been given in (5.5). Then one obtains the deletion rate P_d as:

$$P_d = 1 - \frac{1}{E\{d_i(t)\}} = 1 - \frac{1}{2.5} = 0.6 \qquad (5.6)$$

Since the unconstraint embedding attack can be successful if and only if the deletion rate P_d given below is smaller than 1/2 as given in Section 5.3.4, this attack is not successful on CSDS stream cipher.

For the constrained embedding attack, the attacker considers maximum number d_i of consecutive deletion (d_{max}) and as it is mentioned before; the constrained embedding attack can be successful if the length of the observed output sequence is greater than a value linear in the length of the generator register and super exponential in d_{max}. The attack can not be successful, if the minimum required key stream length is smaller than a value linear in the in the length of the generator register and exponential in d_{max}. Thus, by making d_{max} sufficiently large, practical security can be improved significantly. For CSDS stream cipher d_{max} is 4 which requires very large amount of known key stream. Therefore we can say that the constrained embedding attack on CSDS stream cipher does not seem practical, although it is theoretically possible.

Edit distance and edit probability type correlation attacks are also impractical attacks for the CSDS stream cipher. Since these attacks are correlation attacks on the initial state of the generator registers implying an exhaustive search over all possible initial states. Consequently, the computational complexities of the attacks remain exponential. When we consider this fact and the existence of two separate registers as R2 and R3 for key stream generation, it can be seen that applications of these attacks on the CSDS stream cipher are not practical.

6. CONCLUSION

In this thesis, the theoretical background and two practical implementations of clock-controlled LFSR based stream ciphers were described. The main idea behind these ciphers named as ANER-H and CSDS is the use of characteristic irregular clocking mechanisms. The ciphers demonstrate good key stream properties, such as large period, high linear complexity, high throughput rate and good properties of randomness. The mathematical expressions for the linear complexity, period and throughput rate are given and it is shown that these values satisfy the required conditions. To investigate the randomness of the key stream sequences generated by the proposed ciphers, we have used two test suites which are FIPS 140-2 and NIST Statistical Test Suite. The test results show that; CSDS passes the FIPS 140-2 in proportion of 99.62 % and passes the NIST tests with a significance level of α=0.01 which means about 1 % of the sequences are expected to fail. On the other hand, ANER-H passes the FIPS 140-2 in proportion of 99.52 % and passes the NIST tests with a significance level of α=0.01.

Furthermore, the ciphers offer high speed encryption, good scalability and flexibility, so it can be suitable for various security applications. Also, security of the stream ciphers have been analyzed with respect to currently some well known attacks such as exhaustive key search, time-memory tradeoff attacks, divide-conquer type attacks and correlation attacks. It has been shown that both of CSDS and ANER are secure enough with respect to these known attacks. Finally, we can say that within these design characteristics, we consider our stream ciphers ANER-H and CSDS useful for cryptography.

APPENDIX A: S-BOXES OF CSDS

Table A.1. 8 DES-like S-boxes of s^5 DES used in CSDS

S1-box

9	10	15	1	4	7	2	12	6	5	3	14	8	11	13	0
2	13	8	4	11	1	14	7	12	3	15	9	5	6	0	10
10	12	4	7	9	2	15	1	3	6	13	8	14	5	0	11
4	11	1	13	14	7	8	2	10	0	6	3	9	12	15	5

S2-box

6	3	5	0	8	14	11	13	9	10	12	7	15	4	2	1
9	6	10	12	15	0	5	3	4	1	7	11	2	13	14	8
5	8	3	14	6	13	0	11	10	15	9	2	12	1	7	4
6	3	15	9	0	10	12	5	13	8	2	4	11	7	1	14

S3-box

11	5	8	2	6	12	1	15	7	14	13	4	0	9	10	3
7	8	1	14	11	2	13	4	12	3	6	9	5	15	0	10
8	11	1	12	15	6	2	5	4	7	10	9	3	0	13	14
13	2	4	7	1	11	14	8	10	9	15	0	12	6	3	5

S4-box

13	11	8	14	3	0	6	5	4	7	2	9	15	12	1	10
10	0	3	5	15	6	12	9	1	13	4	14	8	11	2	7
6	5	11	8	0	14	13	3	9	12	7	2	10	1	4	15
9	12	5	15	6	3	0	10	7	11	2	8	13	4	14	1

S5-box

12	6	2	11	5	8	15	1	3	13	9	14	0	7	10	4
15	0	12	5	3	6	9	10	4	11	2	8	14	1	7	13
1	12	15	5	6	11	8	2	4	7	10	9	13	0	3	14
6	3	10	0	9	12	5	15	13	4	1	14	7	11	8	2

S6-box

14	8	2	5	9	15	4	3	7	1	12	6	0	10	11	13
1	13	11	8	2	4	7	14	10	6	0	15	5	9	12	3
4	2	9	15	14	8	3	5	10	7	0	12	13	1	6	11
8	11	7	4	13	1	14	2	5	0	9	10	6	15	3	12

S7-box

4	13	10	3	7	0	9	14	2	1	15	6	12	11	5	8
9	0	15	10	12	6	5	3	14	7	1	13	11	8	2	4
13	10	3	9	10	7	14	4	8	6	5	12	11	1	2	15
10	3	12	6	5	9	0	15	4	8	11	1	14	7	13	2

S8-box

1	10	2	12	15	9	4	7	14	3	5	0	8	6	11	13
14	13	7	11	2	4	1	8	0	10	9	6	5	15	12	3
10	15	12	1	9	2	7	4	13	0	6	11	3	5	8	14
4	8	1	2	7	11	13	14	10	5	15	12	0	6	3	9

REFERENCES

1. Schneier, B., *Applied Cryptography Second Edition: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, 1996.

2. Menezes, A., P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press Inc., 1997.

3. Vernam, G. S., "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications", *Journal of the IEEE*, Vol. 55, pp. 109-115, 1926.

4. Singh, S., *The Code Book,* Fourth Estate, London*,* 1999.

5. Zenner, E., *Cryptanalysis of LFSR-based Pseudorandom Generators - a Survey*, http://madoc.bib.uni-mannheim.de/madoc/portal/inst_inf/fulltext_link.php?id=727, 2004

6. Ekdahl, P., *On LFSR Based Stream Ciphers*, Ph.D. Thesis, Lund University, 2003.

7. Fluhrer, S., I. Mantin and A. Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4”, *Eighth Annual Workshop on Selected Areas in Cryptography*, August 2001.

8. Biryukov A., A. Shamir and D. Wagner, “Real Time Cryptanalysis of A5/1 on a PC”, *Proceedings of FSE 2000, Lecture Notes in Computer Science*, Vol. 1978, pp. 1–18, 2001.

9. Golic, J., V. Bagini and G. Morgari, "Linear Cryptanalysis of Bluetooth Stream Cipher", *Proceedings of EUROCRYPT 2002, Lecture Notes in Computer Science*, Vol. 2332, pp. 238-255, 2002.

10. Wagner, D., L. Simpson, E. Dawson, J. Kelsey, W. Millan, and B. Schneier, "Cryptanalysis of ORYX", *Proceedings SAC'98*, *Lecture Notes in Computer Science*, Vol. 1556, pp. 296-305, 1999.

11. Massey, J. L., "Shift-register Synthesis and BCH Decoding", *IEEE Transactions on Information Theory*, Vol. 15, pp. 122-127, 1969.

12. Siegenthaler, T., "Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications", *IEEE Transactions on Information Theory*, Vol. 30, No. 5, pp. 776-780, Sep. 1984.

13. Geffe, P.R., "How to Protect Data with Ciphers that are Really Hard to Break," *Electronics*, Vol. 46, No. 1, pp. 99–101, Jan 1973.

14. Meier, W. and O. Staffelbach, "Fast Correlation Attacks on Certain Stream Ciphers", *Journal of Cryptology*, Vol. 1, pp. 159–176, 1989.

15. Gunther, C.G., "Alternating Step Generators Controlled by De Bruijn Sequences", *Proceedings of EUROCRYPT 1987, Lecture Notes in Computer Science*, Vol. 304, pp. 31-34, 1988.

16. Coppersmith, D., H. Krawczyk, and Y. Mansour, "The Shrinking Generator," *Proceedings of CRYPTO 1993, Proceedings*, *, Lecture Notes in Computer Science*, Vol. 773, pp. 22-39, 1994.

17. Golic, J., "Cryptanalysis of Alleged A5 Stream Cipher", *Proceedings of EUROCRYPT 1997*, *Lecture Notes in Computer Science*, Vol. 1233, pp. 239-255, 1997.

18. Biham, E. and O. Dunkelman, "Cryptanalysis of the A5/1 GSM Stream Cipher", *Proceedings of INDOCRYPT 2000*, *Lecture Notes in Computer Science,* Vol. 1977, pp. 43–51, 2000.

19. Ekdahl, P. and T. Johansson," Another Attack on A5/1", *IEEE Transactions on Information Theory,* Vol. 49, pp. 1-7, 2003.

20. Barkan, E., E. Biham and N. Keller, "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication", *Proceedings of CRYPTO 2003*, *Lecture Notes in Computer Science*, Vol. 2729, pp. 600-616, 2003.

21. Erguler, I. and E. Anarim, "A Modified Stream Generator for the GSM Encryption Algorithms A5/1 and A5/2", *13th European Signal Processing Conference*, 2005, accepted to be presented.

22. Park, M.O., C.Y. Hee and M.S. Jun, "Modified A5 Stream Cipher Using S-boxes," *The International Conference on Advanced Communication Technology 2004*, pp.508-511, 2004.

23. Neel, J., *Cryptanalysis of Mobile Phone Cryptology*, University of Maryland at College Park, www.cs.umd.edu/Honors/reports/cryptanalysis.doc.

24. FIPS PUB 140-2 Security Requirements for Cryptographic Modules, http://csrc.nist.gov/cryptval/140-2.htm.

25. Federal Information Processing Standards Publication FIPS PUB 140-2, Security requirements for cryptographic modules, NIST, http://csrc.nist.gov/cryptval/140-2.htm, 2001.

26. Kholosha, A., "Clock-controlled Shift Registers and Generalized Geffe Key-stream Generator", *Proceedings of INDOCRYPT 2001, Lecture Notes in Computer Science*, Vol. 2247, pp. 287-296, 2001.

27. Kanso, A. A., "The Alternating Step(r,s) Generator", *Securite des Communications sur Internet*, pp. 29-38, Sep. 2002.

28. Simpson, L. R., E. Dawson, J. Golic and W. Millan, “LILI Keystream Generator”, *Proceedings of SAC 2000, Lecture Notes in Computer Science*, Vol. 2012, pp. 248-261, 2001.

29. Beth, T. and F. C. Piper, “The Stop-and-Go Generator”, in T. Beth, N. Cot and I. Ingemarsson (eds.), *Proceedings of EUROCRYPT 1984, Lecture Notes in Computer Science*, Vol. 209, pp. 88-92, 1985.

30. Golic, J. and M. V. Zivkovic, “On the Linear Complexity of Nonuniformly Decimated PN-Sequences”, *IEEE Transactions on Information Theory*, Vol. 34, pp. 1077-1079, 1988.

31. Golomb, S. W., *Shift Register Sequences*, Aegean Park Press, 1982.

32. Ziv, J. and A. Lempel, “A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, Vol. 23, pp. 337-343, 1977.

33. Kim, K., S. Lee, S. Park and D. Lee, “Securing DES S-boxes against Three Robust Cryptanalysis”, *Proceedings of the Workshop on Selected Areas in Cryptography*, pp. 145-157, 1995.

34. Dawson, E., A. Clark, J. Golic, W. Millan, L. Pena and L. Simpson, “The LILI - 128 Keystream Generator”, *Proceedings of. 1st NESSIE Workshop,* http://www.cosic.esat.kuleuven.ac.be/nessie/.

35. Babbage, S., “A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers”, *European Convention on Security and Detection IEE Conference Publication*, No. 408, 1995.

36. Erguler, I. and E. Anarim, “A New Cryptanalytic Time-Memory Trade-Off for Stream Ciphers” , *The 20th International Symposium on Computer and Information Sciences 2005*, *Lecture Notes in Computer Science*, will be published.

37. Biryukov, A. and A. Shamir, “Cryptanalytic Time/Memory/Data Trade-offs for Stream Ciphers”, *Proceedings of ASIACRYPT 2000, Lecture Notes in Computer Science*, Vol. 1976, pp.1-13, 2000.

38. Golic, J. and L. O'Connor, “Embedding and Probabilistic Correlation Attacks on Clock-controlled Shift Registers”, *Proceedings of EUROCRYPT 1994, Lecture Notes in Computer Science*, Vol. 950, pp. 230-243, 1995.

39. Golic, J. and M. J. Mihaljevic, “A Generalized Correlation Attack on a Class of Stream Ciphers Based on the Levenshtein Distance”, *Journal of Cryptology*, Vol. 3, pp. 201-212, 1991.

40. Golic, J. and S. Petrovic, “A Generalized Correlation Attack with a Probabilistic Constrained Edit Distance”, *Proceedings of EUROCRYPT 1992, Lecture Notes in Computer Science*, Vol. 658, pp. 472-476, 1993.