A NEW NONLINEAR COMBINATION GENERATOR "MYBOUN"

by

Meltem Doğaner Özgan

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

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

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2006

ACKNOWLEDGEMENTS

Many people have contributed to my thesis during last two years . First and foremost, I would like to thank to my supervisor Prof. Emin Anarım for his guidance, and help during my thesis. I am grateful for him to encourage me to work on cryptosystems.

I want to thank to Faculty of the Department of Electrical and Electronic Engineering for their support. And also, many thanks to committee for their kindness of participation to my thesis. I also want to thank to gsm project group for their attandance to all meetings that I found chance to share all details of my design.

The last but not least I would like to thank my parents, my sister Özlem and my dearest Serkan. I always felt their support, courage, endless love, and belief on my success.

ABSTRACT

A NEW NONLINEAR COMBINATION GENERATOR "MYBOUN"

In cryptographic systems, usually linear feedback shift register based keystream generators are used because of the fact that they can produce sequences with large period, good statistical properties. They are suitable for hardware implementation. Their structure can be analyzed by using algebraic techniques. But a linear feedback shift register by itself can not satisfy high linear complexity property that is required for a keystream generator. One of the methodologies used for destroying the linearity of linear feedback shift registers is to use a nonlinear combining function on the outputs of several linear feedback shift registers. This type of keystream generators is called nonlinear combination generators.

In this thesis, a new nonlinear combination generator design named MyBoun is proposed. The main property of MyBoun is resistance against algebraic attacks which is provided by Alternative Bit Search Generator (ABSG) component. It is investigated that MyBoun has all properties of a required keystream generator. It has large period, high linear complexity, good statistical results, high throughput rate, and the characteristic of randomness. MyBoun is designed as a simple hardware oriented, modular keystream generator. Security of MyBoun is analyzed with respect to some known attacks and it is shown that MyBoun has resistance against all those attacks. MyBoun is designed for all applications which need a keystream generator that is secure against attacks, that is easy for hardware implementations, that has characteristic of randomness, good statistical properties, high throughput rate, high period and high linear complexity.

ÖZET

YENİ BİR DOĞRUSAL OLMAYAN BİRLEŞİK ÜRETİCİSİ "MYBOUN"

Şifreleme sistemlerinde , genellikle yüksek periyotlu ve iyi istatiksel özelliklere sahip diziler üretebilmeleri sebebiyle Doğrusal Geri Beslemeli Kayan Saklaç (LFSR) tabanlı anahtar üreticileri kullanılır. Ayrıca, donanım uygulamaları için de uygundurlar. Yapıları cebirsel teknikler kullanılarak analiz edilebilir. Ancak doğrusal geri beslemeli kayan saklaç kendi başına bir anahtar üreticisinde olması gereken yüksek doğrusal karmaşıklığa sahip değildir. Doğrusal geri beslemeli kayan saklaçların doğrusallık özelliğini kırmak için kullanılan yöntemlerden biri de pekçok doğrusal geri beslemeli kayan saklacın çıktılarında doğrusal olmayan birleştirici fonksiyon kullanılmasıdır. Bu tipteki anahtar üreticileri doğrusal olmayan birleşik üreticileri olarak adlandırılırlar.

Bu tezde, MyBoun isminde yeni bir doğrusal olmayan birleşik üretici tasarımı önerilmektedir. MyBoun'un temel özelliği Alternatif İkil Arama Üretici (ABSG) elemanı sayesinde cebirsel ataklara karşı dirençli olmasıdır. Görülmüştür ki, MyBoun talep edilen bir anahtar üreticisinde olması geren tüm özelliklere sahiptir. Yüksek periyoda, yüksek doğrusal karmaşıklığa, iyi istatiksel özelliklere, yüksek çıktı üretim oranına, ve rastgelelik özelliğine sahiptir. MyBoun basit bir şekilde donanıma yatkın ve birimsel bir anahtar üretici olarak tasarlanmıştır. MyBoun'un güvenliği bazı belirli saldırılar açısından incelenmiş ve tüm bu saldırılara karşı dirençli olduğu görülmüştür. MyBoun saldırılara karşı güvenli, donanımsal uygulaması kolay olan, rastgelelik özelliğine sahip, iyi istatiksel özelliklere sahip, yüksek çıktı üretim oranlı, yüksek periyotlu ve yüksek doğrusal karmaşaya sahip anahtar üretici ihtiyacı olan tüm uygulamalar için tasarlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

c_i	i^{th} Ciphertext Element
D	Amount of Real-Time Data Available To The Attacker
F	Nonlinear Combination Function
IV	Initialization Vector
k	Key
K	Secret Key Vector
LC	Linear complexity
L_1, L_2, L_3, L_4, L_5	Lengths of Linear Feedback Shift Registers in MyBoun
m_i	i^{th} Plaintext Element
M	Random Access Memory Requirement
N	Search Space (Number of Possible Secret Session Keys)
P	Period
P_1, P_2, P_3, P_4, P_5	Feedback Polynomials of Linear Feedback Shift Registers in MyBoun
R_1, R_2, R_3, R_4, R_5	Linear Feedback Shift Registers in MyBoun
T	Time Required In the Real-Time Phase of the Attack
z_i	i^{th} Keystream Element
CBC	Cipher-Block Chaining
CFB	Cipher Feedback
CR	Control Register in a Clock Controlled Keystream Generator
ECB	Electronic Codebook Mode
FSR	Feedback Shift Register
gcd	Greatest Common Divider
GR	Generator Register in a Clock Controlled Keystream Generator
lcm	Least Common Multiple
LFSR	Linear Feedback Shift Register
MyBoun	The name of proposed nonlinear combination generator
OFB	Output Feedback
XL	Extension and Linearization Technique in Algebraic Attack

1. INTRODUCTION

Cryptology is the art of transferring a plaintext message in a secure manner from a sender to a receiver. Encryption is defined to be the process changing the plaintext message to a ciphertext message to accomplish the secure transmission. Decryption is the process to turn ciphertext back into plaintext message. During encryption, mostly key-based algorithms are utilized.

In Cryptosystems, there exists two general types of key-based algorithms, symmetric-key and public-key. In public-key algorithms, the sender uses the encrytion key which can be public during encrytion process, on the other hand the receiver uses decryption key which is private during decrytion process. Any stranger can encryt the message with the public encryption key, but only the receiver supplied with private decrytion key can decrypt the message. In symmetric-key algorithms, the receiver and the sender use a private key determined before secure communication. The communication remains safe as long as the private key remains secret. [1]

There are two basic types of symmetric-key algorithms, called as block ciphers and stream ciphers. Block ciphers encrypt a block of characters of a plaintext message once at a time. They are mandatory for some telecommunication applications, as buffering is limited or as characters must be received individually. They have limited or no error propagation, and may also be advantageous in situations where transmission errors are probable.

Stream ciphers are generally classified as being self-synchronizing stream ciphers or synchronous stream ciphers. Self-synchronizing stream ciphers which are also called as asynchronous stream ciphers generate the keystream as a function of a fixed number of previous ciphertext bits and the key. Synchronous Stream Ciphers generate keystream independently of the plaintext and ciphertext messages.

In the rest of this section, general information on key-based algorithms is given. Subsection 1.1 includes explanations over symmetric-key and public-key algorithms. Subsection 1.2 gives information about stream ciphers and block ciphers. Subsection 1.3 gives explanation about methods of attacks against cryptosystems. Ciphertext-only attack, known plaintext attack, chosen plaintext attack, and chosen ciphertext attack are explained. Subsection 1.4 gives outline of this thesis.

1.1. Key-Based Algorithms

Public key algorithms and symmetric key algorithms are types of key-based algorithms. They are described in following subsections 1.1.1 and 1.1.2.

1.1.1. Public Key Algorithms

Public key algorithms are one of the key-based algorithms. They are also called as asymmetric algorithms. In public key algorithms, the key used for encryption and the key used for decryption are different from each other. The sender encrypts the plaintext message with encryption key, which can be public to every user, but only the intended receiver has the private decryption key to return the received ciphertext to original plaintext. The data transmission or communication remains safe, as long as the private decryption key remains safe. The encryption and decryption model for public key algorithms is shown in figure 1.1.

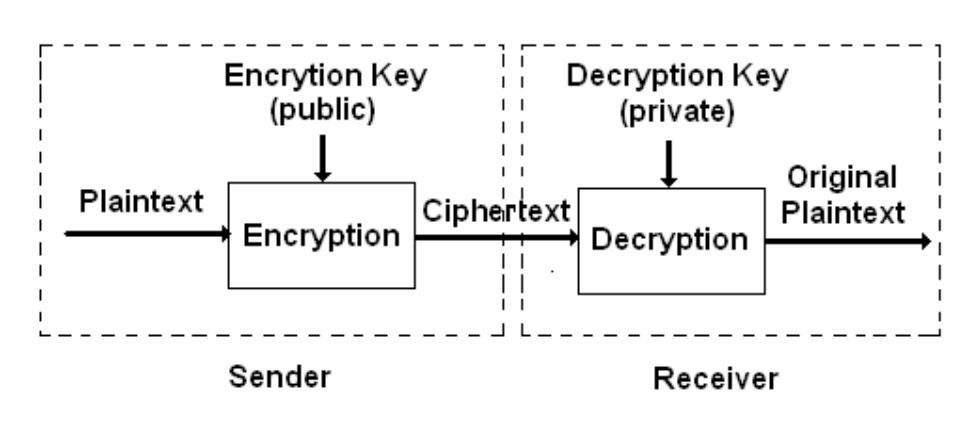

Figure 1.1. The encryption and decryption model for public key algorithms

The encryption function performed by the sender can be denoted as following, where the sub index K refers to public encryption key, M refers to plaintext message and C refers to ciphertext message:

$$E_K(M) = C; \tag{1.1}$$

The decryption function performed by the receiver can be denoted as following, where the sub index K refers to private decryption key, C refers to ciphertext message and M refers to plaintext message:

$$D_K(C) = M; \tag{1.2}$$

1.1.2. Symmetric Key Algorithms

Symmetric Key algorithms are one of the key-based algorithms that are also called as conventional algorithms. In symmetric key algorithms, the key used for encryption and the key used for decryption are mostly same or can be calculated from each other. Before communication starts, sender and receiver determine a key. Not only the sender encrypts the plaintext message with that determined key, but

also the receiver decrypts ciphertext to obtain original plaintext with that determined key. The data transmission or communication remains safe, as long as the determined private key remains safe. The encryption and decryption model for symmetric key algorithms is shown in figure 1.2.

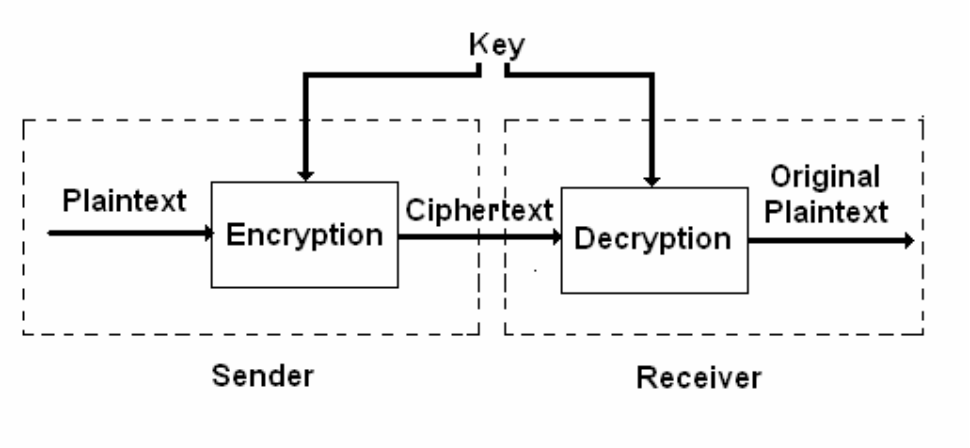

Figure 1.2. The encryption and decryption model for symmetric key algorithms

The encryption function that is used to turn plaintext message to ciphertext message by the sender can be denoted as following, where the sub index K refers to the determined private encryption key, M refers to plaintext message and C refers to ciphertext message:

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

The decryption function that is utilized by the receiver to return ciphertext message to original plaintext message can be denoted as following, where the sub index K refers to the determined private encryption key, C refers to ciphertext message and M refers to plaintext message: as it was in encryption function:

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

1.2. Basic Types of Symmetric Key Algorithms

There are two basic types of symmetric-key algorithms, called as stream ciphers and block ciphers. Their specific properties determine their use of area.

1.2.1. Block Ciphers

Block ciphers encrypt a block of characters of a plaintext message once at a time. The general structure model used for block ciphers is seen in figure 1.3.

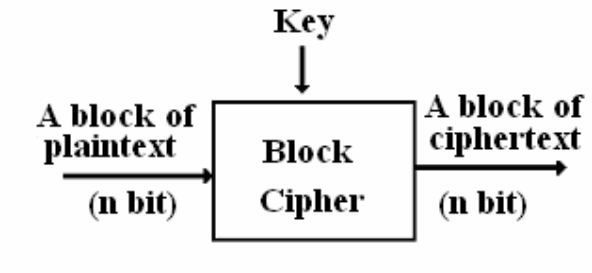

Figure 1.3. The general structure model used for block ciphers

The number of bits in a block is usually chosen as 64 bits of length, which is large enough for security requirements, but small enough to work with. For specific applications, larger values can be chosen. There are four commonly used types of mode of operation in block ciphers. They are ECB mode (Electronic Codebook Mode of Operation), CBC mode (Cipher-Block Chaining Mode of Operation), CFB mode (Cipher Feedback Mode of Operation), and OFB mode (Output Feedback Mode of Operation). [2]

- ECB Mode: In ECB mode, the plaintext messages exceeding n bits are partitioned to n-bit blocks. All blocks are encrypted separately and independently of other blocks. In this mode, encryption of same blocks of plaintexts gives same ciphertext blocks with same key. When a single bit error occurs during the encryption of one block, this only causes a single bit error in the plaintext block during decryption of that block. Other blocks are not affected.

- CBC Mode: In CBC mode, there exists a private n-bit initialization vector. First plaintext block is XORed with that initialization vector before it is encrypted. Obtained ciphertext block is stored in a feedback register and XORed with the next plaintext message before it is encrypted. This process is repeated until the end of messages is reached. In this mode, if identical key is used and the order of plaintext blocks are not changed, identical ciphertext blocks are obtained. Any change in the order of ciphertext blocks changes the order of decrypted plaintext message blocks. When a single bit error in a plaintext block occurs during the encryption, this not only causes an error in that ciphertext block, but also in all subsequent ciphertext blocks. But during decryption process, that affect is reversed and obtained plaintext message only has original single bit error.

- CFB mode: For some applications, data is needed to be encrypted and transmitted without any delay. In CFB mode, encryption of a unit smaller than block size is managed to be performed, although it was impossible in CBC mode to start encryption process before the total block of data is received. Let's give an example for an 8-bit CFB mode working with a 64 bit block algorithm. In CFB mode, block algorithm operates on a queue with 64 bit input block size. Initially, the queue is filled with 64 bit initialization vector. The queue is encrypted and the left-most eight bits of the obtained result are XORed with the first 8-bit character of the plaintext. The first 8-bit character of the ciphertext is obtained at the end of that process. These eight bits are transmitted, and also moved to the right-most eight bit positions of the queue. All the other bits in the queue is shifted eight to the left, with discarding eight left-most bits in the queue before the shift process begins. Then the next plaintext 8 bits are encrypted in the same manner. In CFB mode, a single bit error in plaintext affects all subsequent ciphertexts, but during decryption process, that affect is reversed and obtained plaintext message only has original single bit error. But a single error in the ciphertext not only causes a single error in the plaintext but also by entering the shift register , it causes ciphertext failures until it becomes to be discareded as falling off the other end of the register. In general, it can be said that a single ciphertext error in an n-bit CFB affects the decryption of the current and next m/n-1 blocks, where m refers to block size.

- OFB Mode**:** In OFB mode, a method running a block cipher as a synchronous stream cipher is used. In an n-bit OFB, the same algorithm is used as in the CFB mode, except that *n* bits of the previous output block are moved into the right-most positions of the queue. Decryption does the reverse of that process. OFB mode has no error extension. A single-bit error in the ciphertext causes a single-bit error in the recovered plaintext. This can be useful in some digitized analog transmissions, like digitized voice or video, where the occasional single-bit error can be tolerated but error extension cannot. On the other hand, a loss of synchronization is fatal. If the shift registers on the encryption end and the decryption end are not identical, then the recovered plaintext will be gibberish. Any system that uses OFB mode must have a mechanism for detecting a synchronization loss and a mechanism to fill both shift registers with a new (or the same) initialization key to regain synchronization. [1].

Table 1.1. Summary of block cipher modes [1]

ECB:	CBC:
***Security*:** - Plaintext patterns are not concealed. - Input to the block cipher is not randomized; it is the same as the plaintext. +More than one message can be encrypted with the same key. - Plaintext is easy to manipulate, blocks can be removed, repeated, or interchanged. ***Efficiency*:** + Speed is the same as the block cipher. Ciphertext is up to one block longer than the plaintext, due to padding. - No preprocessing is possible. + Processing is parallelizable. ***Fault-tolerance*:** - A ciphertext error affects one full block of plaintext. - Synchronization error is unrecoverable.	***Security*:** + Plaintext patterns are concealed by XORing with previous ciphertext block. +Input to the block cipher is randomized by XORing with the previous ciphertext block. + More than one message can be encrypted with the same key. +/- Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes. ***Efficiency*:** + Speed is the same as the block cipher. - Ciphertext is up to one block longer than the plaintext, not counting the IV. - No preprocessing is possible. +/- Encryptions not parallelizable; decryption is parallelizable and has a random-access property. ***Fault-tolerance*:** - A ciphertext error affects one full block of plaintext and the corresponding bit in the next block. - Synchronization error is unrecoverable

CFB:	OFB/Counter:
***Security*:** + Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key provided that a different IV is used. +/- Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the last block can be changed, and repetition allows some controlled changes. ***Efficiency*:** + Speed is the same as the block cipher only when the feedback is the same size as the underlying block cipher. + Ciphertext is the same size as the plaintext, not counting the IV. + Processing is possible before the message is seen. - Some preprocessing is possible before a block is seen; the previous ciphertext block can be encrypted. +/- Encryption is not parallelizable; decryption is parallelizable and has a random-access property. ***Fault-tolerance*:** - A ciphertext error affects the corresponding bit of plaintext and the next full block. +Synchronization errors of full block sizes are recoverable. 1-bit CFB can recover from the addition or loss of single bits.	***Security*:** + Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. - Plaintext is very easy to manipulate, any change in ciphertext directly affects the plaintext. ***Efficiency*:** -/+ OFB processing is not parallelizable; counter processing is parallelizable. ***Fault-tolerance*:** + A ciphertext error affects only the corresponding bit of plaintext. -Synchronization error is unrecoverable.

1.2.2. Stream Ciphers

Stream ciphers encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation which varies with time. Stream ciphers are generally faster than block ciphers and have less complex hardware circuitry. They are mandatory for cases (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable. [2]

Stream ciphers are generally classified as being self-synchronizing stream ciphers or synchronous stream ciphers.

1.2.2.1. Self Synchronizing Stream Ciphers. Self synchronizing stream ciphers which are also called as asynchronous stream ciphers generate the keystream as a function of a fixed number of previous ciphertext bits and the key. The model for the encryption and decryption for an asynchronous stream cipher is shown in figure 1.4. In this model, k represents key, g is the function that generates the keystream zi, and. h is the output function that takes the keystream and plaintext mi as parameter and produces ciphertext ci.

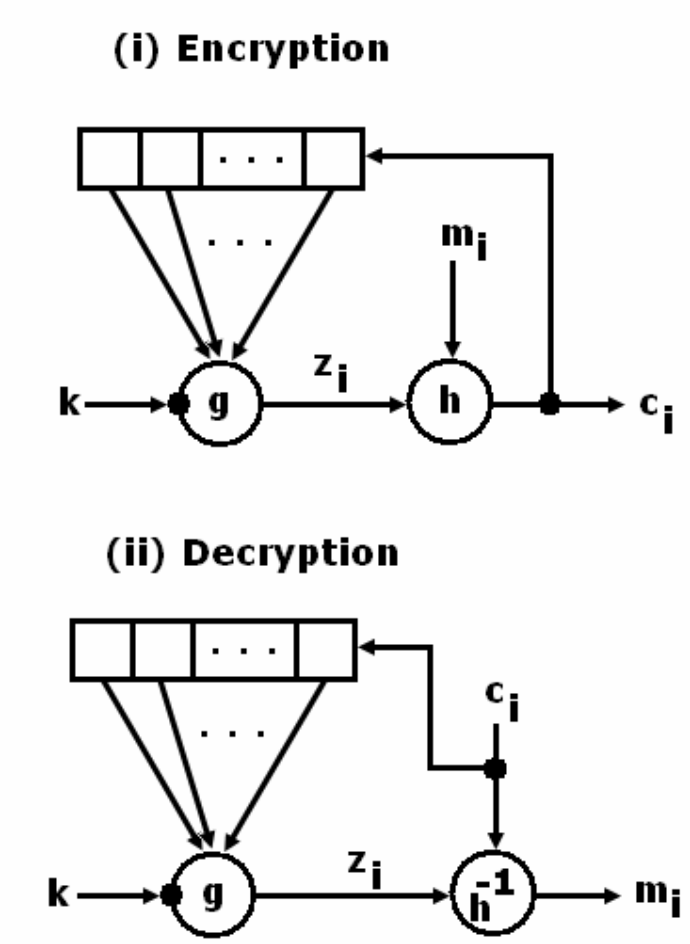

Figure 1.4. General model of a self-synchronizing stream cipher

In asynchronous stream ciphers self synchronization is possible, if any bits are inserted or deleted to ciphertext bits. This stems from the fact that only a fixed number of preceeding ciphertext bits are used in decryption process. Although a fixed number of plaintext characters are unrecoverable, proper decryption is automatically reestablished. Error propagation is down side of the asynchronous stream ciphers .Lets assume that the state of a self synchronization stream cipher depends on n previous ciphertext bits. If an error occurs during transmission of ciphertext, the decryption keystream generator will incorrectly produce n keystream bits. Therefore, for each ciphertext error, there will be n corresponding plaintext errors, until the incorrect bit leaves the internal state of the stream cipher.

1.2.1.2. Synchronous Stream Ciphers. Synchronous Stream Ciphers generate keystream independently of the plaintext and ciphertext messages. The model for the encryption and decryption for an asynchronous stream cipher is shown in figure 1.5. In this model, k stands for key, σ0 represents initial state, f is the next-sate function, g is the function that generates the keystream zi, and h is the output function that takes the keystream and plaintext mi as parameter and produces ciphertext ci.

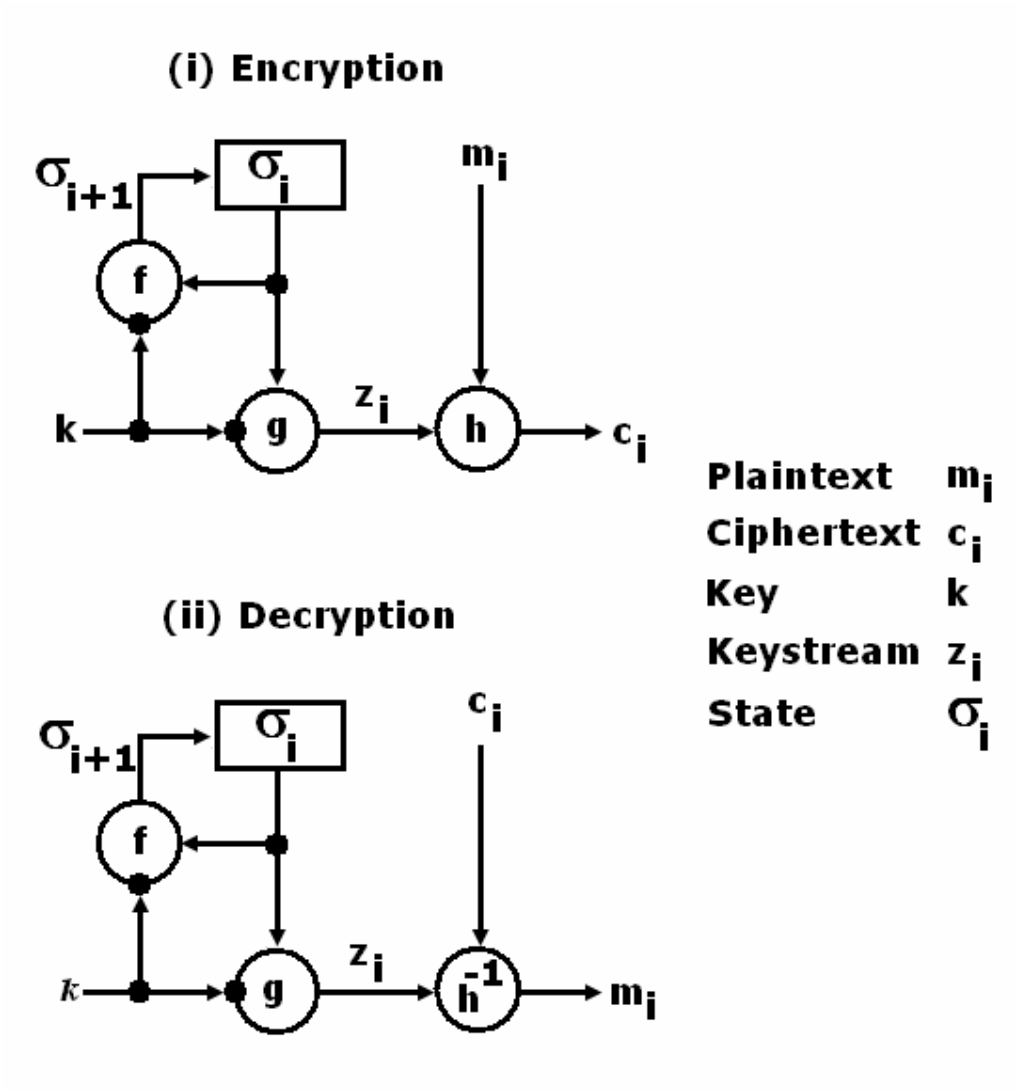

Figure 1.5 General model of a synchronous stream cipher

In synchronous stream ciphers, both the sender and receiver must use the same key and operate at the same state within that key so as to be synchronized. Proper decryption is managed if and only if the receiver and the sender are in synchronization. If synchronization is lost due to ciphertext digits being deleted or inserted during transmission, then decryption fails and can only be restored through additional techniques for resynchronization. Techniques for re-synchronization include re-initialization, placing special markers at regular intervals in the ciphertext, or, if the plaintext contains enough redundancy, trying all possible keystream offsets.[2] Error is not propagated in synchronous stream ciphers. If a single bit in ciphertext is changed during transmission, this does not affect the decryption of other ciphertext bits.

Most of the synchronous stream ciphers are binary additive stream ciphers . in which the key stream, plaintext, and ciphertext digits are binary digits, and the output function is the XOR of plaintext and key stream sequence. In figure 1.6, general model of a binary additive stream cipher is shown.

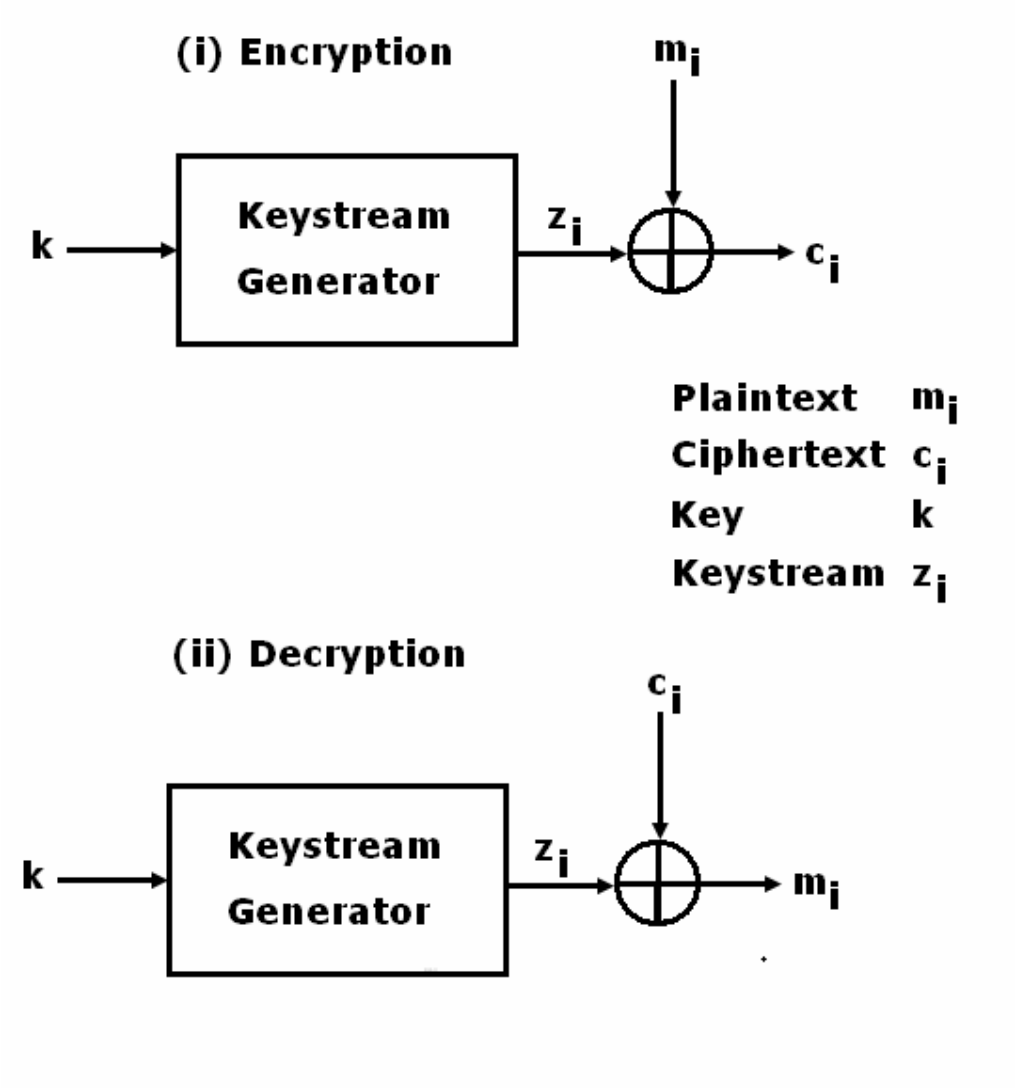

Figure 1.6. General model of binary additive stream ciphers

In most of the keystream generators in binary additive stream ciphers Linear Feedback Shit Registers (LFSRs) are used. Section 2 is dedicated to the analysis of fundamental component of the keystream generators, LFSRs.

1.3. Methods of Attacks against Cryptosystems

An attacker can be defined as a third party in a cryptosytem that has access to all public information, and tries to derive some private information in the system. The

algorithm employed by the attacker is called as attack. The system remains secure until the attacker can not reach to any private information. The first and the 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 thesecrecy of the algorithms.

Thus, the security of the design shell not rely on secret components of the primitive, since it is likely that the design will be leaked to an attacker. [3] The methods of the attacks can be classified into four main groups according to the amount of information the attacker has obtined and the aim of the attacker.

1.3.1. Ciphertext-only Attack

For this kind of attack, the attacker only knows the ciphertext messages cossrresponding to plaintext messages that sender wants to transmit to the receiver. The attacker tries to recover plaintext messages, or tries to obtain the secret key so as to decrypt othet ciphertext messages that are encrypted with that secret key.This is the most diffucult attack, since the attacker has the least amount of information among the other attacks that will be listed below.

1.3.2. Known Plaintext Attack

For this kind of attack, the attacker not only knows the ciphertext messages, but also the corresponding plaintext messages. It might seem imposibble for an attacker to obtain both the ciphertext and the plaintext messages, but there are some cases where this could happen. For example, during telecommunication we usually start and end our conversion with some common words such as hello, hi, bye, or take care etc. If an attacker knows the ciphertext message regarding to that message, he can apply attacks of type known plaintext.

1.3.3. Chosen Plaintext Attack

For this kind of attack, the attacker can obtain every ciphertext message corresponding to any plaintext message he has chosen to enrypt. This attack is more powerful than known plaintext attack, since the attacker can choose plaintext messages that will reveal the key of the cryptosystem.

1.3.4. Chosen Ciphertext Attack

This kind of attack is very similar to known-plain attacks, but this attack is more powerful than it, since the attacker knows much more information. As in the plaintext message, the attacker can obtain every ciphertext message corresponding to any plaintext message he has chosen to enrypt. Additionally, the attacker can choose any ciphertext message except the one he is trying to break, and obtain corresponding plaintext messages after decrypting the ciphertext message he has chosen. [3] The aim of the attacker is to reveal the secret key.

1.4. Thesis Outline

The thesis consists of seven sections. Section 1 is an introduction to cryptosystems. In Section 2, LFSRs (Linear Feedback Shift Registers) and the reasons of their widely usage in many keystream generators are mentioned. In Subsection 2.1, analysis of basic properties of LFSR that are period, linear complexity analysis, and statistical properties is performed. In Subsection 2.2, the methodologies for destroying the linear properties LFSRs are described. Section 3 gives analysis of an existing nonlinear combination generator “Geffe Generator”. This section is very important to observe what should be the design criteria of a nonlinear combination to obtain a keystream generator with desirable properties. In Section 4, a new nonlinear combination design “MyBoun” is introduced. All steps are explained in this section from initialization to output generation. In Section 5, properties of MyBoun are analyzed. The period and linear complexity of MyBoun are computed. Static properties are depicted with the help of FIBS 140-2 and NIST statistical

tests. In Section 6, security of MyBoun is analyzed against brute force attacks, time memory trade off attacks, correlation attacks, and algebraic attacks. Finally, Section 7 gives the conclusions of the thesis.

2. LINEAR FEEDBACK SHIFT REGISTERS (LFSRS)

A feedback shift register is made up of two parts: a shift register and a feedback function . Figure 1.7 shows general model of feedback shift registers. The shift register is a sequence of bits. The length of a shift register is figured in bits; if it is *n* bits long, it is called as an *n*-bit shift register. Each time a bit is needed, all of the bits in the shift register are shifted 1 bit to the right. The new left-most bit is computed with feedback function. The output of the shift register is 1 bit, often the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating. [1]

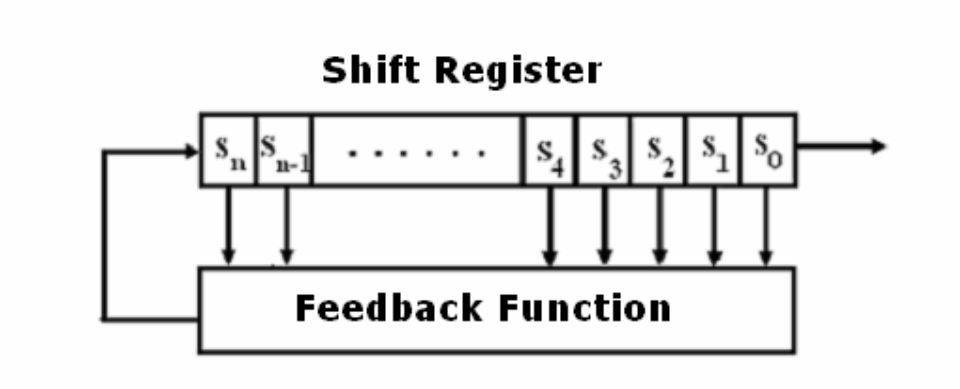

Figure 1.7. General model of feedback shift registers

Linear Feedback Shift Register (LFSR) is a special feedback register. The feedback function of a LFSR is XOR of certain bits in the register. The list of the XORed bits of the register is called as tap sequence. Figure 1.8 shows general model of linear feedback shift registers.

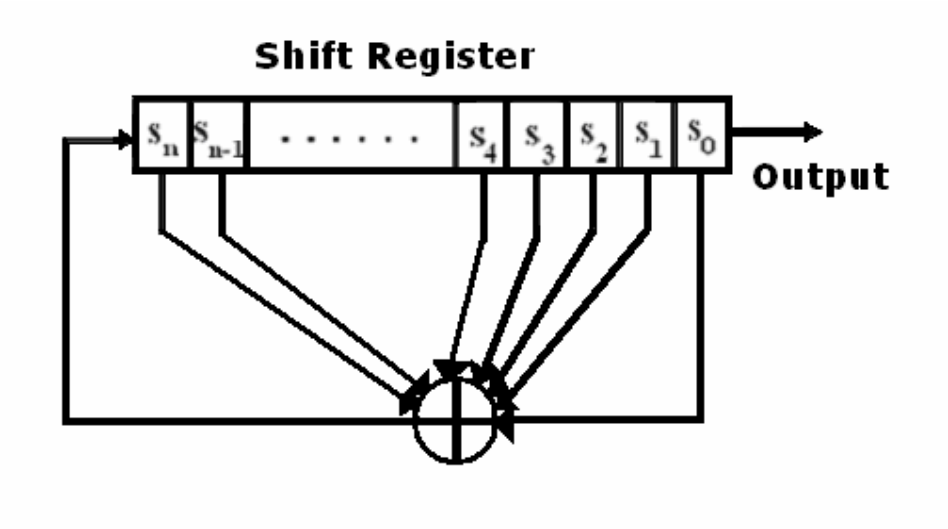

Figure 1.8. General model of linear feedback shift registers

For an n-bit LFSR, initial value vector of the register bits from S_0 to S_n is called as initial state of the LFSR. A register with an initial state full of zeros causes the LFSR to give an output of never ending stream of zeros. Since this is not a useful solution, possible internal states are decreased from 2n to $2^n - 1$. Consequently, there exists $2^n - 1$ possible internal states for an *n*-bit LFSR. LFSRs that cycle through all 2^n-1 internal states generate 2^n-1 bit long pseudo random sequences before repeating. These LFSRs are called as maximal-period LFSRs and the resulting output sequences are called as m-sequences.

There are several reasons why LFSRs are widely used as basic components in keystream generators. A keystream generator should generate output sequences with large periods, high linear complexity and good statistical properties to be unpredictable. With LFSRs, a keystream generator satisfying those conditions can be designed. LFSRs are well suited for hardware implementation. LFSRs produce sequences of large period. They can produce sequences with good statistical properties. They can be readily analyzed using algebraic techniques because of their structure.[2]

Next section gives explanation about the analysis of period, linear complexity and statistical properties of the LFSRs.

2.1. Properties of Linear Feedback Shift Registers

2.1.1. Period

The period of an LFSR is the length of the output sequence before it starts repeating. [1]. n-bit maximal period LFSRs cycle through all possible 2^n–1 internal states, and generate 2^n–1 bit long pseudo random sequences before repeating. In other words, an n-bit maximal period LFSR has period 2^n–1. What determines an LFSR to be a maximal period LFSR or not is the feedback function of that LFSR. Feedback function of an LFSR can be written as below:

$$f=d_1s_1 \oplus d_2s_2 \oplus d_3s_3 \oplus d_4s_4 \oplus ..... \oplus d_ns_n \quad (2.1)$$

In this formula s values are the contents of the register and d values are the feedback coefficients. These coefficients can take the value either 0 or 1. Tap sequences have d values as 1 and their contents are XORed according to feedback function in (2.1). Feedback polynomial of that LFSR can be written as below:

$$f(x)= x^n+d_{n-1}x^{n-1}+.....+d_1x +1 \quad (2.2)$$

In order for a particular LFSR to be a maximal-period LFSR, feedback polynomial must be a primitive polynomial mod 2. The degree of the polynomial is the length of the shift register. A primitive polynomial of degree n is an irreducible polynomial that divides $x^{2n-1}+1$, but not x^m+1 for any m that divides 2^n–1. [1]. There are algorithms that test the primitiveness of the polynomial and also various tables of primitive polynomials over finite fields are presented in the technical literature. [4]

For example, if a four stage LFSR has the feedback polynomial $f(x)=x^4+x^3+x^2+x+1$, then its feedback function can be defined as $f=s_4 \oplus s_3 \oplus s_2 \oplus s_1$. This LFSR is shown in figure

2.1 . Depending on the initial data loaded into its stages, it can generate 1111011110…. , 1000110001…… ,or 0100101001….. All those sequences have period of 5 with poor statistics.

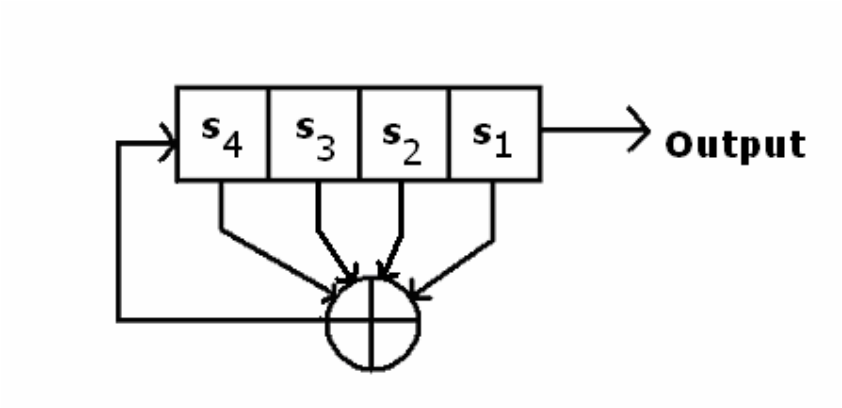

Figure 2.1. Four stage LFSR with feedback polynomial $f(x)=x^4+x^3+x^2+x+1$

If a four stage LFSR has the primitive feedback polynomial $f(x)=x^4+x+1$, then its feedback function can be defined as $f=s_4 \oplus s_3 \oplus s_2 \oplus s_1$. This LFSR is shown in figure 2.2. It generates a single nontrivial sequence of period 15 with good statistics. We have stated that maximum period for an n-bit LFSR is 2^n-1. In our case n is 4, so maximum period is 15 and it is reached by using a primitive feedback polynomial.

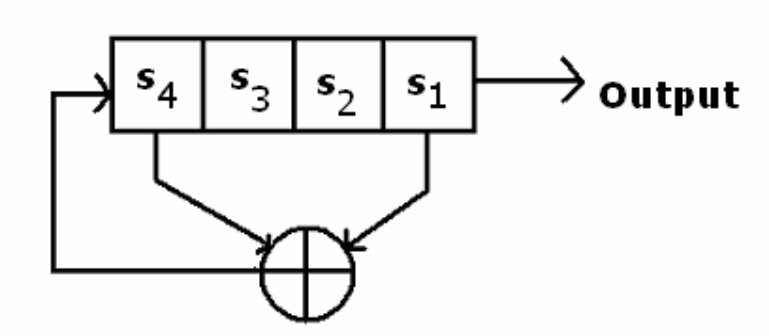

Figure 2.2. Four stage LFSR with primitive feedback polynomial $f(x)= x^4+x+1$

2.1.2. Linear Complexity

If an LFSR generates the output sequence s for some initial state, the linear complexity of that binary sequence s is denoted as L(s). If s is the zero sequence s = 0,0,0,..., then L(s) = 0; if no LFSR generates s, then $L(s)=\infty$; otherwise, L(s) is the length of the shortest LFSR that generates sequence s. Let $s=s_0, s_1, s_2....$be the binary sequence generated by the LFSR, and let L_N denote the linear complexity of the subsequence $s^N = s_0, s_1, ... , s_{N-1}$, $N \geq 0$. The sequence $L_1, L_2, ... , L_n$ is called the linear complexity profile of s.The linear complexity profile of a sequence can be computed by using the Berlekamp Massey Algorithm. [2] According to Berlekamp Massey Algorithm, for a sequence to be claimed as random, the linear complexity profile should be approximately half of the length of the subsequence ($L_N \cong N/2$). When a graph is plotted showing the relation between N and L_N; for a random sequence the expected linear complexity of the random sequence should closely follow the line L=N/2. If a sequence does not satisfy that condition, it can be said that this sequence is not random. Although this condition is necessary for a sequence to be random, it is not sufficient. Sequences with linear complexity profile closely resembling the one of a random sequence may not be random. Extra statistical tests should be applied to the binary sequence so as to decide whether it is random or not.

2.1.3. Statistical Properties

An LFSR can not be regarded as a suitable stream cipher for security applications unless its generated key stream sequences possess certain randomness properties although it has large period and high linear complexity. The number of ones in the generated binary key stream sequence should be approximately equal to the number of zeros so as to be a random. Golomb defined a PN-sequence (pseudo-noise sequence) to be a binary sequence of period *P* that satisfies the three randomness postulates. [5] According to postulates of Golomb ; the number of ones should differ from the number of zeros by at most 1 in a period. A run is defined as consequtive bits of zeros or ones in a sequence. At least half of the runs should have length 1 , at least one fourth should have length 2, at least one eighth

should have length 3, etc. Beside that , for each of these runs, there should be approximately equal number of consequtive zeros and ones. The out-of-phase of the autocorrelation should be constant. These randomness postulates can be applyied to a complete period of a key stream sequence. But, mostly the whole period of the enciphering sequence generated by the keystream generator is not used. Under these contidions, statistical tests that are applied to a subsequence with a shorter length of a whole period become more important.

In the next section ; two important statistical test suits will be explained in a detailed manner. One of them is FIPS140-2 [6] and the other one is NIST Statistical Test suite [7]. Actually there also exists FIPS140-1 statistical tests which are explained in [8]. Every sequence passing FIPS140-2 statistical tests , also passes FIPS140-1 statistical tests. Since FIPS140-2 is the one, of which the range of allowed deviation is decreased according to FIPS140-1. This is why only FIPS140-2 and NIST statistical test suits are explained in the scope of this thesis.

2.1.3.1..FIPS140-2 Statistical Tests. FIPS 140-2 Statistical Test Suite consists of four tests, called as monobit test, poker test, runs test, long runs test. All four tests are applied to the generated 20000 bit length sequence.

- Monobit Test**:** In this test, X is defined to be the number of ones in the observed 20,000 bit stream. The test is passed if the X value is in the below range:

$$9{,}725 < X < 10{,}275 \tag{2.3}$$

- Poker Test: In this test, generated 20,000 bit stream is divided into 5,000 contiguous 4 bit segments. The number of occurrences of each of the 16 possible 4 bit values are counted and stored. *f(i)* is denoted as the number of each 4 bit value i where 0 <= i <= 15. The following number is evaluated.

$$X = (16/5000) * \left[\sum_{i=0}^{15} [f(i)]^2 \right] - 5000 \tag{2.4}$$

The test is passed if the X value is in the below range:

$$2.16 < X < 46.17 \quad [6] \tag{2.5}$$

- Runs Test: A run is defined as maximal sequence of consecutive bits of either all ones or all zeros in 20,000 bit sample stream. The incidences of runs (for both consecutive zeros and consecutive ones) of all lengths (>= 1) in the sample stream are counted and stored. The test is passed if the number of runs that occur (of lengths 1 through 6) is each within the corresponding interval specified below. This must hold for both the zeros and ones; that is, all 12 counts must lie in the specified interval. For the purpose of this test, runs of greater than 6 are considered to be of length 6. [8]

Length of Run	***Required Interval***
1	2,315-2685
2	1,114-1,386
3	527-723
4	240-384
5	103-209
6 +	103-209

Table 2.1. FIPS140-2 Runs test requirements

- Long Runs Test: A long run is defined to be a run of length 26 or more (of either zeros or ones). On the sample of 20,000 bits, the test is passed if there are NO long runs. [8]

2.1.3.2. NIST Statistical Test Suit. The NIST Test Suite is a statistical package consisting of 16 tests that were developed to test arbitrarily long binary sequences to determine

whether or not these sequences satisfy some variety of different types of randomness properties. [7] The descriptions and use of purposes of 16 tests in the package are explained in the following part.

- Frequency (Monobits) Test: The purpose of this test is to verify that the number of ones and zeros in observed sequence are approximately the same as would be expected for a truly random sequence.

- Test for Frequency within a Block: This test focuses on the proportion of zeroes and ones within M-bit blocks. For a truly random sequence, the frequency of ones in an M-bit block is expected to be approximately M/2.

- Runs Test: This test determines whether the number of runs of ones and zeros of various lengths match with the ones obtained from a random sequence. This test also determines whether the oscillation between such substrings is too slow or too fast.

- Test for the Longest Run of Ones in a Block: The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones in a random sequence.

- Random Binary Matrix Rank Test: This test checks for linear dependence among fixed length substrings of the original sequence.

- Discrete Fourier Transform (Spectral) Test: This test focuses on the peak heights in the discrete Fast Fourier Transform. 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 be an indicator of a deviation from the assumption of randomness.

- Non-overlapping (Aperiodic) Template Matching Test: The focus of this test is the number of occurrences of a given non-periodic (aperiodic) pattern in sequences . A specific m-bit pattern is searched in an m-bit window. If the pattern is not found, the

window slides one bit position. When the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

- Overlapping (Periodic) Template Matching Test: The purpose of this test is to detect sequences that show deviations from the expected number of runs of ones of a given length. If a deviation from the expected number of ones of a given length is detected, it means that for that sequence there is also a deviation in the runs of zeroes. For this test, an m-bit window is used to search for a specific m-bit pattern. The window slides one bit position, if the pattern is not found. The window again slides one bit, and the search is resumed, if the pattern is found.

- Maurer's Universal Statistical Test: The purpose of the test is to determine whether or not the sequence can be significantly compressed without loss of information. If the given sequence is overly compressible, then it is considered to be a non-random sequence.

- Lempel-Ziv Complexity Test: The focus of this test is the number of cumulatively distinct patterns in the given sequence. This test determines how far the tested sequence can be compressed. If the given can be significantly compressed and does not have a characteristic number of distinct patterns, then it is considered to be a non-random sequence.

- Linear Complexity Test: The focus of this test is the length of the feedback register that can generate the given sequence. The purpose of this test is to determine whether or not the sequence is complex enough so as to be considered a random sequence. A long feedback register implies randomness.

- Serial Test: This test focuses on the frequency of all possible overlapping m-bit patterns across the entire sequence. The number of occurrences of the 2m m-bit

overlapping patterns is detected and checked if it is approximately the same with the one obtained from a random sequence. The pattern can overlap.

- Approximate Entropy Test: This test compares the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) with the results of random sequence.

- Cumulative Sum (Cusum) Test: The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. This test determines whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. For random sequences, the excursions of random walk away from zero are expected to be near zero.

- Random Excursions Test: This test focuses on the number of cycles having exactly K visits in a cumulative sum random walk. The purpose of this test is to compare the number of visits to a state within a random walk against the one that would be expected for a random sequence.

- Random Excursions Variant Test: The focus of this test is the total number of times that a particular state is visited (i.e., occurs) in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk. [7]

2.2. Methodologies for Destroying Linear Properties of LFSRs

For essentially all possible secret keys, generated output sequence of an LFSR-based keystream generator should have large period, high linear complexity and good statistical properties. An LFSR by itself can not satisfy high linear complexity property. This is why

there are general methodologies used for destroying the linearity properties of LFSRs. These methodologies can be listed as below:

i) Using a nonlinear filtering function on the contents of a single LFSR

ii) Using the output of one of (or more) LFSR to control the clock of one (or more) other LFSRs.

iii) Using a nonlinear combining function on the outputs of several LFSRs.[2]

2.2.1. Nonlinear Filter Generators

One of the techniques used for destroying the linear inherent in LFSRs is to generate the keystream as output of a nonlinear function of the stages of a single LFSR. General model for a nonlinear filter generator is shown in figure 2.3 .

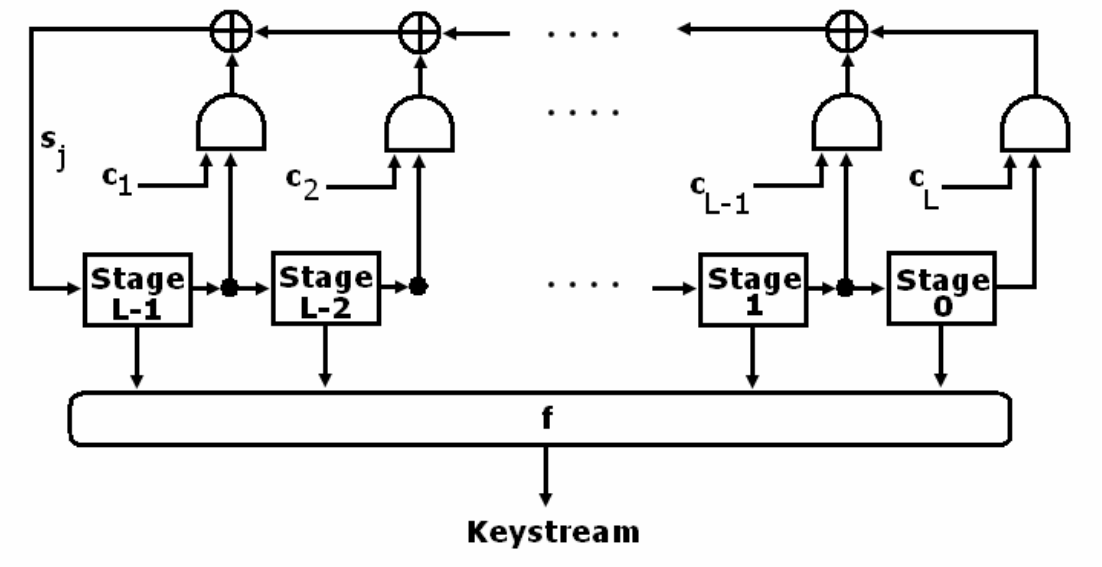

Figure 2.3. General model for a nonlinear filter generator

Period of Nonlinear Filter Generator: If a nonlinear filter generator is constructed by using a maximum-length LFSR of prime length L, then it's period is $P=2^L-1$ as it was stated in subsection 2.1.1.

Linear Complexity of Nonlinear Filter Generator: If a nonlinear filter generator is constructed by using a maximum-length LFSR of length L and a filtering function f of nonlinear order m

(i) The linear complexity of the keystream is at most:

$$L_m = \sum_{i=1}^{m} \binom{L}{i} \tag{2.6}$$

ii) For a fixed maximum-length LFSR of prime length L, the fraction of Boolean functions of nonlinear ordermwhich produce sequences of maximum linear complexity Lm is

$$Pm \approx \exp(-Lm=(L \_ 2^L)) > e-^{1/L} \tag{2.7}$$

Therefore, for large L, most of the generators produce sequences whose linear complexity meets the upper bound in (i).The nonlinear function f selected for a filter generator should include many terms of each order up to the nonlinear order of f. [2]

2.2.2. Clock Controlled Generators

In clock controlled generators, the movement of data of one LFSR is controlled by the output of another LFSR. The register enabling clocking control is called as control register, and denoted as CR. The register which generates keystream according to the output sequence of CR is called as generator register, and denoted as GR. Since GR is clocked in an irregular manner, nonlinearity is introduced into LFSR-based keystream generators. In figure 2.4 general model for a basic clock controlled generator is seen.

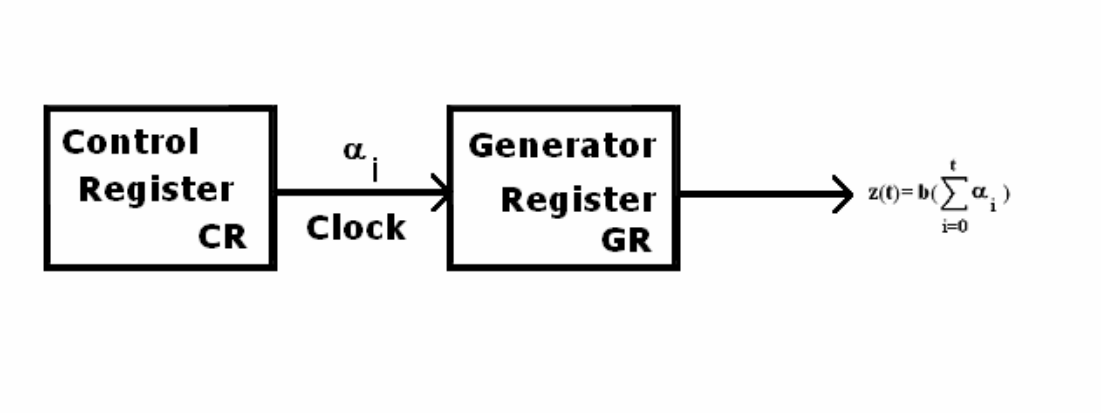

Figure 2.4. General model for a basic clock controlled generator

Period of Clock Controlled Generator: Before computing the period of the clock controlled generator, the parameters that will be used during computations and working principle of that keystream generator will be explained. Let's assume that CR in figure 2.4 generates a sequence of nonnegative integers $a = \{ai\}i\geq 0$. The period of CR is P_{CR}. GR is an LFSR over P= GF(q) with irreducible feedback polynomial $f(x)$ of degree $m > 1$ and order M. Binary sequence $b = \{b(i)\}i\geq 0$ denotes the output sequence from the GR when it is clocked regularly. The Period of GR is P_{GR}. S denotes the summation of clockings of GR in the period duration of CR, P_{CR}. Then S can be written as below:

$$S = \sum_{i=0}^{P_{CR}-1} a_i \tag{2.8}$$

Clock controlled generator works as following:

Let's say that generated sequence of CR is $a = \{2, 1, 3, 2, 2, 1, 3, 2....\}$. The output sequence of GR would be b={1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1}, if it was clocked regularly. But GR is not clocked regularly, it is clocked according to the value of a_i . The initial keystream is $z(0) = b(a0)$. After output z(t) is generated, the CR generates the nonnegative integer $a(t)$ and the GR is clocked $a(t)$ times. Then the next output $z(t+1)$ is produced. For our case, z becomes z={0, 1, 1, 0, 1, 0, 1, 0} which are the underlined bits

of sequence b={1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1}. According to this operation, keystream output z(t)e can be formulated as below:

$$z(t) = b\left(\sum_{i=0}^{t} a_i\right),\ for\ t \geq 0 \tag{2.9}$$

In [9], maximum period for a keystream sequence generated by a clock controlled system having parameters above is defined to be P, and expressed as:

$$P = \frac{P_{CR} M}{\gcd(S, M)} \tag{2.10}$$

In this formula, gcd stands for greatest common divider. For a clock control generaor to satisfy that period, there exists two conditions to hold. They are

i) Degree m of f(x) should be prime and S should not be a multiple of $\frac{M}{\gcd(M, q-1)}$.

ii) f(x) should be a primitive polynomial and $\gcd(S, P_{GR}) \leq q^{m/2}$.

If $f(x)$ is selected as a primitive function, then it can be said that $M = P_{GR} = q^m - 1$. If f(x) is over GF(2), then q=2. As a result $P_{GR}=2^m-1$, and the period of the system can be rewritten as:

$$P = \frac{P_{CR} P_{GR}}{\gcd(S, P_{GR})} \tag{2.11}$$

Linear Complexity of Clock Controlled Generator: A stop and go generator is a basic clock-controlled generator as shown in figure 2.4 and has same parameters mentioned in the period section above. The only difference is that CR generates a sequence

of binary valued $a = \{ai\}i \geq 0$. The linear complexity for that keystream generator is computed in [10] and stated as:

$$L = (2^{L_{CR}} - 1)\, L_{GR} \tag{2.12}$$

In this formula L_{CR} is used for the linear complexity of control register, and L_{GR} for the linear complexity of generator register.

2.2.3. Nonlinear Combination Generators

An alternative method for destroying the linearity inherence is to use several LFSRs in parallel. In this design, the keystream sequence is generated by manipulating the output sequences from multiple LFSRs by means of a nonlinear function f(x). General model for a nonlinear combination generator is shown in figure 2.5.

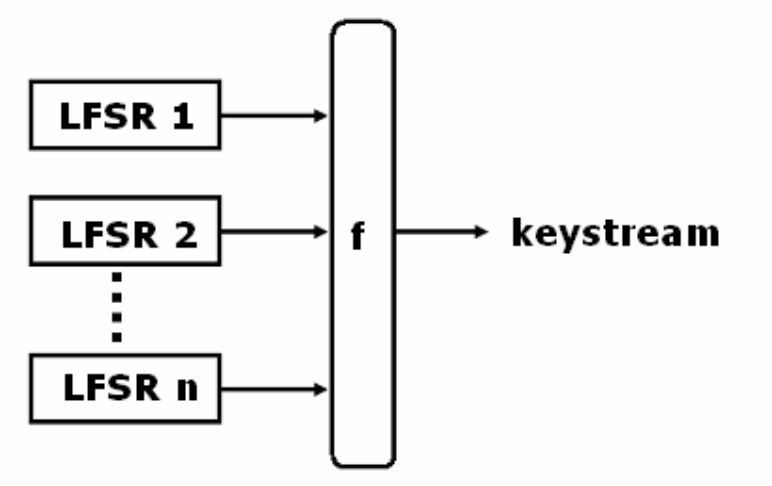

Figure 2.5. A general model for nonlinear combination generators

Period of Nonlinear Combination Generator: Let us assume that a nonlinear combination generator consisting of two LFSRs is designed. One of the LFSRs have length of r and the other one has length of s. According to [11], LFSRs are represented in terms of

the roots of their characteristic equations in a finite filed, and nonlinear operations inject additional roots into the representation.Both two LFSRs have irreducible characteristic polynomials, having degrees r and s. It is proven in [11] that for that design there is no danger of any of the roots of LFSRs vanishing, so the period of the nonlinear combination generator is the least common multiple of the periods of the two generators. If LFSRs are chosen maximum length LFSRs, then period of the keystream generator can be stated as:

$$P = \text{lcm}(2^{r-1}, 2^{s-1}) \tag{2.13}$$

In this formula lcm stands for least comman multiple. It is clearly seen that to maximize the period of a nonlinear combination generator , lengths of the LFSRs should be chosen relatively prime numbers, and also periods of LFSRs should be relatively prime. This is one of the design criterias of our nonlinear combination generator “MyBoun”.

Linear Complexity of Nonlinear Combination Generator: For the nonlinear combination generator described above and shown in figure 2.6 , in [11] its proven that the linear complexity of the resulting sequence is :

$$L = rs \tag{2.14}$$

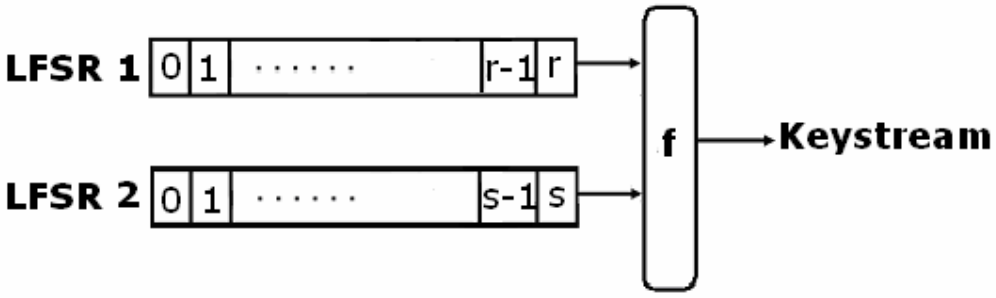

Figure 2.6. A nonlinear combination generator with two LFSRs

3. ANALYSIS OF AN EXISTING NONLINEAR COMBINATION GENERATOR

"GEFFE GENERATOR"

Geffe generator is the most populer nonlinear combination generator in the literature. It deserves to be analyzed in a detailed manner, since it highlights what should be and should not be done during the design of a nonlinear combination generator.

Geffe generator is proposed in [12]. As it is depicted in figure 3.1, the generator consists of three maximum length LFSRs whose lengths are L_1, L_2, L_3. The lengths of the registers are relatively prime. Nonlinear combining function can be written as:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1+ x_2)\, x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3 \qquad (3.1)$$

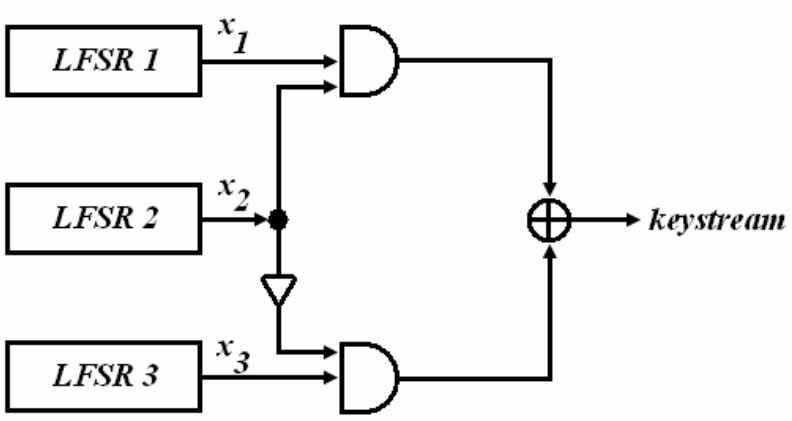

Figure 3.1. Geffe generator

The period of the Geffe generator is $P = (2^{L_1}-1)(2^{L_2}-1)(2^{L_3}-1)$ and linear complexity is $L=L_1L_2+L_2L_3+L_3$. [2]

Geffe generator has high period and moderately high linear complexity, but it can be said that it is cryptographically weak. The reason is that information about the states of the LFSR 1 and LFSR 3 leaks into the output sequence of the generator. Table 3.1 shows the

probable values of x_1, x_2, x_3 and output keystream z value regarding to them. Geffe generator does indeed produce a balanced result. If we assume that the input sequences are uncorrelated and each has an equal number of 1's and 0's, the output will also have and equal number of 1's and 0's. But we see that whatever the output value Z, the probability is 75% that input x_1 has that same value, and the same can be said for x_3. This is an input-to-output correlation or "information leak," and was eventually used to break through the combiner to find the state of the individual LFSR's. [13]

X_1	X_2	X_3	Z	X_1=Z	X_2=Z	X_3=Z
0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	0	1	0	1
0	1	1	0	1	0	0
1	0	0	0	0	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1
				---	---	---
				75%	50%	75%

Table 3.1. Some simple probability results of geffe generator

Correlation probabilities of the sequence $x_1$and sequence x_3 have the value of 0.75 and this makes the Geffe generator open to correlation attacks. We can generalize this fact for any nonlinear combination generator and observe how it causes the system to be weak against correlation attacks. Let's suppose that a nonlinear combination generator is employed by n maximum-length LFSRs R_1, R_2, ... , R_n of lengths L_1, L_2, ... L_n. If the feedback polynomials of the LFSRs and the combining function f are public knowledge, then the number of different keys of the generator is

$$\prod_{i=1}^{n}(2^{Li}-1) \tag{3.2}$$

This key consists of the initial states of each LFSRs. Assume that there is a correlation between the keystream and the output sequence of R_1, with correlation probabilty $p > 0.5$.(as in the case of Geffe Generator). If a sufficiently long segment of the keystream is known, then the initial state of the R_1 can be deduced by counting the number of coincidences between the keystream and all possible shifts of the output sequence of R1, until this number agrees with the correlation probability p. Under these conditions, finding the initial state of R1 takes at most $2^{L1} - 1$ trials. In the case where there is a correlation between the keystream and the output sequences of each of R_1, R_2, ..., Rn, the initial state of each LFSR can be determined independently in a total of about

$$\sum_{i=1}^{n} (2^{Li} - 1) \tag{3.3}$$

This number is far smaller than the number of different keys of the generator. [2] It proves us the fact that, the combining function f should be carefully selected so that there is no statistical dependence between any subset of the the output sequence of the LFSRs and the generated keystream sequence.

In [14], the method for construction of correlation immune nonlinear combination function is shown. The key point is that for a nonlinear combination generator with n LFSR, the combination function should be m^{th} order correlation immune and have balanced output to overcome the correlation attack problem Geffe has failed. The analysis shows that the combining function has maximum product terms of order n-m-1, where m is any integer value satisfying the condition $1 \leq m <$n-1.

4. MYBOUN – A NEW NONLINEAR COMBINATION GENERATOR

4.1. Overview

MyBoun is a nonlinear combination generater based binary additive stream cipher. As it is mentioned in 1.2.1.2, a binary additive stream cipher is a kind of synchronous stream cipher in which all the keystream, plaintext, and ciphertext digits are binary digits. The keystream sequence is bitwise XORed with plaintext sequence to produce the ciphertext sequence. The most important aspect for that nonlinear combination generator is its resistance against the class of algebraic attacks which is one of the most serious threat against security of stream ciphers.

In figure 4.1 general model of MyBoun is shown. It consists of 5 LFSRs named R_1, R_2, R_3, R_4, and R_5. The lengths of the registers are denoted as L_1, L_2, L_3, L_4, L_5 and have values of 89, 61, 31, 47, 59 respectively. They are chosen as relatively prime. Additionally, L_1, L_2 and L_3 are also mersenne prime. In other words, not only L_1, L_2 and L_3 are prime numbers but also $2^{L_1}-1$, $2^{L_2}-1$, $2^{L_3}-1$ are also prime numbers. P_1, P_2, P_3, P_4, and P_5 are primitive feedback polynomials of registers R_1, R_2, R_3, R_4 and R_5 respectively.

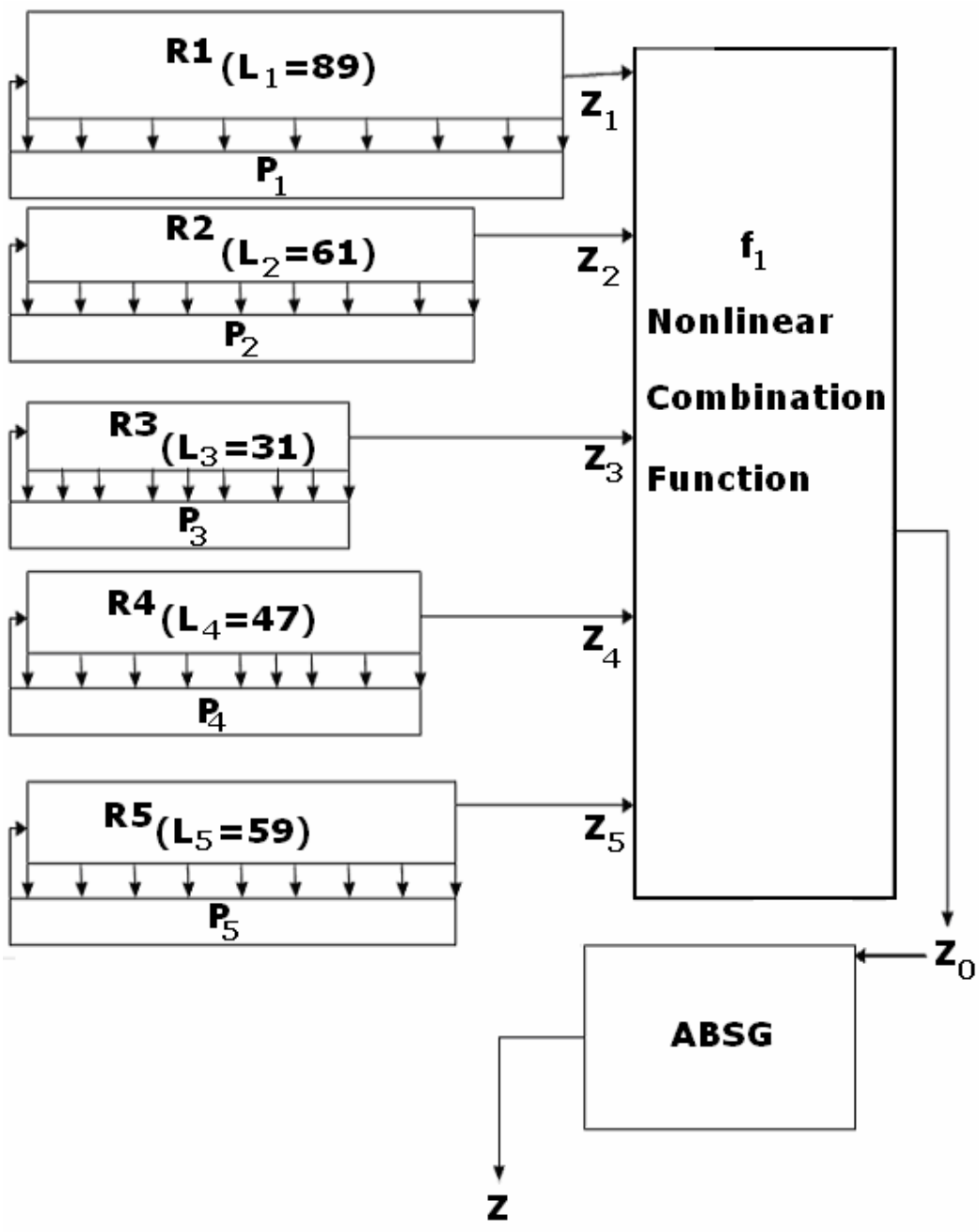

Figure 4.1. General model of MyBoun

The size of the inner state of MyBoun is 287 bits which is the total lengths of the registers. MyBoun is a nonlinear combination generator that can work with multivariable-length key. During software implementation of that combination generator a 128-bit length private key K, and a 128-bit length public initialization vector are used as input to produce keystream sequence. The output keystream is generated after the following processes:

I) Initialization

II) Production of Z_1, Z_2, Z_3, Z_4, Z_5 in figure 4.1 by maximum length LFSRs R1 R_2, R_3, R_4, and R_5 with primitive feedback polynomials P_1, P_2, P_3, P_4, and P_5.

III) Nonlinearization with nonlinear combination function f

IV) ABSG decimation mechanism

4.2. Initialization of MyBoun

MyBoun can work with multivariable-length key, but from point of resistance against attacks, a key with size smaller than 128 bit is not preferred. Since the inner size of MyBoun is 287 bit, any key length with size larger than 287 bit is not practical. During software implementation of that combination generator 128-bit secret key K and 128-bit initialization vector IV are used, and imitialization is performed as follows:

Firstly, 128 bit key vector K and 128 bit initialization vector IV are XORed. After this process, a 128-bit vector M is produced. Register R1 is initialized with first 89 bits of that vector M. The initialization of register R2 starts, where R1 stoped. From 90^{th} bit to the end bit of M initializes first 39 bits of R2, remaining 22 bit is started to be initialized again with 1st bit of M. The initialization of R3 starts from 23^{rd} bit of M and ends with 53^{rd} bit of M. Next 47 bit of M is used to initialize the register R4. The initialization of R5 starts from 101^{st} bit of M, next 28 bit is used to initialize first 28 bit of R5, and remaining 31 bit is initialized with first 31 bit of vector M.

After that initialization, each register generates a sequence of length 574 bit with their feedback polynomials. According to figure 4.1, we have 574 bit length Z_1, Z_2, Z_3,Z_4 and Z_5 sequences produced by the registers R_1, R_2, R_3, R_4 and R_5 with feedback polynomials P_1, P_2, P_3, P_4, and P_5 respectively. Z_1, Z_2, Z_3, Z_4, and Z_5 are used by nonlinear combination function f to produce 574 bit length Z_0 sequence . First 287 bits of that sequence is discarded. Next 287 bit of Z_0, say sequence S is used to initialize the registers. First 89 bits of sequence S initialize register R1, next coming 61 bits initialize register R2,

next 31 bits intialize R3, next 47 bits initialize R4 and and last 59 bits of sequence S initialize R5. The initialization step of MyBoun is completed after that process. Initial states of the register of MyBoun is made up of 287-bit sequence S.

4.3. Production of LFSR Outputs

After initialization step, each register of MyBoun generates output according to feddback polynomials. Primitive Feedback Polynomials of each register are P_1, P_2, P_3, P_4, and P_5. They are listed in table 4.1.

For $R_1 \rightarrow$ $P_1=X^{89}+X^{77}+X^{65}+X^{55}+X^{45}+X^{33}+X^{22}+X^{11}+1$;
For $R_2 \rightarrow$ $P_2=X^{61}+X^{53}+X^{45}+X^{38}+X^{30}+X^{23}+X^{15}+X^{7}+1$;
For $R_3 \rightarrow$ $P_3=X^{31}+X^{27}+X^{23}+X^{19}+X^{15}+X^{11}+X^{7}+X^{3}+1$;
For $R_4 \rightarrow$ $P_4=X^{47}+X^{41}+X^{35}+X^{29}+X^{23}+X^{17}+X^{11}+X^{5}+1$;
For $R_5 \rightarrow$ $P_5=X^{59}+X^{52}+X^{44}+X^{36}+X^{29}+X^{22}+X^{14}+X^{7}+1$;

Table 4.1. Primitive feedback polynomials of LFSRs of MyBoun

Feedback polynomials of each register are selected as primitive polynomials over GF(2), so as to make every register maximum length LFSR. As it was mentioned in the Properties of LFSRs section in this thesis, this helps us to maximize the overall period of the keystream generator. The coefficients of the polynomials are also selected in such a way that mutual seperation between their consecutive pairs is virtually the same. This property allows the implementations to be highly modular. [15]

In every clock cycle, last bits of the generators are produced as output and first bits of the registers are filled with the output value of the feedback function. In figure 4.2, feedback function bit locations of the generator registers of MyBoun is shown. These bits are XORed and the output of that operation becomes the first bit of register in the next clock cycle.

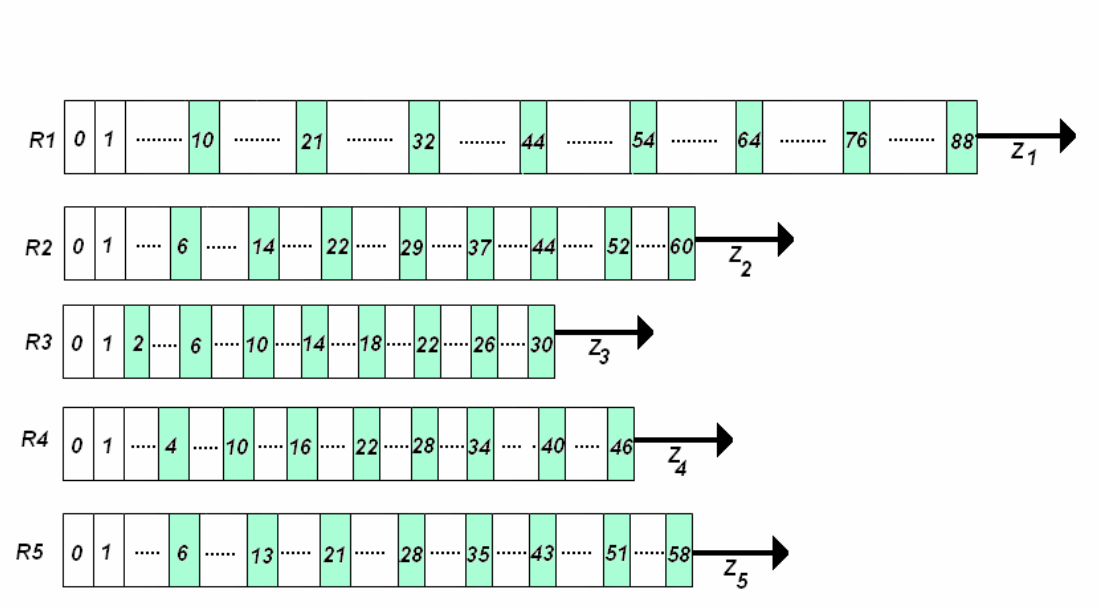

Figure 4.2. Feedback function bit locations of registers of MyBoun

4.4. Nonlinearization with Nonlinear Combination Function

Nonlinearization of MyBoun is managed by the nonlinear combination function f whic is stated below :

$$F(Z_1, Z_2, Z_3, Z_4, Z_5) = Z_1 \oplus Z_2 \oplus Z_4 \oplus Z_3.Z_5 \oplus Z_4 Z_5 \qquad (4.1)$$

This function is reached after applying the method of construction of correlation-immune functions described in [14]. The aim of MyBoun is to obtain a very simple, hardware oriented design beside being a secure one. Second order correlation immune a balanced output nonlinear combination function is obtained with minimum 5 variables, this is why MyBoun consists of 5 LFSRs. During observation of Geffe generator, we have showed that in that design there is an input to output correlation because of suffering from lack of correlation immunuty of the nonlinear combination generator function. In MyBoun nonlinear combination design this problem is not seen. Table 4.2 shows some simple probability results of MyBoun.

X_1	X_2	X_3	X_4	X_5	Z	X_1=Z	X_2=Z	X_3=Z	X_4=Z	X_5=Z
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	1	1	0	0
0	0	1	0	0	0	1	1	0	1	1
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	0	0	1	1	0	0	1
0	0	1	1	1	0	1	1	0	0	0
0	1	0	0	0	0	1	0	1	1	1
0	1	0	0	1	0	1	0	1	1	0
0	1	0	1	0	0	1	0	1	0	1
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	0	1	0	1	1	0	0
0	1	1	0	1	0	1	0	0	1	0
0	1	1	1	0	1	0	1	1	1	0
0	1	1	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	1	1	0
1	0	0	1	0	0	0	1	1	0	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	0	1	1	0	1	0	0
1	0	1	0	1	0	0	1	0	1	0
1	0	1	1	0	1	1	0	1	1	0
1	0	1	1	1	1	1	0	1	1	1
1	1	0	0	0	1	1	1	0	0	0
1	1	0	0	1	1	1	1	0	0	1
1	1	0	1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	1	1
1	1	1	0	1	1	1	1	1	0	1
1	1	1	1	0	0	0	0	0	0	1
1	1	1	1	1	0	0	0	0	0	0
						---	---	---	---	---
						50%	50%	50%	50%	50%

Table 4.2. Some simple probability results of MyBoun

4.5. ABSG Decimation Mechanism

The usual way to design a hardware-oriented 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, and having the clocking of some LFSRs controlled by a combination of (possibly other) LFSRs. One drawback of these techniques is that they can result in a cipher that can be too complicated or slow for the fast synchronous encryption in cheap hardware stream ciphers are usually dedicated to.[16] In MyBoun, instead of increasing the complexity of the nonlinear combination function, ABSG (Alternative Bit-Search Generator) is used to decimate the output of the nonlinear combination function in an irregular way. ABSG mechanism was first presented at the ECRYPT Workshop State of the art of the stream ciphers [17].

The keystream sequence Z is generated from the binary sequence Z_0 through the ABSG decimation algorithm. The sequence Z_0 is split into subsequences of the form $(\bar{b},b^i,\bar{b})$, with $i \geq 0$ and $b \in \{0,1\}$; $\bar{b}$ denotes the complement of 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. [18] Let Z_0=010100111010010001010 10 be the input bit sequence to ABSG. Then, the action of the ABSG on Z_0 can be described by as in the figure 4.3 and for that example the output of ABSG, sequence Z becomes to be 10111001.

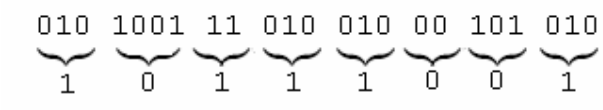

Figure 4.3. An example for ABSG decimation

Algebraic attacks are one of the most important problems of LFSR based stream ciphers that are clocked in a simple and easily predictable way. Adding ABSG to design desynchronizes the output bits of the cipher from the clock of the LFSR, and this helps to resist algebraic attacks. In the security of MyBoun section, how much improvement it adds for resisting the attacks will be analyzed.

ABSG has the advantage over well-known pseudorandom generators the Shrinking generator [19] and the Self Shrinking generator [20] that are also using methods of irregular decimation. Table 4.3 shows a comparison between the ABSG and some well known generators.

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

Table 4.3. Comparison between the ABSG and some well known generators

An output bit is produced after 2 input bits with probability 1/2, after 3 input bits with probability 1/4 etc. In general, an output bit is produced after i + 1 input bits with probability $2-i$ so the average number of input bits needed to produce one output bit is $\sum_{i=0}^{\infty}(i+1).2^{-i}=3$. This shows that the rate of the BSG is asymptotically 1/3. [21] Producing n bits of the output sequence requires, on everage, 3n bits of the input sequence. [16]

After the ABSG decimation, the final keystream sequence Z is reached. This keystream sequence is bitwise XORed with plaintext sequence, and ciphertext sequence is produced.

5. PROPERTIES OF MYBOUN

In this subsection, period, linear complexity and statistical properties of MyBoun are analyzed.

5.1. Period Of MyBoun

Period analysis of MyBoun will be depicted in two steps. Firstly, period of the sequence up to ABSG decimation mechanism which is shown in figure 5.1, next the period of the sequence after ABSG decimation mechanism will be analyzed. The last one will give us the total period of MyBoun.

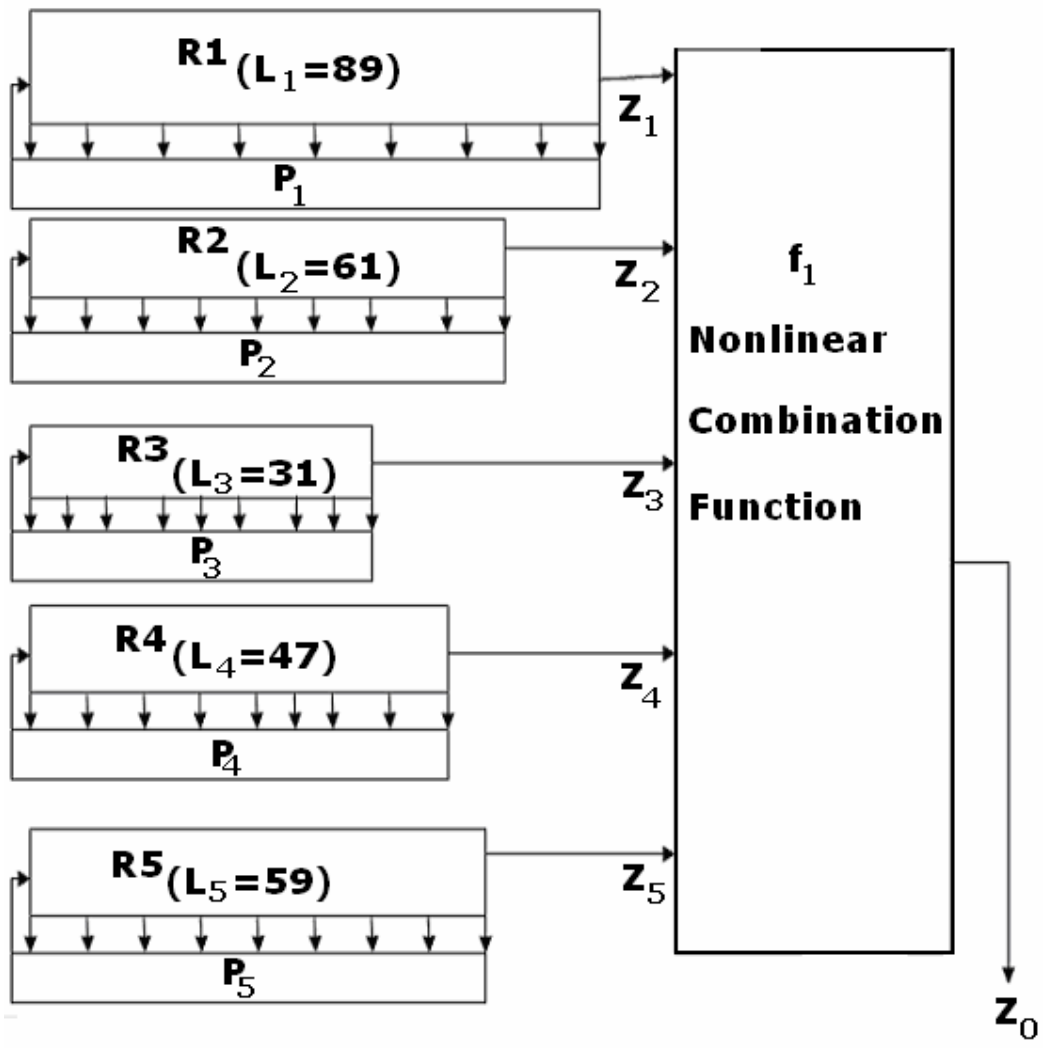

Figure 5.1. First part of MyBoun for period analysis

All registers of MyBoun have primitive feedback polynomials and they are maximum length LFSRs. Hence, periods of R_1, R_2, R_3, R_4, R_5 are 2^{89}-1, 2^{61}-1, 2^{31}-1, 2^{47}-1, 2^{59}-1 respectively. According to discussions we have stated in subsection 2.2.3 period of nonlinear combination generators session, total period of a keystream generator is least common multiple of periods of each maximum length LFSRs. Hence, MyBoun‘s first part seen in figure 5.1 has period

$$T = (2^{89}-1).(2^{61}-1).(2^{31}-1).(2^{47}-1).(2^{59}-1). \quad (5.1)$$

For the second part of the MyBoun shown in figure 5.2, input is sequence Z_0, which was produced by the first part shown in figure 5.1. $Z_0 = (s_0, s_1, \ldots, s_n)$ is an infinite T-periodic sequence , and $s_i = s_{i+T}$ for all $s \geq 0$. The output sequence of ABSG for input sequence Z_0 is denoted ABSG(Z_0). For period analysis of second part of MyBoun design, the notion of state in relation with the output state of the ABSG will be presented next.

Figure 5.2. Second part of MyBoun for period analysis

The implementation of ABSG can be thought as a moving window along the input sequence. After reading the bit inside the window, there are three possible states for the window:

1) If the lastly read bit is the one that was looked for, then the window gets into the empty state. (state φ) In the next window it will not be searching for any exact bit value.

2) If the lastly read bit is 1 while 0 was looked for, or it is 0 while any exact bit was not looked for, then the window gets into “0” state. In the next window it will be searching for the bit 0.

3) If the lastly read bit is 0 while 1 was looked for, or it is 1 while any exact bit was not looked for, then the window gets into “0” state. In the next window it will be searching for the bit 0.

In the ABSG algorithm, the initial window state is state φ. Every time the state of the window returns to φ, one output is generated. The state of the window is updated after reading the next bit in the sequence. Table 5.1 shows how the state is changed when bit 0 or bit 1 is read.

S	0	1
φ	0	1
0	φ	0
1	1	φ

Table 5.1. The state of the window transition according to the bit read

Set of states S={φ ,0,1} is denoted by S and identity, transpositions (φ ,0) and (1,φ) are denoted as Id,t_0, t_1 respectively. The output state of the ABSG Algorithm for (s_0,s_1,….,s_k) is defined as below:

$$OS(s_0,\ldots.,s_k)=t_{sk}\ o\ t_{sk-1}\ o\ldots..ot_0(\varphi) \tag{5.2}$$

Let Z_0 = (s_0,s_1,….,s_n) be an infinite T-periodic sequence of bits, as this is the case for MyBoun, then ABSG(Z_0) is also periodic. The output rate for ABSG is1/3. For the sequence Z_0 = (s_0,s_1,….,s_n), the length of the output sequence will be approximately n/3. If OS(s_0,….,s_{T-1})=φ, then the period of the output sequence will be T/3. If k is assumed to be the smallest integer value satisfying the condition OS(s_0,….,$s_{k(T-1)}$)=φ, then the period of the output sequence will be approximately kT/3. In [17], it is experimentally proved that integer value k can only take the values 1 or 2 and the period of the output sequence is T/3 or 2T/3 respectively, where T= (2^{89}-1).(2^{61}-1).(2^{31}-1).(2^{47}-1).(2^{59}-1).

One of the most important properties of a good output stream generator is high period. MyBoun perfectly satisfies that condition.

5.2. Linear Complexity Of MyBoun

Linear complexity analysis of MyBoun will also be depicted in two steps. Firstly, linear complexity of the sequence up to ABSG decimation mechanism which is shown in figure 5.1, next the linear complexity of the sequence after ABSG decimation mechanism will be analyzed. The last one will give us the total linear complexity of MyBoun.

According to [11], Galois field theory enables one to predict the generator complexity resulting from nonlinear operations. Analysis shows that linear complexity of MyBoun's first part is:

$$LC= L_1+L_2+L_4+ (L_3\ L_5) + (L_4\ L_5) = 89+61+47+31.59+47.59=4799 \qquad (5.3)$$

Experimental results declared in [17] show that the linear complexity is always equal to the period for maximum length LFSRs of degree $L \leq 16$, independent of linear complexity of LFSR. Based on those experimental results, MyBoun is said to have a linear complexity of T/3 for sequences reaching empty state with T input bits, or 2T/3 for sequences reaching empty state with 2T input bits, where $T= (2^{89}-1).(2^{61}-1).(2^{31}-1).(2^{47}-1).(2^{59}-1)=2.4866*10^{26} \cong 2^{287}$.

5.3. Statistical Properties of MyBoun

In order to verify that the keystream sequences generated by MyBoun have statistical properties same as the ones that are random, the statistical tests FIPS 140-2, and NIST statistical test suite are applied. For FIPS 140-2 statistical tests, key sequences of 20000 bits are produced. Tests are applied for all those created sequences. If test results are within the required range mentioned in 2.1.3.1, then these ones are said to pass, otherwise they are noted as fail. It is observed that all generated keystream sequences pass FIPS 140-2 test results. In table 5.2 obtained results for one of the keystream sequences that has passed the FIPS 140-2 tests are shown.

Monobit Test = 10031

Poker Test = 11.001600

Runs Test

Length of Runs Of 1s	**Value**	*Length of Runs Of 0s*	**Value**
1	2448	1	2462
2	1251	2	1254
3	639	3	632
4	294	4	304
5	158	5	150
6 +	163	6 +	163

Long Runs Test = 15

Table 5.2. FIPS140-2 Test results of a keystream produced by MyBoun

For the NIST statistical tests, key sequences of 10^6 bits are produced. The length of the sequences are chosen to be 10^6 bits, because for Overlapping Template Mathing test, Lempel-Ziv Compression test, Linear Complexity test, Random Excursions test in this test suit require minimum of 10^6 bits of sequence as input so as to be interpretted as those tests are passed. The generated sequences pass NIST statistical tests, if they have the characteristic of randomness with a significance level $\alpha=0.01$. All generated keystream sequences of MyBoun has passed those statistical and can be said to have certain randomness characteristics.

5.4. Computational Efficiency of MyBoun

MyBoun has a hardware oriented design. The number of gates in the proposed hardware implementation of MyBoun can be estimated as follows. This estimation uses the

number of gates for elementary components given in [22], i.e. 2.5 gates for an XOR, 1.5 gates for an AND, 5 gates for a MUX, and 12 gates for a flip-flop. MyBoun totally includes 3932 gates.

- **Initialization :** 320 gates corresponding to **:**
  - 128 XORs

- **LFSRs :** 3532 gates corresponding to :
  - 287 Flip-flops
  - 35 XORs

- **Combination Function :**13 gates corresponding to :
  - 4 XORs
  - 2 ANDs

- **ABSG :** 67 gates corresponding to :
  - 4 Flip-flops
  - 2 MUX
  - 3 XORs
  - 1 AND .

6. SECURITY OF MYBOUN

In keystream generator design, resistance against different attacks is the most crucial point that must considered. Therefore, in this subsection, some attacks and the behaviour of MyBoun against these attacks are described. These attacks are type of known plaintext attacks. They are conducted under the assumption that the whole inner internal structure of the keystream generator is known by the cryptanalysts. We have stated that the most important feature of MyBoun design is its resistance against algebraic attacks. In algebraic attacks section this feature of MyBoun will be proved and emphasized.

6.1. Brute Force Attack

This attack is based on the exhaustive search of possible keys. For each key candidate the attacker runs the generator to generate output bits of size inner state size of the generator.and compares the result with the known output stream. If they differ in at least one bit, the guess is discarded as being wrong. Otherwise, the guess is added to the set of possible key candidates. If there exists more than one possible key candidate, the correct one is determined by running the generator to decrypt all the message. [23]

My Boun has key K with a length of 128 bit. Therefore, the number of possible keys for MyBoun is 2^{128}. Therefore, such an attack appears impractical for MyBoun.

6.2. Time Memory Trade off Attack

A general technique for breaking arbitrary block ciphers with N possible keys, with memory M, and in time T was first introduced by Helman with the trade off curve $TM^2=N^2$ for $1 \le T \le N$.[24] Stream ciphers have a very different behavior with respect to time/memory tradeoff attacks. The size N of the search space is determined by the number of internal states of the bit generator, which can be different from the number of keys. The realtime data typically consists of the first D pseudorandom bits produced by the generator,

which are computed by XOR'ing a known plaintext header and the corresponding ciphertext bits.[25]

The simplest time/memory tradeoff attack on stream ciphers was independently described by Babbage [26] and Golic [27]. According to that attack it was stated that $T=D=N/M$ and $P = M= N/D$, where T denotes the time required for the actual attack stage, D denotes the amount of observed key stream, N denotes the size of solution space that is determined by the number of internal states of the key generator,M denotes the required amount of memory, and P denotes the time for the preprocessing phase of the attack. In [25], a combination of the two approaches an improved time-memory trade-off attack is presented by A. Biryukov and A. Shamir. This improved time/memory/data tradeoff for stream ciphers has the form $TM^2D^2 = N^2$ for any $D^2 \leq T \leq N$.

Time memory trade off attacks are practical for key stream generators, which have small size of solution space. The size of the solution space is determined by the number of internal states of the key generator. Small number of internal states concludes to small number of solution space. MyBoun has a solution space of 2^{287}, it is large enough to make those kind of attacks impractical. For the improved time memory trade off, the trade off formula is $TM^2D^2 = N^2$. If M *is* chosen a high value such as 2^{40} , and N is known to be 2^{287}, the formula becomes $T\,D^2=2^{494}$ for any $D^2 \leq T \leq N$. Such a high value shows that MyBoun is secure against time memory trade off attacks.

6.3. Correlation Attacks

The correlation attack was first introduced by Siegenthaler in [14] . In some cases there may be some correlation between input variables to the combination function and outputs from the combination function. In the simplest case, correlation means that the output is equal to one of the input variables with a probability not equal to 1/2. [3] In section 3, we have proved that the number of keys to be searched decreases from

$\prod_{i=1}^{n}(2^{Li}-1)$ to $\sum_{i=1}^{n}(2^{Li}-1)$ for a keystream generator when a correlation exists between input and output of combination generator.

MyBoun does not succumb to correlation attacks by the help its 2nd order correlation immiune combination function. Table 4.2 shows that the output of nonlinear combination function output is equal to each one of the input variables with a probability of 1/2. Additionally, in MyBoun design, the output of the nonlinear combination is not directly used as keystream. The output of the nonlinear combination function is also decimated with ABSG component with a throughputrate of 1/3. This feature also increases the resistance of the design against correlation attacks. Consequently, MyBoun is secure against correlation attacks.

6.4. Algebraic Attacks

Algebraic attacks are the most important threads for LFSR-based keystream generators in recent years. The attack is based on two steps. In first step, a system of equations are set up according to known ciphertext bits, in those equations the bits of the secret K are formulated as the unknown variables. In the second step those equations are solved and the Key is revealed. For nonlinear combination generators, the equations that are set up are nonlinear equations, and the unknown variables are the initial state bits of the registers. To solve those nonlinear equations some techniques called linearization, or XL (eXtension and Linearization) are used. Keystream generators that have low order of combining function, or have regular clocking mechanism are weak against algebraic attacks.

The Linearization Technique : The attacker sets up a system of nonlinear equations, where each equation describes the dependency between one output bit and the corresponding register bits. In linearization technique, each nonlinear monomial is replaced by a dummy variable. This ends up with a system of linear equations which can be solved by known methods such as Gaussian eliminiation algorithm. Finally the dummy variables

are replaced with the original monomials . And then a unique solutions for the unknowns are tried to be find.

Extension and Linearization Technique : Extension and Linearization technique is an improvement of linearization technique presented by Shamir, Patarin, Courtois, and Klimov. [28] In this method the attacker again sets up a system of nonlinear equations, describing the dependency between one output bit and the corresponding register bits. But this time, the attacker constructs additional equations by multiplying the existing ones with monomials of small degree. If the degree of the resulting equation is not greater than a specified threshold, the equation is added to the system of equations. In this way, a nonlinear system with a lot of redundant information is generated. In a second step, the extended system of equations is linearised as described in the linearization technique section above. This time, however, the attacker has a lot more equations at his disposal, reducing the number of output bits required. Still, for the attack to work, the original system of equations has to be over-specified.[23]

MyBoun has a second order combining function which is really a low degree. It would be very weak against algebraic attacks, if the output of combination generator was used as the keystream sequence. This will be proved in the next section. In order to strenghten the design, a more complex, a high degree nonlinear combination function with more registers as input parameters could be chosen. But the intent of MyBoun was to be a simple, paractical, hardware oriented keystrem generator with high period, high linear complexity, good statistical properties, and high resistance against attacks. This is why instead of increasing the complexity of combination function , using ABSG is prefered. As mentioned in [16], algebraic attacks on stream ciphers applly (at least theoretically) to all LFSR based stream ciphers that are clocked in a simple and/or easily predictable way. One interesting approach to help resist such attacks is to add a component that de-synchronizes the output bits of the cipher from the clock of the LFSR. This component in MyBoun design is ABSG. In the following three subsections , three algebraic attacks will be applied to the output sequence of the nonlinear combination generator in order to show that how successful algebraic attacks are against regularly clocked keystream generators with low order combination functions.

6.4.1. Algebraic Attacks on MyBoun without ABSG Component

Ciphertext Only Attack by Using Linearization Technique: Total number of monomial of degree smaller or equal to 2 is T=89+61+47+31.59+47.59=4783 which can also defined to be the linear complexity of the keystream output of MyBoun design without ABSG decimation component. Then we suppose that an adversary has knowledge of T keystream bits such that corresponding equations obtained for each cycle is linearly independent. Therefore we have T equations with T variables which can be solved using Gaussian elimination. We have applied linearization and rename all the quadratic terms with new first degree terms. Then total complexity of solving TxT matrix will be about $T^3 = 2^{36.67}$ which is much lower than trying all possible initial states which is $S=\mathbf{2^{287}}$. Therefore this fact shows us that without ABSG decimation component 2nd order Boolean function is weak against algebraic attacks based on ciphertext only attacks and the regularity of clocking mechanism helps the attackers to solve equations easily.

Known Plaintext Attack by Using Linearization Technique: The number of keystream output bits can be decreased, if the attacker assumed to have known some plaintext as in the case of known plaintext attacks. When the nonlinear combination function of MyBoun is observed, it is clearly seen that if the attacker guesses the internal state of R_5, then the degree of the Boolean function will be decreased. Therefore, a linear equation will be obtained in terms of the other registers' internal states. Since MyBoun is a synchronous binary additive keystream generator, the attacker has the equation $X_1+X_2+X_4+ (X_3\ X_5) + (X_4\ X_5) + z =0$ to be solved for each clock where z represents the ciphertext. Therefore the number of total monomials will be 89+61+31+47=228. If we suppose that the attacker has T=228 keystream output bits such that corresponding equations are all linearly independent. Then complexity of solving the system is $T^3=2^{23.5}$. And the total complexity of attack will be $2^{59+23.5}=2^{82.5}$ which is again much lower than brute force attack complexity $2^{287}$with only knowledge of 228 keystream bits. The need of knowledge of 228 keystream bits also shows us that linear complexity of the system is reduced to 114 bits when applying Berlekamp-Massey attack.

Ciphertext Only Attack by Using XL (Extension and Linearization) Technique : XL algorithm is designed to solve multivariate systems using less keystream outputs. As described in a detailed manner in [29], the XL algorithm consists of multiplying the initial m equations by all possible monomials of degree up to D − d, so that the total degree of resulting equations is D. D is accepted to be an input parameter of XL algorithm and *d* is accepted to be the degree of the equations R is defined to be the number of equations generated in XL, and T is defined to be the number of all monomials. [29]. R and T can be formulated as below , where *m* is the number of keystream output bits required:

$$R = m.(\sum_{i=0}^{D-d} \binom{n}{i}) \tag{6.1}$$

$$T = \sum_{i=0}^{D} \binom{n}{i} \tag{6.2}$$

In this attack, we suppose that adversary is very lucky and all the equations he had generated are linearly independent. Under these circumstances, in order to XL algorithm to succeed the condition of $R \geq T$ should be satisfied. [29] In other words, the following equation should be satisfied:

$$m.(\sum_{i=0}^{D-d} \binom{n}{i}) \geq \sum_{i=0}^{D} \binom{n}{i} \tag{6.3}$$

For MyBoun design without ABSG mechanism, if D=5 and n=287, then *m* will be calculated as 4048. Without guessing any initial states of the registers, the attacker will be able to obtain 4048 equations, if he has the knowledge of about T=4048 keystream bits. Then the constructed system of TxT will be solved with complexity $T^3=2^{35.95}$. It is clearly seen that with ciphertext only attack, the number of required keystream bits are reduced by using XL algorithm if it is compared with the one found by using the linearization technique.

Consequently, all three algebraic attacks applied to MyBoun design without ABSG component described above are successful. But ABSG adds an irregular decimation to MyBoun that makes the system stronger against algebraic attacks. If any equation can be found that describes the output of the ABSG mechanism in terms of the initial states of the registers, algebraic attacks could also be applied to MyBoun. But it is not found. Although such a relation is found, it would have a high degree, and will make those attacks impractical.

6.5. An Attack Against ABSG

In [30] , the most successful attack in the literature against a system of maximum length LFSR and ABSG is proposed. This attack is a type of known plaintext attack. Firstly, this attack will be described. Next, this attack will be modified for MyBoun.

According to the proposed model, the sytem is made up of two components. First component is a maximum-length LFSR with length L, and the second component is MBSG. But it is mentioned that the attack can also be applied to ABSG. For simplicity, it will be assumed to be ABSG in this section. Maximum length LFSR produces sequence s, and that sequence is input of ABSG. The output of ABSG is used as the keystream sequence. The attack consists of two steps. First step is constructing a lookup table and second step is the lookup stage.

In first step, a lookup table is constructed. The size of that table, determines the required memory for that attack. In the look up table, possible internal states of the LFSR and the corresponding output bits of the ABSG are stored. If N represents the solution space of the LFSR, in other words the possible internal states of the LFSR, then for a maximum length LFSR it can be said to be equal to 2^L-1. First of all, a random sample of *m* different 2L-bit blocks are chosen from the solution space N of that LFSR and stored as $Y_1, Y_2, Y_3 ..... Y_m$. Then for each Y_i, the sytem produces input sequences s_i to ABSG . And lastly, ABSG produces output sequences z_i. The length of the sequences z_i is choosen 2L in this attack. In the lookup table, only sequences Y_i and z_i are stored as a pair shown in figure 6.1 , sequence s_i is not stored to save memory.

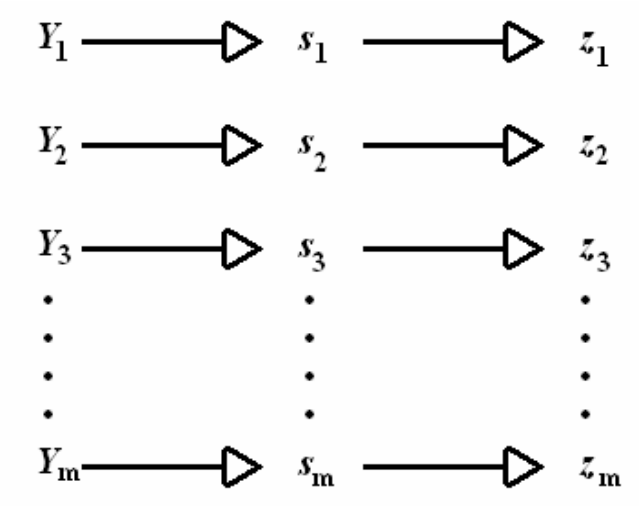

Figure 6.1. Constructed look up table

In the second stage, the attacker assumed to know D successive 2L bit blocks. These belocks are called as sequence 1, sequence 2, ... sequence D. Each sequence is compared with the sequences z_i stored in the look up table. If any match is found such as sequence z_k, corresponding Y_k producing that sequence is found in the table. From the found state state *Yk*, LFSR output sequence *sk* and then the ABSG output can be regenerated. The produced ABSG output is the *t-1* shifted version of the original one. So one can get the unknown future part of the ABSG output . The complexity of the attack is computed as follows: Firstly, randomly chosen *m* candidate initial states of LFSRs are stored in table. If *m* is selected $2^{2L/3}$ as a sensible point on trade-off curve, required memory in the time memory trade off formula becomes $M = 2^{2L/3}$ words. According to time/memory tradeoff attack proposed by Babbage [26] , $MT = N$, and $D \geq T$. D, minimum available number of known output blocks, is computed 2L/3. Since each block has a length of 2*L* bits total required amount of bits become $O(2L+2L/3)$, and time complexity becomes $O(2L/3)$, respectively. The success rate of attack is evaluated by computing how many states out of the solution space, *N*, is captured with usage of a lookup table which contains $2^{2L/3}$ randomly chosen states and $2L+2^{L/3}$ known output bits. The success rate for ABSG is found to be %65 in experiments. [30]

This attack is not practical for MyBoun, since initial states of the generator consists of 287 bits. Solution space N is 2^{287} -1 . For the look up table, *m* pairs of input and output sequences of the generator are stored. İt is impossible to store $2^{2L/3}$ bits of word. MyBoun has the advantage of having a large initial state.

7. CONCLUSIONS

In this thesis, general information about cryptosytems, their widely used components LFSRs, required properties of LFSR based keystream generators are described as theoretical background. In the light of described information, a new nonlinear combination generator MyBoun is presented. The main property of MyBoun is resistance against algebraic attacks which is provided by ABSG component. MyBoun has all properties of a required keystream generator. It has large period, high linear complexity, good statistical results, high throughput rate, and the characteristic of randomness. The periodicity, the linear complexity and the throughput rate are analyzed and computed according to the analysis. All computed values regarding to those properties are satisfactory for a required keystream generator. Generated sequences of MyBoun are tested by FIPS 140-2 and NIST statistical test suites to observe that whether the sequences have the chracteristic of random sequences or not. The results are satisfactory. The keystream sequences generated by MyBoun passes the FIPS 140-2 statistical tests and passes the NIST tests with a significance level of α=0.01 which means that 1 % of the tested sequences are allowed to fail.

Furthermore, MyBoun is a very simple hardware oriented, a modular keystream generator. It is valid for applications requiring high speed encryption. Security of MyBoun is analyzed with respect to some known attacks. Beside brute force attack, time memory trade off attack, correlation attack ; a detailed analysis for algebraic attack is performed for MyBoun. It has been shown that MyBoun has resistance against all those attacks, especially for algebraic attacks.

Finally, we can say that MyBoun can be used for all applications which need a keystream generator that is secure against attacks, that is easy for hardware implementations, that has characteristic of randomness, good statistical properties, high throughput rate, high period and high linear complexity.

REFERENCES

1. Schneier, B., "Applied Cryptography ", *Protocols, Algorithms, and Source Code in C*, JohSn Wiley & Sons, New York, 1996.

2. Menezes, A., Oorschot, P.van, Vanstone, S., *Handbook of Applied Cryptography*, CRC Press, 1996.

3. Ekdahl, P., *On LFSR based Stream Ciphers Analysis and Design*, Ph.D. Thesis, November 21, 2003

4. Zeng, K., Yang, C.H., Wei, D.T., Rao, T.R.N., "Pseudorandom Bit Generators in Stream-Cipher Cryptography" , *IEEE Computer Magazine*, Vol. 24, No. 2, Feb. 1991, pp. 8-17.

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

6. FIPS PUB 140-1 Security Requirements for Cryptographic Modules http://csrc.nist.gov/cryptval/140-1.htm

7. .A Statistical Test Suite For Random and Pseudorandom Number Generators for Cryptographic Applıcations, http://csrc.nist.gov/rng/

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

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

11. Key, E.L "An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators", *IEEE Transactions On Information Theory*, Vol. IT-22, N0.6, November 1976

12. Geffe, P. 1973. "How to protect data with ciphers that are really hard to break" *Electronics*, vol. 46, pp. 99--101, January. 1973.

13. Ritter, T. "The Story of Combiner Correlation: A Literature Survey", http://www.ciphersbyritter.com

14. Siegenthaler, T "Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications", *IEEE Transactions on Information Theory* . 1984 IT-30 :776-780.

15. Rajski. J., Tyszer, J., "Primitive Polynomials Over GF(2) of Degree up to 660 with Uniformly Distributed Coefficients" *Journal of Electronic Testing : Theory and Applications* 19,645-657 , 2003

16. Gouget, A., Sibert, H., Berbain, C., Courtois, N., Debraize, B., Mitchell, C., " Analysis of the Bit-Search Generator and Sequence Compression Techniques" *Fast Software Encryption – FSE 2005*,

17. Gouget, A., Sibert, H., "The Bit-Search Generator*", State of the Art of Stream Ciphers Workshop Brugge,* Belgium, October 14-15 2004.

18. Wu, H., Preneel, B. "Cryptanalysis of Stream Cipher DECIM" http://cr.yp.to/streamciphers/decim/049.pdf

19. Coppersmith, D., Krawczyk, H. , Mansour, Y., "The Shrinking Generator, advances in Cryptology" – *CRYPTO'93 Proceedings*, LNCS 773, Springer-Verlag, D. R. Stinson, ed., (1993), 22–39.

20. Meier, W., Staffelbach, O., "The Self-Shrinking Generator, Advances in Cryptology" – *EUROCRYPT'94 Proceedings*, LNCS 950, Springer-Verlag, A. De-Santis, ed., (1994), 205–214.

21. Hell, M., Johansson, T., "Some Attacks on the Bit-Search Generator" *Fast Software Encryption: 12th International Workshop*, FSE 2005, Paris, France, February 21-23, 2005, 215-227

22. Batina, L., Lano, J., Ors, S.B., Preneel, B., and Verbauwhede, I., "Energy, perfomance, area versus security trade-offs for stream ciphers." *In The State of the Art of Stream Ciphers:Workshop Record*, pages 302–310, Brugge, Belgium, October 2004.

23. Zenner, E., *On Cryptographic Properties of LFSR-based Pseudorandom Generators*, PhD Thesis, Mannheim, 2004

24. Hellman, M. E. "A Cryptanalytic Time-Memory Trade-Off" , *IEEE Transactions on Information Theory*, Vol. IT-26, N 4, pp.401–406, July 1980.

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

26. Babbage, S., "A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers", *European Convention on Security and Detection, IEEE Conference Publication No. 408*, May 1995.

27. Golic, J. “Cryptanalysis of Alleged A5 Stream Cipher”, *Proceedings of Eurocrypt’ 97*, LNCS 1233, pp. 239–255, Springer-Verlag 1997.

28. Shamir, A., Patarin, J., Courtois, N., Klimov,A., “Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations”, *Eurocrypt’2000*, LNCS 1807, Springer, pp. 392-407.

29. Courtois, N.T., “Higher Order Correlation attacks, XL Algorithm and Cryptanalysis of Toyocrypt” *Information Security and Cryptology - ICISC 2002*: 5th International Conference, Seoul, Korea, November 28-29, 2002.

30. Altug, Y., Ayerden, N.P., *On Periodicity Analysis Of BSG (Bit Search Generator) Stream Cipher And Time-Memory Trade-Off Attacks On Its Variants*, Bachelor of Science Thesis , Bogaziçi University, June 2006