AN IRREGULAR CLOCK-CONTROLLED BINARY STREAM CIPHER WITH NONLINEAR FEEDBACK SHIFT REGISTERS
'THE SAFE STREAM CIPHER'

by

Serhat Eren Arslan

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

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

Graduate Program in Electrical and Electronics Engineering

Boğaziçi University

2006

AN IRREGULAR CLOCK-CONTROLLED BINARY STREAM CIPHER WITH NONLINEAR FEEDBACK SHIFT REGISTERS
'THE SAFE STREAM CIPHER'

APPROVED BY:

Prof. Emin Anarım (Thesis Supervisor)	……
Assoc. Prof. Fatih Alagöz	……..
Assist. Prof. Kıvanç Mıhçak	……..

DATE OF APPROVAL: 13.09.2006

ACKNOWLEDGEMENTS

Firstly, I would like to thank my thesis supervisor Prof. Emin Anarım for his inspiring ideas, kind interest, technical advices and his patience. I would like to mention that without his expertly and patiently guidance up to this point, this work would never be finished.

I would like to thank my family and my fiancée for their unfailing support and influence in my life. They have also shared the name of SAFE stream cipher with me. The name SAFE consists of the first letters of the names of my family members.

Finally, I would like to dedicate this work to my family for their endless support and confidence.

This work has been partially supported by the State Planning Organization of Turkey under the project "Next Generation Satellite Networks and Applications, DPT-03K120250".

ABSTRACT

AN IRREGULAR CLOCK-CONTROLLED BINARY STREAM CIPHER WITH NONLINEAR FEEDBACK SHIFT REGISTERS 'THE SAFE STREAM CIPHER'

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. However, this efficient component is not sufficient when we consider security. The designer should use many nonlinear functions and mechanisms to make the system more resistant against cryptanalysis. A stream cipher should have high period, high linear complexity, good statistical properties and be resistant against most recent successful attacks such as algebraic attacks, correlation attacks, time/memory trade-off attacks, and divide and conquer attacks.

In this thesis, a new stream cipher design is proposed. SAFE is designed to be resistant against algebraic and correlation attacks. In the design phase, the objective was to design a stream cipher with good randomness, high period and linear complexity and resistance against many attacks. The innovation in this thesis is the proposal of nonlinear feedback shift registers instead of linear feedback shift registers to provide resistance against correlation and algebraic attacks. In addition, another innovation is the use of a new irregular decimation algorithm, $EBSG_{variant}$, for increasing the security of the cipher. Keystream properties of the cipher and its resistance with respect to some well known cryptographic attacks are investigated. From the mathematical expressions and simulation results, it is shown that the cipher produces keystream sequences with satisfying basic security requirements and provides high resistance against well known attack types. Finally, we can say that SAFE can be appropriate for both software and hardware applications due to its simple design.

ÖZET

DOĞRUSAL OLMAYAN GERİ BESLEMELİ KAYAN SAKLAÇLI DÜZENSİZ SAAT KONTROLLÜ İKİLİ DİZİ TİP ŞİFRELEYİCİ 'SAFE DİZİ TİP ŞİFRELEYİCİ'

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. Fakat bu randımanlı bileşenler güvenliği dikkate aldığımızda yeterli olmamaktadır. Tasarımcı, sistemi kriptanalize karsı daha güçlü yapmak için birçok doğrusal olmayan fonksiyonlar ve mekanizmalar kullanmalıdır. Bir dizi tip şifreleyici yüksek periyoda, yüksek doğrusal karmaşıklığa, iyi istatistiksel özelliklere ve cebirsel saldırılar, ilinti saldırıları, zaman bellek ödünleşimi saldırıları, böl ve fethet saldırıları gibi birçok başarılı güncel saldırıya karsı dayanıklı olmalıdır.

Bu tezde yeni bir dizi tip şifreleyici tasarımı önerilmektedir. SAFE cebirsel ve ilinti saldırılarına karsı güçlü olması için tasarlandı. Tasarım evresinde hedef, iyi rasgeleliğe, yüksek periyoda ve doğrusal karmaşıklığa sahip ve saldırılara karsı dayanıklı bir dizi tip şifreleyici tasarlamaktı. Bu tezde yapılan yeniliklerde birisi, ilinti ve cebirsel saldırılara karşı dayanıklılığı artırmak için doğrusal geri beslemeli kayan saklaçların yerine doğrusal olmayan geri beslemeli kayan saklaçların önerilmesidir. Buna ek olarak, başka bir yenilik ise yeni bir seyreltme algoritmasının, $EBSG_{variant}$, şifreleyicinin güvenliğini artırmak amacıyla kullanılmasıdır. Ayrıca bu algoritmaların ürettikleri çıktı dizilerinin özellikleri ve algoritmaların bilinen bazı saldırılara karsı dirençleri çalışmada verilmektedir. Matematiksel açılımlar ve benzetim sonuçları ışığında şifreleyicinin istenen minimum çıktı özelliklerinin gereksinimleri yerine getirdiği ve bilinen bazı saldırı tiplerine karşı yüksek dirence sahip olduğu gösterilmektedir. Sonuç olarak, SAFE basit tasarımı sayesinde donanım ve yazılım uygulamaları için uygundur diyebiliriz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVATIONS

A_i	Four bit blocks used for mixing of S-box
B_L	Number of different *L*-stage FSRs producing De Bruijn sequences
b	Searched bit in BSG algorithm
b_i	FSR inner state sequence
C	Ciphertext letter
C_i	Ciphertext bit
$C_{s^j,z}$	Cross-correlation function
Cl_i	Clocking tap value of register *Ri*
c_i	Feedback function coefficients
D	Decryption process
D_e	Number of available data points in a TMTO attack
d_{max}	Maximum deletion rate
E	Encryption process
e_j	BNS output
F	Boolean function
F_n	Frame number
F_q	Finite field on q elements
F_{q^1}	Splitting field
$F_{q^1}^*$	Multiplicative group
f	FSR feedback function
f_C	Nonlinear combining function
f_F	Nonlinear filter function
g	Generator function
h_j	Fixed number of ciphertext symbols that keystream depends on
K	Key letter
K_C	Secret key
K_S	Session key
k_{reg}	degree of the feedback polynomial of the generator register

L	Length of an FSR
L_j	Length of the $LFSR_j$
$L(s)$	Linear complexity of a sequence s
LC	Linear complexity
M	Function that is used to calculate the N value
M_e	Memory needed to construct the lookup table in a TMTO attack
N	Value to determine clocking tap positions of the generator registers
N_e	State space of a cipher in a TMTO attack
n_b	Block length of a plaintext / ciphertext
o_j	Binary discriminator output
P	Plaintext letter
P_e	Time needed for pre-computation phase
P_{GR}	Period of data generating register
P_i	Plaintext bit
P_j	Probability that an LFSR output bit and a keystream bit coincides
P_{KG}	Period of clock-controlled keystream generator
P_{Ri}	Period of register R_i
P_S	Period of SAFE
p_d	Deletion rate
R_i	Name of FSR_i
$\vec{r}$	Vector that is used to calculate the N value
r_i	Bit values taken from register R_1 to form $\vec{r}$ vector
S	Total value of the decimating sequence
S_i	Present state of the generator
$s_i(t)$	output of the FSR_i at time t
$\overrightarrow{s_0}$	Initial state vector
T	Time complexity of TMTO attack
$\vec{v}$	Column vector of S-box
$\vec{w}$	Row vector of S-box
$\vec{y}$	Output vector of S-box
$Z(b)$	Number of vectors of length k in a single period
$z(t)$	Keystream bit at time t
z_i	Output bit of a keystream generator

σ	FSR output sequence
φ	Linear complexity of S-box and $EBSG_{variant}$

ABSG	Alternative bit-search generator
AC	Autocorrelation
AES	Advanced encryption standard
BNS	Binary noise source
BSG	Bit-search generator
CMOS	Complementary metal oxide semiconductor
DCVSL	Differential Cascode Voltage Switch Logic
DES	Data encryption standard
DFT	Discrete Fourier transform
EBSG	Editing bit-search generator
FIPS	Federal Information Processing Standard
gcd	Greatest common divisor
GF	Galois field
GSM	Global system for mobile communications
IV	Initialisation vector
LFSR	Linear feedback shift register
MBSG	Modified bit-search generator
NFSR	Nonlinear feedback shift register
NIST	National Institute of Standards and Technology
OTP	One time pad
TMTO	Time/Memory trade-off

1. INTRODUCTION

To many people, cryptography is a strange, secret code used by only military and secret agencies. This view is strengthened by the fact that many people normally do not know or care about the inherent mechanism of their daily-used technical gadgets. Today, cryptology is an integral part of our lives whether we know it or not. If you use the internet to do bank transfers you use cryptography methods to both identify yourself to the bank, and let the bank identify itself to you, as well as keep your transactions private form other uses of internet. Another common usage area is in cellular phones. In the Global System for Mobile communications (GSM), the mobile system used in Europe, there is a cipher called A5/1 to ensure that your conversation is secure.

These are just a few examples of cryptology used in common public applications. Of course, military and government agencies still use strong cryptography to communicate and the military needs for secrecy have been the primary force behind the developments in this area.

The search for new public-key schemes, improvements to existing cryptographic mechanisms, and proofs of security continues at a rapid pace. Various standards and infrastructures involving cryptography are being put in place. Security products are being developed to address the security needs of an information intensive society.

The rest of the chapter is organised as follows: In section 1.1, general information about the cryptology is given. Section 1.2 describes symmetric-key ciphers, while section 1.3 is focused on the cryptanalysis and the methods of attacks against the ciphers. In addition, section 1.4 gives the outline of the thesis.

1.1 Cryptology

Cryptology is uniting name for a broad scientific field in which one studies the mathematical techniques of designing, analysing and attacking information security services. Cryptology is the study of cryptography and cryptanalysis. Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques and more generally information security services. Cryptography is not only means of providing information security but rather one set of techniques. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met. Some of these objectives are listed in Table 1.1.

Table 1.1. Some information security objectives

Privacy or confidentiality	Keeping information secret from all but those who are authorized to see it.
Data integrity	Ensuring information has not been altered by unauthorized or unknown means.
Entity authentication or identification	Corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.).
Message authentication	Corroborating the source of information; also known as data origin authentication.
Signature	A means to bind information to an entity.
Authorization	Conveyance, to another entity, of official sanction to do or be something.
Validation	A means to provide timeliness of authorization to use or manipulate information or resources.
Access control	Restricting access to resources to privileged entities.
Certification	Endorsement of information by a trusted entity.
Timestamping	Recording the time of creation or existence of information.
Witnessing	Verifying the creation or existence of information by an entity other than the creator.
Receipt	Acknowledgement that information has been received.
Confirmation	Acknowledgement that services have been provided.
Ownership	A means to provide an entity with the legal right to use or transfer a resource to others.
Anonymity	Concealing the identity of an entity involved in some process.
Non-repudiation	Preventing the denial of previous commitments or actions.
Revocation	Retraction of certification or authorization.

Of all the information security objectives listed in Table 1.1, the following four form a framework upon which the others will be derived: (1) privacy or confidentiality; (2) data integrity; (3) authentication and (4) non-repudiation [4].

1. *Confidentiality* is a service used to keep the content of information from all but those authorized to have it. *Secrecy* is a term synonymous with confidentiality and privacy.

2. *Data integrity* is a service that addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorised parties. Data manipulation includes such things as insertion, deletion, and substitution.

3. *Authentication* is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons, this aspect of cryptography is usually subdivided into two major classes: *entity authentication* and *data origin authentication*. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

4. *Non-repudiation* is a service that prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorise the purchase of property by another entity and later deny such authorisation was granted. A procedure involving a trusted third party is needed to resolve the dispute.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities [4].

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 good examples to these approaches. A classic example to these systems is Vigenère cipher.

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 20th century, Gilbert Vernam proposed a new type of cipher known as Vernam cipher that uses a secret key as long as the plaintext [46]. The cipher relies on the algorithm that the plaintext is XORed with a random or pseudorandom stream of data the same length to generate the ciphertext. 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 that is an important feature of the stream ciphers.

1.2 Symmetric-key Ciphers

A *cryptosystem* is a general term referring to a set of cryptographic primitives used to provide information security services. Most often, the term is used in conjunction with primitives providing confidentiality, i.e., encryption. Cryptosystems are mainly classified into two groups according to the key type being used, 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 a 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 the use of a secret key for both encryption and decryption. This key must be kept secret to prevent revealing of the secret information by the potential eavesdroppers. In symmetric-key algorithms, the encryption key can be calculated from the decryption key and vice versa [45].

Public-key systems and symmetric-key systems have advantages over each other. The important points in practice are:

1. public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key management;

2. symmetric-key cryptography is efficient for encryption and some data integrity applications.

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

$$E_K(P) = C \tag{1.1}$$

$$D_K(C) = P \tag{1.2}$$

Symmetric key ciphers are divided into two main categories, block ciphers and stream ciphers. Throughout the thesis, we will mainly focus on the symmetric-key cryptosystems (ciphers).

1.2.1 Block Ciphers

Block ciphers operate on large blocks of plaintext data. They encrypt blocks of data using a fixed transformation. General structure of a block cipher is depicted in Figure 1.1.

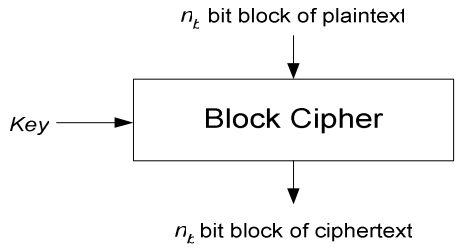

Figure 1.1. General structure of a block cipher

Most well known symmetric-key encryption techniques are block ciphers. Two important classes of block ciphers are s*ubstitution ciphers* and *transposition ciphers*. Substitution ciphers are block ciphers, which replace symbols (or groups of symbols) by other symbols or groups of symbols; while transposition ciphers permute the symbols in a block [4]. Famous block ciphers include the DES (Data Encryption Standard) and the AES (Advanced Encryption Standard). One problem of block ciphers is that patterns in the plaintext are visible for a fixed key. That is; if two plaintext blocks are identical, the block cipher will encrypt them to identical ciphertexts. This is considered as a weakness for an encryption system. There are modes of operations that for example chain the output of one block encryption to the next. The ciphertext block of the first encryption is bitwise added to the plaintext block of the second before encryption. Then the second ciphertext block is added to the third plaintext before encryption, and so on. This solves the problem of patterns but introduces the severe problem of *error propagation*. If the ciphertext is distorted by noise during transmission, all subsequent plaintexts will be distorted during decryption due to this chaining [47].

1.2.2 Stream Ciphers

Stream ciphers form an important class of symmetric-key encryption schemes. They are, in one sense, very simple block ciphers having block length equal to one. A stream cipher operates on individual characters in the underlying alphabet, with a time-varying function. What makes them useful is the fact that the encryption transformation can change for each symbol of plaintext being encrypted. Since the encryption is done using a time-varying function, the problem of patterns in the plaintext being encrypted to identical patterns in the ciphertext is avoided. General structure of a stream cipher is depicted in Figure 1.2.

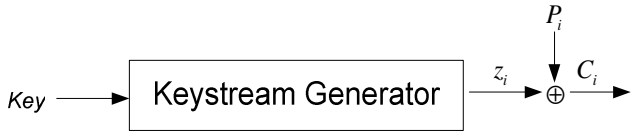

Figure 1.2. General structure of a stream cipher

Since encryption is applied to each bit separately, stream ciphers have limited or no error propagation. In situations where transmission errors are highly probable, stream ciphers are advantageous because of this property. They can also be used when the data must be processed one symbol at a time [4].

1.3 Cryptanalysis and Methods of Attacks

As stated before, cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques and more generally information security services. When evaluating the strength of a cipher, we generally compare it to the generic attack of exhaustively searching over all possible keys to find the right one. This attack is called exhaustive key search. No practical cipher will be more secure than the time it takes to test all keys, so in a sense, this is the highest achievable strength of a cipher. There is however, one famous exception to this, the ultimately secure cipher; the one-time pad (OTP). The OTP is the Vigenère cipher with a key length equal to the length of the message. Shannon shown in 1949 that this system is unconditionally secure. This property means that no matter how big or fast a computer the attacker has, he can never find out which plaintext was sent. The obvious drawback of OTP is that the key must be as long as the message to be encrypted. This is in fact a necessary condition for any cipher claiming unconditional security. Since the key should be secret, one could argue that if we want to send an encrypted message, we first have to send a key that is as long as the message through a secure channel. The OTP is of course unpractical.

For practical ciphers, the situation is not as clear as for the OTP. Often we talk about computationally secure ciphers instead. This means that given the possibilities of today's computers and predicted increase in performance of tomorrow's computers, the adversary cannot defeat the system. We also define the computationally security of a cipher to be the computational effort required, by the best currently known attacks, to break the cipher. Another way of describing the security of a cipher is to try to prove that breaking the cipher is equivalent to solving a difficult mathematical problem, like factoring integers or solving discrete log problem. This way of arguing is called provable security, which is somewhat misleading since there is never a proof of the complexity of the underlying

problem. The notion is that these problems are studied by mathematicians for centuries and are probably very difficult to solve.

The first and most important rule for the designer of a cryptographic primitive is called *Kerckhoff's Principal*: "The security of the encryption scheme must depend only on the secrecy of the key, and not on the secrecy of the algorithms." From *Kerckhoff's Principal* it is assumed that an attacker knows everything concerning the cryptographical system and the protocols, except the secret key. Then we can classify the methods of attack according to the amount of information to the adversary and goal of the attack.

- **Ciphertext-only attack**: All the adversary sees is the ciphertext communicated between sender and receiver. This is the most difficult attack since the attacker has the least amount of information.

- **Known plaintext attack**: In this scenario, the adversary knows both the plaintext and the corresponding ciphertext. It might seem slightly that improbable that both the plaintext and the ciphertext are revealed, but there are many situations where this could happen.

- **Chosen plaintext attack**: Here the adversary has access to an oracle, which can encrypt any given plaintext under the correct key. This is an even more powerful attack than the known plaintext attack, since the attacker can now choose plaintexts that are especially favourable for breaking the cipher.

- **Chosen ciphertext attack**: This is similar to the chosen plaintext attack, but now the adversary has access to two oracles instead, one that encrypts any given plaintext and one that decrypts any given ciphertext except the ones the attack is trying to break. This attack is naturally more powerful than all the previous attacks, since the adversary has more freedom.

1.4 Thesis Outline

The thesis consists of six chapters. Chapter 1 gives the main aspects about the cryptology. Chapter 2 discusses the characteristics of stream ciphers and present information about the types of stream ciphers. Chapter 3 gives information about the nonlinear feedback shift registers. Chapter 4 discusses the irregular decimation algorithms used in stream cipher applications. Chapter 5 consists of the design and hardware considerations of the proposed cipher SAFE. In chapter 6, the security analysis of the proposed cipher is given in detail. Finally, in chapter 7 the conclusion of the study is given.

2. STREAM CIPHERS

As it is given in the previous chapter, a stream cipher inspires the spirit of the onetime pad by using a short key to produce the keystream which appears to be random. Such a keystream sequence is often described as pseudorandom generation of which can be thought as in the field of stream ciphers. Therefore keystream generator can also be known as pseudorandom 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 keystream generator's output is to random, the longer time a cryptanalyst will have breaking it [45].

The stream cipher encryption and decryption can be formulated as follows: Let $z_1, z_2, ..., z_i$ denote the sequence that keystream generator outputs and $P_1, P_2, ..., P_i$ denote the plaintext bits. Then, if $C_1, C_2, ..., C_i$ represents the corresponding ciphertext bits, encryption and decryption are realized according to the equations below respectively.

$$C_i = P_i \oplus z_i \tag{2.1}$$

$$C_i \oplus z_i = P_i \oplus z_i \oplus z_i = P_i \tag{2.2}$$

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

• **Synchronous stream ciphers:** A synchronous stream cipher is a finite state machine for which the keystream is generated from the key, independently of the plaintext message and of the ciphertext [4]. In the encryption side, a keystream generator outputs the keystream bits, one after the other. On the decryption side, another keystream generator produces the identical keystream bits, one after the other. To avoid false decryption so error in communication, the two keystream generators must be synchronised. In case of losing synchronisation during transmission, every ciphertext character after the error will be decrypted incorrectly. To solve this problem, the sender and receiver must

resynchronise their keystream generators before continuing their communication. Techniques for resynchronisation can be reinitialisation 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 single bit or group bits are changed during transmission so error occurs, then only erroneous 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 that produces keystream bits, S_i is the present state of the generator, K_C is the secret key, P_i, z_i and C_i represent plaintext, keystream and ciphertext bits respectively. Most of the stream ciphers are binary additive stream ciphers that are synchronous stream ciphers in which the keystream, plaintext, and ciphertext digits are binary digits and the output function is the XOR of plaintext and keystream sequence.

• **Self-synchronising stream ciphers:** A self-synchronising stream cipher is a finite state machine for which the keystream is generated as a function of the key and a fixed number of the previous ciphertext symbols [47]. In other words, for this type of stream ciphers each keystream bit is produced within a function of a fixed number of previous ciphertext bits. Since the keystream depends on a fixed number of the previous ciphertext symbols say h_j, the cipher will resynchronise after h_j symbols if there is a transmission error. In case of this, the next h_j symbols will be erroneous and the error propagation is thus worse than the case of a synchronous stream cipher. However, if some ciphertext symbols are deleted or inserted during transmission, the self-synchronising cipher will recover after h_j correct ciphertext symbols, whereas the synchronous ciphers will never regain synchronisation [47]. 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 keystream generator.

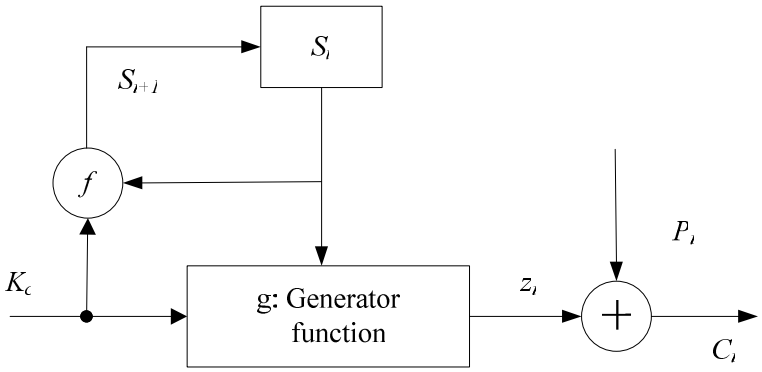

Figure 2.1. A general model for synchronous stream cipher encryption with XOR

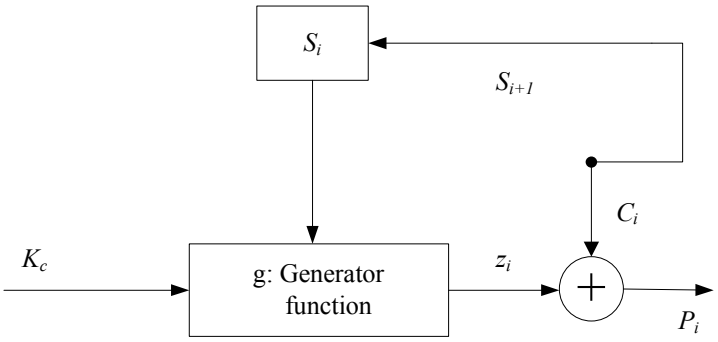

Figure 2.2. A general model for synchronous stream cipher decryption with XOR

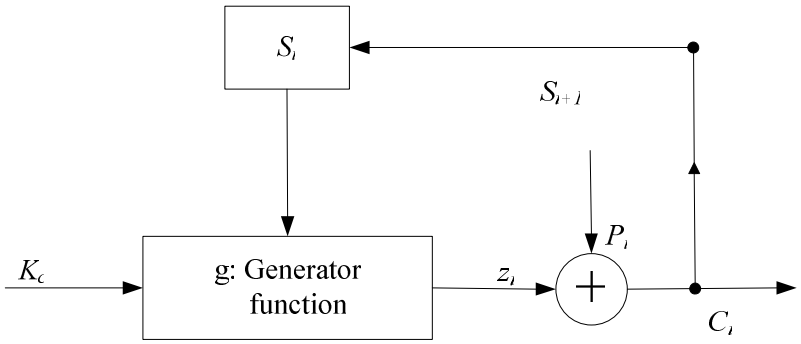

Figure 2.3. A general model for self-synchronising stream cipher encryption with XOR operation

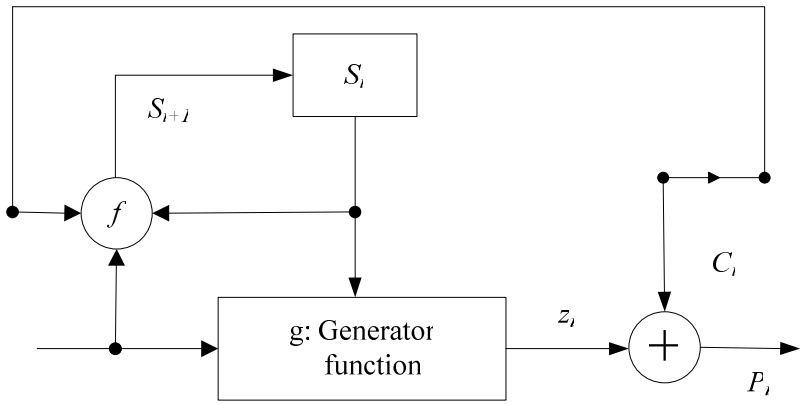

Figure 2.4. A general model for self-synchronising stream cipher decryption with XOR operation

For most of the stream ciphers, keystream sequence is generated independently from plaintext; so in some applications keystream 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. 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 information about clock-controlled stream ciphers that are the main skeleton models for the proposed stream cipher SAFE.

2.1. Linear Feedback Shift Registers

Recalling the keystream generator and its similarity to the OTP, we state that the fundamental property of a keystream generator is to produce as random looking symbols as possible. The distribution of symbols should be uniform and unpredictable. A good start is to use a Linear Feedback Shift Register (LFSR) for achieving a good distribution. The direct output of an LFSR is not a good keystream generator since each symbol produced is

simply a linear combination of the previous symbols, and thus very easy to predict. Nevertheless, LFSRs are widely used components inside stream ciphers.

An FSR (Feedback Shift Register) is a device made up by registers that produce binary sequences or symbols from a field F_q where $q = 2^k$ and k is the symbol size (most of the stream ciphers q is chosen as 2). These registers are the main components of many keystream 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 of 0 or 1, it is called an n-bit shift register. The feedback function maps the state of the shift register according to its 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 [4]. A general structure of a feedback shift register is depicted in Figure 2.5.

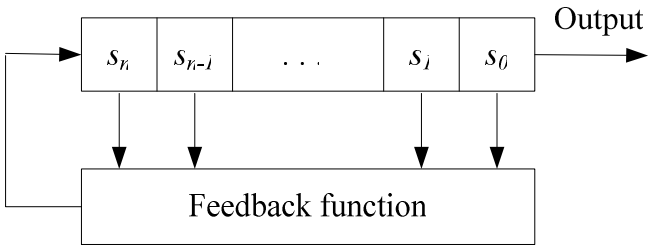

Figure 2.5. General structure of a feedback shift register of length n

The simplest kind of feedback shift register is a linear feedback shift register. In that case, the feedback function can be written as $c_1 s_1 \oplus c_2 s_2 \oplus ... \oplus c_n s_n$, 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 $F_q[x]$ [47].

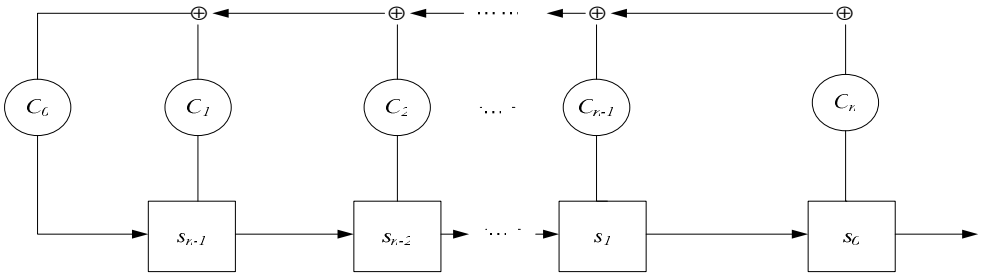

Figure 2.6. General structure of a linear feedback shift register of length *n*

For the binary case, the definition of irreducible polynomial and primitive polynomial can be restricted 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 here after 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.

Table 2.1. State transition of the 3-bit LFSR

State of the LFSR
101
110
111
011
001
100
010

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 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 keystream 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 2*n* bits of the keystream [43]. Thus, if an attacker gets 2*n* bits of keystream 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 keystream. 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 cannot be used as a secure stream cipher, although it has nice properties. To prevent 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 keystream generator, since the output sequences of LFSRs are also easily predictable. Therefore for LFSR based stream ciphers different techniques that 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, keystream 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 keystream 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.

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_C, 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 $f_C(x_1, x_2, ..., x_n)$ can be given as a modulo 2 sum of distinct m^{th} order products of its variables, $0 \leq m \leq n$; which

is called the algebraic normal form of f_C. The nonlinear order of f_C is the maximum of the order of the terms appearing in its algebraic normal form [4]. For instance, $f_C(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_3 \oplus x_2x_3x_4$ 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, an 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 keystream 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 generator to the correlation attack can be defined as the correlation immunity whose details have been shown in [48]. 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, the Geffe generator [2]. The Geffe generator is consisted of three maximal length LFSRs of L_1, L_2 and L_3 respectively as shown in Figure 2.8. The outputs of LFSRs are combined within the function $f_C(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$. The keystream generator uses three LFSRs, combined in a nonlinear manner. If L_1, L_2 and L_3 are pairwise relatively prime, then the period of the generator is $(2^{L_1}-1)(2^{L_2}-1)(2^{L_3}-1)$ and the linear complexity of the keystream sequence becomes $L_1 L_2 + L_2 L_3 + L_3$. For the appropriate values of L_1, L_2 and L_3, large period and high linear complexity is achieved. However when we look at the combining function f_C, if z(t) represents keystream bit at time t, one can realize the probabilistic relation between output of first LFSR and keystream bit as: $P(z(t) = s_1(t)) = P(s_2(t) = 1) + P(s_2(t) = 0)P(s_3(t) = s_1(t)) = \frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{3}{4}$. The output of first LFSR is equal to keystream bit at any time with a probability of 3/4. Thus, one can see that Geffe generator has weaknesses considering the correlation attack.

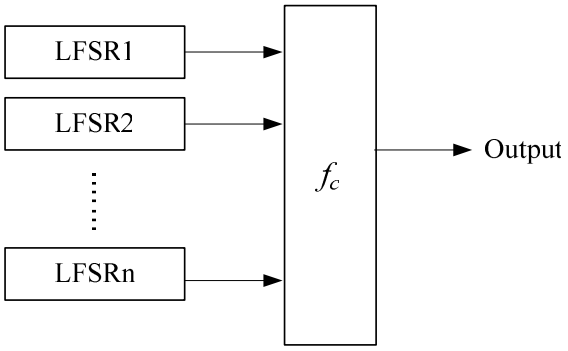

Figure 2.7. A nonlinear combination generator, where n LFSR outputs are combined with a nonlinear Boolean function f_C

Therefore, to have a secure nonlinear combination generator, the combining function f_C must have high algebraic degree, high nonlinearity and a high order of correlation immunity. Also f_C must be a balanced function, which has equal number of ones and zeros in the output column of its truth table, to provide keystream 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_F, appropriate bits of a single LFSR are given. A simple example of nonlinear filter generator is depicted in Figure 2.8, now the function f_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 keystream 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, respectively. 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.

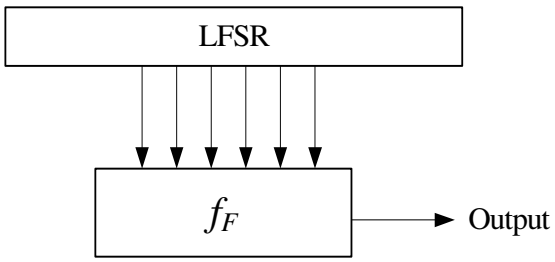

Figure 2.8. A nonlinear filter generator, where a single *n* bit-LFSR's bits are combined with a nonlinear Boolean function f_F to produce keystream sequence

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 nonlinear combining functions are susceptible to the correlation attacks such as fast correlation attacks firstly described in [26]. 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 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 keystream generator. It combines three LFSRs of lengths 19,

22 and 23 bits which are denoted by R_1, R_2 and R_3 respectively [15]. All of these registers have primitive feedback polynomials and each register is updated according to its own feedback polynomial. The taps of R_1 are at bit positions 13, 16, 17, 18; the taps of R_2 are at bit positions 20, 21; and the taps of R_3 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 XORing the most significant bit of each register as shown in Figure 2.9.

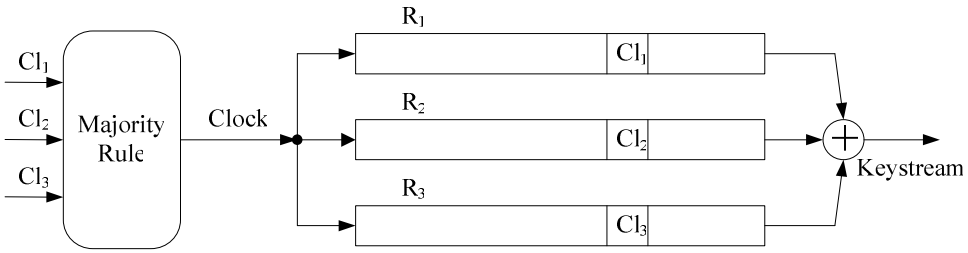

Figure 2.9. The A5/1 stream cipher

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_S and 22 bit frame number F_n XORed (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 keystream sequence.

2.3.2 ORYX

The ORYX cipher is a stream cipher that is used to encrypt wireless digital data as a keystream generator. The output of the generator is a pseudorandom stream of bytes. The generated keystream 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 keystream sequence is XORed with the ciphertext at the receiver side. The ORYX cipher is consisted of three 32-bit LFSRs denoted as R_A, R_B, and R_K, and uses an S-box [49]. The S-box is used for a permutation operation of the numbers between 0 – 255. The block diagram of ORYX is

shown Figure 2.10, where f_K, f_B, f_{A1} and f_{A2} represent the feedback functions of R_K, R_B, and R_A, respectively.

The algorithm works in the following manner: Firstly, R_K is clocked once due to its feedback function. R_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 R_K. Also, R_B is clocked either once or twice depending on another one of the high eight bits of R_K. Then, the last eight bits of R_K is added to the last eight bits of R_A after being permuted with S-box and the last eight bits of R_B after being permuted with S-box, with mod 256 to create 8 bits of keystream.

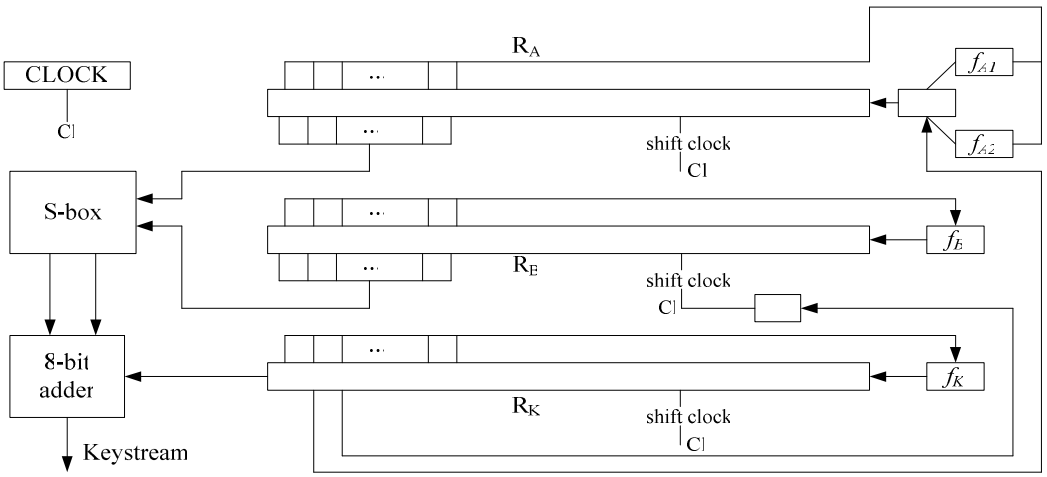

Figure 2.10. The ORYX stream cipher

ORYX was firstly cryptanalyzed by D. Wagner et. al. in [49]. 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.

2.3.3 The Alternating Step Generator

The alternating step generator uses an LFSR R_1 to control the stepping of two LFSRs, R_2 and R_3. The keystream produced is the XOR of the output sequences of R_2 and R_3. The alternating step generator is shown in Figure 2.11.

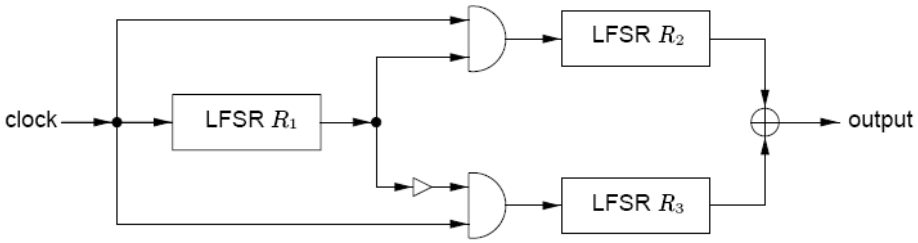

Figure 2.11. The alternating step generator

The clocking mechanism works as follows: Firstly, R_1 is clocked, if its output is 1, then R_2 is clocked and R_3 is not clocked. On the other hand, if the output of R_1 is 0, then R_3 is clocked and R_2 is not clocked. Whether or not a generator register (R_2, R_3) is clocked, it gives its output to keystream generation process; if it is not clocked, it repeats its output. Suppose that R_1 produces a de Bruijn sequence of period 2^{L_1}. Furthermore, suppose that R_2 and R_3 are maximum length LFSRs of lengths L_2 and L_3, respectively, such that $\gcd(L_2; L_3) = 1$. The period of the keystream is then $2^{L_1}(2^{L_2}-1)(2^{L_3}-1)$. Considering the same conditions, the linear complexity becomes $(L_2+L_3)2^{L_1-1} < LC < (L_2+L_3)2^{L_1}$. The LFSRs R_1, R_2, R_3 should be chosen to be maximum-length LFSRs whose lengths L_1, L_2, L_3 are pairwise relatively prime: $\gcd(L_1; L_2) = 1$, $\gcd(L_2; L_3) = 1$, $\gcd(L_1; L_3) = 1$. Moreover, the lengths should be about the same. If $L_1 \approx L$, $L_2 \approx L$, and $L_3 \approx L$, the best-known attack on the alternating step generator is a divide-and-conquer attack on the control register R_1, which takes approximately 2^L steps. Thus, if $L \approx 128$, the generator is secure against all presently known attacks.

3. NONLINEAR FEEDBACK SHIFT REGISTERS (NFSRS)

Combination of several small Linear Feedback Shift Registers (LFSRs) is a well-known method for building stream ciphers. The outputs of the registers are generally combined with a function F, in order to produce one keystream bit (Figure 3.1). A popular example is the algorithm E0, which is used in the Bluetooth technology. Unfortunately, such constructions have some problems that originate from the linearity of the LFSRs. For instance, correlation attacks exploit linear approximations of the function F to attack the whole stream cipher. Another method is algebraic attacks that take advantage of low degree polynomial equations satisfied by F.

Criteria that should be satisfied by the Boolean function F, in order to counter such attacks have been widely studied. However, there appears to be limitations that cannot be overcome like the trade-off between the correlation immunity and high algebraic degree. To improve the designs, it is often suggested to replace linear registers by nonlinear registers [1].

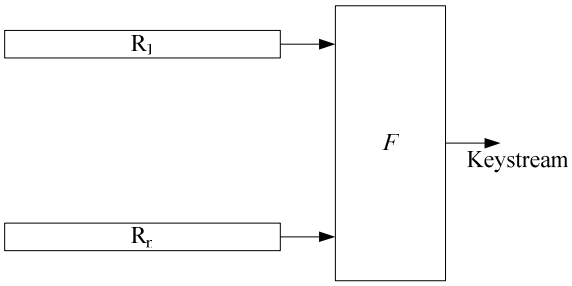

Figure 3.1. Stream cipher built by combination of LFSRs

A (general) *feedback shift register* (FSR) of length L consists of L *stages* (or *delay elements*) numbered $0,1,...,L-1$, each capable of storing one bit and having one input and one output, and a clock which controls the movement of data. During each unit of time the following operations are performed:

- the content of *stage 0* is output and forms part of the *output sequence*

- the content of *stage i* is moved to *stage i − 1* for each *i*, $1 \le i \le L-1$
- the new content of *stage L − 1* is the *feedback bit* $s_j = f(s_{j-1}, s_{j-2}, ..., s_{j-L})$,

where the *feedback function f* is a Boolean function and s_{j-i} is the previous content of stage $L-i$, $1 \le i \le L$. If the initial content of stage i is $s_i \in \{0,1\}$ for each $0 \le i \le L-1$, then $[s_{L-1}, ..., s_1, s_0]$ is called the *initial state* of the FSR. Figure 3.2 depicts an FSR. Note that if the feedback function *f* is a linear function, then the FSR is an LFSR. Otherwise, the FSR is called a *nonlinear* FSR [4].

An FSR is said to be *non-singular* if and only if every output sequence of the FSR (i.e., for all possible initial states) is periodic. An FSR with feedback function $f(s_{j-1}, s_{j-2}, ..., s_{j-L})$ is non-singular if and only if *f* is of the form

$$f = s_{j-L} \oplus F(s_{j-1}, s_{j-2}, ..., s_{j-L+1}) \tag{3.1}$$

for some Boolean function *F*. The period of the output sequence of a non-singular FSR of length *L* is at most 2^L.

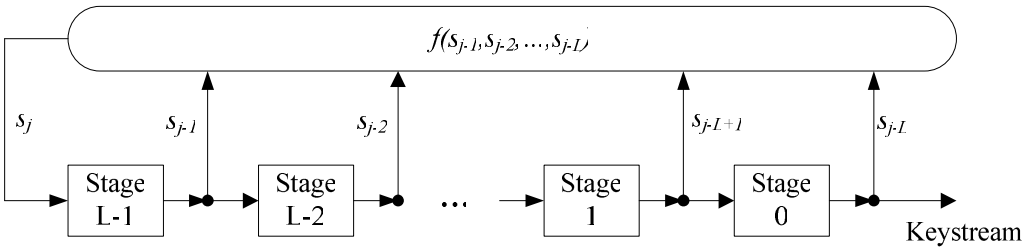

Figure 3.2. A feedback shift register of length *L*

The state diagram of a non-singular FSR may have many small cycles, and the output sequence becomes insecure when the generator falls into one of them. A countermeasure is to design *L*-stage shift registers that generate sequences of the largest possible period 2^L [5]. If the period of the output sequence (for any initial state) of a non-singular FSR of length *L* is 2^L, then the FSR is called a *de Bruijn FSR*, and the output sequence is called a

de Bruijn sequence [4]. The following is a De Bruijn sequence of period 2^4, generated by the four-stage NFSR of Figure 3.3.

1 0 1 1 0 0 1 0 1 0 0 0 0 1 1 1 ...

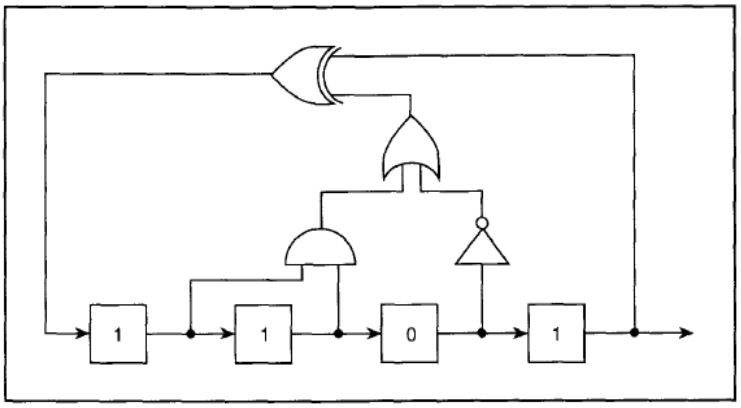

Figure 3.3. A four-stage De Bruijn sequence generator

However, De Bruijn sequences constructible by known algorithms either are technically difficult to implement for fast generation or suffer from severe weaknesses related to their autocorrelational characteristics. For example, with an extensive class of sequences amenable to fast generation, the coincidence probability between $s_i(t)$ and $s_i(t-L)$ for an L-stage FSR is much greater than 1/2. The opponent cryptanalyst can make good use of this feature [5]. In addition, the combination of several such sequences (combined by some Boolean combining function) does not lead to a sequence of larger period. This is obvious, considering that all sequences have periods that are powers of 2. To avoid such weaknesses of de Bruijn sequences, four types of NFSRs are defined by Gammel and Göttfert in [22] for usage in cipher designs, namely *type A*, *type B*, *type C* and *type D*. Some important properties of those registers are given in Table 3.1.

Table 3.1. Properties of certain NFSRs

NFSRs of type	**A**	**B**	**C**	**D**
Length of the shift register	L	L	L	L
Period of output sequence	2^L-1	2^L-1	2^L-2	2^L-2
Forbidden initialisations	(0,0,…,0)	(1,1,…,1)	(0,0,…,0) (1,1,…,1)	(0,1,0,…) (1,0,1,…)
Linear Complexity	2^L-2	2^L-1	2^L-2	2^L-2
Feedback function contains constant term 1	no	yes	no	yes
Distribution of 0's and 1's in the full period	almost equidistributed	almost equidistributed	equidistributed	equidistributed

Among these NFSRs, the ones with period at least 2^L-1 are important for a variety of reasons: (i) they exist, (ii) their output sequences have good statistical properties; (iii) the output sequences have minimal polynomials with a simple algebraic structure [23]. A non-singular binary L-stage feedback shift register is called primitive if for any nonzero binary initial state vector $\overrightarrow{s_0}=(s_0,s_1,...,s_{L-1})$ the corresponding output sequence $\sigma=(s_n)_{n=0}^{\infty}$ has least period 2^L-1. If $f(x)$ is the feedback function of a primitive L-stage FSR, then we clearly must have $f(0)=0$. That is, a primitive FSR fixes the all-zero state. In other words, the zero sequence is an output sequence of any primitive FSR.

Two periodic sequences that are shifted versions of each other (such sequences are called cyclically equivalent) have the same minimal polynomial and, therefore, also the same least period and linear complexity. We mention three basic facts concerning binary primitive FSRs.

- The number of different L-stage feedback shift registers producing de Bruijn sequences is given by

$$B_L = 2^{2^{L-1}-L} \tag{3.2}$$

This was shown by Flye Sainte-Marie in 1894 but the result went unnoticed for a long time until it was rediscovered by De Bruijn.

Although the number B_L is very large, the portion of primitive FSRs among all binary non-singular *L*-stage FSR s is only $1/2^L$. One must also take into account that most primitive FSRs are not suitable for a low-cost hardware implementation as their feedback functions are too complex.

- Let $\sigma = (s_n)_{n=0}^{\infty}$ be a nonzero output sequence of a binary primitive *L*-stage FSR. Let $1 \le k \le L$ and $b_i = (b_1,...,b_k) \in F_2^k$. Let $Z(b_i)$ be the number of n in $\{0,1,...,2^L-2\}$ such that $(s_n, s_{n+1},..., s_{n+k-1}) = b_i$. Then

$$Z(b_i) = \begin{cases} 2^{L-k}-1 & for \quad b_i = 0 \\ 2^{L-k} & for \quad b_i \neq 0 \end{cases} \tag{3.3}$$

Proof. Since the binary sequence σ has least period 2^L-1 and is generated by an *L-s*tage FSR which fixes the all-zero state, every nonzero binary *L*-tuple occurs precisely once in a full portion of the period of σ. From this, the assertion follows at once.

- The *minimal polynomial* of a binary primitive *L*-stage FSR is the product of distinct irreducible binary polynomials whose degrees divide *L* and are greater than 1.

In other words, the minimal polynomial is the characteristic feedback polynomial of the shortest FSR that can produce the given sequence [24]. In particular, the minimal polynomial of σ contains no repeated factors.

NFSR sequences use the monomial spectrum in a more efficient way. Let the initial state of the shift register be given by $\overrightarrow{s_0} = (s_0, s_1,..., s_{L-1})$. If the shift register is linear, then each output bit s_n, $n \ge 0$, of the shift register is the sum of a certain number of initial state

bits taken from the set $\{s_0, s_1, ..., s_{L-1}\}$ If the shift register is nonlinear then the output bit s_n, $n \geq 0$, is the sum of monomials taken from the set

$$\{s_0, s_1, ..., s_{L-1}, s_0s_1, s_0s_2, ..., s_{L-2}s_{L-1}, ..., s_1s_2...s_{L-1}\},$$

where, for most shift registers, each monomial of the set will occur in the representation of some s_n. The set has cardinality $2^L - 2$ and contains all monomials that can be formed out of the initial state bits $\overrightarrow{s_0} = (s_0, s_1, ..., s_{L-1})$ except the two monomials 1 and $s_0s_1...s_{L-1}$. The monomial 1 does not occur because the feedback function $f(x_0, ..., x_{L-1})$ of the NFSR has the property $f(0, ..., 0) = 0$. The monomial $s_0s_1...s_{L-1}$ of degree L does not occur because the feedback function f is balanced [23].

The fact that all the other $2^L - 2$ monomials will occur in the representation of some s_n is not guaranteed for every nonlinear binary FSR. However, it is a typical property that most NFSRs have. For instance, consider the 4-stage primitive LFSR given by $f(x_0, x_1, x_2, x_3) = x_0 + x_1$. Let the initial state of the shift register be (a, b, c, d). The output bits of the shift register appearing in the first period are

$$\begin{aligned} &a, b, c, d, a+b, b+c, c+d, a+b+d, a+c, b+d, a+b+c, b+c+d, \\ &a+b+c+d, a+c+d, a+d. \end{aligned} \tag{3.4}$$

Now consider the 4-stage primitive NFSR given by $f(x_0, x_1, x_2, x_3) = x_0 + x_1 + x_2 + x_1x_3$. With the same initial state, the output bits of the shift register appearing in the first period are

$$\begin{aligned} &a, b, c, d, a+b+c+bd, b+d+ac+bc+bcd, a+b+acd, b+c+abd+bcd, \\ &c+d+cd+abc+acd+bcd, a+b+c+d+ad+bd+abd+acd, \\ &a+b+c+d+ab+ac+abc+abd+bcd, a+d+bc+cd+abc+acd, \\ &a+c+ad+bd+abd, c+d+ab+ac+bd+abc, a+b+d+ac. \end{aligned} \tag{3.5}$$

The sequence of the first $2^L - 1$ output bits of a binary L-stage primitive feedback shift register, where each output bit is expressed as a multivariate polynomial in the initial state bits, is called the *monomial spectrum* of the shift register. Figure 3.4 displays the monomial spectra of the two primitive feedback shift registers, of which the feedback polynomials are given as examples.

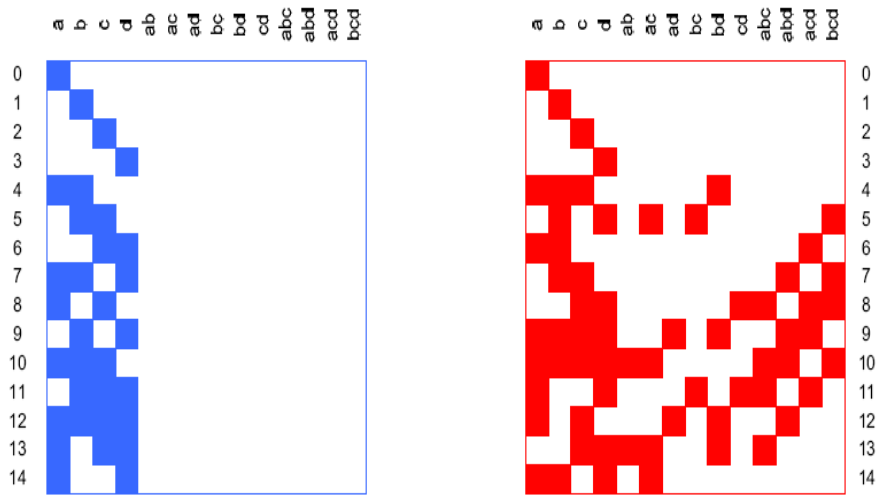

Figure 3.4. Comparison between a primitive LFSR and a primitive NFSR

It is given as an experimental result in [23] that for $2L_j \le k \le 2^{L_j} - L_j$, the *kth* entry in the monomial spectrum of R_j contains close to $2^{L_j - 1}$ different monomials and has in general degree $L_j - 1$.

4. IRREGULAR DECIMATION ALGORITHMS

The usual way to design a stream cipher consists in combining the output of one or several LFSRs in order to obtain a pseudorandom sequence of bits having good properties. Among these properties, the most regarded ones are the period and the linear complexity. These properties are obviously not sufficient, as they do not guarantee that the resulting sequences will resist algebraic or correlation attacks. Usual design techniques to obtain cryptographically suitable pseudorandom sequences include applying (sufficiently) complicated Boolean functions on the outputs or the internal states of several LFSRs. Another interesting technique is to decimate the output of an LFSR in an irregular way. This is the point of two well-known pseudorandom generators: the Shrinking Generator and the Self-Shrinking Generator. Recently a new irregular decimation, the bit-search generator (BSG) is presented by Gouget and Sibert. In addition, to improve the security with the same rate or the rate with the same level of security of the BSG, different variants like Alternative BSG (ABSG), Modified BSG (MBSG) and Editing BSG (EBSG) are proposed.

This section is organised as follows: In section 4.1 the shrinking generator and in section 4.2 the bit-search generator and its variants are described.

4.1 The Shrinking Generator

The shrinking generator (SG) [33] is a well-known keystream generator for stream cipher applications. It consists of two regularly clocked binary linear feedback shift registers. Denote these R_A and R_S, as shown in Figure 4.1, and denote the lengths of these LFSRs as L_A and L_S, respectively. The shrinking generator output is a "shrunken" version or subsequence of the output from R_A, with the subsequence elements selected according to the position of 1's in the output sequence of R_S: the keystream sequence z consists of those bits of the sequence s_A for which the corresponding bit of sequence s_S is 1. The other bits of s_A, for which the corresponding bit of s is 0, are deleted. Under certain

conditions, the output sequences possess a long period, a high linear complexity, and good statistical properties.

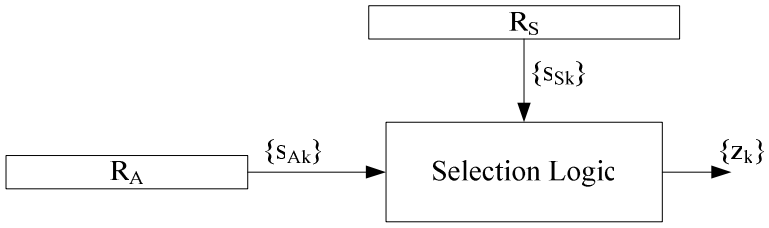

Figure 4.1. The shrinking generator

More precisely, let $\sigma_A = \{s_{A_i}\}_{i=1}^{\infty}$ denote the R_A sequence produced from a nonzero initial state $\{s_{A_i}\}_{i=1}^{L_A}$, and $\sigma_s = \{s_{S_i}\}_{i=1}^{\infty}$ denote the R_S sequence produced from a nonzero initial state $\{s_{S_i}\}_{i=1}^{L_S}$. Let $z = \{z_k\}_{k=1}^{\infty}$ denote the output sequence of the shrinking generator. Then $z_k = s_{Ai_k}$ where i_k is the position of the *k*th 1 in the sequence σ_s. The keystream sequence *z* is an irregularly decimated version of the R_A sequence σ_A, with the decimation controlled by the R_S sequence σ_s.

If the LFSR feedback polynomials are primitive, then *a* and *s* are maximum length sequences with periods $2^{L_A}-1$ and $2^{L_S}-1$, respectively. If in addition L_A and L_S are relatively prime, the period of the keystream is $(2^{L_A}-1)2^{L_S-1}$ and the linear complexity (LC) of the keystream satisfies $L_A 2^{L_S-2} < LC \le L_A 2^{L_S-1}$ [27].

If the LFSR feedback polynomials are fixed, then the secret key of the generator is only the initial states of the two LFSRs. Assuming that all zero initial states are avoided for either LFSR, the total number of secret keys for the generator is $(2^{L_A}-1)(2^{L_S}-1)$. In the worst-case brute force attack, this is the number of trials required to recover the key.

As pointed out in [33], a basic divide-and-conquer attack on the shrinking generator is the linear consistency attack [34] on R_S that requires the exhaustive search through all possible initial states and feedback polynomials of R_S. On the other hand, a probabilistic

correlation attack targeting R_A that requires the exhaustive search through all possible initial states and feedback polynomials of R_A is proposed in [32] and analyzed by computer simulations in [27]. A reduced complexity method based on searching for specific subsequences of the output sequence is suggested in [35], but both the complexity and the required keystream segment length are exponential in the length of R_A.

It is shown in [36] that the output sequence may have a detectable linear statistical weakness if the feedback polynomial of R_A has low-weight polynomial multiples of moderately large degrees. It is suggested in [37] that this weakness may even be used for recovering the R_A feedback polynomial. A theoretical framework for a fast correlation attack targeting the initial state of R_A is also proposed in [37], but the attack is not implemented as it requires a search for specific polynomial multiples of the R_A feedback polynomial.

In [8] Meier and Staffelbach presented a different approach to the shrinking generator. They presented a generator doing the same operation with a single LFSR rather than using two LFSRs. The generator is called the self-shrinking generator (SSG). In self-shrinking generator instead of single output bits, pairs of output bits are considered. If a pair happens to take the value 10 or 11, this pair is taken to produce the pseudo random bit 0 or 1, depending on the second bit of the pair. On the other hand, if a pair happens to be 01 or 00 it will be discarded. The shrinking generator and the self-shrinking generator can be implemented as a special case of the other.

4.2 The Bit-Search Generator

One can consider that both the SG and SSG are methods for bit-search-based decimation. Indeed, both generators use a search for ones along an input bit sequence in order to determine the output bit. Instead of using a search of 1's along a bit sequence in order to determine the output bit as in the case of shrinking and self-shrinking generator, the bit search generator uses the search of some bit *b*, where *b* varies during the process, and the variations depend on the bit sequence. This explains the name of the generator,

which is Bit-Search Generator (BSG). BSG is similar to the Self-Shrinking Generator [8] by using a single LFSR for the decimation process. The BSG operates as follows: The principle of the BSG consists in searching for some bit along the input sequence, and to output 0 if the search ended immediately (that is, if the first bit read during the search was the good one), and 1 otherwise. Consider a window that is located before the first bit on the input sequence. The window moves on to read the first bit of the sequence, and then moves along the sequence until it encounters this bit again. If the window has read only two bits (i.e., the first bit read by the window was followed by the same bit), then the BSG outputs 0, otherwise it outputs 1. The window then reads the next bit following its position, then moves along the input sequence to find it, and so on [7]. For example, for an input sequence 01010011101001000011101 the output is found as follows:

$$\underbrace{01}_{1}\underbrace{0100}_{1}\underbrace{1}_{0}\underbrace{10}_{1}\underbrace{10}_{1}\underbrace{10}_{0}\underbrace{0}_{0}\underbrace{11}_{1}\overset{BSG}{\Rightarrow}11011001$$

The advantage of BSG over its predecessors SG and SSG that it operates at a rate of $\frac{1}{3}$ instead of $\frac{1}{4}$ (i.e. producing *n* bits of the output sequence requires, on average, *3n* bits of the input sequence). The comparison between the BSG and other well-known generators is shown in Table 4.1. However, the disadvantage of the generator is its output can be expressed by using the differential sequence of the input. By using this weakness, basic probabilistic attacks can be efficient against the BSG. To improve the security of the BSG with keeping the same rate, modified versions of BSG are introduced, which are known as MBSG and ABSG.

Table 4.1. Comparison between the BSG and some well-known generators

Generator	Number of LFSRs needed	Rate
Alternating Step	3	1
Shrinking	2	1/2
Self-Shrinking	1	1/4
BSG	1	1/3

The action of the MBSG on the input sequence σ consists in splitting up σ into subsequences of the form $(b,0^i,1)$, with $i \geq 0$ and $b \in \{0, 1\}$. For every pattern of the form $(b,0^i,1)$, the output bit is b. The action of the ABSG on σ consists in splitting up σ into subsequences of the form $(\bar{b},b^i,\bar{b})$, with $i \geq 0$ and $b \in \{0, 1\}$. For every subsequence $(\bar{b},b^i,\bar{b})$, the output bit is $\bar{b}$ for $i = 0$, and b otherwise. Both the MBSG and the ABSG clearly have a rate of $\frac{1}{3}$, like the BSG [42]. For example, for an input sequence 01010011101001000111101, the action of the MBSG and the ABSG is as follows:

$$\underbrace{0}_{0}\underbrace{1}_{0}\underbrace{0}_{0}\underbrace{0}_{1}\underbrace{1}_{0}\underbrace{1}_{0}\underbrace{0}_{0}\underbrace{1}_{0}\underbrace{0}_{0}\underbrace{0}_{0}\underbrace{1}_{1}\underbrace{1}_{0}\underbrace{0}_{1} \overset{MBSG}{\Rightarrow} 000100010$$

$$\underbrace{0}_{1}\underbrace{1}_{0}\underbrace{0}_{1}\underbrace{0}_{1}\underbrace{1}_{1}\underbrace{1}_{0}\underbrace{0}_{1}\underbrace{1}_{0}\underbrace{0}_{1}\underbrace{0}_{0}\underbrace{0}_{1}\underbrace{1}_{1}\underbrace{1}_{0}\underbrace{1}_{0} \overset{ABSG}{\Rightarrow} 10111010$$

As can be seen from the examples above, the action of the ABSG on an input sequence σ is identical to that of the BSG, but their outputs are produced differently. As a result, the main weakness of BSG stemming from the fact that the BSG output can also be expressed by differential of its input sequence is avoided and the ABSG provides higher security in this regard.

In [7], the period of the BSG as an experimental result is given for a maximum length LFSR of length L. The period of the BSG is very close to $\frac{2^L-1}{3}$ or $2x\frac{2^L-1}{3}$ depending on whether the period is read on one or two periods of the input sequence. In addition, for linear complexity no theoretical bounds are given but experimentally, it is given that the linear complexity is usually equal to the period.

In [42] it is given that by making a small change in the BSG algorithm, its rate can be increased from $\frac{1}{3}$ to $\frac{1}{2}$. However, the security of such an algorithm is slightly lower than the BSG algorithm. To improve the rate of the ABSG with keeping its security level, a new algorithm namely the editing bit-search generator is proposed.

4.2.1. The Editing Bit-Search Generator

As mentioned before, the EBSG is a modified version of the bit-search generator proposed by A. Gouget and H. Sibert [7]. It is developed to increase the average rate of the BSG from $\frac{1}{3}$ to $\frac{1}{2}$ with the same level of security. The main difference of the EBSG from the other BSG variants is the use of insertion operation in addition to the decimation used in the BSG and its variants. The algorithm of EBSG is identical to the ABSG except that after the generation of each output bit, a bit is inserted to the input string. The value of inserted bit is the complementary of searched bit *b* for each step. An example for the algorithm is given below:

$$\underbrace{1001}_{0}\overset{0}{\hat{}}\underbrace{0}_{0}\overset{1}{\hat{}}\underbrace{001}_{0}\overset{0}{\hat{}}\underbrace{10}_{1}\overset{1}{\hat{}}\underbrace{01}_{0}\overset{0}{\hat{}}\underbrace{10}_{1}\overset{1}{\hat{}}\underbrace{1}_{1}\overset{EBSG}{\Longrightarrow}0001011$$

For the EBSG any output bit z_k is produced after $i+1$ input bits with probability 2^{-i} for $i \geq 1$. So that the average number of input bits needed to produce one output bit is 3. Since at each output generation one bit is inserted, the net average number of used input bits is 3-1=2. Therefore, the rate of the EBSG is $\frac{1}{2}$.

The EBSG does not provide the same level of security compared to the ABSG. The required keystream length in the case of a basic attack against the EBSG is shorter than the case against the ABSG. So that, a new variant of the EBSG is introduced, the $EBSG_{variant}$. The difference of the variant from the EBSG is the way the inserted bits are computed. For this variant, each output is produced from the pattern $\bar{b}b^k\bar{b}$ for $k \geq 0$ and inserted bit is *t*. If *k* is odd then the value of *t* is changed to its complementary as $t = \bar{t}$, otherwise nothing changes. An example of the $EBSG_{variant}$ is given below. Let σ = 10110001111100110 1001 be the input sequence and *t* which is the value of the inserted bit be set to 0 initially.

$$\underbrace{101}_{0}\overset{1}{\hat{}}\underbrace{1}_{1}\overset{1}{\hat{}}\underbrace{0001}_{0}\overset{0}{\hat{}}\underbrace{11110}_{1}\overset{0}{\hat{}}\underbrace{0}_{0}\overset{0}{\hat{}}\underbrace{110}_{1}\overset{0}{\hat{}}\underbrace{10}_{1}\overset{1}{\hat{}}\underbrace{01}_{0}\overset{EBSG\,\mathrm{variant}}{\Longrightarrow}01010110$$

5. DESCRIPTION OF THE PROPOSED SYNCHRONOUS STREAM CIPHER: SAFE

In this section, the design of a new stream cipher referred to as SAFE is described. The stream cipher SAFE is a keystream generator with a dynamic clock-controlling scheme and an irregular decimation algorithm at the output. It consists of three nonlinear feedback shift registers (NFSRs). In addition, an updated version of 16x16 S-box of Advanced Encryption Standard (AES) is used for permutation of bits of R_1, R_2 and R_3 which are used for dynamic clock-controlling scheme. SAFE uses key length of 128 or 256 bits. At the output of the cipher $EBSG_{variant}$ (Editing Bit-search Generator) is used for decimation of the generator registers sequences. The main idea behind SAFE is provide immunity against correlation and algebraic attacks by using NFSRs and irregular decimation.

In section 5.1, the stream cipher is described. In section 5.2, the detailed description clock-controlling function of the cipher is given. In section 5.3, the initialisation of the cipher and in section 5.4 hardware considerations for the cipher is discussed.

5.1. The Stream Cipher SAFE

The stream cipher uses variable length private key K_C of 128 or 256 bits. SAFE consists of three nonlinear FSRs. NFSR feedback functions do not include the constant term 1 and when initialised with a zero string, the output is a zero string. So, a zero string is a forbidden initialisation vector for NFSRs. According to the table 3.1, the NFSRs are very likely to be *type A*. The reason for choosing nonlinear FSRs (Fig. 3.2) is making the cipher more resistant against correlation and algebraic attacks [1]. One of the NFSRs (R_1) is responsible for determining the clocking tap positions of other two NFSRs. In addition, bits from other two NFSRs (R_2 and R_3) are used for determining the number of clockings of generator registers, R_2 and R_3.

The cipher uses dynamic clocking mechanism that is the cipher does not use the constant inner state values for determining the number of clockings of NFSRs. The inner state values used for this purpose changes in every cycle as the registers are clocked. In addition, the bits from the registers determining the number of clockings are permuted by the S-box in every cycle, so that this mechanism makes it difficult to guess the number of clockings of the registers. Then the output bits of the generator registers are then XORed and used as input to a decimation algorithm.

The proposed stream cipher is shown in Figure 5.1. The lengths of the nonlinear FSRs are as follows:

L_{R1} = 89 bits (Mersenne prime)

L_{R2} = 107 bits (Mersenne prime)

L_{R3} = 127 bits (Mersenne prime)

Feedback polynomials of the NFSRs are non-singular and they can produce a period of length $2^L - 1$, where L is the length of the NFSR. That is, the NFSR produces all of its non-zero states. The feedback polynomials of the NFSRs are as follows:

$$\begin{aligned} f_{R1}(x) = {} & x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x^{23} + x^{1} + x^{47}x^{80} + \\ & x^{47}x^{83} + x^{49}x^{85} + x^{51}x^{81} + x^{55}x^{85} + x^{83}x^{87} + x^{29}x^{49}x^{87} + x^{29}x^{81}x^{87} + \\ & x^{43}x^{83}x^{85} + x^{47}x^{80}x^{87} + x^{47}x^{83}x^{87} + x^{49}x^{81}x^{87} + x^{83}x^{85}x^{87} + \\ & x^{29}x^{43}x^{49}x^{85} + x^{29}x^{43}x^{81}x^{85} + x^{29}x^{49}x^{85}x^{87} + x^{29}x^{81}x^{85}x^{87} + \\ & x^{43}x^{47}x^{80}x^{85} + x^{43}x^{47}x^{83}x^{85} + x^{43}x^{49}x^{81}x^{85} + x^{47}x^{80}x^{85}x^{87} + \\ & x^{47}x^{83}x^{85}x^{87} + x^{49}x^{81}x^{85}x^{87} \end{aligned} \tag{5.1}$$

$$\begin{aligned} f_{R2}(x) = {} & x^{107} + x^{88} + x^{70} + x^{51} + x^{35} + x^{17} + x^{1} + x^{22}x^{43} + \\ & x^{29}x^{35} + x^{29}x^{88} + x^{51}x^{70} + x^{51}x^{97} + x^{51}x^{103} + x^{81}x^{103} + x^{23}x^{31}x^{35} + \\ & x^{23}x^{35}x^{43} + x^{31}x^{35}x^{43} + x^{17}x^{23}x^{31}x^{47} + x^{17}x^{23}x^{31}x^{101} + x^{17}x^{23}x^{43}x^{47} + \\ & x^{17}x^{23}x^{43}x^{101} + x^{17}x^{31}x^{43}x^{47} + x^{17}x^{31}x^{43}x^{101} + x^{23}x^{29}x^{31}x^{35} + x^{23}x^{29}x^{31}x^{88} + \\ & x^{23}x^{29}x^{35}x^{43} + x^{23}x^{29}x^{43}x^{88} + x^{23}x^{31}x^{47}x^{101} + x^{23}x^{43}x^{47}x^{101} + x^{29}x^{31}x^{35}x^{43} + \\ & x^{29}x^{31}x^{43}x^{88} + x^{31}x^{43}x^{47}x^{101} \end{aligned} \tag{5.2}$$

$$f_{R3}(x) = x^{127} + x^{103} + x^{96} + x^{87} + x^{66} + x^{51} + x^{35} + x^{23} + x^{1} + x^{17}x^{103} + x^{23}x^{103} + x^{23}x^{107} + x^{51}x^{87} + x^{51}x^{91} + x^{66}x^{87} + x^{17}x^{23}x^{103} + x^{17}x^{23}x^{107} + x^{17}x^{97}x^{103} + x^{97}x^{101}x^{103} + x^{17}x^{23}x^{87}x^{119} + x^{17}x^{23}x^{97}x^{103} + x^{17}x^{23}x^{97}x^{107} + x^{23}x^{97}x^{101}x^{103} + x^{23}x^{97}x^{101}x^{107} + x^{17}x^{23}x^{87}x^{97}x^{119} + x^{23}x^{87}x^{97}x^{101}x^{119} \quad (5.3)$$

It is clearly seen that the feedback polynomials of NFSRs are in the same form as (3.1) so that the NFSRs are non-singular [4]. Their periods can be $(2^{89}-1)$, $(2^{107}-1)$ and $(2^{127}-1)$, respectively.

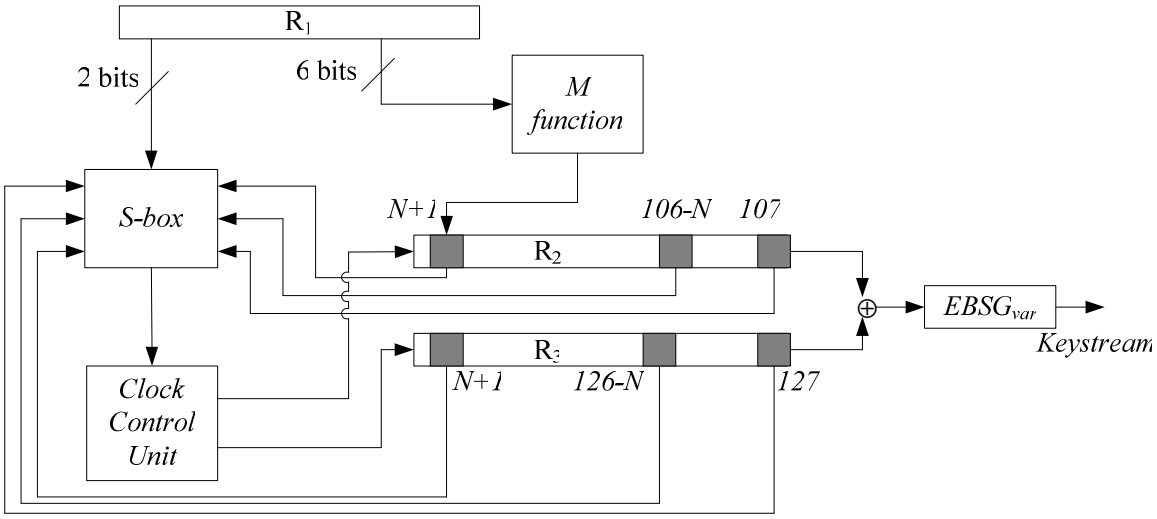

Figure 5.1. The proposed stream cipher

5.2. Clock-Controlling Mechanism and Keystream Generation

As mentioned before, the cipher has a dynamic clocking mechanism, which is the number of clockings of the generator registers R_2 and R_3 are determined by the functions that are changing in every cycle. The cipher uses the updated version of 16x16 S-box of AES in order to permute the incoming bits from the registers. The structure of the S-box of AES is given in Figure 5.2 [6]. Bits from R_1 are used for determining which bits of R_2 and R_3 will be used in S-box. For this purpose, *M* function is introduced. There are six input bits to the *M* function, which are: R_1 (86), R_1 (81), R_1 (77), R_1 (65), R_1 (63) and R_1 (52). Using these bits, a vector $\vec{r} = (r_1, r_2, r_3, r_4, r_5, r_6)$ is formed as follows:

$r_1 = R_1(52), r_2 = R_1(63), r_3 = R_1(65), r_4 = R_1(77), r_5 = R_1(81), r_6 = R_1(86)$. Then the number N is calculated by function M. According to N, the bit positions of R_2 and R_3 which are used as inputs to the S-box are determined. The value N is calculated as follows:

$$N = \sum_{i=1}^{6} r_i 2^{i-1} \qquad 1 \le \mathrm{i} \le 6 \tag{5.4}$$

N takes values between 0 and 63, where r_i denotes the bit values of R_1. After the calculation of the value N, bits from R_2 and R_3 are fed into the S-box. The positions of those bits for R_2 and R_3 are as follows: R_2 (N+1), R_2 (106-N), R_2 (107), R_3 (N+1), R_3 (126-N) and R_3 (127). Since N can be zero, R_2 (N) and R_3 (N) are not used. In a similar manner, R_2 (106-N) is used instead of R_2 (107-N) not to use the same input (R_2 (107)) twice in case of $N=0$.

There are 8 input bits for the S-box. Those bits are from the three registers R_1, R_2 and R_3. From those bits, two vectors are formed for selection of the column and row of the S-box. From the bits of the registers, a vector $\vec{w} = (w_1, w_2, w_3, w_4)$ that selects the row is formed as follows: $w_1 = R_1(32), w_2 = R_2(N+1), w_3 = R_3(N+1), w_4 = R_2(107)$. Similarly, the column vector $\vec{v} = (v_1, v_2, v_3, v_4)$ is formed from the following bits: $v_1 = R_1(44), v_2 = R_2(106-N), v_3 = R_3(126-N), v_4 = R_3(127)$.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 5.2. S-box: substitution values for the byte xy (in hexadecimal format)

According to the values of these bits the row and the column vector is formed and the output of the S-box is observed. The 8-bit output of the S-box determines the number of clockings of the generator registers R_2 and R_3. Let the output of the S-box be a vector $\vec{y} = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$. According to the bit values of y_2 and y_3, the number of clockings of R_2 is calculated. Similarly, according to the values of y_6 and y_7, the number of clockings of R_3 is calculated. The clocking functions of the registers are given below:

$$f_{R2} = y_2 + y_3 + 1 \tag{5.5}$$

$$f_{R3} = y_6 + y_7 + 1 \tag{5.6}$$

As can be seen generator registers R_2 and R_3 are clocked once at minimum and three times at maximum. Since the number *N* takes a different value in every cycle, the clocking functions changing dynamically in every turn.

After the clockings of the generator registers R_2 and R_3, one output bit is produced by registers, respectively. Then these output bits are XORed and fed into the irregular decimation algorithm, $EBSG_{variant}$ for keystream production. Then R_1 is clocked again. In the next turn, again N is calculated from R_1 and number of clockings of R_2 and R_3 is determined. The keystream bit is produced and finally R_1 is clocked once. The operation goes on in this fashion.

5.3. Initialisation

As it is mentioned before, variable key length can be used in SAFE. Nevertheless, we choose 256-bit private key for high level of security. In this case, 256-bit private key is fed into the three registers beginning from R_1. The total length of registers is 323 bits. Therefore, there is a 67-bit length empty space in the third register. An initialisation vector (IV) of 67 bits is loaded into the third register. The first 80-bit block of the key will be divided into twenty 4-bit blocks, A_1 through A_{20}. A_1-A_2, A_5-A_6, A_9-A_{10}, A_{13}-A_{14} and A_{17}-A_{18} represent the row pairs to be swapped and A_3-A_4, A_7-A_8, A_{11}-A_{12}, A_{15}-A_{16} and A_{19}-A_{20} represent the column pairs to be swapped for the S-box. After the registers are filled with the key and S-box is updated, the registers are clocked for 127 times and the produced bits are discarded. Thus the initialisation phase is completed after the registers are clocked for 127 times.

For a fixed key the maximum amount of keystream that can be used for encryption, the so-called frame length, is 2^{60} bits (1024 petabytes). After producing 2^{60} bits, the cipher must be reloaded with key.

5.4. Hardware Considerations

The size of the implementation of an algorithm depends strongly on the minimum feature size of the technology, which is the dimension of the smallest feature actually constructed in the manufacturing process. It also depends on the specific circuit design style, such as CMOS (complementary metal oxide semiconductor) or DCVSL (Differential

Cascode Voltage Switch Logic), and the number of available metal layers for wire routing. Hence, it is necessary to resort to an approximate, technology and circuit style independent measure. A commonly used measure for the size of a design is the *number of NAND gate equivalents* (GE). This is the area of the circuit implementation divided by the area of the smallest NAND gate in the used standard CMOS cell library. Table 5.1 below contains a subset of logical gates taken from a standard cell library for 130 nm CMOS technology. The hardware costs are given units of gate equivalents. One gate equivalent (GE) is the area necessary to implement a 2-input NAND-gate on silicon [23].

Table 5.1. Hardware costs of logical operations

Logical operation	**Binary function**	**Hardware cost**
NAND(a,b)	$ab+1$	1.00 GE
NOR(a,b)	$1+a+b+ab$	1.00 GE
AND(a,b)	ab	1.25 GE
OR (a,b)	$a+b+ab$	1.25 GE
XOR(a,b)	$a+b$	2.25 GE
NAND(a,b,c)	$abc+1$	1.25 GE
NOR (a,b,c)	$1+a+b+c+ab+ac+bc+abc$	1.50 GE
AND(a,b,c)	abc	1.50 GE
OR (a,b,c)	$a+b+c+ab+ac+bc+abc$	1.75 GE
XOR(a,b,c)	$a+b+c$	4.00 GE
MAJ(a,b,c)	$ab+ac+bc$	2.25 GE
MUX(a,b;c)	$a+ac+bc$	2.50 GE

According to the feedback polynomials of the registers, hardware implementation of the register can be given. For instance, consider a register depicted in Figure 5.3 with feedback polynomial:

$$f(x_0, x_1, ..., x_4) = x_0 + x_1 + x_3 + x_1 x_3 \tag{5.7}$$

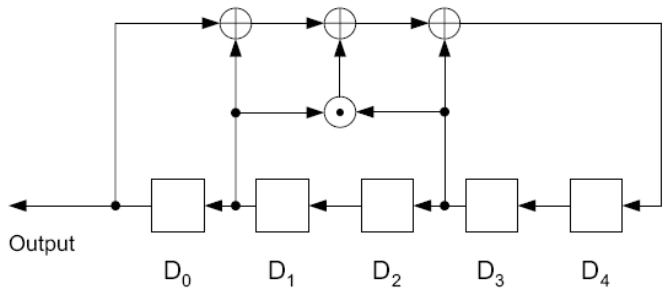

Figure 5.3. A binary nonlinear feedback shift register

According to the Table 5.1, we need three 2-input XOR-gates (3 x 2.25 GE) and one 2-input AND-gate (1.25 GE). The implementation costs of the feedback function are then 8 GE. A better way to implement the feedback function would be to use one 2- input OR-gate (1.25 GE) plus one 2-input XOR-gate (2.25 GE). The implementation costs are then reduced to 3.5 GE. In fact, $OR(a,b) = a \vee b = a + b + ab$ for $a,b \in F_2$, so that (5.7) is equal to

$$f(x_0, x_1, ..., x_4) = x_0 + (x_1 \vee x_3) \tag{5.8}$$

The second implementation is preferable also because it has a lower *logical depth*. The depth of the circuit is the longest path from an input to the output. The logical depth of the first implementation is three while the logical depth of the second implementation is only two. Second implementation is shown in Figure 5.4.

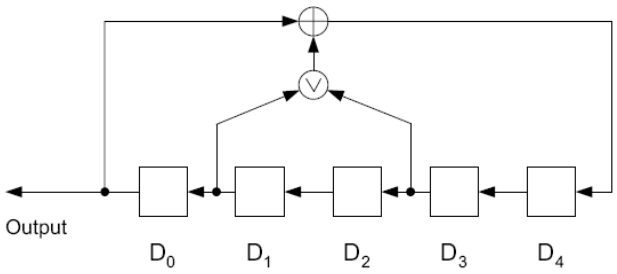

Figure 5.4. A more efficient implementation of the FSR in Figure 5.3

According to the Table 5.1, description of the feedback functions can be given in terms of logical gates whose definition and hardware costs can be found in Table 5.1. The representations show that each feedback function can be implemented with logical depth three using 2-input gates and 3-input gates only. Hardware properties of the registers are given in Table 5.2.

$$\begin{aligned} f_{R1}(x) = & XOR(XOR(x^{89}, XOR(x^{42}, x^{53}, x^{80})), \\ & XOR(XOR(x^{1}, x^{23}, x^{39}), AND(x^{51}, x^{81}), MUX(x^{55}, x^{49}; x^{85})), \\ & MUX(MUX(x^{83}, x^{80}; x^{47}), MAJ(x^{29}, x^{49}, x^{81}); MUX(x^{87}, x^{43}; x^{85}))) \end{aligned} \tag{5.9}$$

$$\begin{aligned} f_{R2}(x) = & XOR(XOR(x^{107}; XOR(x^{1}, x^{17}, x^{88})), \\ & XOR(AND(x^{22}, x^{43}), MUX(x^{70}, x^{97}; x^{51}), MUX(x^{51}, x^{81}; x^{103})), \\ & MUX(MUX(x^{35}, x^{88}; x^{29}), MAJ(x^{17}, x^{47}, x^{101}); MAJ(x^{23}, x^{31}, x^{43}))) \end{aligned} \tag{5.10}$$

$$\begin{aligned} f_{R3}(x) = & XOR(XOR(x^{127}, XOR(x^{1}, x^{23}, x^{35})), \\ & XOR(AND(x^{51}, x^{91}), XOR(x^{66}, x^{87}, x^{96}), MUX(x^{51}, x^{66}; x^{87})), \\ & MUX(MUX(x^{103}, x^{107}; x^{23}), AND(x^{23}, x^{87}, x^{119}); MUX(x^{17}, x^{101}; x^{97}))) \end{aligned} \tag{5.11}$$

Table 5.2. Hardware properties of the NFSRs

Feedback Polynomial of Shift Register	**Number of taps**	**Design size in GE**
f_{R1}	17	31.75
f_{R2}	17	30
f_{R3}	15	31

When implementing a FSR on hardware a considerable amount of area will be used up for the implementation of the memory cells. We can distinguish three types of flip-flops. The simplest and least expensive flip-flop (4.75 GE) is a flip-flop without reset functionality. The more expensive scan flip-flop (6.75 GE) is a flip-flop with an integrated multiplexer. The first two flip-flops in Table 5.3 have one data input and one data output while a scan flip-flop has two data inputs and one data output. The flip-flops used in the

stream cipher SAFE do not need to have reset functionality. There is no need to reset a flip-flop at any time during key loading or resynchronisation.

Table 5.3. Hardware costs of memory units

Memory Unit	**Hardware costs**
Flip-flop	4.75 GE
Reset Flip-flop	5.75 GE
Scan Flip-flop	6.75 GE

6. SECURITY OF SAFE

A good stream cipher must be resistant against different kinds of known plaintext attacks. A *known-plaintext attack* is one where the adversary has a quantity of plaintext and corresponding ciphertext. Also for cryptographic applications, generated keystream of a stream cipher must meet basic security requirements, such as large period, high linear complexity and good statistical distribution.

In section 6.1, the keystream properties of SAFE are examined. In section 6.2 security of the cipher is examined for several scenarios considering the known plaintext attack and the adversary knows the internal structure of the cipher.

6.1. Keystream Properties of SAFE

6.1.1. Period and Linear Complexity

The cipher has non-singular NFSRs. If we assume that the NFSRs are *type A* NFSRs, then the periods of the NFSRs are prime, since the lengths of the NFSRs are Mersenne-prime. The cipher uses mutual clocking so that mathematical modelling of period is not an easy task to perform. However, an upper bound for the period and linear complexity can be given.

In [9], it is shown that the keystream sequences produced by a generator using clock-control register(s) to control the clocking of data generating register (GR), can have maximum period as P_{KG}:

$$P_{KG} = \frac{\lambda P_{GR}}{\gcd(S, P_{GR})} \tag{6.1}$$

where λ is the period of the decimating sequence, P_{GR} is the period of the data generating register and S is the total number of clocking (or the value of the decimating sequence)

applied to the data generating register during one period of index of clock controlling function and "gcd" stands for greatest common divisor.

According to [9], the generator can reach this limit, if one of two conditions are satisfied:

- Degree k_{reg} of f_{GR} is prime and S is not a multiple of $\frac{P_{GR}}{\gcd(P_{GR}, q-1)}$, where f_{GR} is the feedback function of GR over GF(q).
- f_{GR} is a primitive polynomial and $\gcd(S, P_{GR}) \le q^{k_{reg}/2}$

For our case, the feedback function is defined over GF(2), therefore q=2.

If P_1, P_2 and P_3 represent the periods of each register; then periods can take the values $2^{89}-1, 2^{107}-1$ and $2^{127}-1$, respectively. Let S_i denote the sum of clockings of i*th* generator register during the period of R_1, R_2 and R_3, where $i \in \{2,3\}$.

The number of clockings of R_2 and R_3 are not only dependent on R_1 itself, but on themselves as well. There are total of $(2^{89})(2^{127})(2^{107})$ cases at the input of the S-box. For $\frac{(2^{89})(2^{127})(2^{107})}{4}-1$ cases the generator registers are clocked for once. For $\frac{(2^{89})(2^{127})(2^{107})}{2}$ cases the generator registers are clocked for twice and finally for $\frac{(2^{89})(2^{127})(2^{107})}{4}$ cases the generator registers are clocked for three times. Then the value of S becomes $\frac{(2^{89})(2^{127})(2^{107})}{4}-1+2x\frac{(2^{89})(2^{127})(2^{107})}{2}+3x\frac{(2^{89})(2^{127})(2^{107})}{4}=2(2^{89})(2^{127})(2^{107})-1$ .So S is not a multiple of P_{R2} and P_{R3} and the degrees of the feedback polynomials of R_2 and R_3 are prime then we can write the period of the cipher as:

$$PS = \frac{P_{R1}P_{R2}P_{R3}}{\gcd(S, P_{R2})\gcd(S, P_{R3})} x \frac{1}{2} \tag{6.2}$$

Since $\gcd(S, P_{R2})$ and $\gcd(S, P_{R3})$ are equal to 1, because P_{R2} and P_{R3} are prime [9]; the expression for the period of the cipher reduces to:

$$P_s = \frac{P_{R1} P_{R2} P_{R3}}{2} \tag{6.3}$$

The multiple of the periods of registers are divided by $\frac{1}{2}$ because the $\text{EBSG}_{\text{variant}}$ at the output of the cipher reduces the period of the sequence by $\frac{1}{2}$. Consequently, the period the cipher can be $P_s = (2^{89}-1)(2^{127}-1)(2^{107}-1)$ which is about 2^{322}.

For an LFSR of length *n* bits, the linear complexity of its output sequence is *LC* = *n*, provided its feedback polynomial is properly chosen. For a nonlinear register, it is not always easy to compute the linear complexity of its output sequence, but clearly it cannot exceed its period [1]. The upper bound for the linear complexity of an *L*-stage binary primitive FSR is $2^L - 2$. The upper bound $2^L - 2$ also seems to be a typical value for the linear complexity of *L*-stage binary primitive NFSRs [10]. So that individual linear complexities of NFSRs are upper bounded by $2^{127} - 2$, $2^{107} - 2$ and $2^{89} - 2$, respectively. Also according to [11], since $\gcd(S, P_{Ri}) = 1$ for $i \in \{2,3\}$, the period of the clock-control register become a multiplier in the upper bound on the linear complexity of the irregularly decimated sequence. For SAFE, the clock-control registers are R_1, R_2 and R_3. Thus if *LC* denotes the linear complexity of the keystream is very likely to be lower bounded by $(2^{89}-1)(2^{127}-1)(2^{107}-1)\varphi$, where φ denotes the effect S-box and the $\text{EBSG}_{\text{variant}}$.

So in order to apply the Berlekamp-Massey attack, an attacker should intercept at least $(2^{R_1}-1)(2^{R_2}-1)(2^{R_3}-1).2$ plaintext bits. However the cipher will be reinitialised with a different key after a keystream length 2^{60}, therefore this attack seems impractical.

6.1.2. Output Rate

For the production of a single keystream bit, each of the generator registers are clocked once with a probability of $\frac{1}{4}$, twice with a probability of $\frac{1}{2}$ and three times with a probability of $\frac{1}{4}$. Let $a_i(t)$ represent the number of clockings of i*th* register at time *t*, $a_i(t) \in \{1,2,3\}$ .Then the average number of clockings per keystream bit is

$$E\{a_i(t)\} = \frac{1}{4} + 2x\frac{1}{2} + 3x\frac{1}{4} = 2 \tag{6.4}$$

Output rate is then 0.5 for each produced keystream bit. Since $EBSG_{variant}$ is used for the nonuniform decimation of the output of generator registers and it has a rate of 0.5; then the rate of the cipher will be 0.25.

6.1.3 Statistical Properties of the Keystream Sequence

Keystream sequence of the SAFE stream cipher is investigated by using the statistical tests of FIPS 140-2 [44] and NIST Statistical Test Suite. FIPS140-2 is applied to 1000 sequences of 20000 bits that is produced by SAFE. The cipher passes FIPS140-2 in proportion of 99.9%. It is known that the security criteria of FIPS140-2 are stricter than that of FIPS140-1. Therefore, SAFE passes FIPS140-1 in proportion of 100%. We have tested 1000 sequences of length 1 million bits with NIST test. The generated sequences successfully pass this test with a significance level of 0.01 and therefore the success rate for NIST test is 99%.

In addition, autocorrelation test to sequences of length 20000 bits and 50000 bits are applied. The cipher passes autocorrelation test in proportion of 100%. Any significant correlation between the tested sequences and the shifted versions is not recognised.

Furthermore, the spectral test is applied to the cipher. 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 %.

Also the linear complexity profile of the cipher for sequences of 20000 bits is tested. The linear complexity profile is defined to be the measure of change of the linear complexity of a sequence as it becomes longer. The linear complexity profile of a random sequence should approximately follow the line $L = n/2$ where n is the length of the sequence.

Figure 6.1 shows the autocorrelation result for 20000 bits of output of SAFE. The autocorrelation (*AC*) values are normalized to the value at the origin. That is; the maximum AC that can be achieved is 1 which is represented by dashed red line in Figure 6.1. It is noticed that there is no significant peak compared to the normalized value (i.e one) at the origin.

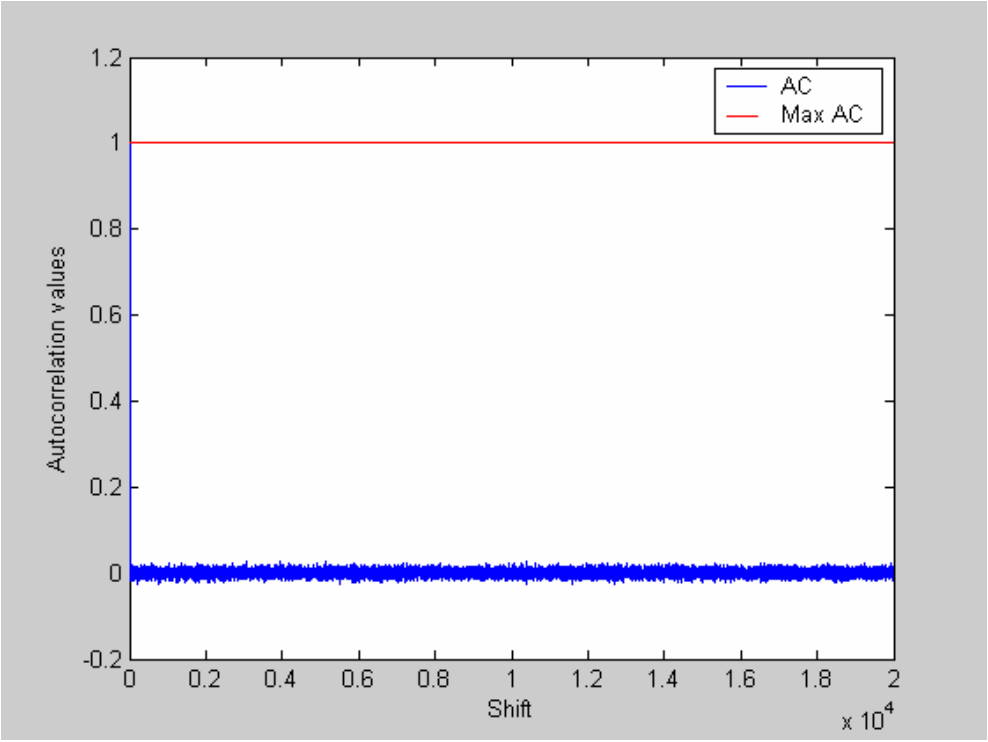

Figure 6.1. Autocorrelation test result for 20000 bits

Figure 6.2 shows autocorrelation values for a sequence of length 50000 bits. Again it is clearly seen that there is no significant peak therefore no significant correlation for the shifted values of the sequence compared to the original sequence.

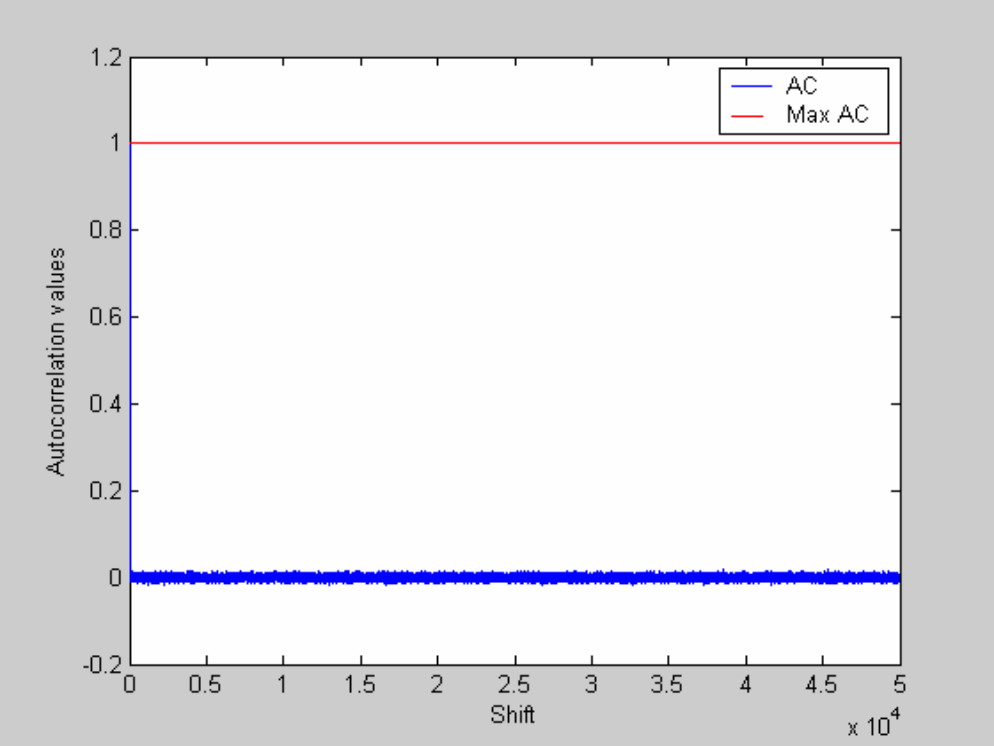

Figure 6.2. Autocorrelation test result for 50000 bits

Table 6.1 shows the spectral test results for SAFE. The *p* value gives the percentage of the peaks. The tests are applied to sequences of length 5000, 20000, 50000 and 100000 bits, respectively. In addition, corresponding spectral test result figures are given in Figures 6.3, 6.4, 6.5 and 6.6. The spectral values are calculated using Discrete Fourier Transform (DFT). Since DFT is symmetric, first *n/2* values are considered for a sequence of length *n*. For each sequence a threshold value is needed to be considered. The threshold value is $\sqrt{3n}$ for a sequence of length *n*. The sequence passes the spectral test since no more than 5% of the peaks surpass the threshold value.

Table 6.1. Spectral test results for SAFE

Length of sequence	***p* value**	**Test Result**
5000	0.3588	passed
20000	0.9269	passed
50000	0.1819	passed
100000	0.9673	passed

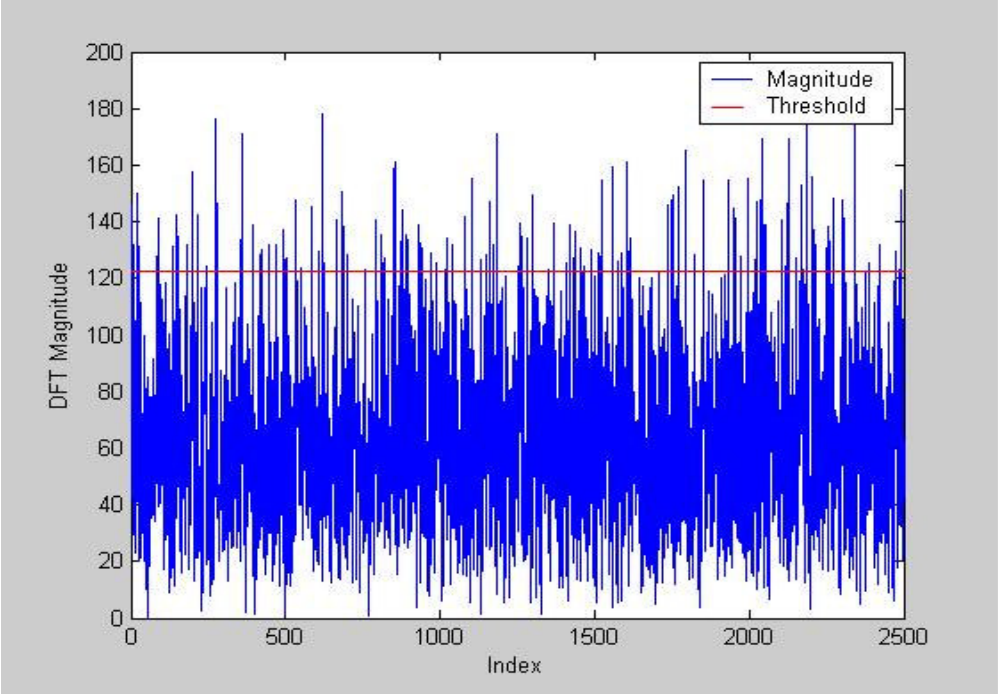

Figure 6.3. Spectral test result for 5000 bits, with threshold value of 122.47

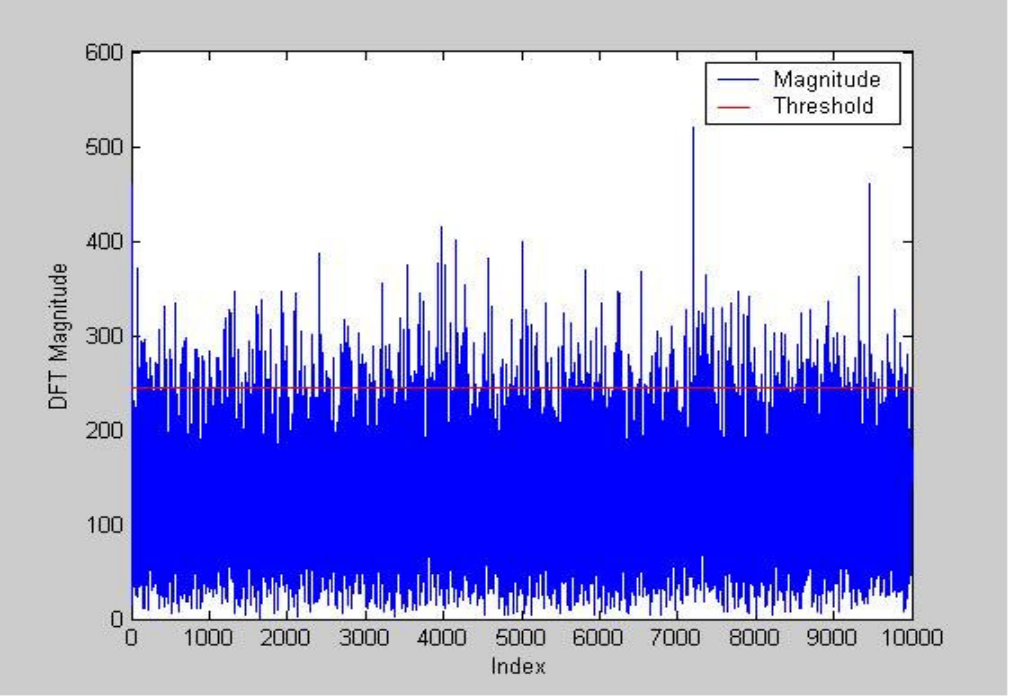

Figure 6.4. Spectral test result for 20000 bits, with threshold value of 244.94

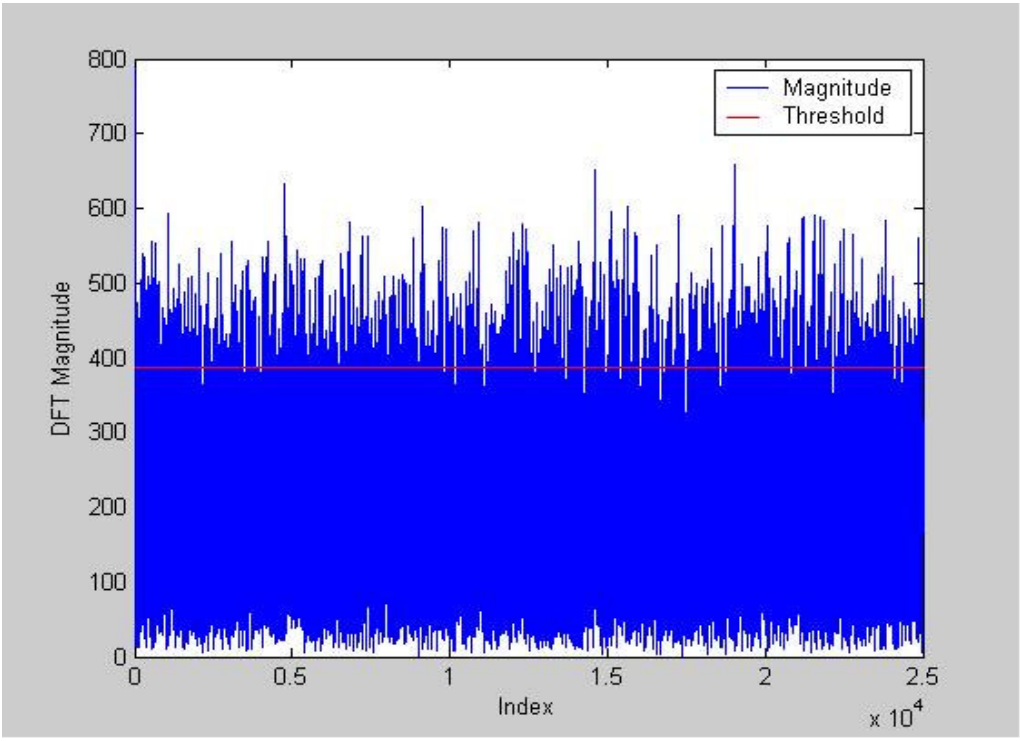

Figure 6.5. Spectral test result for 50000 bits, with threshold value of 387.29

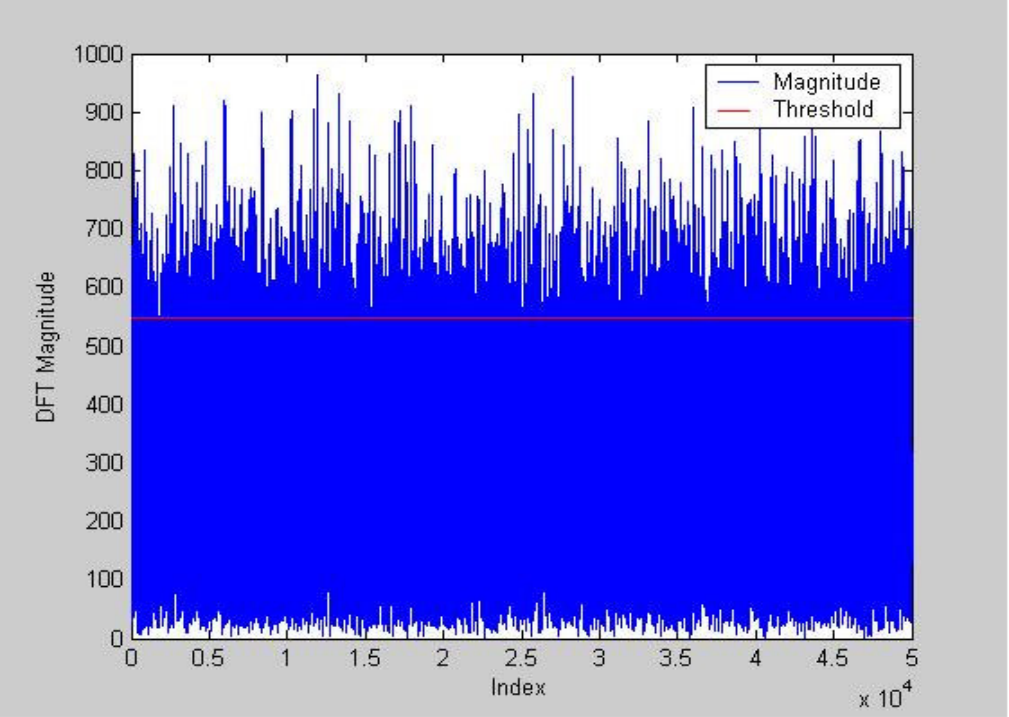

Figure 6.6. Spectral test result for 100000 bits, with threshold value of 547.72

The linear complexity is an important parameter for pseudorandom bit generators since it determines the linear equivalence of a system with a known successive keystream of length twice the linear complexity. The Berlekamp-Massey algorithm described in [43] is an efficient algorithm to determine the linear complexity profile thus the linear complexity by having only some of consecutive keystream bits. In [4], it is said that for a pseudorandom bit generator the linear complexity profile should be close to the line $L = N/2$ where N is the length of the sequence. In Figure 6.7, the linear complexity profile of SAFE for 20000 bits is shown. As can be seen from the Figure 6.7, the linear complexity profile is nearly the same as the $N/2$ line.

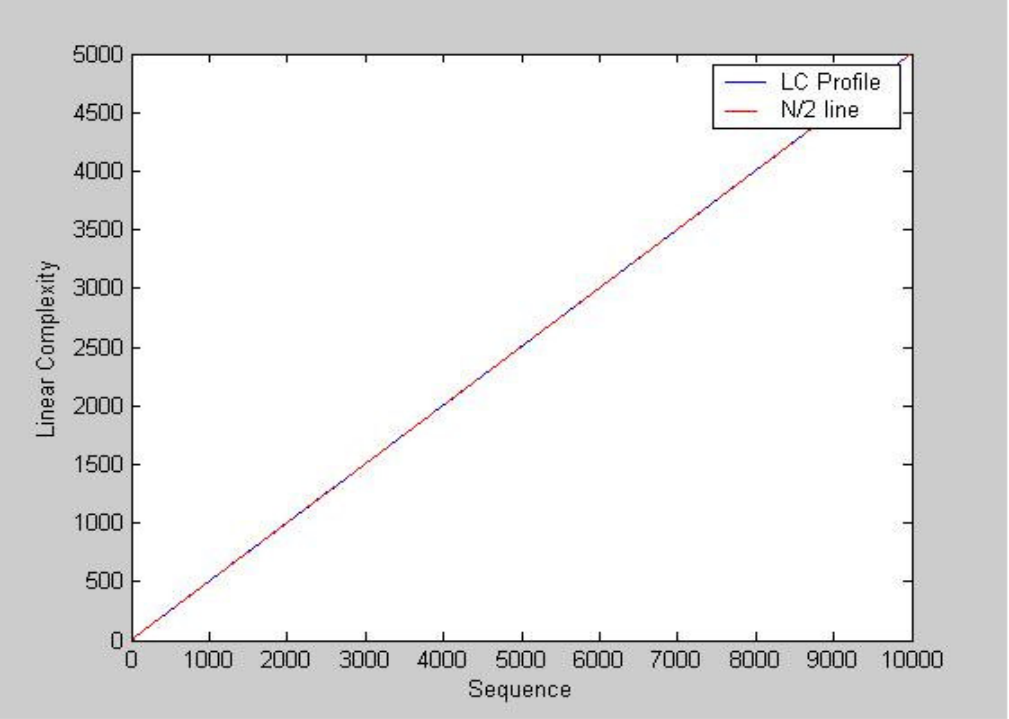

Figure 6.7. The linear complexity profile of SAFE for 20000 bits

6.2. Security of SAFE

While designing a cipher, one of the important points is the resistance against different types of attacks. In this section, the resistance of the cipher against different types of attacks will be analysed. These attacks are known-plaintext attacks conducted under the assumption that the cryptanalyst knows the complete internal structure of the generator.

6.2.1 Exhaustive Key Search

Exhaustive key search, or brute-force search, is the basic technique of trying every possible key in turn until the correct key is identified. To identify the correct key it may be necessary to possess a plaintext and its corresponding ciphertext, or if the plaintext has some recognisable characteristic, ciphertext alone might suffice. Exhaustive key search can be mounted on any cipher [12]. SAFE has a key length of 256 bits, so there are $2^{256} \approx 10^{77,1}$ possible keys. Therefore, this kind of attack seems impractical.

6.2.2 Time/Memory Trade-off Attacks

Generally in time/memory trade-off (TMTO) 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. Suppose that $f:\{0,1\}^n \rightarrow \{0,1\}^n$ is a one-way function, i.e. a function which can be efficiently evaluated in forward direction, but which is hard to invert. The goal of the attacker is to invert this function, i.e. given *f(x)* to find *x*, while keeping complexity of the inversion algorithm as low as possible [16]. The attack consists of two stages: pre-processing stage (offline stage) and processing stage (online stage). In the *offline* stage, a table related to the system in consideration is constructed. P_e will denote the time needed for this pre-computation, and M_e will denote the size of memory needed in constructing and storing this table. During the *online* phase, an explicit target is given and the attacker is asked to return an element related to this target from some search space of size N_e. The time taken for this reply will be denoted by T. Complexity of the attack is usually taken to be the sum or maximum of T and M_e. Hence, for a meaningful TMTO, both T and M_e should be at least small than N_e. It is customary not to take the pre-computation time P_e as adding to the attack complexity. In other words, the attacker is given unlimited amount of time in preparation [17].

In 1980, Hellman introduced the technique of time/memory trade-off (TMTO) attack on block ciphers. That is, if there are N_e possible solutions to search over, the time/memory trade-off allows the solution to be found in T operations (time) with M_e words of memory, provided the time/memory product TM_e equals N_e [13]. Using precomputation time of N_e, Hellman showed that the online time T and memory M_e satisfy the relation $TM_e^{\,2} = N_e^{\,2}$, where $N_e = 2^n$, for a key of length n. Consequently, the attack is called the time/memory trade-off (TMTO) algorithm and the last equation is called the TMTO curve. In the context of block ciphers with reasonably long keys, the original Hellman attack is typically not considered to be of a threat since its precomputation time is the same as the exhaustive search of the key. Furthermore, the attack works for a single chosen plaintext encryption and cannot benefit if more plaintext- ciphertext pairs are available to the attacker since the

precomputed tables are related to a fixed plaintext [16]. Stream ciphers received the time memory trade-off attack for the first by Babbage [14] and Golic [15] through independent works. In the context of stream ciphers, the function from state to a substring of the keystream can be considered to be a one-way function. The difference from the block cipher scenario is that in this case, the obtained keystream provides multiple data points, inverting any of which yields a state of the stream cipher and constitutes an attack. Babbage and Golic investigated this situation and obtained an attack with the relations $TM_e = N_e$ and $T = D_e$, where D_e is the number of available data points. Biryukov and Shamir [19] incorporated multiple data into the Hellman attack and obtained the TMTO curve $TM_e^2 D_e^2 = N_e^2$ and $1 \leq D_e^2 \leq T$. This kind of attacks seems impractical for SAFE because, SAFE has a solution space of 2^{323}. In case of the attack in [19], if we choose the memory value to be 2^{50}(1024 terabytes), according to the TMTO curve $TM_e^2 D_e^2 = N_e^2$, $T D_e^2 = 2^{546} \approx 10^{164,36}$. This result indicates that although the attacker may have large amount of data, a TMTO attack seems impractical for SAFE.

6.2.3 Correlation Attacks

One of the most important attacks against stream ciphers are correlation attacks. Let us consider a combination generator where the output sequences of several linear feedback shift registers are combined by some function *f*. The function *f* should produce a sequence of adequate period and is desired to be nonlinear. For suitable chosen LFSRs many *f* produce keystreams that have a long period and a large linear complexity. However, having a keystream of long period and large linear complexity is not enough. It is also required that the combining function *f* provides confusion that is the property whereby the relation between the simple statistics of the keystream bits and the simple description of the key. Blaser and Heinzmann [20] were the first to point out a possible problem with the relationship between the keystream and the sequences used to produce it. Siegenthaler [3] was the first to propose a model that could exploit this relationship to the detriment of the combination generator.

6.2.3.1 The model of Siegenthaler

The practical setup consists of k LFSRs with LFSR$_i$ having length L_i, $i=1,....,s$. The characteristic polynomial of each LFSR is primitive, and is assumed to be known. The combining function F is a known, nonlinear, arbitrary Boolean function. The secret key of the keystream generator specifies the initial states of each LFSR$_i$. The total number of key bits required to specify the initial states of the keystream generator is $\sum_{i=1}^{k} L_i$.

Siegenthaler modelled the input sequences $(s_t^j), j = 1,...,k$, of the function F as outcomes of independent and uniformly distributed binary random variables S_t^i with probability distribution such that $P(S_t^i = 0) = P(S_t^i = 1)$ for all i and t. The output of F is an independent and uniformly distributed random variable $Z_t = F(S_t^1,...,S_t^k)$ with probability distribution P_Z where $P(Z_t = 0) = P(Z_t = 1)$. The probability that the keystream bit z_t coincides with the input bit s_t^j is given by $P_j = P(Z_t = S_t^j)$. The keystream is said to leak information about LFSR$_j$ if $P_j \neq 0.5$.

Siegenthaler showed that if correlation exists, it is possible to determine the initial state of each LFSR independently, thereby reducing the cryptanalytic attack to a divide-and-conquer attack, with approximate complexity $\sum_{i=1}^{k} 2^{L_i}$. Siegenthaler's attack amounts to an exhaustive search through the state space of each individual LFSR. For each state by computing the cross-correlation function

$$C_{s^j,z} = \frac{1}{n}\sum_{i=0}^{n-1}(-1)^{z_i}(-1)^{s_i^j} \tag{6.5}$$

between the known keystream bits $z_0, z_1,....., z_{n-1}$ of the sequence (z_t) and the suspected output (s_t^j) of LFSR$_j$, the correct initial key can be found. A different model for the correlation attack is depicted in Figure 6.8.

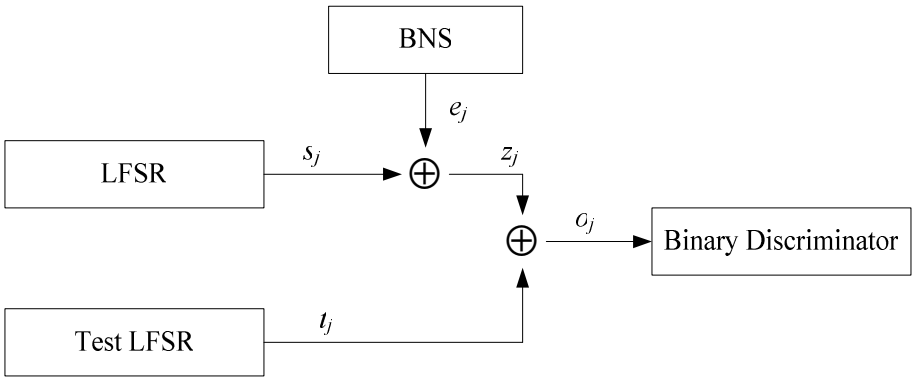

Figure 6.8. The model for the correlation attack

According to the model above, the corruption of the internal LFSR sequence under discussion due to other LFSRs in the stream cipher may be modelled as "errors" in the sequence. So the model can be thought as an LFSR and a binary noise source (BNS) which introduces the errors to the LFSR sequence. In this attack, a test LFSR is stepped through all of its $2^L - 1$ non-zero initial states and the output is XORed with the output of the stream cipher model, as model shown in Figure 6.1. The amount of correlation between the LFSR sequence and the ciphertext can be adjusted, by changing the probability $p=P(e_j=1)$ of the BNS emitting a 1. A high level of correlation implies that only very few 1's are injected into the LFSR output sequence by the BNS. In general, the output sequence (o_j) of the model will appear to be "random", since it is the XOR of two sequences, the number of 0s and 1s in the sequence being roughly equal. However, when the test LFSR is initialised with the *correct* initial state (identical to the initial state of the LFSR under attack), the output sequence (o_j) will be *unbalanced*, consisting mainly of long runs of 0's, interspersed with a few 1's [25]. However, such a search is not very realistic when the degree of the feedback polynomial of the LFSR exceeds 60 [21].

Since the exhaustive search in Siegenthaler's model is not practical, it was shown by Meier and Staffelbach [26] that in certain cases one can avoid this exhaustive search.

6.2.4. Fast Correlation Attacks

In [26] Meier and Staffelbach proposed that under certain circumstances like few feedback taps and long LFSR lengths, faster correlation attacks can be successful without making an exhaustive search. They proposed two attacks (Algorithm A and Algorithm B) which are much faster than the above attack and work for LFSR length $L >> 60$ if the LFSR in question have only a few feedback taps (which is sometimes preferred in practice for ease of hardware.) Under suitable conditions, correlation attacks against LFSRs of length $L = 1000$ or even greater are feasible.

There are several papers based on the ideas of Meier and Staffelbach, such as: [21], [28], [29] and [30]. Most of the algorithms work under the condition that the LFSR has a low weight feedback polynomial. However in [31], Johansson and Jonsson developed the approach to feedback polynomials based on the theory of convolutional codes. Their method can be applied to arbitrary LFSR feedback polynomials, in opposite to the previous methods, which mainly focused on feedback polynomials of low weight.

All of the algorithms use the approach of viewing the problem as a decoding problem. That is, the keystream is regarded as the output of a binary memoryless symmetric channel (BSC) where the LFSR sequence is regarded as the input to the channel. The correlation probability $1 - p$, defined by $1 - p = P(s_i = z_i)$, gives p as the crossover probability (error probability) in the BSC. W.l.o.g we can assume $p < 0.5$. The model is shown in Figure 6.9.

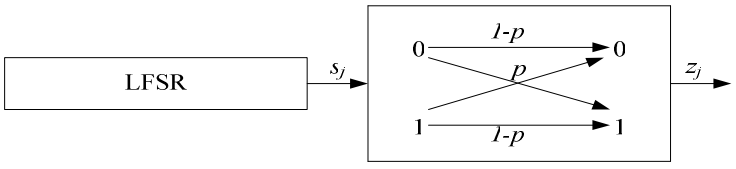

Figure 6.9. Model for a fast correlation attack

All algorithms for fast correlation attacks operate in two phases: In the first phase the algorithms find a set of suitable parity check equations based on the feedback taps from the

LFSR. The second phase uses these parity check equations in a fast decoding algorithm to recover the transmitted codeword and thus the initial state of the LFSR.

In order to give the results of the attack in [26] more clearly, assume that n digits of the output sequence $\boldsymbol{z}$ are given, and correlated with probability $p > 0.5$ to an LFSR sequence s_a, produced by an LFSR with t taps. In addition, it is assumed that the feedback connection is known. This is not an essential restriction as there are only a very limited number of maximum-length feedback connections with few taps. Hence exhaustive search over all primitive feedback connections is possible. The attack applies to an arbitrary number t of taps but the analysis is restricted to even values of t since irreducible feedback connections (of length greater than 1) necessarily have an even number of taps [26].

In this attack the keystream sequence $\boldsymbol{z}$ can be viewed as a perturbation of the LFSR sequence s_a by a binary memoryless noise source (with Prob(0)=p). For the purpose of reconstructing the LFSR sequence s_a from sequence $\boldsymbol{z}$ the following principle is essential to the algorithms (Algorithm A and B): every digit s_{ai} of s_a satisfies linear relations derived from the basic feedback relation, all of them involving t other digits of s_a. By substituting the corresponding digits of $\mathbf{z}$ in these relations, equations are obtained for each digit z_i, which either may or may not hold. To test whether $z_i = s_{ai}$, the number of all equations hold for z_i is counted. Then the greater the number of equations hold, the higher is the probability that z_i will agree with s_{ai}.

In algorithm A, a test is used for correct digits. This is done by selecting those digits that satisfy the most equations. In this way an estimate of the sequence s_a at the corresponding positions can be obtained. Under favourable conditions these digits have a high probability of being correct, which means that only a slight modification of the estimate is necessary. This results in a considerably reduced exhaustive search to sort out sufficiently many correct digits, in order to determine the LFSR sequence s_a by solving linear equations. The computational complexity of the attack is of order $O(2^{cL})$, where $c <$ 1 is a function of number of feedback taps t, the probability of correlation p and n/L (n is the length of the available keystream and L is the length of the register). To give an

example for the value of c, it is smaller than 0.25 for t=2, n/L=10^6 and $p \geq 0.6$. Moreover, for $p > 0.67$, the value of c is below 0.001. This is a considerable improvement when compared to the exhaustive search where c=1. However for large t ($t \geq 10$) the value of t becomes very close to binary entropy function that is Algorithm A gives no advantage over exhaustive search [26].

In Algorithm B, the most reliable digits are not searched. Instead, all digits of $\boldsymbol{z}$ are taken into account together with their probabilities of being correct. *A priori*, with probability p a digit of $\boldsymbol{z}$ agrees with the corresponding digit of s_a. Then for each digit z_i of $\boldsymbol{z}$ a new probability p^* is assigned, which is the probability for $z_i = s_{ai}$, conditioned by the number of equations satisfied. This procedure can be iterated with the varied new probabilities p^* as input to every round. After a few rounds, all those digits of $\boldsymbol{z}$ whose probability p^* is lower than a certain threshold are complemented. Under suitable conditions it is expected that the number of incorrect digits decreases. In this case, the whole process is restarted several times until the original LFSR sequence s_a is obtained.

To obtain conditions under which Algorithm B succeed, a function $F(p,t,n/L)$ is introduced to measure the correction effect. If $F(p,t,n/L) \leq 0$ there is no correction effect and algorithm B will not be able to reproduce the LFSR sequence s_a [26]. Therefore a definite limit to the attack is obtained (which is attained for $t \geq 10$ if $p \leq 0.75$). In other direction, for t = 2 or t = 4 taps Algorithm B still remains effective for small correlations, and, in fact, for t = 2, even for correlation probabilities quite close to 0.5. This means in particular that correlation to LFSRs with only two feedback taps can be very dangerous. The striking efficiency of Algorithm B is that the computational complexity is of order $O(L)$ (i.e. linear in length L of the LFSR) [26].

Algorithms A and B enable attacks against LFSRs of considerable length (e.g., L = 1000 or greater) with software implementation. However a comparison shows that Algorithm A is preferable if $c << 1$ and p is near 0.75 whereas Algorithm B becomes more efficient for probabilities p near 0.5.

To prevent attacks based on these methods, suitable precautions are necessary. This leads to new design criteria for stream ciphers:

1. Any correlation to an LFSR with less than 10 taps should be avoided.
2. There should be no correlation to a general LFSR of length shorter than 100 (especially for the case of known feedback polynomial.)

In the attack, for number of *t* taps, the weight of the feedback polynomial will be *t+1*. So for a fixed bit s_{ai} there are *t+1* number of relations. Also using the squaring operation, $f(x)^j = f(x^j)$ for $j = 2^i$, more relations involving the fixed bit is obtained. The squaring operation continues until the degree of the polynomial is greater than the length of the observed keystream. The obtained equations are called *parity check equations*. The *m* equations are written below:

$$\begin{aligned} s_{ai} + b_1 &= 0, \\ s_{ai} + b_2 &= 0, \\ &\vdots \\ s_{ai} + b_m &= 0, \end{aligned} \tag{6.6}$$

where each b_i the sum of *t* different positions of s_a. Then these equations are applied to the keystream bits

$$\begin{aligned} z_i + y_1 &= L_1 \\ z_i + y_2 &= L_2 \\ &\vdots \\ z_i + y_m &= L_m \end{aligned} \tag{6.7}$$

where y_i is the sum of the positions in the keystream corresponding to the positions in b_i. Assume that *h* of *m* equations hold, i.e.

$$h = |\{i : L_i = 0, 1 \le i \le m\}| \tag{6.8}$$

when we apply them to the keystream. Then it is possible to calculate the probability $p^* = P(a_n = z_n \mid h \text{ equations hold})$ as

$$p^* = \frac{ps^h(1-s)^{m-h}}{ps^h(1-s)^{m-h} + (1-p)(1-s)^h s^{m-h}} \tag{6.9}$$

where $p = P(z_n = a_n)$ and $s = P(b_i = y_i)$. Then one of two algorithms given above (Algorithm A and B) are used for reconstructing the initial state.

Since the correlation between the generators registers and the output is %50, correlation attacks cannot be applied to SAFE. However if one analyses the output and detects a correlation different than %50, correlation attacks can be applied. Let us consider the model of Siegenthaler for the correlation attack against SAFE. The generator has an irregular clocking mechanism, so it is resistant against correlation attacks. The attacker should make an exhaustive search for guessing the clocking mechanism and structure of S-box. However, since there is no one-to-one corresponce between the generator register outputs and the keystream, Siegenthaler's model cannot be applied.

When we apply fast correlation attack to SAFE, we consider two scenarios. In scenario 1, we omit the effect of $EBSG_{variant}$. In Meier and Staffelbach's model, the number of taps of the NFSRs must be less than 10 for the attack to be successful, where in SAFE the number of taps of the registers is more than 10. Therefore, the model of Meier and Staffelbach seems infeasible for SAFE for scenario 1. However, the model of Johansson and Jonsson's model can be applied to arbitrary feedback polynomials. So in scenario 1, without the effect of $EBSG_{variant}$, SAFE is theoretically secure against Meier and Staffelbach's model and insecure against Johansson and Jonsson's model. Let us consider the effect of $EBSG_{variant}$ for scenario 2. All of the fast correlation attacks use binary memoryless symmetric channel for the attack model. However, $EBSG_{variant}$ has a memory; that is two consecutive bits cannot be observed at the output at the same time. For every pattern of the form $\bar{b}b^k\bar{b}$, $k \ge 0$ there is a searched bit b. In order to produce an output, the

selection logic has to remember the bits before the searched bit. Therefore, the selection of two consecutive bits is not possible because of the structure of the algorithm. In scenario 2, considering the memory effect of $EBSG_{variant}$, the model of Johansson and Jonsson cannot be applied. Consequently, fast correlation attacks cannot be applied to SAFE because of the existence of the $EBSG_{variant}$.

6.2.5 Embedding Attacks

Embedding attacks are based on the possibility of embedding one binary string into another. Let $X = \{x_t\}_{t=1}^{\infty}$ denote the output sequence of a regularly clocked binary shift register with not necessarily linear feedback. Let a decimation sequence be defined in terms of its increments, that is, as a non-negative integer sequence $D = \{d_t\}_{t=1}^{\infty}$. In practice, *D* is produced by a clock-control generator, and is therefore ultimately periodic. The output sequence $Y = \{y_t\}_{t=1}^{\infty}$ of the clock-controlled shift register is defined as a decimated sequence

$$y_t = x_{(\sum_{i=1}^{t} d_i)} \tag{6.10}$$

Note that the decimation operation actually means that in order to obtain the next output symbol y_t, after producing y_{t-1}, one has to delete $d_t - 1$ consecutive symbols from *X* if $d_t \geq 1$or has to repeat y_{t-1} if $d_t = 0$. There are two types of embedding attack, unconstrained and constrained. In the unconstrained case, there is arbitrary number of deletions at a time and in constrained case at most *d* deletions at a time. The objective of the attack is to reconstruct the initial state of the clock-controlled shift register based on a given segment of the output sequence, without knowing the decimation sequence.

In [32] Golic and O'Connor described the condition for the unconstrained embedding attack. That is, for a successful attack the deletion rate of the cipher must be smaller than 0.5. If the deletion rate of the cipher greater than or equal to 0.5, then the cipher is secure

against the unconstrained embedding attack. The deletion rate of SAFE is obtained using (6.4),

$$p_d = 1 - \frac{1}{2xE\{a_i(t)\}} = 0.75 \tag{6.11}$$

The average number of clockings, $E\{a_i(t)\}$, is multiplied by 2, because the average rate of the $EBSG_{variant}$ is $\frac{1}{2}$. The deletion rate of SAFE does not satisfy the condition for unconstrained embedding attack, and it is theoretically secure against unconstrained embedding attack.

In constrained case, let the maximum number of deletions as $d_{\max} = 2.\max(a_i(t)) = 6$. The attacker considers information of $d_{\max}$ as opposed the idea behind the unconstrained embedding attack. In [32], 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 superexponential in $d_{\max}$. The attack cannot be successful, if the minimum required keystream 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. The amount of required keystream for a successful reconstruction is given by

$$n \geq L\, 2^{(d+2)(1+2^{d+2})} \ln 2 \tag{6.12}$$

which is linear in the register length *L* but superexponential in *d*. For SAFE stream cipher $d_{\max}$ is 6 which requires prohibitively large amount of known keystream. Therefore, we can say that constrained embedding attack on SAFE stream cipher does not seem practical, although it is theoretically possible.

6.2.6. Probabilistic Attack

Embedding attacks make no use of the probability distribution of the decimation sequence. Therefore, they are not optimal in general. The probabilistic attack based on the joint probability of the original and decimated sequences is optimal [32]. In this attack, an exhaustive search is made for the initial states of the shift registers and a joint probability value of the keystream and each shift register output is calculated according to the equation below

$$P(i,k) = P(i-1,k)\,p + P(i,k-1)(1-p)\delta(x_{i+k}, y_k) \tag{6.13}$$

where $P(i,k)$ denote the partial joint probability for the prefix $X^{i+k} = \{x_t\}_{t=1}^{i+k}$ of X of length $i+k$ and the prefix $Y^k = \{y_t\}_{t=1}^{k}$ of Y of length k, for any $1 \le k \le n$ and $0 \le i \le m\text{-}n$. Let $\delta(x, y)$ denote the substitution probability defined to be equal to 0.5 if x and y are equal and zero otherwise. Also p is the deletion rate of the keystream generator. Initial values for (6.12) are $P(i,0) = p^i$, $0 \le i \le m\text{-}n$ and $P(-1,k) = 0$, $1 \le k \le n$ [32]. The initial states of the shift register with a high joint probability value are candidate initial states. The attack thus requires exhaustive search over all the initial states (phases) of FSR, so its computational complexity is $O(2^L L^2)$, where L is the length of the shift register [27].

When we apply probabilistic attack to SAFE, it requires an exhaustive search through the initial states of generator registers. The amount of search is $2^{234} \approx 10^{70,44}$ state spaces. After this computation, a joint probability value is calculated for each initial state. The initial states with a high joint probability value are named as candidate initial states. The complexity of the attack is $O(2^{234}234^2)$. The probabilistic attack is theoretically possible but impractical for SAFE.

6.2.7. Algebraic Attacks

Algebraic attacks are a new family of cryptographic technique based on defining a cipher system by an initial state-output relation of some degree *d.* If one can find a relation for a system then he can theoretically find the initial state of the cipher by solving

multivariate equations with some or may be a huge amount of known consecutive keystream. In algebraic attacks, the aim is to recover the initial internal state. It is assumed that an adversary has knowledge of the algorithm and some keystream bits.

For some ciphers algebraic attacks outmatched all previously known attacks since it decreases the linear complexity of the cipher. In [38], Nicolas T. Courtois presented the first algebraic technique against the Toyocrypt cipher. In [39], Courtois and Meier presented new algebraic attacks against Toyocrypt and LILI-128. Frederik Armknecht proposed an algebraic attack against the Bluetooth Key Stream Generator (E0) in [40]. He had successfully described the E0 system in terms of internal state-output relation with a degree of $d = 4$. In [41] Nicolas T. Courtois proposed a Fast Algebraic Attack against stream ciphers using the lower degree terms of relation equation however with more keystream bits which also should be consecutive.

Most of the algebraic attacks were made against regularly clocked stream ciphers, non-linear filters and non-linear combiners with or without memory. If we sum up algebraic attacks have the following general steps:

1. Set up a system of equations in the unknowns (initial internal state of the cipher).
2. Insert the observed keystream bits into the identifiers z_t.
3. Recover the unknowns by solving the resulting system of equations using linearization, re-linearization or XL algorithm

In first step, an adversary tries to find an exact internal state-output relation for a system. For systems that use LFSR and Boolean functions as combiners or nonlinear filters it is not a difficult task to achieve if the system is clocked in regular or known way. In fact, it is the most important part if one desires to make an algebraic attack. For simple systems such as combiners with or without memory, nonlinear filters with regular clocking and clock-controlled systems with simple or known clocking, existence of such equations is obvious.

The second step of algebraic attacks is the substitution of observed keystream bits into the equations derived in the first section. Normal algebraic attacks have substitution complexity which is less than the actual attack. However, fast algebraic attacks have substitution complexity higher than the actual attack if the substitution is done in a naive manner.

The third step of algebraic attacks is solving the multivariate equations obtained in step 1. Many techniques have been proposed to solve these systems. The best-known and simple technique is the linearisation. The basis of this technique is to linearise a system of nonlinear algebraic equations by assigning a new unknown variable to each monomial term that appears in the system. The same monomial term appearing in distinct equations is assigned the same new unknown variable. The system of equations then changes from a system of nonlinear equations (with few unknown variables) into a system of linear equations (with a large number of unknown variables). If the number of linear equations exceeds the number of new unknown variables, then the attacker can solve the system to obtain the new unknown variables of the linear system (which will in turn reveal the unknown variables of the non-linear system).

To provide immunity against algebraic attacks, nonlinear FSRs are used in SAFE. As mentioned on section 3, NFSRs provide resistance against algebraic attacks which can be clearly seen in Figure 3.4, since algebraic attacks use the monomials for making up the equations. We give a brief example about the effect of NFSRs. Let us consider an LFSR of length 127 bits. In order to apply an algebraic attack to this LFSR, an attacker needs $n = 127$ keystream bits to create 127 equations. To solve this system, the attacker needs a memory of 127^2 bits. The complexity of the attack is 127^3, using the Gaussian elimination method. Now let us consider the third NFSR of SAFE of length 127. The feedback function of NFSR contains 9 terms of first degree, 5 terms of second degree, 4 terms of third degree, 5 terms of fourth degree and 2 terms of fifth degree. Then the attacker needs $n = 9\ x\ \binom{127}{1} + 5\ x\ \binom{127}{2} + 4\ x\ \binom{127}{3} + 5\ x\ \binom{127}{4} + 2\ x\ \binom{127}{5} = 10^{8.75}$ bits to create a system of n equations. The size of the memory needed is $n^2 = 10^{17.5}$ bits and the complexity of the attack is $n^3 = 10^{26.25}$.

SAFE has a dynamic clock-control mechanism so that some of the bits at outputs of the generator registers are discarded. In addition, an irregular decimation algorithm is used at the output. In order to apply an algebraic attack to SAFE, the clocking and structure of S-box should be determined (guessed). In order to write equations for initial states and outputs of the registers, one has to determine the clocking mechanism of the system for determining the discarded bits. The clocking mechanism depends on all of the registers, so that an exhaustive search for all the registers, that is, a search for $2^{323} \approx 10^{97.23}$ spaces should be done. In addition, to guess the structure of the S-box, the attacker has to make an exhaustive search of 2^{80} bits. The total amount of search is thus $2^{403} \approx 10^{121.31}$. After a search of $10^{121.31}$ spaces, the attacker can write equations for initial states-outputs of the generator registers. Nevertheless, this amount of search is impractical, so algebraic attack is theoretically possible but infeasible for SAFE.

7. CONCLUSION

In this thesis, the theoretical background stream ciphers and an implementation of dynamically clock-controlled stream cipher with nonlinear feedback shift registers were described. The main idea behind SAFE is the usage of nonlinear feedback shift registers and irregular clocking for providing immunity against well known correlation and algebraic attacks. To investigate the randomness of the keystream sequences generated by the proposed ciphers, we have used two test suites which are FIPS 140-2 and NIST Statistical Test Suite. SAFE passed all of the statistical tests.

In addition, the security of the cipher is analysed with respect to currently some well known attacks such as exhaustive key search, time/memory trade-off attacks, correlation attacks and algebraic attacks. It is shown that SAFE is secure enough to resist these attacks.

Consequently, we have observed that SAFE meets the design objectives and the security requirements. SAFE may be treated as secure stream cipher to be used for high speed communications. Cipher can be implemented in hardware and also an effective assembly implementation of SAFE would be appropriate for software applications.

APPENDIX A: MATLAB CODES

This section gives the simulation codes of SAFE. The implementation is evaluated in MATLAB environment. One can also use MATLAB compiler to convert the MATLAB implementations to run in C language. The MATLAB functions used to simulate the stream cipher and tests such as autocorrelation, DFT, Linear complexity for statistical observations are listed below.

1. safe.m : Simulates SAFE stream cipher.

2. clockctrl.m : Simulates the clock-control unit of SAFE.

3. Key.m : Generates 256 bit random private key.

4. IV.m : Generates 67 bit random publicly known initialization vector.

5. dec.m : Simulates the decimation of a binary vector.

6. **Sbox_initialise.m** : Initialization of AES S-box.

7. Initialisation.m : Simulates the initialisation procedure.

8. EBSG_var.m : Simulates the EBSG variant.

9. **autocorrelation.m** : Simulation of the autocorrelation test.

10. spectral.m : Simulation of spectral (DFT) test of a binary sequence.

11. LCP.m : Simulates the linear complexity profile of a binary sequence.

1. safe.m

```
function [ Output ] = safe(K,iv,T)

% K , 256 bit private key
% iv, 67 bit initialization vector
% T, Number of cycles

disp ('*******************************************')
disp ('*                                         *')
disp ('*                S A F E                  *')
disp ('*                                         *')
disp ('*******************************************')

[ R1, R2, R3 ]=Initialisation(K,iv);
L1=length(R1); L2=length(R2); L3=length(R3);
Sbox=Sbox_initialise(K(1:80));
for t=1:127

   % R1 is shifted once

temp=mod(R1(89)+R1(83)+R1(80)+R1(55)+R1(53)+R1(42)+R1(39)+R1(23)+R1(
1)....

+R1(47)*R1(80)+R1(47)*R1(83)+R1(49)*R1(85)+R1(51)*R1(81)+R1(55)*R1(85
)....

+R1(87)*R1(83)+R1(29)*R1(49)*R1(87)+R1(29)*R1(81)*R1(87)+R1(43)*R1(83)
*R1(85)....

+R1(47)*R1(80)*R1(87)+R1(47)*R1(83)*R1(87)+R1(49)*R1(81)*R1(87)+R1(87)
*R1(83)*R1(85)....
```

```
+R1(43)*R1(29)*R1(85)*R1(49)+R1(43)*R1(29)*R1(85)*R1(81)+R1(49)*R1(29)
*R1(85)*R1(87)....

+R1(81)*R1(29)*R1(85)*R1(87)+R1(43)*R1(47)*R1(85)*R1(80)+R1(43)*R1(47)
*R1(85)*R1(83)....

+R1(43)*R1(49)*R1(85)*R1(81)+R1(47)*R1(80)*R1(85)*R1(87)+R1(47)*R1(83)
*R1(85)*R1(87)....
      +R1(49)*R1(81)*R1(85)*R1(87),2); %feedback relation of R1
   for j=1:(L1-1)
     R1(L1+1-j)=R1(L1-j); % shifting to right
   end
   R1(1)=temp; %update the first register using the primitive polynomial
   r=[R1(86) R2(81) R1(77) R1(65) R1(63) R1(52)];
   N=1+dec(r);
   w=[R1(32) R2(N) R3(N) R2(107)];
   v=[R1(44) R2(107-N) R3(127-N) R3(127)];
   y=dec2bin(Sbox(1+dec(w),1+dec(v)),8); %Feed the s-box
   y=mod(double(y),48); %S-box output

   %Gathering the clocking information for irregularly clocked LFSRs R2&R3
   [clock_2 clock_3]=clockctrl(y);

   %R2
   for i=1:clock_2

temp2=mod(R2(107)+R2(88)+R2(70)+R2(51)+R2(35)+R2(17)+R2(1)+R2(22)*R2
(43)....

+R2(29)*R2(35)+R2(29)*R2(88)+R2(51)*R2(70)+R2(51)*R2(97)+R2(51)*R2(10
3)+R2(81)*R2(103)....
```

```
+R2(23)*R2(31)*R2(35)+R2(23)*R2(35)*R2(43)+R2(43)*R2(31)*R2(35)+R2(17)
*R2(31)*R2(23)*R2(47)....

+R2(17)*R2(31)*R2(23)*R2(101)+R2(17)*R2(43)*R2(23)*R2(47)+R2(17)*R2(4
3)*R2(23)*R2(101)....

+R2(17)*R2(31)*R2(43)*R2(47)+R2(17)*R2(31)*R2(43)*R2(101)+R2(29)*R2(3
1)*R2(23)*R2(35)....

+R2(29)*R2(31)*R2(23)*R2(88)+R2(29)*R2(35)*R2(23)*R2(43)+R2(29)*R2(43)
*R2(23)*R2(88)....

+R2(23)*R2(31)*R2(47)*R2(101)+R2(47)*R2(43)*R2(23)*R2(101)+R2(29)*R2(
31)*R2(35)*R2(43)....
        +R2(29)*R2(31)*R2(43)*R2(88)+R2(47)*R2(31)*R2(43)*R2(101),2);
%feedback relation of R2

      for j=1:(L2-1)
      R2(L2+1-j)=R2(L2-j); % shifting to right
      end
   R2(1)=temp2; %replace the R2 content with temp2
   end

   %R3
   for i=1:clock_3

temp3=mod(R3(127)+R3(103)+R3(96)+R3(87)+R3(66)+R3(51)+R3(35)+R3(23)+
R3(1)....

+R3(17)*R3(103)+R3(23)*R3(103)+R3(107)*R3(23)+R3(51)*R3(87)+R3(51)*R3
(91)....
```

```
+R3(87)*R3(66)+R3(17)*R3(23)*R3(103)+R3(17)*R3(23)*R3(107)+R3(17)*R3(
97)*R3(103)....

+R3(97)*R3(101)*R3(103)+R3(17)*R3(23)*R3(87)*R3(119)+R3(17)*R3(23)*R3
(97)*R3(103)....

+R3(17)*R3(23)*R3(97)*R3(107)+R3(97)*R3(23)*R3(101)*R3(103)+R3(97)*R3
(23)*R3(101)*R3(107)....

+R3(17)*R3(23)*R3(87)*R3(97)*R3(119)+R3(101)*R3(23)*R3(87)*R3(97)*R3(
119),2); %feedback relation of R3

      for j=1:(L3-1)
      R3(L3+1-j)=R3(L3-j); % shifting the contents of register to right
      end

      R3(1)=temp3; %replace the R3 content with temp3
   end
end



disp ('*******************************************')
disp ('*                                         *')
disp ('*          INITIALISATION HAS FINISHED    *')
disp ('*                                         *')
disp ('*******************************************')

disp ('*******************************************')
disp ('*                                         *')
disp ('*               ENCRYPTION STARTED        *')
disp ('*                                         *')
disp ('*******************************************')
```

```
 for t=1:T

    % R1 is shifted once

temp=mod(R1(89)+R1(83)+R1(80)+R1(55)+R1(53)+R1(42)+R1(39)+R1(23)+R1(
1)....

+R1(47)*R1(80)+R1(47)*R1(83)+R1(49)*R1(85)+R1(51)*R1(81)+R1(55)*R1(85
)....

+R1(87)*R1(83)+R1(29)*R1(49)*R1(87)+R1(29)*R1(81)*R1(87)+R1(43)*R1(83)
*R1(85)....

+R1(47)*R1(80)*R1(87)+R1(47)*R1(83)*R1(87)+R1(49)*R1(81)*R1(87)+R1(87)
*R1(83)*R1(85)....

+R1(43)*R1(29)*R1(85)*R1(49)+R1(43)*R1(29)*R1(85)*R1(81)+R1(49)*R1(29)
*R1(85)*R1(87)....

+R1(81)*R1(29)*R1(85)*R1(87)+R1(43)*R1(47)*R1(85)*R1(80)+R1(43)*R1(47)
*R1(85)*R1(83)....

+R1(43)*R1(49)*R1(85)*R1(81)+R1(47)*R1(80)*R1(85)*R1(87)+R1(47)*R1(83)
*R1(85)*R1(87)....
        +R1(49)*R1(81)*R1(85)*R1(87),2); %feedback relation of R1
    for j=1:(L1-1)
       R1(L1+1-j)=R1(L1-j); % shifting to right
    end
    R1(1)=temp; %update the first register using the primitive polynomial
    r=[R1(86) R2(81) R1(77) R1(65) R1(63) R1(52)];
    N=1+dec(r);
    w=[R1(32) R2(N) R3(N) R2(107)];
```

```
    v=[R1(44) R2(107-N) R3(127-N) R3(127)];
    y=dec2bin(Sbox(1+dec(w),1+dec(v)),8); %Feed the s-box
    y=mod(double(y),48); %S-box output

    %Gathering the clocking information for irregularly clocked LFSRs R2&R3
    [clock_2 clock_3]=clockctrl(y);

    %R2
    for i=1:clock_2

temp2=mod(R2(107)+R2(88)+R2(70)+R2(51)+R2(35)+R2(17)+R2(1)+R2(22)*R2
(43)....

+R2(29)*R2(35)+R2(29)*R2(88)+R2(51)*R2(70)+R2(51)*R2(97)+R2(51)*R2(10
3)+R2(81)*R2(103)....

+R2(23)*R2(31)*R2(35)+R2(23)*R2(35)*R2(43)+R2(43)*R2(31)*R2(35)+R2(17)
*R2(31)*R2(23)*R2(47)....

+R2(17)*R2(31)*R2(23)*R2(101)+R2(17)*R2(43)*R2(23)*R2(47)+R2(17)*R2(4
3)*R2(23)*R2(101)....

+R2(17)*R2(31)*R2(43)*R2(47)+R2(17)*R2(31)*R2(43)*R2(101)+R2(29)*R2(3
1)*R2(23)*R2(35)....

+R2(29)*R2(31)*R2(23)*R2(88)+R2(29)*R2(35)*R2(23)*R2(43)+R2(29)*R2(43)
*R2(23)*R2(88)....

+R2(23)*R2(31)*R2(47)*R2(101)+R2(47)*R2(43)*R2(23)*R2(101)+R2(29)*R2(
31)*R2(35)*R2(43)....
        +R2(29)*R2(31)*R2(43)*R2(88)+R2(47)*R2(31)*R2(43)*R2(101),2);
%feedback relation of R2
```

```
    if i==clock_2
       out_2=R2(107);    % For the last clock of R2, generate the output
    end

    for j=1:(L2-1)
    R2(L2+1-j)=R2(L2-j); % shifting to right
    end
  R2(1)=temp2; %replace the R2 content with temp2
  end


  %R3
  for i=1:clock_3

temp3=mod(R3(127)+R3(103)+R3(96)+R3(87)+R3(66)+R3(51)+R3(35)+R3(23)+
R3(1)....

+R3(17)*R3(103)+R3(23)*R3(103)+R3(107)*R3(23)+R3(51)*R3(87)+R3(51)*R3
(91)....

+R3(87)*R3(66)+R3(17)*R3(23)*R3(103)+R3(17)*R3(23)*R3(107)+R3(17)*R3(
97)*R3(103)....

+R3(97)*R3(101)*R3(103)+R3(17)*R3(23)*R3(87)*R3(119)+R3(17)*R3(23)*R3
(97)*R3(103)....

+R3(17)*R3(23)*R3(97)*R3(107)+R3(97)*R3(23)*R3(101)*R3(103)+R3(97)*R3
(23)*R3(101)*R3(107)....

+R3(17)*R3(23)*R3(87)*R3(97)*R3(119)+R3(101)*R3(23)*R3(87)*R3(97)*R3(
119),2); %feedback relation of R3

    if i==clock_3
```

```
        out_3=R3(127);    % For the last clock of R3, generate the output
      end

      for j=1:(L3-1)
      R3(L3+1-j)=R3(L3-j); % shifting the contents of register to right
      end

      R3(1)=temp3; %replace the R3 content with temp3
   end

   Output(t)=mod(out_2+out_3,2);

end
```

2. clockctrl.m

```
function [ C1, C2 ] = clockctrl(x)

C1=x(2)+x(3)+1;
C2=x(6)+x(7)+1;
```

3. Key.m

```
function [ K ] = Key()

K=rand(1,256);

for i=1:length(K)
   if K(i)<=0.5
      K(i)=0;
   else
      K(i)=1;
   end
```

```
end
```

4. IV.m

```
function [ K ] = IV()

K=rand(1,67);

for i=1:length(K)
   if K(i)<=0.5
      K(i)=0;
   else
      K(i)=1;
   end
end
```

5. dec.m

```
function [ out ] = dec( x )
% Converts a binary vector into decimal form
L=length(x);
out=0;
for i=1:L
   out=out+x(i)*(2^(L-i));
end
```

6. Sbox_initialise.m

```
function [ Sbox ] = Sbox_initialise( x )


B= [99 124 119 123 242 107 111 197 48 1 103 43 254 215 171 118
   202 130 201 125 250 89 71 240 173 212 162 175 156 164 114 192
```

```
   183 253 147 38 54 63 247 204 52 165 229 241 113 216 49 21
   4 199 35 195 24 150 5 154 7 18 128 226 235 39 178 117
   9 131 44 26 27 110 90 160 82 59 214 179 41 227 47 132
   83 209 0 237 32 252 177 91 106 203 190 57 74 76 88 207
   208 239 170 251 67 77 51 133 69 249 2 127 80 60 159 168
   81 163 64 143 146 157 56 245 188 182 218 33 16 255 243 210
   205 12 19 236 95 151 68 23 196 167 126 61 100 93 25 115
   96 129 79 220 34 42 144 136 70 238 184 20 222 94 11 219
   224 50 58 10 73 6 36 92 194 211 172 98 145 149 228 121
   231 200 55 109 141 213 78 169 108 86 244 234 101 122 174 8
   186 120 37 46 28 166 180 198 232 221 116 31 75 189 139 138
   112 62 181 102 72 3 246 14 97 53 87 185 134 193 29 158
   225 248 152 17 105 217 142 148 155 30 135 233 206 85 40 223
   140 161 137 13 191 230 66 104 65 153 45 15 176 84 187 22];

for i=1:20
  A(i,:)=x((4*(i-1)+1):4*i);
end




% Swap 10 rows
B([1+dec(A(1,:)) 1+dec(A(2,:))],:) = B([1+dec(A(2,:)) 1+dec(A(1,:))],:);
B([1+dec(A(5,:)) 1+dec(A(6,:))],:) = B([1+dec(A(6,:)) 1+dec(A(5,:))],:);
B([1+dec(A(9,:)) 1+dec(A(10,:))],:) = B([1+dec(A(10,:)) 1+dec(A(9,:))],:);
B([1+dec(A(13,:)) 1+dec(A(14,:))],:) = B([1+dec(A(14,:)) 1+dec(A(13,:))],:);
B([1+dec(A(17,:)) 1+dec(A(18,:))],:) = B([1+dec(A(18,:)) 1+dec(A(17,:))],:);



%Swap 10 coloumns
B(:,[1+dec(A(3,:)) 1+dec(A(4,:))]) = B(:,[1+dec(A(4,:)) 1+dec(A(3,:))]);
B(:,[1+dec(A(7,:)) 1+dec(A(8,:))]) = B(:,[1+dec(A(8,:)) 1+dec(A(7,:))]);
```

```
B(:,[1+dec(A(11,:)) 1+dec(A(12,:))]) = B(:,[1+dec(A(12,:)) 1+dec(A(11,:))]);
B(:,[1+dec(A(15,:)) 1+dec(A(16,:))]) = B(:,[1+dec(A(16,:)) 1+dec(A(15,:))]);
B(:,[1+dec(A(19,:)) 1+dec(A(20,:))]) = B(:,[1+dec(A(20,:)) 1+dec(A(19,:))]);




Sbox=B;
```

7. Initialisation.m

```
function [ R1, R2, R3 ] = Initialisation(K,iv)

R1=K(1:89); R2=K(90:196); R3=[K(197:256) iv];
```

8. EBSG_var.m

```
function[output]=EBSG_var(s)
N=length(s);
i=1; j=1;t=0;
while i<N
  e=s(i); y=s(i+1);
  i=i+1;
  k=0;
  while s(i)==mod(e+1,2)
      i=i+1;
      k=k+1;
  end
  i=i+1;
  if mod(k,2)==1;
    t=mod(t+1,2);
  end
  N=N+1;
```

```
  s(i+1:N)=s(i:N-1);
  output(j)=y;
  s(i)=t;
  j=j+1;
end
```

9. autocorrelation.m

```
function [ C ] = autocorrelation( x )
%AUTOCORRELATION test

T=length(x)/2;
for i=0:(T-1)
   C(i+1)=sum((2*x(1:T)-1).*(2*x(1+i:T+i)-1));
end
C=C/T;
plot(0:T-1,C,0:T-1,1,'--r')
xlabel('Shift');
ylabel('Autocorrelation values');
legend('AC','Max AC')
```

10. spectral.m

```
function [ P_value ] = spectral( x )
%SPECTRAL_TEST

n=length(x);
for i=1:n
   if x(i)==0
      x(i)=-1;
   end
end
```

```
S=fft(x); %DFT of sequence x calculated using FFT
M=abs(S(1:n/2));
T=sqrt(3*n); % Threshold value
N0=0.95*n/2;
N1=length(find(M<T));
d=(N1-N0)/sqrt(n*0.95*0.05/2);
P_value=erfc(abs(d)/sqrt(2))

if P_value>=0.01
   display('SPECTRAL TEST PASSED');
else
   display('SPECTRAL TEST FAILED');
end

plot(1:n/2,M,'-',1:n/2,T,'--r')
legend('Magnitude', 'Threshold');
xlabel('Index'); ylabel('DFT Magnitude');
```

11. LCP.m

```
function [ L ] = LCP( s )
% Linear Complexity Profile for a seqeunce
% The function implements the well-known Berlekamp-Massey algorithm


n=length(s);
%Initial values
C=[1]; l=0; m=-1; B=[1]; N=1; l=0;
while N<=n
   if l==0
      d=s(N);
   else
      c=C(2:length(C));
```

```
    d=mod(s(N)+sum(c.*s(N-1:-1:N-length(c))),2); %Next Discrepancy
  end

  if d==1
  T=C;
  P=gfconv(B,[zeros(1,N-1-m) 1]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if length(C)<length(P)            %                    %
    C=[C zeros(1,length(P)-length(C))]; % Rows from 18-25 is equal to %
  elseif length(C)>length(P)         % say C(D)<--C(D)+B(D).D^(N-m)%
    P=[P zeros(1,length(C)-length(P))]; %                    %
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    C=mod(C+P,2);

  if l<=(N-1)/2
    l=N-l;
    m=N-1;
    B=T;
  end
  end
  L(N)=l;
  N=N+1
end
y=0:0.5:n/2;
plot(0:n,[0 L],0:n,y,'--r')
legend('LC Profile', 'N/2 line');
xlabel('Sequence'); ylabel('Linear Complexity');
```

REFERENCES

1. Johansson, T., W. Meier, and F. Muller, *Cryptanalysis of Achterbahn*, http://www.ecrypt.eu.org/stream/papersdir/064.pdf, 2006.

2. 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.

3. Siegenthaler, T., "Decrypting a Class of Stream Ciphers Using Ciphertext Only", *IEEE Transactions on Computer*, Vol. C-34, pp. 81-85, 1985.

4. Menezes, A., P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography", CRC press, 1996.

5. Zeng, K., C. Yang, D. Wei, and T.R.N Rao, "Pseudorandom bit generators in stream-cipher cryptography", *IEEE Computer*, Vol. 24, pp.8-17, 1991.

6. Advanced Encryption Standard, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

7. Gouget, A. and H. Sibert, "The Bit-Search Generator", *The State of the Art of Stream Ciphers: Workshop Record*, pp. 60-68, 2004.

8. Meier, W. and O. Staffelbach, "The Self-Shrinking Generator", in A. DeSantis (ed.) *Advances in Cryptology – EURO-CRYPT'94 Proceedings*, LNCS 950, pp. 205–214, 1994.

9. 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.

10. Gammel, B., R. Göttfert, and O. Kniffler, "The Achterbahn Stream Cipher", *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/002, 2005.

11. 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.

12. RSA Labs, *Frequently Asked Questions on Today's Cryptography*, http://www.rsasecurity.com/rsalabs/node.asp?id=2152, 2000.

13. Hellman, M., "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory*, Vol. 26, pp. 401–406, 1980.

14. Babbage, S., "Improved exhaustive search attacks on stream ciphers", *European Convention on Security and Detection*, IEE Conference publication No. 408, pp. 161-166, 1995.

15. Golic, J., "Cryptanalysis of alleged A5 stream cipher", *Eurocrypt'97*, LNCS Vol. 1233, pp. 239-255, 1997.

16. Biryukov, A., S. Mukhopadhyay, P. Sarkar, "Improved Time-Memory Trade-offs with Multiple Data", *Lecture Notes in Computer Science*, proceedings of SAC'2005.

17. Hong, J. and P. Sarkar, "Rediscovery of Time Memory Tradeoffs", *Cryptology ePrint Archive*, Report 2005/090, 2005.

18. Mukhopadhyay, S. and P. Sarkar, "A New Cryptanalytic Time/Memory/Data Trade-off Algorithm", *Cryptology ePrint Archive*, Report 2006/127, 2006.

19. Biryukov, A. and A. Shamir, "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers", *Proceedings of Asiacrypt 2000*, LNCS Vol. 1976, pp 1-13, 2000.

20. Blaser, A. and P. Heinzmann, “New cryptographic device with high security using public key distribution”, *Proceedings of IEEE Student Paper Contest 1979-1980*, pp. 145-153, 1982.

21. Chepyzhov, V. and B. Smeets, “On A Fast Correlation Attack on Certain Stream Ciphers”, *Advances in Cryptology -- EUROCRYPT '91,* pp. 176-185, 1991.

22. Gammel, B. and R. Göttfert, “Combining certain nonlinear feedback shift registers”, *Workshop Record of SASC – The State of the Art of Stream Ciphers*, pp. 234–248, Brugge, Belgium, 2004.

23. Gammel, B., R. Göttfert, and O. Kniffler, “ACHTERBAHN-128/80”*, eSTREAM, ECRYPT Stream Cipher Project*, 2006.

24. Kanso, A. A., “The alternating step(*r; s*) generator” ,*Technical report*, King Fahd University of Petroleum and Minerals, Hail, Saudi Arabia, September 2002.

25. Bruwer, C. S., “Correlation attacks on stream ciphers using convolutional codes”, *M.S. Thesis*, University of Pretoria, October 2004.

26. Meier, W. and O. Staffelbach, “Fast correlation attack on certain stream ciphers”, *Journal of Cryptology*, pp. 159-176, 1989.

27. Simpson, L., J. Golic, and E. Dawson, "A probabilistic correlation attack on the shrinking generator," Information Security and Privacy - Brisbane '98, *Lecture Notes in Computer Science*, Vol. 1438, pp. 147-158, 1998.

28. Chepyzhov, V., T. Johansson, and B. Smeets, “A simple algorithm for fast correlation attacks on stream ciphers”, *In Fast Software Encryption, FSE 2000,* LNCS Vol. 1978, pp. 181–195, 2001.

29. Mihaljevic, M., M. Fossorier, and H. Imai," A low-complexity and high-performance algorithm for the fast correlation attack", *In Fast Software Encryption, FSE 2000,* LNCS Vol. 1978, pp. 196-212, 2000.

30. Canteaut, A. and M. Trabbia, "Improved fast correlation attacks using parity-check equations of weight 4 and 5", in *Advances in Cryptology - EUROCRYPT 2000*, LNCS Vol. 1807, pp. 573–588, 2000.

31. Johansson, T. and Jonsson F., "Improved fast correlation attack on stream ciphers via convolutional codes", *Advances in Cryptology - EUROCRYPT'99*, *Lecture Notes in Computer Science*, Vol. 1592, pp. 347-362, 1999.

32. Golic, J. and L. O'Connor, "Embedding and probabilistic correlation attacks on clock controlled shift registers", *Advances in Cryptology - EUROCRYPT '94*, *Lecture Notes in Computer Science*, Vol. 950, pp. 230-243, 1995.

33. Coppersmith, D., H. Krawczyk, Y. Mansour, "The Shrinking Generator", in D. R. Stinson (ed.), *Advances in Cryptology – CRYPTO'93 Proceedings*, LNCS Vol. 773, pp. 22–39, 1993.

34. Zeng, K., C. H. Yang, and T. R. N. Rao, "On the linear consistency test (LCT) in cryptanalysis with applications," *Advances in Cryptology - CRYPTO '89*, *Lecture Notes in Computer Science*, Vol. 435, pp. 164-174, 1990.

35. Johansson, T., "Reduced complexity correlation attacks on two clock-controlled generators" *Advances in Cryptology - ASIACRYPT '98*, *Lecture Notes in Computer Science*, Vol. 1514, pp. 342-357, 1998.

36. Golic, J., "Intrinsic statistical weakness of keystream generators", *Advances in Cryptology - ASIACRYPT '94, Lecture Notes in Computer Science*, Vol. 917, pp. 91-103, 1995.

37. Golic, J., “Towards Fast Correlation Attacks on Irregularly Clocked Shift Registers”, *Advances in Cryptology - EUROCRYPT '95, Lecture Notes in Computer Science*, Vol. 921, pp. 248-262, 1995.

38. Courtois, N., “Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt”, *ICISC 2002,* LNCS Vol. 2587, Seoul, Korea, 2002.

39. Courtois, N. and W. Meier, “Algebraic Attacks on Stream Ciphers with Linear Feedback”, *Eurocrypt 2003*, LNCS Vol. 2656, pp. 345-359, Warsaw, Poland, 2003.

40. Armknecht, F., *A Linearization Attack on the Bluetooth Key Stream Generator*, http://eprint.iacr.org/2002/191/, 2002.

41. Courtois, N., “Fast Algebraic Attacks on Stream Ciphers with Linear Feedback”, *Crypto 2003*, LNCS Vol. 2729, pp.177-194, 2003.

42. Gouget, A., Sibert H., Berbain C., Courtois N., Debraize B., Mitchell, C., “Analysis of the Bit-Search Generator and Sequence Compression Techniques”, in H. Gilbert and H. Handschuh (eds.), *FSE 2005, Lecture Notes in Computer Science*, Vol. 3557,196-214, 2005.

43. Massey, J.L., “Shift Register synthesis and BCH decoding”, *IEEE Transactions on Information Theory*, IT-15, pp. 122-127, 1969.

44. FIPS PUB 140-2, *Security Requirements for Cryptographic Modules*, http://csrc.nist.gov/cryptval/140-2.htm, 2002.

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

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

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

48. 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.

49. 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.

REFERENCES NOT CITED

Erguler, I., *The Editing bit-search generator,* Boğaziçi University, 2006.