A DYNAMICALLY CLOCK-CONTROLLED BINARY STREAM CIPHER WITH MEMORY AND DYNAMIC MULTIPLEXING 'THE MONO STREAM CIPHER'

by

Orçun Uzun

B.S., Electrical-Electronics Engineering, Çukurova 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

To my family
You are my hero
Thank you for all !

ACKNOWLEDGEMENTS

Firstly, I would like to thank to my thesis supervisor Prof. Dr. Emin Anarım for his kind interest, technical advices and his patience. I'm indebted to him for his tolerance and guidance.

Additionally, I would like to thank to Frederik Armknecht from NEC Europe Ltd. and Nicolas Courtois from CP8 Crypto Lab, SchlumbergerSema for their kind interest to my questions and for discussions on algebraic attacks on stream ciphers.

The last but not least I would like to thank to my father Mustafa Uzun, my mother Nuran Uzun and my brother Onur Uzun for their support during my whole life. They have also shared the name of MONO stream cipher with me. MONO consists of first letters of the names of my family members.

ABSTRACT

A DYNAMICALLY CLOCK-CONTROLLED BINARY STREAM CIPHER WITH MEMORY AND DYNAMIC MULTIPLEXING 'THE MONO STREAM CIPHER'

Streams ciphers are appropriate for high speed encrypted communications because this type of ciphers provide high output speed and ease of implementation in hardware. Generally, stream ciphers makes use of Linear Feedback Shift Registers (LFSRs) due to its simplicity and its ease of realization both in software and hardware. However, this efficient component is not sufficient when we consider security. The designer should use many nonlinear functions and mechanisms to make the system more resistant against cryptanalysis. A stream cipher should have high period, high linear complexity, good statistical properties and be resistant against most recent successful attacks such as algebraic attacks, correlation attacks, time memory tradeoff attacks, and divide and conquer attacks.

In this thesis, a new stream cipher design is proposed. MONO is designed to be resistant against algebraic and correlation attacks. In the design phase, the objective was to design a stream cipher with good randomness, high period and linear complexity and resistance against many attacks. The other objective was to design a realizable stream cipher. Also the innovation in this thesis is the proposal of a dynamically clock controlled filter generator with the use of memory and dynamic multiplexing. The initialization step of MONO is based on solving a mathematical problem which is known to be difficult, to deduce the secret key. It is showed that MONO stream cipher satisfies high period and linear complexity. Also, system has good statistical properties. We have made several statistical tests to see whether MONO cipher satisfies basic requirements for random number generation. MONO passed all of the tests. MONO is secure against many important attacks such as correlation attacks, algebraic attacks, divide and conquer attacks, and time memory trade off attacks. The hardware implementation of MONO has also been investigated and we had observed appropriate results. Consequently, we can say that MONO is appropriate for both hardware and software applications.

ÖZET

BELLEKLİ VE DİNAMİK ÇOĞULLAMALI DİNAMİK SAAT DENETİMLİ İKİLİ DİZİ TİP ŞİFRELEYİCİ 'MONO DİZİ TİP ŞİFRELEYİCİ'

Dizi tip şifreleyiciler yüksek hızlı şifreli iletişim için uygundur çünkü bu tip şifreleyiciler yüksek çıktı hızı ve donanımda kolaylık sağlamaktadır. Genel olarak, dizi tip şifreleyiciler basitlik ve yazılım ve donanımda gerçekleştirilmelerindeki kolaylık nedeniyle Doğrusal Geri Beslemeli Kayan Saklaçları (LFSRs) kullanmaktadır. Fakat, bu randımanlı bileşenler güvenliği dikkate aldığımızda yeterli olmamaktadır. Tasarımcı, sistemi kriptanalize karşı daha güçlü yapmak için birçok doğrusal olmayan fonksiyonlar ve mekanizmalar kullanmalıdır. Bir dizi tip şifreleyici yüksek periyoda, yüksek doğrusal karmaşıklığa, iyi istatistiksel özelliklere sahip olmalı ve cebirsel saldırılar, ilinti saldırıları, zaman bellek ödünleşimi saldırıları, böl ve fethet saldırıları gibi birçok başarılı güncel saldırıya karşı dayanıklı olmalıdır

Bu tezde yeni bir dizi tip şifreleyici tasarımı önerildi. MONO cebirsel ve ilinti saldırılarına karşı güçlü olması için tasarlandı. Tasarım evresinde, hedef iyi rasgeleliğe, yüksek periyoda ve doğrusal karmaşıklığa sahip ve saldırılara karşı dayanıklı bir dizi tip şifreleyici tasarlamaktı. Diğer hedef gerçekleştirilebilir bir dizi tip şifreleyici tasarlamaktı. Ayrıca bu tezdeki yenilik, bellek ve dinamik çoğullama kullanarak bir dinamik saat denetimli süzgeç üretecinin önerilmesidir. MONO başlangıç aşaması, gizli anahtarın ele geçirilebilmesi için zorluğu bilinen matematiksel problemlerin çözümüne dayanmaktadır. MONO dizi tip şifreleyicinin yüksek periyot ve doğrusal karmaşıklığı sağladığı gösterilmiştir. Ayrıca sistem iyi istatistiksel özelliklere sahiptir. MONO şifreleyicisinin rasgele sayı üretimi temel gereksinimlerini sağlayıp sağlamadığını görmek için birçok istatistiksel test uyguladık. MONO bütün testleri geçti. MONO ilinti , cebirsel, böl ve fethet, zaman bellek ödünleşimi saldırıları gibi birçok önemli saldırıya karşı güvenli. Ayrıca MONO donanım olarak da incelendi ve uygun sonuçlar alındı. Sonuç olarak MONO donanım ve yazılım uygun için uygun bir dizi tip şifreleyici diyebiliriz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

A	Effect of S-box on the upper bound of *LC* of MONO
A_i	Four bit blocks used for mixing of S-box
B	Output of BBS generator
B_n	Set of Boolean functions in n variables
b_t	Observed keystream at time t
C	Multiplier term in the lower bound of *LC* of MONO
C_i	i^{th} character of ciphertext
c_i	Feedback coefficients
Cl_i	Clock of register Ri
D	Parameter of XL algorithm
d	Algebraic degree of a Boolean function
$d_{\max}$	Parameter used in constrained embedding attack
dec	Decimation of a binary vector
E	Total number of monomials of degree less than or equal to e
$E\{x\}$	Expected value of x
e	Degree of low degree terms in Fast Algebraic Attacks
f	A Boolean function
$f(x)$	A Boolean function of a variable x
f_1^t	First clocking function at time t
f_2^t	Second clocking function at time t
f_{C2}	Clocking function of R2
f_{C3}	Clocking function of R3
$f_{R1}(x)$	Feedback polynomial of R1
$f_{R2}(x)$	Feedback polynomial of R2
$f_{R3}(x)$	Feedback polynomial of R3
$\mathcal{F}_q$	A field defined on q elements
$\mathcal{F}^n{}_2$	A field of two elements and n variables.
$\mathcal{F}^*{}_q^l$	A generator multiplicative group of q^l elements.

F	Output Boolean function of MONO
F_1	1st degree terms in F
F_2	2nd degree terms in F
F_3	3rd degree terms in F
F_4	4th degree terms in F
$Free$	Number of linearly independent equations in R
GF(q)	Galois Field on q elements.
h_i	Number of clock to the register Ri
I	384 bit block obtained from $I = (B \oplus K) \| IV$
IV	Initialization Vector
K	Private Key
K^d	Terms of degree d in secret key values
k	Order of product in algebraic normal form
k_{reg}	Degree of feedback polynomial of generator register
k_j	j^{th} initial state of an LFSR
L_{R1}	Length of R1
L_{R2}	Length of R2
L_{R3}	Length of R3
L_{key}	Length of key for a block cipher
$L(\mathbf{s})$	Linear complexity of a sequence **s**
L^t	Linear state transition of an LFSR at time t
l_i	Initial equation derived in XL algorithm
m	Number of available keystreams in algebraic attacks
M	Memory required in time memory trade off attack
M_i	i^{th} character of plaintext
n_b	Block size of a block cipher
n	$n=pq$
n_{var}	Number of variables in a Boolean function f
N	Five bit value to be used in clock control
N^t	Value of N at time t
N	Length of the keystream sequence

N_s	State space in time memory trade off attacks
$O(x)$	Total computational complexity of an operation x
$p(x)$	A polynomial of x
P	Period of a primitive LFSR
P_i	Period of register Ri
P_{di}	Period of decimated version of register Ri
P_d	Deletion rate
P_z	Period of a keystream generator
P_{GR}	Period of the generator register
$\Pr\{x\}$	Probability of x
p	Large prime of BBS
q	Large prime of BBS
q	Number of elements in a field
Q	Six bit memory value to be used for multiplexing
R	Number of equations generated in XL algorithm
R1	First LFSR
R2	Second LFSR
R3	Third LFSR
R1(i)	i^{th} stage of R1
R2(i)	i^{th} stage of R2
R3(i)	i^{th} stage of R3
$R_2(\beta_t^6)$	Six bit output from R2 using consecutive β value at time t
S	Total value of decimating sequence
S_i	Symbols occurring in each register
s	Session value to be used in initialization
S_t^6	Six bit middle S-box output at time t
s_i	i^{th} element of S-box output
$\vec{s}$	Edge two bit of S-box output
t	Time
t_{corr}	Order of correlation immunity of a Boolean function f
T	Number of monomials occurring in XL algorithm
T_{attack}	Attack time

W	Symbol size of the stream cipher $q=2^W$
X_i	Independent binary random variables
x_i	Variables of F
$\vec{x}$	Five bit vector used in permutation of I
$\vec{y}$	Five bit vector used in permutation of I
z_i	i^{th} character of keystream
z^{-1}	Delay element

α	Linear dependency in Fast Algebraic Attack
δ	Size of α
σ	Seven bit initial value to determine location of β s
β	Six consecutive stages of R2 to be used in memory function
β_i	Each element of β
λ	Period of decimating sequence

ABSG	Alternative Bit Search Generator
AC	Autocorrelation
AES	Advanced Encryption Standard
AI	Algebraic Immunity
ANF	Algebraic Normal Form
BBS	Blum Blum Shub Generator
BSG	Bit Search Generator
CMOS	Complementary Metal-Oxide Semiconductor
DCVSL	Differential Cascode Voltage Switch Logic
DES	Data Encryption Standard
DFT	Discrete Fourier Transform
FIPS	Federal Information Processing Standard
gcd	Greatest Common Divisor
GE	Gate Equivalent
GR	Generator Register

IV	Initialization Vector
LC	Linear Complexity
LFSR	Linear Feedback Shift Register
NIST	National Institute of Standards and Technology
OTP	One Time Pad
S-box	Substitution Box
XL	Extension and Linearization

1. INTRODUCTION

In cryptographical terms symmetric cryptosystems are classified into two main parts according to their algorithms and output generation schemes. Block ciphers simultaneously encrypt blocks of characters of a plaintext message with a fixed transformation, whereas stream ciphers operate on individual plaintext digits (usually bits or sometimes byte) at a time with a time varying transformation.

Stream ciphers seem to be one of the best alternatives to high-speed communications, since they offer required security for high data rate applications and have algorithms that are popular in fast implementations. They are generally faster than block ciphers in hardware and have less complex hardware circuitry. Furthermore, stream ciphers can be more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Since stream ciphers have limited or no error propagation, they can be advantageous in situations where transmission errors are highly probable [1]. Stream ciphers are categorized into three main groups which are filter generators, combination generators and clock-controlled generators. Each of the groups have their own advantages and disadvantages. Stream ciphers generally make use of linear feedback shift registers for easy and efficient implementation and Boolean functions for their resistance against many attacks and efficiency. Combination and filter generators are the ones that use both the linear feedback shift registers and Boolean functions. Sometimes we encounter a mixed usage of Boolean functions and irregular clock-controlling in addition to other system components.

The rest of this chapter makes introduction to cryptology, symmetric key systems and cryptanalytic methods. Section 1.1 gives the motivation ideas to produce this thesis. Section 1.2 makes a brief introduction to cryptology. Section 1.3 discusses symmetric key ciphers and section 1.4 gives information about cryptanalysis and attack methods. Finally section 1.5 gives the outline of the thesis.

1.1. Motivation

LFSR based stream ciphers are attractive in cryptographical applications as these systems are easy in hardware implementation and appropriate for high speed data communications. They have also low error propagation, limiting the number erroneous decryption in a noisy transmission media.

Many stream cipher implementations use simple LFSRs , Boolean combining and filtering functions, clock control or irregular decimation to make a system with high period, high linear complexity, good statistical properties and resistance against well known cryptanalytic attacks. However, many systems had been observed to be weak against some attacks. Many of them have practical strength attacks, nevertheless it is important to develop systems having theoretical strength rather than practical. Because a successful theoretical attack shows the cryptological weakness in the algorithm. Therefore it is important to design stream ciphers with theoretical resistance against well known and recent attacks. Additionally, high period, high linear complexity, good statistical properties should be satisfied to result in a strong system.

MONO stream cipher is designed to resist against correlation and algebraic attacks. These two attacks are the most important threat against LFSR based stream ciphers. Many LFSR based stream ciphers are weak against algebraic and correlation attacks. With MONO, it is desired to design a new a stream cipher which is resistant against algebraic, correlation, time-memory trade off and divide-conquer attacks. Also it desired to satisfy high period, high linear complexity and pseudo-randomness theoretically.

1.2. Cryptology

Cryptology is uniting name for a broad scientific field in which one studies the mathematical techniques of designing, analysing and attacking information security services. Cryptology consists of two subfields; cryptography and cryptanalysis. Cryptography is the field in which one studies techniques for providing security services

and cryptanalysis is the field in which one studies techniques for defeating the security services. In [1], the goals for cryptography services are distinguished in four;

- **Data Integrity**: The goal is to ensure that only authorized users can alter the information without it being noticed.

- **Confidentiality**: The goal is to ensure that the information is only available to authorized users. Synonymous terms are privacy and secrecy.

- **Authentication**: The goal is to identify data origin or destination. Both parties in a communication sometimes need to ensure that other is a legitimate user. This can also be applied to the data itself, as to ensure a specific date and time that the message was sent.

- **Non-repudiation**: The goal is to ensure that someone cannot deny a previous commitment or action.

The classic information security goal, to provide security, is obtained by using a primitive called a cipher. Some examples of other primitives are signing primitives which should provide the same functionality as a handwritten signature, or Message Authentication Codes which provide means to ensure that a message has not been altered on the route to the receiver.

The cryptographical primitives can be divided into three categories; unkeyed primitives, symmetric-key primitives, and asymmetric-key primitives, where the latter are also known as public-key primitives.

The unkeyed primitives include, for example, hash functions which are also used in databases and search algorithms. Keyed primitives are the symmetric-key and asymmetric-key primitives, the focus will be on the cipher primitive and the symmetric- key primitive, as the topic of this thesis is symmetric stream cipher design.

1.3. Symmetric Key Ciphers

Symmetric-key ciphers are divided into two general categories. Block ciphers and stream ciphers. Block cipher is a device that takes as L_{key} bit key and an n_b bit plaintext string and produces n_b bit ciphertext, where plaintext is the cleartext message to be encrypted, ciphertext is the encrypted version of the plaintext and the key is the only secret information used in both sides of communication to have secret data communications. The parameter n_b is called the block size. Figure 1.2 shows the general structure of block ciphers.

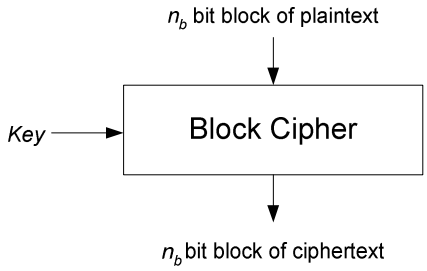

Figure 1.1. General structure of a block cipher

A block cipher must define a permutation on the set of n_b bit binary strings. Thus, if the key is fixed and we encrypt all possible n_b bit messages, then we will get all possible n_b bit strings as output. Famous block ciphers include the DES (Data Encryption Standard) and the AES (Advanced Encryption Standard) [1].

Stream ciphers on the other hand operates on individual characters in the underlying alphabet, with a time-varying function. Since the encryption is time-varying, we do not have the problem of patterns in the plaintext being encrypted to identical patterns in the ciphertext. Figure 1.2 shows the general structure of stream ciphers.

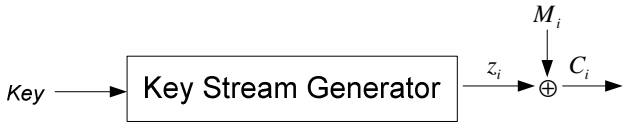

Figure 1.2. General structure of a stream cipher

Instead of using the key directly as in block ciphers, stream ciphers use key stream symbols z_i to additively encrypt the plaintext M_i to generate ciphertext C_i. Compared to block ciphers, stream ciphers are also easier, smaller and cheaper to build in hardware. These properties make stream ciphers a suitable cryptographical solution in telecommunication systems or high speed network devices for attaining secrecy.

1.4. Cryptanalysis and Attack Methods

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

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

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

The first and most important rule for the designer of a cryptographic primitive is called *Kerckhoff's Principal* : "The security of the encryption scheme must depend only on the secrecy of the key, and not on the secrecy of the algorithms."

From *Kerckhoff's Principal* we assume that an attacker knows everything concerning the cryptographical system and the protocols, except the secret key. Then we can classify the methods of attack according to the amount of information to the adversary and goal of the attack.

- **Ciphertext-only attack**: All the adversary sees is the ciphertext communicated between sender and receiver. This is the most difficult attack since the attacker has the least amount of information.
- **Known plaintext attack** : In this scenario the adversary knows both the plaintext and the corresponding ciphertext. It might seem slightly that improbable that both the plaintext and the ciphertext are revealed, but there are many situations where this could happen.
- **Chosen plaintext attack** : Here the adversary has access to an oracle, which can encrypt any given plaintext under the correct key. This is an even more powerful attack than the known plaintext attack, since the attacker can now choose plaintexts that are especially favorable for breaking the cipher.
- **Chosen ciphertext attack**: This is similar to the chosen plaintext attack, but now the adversary has access to two oracles instead, one that encrypts any given plaintext and one that decrypts any given ciphertext except the ones the attack is trying to break. This attack is naturally more powerful than all the previous attacks, since the adversary has more freedom.

1.5. Thesis Outline

The rest of the thesis is organized in the following fashion. Section 2 makes introduction to stream ciphers. In this part, the definition and properties of a stream cipher, linear feedback shift registers and Boolean functions are given. Also some basic stream cipher designs are discussed. In section 3, we give a comprehensive information about the proposed stream cipher MONO. In this part, all the components, initialization and design objectives of MONO are given. In section 4, the keystream properties of MONO are discussed. In section 5, we give the results of statistical test results on MONO keystream output. In section 6, we give security analysis. Here many recent cryptanalytic attacks are discussed. In section 7, we give the hardware equivalent , therefore the hardware complexity of MONO. Section 8 concludes the thesis.

2. INTRODUCTION TO STREAM CIPHERS

2.1. Stream Ciphers

Stream ciphers are divided into synchronous and self-synchronous. In [1] detailed information about this distinction is given. A synchronous stream cipher is a finite machine for which the keystream is generated from the key, but independently of the plaintext message and the ciphertext. A self-synchronizing stream cipher is a finite state machine for which the keystream is generated as a function of the key and a fixed number of the previous ciphertext symbols.

It is very common to use the xor function as the output function. since the xor is the filed addition operation we usually denote such a cipher, an additive stream cipher. We can also describe the additive stream cipher as a pseudo-random number generator or a keystream generator whose output is xored to the plaintext. The key is used to initialize or seed the generator, which starts to produce pseudo-random bits. We note the similarity to the OTP; instead of having complete keystream as the secret key, we want to have a smaller key which is used for seeding. Then we want the generator to produce a keystream generator used as an additive stream cipher.

A synchronous stream cipher is not particularly vulnerable to errors in the transmission, since each symbol is encrypted independently. An active attacker can change a particular ciphertext symbol and render the corresponding plaintext symbol erroneous, but all other symbols will remain correct. The receiver has no direct possibility of validating the received message, thus additional mechanisms for message authentication are needed to defend against these kind of attacks.

However, if the attacker deletes or inserts a symbol into the ciphertext or if that happens by accident, the sender and receiver lose synchronization and all plaintext symbols following the deleted one will become erroneous during decryption. To defeat this problem, perfect synchronization between the sender and receiver needs to be ensured and techniques for detecting loss of synchronization employed.

There are many applications where the synchronization problem is much more severe than a corrupt ciphertext symbol. For example, in a streaming video sequence some erroneous symbols will only affect the picture over a limited are and during a small time frame. But a synchronization error will generate a completely useless video. In such a setting, we could use a frame based communication protocol, where the message sequence is first divided into smaller frames which are numbered with a frame number. We then add a feature called to the stream cipher called an Initialization Vector (IV), which is publicly known value used in the initialization of the stream cipher together with the secret key. Now, with a fixed key but with a changing IV, the stream cipher will produce different sequences of keystream material for each IV. For each frame the receiver tires to decrypt, he looks at the public frame number attached to the frame of encrypted information and pre-initializes the stream cipher with the new frame number as IV and the secret key, and then decrypts the information. If synchronization is lost for a single frame it will only affect a small amount of information, until a new frame arrives and he can resynchronize. An important practical issue is that the re-initialization of the cipher with a new IV should be vary fast, to be able to handle high information rate sources such as streaming video.

Before we present some stream cipher designs, we shall introduce some of the basic structures often found inside stream ciphers, such as linear feedback shift registers, Boolean functions.

2.2. Linear Feedback Shift Registers

Recalling the keystream generator and its similarity to the OTP, we state that the fundamental property of a keystream generator is to produce as random looking symbols as possible. The distribution of symbols should be uniform and unpredictable. A good start is to use a Linear Feedback Shift Register (LFSR) for achieving a good distribution. The direct output of an LFSR is not a good keystream generator since each symbol produced is simply a linear combination of the previous symbols, and thus very easy to predict. Nevertheless, LFSRs are widely used components inside stream ciphers.

An LFSR is a device made up by registers, able to hold one symbol at a time. The symbols are elements from a field $\mathcal{F}_q$, over which we have chosen to define the LFSR. In stream cipher applications we often have $q=2$ (binary field) or some extension field of the binary field $q=2^W$, where W is the symbol size of the stream cipher. Initially we can think of an LFSR as a hardware construction, though it is very easy to implement in software as well. Thus we assume a system clock which is responsible for the timing of all events. Figure 2.1 shows a general LFSR, where the circles denote multiplication with constant c_i and $\oplus$ is the field addition operation. At each clocking of LFSR, the registers read a new symbol from their respective input, and as the registers are coupled in series, the symbols move forward at each clocking. The first register receives a new symbol from their respective input, and as the register receives a new symbol which is linear combination of the symbols found in the registers after the previous clocking. The exact linear combination used for producing the feedback symbol is determined by the feedback coefficients c_0, c_1,…, c_l shown in Figure 2.1. Since we need the actual feedback connection c_0 to get any symbols into the register, one normally assumes c_0=1. As we do not need more registers than necessary to make the feedback connection work, we also assume $c_l \neq 0$ and define the length of the LFSR to be l.

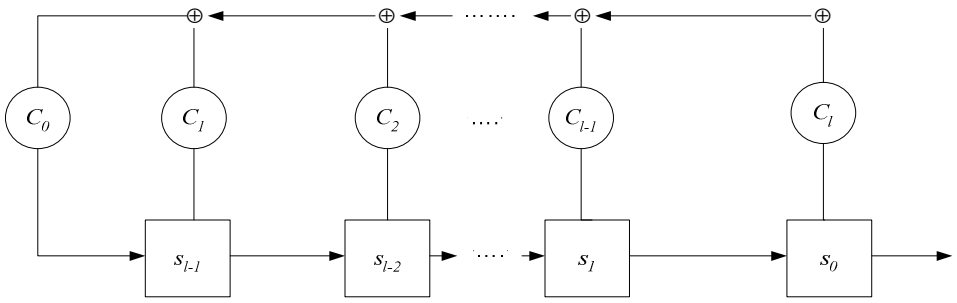

Figure 2.1. General form of an LFSR of length l

At each time t≥0 the device is clocked, and we obtain a new symbol $s_t \in \mathcal{F}_q$ at the output of the device. Due to the linear feedback, the symbols s_t will always fulfill the linear recurrence equation $c_0^{-1}s_{t+1} - c_1 s_{t+l-1} - c_2 s_{t+l-2} - \ldots - c_l s_t = 0, \quad t \geq 0.$

If the field characteristic is 2, and we assume c_0=1, we get the simpler form

$$\sum_{j=0}^{l} c_j s_{t+l-j} = 0, \quad t \geq 0 \tag{2.1}$$

The connection at c_0 is called the feedback position and each $c_j \neq 0$, $1 \leq j \leq l$ is called a tap position. The content of the register at time *t* is $\mathbf{S}_t=(s_{t+l-1}, s_{t+l-2}, ..., s_t)$ and is called the state of the LFSR at time *t.* The first l symbols to be produced, $\mathbf{S}_0=(s_{l-1}, s_{l-2}, ..., s_0)$, are loaded into the registers at the start, and we denote this state the initial state or the starting state of the LFSR.

Since there are only a finite number of possible states q^l, the sequence produced by the LFSR must repeat itself after a finite period, i.e. for every starting state we can find a *T* such that $s_t = s_{t+T}$, $t \geq 0$. The period depends on properties of the feedback polynomial.

A polynomial $p(x) \in \mathcal{F}_q[x]$ is said to be irreducible over $\mathcal{F}_q$ if it can not be factored into polynomials of smaller positive degrees in the ring of polynomials $\mathcal{F}_q[x]$. An irreducible polynomial $p(x) \in \mathcal{F}_q[x]$ of degree l is said to be primitive id the root of $p(x)$ is a generator of the multiplicative group $\mathcal{F}^*{}_q^l$. The period *P* of an LFSR with primitive feedback polynomial $p(x) \in \mathcal{F}_q[x]$ of degree l and with non-zero starting state is $P = q^l - 1$. Thus if we start with a non-zero state in the LFSR and use a primitive feedback polynomial, all possible states except the all-zero state will appear during a period. An LFSR with a primitive feedback polynomial is called a maximum-length LFSR, and the sequence produced is called a maximum-length sequence. Note that again the initial state must be non-zero for the sequence to be maximum-length, and hereafter it is assumed that the starting state is as such.

The linear complexity (*LC*) of a sequence $\mathbf{s}=s_0, s_1, ..., s_i \in \mathcal{F}_q$, denoted $L(\mathbf{s})$, is the length of the shortest LFSR, defined over $\mathcal{F}_q$, that generates the sequence. If the sequence is the all-zero sequence, then the linear complexity is 0, and if no LFSR can generate the sequence, then $L(\mathbf{s})=\infty$. The linear complexity can be determined with the Berlekamp-Massey algorithm [16] which efficiently computes the feedback polynomial of the LFSR given at least 2 $L(\mathbf{s})$ of output symbols. A pure LFSR is not a good keystream generator. As an example we can take a binary LFSR of length 256, which produces a maximum length sequence of length 2^{256}-1. The Berlekamp-Massey algorithm needs only 512 consecutive bits in a known-plaintext attack to fully determine the feedback polynomial and thus the complete sequence.

Sequences generated by maximum-length LFSRs have good statistical properties, desirable for keystream generator construction, but we need to destroy the linearity, i.e. increase the linear complexity, before the sequence can be used. There are several methods for doing this as we will see in Section 2.4. Next we touch upon the Boolean functions in Section 2.3 .

2.3. Boolean Functions

A Boolean function is essentially a function which maps one or more binary input variables to one binary output variable. That is, we write this as a mapping from a vector of binary vector of binaries $\mathbf{x} = (x_1, x_2, ..., x_n)$, where $x_i \in \mathcal{F}_2$, $1 \le i \le n$ to a single output bit.

$$f : \mathcal{F}^n{}_2 \rightarrow \mathcal{F}_2$$

For n input variables there exist 2^{2^n} distinct Boolean functions and we denote the set of Boolean functions in n variables by B_n. There are several ways to represent a Boolean function. If the number of input variables is small, a truth table can be constructed where all possible input vectors are listed with the corresponding output value. Table 2.1 shows a truth table for a particular Boolean function of three variables.

Table 2.1. The truth table of the Boolean function $f(x_1, x_2, x_3) = x_1x_2 + x_2 + x_3$

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

If the number of input variables is large, the listing of all possible vectors is infeasible and we have to resort to a more compact description, for example the algebraic normal form (ANF)

$$f = \sum_{\mathrm{u}\in\{0,1\}} \lambda_{\mathrm{u}} \left(\prod_{i=1}^{n} x_i^{u_i} \right), \qquad \lambda_u \in \{0,1\}, \quad \mathrm{u} = (u_1, u_2, ..., u_n) \tag{2.2}$$

A product of k distinct variables is called an kth order product of the variables. To simplify, we can state that every Boolean function can be written as a (modulus 2) sum of distinct kth order products of its variables.

The Hamming weight of a binary vector is the number of ones in the vector, and the algebraic degree of a Boolean function is the highest order of the terms in the ANF. For cryptographical applications, several other interesting properties of Boolean functions have to be considered. First of all, we say that a Boolean function is balanced if the number of zeros in the output column of the truth table is equal to the number of ones. Hamming distance between two functions is defined as the number of entries in the truth table output column where these two functions differ.

The nonlinearity of a Boolean function is the Hamming distance to the nearest affine function. a high nonlinearity is a desirable property since it will decrease the correlation between the output and the input variables or a linear combination of input variables. Many of the known attacks on stream ciphers utilize the weakness of such a correlation between the combining Boolean function and some affine function.

Another important measure is the correlation immunity of the Boolean function. In [13] , author defines the correlation immunity as follows. Let $X_1, X_2, ..., X_n$ be independent binary random variables, each taking the values 0 or 1 with probability 1/2. A Boolean function $f(x_1, x_2, ..., x_n)$ is said to be *t*-th order correlation immune, if for each subset of *t* variables $X_{i_1}, X_{i_2}, ..., X_{i_t}$ with $1 \le i_1 \le i_2 \le ... \le i_t \le n$, the random variable $Z = f(X_1, X_2, ..., X_n)$ is statistically independent of the random vector $(X_{i_1}, X_{i_2}, ..., X_{i_t})$.

A Boolean function which is both balanced and *t*-th order correlation immune is said to be *t*-resilient. It is shown in [13] that there is a trade of between the algebraic degree and the order of correlation immunity. Let $f(x)$ be a balanced Boolean function in n_{var} variables of algebraic degree d which is t_{corr}-th order correlation immune. Then the following upper bound must hold

$$\begin{cases} d + t_{corr} \le n_{\text{var}} - 1 & \text{if } 1 \le t_{corr} \le n_{\text{var}} - 2 \\ d + t_{corr} \le n_{\text{var}} & \text{if } t_{corr} = n_{\text{var}} - 1 \end{cases} \tag{2.3}$$

To summarize the properties of Boolean functions and the implications of their value when used as a combining function in a keystream generator; algebraic degree is desirable since it increases the linear complexity of the resulting keystream, a high nonlinearity gives a weaker correlation between the input variables and the output variable and increases the resistance to correlation attacks and high correlation immunity forces the attacker to consider several variables jointly and thus decreases the vulnerability of divide-and-conquer attacks.

2.4. Some Basic Stream Cipher Designs

Stream ciphers based on LFSRs can be divided into three general categories, nonlinear combination generators, nonlinear filter generators, and clock-controlled generators. Naturally, the classification of ciphers into these categories is not distinct as several design methods can be used simultaneously.

The nonlinear combination generator consists of n LFSRs, whose outputs are combined in a Boolean function *f*. The output of the Boolean function is the keystream output. The Boolean function must have high algebraic degree, high nonlinearity and preferably a high order of correlation immunity. We know that we must employ a large number of LFSRs to get both high algebraic degree and a high order correlation immunity. Figure 2.2 shows the general structure of a nonlinear combination generator.

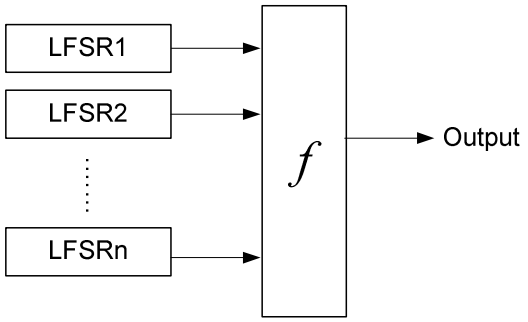

Figure 2.2. Structure of a nonlinear combination generator

The nonlinear filter generator takes a different approach and uses one single LFSR, from which the inputs to the Boolean functions are taken. Figure 2.3 shows the structure of a nonlinear filter generator.

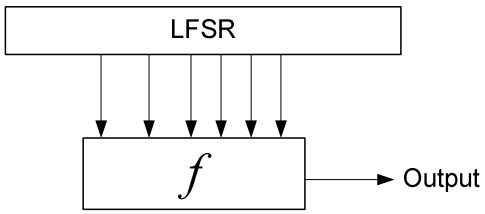

Figure 2.3. Structure of a nonlinear filter generator

In this case the Boolean function is called the filtering function. Not all the elements of the LFSR need to be taken as inputs to the filtering function. The upper bound on the linear complexity for a nonlinear filter generator is $L(\mathbf{s}) \leq \sum_{i=1}^{d} \binom{l}{i}$, where d is the algebraic degree of the filtering function, l is the length of the LFSR and s is the generated keystream.

We could also extend the ideas and use a time varying function or a combiner/filter mechanism with memory. E0 which is the stream cipher used in the Bluetooth system is a combiner with memory. The principal difference from the pure combiner/filter is that a part of the state space is updated via a nonlinear process. Hence for systems with memory the properties of pure Boolean functions are not directly applicable, especially the tradeoff between algebraic degree and correlation immunity does not hold.

Another interesting approach is to include key dependent S-boxes in the combining filtering function. this makes cryptanalysis much harder than for a pure Boolean function since effectively, the cipher is not completely specified until the key is known.

The final category of stream ciphers is the clock-controlled generators. They differ from the above in the sense that the LFSRs are not clocked irregularly. In the nonlinear combination generator and nonlinear filter generator we have a system clock which advances the LFSR one step for each clocking and for each clocking we produce a new keystream bit or several keystream bits. The idea in a clock-controlled generator, is to control the number of clockings of the LFSRs using some irregular signal. The clocking signal could be the output of another LFSR or some other internal variable of the cipher. The linearity of the output of the irregularly clocked LFSR is then destroyed and attacks based on a regular motion of the LFSR are harder to mount. One classical example of a clock-controlled generator is the alternating step generator. In this cipher, operation is as follows. Register R1 is clocked by the system clock. If the output of R1 is 1 then R2 is clocked and R3 is not clocked but its previous output bit is repeated. If the output of R1 is 0 then R3 is clocked and R2 is not clocked but its previous output bit is repeated. The sum of the outputs from R2 and R3 is taken as the keystream output. Figure 2.4 shows the alternating step generator.

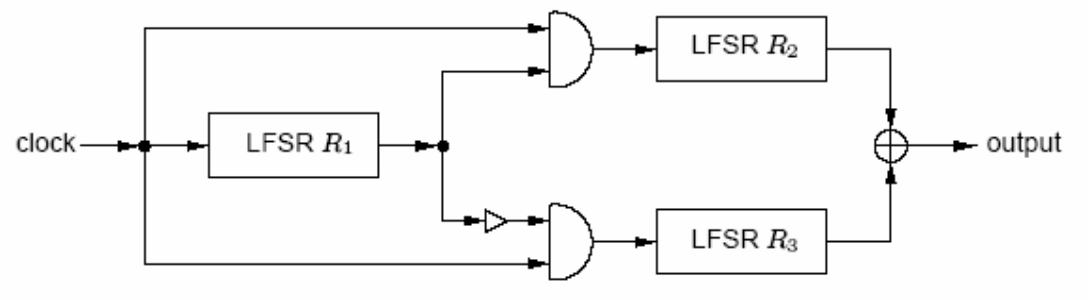

Figure 2.4. Alternating step generator

Two other very elegant clock-controlled generators are shrinking and self-shrinking generators. These two and other designs are discussed in detail in [1].

3. MONO STREAM CIPHER

In this thesis, I propose a new LFSR based binary stream cipher. The stream cipher is basically a clock-controlled stream cipher [2,3,4] which also uses dynamic multiplexing [5] and 16x16 S-box of AES (Advanced Encryption Standard), memory and a 3-resilient output Boolean function. The stream cipher will use a 128 bit private key K, private primes p and q; where $n=pq$ is a publicly known value and a publicly known 256 bit initialization vector IV.

The cipher consists of three linear feedback shift registers (LFSRs). One of them is responsible for determining the number of clocking of the other two shift registers and generating parameters for memory and clock control. Only one of the registers contributes the keystream output generation which is R3. Another LFSR (R2) contributes to multiplexing and producing a value (memory) for clock-control unit in order to control clocking of the registers dynamically.

One of the most important properties of this cipher is its clocking mechanism. The clocking mechanism depends on dynamic functions. That is clock-control unit does not use predetermined stages of the control register. In fact, the stage positions used in the functions of the clock-control unit varies for every cycle of the cipher algorithm with the memory bits.. The cipher is indeed a dynamically clock-controlled filter with memory and multiplexing. The registers are clocked according to a rule governed by clock-control unit and the output of the data generating LFSR R3 is subject to a balanced Boolean function. A dynamic term which is the memory part of the cipher is used to dynamically change the clocking values and multiplexed tap values of R3. The memory information is generated for each cycle of the cipher and used at the next cycle. Therefore these terms are saved in memory for one duration of cycle. Also AES S-box is used to filter 8-bits of clock-controlling LFSR R1 in order to make the system stronger.

3.1. Introduction to MONO Stream Cipher

The proposed stream cipher is depicted in Figure 3.1. The length and the feedback polynomials of each LFSR is given below :

L_{R1} = 59 bits (Prime)

L_{R2} = 61 bits (Mersenne Prime)

L_{R3} = 127 bits (Mersenne Prime)

Feedback polynomials are chosen in such a way that each LFSR will produce all of its non-zero states. That is, feedback polynomials of each register should be primitive to produce the maximum period individually. All of the registers use primitive polynomials with seven coefficients. As it can be understood from the equations; the coefficients are uniformly separated from each other [11]. The length of each register is a prime number. The reason of this choice will be understood when we discuss keystream period and linear complexity in section 3. Also the length of R2 and R3 are Mersenne prime, that is, period of these registers are also prime. LFSRs with Mersenne prime length generator registers are being used very often in design of hardware based crypto systems such as stream ciphers.

$$f_{R1}(x) = x^{59} + x^{49} + x^{38} + x^{27} + x^{18} + x^{9} + 1 \quad (3.1)$$

$$f_{R2}(x) = x^{61} + x^{52} + x^{41} + x^{31} + x^{20} + x^{10} + 1 \quad (3.2)$$

$$f_{R3}(x) = x^{127} + x^{106} + x^{84} + x^{63} + x^{42} + x^{21} + 1 \quad (3.3)$$

Each polynomial satisfies maximum possible states for each register. Therefore R1, R2 and R3 will have periods of $(2^{59}-1)$, $(2^{61}-1)$, $(2^{127}-1)$ respectively.

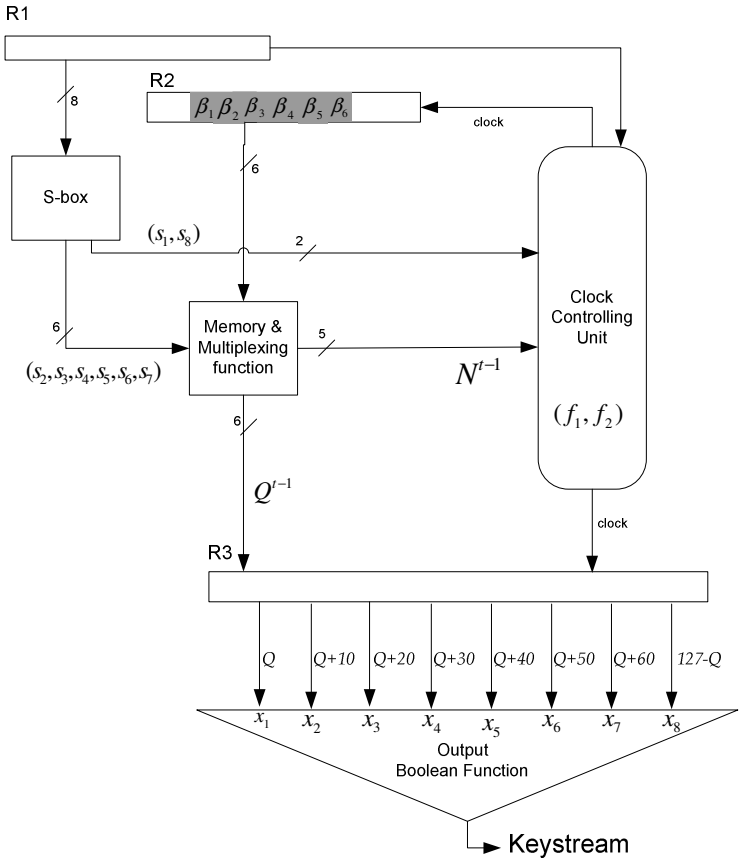

Figure 3.1. The MONO stream cipher

3.2. Initialization

Initialization of the cipher consists of three major parts. The first one is the Blum Blum Shub Generator. The second one is the shift-substitute process. The last one is the generation of six β s.

As it was told before, the cipher uses a 128-bit private key *K* and a publicly known 256 bit initialization vector *IV*. The cipher also uses two private prime numbers p and q . These two numbers will be used in the Blum Blum Shub (BBS) generator described in [9]. Let us give a short information about the procedure of this generator. Two large primes, p and q, will be chosen such that both have a remainder of 3 when divided by 4. That is ;

$$p \equiv q \equiv 3 \pmod 4 \tag{3.4}$$

Let $n=p$xq. Then a number s which is relatively prime to n is chosen. Then the BBS generator produces a 128-bit sequence according to the following algorithm:

$$\begin{aligned} &X_0 = s^2 \bmod n \\ &for\ i = 1\ to\ 128 \\ &\qquad X_i = (X_{i-1})^2 \bmod n \\ &\qquad B_i = X_i \bmod 2 \\ &end \end{aligned} \tag{3.5}$$

At the end of this process, 128-bit sequence B is generated. Then using K, IV and B the following sequence I is calculated.

$$I = (B \oplus K) \| IV \tag{3.6}$$

As can be understood from (3.6); B and K are XOR'ed and then the result is concatenated with the initialization vector IV. So the final result is a 384-bit sequence I.

Then I is divided in to 32 blocks of length 12. The operation is shown in the Figure 3.2.

12 bits	12 bits	……	12 bits	12 bits
I(1)	I(2)	……	I(31)	I(32)

Figure 3.2. 32 blocks of I

Then the all the blocks are changed and shifted according to the following algorithm:

$$
\begin{aligned}
&\vec{x} = 1 \text{ to } 5 \\
&\vec{y} = 8 \text{ to } 12 \\
&I(1) = I(1 + dec(I(1, \vec{x}))) \oplus I(1 + dec(I(1, \vec{y}))) \\
&Swap\ I(1 + dec(I(1, \vec{x}))) \ \&\ I(1 + dec(I(1, \vec{y}))) \\
&for\ i = 2\ to\ 32 \\
&\qquad \vec{x} = 1 + \{[\vec{x} + dec(I(i, \vec{x}))] \bmod 12\} \\
&\qquad \vec{y} = 1 + \{[\vec{y} + dec(I(i, \vec{y}))] \bmod 12\} \\
&\qquad I(i) = I(1 + dec(I(i, \vec{x}))) \oplus I(1 + dec(I(i, \vec{y}))) \\
&\qquad Swap\ I(1 + dec(I(i, \vec{x}))) \ \&\ I(1 + dec(I(i, \vec{y}))) \\
&end \\
&I = I(1) \,||\, I(2) \,||\, ... \,||\, I(31) \,||\, I(32)
\end{aligned}
\tag{3.7}
$$

where $dec(I(i, \vec{x}))$ is the decimal value of 5-bit length vector $\vec{x}$ in actual i^{th} block of I. Since each block is 12-bits length, there can be some occasions that intermediate values of 5-bit vectors $\vec{x}$ and $\vec{y}$ may be in position greater than 12; then the remaining consecutive bits will start from the first bit of the same block. The algorithm given above is an efficient block cipher designed to generate speed highly nonlinear initialization for MONO.

After these operations, we drop the last 38 bits and then we have 346 bits remained. Then the initial 7 bits of I will be our σ. This value will be used to determine the final values of βs. Below we show how to determine the value of σ.

$$\sigma = \sum_{i=1}^{7} 2^{7-i} I(i) \tag{3.8}$$

After this operation, next 12 bits will be used to calculate initial Q and N. Then following 80 bit block will be divided into twenty 4-bit blocks which are A_1 through A_{20}. A_1-A_2, A_5-A_6, A_9-A_{10}, A_{13}-A_{14} and A_{17}-A_{18} represent the row pairs to be swapped and A_3-A_4, A_7-A_8, A_{11}-A_{12}, A_{15}-A_{16} and A_{19}-A_{20} represent the column pairs to be swapped for the S-box.

Then starting from the register R1 to R3; the remaining 247 bits of I will determine the initial values of each register by chunking the sequence into 59, 61 and 127 bit blocks.

3.3. Components of MONO Stream Cipher

3.3.1. The AES S-box

The cipher uses 16x16 s-box of AES which is given in Figure 3.3. Each content of the s-box contains 1-byte (8 bit) of data. For each cycle of the algorithm, the S-box takes 8 bits from R1 as input. These input values will come from the stages R1(1), R1(10), R1(18), R1(26), R1(35), R1(43), R1(51), R1(59). The output is determined as follows : The four bits in the middle of the 8-bit block will determine the row and concatenation of the first two and last two bits will determine the column to be used in the S-box. Suppose that the vector $\vec{b}$ = (00001111) is input to S-box then the row will be determined from the decimation of the sub-vector $\vec{b}_1$ = (0011) which is "3" and column will be determined from the decimation of the sub-vector $\vec{b}_2$ = (0011) which is "3" again. Then we select the content on the intersection of the 3rd row and 3rd column. The result is {C3} in hexadecimal notation; and in binary form it is (11000011).

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

Figure 3.3. AES S-box

3.3.2. The Memory & Multiplexing Function and N, Q values

Figure 3.4 shows the internal structure of memory and multiplexing function.

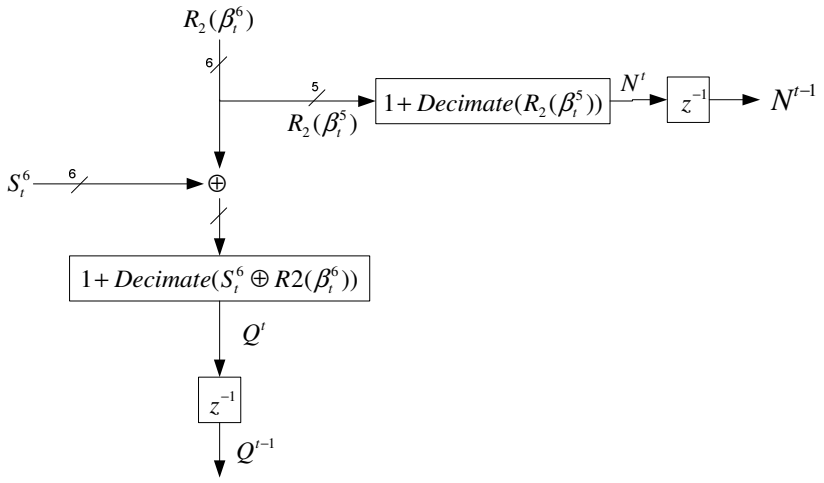

Figure 3.4. Internal structure of Memory & Multiplexing function

The memory function is also an important tool for the proposed cipher. This function generates a value Q to be used for multiplexing and N to be used in clock-control. z^{-1} represents the time delay element. For each cycle of the cipher a new Q and N value is calculated to be used for another cycle. The memory and multiplexing function calculates the N value and the Q value at time t in the following manner ;

$$N^t = 1 + \sum_{j=1}^{5} R_2^t(\beta_j).2^{j-1} \quad j \in \{1,2,3,4,5\} \tag{3.9}$$

$$Q^t = 1 + \sum_{i=1}^{6} \left(R_2^t(\beta_t^6) \oplus S_t^6 \right)_i .2^{i-1} \qquad i \in \{1,2,3,4,5,6\} \tag{3.10}$$

As can be understood from the equation (3.9), N value is determined from the five stages of R2. Q value is obtained from the decimation of bitwise XOR of middle 6-bit S-

box output and the 6 stages of R2 named β. The position of the stages depends on the β calculated in the initialization phase. In fact final β values are determined after a cyclic shift of predetermined consecutive stage indexes by σ. We decided to choose R2(29), R2(30), R2(31), R2(32) ,R2(33) and R2(34) as predetermined stages. The final β values will be ;

$$\beta_1 = R2\{1 + ((29 + \sigma) \bmod 61)\} \quad (3.11)$$

$$\beta_2 = R2\{1 + ((30 + \sigma) \bmod 61)\} \quad (3.12)$$

$$\beta_3 = R2\{1 + ((31 + \sigma) \bmod 61)\} \quad (3.13)$$

$$\beta_4 = R2\{1 + ((32 + \sigma) \bmod 61)\} \quad (3.14)$$

$$\beta_5 = R2\{1 + ((33 + \sigma) \bmod 61)\} \quad (3.15)$$

$$\beta_6 = R2\{1 + ((34 + \sigma) \bmod 61)\} \quad (3.16)$$

As it was emphasized before, we got six different β values. These β's have values between 1 and 61. Therefore it is possible to choose any value of R2 for a particular session.

We have strictly touched upon the importance of six different β values. From its definition it can easily be seen that number of different *N* values is 2^5=32. For each same β value, the number of different memory values will decrease by ½ . For example if 3 of first 5 β values were same, the number of different *N* values will decrease to 2^5 /2^2=8. And this will lead to a non-uniform memory and clocking value distribution. Also as the number of memory bits decrease, the degree of a initial state-output relation for multivariate cryptanalysis decreases. That is, it would be possible to find a lower degree relation and therefore less operations and known keystream would be needed. By having different stages to be used to generate the memory *N,* we have lowered those possibilities.

The N value is fed into clock-controlling unit to choose dynamic stages from R1 to determine clocking number of R2 and R3. Also Q is multiplexed into R3 to mix the keystream output. *N* will have values between 1 and 32 ; therefore these values can be

represented using 5-bits. *Q* have values between 1 and 64 ; therefore these values can be represented using 6-bits. The memory *Q* and *N* are designed is such a way that, memory bits are not fed into output function directly. On the contrary; *N* is used for dynamic irregular clocking and *Q* is used for dynamic multiplexing of output generating register R3. This is important, because the use of memory bits to contribute the output directly, increases the possibility of existence of a relation between internal states of LFSRs and the key stream outputs. In [22], author find a relation for Bluetooth system, since E0 uses memory bit in the output Boolean function.. The author stated that this property made internal state/output relation of the system be free of the memory bits. In MONO, memory bits are used to determine the clocking of LFSRs and multiplexing.

3.3.3. The Clock-Controlling Unit

The clock-controlling unit makes use of R1 , memory & multiplexing function and two edge bits of secret structured S-box in order to control the number of clockings of the registers R2 and R3. In fact R1 is used to generate bits to be used by this controlling unit and the memory & multiplexing function. f_1 and f_2 are dynamic functions which are dependent on *N* and internal values of R1. Also the choice of these functions depends on the two bits of S-box. These functions are given as follows ;

$$f_1^t\left\{R_1^t(a_1^t), R_1^t(b_1^t)\right\} = 2R_1^t(a_1^t) + R_1^t(b_1^t) + 1 \tag{3.17}$$

where $a_1^t = N^{t-1}$ and $b_1^t = 59\text{-}N^{t-1}$.

$$f_2^t\left\{R_1^t(a_2^t), R_1^t(b_2^t)\right\} = 2R_1^t(a_2^t) + R_1^t(b_2^t) + 1 \tag{3.18}$$

where $a_2^t = N^{t-1} + 27$ and $b_2^t = 33\text{-}N^{t-1}$.

The clock-controlling unit makes use of 2 edge bits of S-box which is $\vec{s} = (s_1, s_8)$. If bitwise XOR of elements of $\vec{s}$ is one then R2 uses f_1 and R3 uses f_2 for clock control. However, if bitwise XOR of elements of $\vec{s}$ is zero then R2 uses f_2 and R3 uses f_1 for clock control. In equation form we can define the clocking functions of R2 and R3 respectively;

$$f_{C2} = (s_1 \oplus s_8) f_1 \oplus (1 \oplus s_1 \oplus s_8) f_2 \quad (3.19)$$

$$f_{C3} = (s_1 \oplus s_8) f_2 \oplus (1 \oplus s_1 \oplus s_8) f_1 \quad (3.20)$$

Table 3.1. shows the clocking functions used for each possible $\vec{s}$. That is, the result of f_{C2} and f_{C3} is given by substituting possible values of $\vec{s}$ into equations (3.19) and (3.20).

Table 3.1. Clocking functions used for R2 and R3

s_1	s_8	$s_1 \oplus s_8$	**Clock-R2**	**Clock-R3**
0	0	0	f_2	f_1
0	1	1	f_1	f_2
1	0	1	f_1	f_2
1	1	0	f_2	f_1

Therefore one can see that registers R2 and R3 will be clocked once at minimum and four times at maximum for one clock of R1. Also, since the N value changes for each cycle of the cipher (a new N value is calculated) ; the stages chosen from R1 in functions f_1 and f_2 are updated. This is the important property of the clock-controlling unit which makes it strong.

3.3.4. Ciphering Algorithm

- Starting with the first *N* and *Q* value; the clock-controlling unit and multiplexing operates according to the rules we have discussed before. By the way, R1 is clocked once for each cycle.
- For each clock of R1, 8 bits from the predetermined stages of R1 are fed into the S-box and the middle 6-bit output of S-box (S_t^6) is used for multiplexing and remaining two edge bits ($S_t^2 = \vec{s}$) are used for clock-control.
- Using $\vec{s}$, N , f_1 and f_2 ; the clocking of R2 and R3 is determined.

- For each clock of R1 ; R3 produces bits from the stages *Q*, *Q+10, Q+20, Q+30, Q+40, Q+50, Q+60, 127- Q* of R3 as shown in Figure1. Using these stages of R3 as input to $(x_1x_2x_3x_4x_5x_6x_7x_8)$, the output Boolean function gives an keystream output.

The output Boolean function *F* is balanced and 3rd-order correlation immune [12,13]. We have to set at least four variables to zero in order that *F* becomes linear.

$$F = x_1 + x_2 + x_3 + \sum_{4\le i\le j\le 8} x_i x_j + \sum_{4\le i\le j\le k\le 8} x_i x_j x_k + \sum_{4\le i\le j\le k\le l\le 8} x_i x_j x_k x_l \qquad (3.21)$$

The output function *F* is used in order to prevent the cipher against algebraic and correlation attacks. However one can also define an irregular decimation (shrinking generator, ABSG, BSG) at the output of the data generating LFSRs. In this case we will need buffer mechanism in order to synchronize the plaintext and keystream as these irregular decimating systems neglect some of the outputs depending of the internal algorithm. Therefore I think that the 3rd-order correlation immune function *F* will help to resist against algebraic attacks (a new successful approach for cryptanalysis against regularly clocked stream ciphers). Therefore an adversary will have to guess internal state of R1, R2, S-box and the position of β values and then solve multivariate equations given by *F* in order to get the initial values of R3 . As one can see that this attack can theoretically be possible however, it will need huge amount of guess, memory, processing complexity , considerable amount of known keystream and the difficulty of solving multivariate equations.

3.4. Design Objectives

In this section we describe the design ideas to constitute the MONO stream cipher. MONO stream cipher is designed to meet the requirements such as good statistical properties such as high keystream period, high linear complexity and random behavior , security against well known cryptographic attacks. We believe that system is also fast enough to be used in communication techniques requiring high data rates. We discuss design objectives of MONO part by part.

3.4.1. Choice of LFSRs

All the LFSRs used in the system has primitive polynomials to achieve maximum period. Also R2 and output generating LFSR R3 has Mersenne prime lengths to achieve maximum possible total period for the system. Feedback polynomials are chosen in such a way that; the total tap numbers used in the polynomials are not very sparse to resist against attacks using this fault. Additionally, total number of tap positions are not large, so that it could be implemented in hardware with lower cost.

3.4.2. Usage of Memory

Systems with low nonlinearity and linear complexity uses memory in order to increase the nonlinearity and linear complexity without making concession in correlation immunity. Stream cipher with memory, proposed so far, had used the memory information in the output function with the output of LFSRs. This leads to increase in nonlinearity and linear complexity however, the same property makes the system representable by low degree algebraic relations between internal state and output, free of memory information.

MONO also uses memory (Q and N). However in MONO the memory is not used in the output Boolean function because of the reasons above. The memory information is used to make the clocking of the system dynamic. Also memory information is used to dynamically multiplex bits from the register R3. This fact also provides nonlinearity and linear complexity for MONO. In our simulations all the memory values are distributed in an approximate uniform fashion. We observed that the probability of each memory value is nearly the same. This property was desired at the design phase of MONO. Also another useful property is that , the memory function can easily be implemented in hardware.

3.4.3. Dynamic irregular clocking

Regularly clocked systems are very open to correlation and algebraic attacks. In order to reduce the correlation between the output bit and state of LFSRs, we made clocking dependent on the state of clock controlling register R1, R2, secret S-box and

finally memory. By this way, its very difficult to estimate the clocking of data generating registers since it depends on secret and dynamic functions. This secrecy comes from the operations made during initialization.

Irregularly clocked systems are resistant against many correlation, divide-conquer and algebraic attacks. However the clocking shouldn't be achieved in a simple way. The clocking function of MONO is highly nonlinear. We desired to design MONO in such a way that, clocking values of each register are very close to uniform. Therefore attacks such as using the large difference between clocking values of LFSRs are not applicable to MONO.

3.4.4. Output Boolean function

The Boolean function f is used to introduce system high nonlinearity and high linear complexity. The Boolean function of MONO is balanced, that is, the number of ones and zeros appearing in the output is the same. This property is very important. An adversary can not use output difference between ones and zeros since there is no difference. Therefore it provides good statistical properties. The algebraic degree of this function is 4. Non-sparse Boolean functions with high algebraic degree has high hardware complexity. In order to implement this function easily we chose the algebraic degree to be 4. This algebraic degree seems to be sufficient to protect the system against attacks.

f of MONO doesn't have very good correlations with respect to low degree functions. Moreover, it doesn't have any annihilator of degree less than the algebraic degree of this function. f is also not sparse.

Some recent systems use irregular decimation on outputs of data generating LFSRs or Boolean functions. For example in [39], the proposed system (DECIM V2) is a basic nonlinear filter, filtering a LFSR with a 2^{nd} degree Boolean function. This system would be very open to several attacks if irregular decimating component ABSG wouldn't have been used. ABSG basically takes the output of the Boolean function and decimates it using an efficient rule. Therefore system became very resistant against correlation and algebraic attacks basically. However usage of such system like ABSG requires buffer at the output

of this decimation scheme. The rate of the output of ABSG mechanism is irregular, therefore one needs a well designed buffer in order to guarantee a constant throughput and synchronization with the plaintext. If such a buffer is not used, the use of irregular decimation has no meaning because of the dynamic throughput.

3.4.5. The AES S-box

We have thought a non-linear system to filter some bits of clock-controlling register and to generate a parameter to be used in the dynamic clocking. We chose AES S-box, as it is very good against differential and linear cryptanalysis. Also the algebraic structure of AES S-box has not been proved. Therefore this property will increase the resistance of MONO against multivariate cryptanalysis. The structure (internal state) of AES S-box is changed during the initialization phase of MONO. After the initialization the structure of S-box becomes key-dependent, since S-box structure changed with the bits that are mixture of private key *K,* secret values *p* and *q,* and the publicly known initialization vector *IV*. This S-box can also be realizable in practice. Mobile Communication system of North America uses a stream cipher called ORYX [41]. This system uses a 16x16 S-box having internal values between 0-255 to filter LFSRs. This S-box resembles AES S-box. Therefore by giving an practical example, we can say that it can be implemented for stream ciphers.

3.4.6. Initialization

Initialization is important as it generates important parameters to be used during encryption process. An adversary achieving the initial internal states of LFSRs in MONO shouldn't be able to deduce the secret key. In order to protect the secret key from this attack we used BBS generator, to generate secret values in order to hide the secret key. The attacks against BBS requires the difficulty of solving discrete logarithms. The purpose of BBS usage is to hide the secret key with an algorithm which is cryptographically secure. An adversary having initial state values may obtain a linear relation in the initialization. Therefore instead of using a larger key such as 256 bits, it is decided to use 128 bit masked with 128 bit BBS generator output. Also after the BBS function; the generated bits are

mixed and permuted in a highly nonlinear way. As a result, MONO may be regarded as secure against attacks aiming to deduce the secret key from initial internal states of LFSRs.

4. KEYSTREAM PROPERTIES OF MONO

4.1. Keystream period

MONO has LFSRs with primitive feedback polynomials of length prime. Therefore the periods of LFSRs will be maximum for each. Additionally, periods of R2 and R3 are Mersenne prime.

It is obvious that the maximum period that can be achieved is the product of periods of each register. This upper bound is about to be achieved since the total number of clocking applied to data generating register R3 during product of periods of R1 and R2 satisfies the criterion given in [3] to have maximum possible period.

In [3], it is shown that the keystream sequences produced by a generator using clock-control register(s) to control the clocking of data generating register (GR), can have maximum period as P_z:

$$P_z = \frac{\lambda P_{GR}}{\gcd(S, P_{GR})} \tag{4.1}$$

where λ is the period of the decimating sequence, P_{GR} is the period of the data generating register and S is the total number of clocking (or the value of the decimating sequence) applied to the data generating register during one period of index of clock controlling function and "gcd" stands for greatest common divisor.

According to [3], the generator can reach the limit given in (3.1.1) if one of two conditions are satisfied:

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

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

Let S be total clocking value of data generating register during the period of the decimating sequence. It is clear that $\gcd(2^{L1}-1, 2^{L2}-1)=1$ since L2 is Mersenne prime and L2>L1. Therefore, there will be $(2^{L1}-1)(2^{L2}-1)$ different indices for clocking functions. As a result the period of the decimating sequence is $\lambda = P_1P_2 = (2^{L1}-1)(2^{L2}-1) = (2^{59}-1)(2^{61}-1)$. During one period of the decimating sequence data generating register R3 will be clocked at least $(2^{59}-1)(2^{61}-1)$ and at most $4(2^{59}-1)(2^{61}-1)$. Therefore S is in the interval $(2^{59}-1)(2^{61}-1) \le S \le 4(2^{59}-1)(2^{61}-1)$. S is not a multiple of $P_3 = 2^{127}-1$. Therefore gcd(S, P_3)=1.

Let us find the period of decimated version of register R3 , defined as P_{d3} . As gcd(S, P_3)=1, then we can write the period of decimated versions of registers as,

$$P_{d3} = \lambda P_3 \tag{4.2}$$

Let P_z be the period of the key stream sequence, then P_z will be

$$P_z = P_{d3} = \lambda P_3 = P_1P_2P_3 = (2^{59}-1)(2^{61}-1)(2^{127}-1) \approx 2^{247} \tag{4.3}$$

We see that, we may theoretically achieve maximum possible period which is $(2^{59}-1)(2^{61}-1)(2^{127}-1) \approx 2^{247}$. The period of the system is quite large, therefore one of the requirements for a stream cipher has been achieved.

4.2. Linear Complexity

Using the idea in [28], since gcd(S, P_3)=1; the period of the clock control registers becomes a multiplier in the upper bound of the linear complexity of non-uniformly decimated sequence. Also according to [21], if the decimating sequence is randomly chosen, then the probability that maximum linear complexity is obtained can be made arbitrarily close to one for appropriately chosen generator register lengths and the period of the clock control register. For MONO, the clock control register is R1 whose period is

(2^{59}-1). More important the decimating sequence has a period of $(2^{59}-1)(2^{61}-1)$. Therefore, if *LC* represents the linear complexity of the key stream sequence generated by MONO, *LC* is very likely to lower bounded by $(2^{L1}-1)(2^{L2}-1)$ $C = (2^{59}-1)(2^{61}-1)\,C$, where C represents the effect of register R3, S-box, memories Q and N, and output Boolean function in the linear complexity.

If we would like to make an estimation of upper bound on *LC*, we should consider and eliminate the non-linear operations occurred during ciphering. After a careful observation, one may note that; finding the upper bound on *LC* of stream ciphers may be defined as trying all the ways to have a multivariate or univariate estimation of total number of monomials occurring in the system. If we refer to [1]; a nonlinear combiner will have *LC* of at most the value of Boolean function combining the registers when the length of each register is substituted in the Boolean function. We will discuss four different basic example to conjecture the upper bound on *LC* of MONO.

1. Think of a nonlinear combiner composed of five LFSRs each of which have primitive polynomials of degree L_1, L_2, L_3, L_4 and L_5. The Boolean function combining the registers is $f(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 x_5 \oplus x_1 x_3 x_4 x_5$. The *LC* of this system is basically $L_1 + L_2 + L_3 + L_4 L_5 + L_1 L_3 L_4 L_5$. Since the system is regularly clocked; the only nonlinear part is the output Boolean function f. The *LC* of the combiner is equal to the number of total monomials in the equation occurring for each clock if we would like to solve the system by linearization.

2. If we discuss a nonlinear filter with a maximum length LFSR of length l and a Boolean function f with nonlinear algebraic order d, the linear complexity of the system is at most $\sum_{i=1}^{d} \binom{l}{i}$ where this value is also the maximum number of monomials occurring when we try to express the system by its internal state- output relation .

3. In the examples given above, the component LFSRs are clocked regularly. The only nonlinearities for those system are the Boolean functions used for combining and filtering. The idea behind clock controlled generators is to introduce nonlinearity into

LFSR based key stream ciphers by having the output of one LFSR control the clocking of others. If we consider the Alternating step generator, one should guess the clock control register to make a linear solution to the system. Therefore an alternating step generator with clock control register of L_1 producing a Bruijn sequence and generator maximum-length LFSRs with lengths L_2 and L_3, then the LC of the system will be at most $(L_2 + L_3).2^{L_1}$. It is quite clear that, if someone desires to make linear estimation to the system, he should first guess the internal state of R1 and then solve the linear equations produced by remaining registers.

After these observations; in order to make linear estimations for the system, one should take into account R1, R2, the position of βs, the output Boolean function, and the S-box. After guessing R1, R2, position of β values to get the memory value for each clock and the s-box, the only remaining nonlinear part of the system is the output Boolean function of degree d. To make the system linear, we should find the total number of monomials occurring with terms of R3 finally. Therefore we may conjecture that an upper bound on LC may be;

$$(2^{L1}-1).(2^{L2}-1).\,L2\,.\,A.\sum_{i}^{d}\binom{L_3}{d} \approx 2^{149}.A \qquad (4.4)$$

where d is the degree of the output Boolean function and A is the effect of S-box.

From our observations, if one considers Berlekamp-Massey attack [16] for MONO, he needs to intercept at least $2^{L1+L2+1}=2^{121}$ successive key stream bits to realize the attack in the lower bound. Before this amount of key stream is generated, registers of MONO will be reinitialized with a different initialization vector; therefore MONO is secure enough for such an attack.

4.3. Output Rate

Let $\Pr\{Cl_i = j\} \approx \frac{1}{4} = 0.25$ where $i = 2,3$ and $j = 1,2,3,4$. Cl_i is the clock of register R_i. In section 3 , we have tested that the probability given above is approximately true. Also, let h_i be the number of clock applied to register R_i at each time *t.* The average (mean) value of h_i is , $E\{h_i(t)\} = \left\{\frac{1}{2}.\frac{1}{4}(1+2+3+4) + \frac{1}{2}.\frac{1}{4}(1+2+3+4)\right\} = 2.5$. MONO generates one bit at a time, therefore the output rate will be 1/2.5=0.4. Therefore the theoretical value of output rate is equal to the simulation results. The output rate could be increased by having multiple outputs at a time. However, in [24] it is given that, as the number of outputs per cycle increases the existence of lower degree relations increases rapidly. Therefore, this will cause MONO to be less resistant against algebraic attacks.

5. STATISTICAL TESTS ON MONO KEYSTREAM OUTPUT

We have applied some important statistical tests to MONO. FIPS140-1 [14] and FIPS140-2 [15] are applied to 1000 sequences of 20000 bits which each of them is produced by MONO. Our cipher passes FIPS140-2 in proportion of 99.9%. It is known that the security criteria of FIPS140-2 is more strict than that of FIPS140-1. MONO passes FIPS140-1 in proportion of 100%. We have tested 100 sequences of length 1 million bits with NIST test. The generated sequences successfully pass these test with a significance level of 0.01 and therefore the success rate for NIST test is 99%.

Also we have applied autocorrelation test to sequences of 20000 bits and 50000 bits. Our cipher passes autocorrelation test in proportion of 100%. We have not recognized any significant correlation between the tested sequences and the shifted versions.

We have also applied spectral test to our cipher. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 %.

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

Figure 5.1 shows the autocorrelation result for 20000 bits of output of MONO. The autocorrelation (AC) values are normalized to the value at the origin. That is; the maximum AC that can be achieved is 1 which is represented by dashed red line in Figure 5. We can see that there is no significant peak compared to the normalized value (i.e one) at the origin.

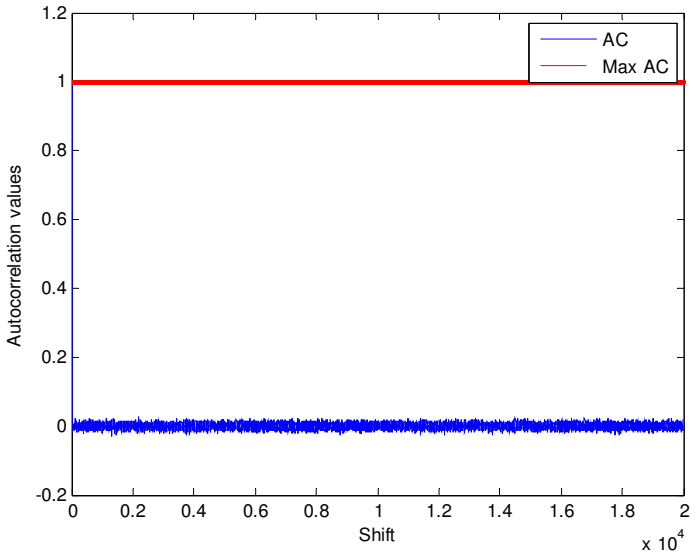

Figure 5.1. Autocorrelation test result for 20000 bits

Figure 5.2 shows autocorrelation values for a sequence of length 50000 bits. Here we can see that there is no significant peak therefore no significant correlation for the shifted values of the sequence compared to the non-shifted sequence at the origin

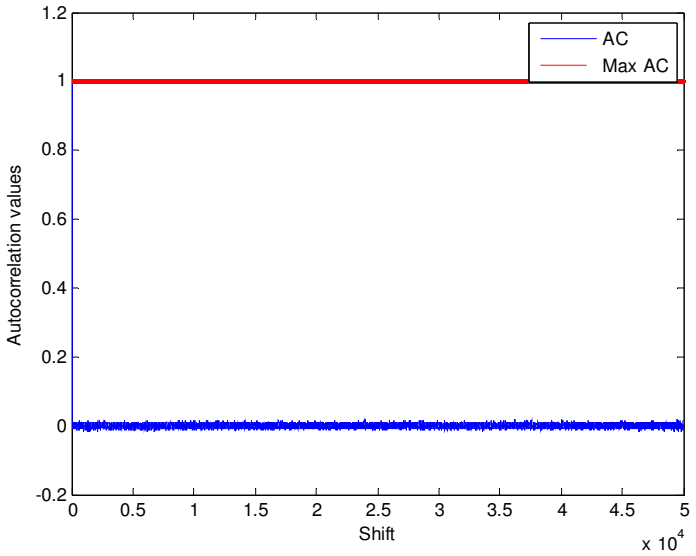

Figure 5.2. Autocorrelation test result for 50000 bits

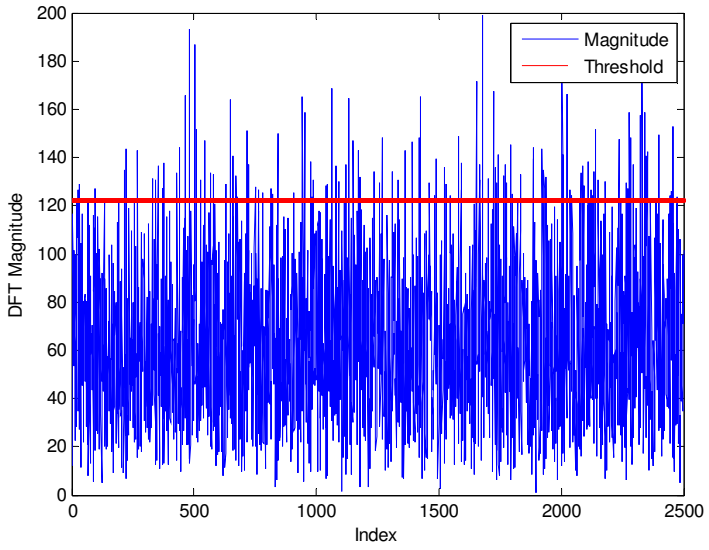

Figure 5.3 Spectral test result for 5000 bits. Threshold is 122.47

Figure 5.3 shows the spectral test result for MONO. The test is applied to a sequence of 5000 bits and its spectral values are observed. The spectral values are calculated using Discrete Fourier Transform (DFT). Since DFT is symmetric, first N/2 values are considered for a sequence of length N. For each sequence a threshold value is needed to be considered.

The threshold value is $\sqrt{3N}$ for a sequence of length N . For the sequence in Figure 6 , the threshold values is 122.47. The sequence passes the spectral test since, no more than 5% of the peaks surpass the threshold value. Figure 5.4 shows spectral test for a sequence of length 20000 bits. The threshold value is 244.95. The sequence passes the spectral test since, no more than 5% of the peaks surpass the threshold value.

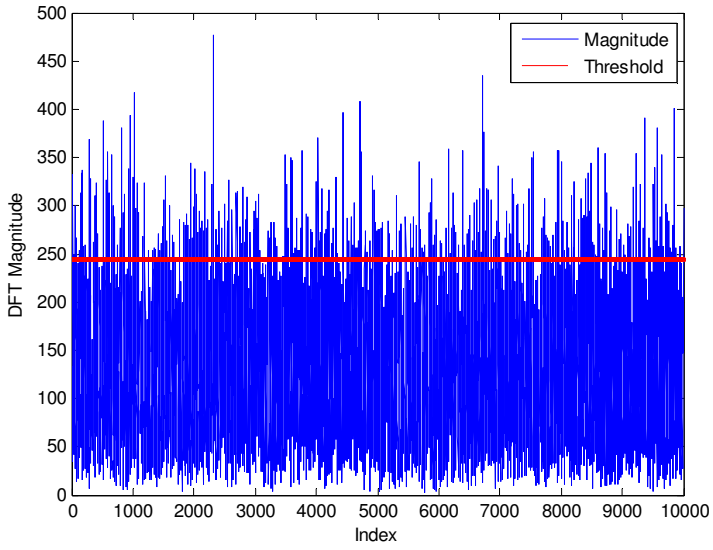

Figure 5.4. Spectral test result for 20000 bits. Threshold is 244.95

Figure 5.5 shows spectral test for a sequence of length 50000 bits. The threshold value is 387.30. The sequence passes the spectral test since, no more than 5% of the peaks surpass the threshold value

In order to give more understandable results on spectral test of MONO, we have tested again different sequences with different lengths from 5000 bits to 1 Megabits. Table 5.1 gives the results on the spectral test. The p-value is the critical parameter in spectral test. If the calculated p-value for each sequence is greater than or equal to 0.01 then the tested sequence passes the spectral test.

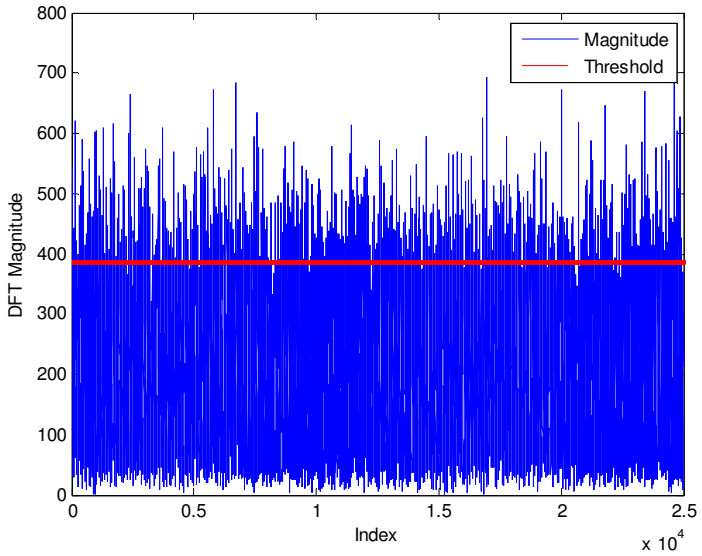

Figure 5.5. Spectral test result for 50000 bits. Threshold is 387.30

Table 5.1. Spectral test results of sequences of different lengths

Sequence length (bits)	p-value	Test Result
5000	0.2708	PASSED
10000	0.0692	PASSED
20000	0.6138	PASSED
50000	0.8390	PASSED
100 K	0.4002	PASSED
500 K	0.3985	PASSED
1 Mega	0.5857	PASSED

The linear complexity is an important value for pseudo-random bit generators since it determines the linear equivalence of a system with a known successive keystream of length twice the linear complexity. The Berlekamp-Massey algorithm described in [16] is an efficient algorithm to determine the linear complexity profile thus the linear complexity by having only some of consecutive keystream bits. In [1], it is said that for a pseudo-random bit generator the linear complexity profile should be close to the line L=N/2 where N is the length of the sequence. In Figure 5.6, the linear complexity profile for 20000 bit of

MONO stream cipher is given. It is seen that linear complexity (*LC*) profile is very close to the line L=N/2. In fact *LC* follows the line L=N/2. As it can be seen from the figure dashed red line is the L=N/2 line and part by part we can see some blue points which are the *LC* profile values. So *LC* profile is very very close the requirements.

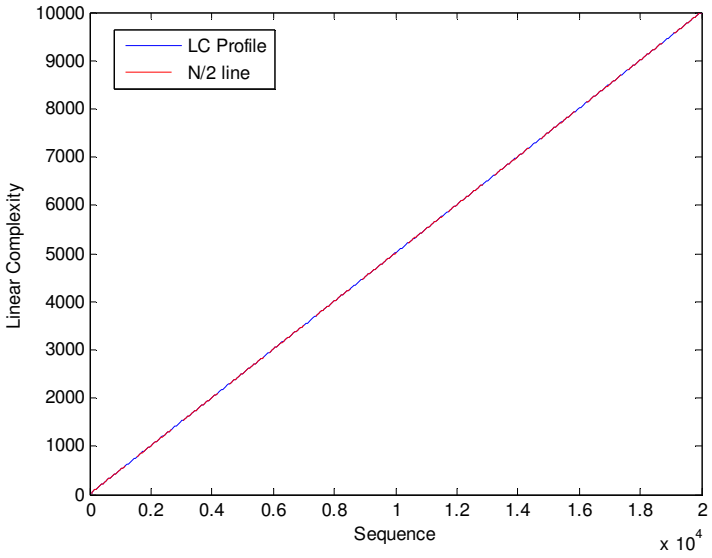

Figure 5.6. Linear complexity profile of MONO

We should also pay attention to the value of the memory term Q . From its definition this term have values between 1 and 64. For each cycle of the cipher the memory is updated. If the value obtained for each cycle is supposed to be equally likely random, then the memory bits should approximately be distributed uniformly. Therefore we may define the probability of the each memory value as ,

$$\Pr\{Q = x\} = \frac{1}{64} = 0.015625 \quad \text{where } x = 1, 2, ..., 64 \tag{5.1}$$

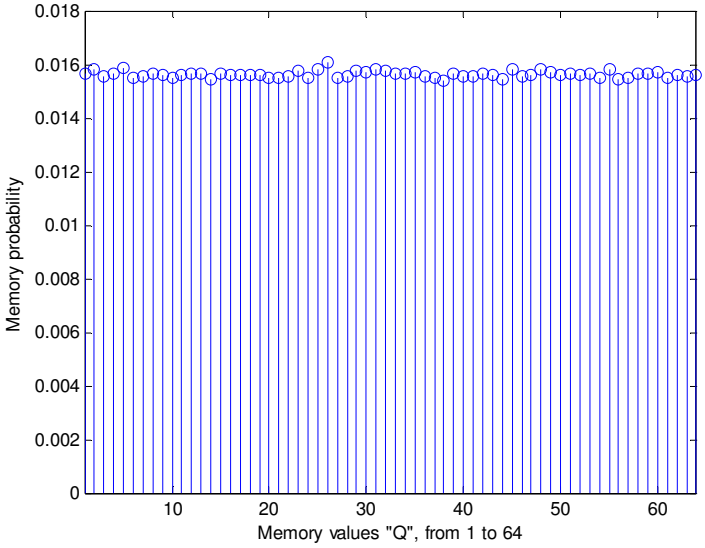

Figure 5.7. Memory distribution of MONO for 1 Megabits

Figure 5.7 shows the memory distribution of MONO. We can see that the memory percentage for values from 1 to 64 is close to the probability value given before (i.e. 0.015625). Memory value distribution is tested over 1 Mega bit sequence. We hope that distribution will be more uniform as the sequence grows. All of the memory values are used approximately in a uniform fashion. Therefore we can state the data generating sequence R3 is multiplexed in a uniform fashion, as we had desired in the design phase.

For clock-control unit, we expect each clocking value Cl_i have values 1, 2, 3 and 4 uniformly. Therefore we define the probability of clock values as,

$$\Pr\{Cl_i = j\} \approx \frac{1}{4} = 0.25 \quad \text{where } i = 2,3 \text{ and } j = 1,2,3,4 \tag{5.2}$$

Again we tested distribution of clocking values of R2 and R3 for 1 Mega bits. Results are shown in table below,

Table 5.2. Clocking distribution of R2 and R3 for 1 Megabits

	Clock Value	Occurence	% Proportion in 1 Megabits	Approximity to Uniformity
Register 2	1	254454	25,4454	98,22%
	2	246519	24,6519	98,61%
	3	245887	24,5887	98,35%
	4	253140	25,314	98,74%
Register 3	1	253683	25,3683	98,53%
	2	246120	24,612	98,45%
	3	247920	24,792	99,17%
	4	252277	25,2277	99,09%

We can see from the observed clock value distributions that clocking times of R2 and R3 have values 1,2,3 and 4 close to uniform. This observation shows that an adversary couldn't use the clocking distribution information as the distribution is very close to uniform.

6. SECURITY ANALYSIS

6.1. Exhaustive Key Search

The proposed cipher uses 128 bit secret key. Additionally, an opponent should have session values p and q in order to recover secret key K. The total amount of secret keys and B values are 2^{256} for each. Therefore on needs 2^{256} trials to reach the secret key which seems to be impractical.

6.2. Divide and Conquer Attacks

In this attack, attacker guesses the internal state of some registers and finds the remaining bits using this guess [8]. Since the proposed cipher is a type of clock-controlled stream cipher, the attacker would want to guess the internal of the registers contributing clock-control. For this cipher, the clock-control is not only dependent on R1 but also R2, S-box and memory N. One needs to guess the internal structure of R1 and R2. This results in a complexity of $2^{(59+61)}=2^{120}$. Guessing R1 and R2 is not sufficient to get knowledge about the memory and clocking values. One should also guess the position of β values. Also one needs to determine the structure of S-box to get back to R1. We should also take into account the multiplexing, the contribution of output Boolean function to the output. Therefore this type of attack does not seem to be practical for this cipher.

6.3. Time Memory Tradeoff Attacks

According to the improved time-memory trade-off attack of Biryukov and Shamir [7]; trade-off relation is given by $T_{attack}M^2D_{tot}^2=N_s^2$ where T_{attack} is the attack time, D_{tot} is the total amount of known keystream sequence, N_s is the state space and M is the memory for any $D_{tot}^2 \le T_{attack} \le N_s$. We know that time-memory trade off attacks can be practical if the state space of the stream cipher is small. One should note that this kind of attacks do not seem to be applicable to MONO, since the solution space is sufficiently large (2^{247}). Suppose that $M=2^{55}$ (4 Pentabytes), and for this algorithm the state space $N_s=2^{247}$. Then $T_{attack}D_{tot}^2=2^{384}$ and this shows that attacker should waste several years to be successful.

6.4. Correlation Attacks

A stream cipher should be resistant against many important attacks. The family of correlation attacks become serious threat against stream ciphers using regularly clocked LFSRs and non-linear filters or combiners. The basics of these attacks to find a low degree approximation to Boolean functions using some probabilistic information and also some of them tries to find correlation between the LFSRs used in the cipher and the output.

MONO stream cipher uses irregular clocking with the help of its internal memory property. Also at the output, a 3-resilient Boolean function is used. Therefore, we should pay attention to correlation attacks of the type; Higher Order Correlation Attack using Low Degree Approximations [18] since MONO has a Boolean function and constrained embedding , unconstrained embedding, edit distance and edit probability correlation attacks [30,31,32] since the system uses irregular clocking.

6.4.1. Higher Order Correlation Attack with Low Degree Approximation

In [18], a Higher Degree Correlation Attack is applied for the Toyocrypt stream cipher [20]. Toyocrypt stream cipher uses a regularly clocked LFSR followed by filtering with a 63rd degree Boolean function. The Boolean function used in the system has a few higher order monomials, one of degree 4, one of degree 17 and one of degree 63. And everything else is quadratic. Author of [18], made a low degree approximation of the Boolean with a probability *p*. After these approximation, using known keystream of amount m , a lower degree multivariate relation has been established. However, m shouldn't exceed the sufficient amount such that p^m is less than $\frac{1}{2}$. If we need such amount of keystream than the whole attack should be repeated about p^{-m} times to have a possibility to be successful in attacking the system. Below we describe the actual attack.

Given m bits of the keystream b, we have the following m equations to solve :

$$\forall_i = 1...m, \qquad b_{t_i} = f(L^{t_i}(k_0, ..., k_{n-1})) \tag{6.1}$$

f is the non-linear function (including irregular clocking, S-box and memory), L^{t_i} is the linear state transition of LFSRs which are all publicly known and k_j is the initial state of the LFSRs. For MONO, it is a challenging task to find a relation as in (5.4.1.1), since MONO has dynamic clocking, memory and secret s-box. Therefore an adversary should make approximations and sum guesses in order to convert the system into a regular and less complex form.

An adversary needs to make a low degree approximation of the output function. It is not enough to give a solution to the initial internal state since the clocking is irregular and depends on secret structured S-box and the memory of the previous clock. Therefore one also needs to guess the structure of S-box, initial value of R2, the value of βs and the clock control register R1. The remaining is to solve R3 with low degree approximation of output Boolean function. We recall that the Boolean function of MONO is

$$x_1 + x_2 + x_3 + \sum_{4\le i\le j\le 8} x_i x_j + \sum_{4\le i\le j\le k\le 8} x_i x_j x_k + \sum_{4\le i\le j\le k\le l\le 8} x_i x_j x_k x_l \qquad (6.2)$$

The best approach to this function is linear with probability $p=\frac{9}{16}$ and given below;

$$x_1 + x_2 + x_3 + x_4 \qquad (6.3)$$

After guess of required parts, the function will be a linear function of four terms from R3, $x_1 + x_2 + x_3 + x_4$. A lucky adversary needs at least $L_3 = 127$ keystream bits (It is possible that needed keystream is higher, because of dynamic multiplexing of R3 an adversary may come across the same initial state for a different keystream bit. However, we should consider the worst case) to get the initial value of the registers R3. For 127 bits the probability will decrease to $\left(\frac{9}{16}\right)^m = \left(\frac{9}{16}\right)^{127} = 1.84\times 10^{-32}$ which is much less than $\frac{1}{2}$. Therefore we need to repeat the whole attack about $\left(\frac{9}{16}\right)^{-m} = \left(\frac{9}{16}\right)^{-127}$ times to get a probable solution of R3 .

We know that solving M linearly independent equations using Gaussian Elimination will have a work load about $O(M^3)$. Therefore the higher order correlation attack will have a complexity about; $2^{(80+59+61)}.61.\left(\frac{9}{16}\right)^{-127}.(127)^3 \approx 2^{332}$ using only 127 keystream bits. Let us explain why we have this result. The structure of S-box is changed with 80 bits during the initialization part, this gives a 2^{80} complexity. Guessing initial state of R1 and R2 will give $2^{(59+61)}$ complexity. The guess of R2 will not be sufficient to get the five β s to know the values of the memory Q and N. Because the position of β s are shifted with σ in the initialization. Therefore we need to guess these six consecutive β s with a work load of $L_2 = 61$. Other terms in the complexity is the probability and the complexity of solving a 127x127 linear system. The complexity of this attack (2^{332}) is extremely higher than the exhaustive search of initial internal states of LFSRs which is $2^{(59+61+127)} = 2^{247}$. Therefore we see that, this type of attack is not useful against MONO.

6.4.2. Higher Order Correlation Attack with Low Degree Approximation and Sequential Clocking

The guess of the structure of S-box, R1 and R2 will make any of the attack infeasible. The previous attack needs very small value of keystream however, the complexity of the attack becomes higher than the exhaustive search of initial states.

In [19] a different approach for cryptanalysis of LILI keystream generator [21] is given. LILI-128 uses a clock control register of length 39 controlling the decimation of generator of length 89 filtered by a Boolean function of degree 6. Instead of guessing clock control register to get clocking information, it is suggested in [19] that, clocking the clock control register 2^{39}-1 steps at a time will give the information about how many times the generator register is clocked during one period of clock controlling register. There are 2^{39}-1 values in the index of clocking function therefore an adversary will know how many times the generator register is clocked for a particular (2^{39}-1).m th observed keystream which is indeed S=(5.2^{38})-1. Therefore if an adversary needs a total of T keystream bits in order to break the system, then he will need to observe (2^{39}-1).T consecutive keystream bits.

When we look for the feasibility of such an attack to MONO, an adversary will get information when the register R1 is clocked $(2^{59}-1).(2^{61}-1) \approx 2^{120}$ times since the total number of values in the index of clocking function is $(2^{59}-1).(2^{61}-1)$. Therefore an adversary will know how many times R3 is clocked if R1 is clocked $(2^{59}-1).(2^{61}-1)$ steps at a time. This operation is not sufficient to make linear, correlation and algebraic attacks to the system. One also needs to find which tap position of R3 is used for the particular $(2^{59}-1).(2^{61}-1)m$ th keystream. This is same as knowing the memory value. However in order to get information about the memory value Q , an adversary needs to know the structure of S-box, R2 and R1 then the remaining thing is the cryptanalysis of R3. We see that this attack worse than the previous attack. Since the complexity will approximately be the same but required consecutive keystream is prohibitively large which is multiple of $(2^{59}-1).(2^{61}-1)$.

This amount of consecutive keystream bits do not seem to be observable. Let us explain the difficulty of observing such amount of consecutive data. One of the highest bit rate required area is television broadcasting . The new age of television technology uses high definition digital video to provide customers high quality vision service. Of course this technology uses data compressing techniques such as MPEG-2 and useful modulation techniques to efficiently allocate the transmission media, however we will use uncompressed data in order to determine the size of required keystream in this attack. Uncompressed high definition digital video; HD-SDI; (High Definition Serial Data Interface) has the bit rate of 1.485 Gigabps. Suppose that this content is encrypted using MONO and transmitted via a fiber channel. Then an adversary needs to capture this consecutive content for about 2.64×10^{19} years to be successful using this attack. Here we see that this attack is not feasible. Therefore we can state that sequential clocking technique used in [19,23] is not applicable to MONO.

6.4.3. Constrained and Unconstrained Embedding; Edit Distance and Edit Probability Attacks

Unconstrained embedding attack described in [30], tries to embed the known keystream Z of length n into a string of length m of X which is the sequence generated by the initial state of the generator register for $m \geq n$. The attack is suppose to be successful,

if the deletion rate P_d is smaller than $\frac{1}{2}$. The deletion rate is related to the average number of clocking occurred for a clock-controlled stream cipher to produce one keystream bit. For MONO, this value is calculated as $E\{h_i(t)\}$ in section 3.3. The deletion rate is defined to be ;

$$P_d = 1 - \frac{1}{E\{h_i(t)\}} \tag{6.4}$$

Therefore, the deletion rate for MONO becomes,

$$P_d = 1 - \frac{1}{E\{h_i(t)\}} = 1 - \frac{1}{2.5} = 0.6 \tag{6.5}$$

If we recall, the attack will be successful, if the deletion rate is smaller than $\frac{1}{2}$. Since the deletion rate of MONO is greater than this value, such an attack won't be successful against MONO stream cipher.

Constrained embedding attack described in [30], uses a different parameter d_{max} in order to embed the observed keystream Z into string X. It is shown that, in order to be successful using constrained embedding attack, an adversary should observe keystream sequence of length greater than a value linear in the generator length and super exponential in d_{max}. The needed observed sequence for successful attack is determined by an approximate formula given in [30];

$$n \geq r \; 2^{(d_{\max}+2)(1+2^{d_{\max}+2})} \ln 2 \tag{6.6}$$

Here r is supposed to be the length of the generator registers. For MONO $d_{max} = max(h_i(t)) = 4$, as a result needed keystream sequence will be extremely large making MONO secure against this attack.

Edit distance and edit probability attacks given in [31] and [32] are the attack schemes which consider the initial state of the data generating register which can be thought as exhaustive search over all possible states of this register. R3 is the generating register of MONO. In order to get the possible initial state of R3 an adversary calculates the correlation between the regularly and irregularly decimated sequences. He tries this correlation for all possible states of R3. However again the adversary will need the knowledge of Q in order to know which tap value is used for each clock, as the tap values used in R3 changes with the dynamic value Q . Therefore in order to get this information, the attacker will need another correlation attack or just exhaustive search on R2, but it is still insufficient. The knowledge of R1 and S-box is needed additionally. We see that this type of correlation attacks are also infeasible for MONO.

6.5. Algebraic Attacks

Algebraic attacks are a new family of cryptographic technique based on defining a cipher system by a initial state-output relation of some degree d . If one can find a relation for a system then he can theoretically find the initial state of the cipher by solving multivariate equations with some or may be a huge amount of known consecutive keystream. In algebraic attacks, the aim is to recover the initial internal state . It is assumed that an adversary has knowledge of the algorithm and some keystream bits.

For some ciphers algebraic attacks outmatched all previously known attacks since it decreases the linear complexity of the cipher. In [18], Nicolas T. Courtois presented the first algebraic technique against the Toyocrypt [20] cipher. In [19], Courtois and Meier presented new algebraic attacks against Toyocrypt and LILI-128 [21]. Frederik Armknecht proposed an algebraic attack against the Bluetooth Key Stream Generator (E0) in [22]. He had successfully described the E0 system in terms of internal state-output relation with a degree of d=4. In [23] Nicolas T. Courtois proposed an Fast Algebraic Attack against stream ciphers using the lower degree terms of relation equation however with more keystream bits which also should be consecutive.

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

1. Set up a system of equations in the unknowns (initial internal state of the cipher).

2. Insert the observed keystream bits into the identifiers z_t.

3. Recover the unknowns by solving the resulting system of equations using linearization, re-linearization or XL algorithm (or another useful multivariate equation solver).

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

In [22] and [37], the author made an algebraic attack against the E0 stream cipher used in Bluetooth Wireless interface. E0 is a combiner with memory. He had successfully determined a 4^{th} degree internal state-output relation free of the memory bits and claimed that the system can be solved in $2^{68.48}$ complexity using about $2^{23.07}$ non-consecutive keystream bits. Since E0 has a internal space of total size 2^{128} , this attack shows that the design behind the E0 is poor.

In [33], authors gave a general result on existence of internal state-output relations for summation generator. The summation generator is a nonlinear combiner with memory. System uses k LFSRs and combine them with the memory information calculated using the previous keystream output bit. It is shown that for a summation generator that uses k LFSRs, an algebraic equation relating the keystream bits and LFSR output bits can be made of degree less than or equal to $2^{\lceil \log_2 k \rceil}$, using $\lceil \log_2 k \rceil + 1$ consecutive keystream bits.

This is much lower than the upper bound on the degree of algebraic equations that is guaranteed by the general works [24,37]. Results given in [33], agrees with the attack given in [22,37], since E0 uses 4 LFSRs and a 4^{th} degree equation has been found for this cipher.

In [34], authors proposed algebraic attacks for several clock-controlled stream cipher designs such as Stop and Go Generator and Alternating Step Generator. However these examples are very simple designs and guessing the clock-control register makes all the operations linear. That is, it easy very easy to establish an algebraic relation after guessing the clock-control register. In [38], an algebraic attack against A5/2 stream cipher has been proposed. A5/2 is the weak version of A5/1 used in GSM encryption. A5/2 uses clock-control register to control the clocking of other three generator registers. The clocking also depends on the internal state of the control register only. Therefore by only guessing the clock control register it easy to establish a quadratic algebraic relation between remaining registers and keystream output hence solve this system by using only several hundreds of keystream information.

In [24], algebraic attacks against well known two stream ciphers has been proposed. These ciphers are LILI keystream generator [21] and the Toyocrypt [20]. LILI keystream generator uses two LFSRs. One of them is used to clock another . And at the end of the data generating register, there is a 6^{th} degree nonlinear Boolean function which filters the internal bits of this register. The design is weak against algebraic attack, since the clocking can be vanished by guessing the initial state or clocking several steps at a time. Therefore the remaining thing is to solve the equations provided by the Boolean function. Also, the Boolean function has not been designed well, so the degree of this function has reduced by multiplying this function by well chosen polynomials. Therefore the algebraic immunity (*AI*) of Boolean functions is an important measure. Algebraic immunity of degree d , guarantees the nonexistence of Annihilators of degree $\leq d$. This fact and the existence of Annihilators will be discussed in the section 5.5.2. On the other hand, Toyocrypt stream cipher uses a regularly clocked LFSR followed by filtering with a 63^{rd} degree Boolean function. The Boolean function used in the system has a few higher order monomials, one of degree 4, one of degree 17 and one of degree 63. And everything else is quadratic. In the same way as the attack applied on LILI, the degree of Boolean

function used in Toyocrypt is reduced by multiplying it by well chosen polynomials. Therefore the attack complexity is reduced. The additional advantage of this attack is that it does not require keystream bits that are consecutive.

In [23], a fast version of the algebraic attacks has been proposed. The purpose of this attack is to lower the total complexity of the algebraic attacks by using only the lower degree equations instead of higher ones occurring in the algebraic relation. By decreasing the total number of monomials occurring in the linear system to be solved, the attack also decreases the required memory to store the established equations. Attack cancels all the higher degree terms in algebraic relation in a pre-computation step. In this step, attacker finds a linear dependency on higher degree terms and applies it in the solution step of lower degree terms. The disadvantage of this attacks is that, it requires consecutive keystream bits and a high keystream substitution complexity which will be discussed in the later sections.

The second step of algebraic attacks is the substitution of observed keystream bits into the equations derived in the first section. Normal algebraic attacks have substitution complexity which is less than the actual attack. However, fast algebraic attacks have substitution complexity higher than the actual attack if the substitution is done in a naïve manner. The fast algebraic attack described in [23], has underestimated the total substitution complexity. It is given in [35] that the substitution complexity exceeds the actual fast algebraic attack if this substitution is done in a simple way. Therefore there is no meaning of attacking in this situation. In [35], a new way of substituting the keystream bits into the derived algebraic equations has been proposed. The author used Fast Fourier Transform (FFT) to reduce the substitution complexity under the actual attack complexity.

The third step of algebraic attacks is solving the multivariate equations obtained in step 1. Solving multivariate equations is still an open research problem. Many techniques has been proposed to solve these systems. The best known and simple technique is the linearization. The basis of this technique is to “linearize” a system of nonlinear algebraic equations by assigning a new unknown variable to each monomial term that appears in the system. The same monomial term appearing in distinct equations is assigned the same new unknown variable. The system of equations then changes from a system of non-linear

equations (with few unknown variables) into a system of linear equations (with a large number of unknown variables). If the number of linear equations exceeds the number of new unknown variables, then the attacker can solve the system to obtain the new unknown variables of the linear system (which will in turn reveal the unknown variables of the non-linear system). The question is what to do when sufficient number of equations therefore keystreams are not available. A number of techniques has been proposed to solve a system of multivariate equations when the available keystream is less than the total monomials appearing in the system. In fact most important thing is producing linear independent equations. In [26], the relinearization technique has been produced. We will not go into detail of this technique since this technique cannot solve systems of high degree, because high degree relinearization leads into many linearly dependent equations. A better technique called XL (Extension and Linearization) has been proposed in [27] and extended in [18]. XL algorithm is an efficient tool to solve overdefined systems of multivariate equations. The algorithm basically multiplies all equations with monomials of degree less than a value and produce additional equations from the initial equations. By this way, a multivariate system can be solved even if the number of keystreams observed is less than the total number of monomials. The XL algorithm is described in section 5.5.1.

In section 6.5.1, the XL algorithm is given. We use this technique in order to solve systems of multivariate equations. We have also simulated the results of XL algorithm which gives important results. In section 6.5.2, we discussed the existence of a algebraic relation for MONO and its Boolean function. Therefore we touched upon Annihilator sets and the notion of algebraic immunity (*AI*). We have also made a simulation and tested if there is a Annihilator of degree less than the degree of Boolean function of MONO. In section 6.5.3, we made an algebraic attack against MONO using linearization. In section 6.5.4, we have applied XL algorithm to solve the system. In Section 6.5.5, we applied XL algorithm with higher degree in order to reduce the required keystream. In section 6.5.6, we have discussed fast algebraic attacks and applied a fast algebraic against MONO. In section 6.5.7 we have touched upon general results of algebraic attacks applied against MONO and in section 6.5.8 given some comments on resistance against algebraic attacks and applicability of algebraic attack resistant systems.

6.5.1 The XL Algorithm

The XL algorithm given in [18,27], is an efficient algorithm for solving systems of multivariate equations. Its most important property is that it generally reduces the needed keystream bits to solve equations.

Let $l_i = (x_0, ..., x_{n-1}) = 0$ be the initial m equations derived for m keystream bit observations, $i = 1....m$ with n variables $x_i \in GF(2)$. And let D be the parameter of the XL algorithm. The XL algorithm consists of multiplying both sides of these equations by products of variables. XL executes the following steps:

1. **Multiply:** Generate all the products $\prod_{j=1}^{k} x_{i_j} . l_i$ with $k \leq D - d$, so that the total degree in the x_i of these equations is $\leq D$.
2. **Linearize:** Consider each monomial in the x_i of degree $\leq D$ as a new variable and apply Gaussian elimination on the equations obtained in 1. The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last.
3. **Get a Simpler Equation:** Assume that step 2 yields at least one univariate equation in the powers of x_1. Solve this equation over the finite field (e.g., with Berlekamp's algorithm).
4. **Final step:** It shouldn't be necessary to repeat the whole process. Once the value x_1 is known, we expect that all the other variables will be obtained from the same linear system.

It is expected to find one solution to the system, the complexity of XL will be essentially the complexity of one Gaussian reduction in step 2. The XL algorithm consists of multiplying the initial m equations l_i by all possible monomials of degree up to D-d (d is the degree of initial equations), so that the total degree of resulting equations is D. Let R be the number of equations generated in XL, and T be the number of all monomials. We have,

$$R = m.\left(\sum_{i=0}^{D-d}\binom{n}{i}\right) \tag{6.7}$$

$$T = \sum_{i=0}^{D}\binom{n}{i} \tag{6.8}$$

The main problem in XL algorithm is that in practice not all the equations generated are independent. Let *Free* be the exact number of equations that are linearly independent in XL. We have $Free \leq R$. We also have necessarily $Free \leq T$. The main idea behind XL is the following: it can be seen that for some *D* we have always $R \geq T$. Then it is expected that $Free \approx T$, as obviously it cannot be bigger than *T*. More precisely, following [27], when $Free > T - D$, it is possible by Gaussian elimination, to obtain one equation in only one variable, and XL will work. Otherwise we will need a bigger *D*, which will increase the complexity of the attack.

The fastest algorithm for Gaussian reduction operations is Strassen's algorithm [25] that requires about $7.T^{\log_2 7}$ operations for a *T*x*T* matrix.

6.5.2 Existence of algebraic relations, Algebraic Immunity, Annihilators

In [19], it is stated that that there should be no non-trivial multivariate relations of low degree that relate the key-bits and one or many output bits of the cipher. Otherwise, if only one such multivariate relation exists (for any reason), an algebraic attack will exist. If for one state we are able, by some method, to deduce from the output bits, only one multivariate equation of low degree in the state bits, then the same can be done for many other states.

Therefore existence of such algebraic equations will lead to successful algebraic equations. We should pay attention if such an relation exists for MONO. MONO stream cipher uses secret structured S-box, dynamically clock controlled registers and memory value calculated with secret stages of R2 and S-box. Therefore structure of S-box and memory stages can be regarded as key dependent variables and changes for different

sessions. The output Boolean function has a algebraic form but the inputs used in this function are not regularly clocked, also the input taps of the Boolean function changes for every clock of the system in addition to irregular clocking. Because of the described reasons it is very difficult for an adversary to find a algebraic relation only consisting of LFSR internal states and the key stream output for any time and any clock of the system. In [24], a general criterion on the degree of such relations for combiners with memory and several outputs is given. The general criterion is that, for any combiner with k inputs, l bits of memory and m outputs there is a algebraic relation of degree $d = \left\lceil \frac{kM}{2} \right\rceil$ where $M = \left\lceil \frac{(l+1)}{m} \right\rceil$. However, we should state that this bound is valid for systems using regular clock-control and memory in the output function. The use of memory in the output function will of course increase linear complexity and correlation immunity however the same fact leads to a memory free algebraic relation between internal states and output bits. Examples are given in [33,37]. For MONO, it is difficult to determine such a degree. The memory value of MONO is not used in the output Boolean function, but used for dynamic clocking and multiplexing.

As MONO uses 4th degree Boolean function for output generation, it would be useful if the degree of this function is reduced by multiplying the function with well chosen polynomials of any degree. In principle, the degree of Boolean function used in a stream cipher shouldn't be decreased by well known methods. This fact leads to algebraic immunity (*AI*). The algebraic immunity (*AI*) of a Boolean function f is the minimum value of d such that f or $f+1$ admits an annihilating function of degree d [36] . The annihilator of f is a nonzero function g such that $f*g=0$.

It is given in [36] that, for any Boolean function with k inputs there is a Boolean function $g \neq 0$ of degree at most $\left\lceil \frac{k}{2} \right\rceil$ such that $f*g$ is of degree at most $\left\lceil \frac{k}{2} \right\rceil$. MONO has output Boolean function of degree d=4 and has k=8 inputs. Therefore it would be possible to find an annihilator of that degree when multiplied by f will result in a function of degree d. But this would not reduce the degree of original equation, it will still be 4. Therefore we

need to look for existence of annihilators of degree $d \le 3$. In [36], an algorithm to find existence of low degree annihilators is given. The algorithm is as follows ;

1. Substitute all N arguments x with $f(x) = 1$ in the algebraic normal form of a general Boolean function $g(x)$ of degree *d*. This gives a system of *N* linear equations for the coefficients of $g(x)$.

2. Solve this linear system.

3. If there is no (non-trivial) solution, output "no annihilator of degree *d*", else determine set of coefficients for linearly independent annihilators.

We have made a simulation of this algorithm and looked for existence of annihilators of degree $d \le 3$ for MONO output Boolean function. There is no annihilator of f or f+1 of degree $d \le 3$. Therefore the degree of MONO output Boolean function can not be reduced. The algebraic immunity of MONO Boolean function is therefore 4.

6.5.3. Algebraic Attack against MONO using Linearization

In order to apply an algebraic attack on MONO, the clocking and dynamic multiplexing, βs, structure of S-box should be determined (guessed). After these guesses the remaining part is the 4th degree function consisting of variables from R3. First of all an adversary should make guesses of amount $2^{(80+59+61)}.61 \approx 2^{205.9}$. The remaining part can be solved with $T = \sum_{i=0}^{4} \binom{L_3}{i} = 2^{23.3}$ keystreams such that corresponding equations are all linearly independent. Then this linear system of TxT can be solved with complexity $7.T^{\log_2 7} = 2^{68.3}$ with required memory of $T^2 = 2^{46.6}$ bits. The total complexity of the attack is $2^{(205.9+68.3)} = 2^{274.2}$. The complexity is much more higher than the exhaustive key search over initial states which is 2^{247}. The question is if all of the observed keystream bits leads to linearly independent equations. Of course it doesn't. It is obvious that we need more keystream to observe and if we haven't got any chance to increase the number of observed keystreams, we will use XL algorithm to solve the system with less keystream.

6.5.4. Algebraic Attack against MONO using XL Algorithm

We have made a simulation of estimation on XL results. XL algorithm fails with the number of keystream outputs available in the section 5.5.3. The reason is that the number of linearly independent equations is much less than the number of total monomials. The XL algorithm works with at least $2^{23.5}$ keystream bit instead of $2^{23.3}$ given in the previous section. The simulation has been made for equation of degree at most $D = 4$. That is the XL parameter is $D = 4$. Therefore the attack complexity and obviously the required memory is same as the previous section but the we need additional $2^{20.5}$ keystream bits since $2^{23.3}+2^{20.5} \approx 2^{23.5}$. In order to reduce the required keystream bits we should increase the degree of XL parameter *D*. This is given in the following section.

6.5.5. Algebraic Attack against MONO using XL Algorithm with less keystream bits

In order to reduce required keystream bits, we need to increase *D*. For D=4 the minimum required keystream was $2^{23.5}$. When we increase D to 5 then the required keystream bit will decrease to 2^{21}. However, the complexity of the attack will increase. The total number of monomials in the system will be $T = \sum_{i=0}^{D=5} \binom{L_3}{\mathrm{i}} = 2^{28}$ and therefore the attack complexity will be $7.T^{\log_2 7} = 2^{81.3}$ with about $T^2 = 2^{56}$ bits of memory which equals 8 Petabytes. The total attack complexity will be $2^{(205.9+81.3)} = 2^{287.2}$ which is much worse than the previous attack.

6.5.6. Fast Algebraic Attacks

In [23], author introduced the fast version of algebraic attacks. The purpose of this attack is to lower the total complexity of the algebraic attacks by using only the lower degree equations instead of higher ones occurring in the algebraic relation. By decreasing the total number of monomials occurring in the linear system to be solved, the attack also decreases the required memory to store the established equations. Attack cancels all the higher degree terms in algebraic relation in a pre-computation step. In this step, attacker

finds a linear dependency on higher degree terms and applies it in the solution step of lower degree terms. The disadvantage of this attacks is that, it requires consecutive keystream bits and a high keystream substitution complexity if it is done in a naïve way.

In fast algebraic attacks, the attacker tries to find Double-decker equations in the multivariate system. The Double-decker equations of the type $K^d \cup K^e B^f$ will be useful for algebraic attacks. *K* represents the monomials consisting of only the initial state terms and *B* represents the observed keystream. The fast algebraic attacks are described below;

- We find a double-decker equation with degrees (*d*,*e*,*f*)
- Then we will eliminate all the monomials of types $k^{e+1}....k^d$ leaving only monomials of type $K^e B^f$ with a maximum degree *e* in the k_i.

Write the monomials of degree d to the left and monomials of degree $\leq e$ to the right as shown below.

$$\begin{aligned} &Left(k_0,.....,k_{n-1}) = Right(k_0,.....k_{n-1},b) \\ &. \\ &. \\ &. \\ &Left(L^t(k_0,.....,k_{n-1})) = Right(L^t(k_0,.....k_{n-1}),b_t) \end{aligned} \tag{6.9}$$

where L is the linear state function applied to the system. For at least $\sum_{i=0}^{d}\binom{n}{i}$ consecutive equations, a linear dependency α must exist. This linear dependency doesn't depend on the outputs b_i. Let δ be the size of the smallest such linear dependency α. We have $\delta \leq \sum_{i=0}^{d}\binom{n}{i}$. One dependency will be enough. Due to the recursive structure of the equations, this dependency α can be applied at any place,

$$\forall k \quad 0=\sum_{t=0}^{\delta-1}\alpha_t Left(L^t(k)) \quad \Rightarrow \quad \forall k \forall i \;\; 0=\sum_{t=0}^{\delta-1}\alpha_t Left(\delta^{t+i}(k)) \tag{6.10}$$

Here we find one α and reuse it on $\sum_{i=0}^{e}\binom{n}{i}$ successive windows of $\delta \le \sum_{i=0}^{d}\binom{n}{i}$ equations. Given 2δ bits of sequence, we choose a random key k' and will compute 2δ bits of this $c_t = Left(L^t(k'))$ for $t = 0,....,2\delta - 1$. Then we apply the well known Berlekamp-Massey algorithm to find connection polynomial of this LFSR that will be essentially α.

First of all, it is observed that all of the $L^i(k')$ can be computed in time about $O(\delta n^2)$ and even in $O(\delta n)$ if *L* is computed only of a combination of LFSRs. To recover α takes $O(\delta^2)$ computations using Berlekamp-Massey Algorithm, but it will take only $O(\delta \log_2 \delta)$ operation using improved asymptotically fast versions of the Berlekamp-Massey Algorithm.

We recall that the same linear dependency α will be used $\sum_{i=0}^{e}\binom{n}{i}$ times.

$$\forall i \qquad 0 = \sum_{t=i}^{\delta+i-1} \alpha_t Right(\delta^t(k); b_t ... b_{m+t-1}) \tag{6.11}$$

Summary of fast algebraic attacks:

1. Given about $m = \sum_{i=0}^{d}\binom{n}{i} + \sum_{i=0}^{e}\binom{n}{i}$ consecutive keystream bits.

2. We compute the linear dependency α using an improved version of the Berlekamp-Massey algorithm. Complexity is $O(\delta \log_2(\delta) + \delta n)$ steps with $\delta \approx \sum_{i=0}^{d}\binom{n}{i}$. This α is our pre-computed information. It allows given a sequence b_i of consecutive keystream bits, to recover the secret key k (the secret key is the initial internal states) by solving a system of equation of degree only $e \le d$.

3. The second step take about $O\left(\frac{TE^2}{2}+7E^{\log_2 7}\right)$ operations where $E=\sum_{i=0}^{e}\binom{n}{i}$ and $T=\sum_{i=0}^{d}\binom{n}{i}$. First part of the complexity is the keystream substitution complexity and second part is solving lower degree equations of degree *e*.

For many systems the substitution complexity exceeds the attack itself. That is mostly, $\frac{TE^2}{2}$ exceeds $7E^{\log_2 7}$. Recently, in [35], a new way of substituting the keystream bits into the derived algebraic equations; has been proposed. The author used Fast Fourier Transform (FFT) to reduce the substitution complexity under the actual attack complexity. By using FFT, the total substitution complexity is reduced to $O(2ET\log_2 T)$, therefore the total complexity becomes $O(2ET\log_2 T+7E^{\log_2 7})$.

In the following section we will give example of fast algebraic attack applied on MONO. It will use the naïve approach. That is guessing R1, R2, β s, structure of s-box and then applying algebraic attack for remaining LFSR R3 filtered with Boolean function. Before proceeding, we should say that, because of the structure of Boolean function used in MONO, it is assumed that there is a double-decker equation of the form $K^d \cup K^e B^f$ where *d*=4, *e*=3 and *f*=0.

6.5.6.1. Fast algebraic attack against MONO : In order to apply an algebraic attack on MONO, the clocking and dynamic multiplexing, β s, structure of S-box should be guessed. After these guesses the remaining part is the 4th degree function consisting of variables from R3. First of all an adversary should make guesses of amount $2^{(80+59+61)}.61 \approx 2^{205.9}$. Then system is supposed to be appropriate for fast algebraic attacks.

In the pre-computation step , the complexity will be about $O(\delta\log_2(\delta)+\delta n)=O\left(\sum_{i=0}^{4}\binom{L_3}{\mathrm{i}}\log_2\left(\sum_{i=0}^{4}\binom{L_3}{\mathrm{i}}\right)+\sum_{i=0}^{4}\binom{L_3}{\mathrm{i}}(L_3)\right)$ = $O(2^{30.58})$. Given about

$T+E = 2^{23.3}+2^{18.3}=2^{23.4}$ consecutive keystream bits, where $T=\sum_{i=0}^{4}\binom{L_3}{i}$ and $E=\sum_{i=0}^{3}\binom{L_3}{i}$. And the complexity of the fast algebraic attack is $O\left(2ET\log_2 T+7E^{\log_2 7}\right)= 2^{54.4}$. Therefore the total complexity of the attack becomes $2^{(205.9+54.4)}=2^{260.3}$. Attack requires $E^2=2^{36.6}$ bits of memory. This attack is much faster than the attack explained in 5.5.3, however it is still greater than the exhaustive keysearch over initial internal states of registers. The disadvantage of attack is the requirement of consecutive keystream bits.

Here if we want to apply XL algorithm for fast algebraic attacks, we should apply this to the terms of lower degree not of degree 4. Applying XL to 3rd degree terms will decrease E but T will still be the same. Lowering E and therefore increasing the total complexity will not decrease total required keystream and it will be at least $T \approx 2^{23.3}$. An attacker can increase the attack complexity by decreasing the required keystream in the solution part of multivariate equation of degree e=3. However the total observed keystream should still be at least $T \approx 2^{23.3}$ in any case. We see that XL algorithm does not provide any advantage in this attack.

6.5.7. General Results on Security of MONO

Table 6.1, shows the results obtained for the higher order correlation and algebraic attacks against MONO stream cipher. First column shows the section of the attack where, second column represents the minimum value of required known keystream bits to achieve the attack, third column tells whether the observed keystreams are successive or not, fourth column gives the required memory capacity to establish attacks, fifth column shows whether the attack requires pre-computation, sixth column gives the total complexity of attacks and seventh column tells if the attack depends on probabilistic approximations.

Table 6.1. Higher order correlation and algebraic attacks applied against MONO

Attack	Data	Successive	Memory	Pre-comp.	Complexity	Probabilistic
6.4.1	127	No	2^{14}	------	2^{332}	Yes
6.4.2	2^{127}	Yes	2^{14}	------	2^{332}	Yes
6.5.3	$2^{23.3}$	No	$2^{46.6}$	------	$2^{274.2}$	No
6.5.4	$2^{23.5}$	No	$2^{46.6}$	------	$2^{274.2}$	No
6.5.5	2^{21}	No	2^{56}	------	$2^{287.2}$	No
6.5.6.1	$2^{23.4}$	Yes	$2^{36.6}$	$2^{30.58}$	$2^{260.3}$	No

6.5.8. Comments on Security Analysis

From the observations given in Table 6.1, we see that algebraic attacks and higher order correlation attacks with or without sequential clocking is infeasible and theoretically has no meaning as all the attacks has complexity greater than the exhaustive search over internal states of R1, R2 and R3. The best attack is the fast algebraic attack given in 6.5.6.1. However it is still higher than the exhaustive search which is 2^{247}. Computational complexity may be decreased with the use of several super-machines running special attacks. However there is no meaning of trying and mounting such an attack since it is higher than exhaustive search. The disadvantage of this attack is that it requires consecutive keystream bits and also it needs a pre-computation step. Consequently we see that MONO stream cipher is secure against important attacks.

6.6. Security of BBS Generator

The BBS is thought to be cryptographically secure pseudorandom bit generator. Because, given the first m bits of the sequence, there is not a practical algorithm that can allow someone to state that the next bit will be 1 (or 0) with probability greater than ½. For practical purposes, the sequence is unpredictable since the security of BBS is based on difficulty of factoring $n= pq$. As we know, almost many attacks aims to get the initial values of the LFSRs of the stream cipher. For many constructions it is possible to get the private key K , from the initial states. For MONO, an adversary needs to factor n which is a

the multiplication of two large primes. If an adversary finds these two primes p and q, he can find the private key K since the initialization vector IV is supposed to be publicly known value. So an adversary has two options. First one is to factor multiplication of two primes which is known to be extremely difficult, another one is to try all the possible values of B. However this option has the complexity of 2^{128}, therefore impractical. So we can see that BBS generator used in the initialization gives very high security to protect the private key K, if an adversary achieves to get the initial values somehow.

7. HARDWARE CONSIDERATIONS

As emphasized before stream ciphers are preferred as they are efficient in hardware when they use feedback shift registers. The size of the implementation of an algorithm depends strongly on the minimum feature size of the technology, which is the dimension of the smallest feature actually constructed in the manufacturing process. It also depends on the specific circuit design style, such as CMOS or DCVSL, and the number of available metal layers for wire routing. Hence, it is necessary to resort to an approximate, technology and circuit style independent measure. A commonly used measure for the size of a design is the number of NAND gate equivalents (GE). This is the area of the circuit implementation divided by the area of the smallest NAND gate in the used standard CMOS cell library. Table 7.1 below contains a subset of logical gates taken from a standard cell library for 130 nm CMOS technology. The hardware costs are given units of gate equivalents. One gate equivalent (GE) is the area necessary to implement a 2-input NAND-gate on silicon [42].

Table 7.1. Hardware costs of logical operations

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

Using this information we will try to determine the hardware equivalent of the keystream generation part of MONO. The initialization part is not considered to be implemented in hardware, therefore it may be implemented , in software. In [42], it is given that an LFSR stage can be implemented using 4.75 GE area. MONO has 247

registers in total therefore 4.75 * 247 = 1173,25 GE are is needed to implement these registers. The feedback functions of R1, R2 and R3 have six taps. These feedback function can be implemented using 3 two input XOR gates and 1 three input XOR gates which makes 10.75 GE. For three registers it makes 32.25 GE. In [43], a hardware implementation of AES S-box had been proposed. The proposal needs an area of 256 GE. We use this proposal in our hardware area calculations. Therefore the area needed to implement AES S-box is 256 GE. The clock control function makes use of f_1 and f_2 which can be implemented using 2 bit full adders. As given in [44], a full adder needs 3 AND, 2 OR and 2 XOR gates to be implemented. And this makes 10.75 GE area. As f_1 and f_2 needs two bit full adders, we need 43 GE area to implement these functions. Also f_{C2} and f_{C3} are used to select clocking functions at each cycle. Each of these two functions can be implemented using 3 XOR and 2 AND functions which makes 9.25 GE. To implement two functions we need 18.5 GE. Therefore, the clock control function needs 61.5 GE totally. *Q* and *N* values can be implemented using 6 bit full adder and 5 bit full adder respectively. This makes about 64.5 GE and 53.75 GE respectively. Also the memory function makes use of memory elements. There are 11 bit of memory used in the system which makes 52.25 GE since a memory is same as an LFSR of one stage. The memory function also uses 6 bit XOR which makes 13.5 GE. The last thing to be implemented is the output Boolean function. Below, the hardware implementation of this function is given. This function can be implemented using 67 GE area.

$$
\begin{aligned}
F_1 &= XOR(x_1, x_2, x_3) \\
F_2 &= XOR(MAJ(x_4, x_5, x_6), XOR(MAJ(x_4, x_7, x_8), \\
&\quad MAJ(x_5, x_7, x_8), MAJ(x_6, x_7, x_8))) \\
F_3 &= XOR(AND(x_4, x_5, XOR(x_6, x_7, x_8)), AND(x_6, x_8, XOR(x_4, x_5, x_7)), \\
&\quad AND(x_6, x_7, XOR(x_4, x_5)), AND(x_7, x_8, XOR(x_4, x_5))) \\
F_4 &= XOR(AND(AND(x_4, x_5, x_6), XOR(x_7, x_8)), AND(AND(x_6, x_7, x_8), \\
&\quad XOR(x_4, x_5)), AND(x_4, AND(x_5, x_7, x_8))) \\
F &= XOR(XOR(F_1, F_2, F_3), F_4)
\end{aligned}
\tag{7.1}
$$

Here F_1 represents 1st degree terms in *F*, F_2 represents 2nd degree terms in *F*, F_3 represents 3rd degree terms in *F* and F_1 represents the 4th degree terms in *F*. Table 7.2 below summarizes the hardware complexity of MONO.

Table 7.2. Gate Equivalent of MONO

Component		*Gate Equivalent*
LFSRs	R1 (59 bits)	280.25 GE
	R2 (61 bits)	289.75 GE
	R3 (127 bits)	603.25 GE
Feedback Functions	f_{R1}	10.75 GE
	f_{R2}	10.75 GE
	f_{R3}	10.75 GE
AES S-box		256.00 GE
Clock Control		61.50 GE
Memory and Multiplexing Function	*Q*	64.50 GE
	N	53.75 GE
	11 bit Memory	52.25 GE
	6 bit XOR	13.50 GE
F		67.00 GE
TOTAL		1774.00 GE

To further reduce the hardware complexity of MONO, we can reduce the length of registers R1, R2 and R3 without changing other functions and working of the cipher. Suppose that we choose lengths of R1, R2 and R3 as 19, 31 and 61. Then we will have a keystream period of approximately 2^{111}, also the minimum linear complexity will be about 2^{50} which is quite sufficient for cryptographic applications. Of course the complexity of previous attacks will be smaller however they will again be higher than brute force. Therefore system will be secure. On the other hand the hardware complexity would be much much smaller. With these register lengths total gate equivalent of the system will be 1128 GE. We see that the hardware complexity is lowered by this way. This amount of area is tolerable since the implementation of E0 in Bluetooth in [44] is calculated to be 1637 GE .

8. CONCLUSION

We have proposed a dynamically clock-controlled stream cipher with memory and dynamic multiplexed filtering, called MONO. MONO is designed to be secure against well-known and recent successful attacks such as algebraic attacks and correlation attacks.

The statistical properties of the system is excellent. We have applied FIPS 140-2 and NIST tests for several output sequences and MONO has passed all the statistical tests. The memory and clock distribution in the system is very likely to be uniform. This is a very good property. Also the period and linear complexity is very high to resist relevant attacks.

We have applied some correlation and algebraic attacks in order to test resistance of the system against these attacks. MONO seems to be secure against attacks discussed.

Consequently, we observed that we had achieved the design objectives and MONO may be treated as a secure stream cipher to be used for high speed encrypted communications. Cipher can be implemented in hardware. Also an effective assembly implementation of MONO would be appropriate for software applications.

APPENDIX : MATLAB CODES

In this part we give the simulation codes of MONO. The implementation is evaluated in MATLAB R14 environment. We choose MATLAB as it provides efficient computations on vectors and matrices. One can also use MATLAB compiler to convert the MATLAB implementations to run in C language. Below we have listed the MATLAB functions used to simulate the stream cipher and tests such as autocorrelation, DFT, Linear complexity for statistical observations and XL Algorithm test for multivariate equation solving, Gaussian elimination in GF(2) and Existence of Annihilator for MONO.

1. **mono.m** : Simulates the MONO stream cipher.
2. **BBS.m** : Simulates the Blum Blum Shub generator
3. **clock_ctrl.m** : Simulates the clock-control unit of MONO.
4. **dec.m** : Simulates the decimation of a binary vector.
5. **Key.m** : Generates 128 bit random private key.
6. **IV.m** : Generates 256 bit random publicly known initialization vector.
7. **Sbox_initialize.m** : Initialization of AES S-box.
8. **output_function.m** : Simulation of the output Boolean function.
9. **autocorrelation.m** : Simulation of the autocorrelation test.
10. **LCP.m** : Simulates the linear complexity profile of a binary sequence.
11. **spectral.m** : Simulation of spectral (DFT) test of a binary sequence
12. **XL_test.m**: Determines if XL algorithm is successful over solving a overdefined multivariate system of equations. And calculates the number of linearly independent equations.
13. **total_mon.m**: Calculates the total number of monomials in a multivariate equation of algebraic degree *d* in *n* variables.
14. **Annihilator.m**: Determines whether the output Boolean function used in MONO has Annihilator function of some degree less than the degree of the output function.
15. **gauss_elm.m**: Implements the Gaussian elimination on binary square matrices and determines whether the matrix has a solution.
16. **test_g_1.m:** General function of degree 1 to be tested as Annihilator

17. test_g_2.m: General function of degree 2 to be tested as Annihilator

18. test_g_3.m: General function of degree 3 to be tested as Annihilator

1- mono.m

```
function [ Keystream ] = mono(p, q, s, K, IV, T)
% MONO stream cipher
% The function has 6 inputs. Namely;
% p, prime number to be used in BBS such that congruent to 3 mod 4
% q, prime number to be used in BBS such that congruent to 3 mod 4
% s, any number relatively prime to p*q
% An example of p,q and s is 383, 503, 101355
% K , 128 bit private key
% IV, 256 bit Initialization vector
% T, Number of keystream outputs

disp ('*******************************************')
disp ('*                                         *')
disp ('*                M O N O                  *')
disp ('*                                         *')
disp ('*******************************************')


% 1-INITIALIZATION

% B is a 128-bit binary vector which is the result of BBS generator.
% I is the 384 bit vector to be used for initialization.

B=BBS(p, q, s); I=[mod(B+K,2) IV];

% Substitute and shift operations on I.

for i=1:(length(I)/12)
   H(i,:)=I((12*(i-1)+1):12*i);
end

x=1:5; y=8:12;
H(1,:)=mod(H(1+dec(H(1,x)),:)+H(1+dec(H(1,y)),:),2);
H([1+dec(H(1,x)) 1+dec(H(1,y))],:) = H([1+dec(H(1,y)) 1+dec(H(1,x))],:);

for r=2:(length(I)/12)
    x=1+mod(x+dec(H(r,x)),12);
    y=1+mod(y+dec(H(r,y)),12);
    H(r,:)=mod(H(1+dec(H(r,x)),:)+H(1+dec(H(r,y)),:),2);
    H([1+dec(H(r,x)) 1+dec(H(r,y))],:) = H([1+dec(H(r,y)) 1+dec(H(r,x))],:);
end

for z=1:(length(I)/12)
    I((12*(z-1)+1):12*z)=H(z,:);
end

%Drop last 38 bits
I=I(1:346);

%Sigma determines the final beta values.

sigma=dec(I(1:7));

%Final beta values are calculated
for h=1:6
    beta(h)=1+mod(28+h+sigma,61);
end

%First Q value (6-bit) is calculated

Q(1)=1+dec(I(8:13));

%First P value (6-bit) is determined

P(1,:)=I(14:19);
```

```
% This 80-bit vector will be used for S-box row&coloumn swapping.

I_80=I(22:101);

%Initialize the Sbox.
%Swap rows and columns.
S_box=Sbox_initialize(I_80);

% R1=59 bits; R2=61 bits; R3=127 bits.
R1=I(88:146); R2=I(147:207); R3=I(208:334);
L1=length(R1); L2=length(R2); L3=length(R3);


disp ('*******************************************')
disp ('*                                         *')
disp ('*       INITIALIZATION HAS FINISHED       *')
disp ('*                                         *')
disp ('*******************************************')


disp ('*******************************************')
disp ('*                                         *')
disp ('*            CIPHERING STARTED            *')
disp ('*                                         *')
disp ('*******************************************')


% 2-DATA GENERATION

%Here we start ciphering with clock-control mechanism

for t=1:T

    % R1 is shifted once
    temp=mod(R1(9)+R1(18)+R1(27)+R1(38)+R1(49)+R1(59),2); %feedback relation of R1
    for j=1:(L1-1)
        R1(L1+1-j)=R1(L1-j); % shifting to right
    end
    R1(1)=temp; %update the first register using the primitive polynomial

    C1=[R1(18) R1(26) R1(35) R1(43)];
    C2=[R1(1) R1(10) R1(51) R1(59)];
    U=dec2bin(S_box(1+dec(C1),1+dec(C2)),8); %Feed the s-box
    U=mod(double(U),48); %S-box output
    Y=U(2:7); %6-middle bits of S-box output
    %Update the next memory values
    P(2,:)=[R2(beta(1)) R2(beta(2)) R2(beta(3)) R2(beta(4)) R2(beta(5)) R2(beta(6))];
    Q(2)=1+dec(mod(Y+P(2,:),2));

    %Two edge bits of S-box output to be used in clock control
    y=[U(1) U(8)];

    %Gathering the clocking information for irregularly clocked LFSRs R2&R3
    [clock_2 clock_3]=clock_ctrl(y,R1,1+dec(P(1,1:5)));


    %R2
    for i=1:clock_2
    temp2=mod(R2(10)+R2(20)+R2(30)+R2(41)+R2(52)+R2(61),2); %feedback relation of R2

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


    %R3
    for i=1:clock_3
    temp3=mod(R3(21)+R3(42)+R3(63)+R3(84)+R3(106)+R3(127),2); %feedback relation of R3

        if i==clock_3
            out_1=R3(Q(1));    % For the last clock of R3, generate the output
```

```
            out_2=R3(Q(1)+10);
            out_3=R3(Q(1)+20);
            out_4=R3(Q(1)+30);
            out_5=R3(Q(1)+40);
            out_6=R3(Q(1)+50);
            out_7=R3(Q(1)+60);
            out_8=R3(127-Q(1));
        end

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

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


    %Generating the keystream output using output Boolean function
    Keystream(t)=output_function(out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8);

    %Shift the memory values
    P(1,:)=P(2,:);
    Q(1)=Q(2);
end

disp ('*****************************************')
disp ('*                                       *')
disp ('*            CIPHERING FINISHED         *')
disp ('*                                       *')
disp ('*****************************************')
```

2- BBS.m

```
function [B] = BBS(p,q,s)
% BBS , Blum Blum Shub Generator
% p and q are two large primes, and chosen such that both have
% remainder 3 when divided by 4.

n=p*q;
X(1)=mod(s^2,n);

for i=1:128
    X(i+1)=mod(X(i)^2,n);
    B(i)=mod(X(i+1),2);
end
```

3- clock_ctrl.m

```
function [ clock_2, clock_3] = clock_ctrl(x,R,N)
%CLOCK_CTRL is the clocking control mechanism of MONO stream cipher
% x is the 2-bit vector consisting of 1st and 8th elements of S-box output.
% R is the current values of the register which is used for clock control.
% Q is the v function output.

S=mod(x(1)+x(2),2);

A1=N; B1=59-N;
f1=2*R(A1)+R(B1)+1; %first clocking function

A2=N+27; B2=33-N;
f2=2*R(A2)+R(B2)+1; %second clocking function


if S==1
   clock_2=f1;
   clock_3=f2;
else
   clock_2=f2;
   clock_3=f1;
end
```

4- dec.m

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

5- Key.m

```
function [ K ] = Key()

K=rand(1,128);

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

6- IV.m

```
function [ K ] = IV()

K=rand(1,256);

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

7- Sbox_initialize.m

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


B= [99 124 119 123 242 107 111 197 48 1 103 43 254 215 171 118
    202 130 201 125 250 89 71 240 173 212 162 175 156 164 114 192
    183 253 147 38 54 63 247 204 52 165 229 241 113 216 49 21
    4 199 35 195 24 150 5 154 7 18 128 226 235 39 178 117
    9 131 44 26 27 110 90 160 82 59 214 179 41 227 47 132
    83 209 0 237 32 252 177 91 106 203 190 57 74 76 88 207
    208 239 170 251 67 77 51 133 69 249 2 127 80 60 159 168
    81 163 64 143 146 157 56 245 188 182 218 33 16 255 243 210
    205 12 19 236 95 151 68 23 196 167 126 61 100 93 25 115
    96 129 79 220 34 42 144 136 70 238 184 20 222 94 11 219
    224 50 58 10 73 6 36 92 194 211 172 98 145 149 228 121
    231 200 55 109 141 213 78 169 108 86 244 234 101 122 174 8
    186 120 37 46 28 166 180 198 232 221 116 31 75 189 139 138
    112 62 181 102 72 3 246 14 97 53 87 185 134 193 29 158
    225 248 152 17 105 217 142 148 155 30 135 233 206 85 40 223
    140 161 137 13 191 230 66 104 65 153 45 15 176 84 187 22];

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

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


%Swap 10 coloumns
B(:,[1+dec(A(3,:)) 1+dec(A(4,:))]) = B(:,[1+dec(A(4,:)) 1+dec(A(3,:))]);
B(:,[1+dec(A(7,:)) 1+dec(A(8,:))]) = B(:,[1+dec(A(8,:)) 1+dec(A(7,:))]);
B(:,[1+dec(A(11,:)) 1+dec(A(12,:))]) = B(:,[1+dec(A(12,:)) 1+dec(A(11,:))]);
B(:,[1+dec(A(15,:)) 1+dec(A(16,:))]) = B(:,[1+dec(A(16,:)) 1+dec(A(15,:))]);
B(:,[1+dec(A(19,:)) 1+dec(A(20,:))]) = B(:,[1+dec(A(20,:)) 1+dec(A(19,:))]);


Sbox=B;
```

8- output_function.m

```
function [ Keystream ] = output_function( x1, x2, x3, x4, x5, x6, x7, x8)
%OUTPUT_FUNCTION is a 3rd degree correlation immune balanced boolean
%(3-resilient) function

A=mod(x1+x2+x3,2);
B=mod(x4*x5+x4*x6+x4*x7+x4*x8+x5*x6+x5*x7+x5*x8+x6*x7+x6*x8+x7*x8,2);
C=mod(x4*x5*x6+x4*x5*x7+x4*x5*x8+x4*x6*x7+x4*x6*x8+x4*x7*x8+x5*x6*x7+x5*x6*x8+x5*x7*x8+x6*x
7*x8,2);
D=mod(x4*x5*x6*x7+x4*x5*x6*x8+x4*x6*x7*x8+x5*x6*x7*x8+x4*x5*x7*x8,2);

Keystream=mod(A+B+C+D,2);
```

9- autocorrelation.m

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

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

10- LCP.m

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


n=length(s);
%Initial values
C=[1]; l=0; m=-1; B=[1]; N=1; l=0;
while N<=n
    if l==0
        d=s(N);
    else
        c=C(2:length(C));
        d=mod(s(N)+sum(c.*s(N-1:-1:N-length(c))),2); %Next Discrepancy
    end

    if d==1
    T=C;
    P=gfconv(B,[zeros(1,N-1-m) 1]);         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

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

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

11- spectral.m

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

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

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

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

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

12- XL_test.m

```
function [ Result, Free ] = XL_test(n ,m ,d ,D ,eps)
% This function can be used to make an estimation on number of linearly
% independent (FREE) equations to solve multivariate equations of degree d.
% From resulting value of FREE equations, the success of XL algorithm is
% determined.

% d : degree of Boolean function to be solved by XL
% D : parameter of XL algorithm and should be greater than or equal to d
% n : number of variables
% m : number of available keystream
% eps: a number between 0 and 3
% WF : The Work Factor of XL attack if it is succesful given as an exponent
% on 2.

% TEST IS VALID FOR PARAMETERS OF D UP TO D<3d

% T : total number of monomials
```

```
T=total_mon(n,D);
% R : total number of generated equations
R=m*total_mon(n,D-d);

% Number of Free equations are calculated according to the results
% given in paper "HIGHER ORDER CORRELATION ATTACKS, XL ALGORITHM and
% CRYPTANALYSIS OF TOYOCRYPT".

if d<=D<=(2*d)-1
    Free=min(T,R);
elseif 2*d<=D<=3*d-1
        if D==2*d
            Free=min(T,R-(total_mon(m,2)-1))-eps;
        else
            Free=min(T,R-(total_mon(n,D-2*d)*(total_mon(m,2)-1)))-eps;
        end
end

Result=Free/(T-D);

if Result>=1
    display('XL ALGORITHM IS SUCCESFUL');
    WF=log2(7*T^log2(7));
    ['Total Complexity of XL Attack is 2^',num2str(WF)]
else
    display('XL FAILED')
end
```

13- total_mon.m

```
function [ T ] = total_mon( n, d )
% Computes the total number of monomials of degree <=d for n variables
T=1;
for i=1:d
    A=n:-1:n-i+1;
    B=1:i;
    C=prod(A)/prod(B);
    T=T+C;
end
```

14- Annihilator.m

```
% This file decides whether there exits an Annihilator of degree 1,2 or 3
% for 4th degree MONO output function.

% Existence of low degree annihilators are important to make better
% algebraic attacks since the complexity and required known keystream
% will be reduced.

% b= number of variables used inthe output Boolean function of MONO
% A: matrix containing the 8-bit vectors correponding to 1s at output.

b=8; a=0:2^b-1; A=zeros(2^(b-1),b);

for i=1:2^b
    I=dec2bin(a(i),b);
    I(i,:)=mod(double(I),48);
end

t=1;
for j=1:2^b
    x=I(j,:);
    if output_function(x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8))==1
        A(t,:)=I(j,:);
        t=t+1;
    end
end

for k=1:2^(b-1)
```

```
    [G1(k,:) R1(k,:)]=test_g_1(A(k,:));
    [G2(k,:) R2(k,:)]=test_g_2(A(k,:));
    [G3(k,:) R3(k,:)]=test_g_3(A(k,:));
end

%Apply Binary Gaussian elimination on each coefficient matrix
%If any of them gives a solution then we will know that
%there exits Annihilator of MONO output Boolean function
%of degree 1,2, or 3 for G1,G2 or G3

gauss_elm(G1,R1);
gauss_elm(G2,R2);
gauss_elm(G3,R3);
```

15- gauss_elm.m

```
function [] = gauss_elm(a,b)

%GAUSSIAN ELIMINATION in GF(2)
%a is the nxn coefficient matrix
%b is the nx1 result vector

L=size(a,2);
M=size(a,1);
for k=1:L
    s=k;
    while a(s,k)==0
    s=s+1;
    if s>M
        display('SYSTEM HAS NO SOLUTION')
        return
    end
    end
    % Swap rows of coefficient matrix "a"
    u=a(s,:); v=a(k,:);
    a(s,:)=v; a(k,:)=u;
    % Swap rows of result vector "b"
    x=b(s); w=b(k);
    b(s)=w; b(k)=x;
        for i=1:M
            if ((i~=k) & a(i,k)==1)
                a(i,:)=mod(a(i,:)+a(k,:),2);
                b(i)=mod(b(i)+b(k),2);
            else
                a(i,:)=a(i,:);
                b(i)=b(i);
            end
        end
end

B=num2str(b');

['THE SOLUTION OF THE SYSTEM IS [',B,']']
```

16- test_g_1.m

```
function [ g,r] = test_g_1(x)

% General function of degree 1 to be tested as Annihilator
x1=x(1); x2=x(2);x3=x(3);x4=x(4);x5=x(5);x6=x(6);x7=x(7);x8=x(8);
g=[1 x1 x2 x3 x4 x5 x6 x7 x8 zeros(1,119)];
r=mod(sum(g),2);
```

17- test_g_2.m

```
function [g,r] = test_g_2(x)

% General function of degree 2 to be tested as Annihilator
x1=x(1); x2=x(2);x3=x(3);x4=x(4);x5=x(5);x6=x(6);x7=x(7);x8=x(8);
```

```
g=[1 x1 x2 x3 x4 x5 x6 x7 x8 x1*x2 x1*x3 x1*x4 x1*x5 x1*x6 x1*x7 x1*x8....
    x2*x3 x2*x4 x2*x5 x2*x6 x2*x7 x2*x8 x3*x4 x3*x5 x3*x6 x3*x7 x3*x8 x4*x5....
    x4*x6 x4*x7 x4*x8 x5*x6 x5*x7 x5*x8 x6*x7 x6*x8 x7*x8 zeros(1,91)];

r=mod(sum(g),2);
```

18- test_g_3.m

```
function [g,r] = test_g_3(x)

% General function of degree 3 to be tested as Annihilator
x1=x(1); x2=x(2);x3=x(3);x4=x(4);x5=x(5);x6=x(6);x7=x(7);x8=x(8);

g=[1 x1 x2 x3 x4 x5 x6 x7 x8 x1*x2 x1*x3 x1*x4 x1*x5 x1*x6 x1*x7 x1*x8....
    x2*x3 x2*x4 x2*x5 x2*x6 x2*x7 x2*x8 x3*x4 x3*x5 x3*x6 x3*x7 x3*x8 x4*x5....
    x4*x6 x4*x7 x4*x8 x5*x6 x5*x7 x5*x8 x6*x7 x6*x8 x7*x8 x1*x2*x3....
    x1*x2*x4 x1*x2*x5 x1*x2*x6 x1*x2*x7 x1*x2*x8 x1*x3*x4 x1*x3*x5....
    x1*x3*x6 x1*x3*x7 x1*x3*x8 x1*x4*x5 x1*x4*x6 x1*x4*x7 x1*x4*x8....
    x1*x5*x6 x1*x5*x7 x1*x5*x8 x1*x6*x7 x1*x6*x8 x1*x7*x8 x2*x3*x4....
    x2*x3*x5 x2*x3*x6 x2*x3*x7 x2*x3*x8 x2*x4*x5 x2*x4*x6 x2*x4*x7....
    x2*x4*x8 x2*x5*x6 x2*x5*x7 x2*x5*x8 x2*x6*x7 x2*x6*x8 x2*x7*x8....
    x3*x4*x5 x3*x4*x6 x3*x4*x7 x3*x4*x8 x3*x5*x6 x3*x5*x7 x3*x5*x8....
    x3*x6*x7 x3*x6*x8 x3*x7*x8 x4*x5*x6 x4*x5*x7 x4*x5*x8 x4*x6*x7....
    x4*x6*x8 x4*x7*x8 x5*x6*x7 x5*x6*x8 x5*x7*x8 x6*x7*x8 zeros(1,35)];

r=mod(sum(g),2);
```

REFERENCES

1. Menezes, A., P. Van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press Inc., 1996.

2. Kanso, A. A., *Clock Controlled Generators*, Ph. D. Thesis University of London, 1999.

3. 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.

4. Gollman, D. and W. G. Chambers, "Clock-Controlled Shift Registers: A Review", *IEEE Journal on Selected Areas in Communications*, Vol.7, pp.525-533, 1989

5. Zeng, K., Chung-Huang Yang, Dah-Yea Wei and T.R.N Rao, "Pseudorandom Bit Generators in Stream-Cipher Cryptography", *IEEE Computer*, Vol.24, pp.8-17, 1991

6. Stallings, W., *Cryptography and Network Security Principles and Practice, Third Edition*, Prentice Hall Press, 2003.

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

8. Golic, J., "Cryptanalysis of Alleged A5 Stream Cipher", *Proceedings of EUROCRYPT 97*, *Lecture Notes in Computer Science*, Vol. 1233, pp. 239-255, 1997.

9. Blum, L., Manuel Blum and Michael Shub, "Comparison of two pseudo-random number generators" , *Proceedings of CRYPTO 82*, pp. 61-78, 1983.

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

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

12. Gammel B. M., R. Göttfert and O. Kniffler, *Improved Boolean Combining Functions for Achterbahn* , http://cr.yp.to/streamciphers/achterbahn

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

14. Federal Information Processing Standards, FIPS PUB 140-1 Security Requirements for Cryptographic Modules http://csrc.nist.gov/cryptval/140-1.htm

15. Federal Information Processing Standards, FIPS PUB 140-2 Security Requirements for Cryptographic Modules, http://csrc.nist.gov/cryptval/140-2.htm

16. Massey, J.L., "Shift Register synthesis and BCH decoding", *IEEE Transactions on Information Theory*, Vol. 15, pp. 122-127, 1969.

17. Filiol, E., "Decimation Attack of Stream Ciphers ", *Progress in Cryptology INDOCRYPT 2000 , Lecture Notes in Computer Science,* Vol. 1977, pp.31-42, 2000

18. Courtois, N., "Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt", *Information Security and Cryptology-ICISC 2002*, Vol. 2587, 2002.

19. Courtois, N. and W. Meier, " Algebraic Attacks on Stream Ciphers with Linear Feedback", *Proceedings of EUROCRYPT 2003*, *Lecture Notes in Computer Science*, Vol. 2656, pp. 345-359, 2003.

20. Mihaljevic, M. and H. Imai, "Cryptanalysis of Toyocrypt-HS1 stream cipher", *IEICE Transactions on Fundementals*, Vol. E85-A, pp. 66-73, Jan. 2002.

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

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

23. Courtois, N., " Fast Algebraic Attacks on Stream Ciphers with Linear Feedback", *Advances in Cryptology-CRYPTO 2003*, *Lecture Notes in Computer Science*, Vol. 2729, pp.177-194, 2003.

24. Courtois, N., "Algebraic Attacks on Combiners with Memory and Several Outputs", *Information Security and Cryptology-ICISC 2004*, Vol. 3506, 2005

25. Strassen, V., "Gaussian Elimination is not optimal", *Numerische Mathematik*, Vol. 13, pp. 354-356, 1969.

26. Shamir, A. and A. Kipnis, "Cryptanalysis of the HFE Public Key Cryptosystem", *Proceedings of CRYPTO 99*, *Lecture Notes in Computer Science* , Vol. 1666, pp. 19-30, 1999.

27. Shamir, A., J. Patarin, N. Courtois and A. Klimov, "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations", *Proceedings of EUROCRYPT 2000*, *Lecture Notes in Computer Science* , Vol. 1807, pp. 392-407, 2000.

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

29. A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications, http://csrc.nist.gov/rng/rng2.html

30. Golic, J. and L. O'Connor, "Embedding and Probabilistic Correlation attacks on Clock-Controlled Shift Registers", *Proceedings of EUROCRYPT 1994*, *Lecture Notes in Computer Science*, Vol. 950, pp.230-243, 1995.

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

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

33. Lee, D. H., J. Kim, J. Hong, J. W. Han and D. Moon, "Algebraic Attacks on Summation Generators", *Proceedings of FSE 2004*, *Lecture Notes in Computer Science*, Vol.3017, pp. 34-48. 2004.

34. Hinai, S., L. Batten, B. Colbert and K. Wong, "Algebraic Attacks on Clock-Controlled Stream Ciphers", *ACISP 2006, Lecture Notes in Computer Science* , Vol. 4058, pp. 1-16, 2006.

35. Hawkes, P. and G. Rose, "Rewriting The Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers", *Proceedings of CRYPTO 2004*, *Lecture Notes in Computer Science* , Vol. 3152, pp. 390-406, 2004.

36. Meier, M., E. Pasalic and C. Carlet , "Algebraic Attacks and Decomposition of Boolean Functions", *Proceedings of EUROCRYPT 2004*, *Lecture Notes in Computer Science*, Vol. 3027, pp. 474-491, 2004.

37. Armknecht, F. and M. Krause, “Algebraic Attacks on Combiners with Memory”, *Proceedings of CRYPTO 2003*, *Lecture Notes in Computer Science* , Vol.2729, pp. 162-175, 2003.

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

39. Berbain, C., O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin and H. Sibert, “Decim: A Stream Cipher For Hardware Applications”, *ECRYPT Stream Cipher Project Report 2005*, http://www.ecrypt.eu.org/stream/

40. Gouget, A., H. Sibert, C. Berbain, N. Courtois, B. Debraize and C. Mitchell, “Analysis of the Bit-Search Generator and sequence compression techniques”, *Fast Software Encryption*, *Lecture Notes in Computer Science*, Vol. 3557, pp. 196-214, 2005

41. Wagner, D., L. Simpson, E. Dawson, J. Kelsey, W. Millan and B. Schneier, *Cryptanalysis of ORYX* , http://www.schneier.com/paper-oryx.html

42. Gammel B., R. Göttfert and O. Kniffler, “ACHTERBAHN-128/80”*, eSTREAM*, *ECRYPT Stream Cipher Project*, 2006, http://www.ecrypt.eu.org/stream

43. Yu, N. and H. M. Heys, "Investigation of Compact Hardware Implementation of Advanced Encryption Standard", *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering* , pp. 1069-1072.

44. Batina, L., J. Lano, N. Mentens, B. Preneel, I. Verbauwhede and S. B. Örs, "Energy, Performance, Area versus Security Trade-offs for Stream Ciphers", *ECRYPT Workshop, SASC - The State of the Art of Stream Ciphers,* pp. 302-310, 2004.