**T.C.**
**BAHÇEŞEHİR ÜNİVERSİTESİ**

# PROBABILISTIC PROPERTIES OF BIT-SEARCH TYPE COMPRESSION ALGORITHMS AND THE BACKTRACKING ATTACK

MS Thesis

ORHAN ERMİŞ

**İSTANBUL, 2007**

**T.C.**

**BAHÇEŞEHİR ÜNİVERSİTESİ**

**INSTITUTE OF SCIENCE**

**COMPUTER ENGINEERING**

# PROBABILISTIC PROPERTIES OF BIT-SEARCH TYPE COMPRESSION ALGORITHMS AND THE BACKTRACKING ATTACK

MS Thesis

**ORHAN ERMİŞ**

**Supervisor: PROF. DR. YALÇIN**
**Co-Supervisor: PROF. DR. EMİN ANARIM**

**İSTANBUL, 2007**

**T.C.**
**BAHÇEŞEHİR ÜNİVERSİTESİ**
**INSTITUTE OF SCIENCE**
**COMPUTER ENGINEERING**

Name of the thesis: Probabilistic Properties of Bit Search Type Compression Algorithms and the Backtracking Attack
Name/Last Name of the Student: Orhan ERMİŞ
Date of Thesis Defense:12/09/2007

The thesis has been approved by the Institute of Science.

Assoc. Prof. Dr. İrini DİMİTRİYADİS
Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Adem KARAHOCA
Program Coordinator

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

| Examining Comittee Members | Signature |
|---|---|
| Prof. Dr. Şenay YALÇIN | _____ |
| Prof. Dr. Emin ANARIM | _____ |
| Prof. Dr. Ali GÜNGÖR | _____ |
| Prof. Dr. Nizamettin AYDIN | _____ |
| Assoc. Prof. Dr İrini DİMİTRİYADİS | _____ |

# ABSTRACT

PROBABILISTIC PROPERTIES OF BIT-SEARCH TYPE

COMPRESSION ALGORITHMS

AND

THE BACKTRACKING ATTACK

ERMİŞ, Orhan

M.S. Department of Computer Engineering

Supervisor: Prof. Dr. Şenay YALÇIN

Co-Supervisor: Prof. Dr. Emin ANARIM

September, 2007, 60 pages

Linear Feedback Shift Registers (LFSRs) are the pseudorandom number generators that are used as keystream generators in Stream Ciphers. LFSRs are algebraically weak systems that have some vulnerability. To overcome this weakness of LFSRs, lots of nonlinear structures are used. In this thesis, we will deal with the most common technique to use nonlinearity in stream ciphers, which is called the compression algorithms. We investigate the probabilistic properties of the most common ones of these compression algorithms. They are SSG, BSG, ABSG, MBSG and EBSG. We also proposed a new attack to the EBSG algorithms that is called the backtracking attack.

**Keywords:** Bit Search Generator, Pseudo-Random Sequence, Backtracking Attack, Compression, Self-Shrinking Generator, ABSG, MBSG, EBSG.

# ÖZET

BİT ARAMA TİPİ
SIKIŞTIRMA ALGORİTMALARININ
OLASILIKSAL OZELLIKLERI
VE
GERİLEME SALDIRISI

ERMİŞ, Orhan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Prof. Dr. Şenay YALÇIN

Eş Danışman: Prof. Dr. Emin ANARIM

Eylül 2007, 60 sayfa

Doğrusal Geri Beslemeli Kaydırmalı Yazmaçlar(LFSR), akış şifreleme yönteminde anahtar üreteci olarak kullanılırlar. LFSRlar cebirsel olarak bazı zayıflıklara sahiptirler. Bu zayıflıkların üstesinden gelmek için doğrusal olmayan yöntemler kullanılır. Bu yöntemlerden bir tanesi de sıkıştırma algoritmalarıdır. Bu tezde, sıkıştırma algoritmalarının en çok önem sahibi olan SSG, BSG, ABSG, MBSG ve EBSG algoritmalarının çıkış verme durumlarının olasılıksal özellikleri üzerine çalışılmıştır. Ve bu algoritmaların en yenisi olan EBSG üzerine gerileme saldırısı adı verilen bir saldırı gerçekleştirilmiştir.

**Anahtar Kelimeler:** Bit Arama Üreteci, Sahte Rasgele Dizi , Gerileme Saldırısı, Sıkıştırma, Kendini Küçültme Üreteci, ABSG, MBSG, EBSG.

# ACKNOWLEDGMENTS

This thesis is dedicated to **my family and my girl friend** for their patience and understanding during my master's study and the writing of this thesis.

I would like to express my gratitude to **Prof. Dr. Emin ANARIM and Prof Dr. Şenay YALÇIN**, for not only being such great supervisors but also encouraging and challenging me throughout my academic program.

I wish to thank **Asst. Prof. Ahmet BEŞKESE, Asst. Prof. F. Tunç BOZBURA**, **Asst. Prof. Dr. H. Fatih UĞURDAĞ, Asst. Prof. Dr. M. Kıvanç MIHÇAK and Yücel ALTUĞ** for their help on various topics in the areas of probability and cryptology, for their advice and time.

I also thank my **colleagues,** for their patience and help.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| **⊕:** | XOR, Exclusive or, modulo 2 addition |
| **LFSR:** | Linear Feedback Shift Register |
| **IID:** | Independent Identically Distributed |
| **PRNG:** | Pseudo Random Number Generator |
| **SSG:** | Self Shrinking Generator |
| **BSG:** | Bit Search Generator |
| **MBSG:** | Modified Bit Search Generator |
| **EBSG:** | Editing Bit Search Generator |
| **ANF:** | Algebraic Normal Form |
| **SG:** | Shrinking Generator |
| **s :** | Input Sequence |
| $\mathbf{N_{n:}}$ | Number of nodes at depth n |
| $\mathbf{L_n}$**:** | Number of leaves at depth n |
| **Rate(C, *f*):** | Output rate of (C, *f*) |
| $\mathbf{P_{odd}}$**:** | Probability of getting odd codeword in the first state |
| $\mathbf{P_{even}}$**:** | Probability of getting even codeword in the first state |
| $\mathbf{P'_{odd}}$**:** | Probability of getting odd codeword after the first state |
| $\mathbf{P'_{even}}$**:** | Probability of getting even codeword after the first state |

# 1. INTRODUCTION

Security is the most important concept as far as communication is concerned. Cryptography is a science that deals with hiding the content of the messages that will be transmitted one point to another. Before the modern era, people were dealing with simple cryptographic algorithms which are simple and small effects on the message such as changing the order of letters, shifting letters in the alphabet, etc. Development in technology leads people to make more secure algorithms. These algorithms are called symmetric and asymmetric encryption. During the thesis, we will deal with the key stream generation in stream ciphers which are the subset of symmetric key encryption.

The most important subject in stream ciphers is to design a system that produces random looking output sequences. Stream ciphers use pseudorandom number generators as a secret key.

Linear Feedback Shift Registers (LFSRs) are the pseudorandom number generators that they form the core like structure of most of the stream ciphers or we use them as a stream cipher. LFSRs produce random sequences with using their linear function. The resulting output sequence is a linearly dependent structure and the input sequence can be easily evaluated within a small set of algebraic operations from any subpart of the output sequence. To overcome this weakness of LFSRs, lots of nonlinear structures are used.

In this thesis, we deal with one of the most common technique to use to add nonlinearity to the stream ciphers, which is called compression algorithms. We investigate the probabilistic properties of the most common ones of these compression algorithms. They are SSG, BSG, ABSG, MBSG and EBSG. We also propose a new attack for the EBSG algorithm that is called the backtracking attack.

The outline of thesis is like that in section 2, some detailed definitions on cryptology are given and in section 3, stream ciphers are discussed, in section 4, the compression algorithms are introduced. The derivation of probabilistic properties of compression algorithms are given in section 5. The section 6 discusses the new approach for cryptanalysis of EBSG called the backtracking attack.

# 2. CRYPTOLOGY

Cryptology is the science that provides ways to protect and to capture the information in an online or offline transmission. It consists of two subfields called cryptography and cryptanalysis. The cryptography is deals with the protection of data, developing new algorithms, protocols, systems, etc. The cryptanalysis is a necessity of improving the cryptography; people also develop new structures, theorems, etc. to attack the cryptographic system.

## 2.1 CRYPTOGRAPHY:

As mentioned before, cryptography is the way of protecting the information. The cryptographic systems are mostly related with four basic security services which are listed below in Stallings (2003):

- **Confidentiality:** The protection of data from unauthorized disclosure.
- **Data Integrity:** The assurance that data received are exactly as sent by an authorized entity.
- **Authentication:** The assurance that the communicating entity is the one that it claims to be.
- **Non-Repudiation:** Provides protection against denial by one of the entities involved in a communication data block: takes the form of determination of whether the selected fields have been modified.

The necessity of such services affects the selection of system to protect our data. Generally, besides the usage of security services the cryptographic algorithms also take an important point in our security structures. We can firstly group the cryptographic algorithms in two, the symmetric key encryption algorithms and the asymmetric key encryption.

Before defining the types of encryption algorithms, we have to define the concept, encryption. Encryption is a way to hide the content of the data with using set of rules and a secret key. The considerations of encryption stated with lots of principle. Te

most known one is that the Kerckhoff's Principle Stamp and Low (2007), due to this principle our encryption method is publicly known and the secret key will only be known for parties who use the secure communication line.

We have also introduced the basic terms that are used in the cryptographic encryption algorithm. These terms are:

- ***Secret Key*** is a value for which we use to alter the input message.
- ***Plaintext*** is a input message that will be encrypted using an algorithm and a secret key.
- ***Ciphertext*** is a resulting value which is encrypted by an encryption algorithm and using the secret key.

There are also methods that do not need any secret key for producing a secret contented output (ciphertext), for example, the hash functions Stallings (2003), which we will not introduce it in our study.

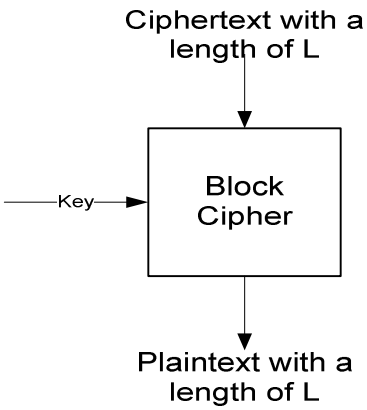
Symmetric key encryption systems are divided into two groups, called as block and stream ciphers. The most important property of the symmetric key encryption is that the same key is used for both encryption and decryption.

The term asymmetric key encryption is generally known as public key encryption Goldreich (2001). Comparing with the symmetric key encryption techniques, the asymmetric encryption systems use different keys for encryption and the decryption. On the other hand, as mentioned in Stallings (2003), the number theory plays an important role in public key encryption. We generally use asymmetric key encryption for key distribution systems.

As mentioned above the symmetric key encryption techniques can be classified into two groups, the block ciphers and the stream ciphers.

Block ciphers is a system that takes L-bit input and produces L-bit output with using a variable length (key length is up to the specifications of algorithm) secret key. Block by block encryption is realized in Block Ciphers that is why we call them block

ciphers. The most important property of block cipher is the Feistel Network Stallings (2003), which was built by Horst Feistel. This structure is used for mostly all modern block cipher algorithms.

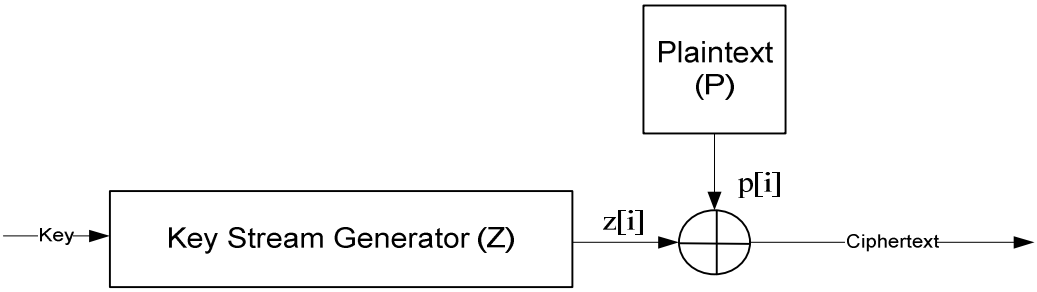



Stallings, W., 2003. *Cryptogrphy and the Network Security.* New Jersey : Prentice Hall, 3rd Edition

**Figure 1: The Basic Block Cipher**

The most common block cipher algorithms are DES (Data Encryption Standard) Mao (2003) and AES (Advanced Encryption Standard) Stallings (2003).

Stream ciphers (Menezes at al. 1997), Stamp and Low (2007) are another important class of symmetric key encryption systems which use bit by bit encryption instead of block by block encryption. In block ciphers, we use an algorithm that provides diffusion and collusion like properties to the ciphertext. But in stream cipher the most important think is the producing a pseudo-random keystream from secret key. The encryption operation is realized with a simple XOR operation. The basic structure of the stream ciphers can be summarized as in the following figure.




Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

**Figure 2: Basic Stream Cipher**

Stream ciphers are more suitable for fast implementation in hardware implementation than block ciphers.

## 2.2 CRYPTANALYSIS

As mentioned before, cryptanalysis is the way to improve the cryptographic systems. In other words, cryptanalysis is a study for breaking the cipher. In general, we use the "cryptanalytic attack" for the operation of studying all of the properties of the system and finding weaknesses as a result of that to breaking the cipher. According to the Stallings (2003), cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

The most common cryptanalysis technique is the brute force attack, which is also known as exhaustive key search, is the upper bound for the complexity of breaking the cipher. This attack tries all the possible keys in the algorithm to get the correct one. The cryptanalyst, who wants to make his attack more efficient than exhaustive search, has to decrease search space of his cryptanalysis algorithm.

We know that the upper bound for the complexity of an attack is the complexity of the brute force attack for every algorithm. So our purpose has to decrease the number of possibilities to make the cryptanalysis more efficient. Cryptographic algorithms consist of linear and non-linear structures. These structures always improve the security of the algorithm but sometimes they can be the weakest part of the system. The cryptanalysis of the algorithm is a complicated work that the analysis of the system that we consider the length of the ciphertext, plaintext and the secret key and algebraic, statistical, etc. like properties of the algorithm to break the cipher.

According to the Stallings (2003), the most common types of cryptanalytic attacks and their properties for the Kerckhoff's Principle are listed below:

- **Ciphertext Only Attack:** The knowledge of the encryption algorithm and the ciphertext to be decoded is enough to break the cipher

- **Known Plaintext Attack:** We have to know the encryption algorithm, ciphertext to be decoded and one or more ciphertext-plaintext pair that are formed with the secret key.

- **Chosen Plaintext Attack:** The encryption algorithm, ciphertext to be decoded and plaintext message chosen by the cryptanalyst, together with its corresponding ciphertext generated with the secret key are required.

- **Chosen Ciphertext Attack:** The encryption algorithm, ciphertext to be decoded and purported ciphertext chosen by the cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key are necessary to crack the algorithm.

- **Chosen Text Attack:** The encryption algorithm, ciphertext to be decoded, plaintext message chosen by the cryptanalyst, together with its corresponding ciphertext generated with the secret key and the purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key.

# 3. STREAM CIPHERS

As mentioned before stream ciphers are kind of symmetric encryption algorithms that they operate on the plaintext bit by bit to produce the ciphertext. Stream ciphers can be classified into three groups: the one time pad, the synchronous stream ciphers and the self-synchronous stream ciphers to the (Menezes *et al.* 1997).

In some resources, one time pad cannot be accepted as a stream cipher. The one time pad means that we have a system that encrypts the plaintext using a key that has the same length with plaintext and also the length of resulting ciphertext will be the same as the secret key. The most known example for one time pad type stream ciphers the Vernam Cipher, see (Menezes at al. 1997). While a one-time pad cipher is provably secure (provided it is used correctly), it is generally impractical since the key is the same length as the message.

According to the definition in (Menezes at al. 1997), a synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext. Properties of synchronous stream cipher are defined below.

- In a synchronous stream cipher, both the sender and receiver must be synchronized using the same key and operating at the same position within that key – to allow for proper decryption. If synchronization is lost due to ciphertext digits being inserted or deleted during transmission, then decryption fails and can only be restored through additional techniques for re-synchronization. Techniques for re-synchronization include re-initialization, placing special markers at regular intervals in the ciphertext, or, if the plaintext contains enough redundancy, trying all possible keystream offsets.

- A ciphertext digit that is modified (but not deleted) during transmission does not affect the decryption of other ciphertext digits.

- As a consequence of the first property, the insertion, deletion, or replay of ciphertext digits by an active adversary causes immediate loss of synchronization,

and hence might possibly be detected by the attacker. As a consequence of the second property, an active adversary might possibly be able to make changes to selected ciphertext digits, and know exactly what affect these changes have on the plaintext. This illustrates that additional mechanisms must be employed in order to provide data origin authentication and data integrity guarantees.

There are two more structures that are really an important issue in stream ciphers called the Linear Feedback Shift Registers (LFSRs) and the Boolean Functions.

### 3.1 LINEAR FEEDBACK SHIFT REGISTERS (LFSRs)

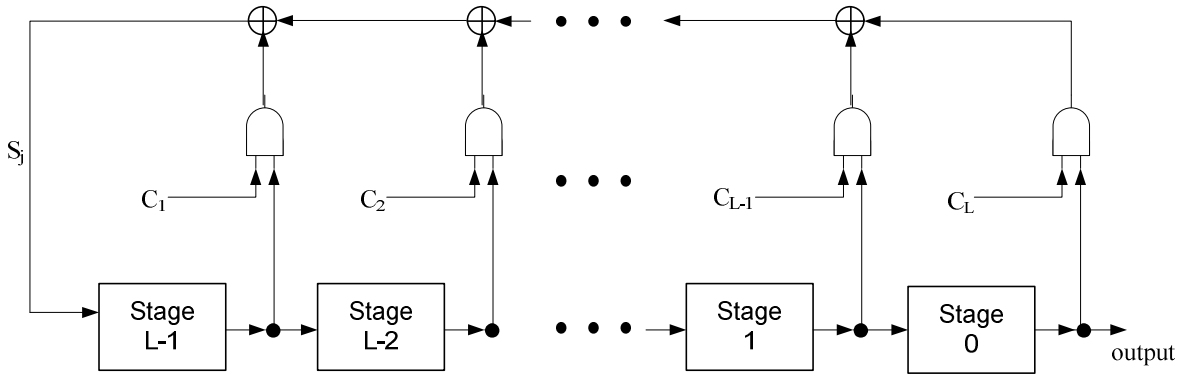
In (Menezes *et al.* 1997), Linear Feedback Shift Registers are used in many of the keystream generators that have been proposed in the literature. The key point that makes LFSRs important in stream cipher design is that they can produce random looking numbers to the given key value.

There are several reasons that make LFSRs important:

- LFSRs are well-suited to hardware implementation
- They can produce sequences of large periods
- They can produce sequences with good statistical properties
- Because of their structure, they can be readily analyzed using algebraic techniques.

To sum up, we use LFSRs for achieving the requirements in the former section.

The following figure defines the working principle of the LFSR with a length of L, each $C_i$ represent the feedback coefficient, the closed semi-circles shows the logical AND gates, and the feedback bit $S_j$ is the modulo 2 sum of the contents of those stages i, $0 \leq i \leq L-1$, for which $C_{L-i} = 1$ (Menezes et al. 1997).




Jansen, S.J.A., 2004. *Stream Cipher Design: Make your LFSR jump!* : Workshop Record ECRYPT-State of the Art of Stream Ciphers, pp. 94-108

**Figure 3: The Linear Feedback Shift Register**

**Definition 2.1:** Figure 2.1 denotes polynomial $F(D) = 1+C_1D+C_2D^2+\ldots+C_LD^L$. This polynomial is called connection polynomial which is also called the feedback polynomial Jansen (2004, pp. 94-108). and defined as

$$F(D) := \sum_{i=0}^{L} C_i D^i \qquad (1)$$

The degree of the connection polynomial is equal to the length of the LFSR.

**Definition 2.2:** Assume that, we have an LFSR with a length of L, the Lth order recursion is commonly represented by its Characteristic Polynomial, C(D), also of degree L Jansen (2004, pp. 94-108), as shown in below:

$$C(D) := \sum_{i=0}^{L} C_i D^{L-i} \qquad (2)$$

**Definition 2.3:** The functions F and C are reciprocal of each other. That means, this relation is expressed as $C(D) = D^L F(D^{-1})$ Jansen (2004, pp. 94-108).

Another way to look at the LFSR is to consider it as a Linear Finite State Machine as in Jansen (2004, pp. 94-108). In this case the state of the LFSM is represented by a vector $\sigma^t = (\sigma^t_{n-1}, \sigma^t_{n-2}, \ldots, \sigma^t_0)$, where $\sigma^t_i$ denotes the content of memory cell Mi after t transitions. As the finite state machine is linear, transitions from one state to

the next can be described by a multiplication of the state vector with a transition matrix T, i.e. $\sigma^{t+1} = \sigma^t$ T, for t≥0. The transition matrix is as given below:

$$T = \begin{pmatrix} 0 & 0 & K & 0 & c_L \\ 1 & 0 & K & 0 & c_{L-1} \\ 0 & 1 & K & 0 & c_{L-2} \\ M & M & O & M & M \\ 0 & 0 & K & 1 & c_1 \end{pmatrix}$$

It can be seen that the matrix is equal to the so called companion matrix of the polynomial C(D). The characteristic polynomial of T in linear algebra sense, i.e. det(DI-T), precisely equals this polynomial and, hence, C(T) = 0. So the companion matrix plays the role of a root of C and, consequently it can be used to form solutions of the recursion equation.

**Definition 2.4:** Assume that we have an LFSR with a period of L. If the LFSR produce a sequence with a length of 2L-1 without any recursion, these LFSRs are called the maximum length LFSR

At last, we must be careful about the initial key value of the LFSR. Because, if we use the key with all zero will makes LFSR to produce a sequence of all zero.

### 3.2 BOOLEAN FUNCTIONS:

Boolean functions are another important element of the stream cipher concept. They maps one or more binary input variable to one binary output.

**Definition 2.5:** In Boztaş (2007), Boolean functions $f$: $F_2^n \rightarrow F_2$ map binary vectors of length n to the finite field $F_2$.

There are $2^{2^n}$ distinct Boolean functions to the n different binary input variables and we denote the set of Boolean functions in n variables by $B_n$. According to the Carlet

(2006), among the classical representations of Boolean functions, the one in which is most usually used in cryptography and coding is the n-variable polynomial representation over , $F_2$ of the form

$$f(x) = \bigoplus_{i \in P(N)} a_i \left( \prod_{j \in i} x_j \right) = \bigoplus_{i \in P(N)} a_i x^i ,$$

where P(N) denotes the power set of N = {1, 2, …, n}. Every coordinate $x_i$ appears in this polynomial with exponents at most 1, because every bit in $F_2$ equals its own square. This representation belongs to $F_2 [x_1, \ldots x_n]/( x_1^2 \oplus x_1, \mathsf{K}, x_n^2 \oplus x_n )$. This is called the Algebraic Normal Form (ANF) Carlet (2006).

**Definition 2.6:** According to the Boztaş (2007), $w_H(f) := \#\{x \in F_2^n : f(x) \neq 0\}$ is the Hamming weight of a Boolean function while the Hamming distance between two such functions is

$$\#\{x \in F_2^n : f(x) \neq g(x)\} = w_H(f \oplus g)$$

In other words, Hamming weight is the number of ones in the vector and the Hamming distance of two functions is that the Hamming weight of modulo two-addition of these two functions.

**Definition 2.7:** According to the Seren (2007), functions of degree at most one are called affine. The set of all affine functions in n variables is denoted as $A_n$. We can write

$$A_n = \{a_0 + a_1 x_1 + a_2 x_2 + \mathsf{K} + a_n x_n : a_i \in F_2, 0 \leq i \leq n\}.$$

And also According to the Boztaş (2007), while there are only 2n+1 (out of $2^{2^n}$ total) affine Boolean functions, they form a significant class of functions and are extensively used in applications.

Continuing from Seren (2007), High nonlinearity for Boolean function is desired objective because it decreases the correlation between the output and the input

variables or a linear combination of input variables many of the attacks against stream ciphers succeed with the help of weakness of such a correlation between the combining Boolean function and some affine function.

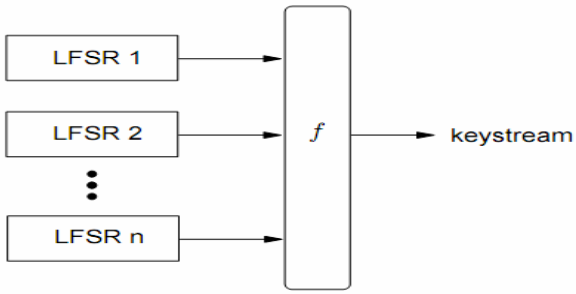
**Definition 2.9:** According to the Seren (2007), let $X_1, X_2, \mathrm{K}, X_n$ be independent random variables, each taking the values 0 and 1 with probability 1/2. A Boolean function $f(x_1, x_2, \mathrm{K}, x_n)$ is said to be *t*-th order correlation immune, if for each subset of *t* variables $X_1, X_2, \mathrm{K}, X_n$ with $1 \le i_1 \le i_2 \le \mathrm{K} \le i_t \le n$, the random variable Z = $f(X_1, X_2, \mathrm{K}, X_n)$ is statistically independent of the random vector X.

## 3.3 TYPES OF STREAM CIPHERS

Generally, stream ciphers are divided into three groups, in (Menezes et al. 1997):

- Nonlinear Combination Generators
- Nonlinear Filtering Generators
- Clock-Controlled Generators

Continuing from (Menezes et al. 1997), one general technique for destroying the linearity inherent in LFSRs is to use several LFSRs in parallel. The keystream is generated as a nonlinear function *f* of the outputs of the component LFSRs; this construction is illustrated in the following figure. Such keystream generators are called nonlinear combination generators, and *f* is called the combining function.

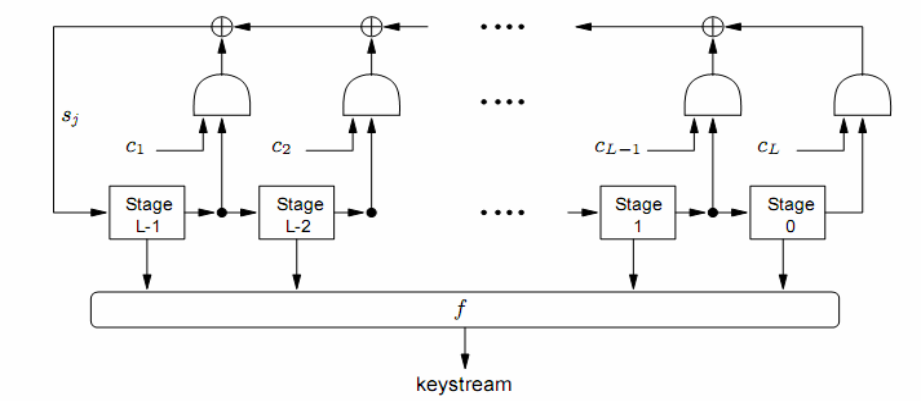



Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

**Figure 4: Nonlinear Combination Generator**

**Definition 2.10:** According to the (Menezes et al. 1997), the definition of a nonlinear combination generator is like that a product of m distinct variables is called an $m^{th}$ order product of the variables. Every Boolean function f ($x_1$, $x_2$, …, $x_n$) can be written as a modulo 2 addition of distinct $m^{th}$ order products of its variables, $0 \leq m \leq n$; this expression is called the algebraic normal form of *f*. the nonlinear order of is the maximum of the order of the terms appearing in its algebraic normal form.

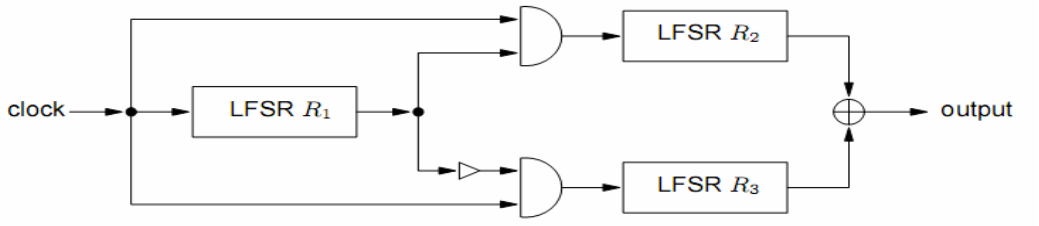
According to the Seren (2007), the nonlinear filter generator has different design principle. There is one LFSR and its different elements are used as an input of the Boolean function. There is an example for the nonlinear filter generator in the following figure




Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

**Figure 5: Nonlinear Filter Generator**

The clock-controlled generators can be expressed into two groups. The first one is the alternating step generator. In this type of clock-controlled generator one LFSR is used to clock the other two LFSRs as shown in the following figure.




Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

**Figure 6: Alternative Step Generator**

We can summarize the operation of alternating step generator, as in (Menezes *et al.* 1997), as follows:

**1.** Register $R_1$ is clocked.

**2.** If the output of $R_1$ is 1 then:

$R_2$ is clocked; $R_3$ is not clocked but its previous output bit is repeated.

(For the first clock cycle, the "previous output bit" of $R_3$ is taken to be 0.)

**3.** If the output of $R_1$ is 0 then:

$R_3$ is clocked; $R_2$ is not clocked but its previous output bit is repeated.

(For the first clock cycle, the "previous output bit" of R is taken to be 0.)

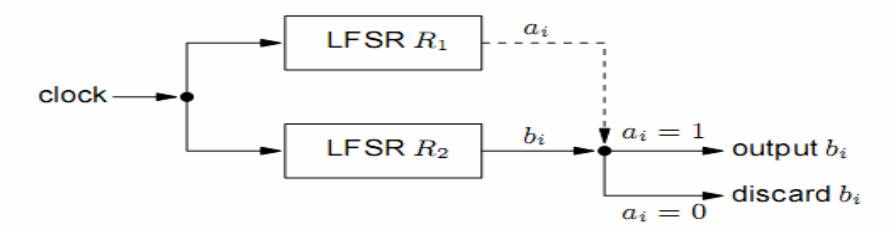
**4.** The output bits of $R_2$ and $R_3$ are XORed; the resulting bit is part of the keystream.

Second type of clock-controlled generators are that the Shrinking Generator. The working principle of the Shrinking generator is as followsn summarize the operations of this generator in the following three steps (Menezes et al. 1997):

**1.** Registers R1 and R2 are clocked.

**2.** If the output of $R_1$ is 1, the output bit of $R_2$ forms part of the keystream.

**3.** If the output of $R_1$ is 0, the output bit of $R_2$ is discarded.




Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

**Figure 7: Shrinking Generator**

## 3.4 SECURITY OF STREAM CIPHERS

As far as the cryptanalysis of a stream cipher concerned the following concepts are playing an important role to attack the system Seren (2007):

- **Time Complexity:** Can be defined as required number of operations needs to be processed to apply attack and to reach success.
- **Data Complexity:** It can be defined as the required amount of keystream material which is needed to guarantee the success of attack.
- **Memory Complexity:** It can be defined as the required memory to attack the cipher. It is just like a combination of both the time and data complexity.

The list of the important cryptanalysis methods are listed below

**Trade-off Attacks:** These types of attacks are generally related with two of the three concepts that we have declared above, time, data and memory complexity. The main purpose of the trade-off attacks is to decrease the search complexity of the exhaustive search into data and time complexity, see Babbage (2006) for detailed information.

**Algebraic Attacks:** These types of attacks try to find a linear equation with a higher degree between input and output of the stream cipher. Then, using some techniques that are declared in Shamir and Kipnis (1999), Courtois and Patarin (2003) and Armknecht (2006), try to solve this multivariate equations.

**Distinguishing Attacks:** distinguishes a stream of bits from a perfectly random stream of bits, that is, a stream of bits that has been chosen according to the uniform distribution. See Johansson and Jönsson (1999) for more information.

**Correlation Attacks:** According To Seren (2007) and Johansson and Jönsson (1999), in the correlation attack, attacker aims to find a correlation between input variables to the combining function and the output from the combining function, and then use this correlation to obtain information about the correlated input variables.

# 4. COMPRESSION ALGORITHMS

Generally, we use compression algorithms to compress the data to make them portable. And also the resulting compressed data must be as small as possible and this small amount of data have to contain much of data from the original one. It is more different, when we think about compression in cryptographic manner. The cryptographic compression algorithms have to contain least data in a maximum size that it can hide from the original one.

The cryptographic compression algorithms are especially used to add nonlinearity to the pseudo-random sequences. As we mentioned that pseudo-random sequences has weaknesses to the Algebraic attacks that you can find more information about them in Shamir and Kipnis (1999), Courtois and Patarin (2003) and Armknecht (2006).

We can use compression function while compressing the output of the LFSR like pseudo-random number generator. The compression function takes n bits of input and produces m bits of output, where $n \geq m$. The efficiency of a compression algorithm can be classified to the value of output rate.

Decimation based compression algorithms are good example for cryptographic compression algorithms. These are used as a second structure to compress the output of the PRNG structure. The most common decimation algorithms are SSG, BSG, ABSG, MBSG and EBSG.

## 4.1 A COMPRESSION MODEL FOR PSEUDO-RANDOM GENERATION

As we mentioned before our purpose is to decrease the vulnerabilities of a pseudo-random generator. To make the keystream more secure, we will use some compression algorithms. These compression algorithms delete or insert bits to the original sequence to prevent the keystream from attacks that are related with the algebraic or correlation type properties of pseudo-random outputs.

In Gouget and Sibert (2006, pp. 129-146), the term random input sequences is introduced, those sequences that follow the uniform distribution of binary words:

each word $w$ is a prefix of a random input sequence with probability $1/2^{|w|}$, and all words are assumed to be independent.

### 4.1.1 Prefix Codes and Binary Trees

The definition in Gouget and Sibert (2006, pp. 129-146) is that a binary code is a subset of words of $\{0, 1\}^+$. The language C* of a binary code C is the set of all binary words that are concatenation of words in C. a code C is a prefix code if no codeword has a strict prefix in C. Notice that, in this case, the words of C* parse into codewords in a unique manner. A code is a maximal prefix, when no other prefix code contains it. A code C is right complete if every word $w$ can be completed into a word $v = ww'$ in C.

According to proposition 3.1 in the Gouget and Sibert (2006, pp. 129-146), a code is maximal prefix if, and only if, it is prefix and right complete

*Proof.* Suppose C is maximal prefix. Let $w$ be a non-empty word which has no prefix in c. As c is maximal prefix, $C \cup \{w\}$ is not a prefix code, so $w$ has a right multiple in C. Hence, C is right complete. Conversely, let C be prefix and right complete, and C' be a prefix code that contains C. Let $w \in C$. As C is right complete, $w$ has a right multiple $w'$ in C*. Let them be the smallest prefix of $w'$ in C. As C is prefix, this implies m = w, so we have $w \in C$, and consequently $C' = C$ .Therefore, C is maximal prefix.

### 4.1.2 General Framework

We have an infinite input sequence of bits $s = (s_i)_{i \geq 0}$ ,a binary prefix code C and a mapping $f : C \to \{0,1\}^*$ called the compression function Gouget and Sibert (2006, pp. 129-146). We call *f(C)* the output set. The sequence consist of sequence of codewords $w = (w_i)_{i \geq 0}$ , each $w_i$ being the unique codeword such that $w_0 \, \mathsf{K} \, w_i$ is a prefix of s that belongs to C*. Each $w$ is the n mapped by *f* to its image in *f(C).* The output sequence is $(f(w_i))_{i \geq 0}$ , seen as a bit sequence. We denote this output sequence by

$$y = Enc_{C,f(s)}$$

**Definition 3.1:** The output rate of the pair (C,f), denoted by Rate(C,f), is the average number of output bits generated by one bit of a random input sequence Gouget and Sibert (2006, pp. 129-146).

However, some exceptional cases hold in this situation. For example, we cannot get an output 1 from an input sequence, C = {00}. Continuing from the Gouget and Sibert (2006, pp. 129-146), In order to apply the framework to every possible input sequence, it is then necessary to determine what the requirements on the following components are:

- The choice of C must enable the parsing of every random input sequence,
- The choice of f must be such that, for uniformly distributed input sequences, the corresponding output sequences also follow the uniform distribution.

**4.2 REQUIREMENTS ON C**

First, there are some straight requirements on C. Only the prefix codes are considered in the framework that is expressed in Gouget and Sibert (2006, pp. 129-146). Indeed, if C contained two distinct words $w$ and $w'$ with $w$ a prefix of $w'$, then w would never appear in the decomposition $w$ of s. Therefore, we may delete from C all the codewords that already have a prefix in C without loss of generality, thus transforming C into a binary prefix code. Next, we want every random input sequence to be processable. This implies that C is right complete. Overall, in order to effectively process any random input, we introduce the following definition:

**Definition 3.2:** A binary code C is suitable if it is prefix and if the expected length E(C) of an element of C in the decomposition of a random input sequence is finite.

As mentioned in proposition 3.2 in the Gouget and Sibert (2006, pp. 129-146)**:** For a suitable code C, the following equality holds:

$$\sum_{w \in C} \frac{1}{2^{|w|}} = 1$$

*Proof.* A binary tree corresponds to C. $L_n$ and $N_n$ are denoted as the number of leaves and nodes of depth n. If we have $L_0 = 0,$ then we will have $N_0 = 1,$ and where n ≥ 1, $L_n + N_n = 2N_{n-1}$.

$$S_n = \sum_{0 \le k \le n} \frac{L_n}{2^n} \tag{1}$$

$$\frac{L_n}{2^n} = \frac{N_{n-1}}{2^{n-1}} - \frac{N_n}{2^n} \tag{2}$$

$$S_n = N_0 - \frac{N_n}{2^n} = 1 - \frac{N_n}{2^n} \tag{3}$$

The proof of $N_n = O(2^n)$ will be enough. Now, $N_n$ is the number of nodes of depth n, and a random input sequence begins with n bits corresponding to such a node with probability $\frac{N_n}{2^n}$. For each one of these nodes, the first word of the input sequence recognized as a word of C has length at least n. Thus, these nodes contribute at least $n\frac{N_n}{2^n}$ to E(C). As E(C) is finite, this implies that tends to 0 when n tends to $\infty$.

Therefore, in the case of a suitable code, E(C) is equal to the mean length of the words of C for the uniform distribution on the alphabet {0, 1}. So, according to the proposition 3.3 in the Gouget and Sibert (2006, pp. 129-146), C be a suitable code. Then, the equality is as follows:

$$E(C) = \sum_{w \in c} \frac{|w|}{2^{|w|}}$$

### 4.3 REQUIREMENTS OF f

When we look at the requirements on *f(C)*, *f(C)* can be any set of words including $\varepsilon$ (the empty word). Furthermore, there will be at least two more non-empty words. One of them will star with 1 and the other is 0. It is possible to make random looking

output for random inputs. Moreover, it must be possible to construct every binary sequence with the elements of *f(C)*.

In order to be able to process every random input sequence, we introduce the following definition, which corresponds to the requirement of Definition 3.2:

**Definition 3.3:** Assume that C is suitable and f be the compression function with the definition: $f : C \to \{0,1\}^*$. We can easily say that the pair *(C,f)* is a proper encoder if the expected length E(f(C)) of the image by *f* of an element of C in the decomposition of a randomly chosen input sequence is finite and nonzero, Gouget and Sibert (2006, pp. 129-146).

The Proposition 3.4 in Gouget and Sibert (2006, pp. 129-146) says that for a proper encoder *(C, f)*, the expected length of the image by *f* of an element of C in the decomposition of a randomly chosen input sequence, denoted by E (f(C)), is given by

$$E(f(C)) = \sum_{w \in c} \frac{|f(w)|}{2^{|w|}}$$

Definitions 3.2 and 3.3 ensure the finiteness of E(C) and E(f(C)), so according to the proposition 3.5 in Gouget and Sibert (2006, pp. 129-146), the output rate of a proper encoder *(C, f)* is given by

$$Rate(C, f) = \frac{E(f(C))}{E(C)}$$

The randomness properties and the equality in number of 1s and 0s in the output sequence are provided by the distribution of output sequences. Therefore, we need, for every n ≥ 1:

$$\sum_{w \in C, |f(w)| \geq n, f(w)_n = 0} \frac{1}{2^{|w|}} = \sum_{w \in C, |f(w)| \geq n, f(w)_n = 1} \frac{1}{2^{|w|}}$$

where $f(w)_n$ is the n-th bit of the word *f(w)*.

### 4.3.1 The Prefix Code Output Case

First of all, considering the case where *f(C)* is a prefix code. If it contains two elements, the only possible choice such that the probability distribution of the output for random inputs is that of a random sequence is *f(C)* = {0,1}. In this case, 0 and 1 must have probability 1/2 to appear in the output sequence for a random input sequence Gouget and Sibert (2006, pp. 129-146).

If there are more than 2 elements for *f(C),* then, given a random input sequence, knowing the output sequence, we can retrieve more information on the first element of C than in the case *f(C)* = {0,1}.

According to the proposition 3.6 in Gouget and Sibert (2006, pp. 129-146) again, Let (C, f) be a proper encoder, and, for $x \in f(C)$, let $P(x) = \sum_{w \in f^{-1}} \frac{1}{2^{|w|}}$. Then, for a random input sequence s, each word of the decomposition of s over C has average length E(C), and it is known with average entropy.

$$E(C) + \sum_{x \in f(C)} P(x) \log P(x)$$

*Proof.* For x ∈ *f(C)*, let us denote by $C_x$ the preimage of x in C. Then, the probability that the first element of C recognized in a random input sequence is $P(x) = \sum_{w \in C_x} \frac{1}{2^{|w|}}$. Similarly, the expected length of an element in the preimage of x is $E(C_x) = \frac{1}{P(x)} \sum_{w \in C_x} \frac{|w|}{2^{|w|}}$. At least we compute the entropy on the elements $C_x$ in:

$$H(C_x) = -\sum_{w \in C_x} \frac{1}{P(x)2^{|x|}} \log \frac{1}{P(x)2^{|x|}} = -\sum_{w \in C_x} \frac{1}{P(x)2^{|x|}} \log(P(x) + |w|)$$

$$= \frac{1}{P(x)} \sum_{w \in C_x} \frac{|w|}{2^{|w|}} + \frac{\log P(x)}{P(x)} \sum_{w \in C_x} \frac{1}{2^{|w|}} = E(C_x) + \log P(x)$$

The average number of bits retrieved is therefore $\sum_{x\in f(C)} P(x)E(C_x) = E(C)$ for a random input sequence, so it does not depend on *f(C).* The average entropy is

$$\sum_{x\in f(c)} P(x)(E(C_x) + \log P(x)) = E(C) + \sum_{x\in f(c)} P(x)\log P(x),$$

, with $\sum_{x\in f(c)} P(x)\log P(x) = 1$.

It is usually possible that the given a suitable code C, to divide C into two equiprobable subsets (the probabilities of leaves in the tree being of the form $\frac{1}{2^n}$ with n ≥ 1, and their sum being 1). Thus for every suitable code, there exists a mapping $f : C \to \{0,1\}$ such that 0 and 1 are output with probability 1/2.

Therefore, in order to maximize the entropy for a given suitable code C, the value of $|\sum_{x\in f(c)} P(x)\log P(x)|$ should be as small as possible, which implies *# (f(x))* = 2. Therefore the optimal set is f(C) = {0, 1}, with 0 and 1 having the probability 1/2 to be output for a random input sequence, Gouget and Sibert (2006, pp. 129-146)

### 4.3.2 The Non-prefix Output Case

Consider the cases there are no ε in *f(C),* but also *f(C)* is not a prefix code. Let C(y) be the set of words of C such that, for every $w \in C(y)$, the sequence y begins with *w*. Then, the probability that s begins with w is up to y.

**General Case.** We now suppose that $\varepsilon$ can belong to the output set *f(C).*

According to the proposition 3.7 in Gouget and Sibert (2006, pp. 129-146), Let *(C, f)* be a proper encoder such that *f(C)* contains $\varepsilon$. Then, there exist a proper encoder *(C', f')* such that *f(C')* does not contain $\varepsilon$ and that, for every infinite binary sequence s, that is

$$Enc_{C,f}(s) = Enc_{C',f'}(s).$$

Moreover, defining $P_\varepsilon = \sum_{w \in f^{-1}(\varepsilon)} \frac{1}{2^{|w|}}$, so,

$$E(C') = \frac{1}{1-P_\varepsilon} E(C) \text{ and } E(f'(C')) = \frac{1}{1-P_\varepsilon} E(f(C)).$$

*Proof.* Denote by $C_\varepsilon$ the set of preimages of $\varepsilon$, and by $C_{\overline{\varepsilon}}$ the complement of $C_\varepsilon$ in C. Let C' be the binary code defined by $C' = C_\varepsilon^* C_{\overline{\varepsilon}}$, that is, the set of binary words that parse into a sequence of words of $C_\varepsilon$, followed by a word of $C_{\overline{\varepsilon}}$. Consider the function *f'* that maps each element *ww'* of C', with $w \in C_\varepsilon^*$, and $w' \in C_{\overline{\varepsilon}}$, to *f(w').* As the decomposition is unique, *f' is* well-defined. Moreover, for every input sequence s, the equality $Enc_{C,f}(s) = Enc_{C',f'}(s)$ is obvious satisfied. In the end, $f(C') = f(C) \setminus \{\varepsilon\}$, so the image of *f'* does not contain $\varepsilon$.

There remains to show that the new pair *(C', f')* is also a proper encoder. First, C' is also a prefix code because of ubiquity of the decomposition over C.

Next, as the length of $\varepsilon$ is 0, we have

$$E(f'(C')) = \sum_{v \in C_\varepsilon^*, w \in C_{\overline{\varepsilon}}} \frac{|f(w)|}{2^{|v|+|w|}} = \sum_{n \geq 0} \left( \sum_{v \in C_\varepsilon} \frac{1}{2^{|v|}} \right)^n \times \sum_{v \in C_{\overline{\varepsilon}}} \frac{|f(w)|}{2^{|w|}} = \frac{E(f(C))}{1-P_\varepsilon}$$

As the two encoders *(C,f)* and *(C',f' )* are equivalent, they have the same output rate, which yields the same relation between E(C ) and E(C). Hence, (C ,*f*) is a proper encoder.

Gouget and Sibert (2006, pp. 129-146) proposed that without loss of generality, assume that *f(C)* does not contain $\varepsilon$. So, the optimal choice for *f(C)* is *f(C)* = {0,1}.

## 4.4 TYPES OF COMPRESSION ALGORITHMS

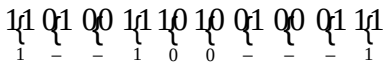
### 4.4.1 The SSG Algorithm

Self-Shrinking Generator is a modified version of the shrinking generator and was firstly presented in (Zenner at al. 2001, pp. 21-35). Assume that we have a random variable X | X = {$x_0$, $x_1$, $x_2$ …}, which was generated by LFSR, used as an input sequence to the SSG. The random variable Z | Z = {$z_0$, $z_1$, $z_2$…}, which was generated from X, is accepted as output. The output rate of the Self-Shrinking Generator is 1/4. According to the definition in (Zenner *at al.* 2001, pp. 21-35), SSG algorithm searches the bits that have the even position, if the value of bit is 1, sets the output bit as the latter bit of the even positioned bit, else the value of even positioned bit is 0, and the algorithm gives no output. We can summarize the SSG algorithm as follows:

```
Set i := 0, j := 0
while (true)
    if (x[i] ≠ 0) z[j] := x[i+1]
    j := j + 1
    i := i + 2
```

The output rate calculation is very simple for SSG algorithm. Assume that we have an evenly distributed input sequence, the occurrence of 1 in even position is 1/2 and also we know that the algorithm gives one output bit to the given two input bits that means we have L/2 output bits, if all of the bits at the even positions are 1 with the input sequence length of a L. Under the conditions that are mentioned above, we can easily say that we have the output rate of 1/4.

*Example 3.1:* Let X = 11010011101001000111 be the input sequence. Then the action of SSG on X can be described as follows:



### 4.4.2 The BSG Algorithm

The Bit Search Generator algorithm is a kind of cryptographic compression algorithm that takes a pseudorandom input with size of L and produces the output with a size of L/3. This algorithm was firstly proposed in Gouget and Sibert (2004, pp. 60-68). And According to the (Gouget *et al.* 2005) and Mitchel (2004) there are two different but equivalent ways to describe it.

The random variable X | X = {$x_0$, $x_1$, $x_2$ …}, which was generated by the LFSR, is accepted as an input to the BSG. The random variable Z | Z = {$z_0$, $z_1$, $z_2$, …}, which was constructed from X, is accepted as output. The BSG algorithm works like this, first of all the $x_0$ from input bit is set as a search bit and then, the algorithm starts to search the bit which has the same value with $x_0$. Assume that we find the correct bit at the position l, if there are no bits between $x_l$ and the search bit, then, the resulting bit will be 0 for output, otherwise, the output bit will be 1. The searching operation can be summarized as follows, the BSG algorithm searches for the patterns $\bar{b}\, b^i \bar{b}$, where $i \geq 0$ and $b \in \{0, 1\}$. If i is equal to 0, then the output will be 0, else the output will be 1. The working principles of BSG algorithm in pseudo code format is as shown in below:

```
Set i := -1, j := -1
while(true)
    i := i + 1
    j := j + 1
    b := x[i]
    i := i + 1
    if (x[i] = b) set z[j] := 0
    else
    while(s[i] ≠ b) i := i + 1
```

The second definition of BSG algorithm called as $BSG_{Diff}$. $BSG_{Diff}$ algorithm works on differential sequence $d_i = x_i \oplus x_{i+1}$, $i \geq 0$. In Hell and Johanssoni (2005), the action of the $BSG_{Diff}$ on the input differential sequence d consist in splitting up the subsequence d into subsequences of the form either (0, b) or (1, $0^i$, 1, b) with $i \geq 0$; for

every such subsequence, the output bit is the first bit of the subsequence. The pseudo code of the $BSG_{Diff}$ is as follows:

```
Set i := 0, j :=0
while (true)
    z[j] := d[i]
    if(d[i] = 1)
        i := i + 1
        while (d[i] = 0) i := i + 1
    i := i + 2
    j := j + 1
```

*Example 3.2:* Let X = 0101001110100100011101 be the input sequence. Then the action of BSG on X can be described as follows:

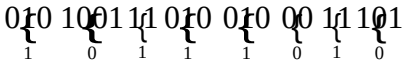
01 0 10 01 1 1 01 0 01 0 00 1 1 101
{ { { { { { { {
1 1 0 1 1 0 0 0

### 4.4.3 The ABSG Algorithm

The ABSG algorithm is the improved version of the BSG algorithm which was proposed in (Gouget *et al.* 2005). The working principle of the ABSG is most likely to the BSG algorithm except the determination of the output bits. In ABSG algorithm, the output bit is selected from the input bit, the second bit of the codeword is used as an output bit. There is no change in the searching process, we look for the codewords $\bar{b}\, b^i \bar{b}$, where i $\geq$ 0. And also the output rate of the ABSG is 1/3. That means, the ABSG algorithm gives N-bit output to the given 3N bit input. If i is equal to zero the output bit will be $\bar{b}$, otherwise *b*. With using the definitions about the input and output bits in section 3.1, we can summarize the algorithm as pseudo code below:

```
Set i := 0, j := 0
while(true)
    b := x[i]
    z[j] := x_{i+1}
    i := i + 1
    while(x[i] = b̄ ) i := i + 1
        i := i + 1
        j := j + 1
```

*Example 3.3:* Let X = 0101001110100100011101 be the input sequence. Then the action of ABSG on X can be described as follows:

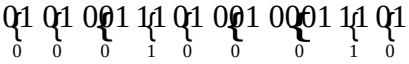


**4.4.4 The MBSG Algorithm:**

MBSG algorithm is the short form of the Modified Bit Search Generator. This algorithm was firstly introduced in (Gouget *et al.* 2005). MBSG has the output rate of 1/3 for evenly distributed inputs. This algorithm searches the subsequences $b0^i1$, where $i \geq 0$ and b is selected as the output bit.

```
Set i := 0, j := 0
while (true)
    y_i := x_i
    i := i + 1
    while (x_i = 0) i := i + 1
    i := i + 1
    j := j + 1
```

*Example 3.4:* Let X = 0101001110100100011101 be the input sequence. Then the action of MBSG on X can be described as follows:



**4.4.5 The EBSG Algorithm:**

The EBSG algorithm, which is the short form of the Editing Bit – Search Generator, was firstly proposed in (Ergüler et al. 2006). This algorithm is the last version of Bit – Search type decimation algorithms that gives the output rate of ½ from N – bit input sequence.

When we consider the working principles of EBSG, it shows similarities with ABSG algorithm except the difference of inserted memory bits. This memory bits after the input sequence of EBSG algorithm and defines the start points of output bits.

Assume that, we have an input sequence X = $\{x_i\}_1^N$, and the corresponding output sequence Z = $\{z_j\}$, and also we have memory bit t with the initial of 0. First of all, the algorithm value searches the codeword $\bar{b}\, b^i \bar{b}$, where $i \geq 0$ from X, if $i$ is odd, the value of t will be $\bar{t}$, otherwise it will protect the its initial value and then t will be inserted into the sequence at the position of after the codeword. the search for new codeword starts for the inserted bit. This operation continues to the end of the input sequence.

```
Set i := 0, j := 0, t := 0
while (true)
    z[i] := x[i+1], b = s[i]
    i := i +1
    while (x[i] = b̄ ) i := i +1, t := t̄
    s[i] := t
    j := j+1
```

*Example 3.5:* Let X = 0101001110100100011101 be the input sequence. Then the action of EBSG on X can be described as follows:

$$\underset{1}{\{}0\underset{1}{\{}1\underset{0}{\{}0\underset{1}{\{}1\underset{1}{\{}1\underset{0}{\{}0\underset{0}{\{}0\underset{0}{\{}0\underset{1}{\{}1\underset{0}{\{}0\underset{1}{\{}1\underset{1}{\{}1$$

# 5. PROBABILISTIC PROPERTIES OF COMPRESSION ALGORITHMS

Generally, decimation algorithms compress the output of the pseudo-number generators. But it is possible that, this algorithm can be used to compress different types of data such as image, voice, etc. In this section, the output rates and output distributions of the compression algorithms (SSG, BSG, ABSG, MBSG and EBSG) are determined for the input sequence of independent identically distributed (i.i.d.) Bernoulli input with *p*. We found that the best result for i.i.d. Bernoulli input is 1/2 which we can say that it is also pseudo-random sequence. If we change the input distribution with respect to *p*, where $0.1 \leq p \geq 0.9$, the output rates and output distribution will be affected too. So, the system will give more information about input text than before (i.i.d. Bernoulli with 1/2). Then, we can analyze the system more easily.

## *5.1 BSG & ABSG:*

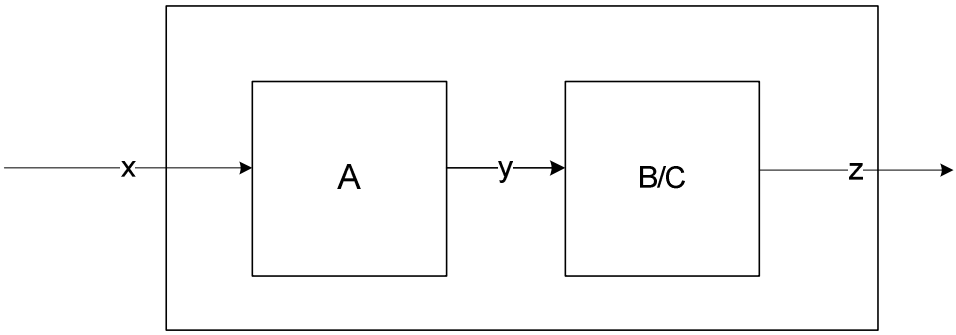
Because of the similarities in their searching structures, in this subsection we will show the probabilistic properties of BSG and ABSG algorithms together.

First of all, we will make some general definitions, the input sequence which is called $X \mid X = \{x_1, x_2 ...\}$, is general for the entire algorithm in the paper. And the output sequence will be the $Z \mid Z = \{z_1, z_2 ...\}$.

We will define the BSG and ABSG together, because they show nearly some operation on the input sequence. According to the Gouget and Sibert (2004, pp. 60-68), BSG algorithm searches the codeword $\bar{b}b^i\bar{b}$, $b \in \{0,1\}$ and $i \geq 0$. If i = 0, the output bit will be 0 otherwise the output bit will be 1. In the ABSG algorithm, the output bits will be b or $\bar{b}$ for the same codewords and rules.

Situation is a bit different. According to the (Gouget et al. 2005), ABSG searches the same codeword $\bar{b}b^i\bar{b}$, if i = 0 the output will be $\bar{b}$, otherwise b.

In the (Altuğ *et al.* 2007), another description of these algorithms is introduced. And the following figure (Figure 2.1) summarizes it. For BSG we use A + B and for ABSG we use A + C to get the output.




Altuğ, Y., Ayerden, N.A., Mihçak, M.K., Anarım, E., December 2006. *A Note on the Periodicity and the Output Rate of Bit –Search Type Generators, IEEE Transactions on* Information Theory (Submitted).

**Figure 8: Block Diagram Representation of BSG and ABSG**

Both ABSG and BSG algorithm use the "A" part of the figure above and A contains a function M (.) which produces the random variable Y | Y = {y1, y2} y to the given x. the definition of M is given the table below. The Ø symbol means that system will give an output. And the initial value of Y random variable is $y_0$ = Ø.

**Table 5.1: Mapping for BSG & ABSG**

| $y_i \backslash x_{i-1}$ | 0 | 1 |
|---|---|---|
| Ø | 0 | 1 |
| 0 | Ø | 0 |
| 1 | 1 | Ø |

Gouget, A., Sibert, H., 2004. *The Bit Search Generator*. The State of the Art of Stream Ciphers: Workshop Recofrd, ss. 60-68.

We can get the following matrix from the table above

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

And the equation

$$A^{2n} = I_3 + \frac{1}{3}(2^{2n} - 1)W_3$$

Continuing from (Altuğ et al. 2007), define α for Ø, β for 0 and θ for 1.And we have the vector $P = (\alpha, \beta, \theta)^T$. And we know that $P_0 = (1, 0, 0)^T$. So,

$$P_1 = A \,.\, P_0$$
$$P_2 = A \,.\, P_1$$
$$\dots$$
$$P_n = A \,.\, P_{n-1}$$

As we know the sequence X is not evenly distributed. The probability of 0 in X is p and also 1 is 1 – p. We have

$$\alpha_n = p.\beta_{n-1} + (1 - p).\theta_{n-1}$$
$$\beta_n = p.\alpha_{n-1} + (1 - p).\beta_{n-1}$$
$$\theta_n = p.\theta_{n-1} + (1 - p).\alpha_{n-1}$$

We will get the probability vector for the first state of the Markov process is that

$$P_1 = (0, p, 1 - p)^T.$$

So the resulting equation is

$$\vec{P}_{2n+1} = A^{2n} \cdot \vec{P}_1$$
$$A^{2n}\vec{P}_1 = (I + \frac{1}{3}(2^{2n} - 1)W_3)\vec{P}_1$$

And we will have the vector

$$\vec{P}_{2n+1} = \left(\frac{1}{3}(2^{2n} - 1), p + \frac{1}{3}(2^{2n} - 1), (1 - p) + \frac{1}{3}(2^{2n} - 1)\right)^T$$
$$= (\alpha_{2n+1}, \beta_{2n+1}, \theta_{2n+1})^T$$

When the working principles of ABSG and BSG algorithms are concerned, they show a few dissimilarities and these are described in B and C parts of the figure 2.1. The definition of B is as follows

$$Z_j = \begin{cases} 0, & if\ y_i = \phi \text{ and } y_{i-2} = \phi \\ 1, & if\ y_i = \phi \text{ and } y_{i-2} \neq \phi \end{cases}$$

, where $j \leq i$

The algorithm C can be given as follows

$$Z_j = \begin{cases} y_{i-1}, & if\ y_i = \phi \text{ and } y_{i-2} = \phi \\ y_{i-2}, & if\ y_i = \phi \text{ and } y_{i-2} \neq \phi \end{cases}$$

, where $j \leq i$

The probabilities of the $Y_1^N$ which is a Markov process of memory one with the initial condition $Y_0 = \phi$, is as follows

$$\frac{1}{2} = \Pr(Y_i = \phi \mid Y_{i-1} \neq \phi) = \Pr(Y_i \neq \phi \mid Y_{i-1} \neq \phi)$$

$$1 = \Pr(Y_i \neq \phi \mid Y_{i-1} = \phi)$$

These probabilities are independent of the distribution of the input sequence. As we can easily see that from Table 1, whatever the p values of X is that the output rates of the BSG and the ABSG won't change. So the result will be same as in the (Altuğ et al. 2007).

$$E[H/N] = \frac{E[H]}{N} = \frac{1}{3} - \frac{2}{9N} + \frac{2}{9N}\left(-\frac{1}{2}\right)^N$$

## *5.2 MBSG*

MBSG is another improved version of the BSG algorithm. As we defined before, the MBSG, which was proposed in Mitchel (2004), searches the codeword $b0^i1$, where $i \geq 0$ and $b \in \{0,1\}$. In addition to that, we can also define the MBSG algorithm as follows:

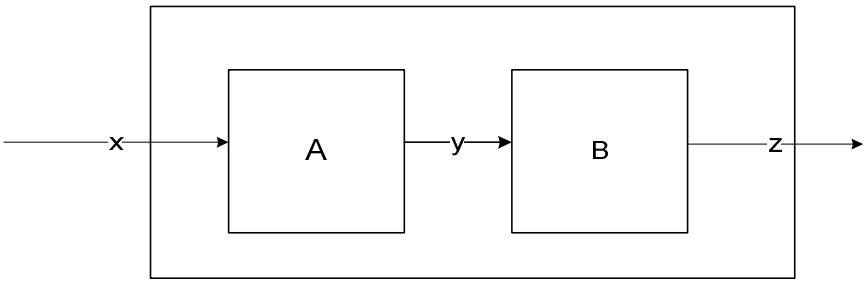



**Figure 9: Another definition of the MBSG algorithm**

The algorithm A contains the function M (.) which is defined in the following table

**Table 5.2: Mapping for MBSG**

| $y_{i-1} \setminus x_i$ | 0 | 1 |
|---|---|---|
| Ø | 0 | 1 |
| 0 | 0 | Ø |
| 1 | 1 | Ø |

, where $y_0 = Ø$

The definition of the B algorithm is that $Z_j = y_{i-1}$ if $y_i = Ø$, where $j \leq i$

From Table 2, we will get the matrix A, which is as same as in (Altuğ et al. 2007), is shown below.

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

And we will also use the same equation

$$A^{2n} = I_3 + \frac{1}{3}(2^{2n} - 1)W_3$$

where,

$$W_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For calculating the output distribution of MBSG we will define the probability vector P = {α , β , θ} which is as same as in the (Altuğ et al. 2007). Continuing from (Altuğ et al. 2007) we will get the $\vec{P}_1$ = (0, 1-p, 1-p).

To sum up, the resulting equation is that

$$\vec{P}_{2n+1} = A^{2n}\vec{P}_1 = \left( I + \frac{1}{3}(2^{2n}-1)W_3 \right)\vec{P}_1$$
$$= \left( \frac{2}{3}(2^{2n}-1)(1-p), \quad (1-p)(1+\frac{2}{3}(2^{2n}-1)), \quad (1-p)(1+\frac{2}{3}(2^{2n}-1)) \right)^T$$
$$= \left(\alpha_{2n+1}, \quad \beta_{2n+1}, \quad \theta_{2n+1}\right)$$

Because of the structure of the M function in MBSG the output rate will be change due to the probability distribution of input sequence. As far as the working principle of MBSG concerned we can easily see the same results as in the later subject. Because, the MBSG searches for the codeword b $0^i$ 1, that means, MBSG looks for 1 after starting a new code to give an output. So we can say that the output probabilities are as shown in below with the initial condition $y_0 = \phi$

$$\Pr(y_i = \phi \mid y_{i-1} \neq \phi) = 1 - p$$
$$\Pr(y_i \neq \phi \mid y_{i-1} \neq \phi) = p$$
$$\Pr(y_i \neq \phi \mid y_{i-1} = \phi) = 1$$

Since we are dealing with the Ø; we can define a new random variable an as in (Altuğ et al. 2007) to clarify the calculating output rate as in follows,

$$Q_i = \begin{cases} 1, & if\ \mathrm{y_i} = 0. \\ 0, & otherwise. \end{cases}$$

we have

$$\Pr(Q_i = 1 \mid Q_{i-1} = 0) = 1 - p$$
$$\Pr(Q_i = 0 \mid Q_{i-1} = 0) = p$$
$$\Pr(Q_i = 0 \mid Q_{i-1} = 1) = 1$$

Assume that we have n blocks that contribute the b $0^i$ 1, i ≥ 0 structure and also we have a function L(.), which gives the length of the block as in (Altuğ et al. 2007). So we have the following equation,

$$\Pr(L(Block_n) = l, Q_n = 1)$$

$$= \Pr(Q_n = 1, (Q_i = 0, n < i < n + l), Q_{n+l})$$

$$= \Pr(Q_{n+l} = 1, Q_{n+l-1} = 0, \kappa, Q_{n+1} = 0, Q_n = 1)$$

$$= \underbrace{\Pr(Q_{n+l} = 1 \mid Q_{n+l-1} = 0)}_{1-p} \cdot \underbrace{\prod_{i=n+2}^{n+l-1} \left[\Pr(Q_i = 0 \mid Q_{i-1} = 0)\right]}_{p^{m-n-2}}$$

$$\cdot \underbrace{\Pr(Q_{n+1} = 0 \mid Q_n = 1)}_{1} \cdot \underbrace{\Pr(Q_n = 1)}_{\alpha_n}$$

$$= (1-p) \cdot p^{l-2} \cdot \alpha_n$$

We have found an expression for the probability of occurring a length *l* block, it is time to find the expected $W_{Ø}(Y_1^N)$, which is created by A algorithm and independent identically distributed Bernoulli with *p*. We have also two cases for this situation as in (Altuğ et al. 2007).

*i.* $Q_N = 1,\ \Pr(H = k \mid Q_N = 1)$

$$= \sum_{\forall l_i \mid 1<i<k+1} \Pr(L(Block_1) = l_1, \kappa, L(Block_k) = l_k \mid Q_0 = 1)$$

$$= \sum_{\forall l_i \mid 1<i<k+1} (1-p) \cdot p^{L_1-2} \cdot (1-p) \cdot p^{L_2-2} \kappa (1-p) \cdot p^{L_k-2}$$

$$= \sum_{\forall l_i \mid 1<i<k+1} (1-p)^k \cdot p^{N-2k}$$

And we have $\begin{pmatrix} N-k-1 \\ k-1 \end{pmatrix}$ ways to put $k-1$ "10" pattern in N – 2 locations.

So we will have,

$$\begin{pmatrix} N-k-1 \\ k-1 \end{pmatrix} \cdot (1-p)^k \cdot p^{N-2k-1}$$

*ii.* $Q_N = 0,$ that means we will have k "10" pattern in N-1 locations so,

$$\Pr(H = k \mid Q_N = 0) = \begin{pmatrix} N-k-1 \\ k \end{pmatrix} \cdot (1-p)^k \cdot p^{N-2k-1}$$

When we try to calculate first and second moments to get the output rate of MBSG algorithm which is E[H/N] as in (Altuğ et al. 2007).

$$E[H]=\sum_{i=1}^{N}\Pr(Q_i=1)=\sum_{i=1}^{N}\frac{2}{3}-\frac{2p}{3}\cdot(2^i-1)$$

$$=\frac{2N}{3}+\frac{2Np}{3}-\frac{2p}{3}\cdot(1-2^N)=\frac{2}{3}(N+p^N-p-2^N p)$$

Since,

$$E[H/N]=\frac{E[H]}{N}=\frac{2}{3}+\frac{2p}{3}-\frac{2p}{3N}-\frac{2^{N+1}p}{3N}$$

## *5.3 SSG*

According to the definition of SSG, which was introduced in the former section, divides the input sequence into 2 – bit codewords, if the first bit of the codeword is zero, there will be no output; otherwise, the output will be the second bit of the codeword.

In other words, the definition of SSG is as follows; assume that the input sequence is $X=(X_i)_1^N$ and resulting output sequence is $Y=\{y_i\}_1^N$, which $y_j = x_{2i}$, if $x_{2i-1}=1$ and $0 < j \leq i$.

When we consider the probabilistic properties of SSG, it normally has the output rate of ¼ that means if we have 4n – bit input, we will have n – bit output.

**Claim:** $\Pr[y_i=1]=(1-p),\ \Pr[y_j=0]=p$

**Proof:** $\Pr[y_j=1]=\Pr[x_{2i}=1]=1-p,$ for some i > j

**Claim:** (Given N – bit long input)

**Proof:** $\Pr[M=i]=\binom{\lfloor N/2 \rfloor}{p}\cdot(1-p)\cdot p^{\lfloor N/2 \rfloor - i}$, for $0 \leq i \leq \lfloor N/2 \rfloor$

**Corollary:** The output rate of the SSG algorithm for the i.i.d. Bernoulli with p input is

$$\frac{E[M]}{N} = \frac{\lfloor N/2 \rfloor \cdot (1-p)}{N}$$

As $N$ à $\infty$,

$$M \approx N(\lfloor N/2 \rfloor (1-p), \lfloor N/2 \rfloor (1-p).p)$$

## *5.4 EBSG*

In the previous subsections, the output rates of the algorithms have been calculated to the codeword and it's characteristic. However, in EBSG, the output rate of the algorithm is up to the number of inserted memory bits. As described above, every inserted bit starts the new codeword that will give output. So, we will have nearly n + 1 output bits (n is used to show the number of inserted bits).

**Definition 4.1:** We can define the number of inserted bits as a random variable, $T = \{t_i\}_0^n$.

**Claim:** Assume that the inserted bits are the unknown sized which means that the number of memory bit is up to the given input sequence (if the input sequence is not an evenly distributed sequence), the inserted bits can be represented as a Markov process with infinite memory.

**Proof:** Before proving the inserted memory bits are Markov process with order 1, we have to make some definitions.

**Definition 4.2:** The most important part of the EBSG is that the value of i in $\bar{b}b^i\bar{b}$ codeword. If it is even value of the inserted memory bit won't be change. Else we will take the compliment of the previous one.

**Definition 4.3:** There are some differences between the odd case and the even case of the first codeword. These are summarized as follows.

**Note:** Consider the probability of 0 in the input pattern is *p* and also for 1 we have the probability *1 – p.*

**Even Case:** we know that the initial condition of t is 0. For the case that getting the $t_1 = 0$, we have to find the following patterns:

00, 11, 0110, 1001 …

The probability of even case is

$$P_{even} = p^2 \sum_{i=0} (1-p)^{2i} + (1-p)^2 \sum_{i=0} p^{2i}$$

$$= \frac{p^2}{1-(1-p)^2} + \frac{(1-p)^2}{1-p^2} = \frac{p}{2-p} + \frac{1-p}{1+p} = \frac{2-2p+2p^2}{2+p-p^2}$$

**Odd Case:** To get the value of 1 for the first memory bit t1, we have to find the following patterns:

010, 101, 01110, 10001, …

$$P_{odd} = p^2 \sum_{i=0} (1-p)^{2i+1} + (1-p)^2 \sum_{i=0} p^{2i+1}$$

$$= \frac{p^2(1-p)}{1-(1-p)^2} + \frac{(1-p)^2 p}{1-p^2} = \frac{p-p^2}{2-p} + \frac{p-p^2}{1+p} = \frac{3p(1-p)}{2+p-p^2}$$

**Definition 4.4:** As we mentioned before the insertion operation in EBSG algorithm starts after the first codeword. So that we have to find the following patterns:

**Even Case:**

0, 1, 110, 001, …

$$P'_{even} = p \sum_{i=0} (1-p)^{2i} + (1-p) \sum_{i=0} p^{2i} = \frac{1}{1+p} + \frac{p}{2-p} = \frac{2+p^2}{2(1+p-p^2)}$$

**Odd Case:**

01, 10, 1110, 0001,…

$$P'_{odd} = p \sum_{i=0} (1-p)^{2i+1} + (1-p) \sum_{i=0} p^{2i+1} = \frac{1-p}{2-p} + \frac{p}{1+p} = \frac{2p^2-2p-1}{4(p^2-p-1)}$$

Without loss of generality, the inserted memory bits can be represented as in the following state machine figure.

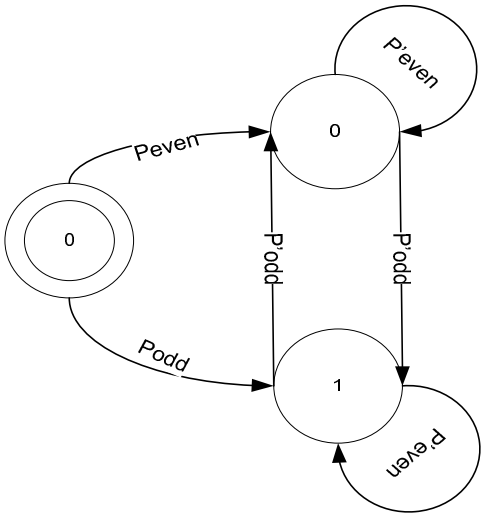



**Figure 10: STM for probabilities of memory bits**

As mentioned before, the bit insertion operation is the Markov process with infinite memory and the following table summarizes this condition. Assume that the initial value of T, $t_0 = 0$.

**Table 5.3: Mapping for Insertion bits**

| $t_{i-1} \backslash m_j \% 2$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

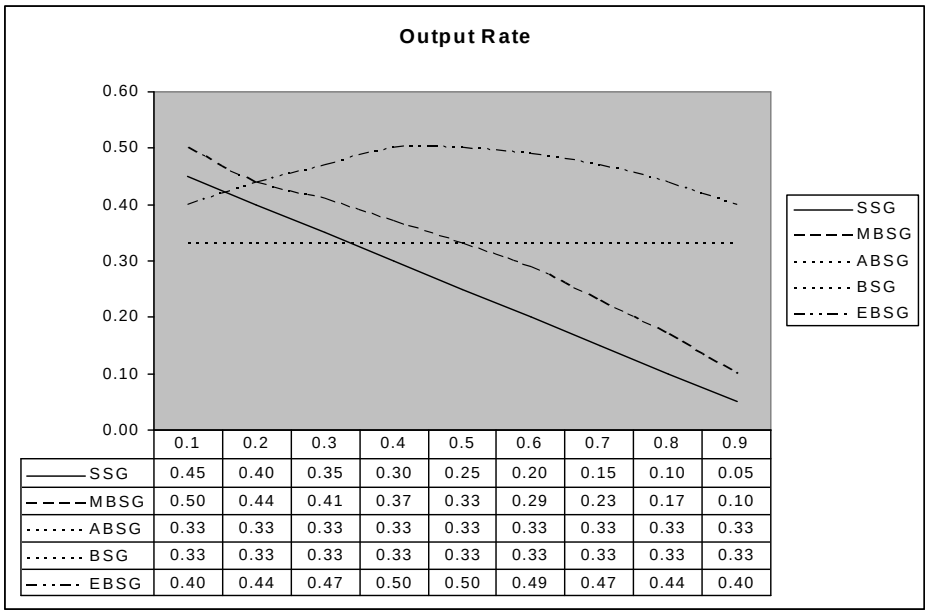
The reset operation that we have mentioned in the former sections does not appear in the table above. The reset operation clear the states and we do not have to know anything before. This situation also shows that the insertion bits are dependent. This dependency makes the calculation of output rate more complicated and insufficient. So, the output rate will be stay at an experimental level for this study.

The properties of EBSG that we have introduced above, brings some weaknesses for the security concept. Considering the structure of EBSG we have proposed an attack, for which the details of the attack was mentioned in the latter section, is called the backtracking attack.

## 5.5 EXPERIMENTAL RESULTS

In this subsection we will deal with experimental results of different distributions of input sequences on the output rate and output distribution. As we mentioned before compression algorithms use pseudo-random inputs which is i.i.d. Bernoulli with 1/2. When we used another distribution rate instead of 1/2, the change in the output rates and the output distribution will give more information about the input sequence.

First of all, if we consider the output rates of the algorithms as shown in the figure below. The strongest algorithms are BSG and ABSG because of they produces outputs with the same rates. As far as derivations in section 5.1 is concerned, you can easily see that the input distribution of these algorithms do not affect the output rates of the output rates. But we cannot say such things for SSG, MBSG and EBSG algorithm; they produce different output rates for different input distributions. The last thing that we can say for the output rate is that if the algorithm produces output with respect to the some constant (e.g. we can say 1 for MBSG and SSG), we cannot get the constant output rate for this algorithm.



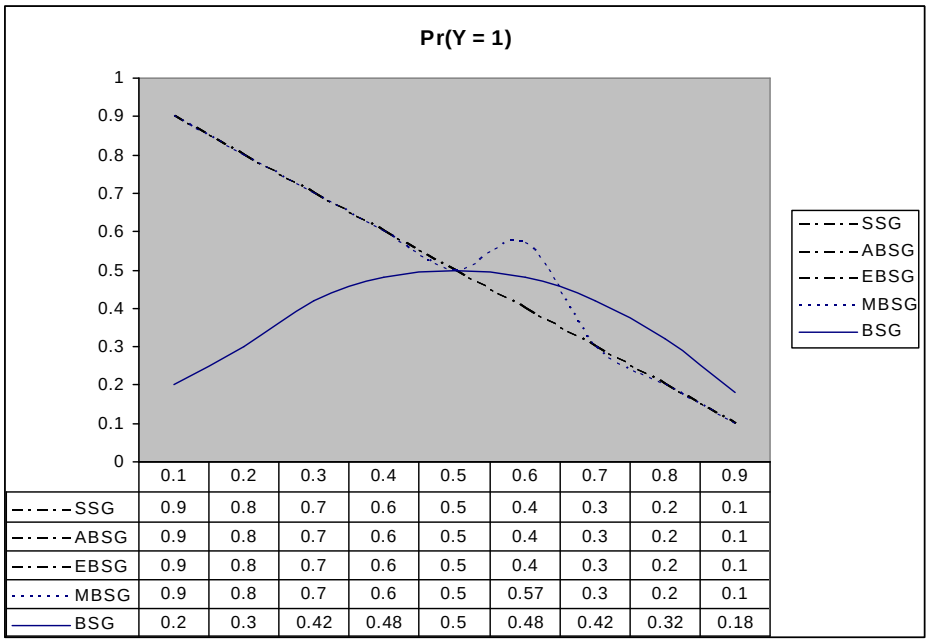

| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|
| SSG | 0.45 | 0.40 | 0.35 | 0.30 | 0.25 | 0.20 | 0.15 | 0.10 | 0.05 |
| MBSG | 0.50 | 0.44 | 0.41 | 0.37 | 0.33 | 0.29 | 0.23 | 0.17 | 0.10 |
| ABSG | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 |
| BSG | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 |
| EBSG | 0.40 | 0.44 | 0.47 | 0.50 | 0.50 | 0.49 | 0.47 | 0.44 | 0.40 |

**Figure 11: Variation of the output rates of compression algorithms**

Another important point is that the distribution of 0s and 1s in the output sequence. As shown in the figures (figure 11 and figure 12) below, it is easily seen that we can get the input distribution from output distributions for all algorithms (except BSG).

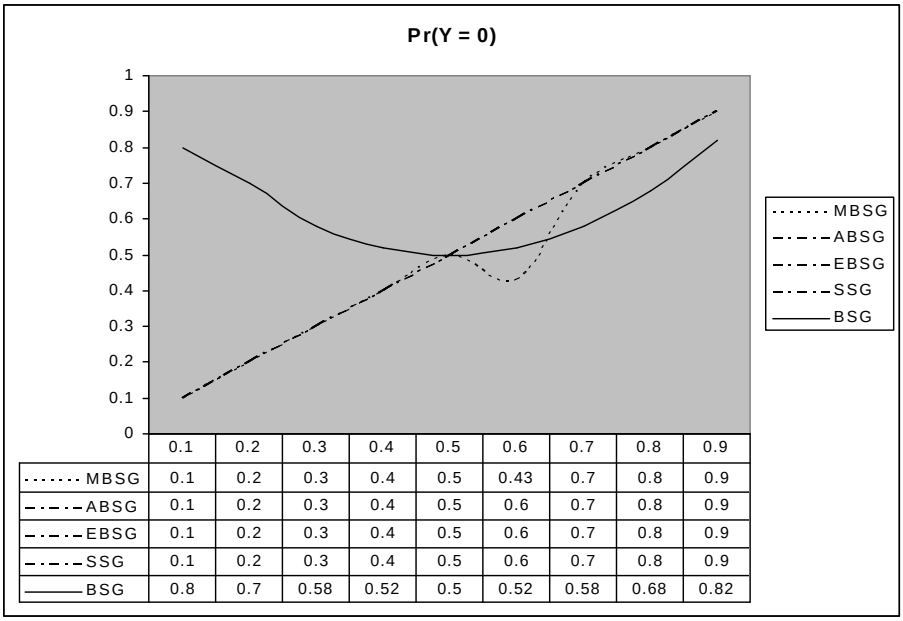
As mentioned before, we use compression algorithms for adding nonlinearity to the linear pseudo-random key sequence. But in this case we give more information about the input sequence because its distribution is as same as the output distribution. Attacker, who wants to crack the system, will perform less effort to get the correct input sequence. And also it is important to remember that we have to be careful about input – output correlation of ciphers, especially in the key stream design stage.

One the most important thing in the cryptographic algorithm design is that the hiding the content of secret key. As far as the experimental results and theoretical derivations of compression algorithms are concerned, these algorithms are not suitable for key stream design if we don't use pseudo-random inputs.




| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|
| SSG | 0.9 | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 | 0.3 | 0.2 | 0.1 |
| ABSG | 0.9 | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 | 0.3 | 0.2 | 0.1 |
| EBSG | 0.9 | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 | 0.3 | 0.2 | 0.1 |
| MBSG | 0.9 | 0.8 | 0.7 | 0.6 | 0.5 | 0.57 | 0.3 | 0.2 | 0.1 |
| BSG | 0.2 | 0.3 | 0.42 | 0.48 | 0.5 | 0.48 | 0.42 | 0.32 | 0.18 |

**Figure 12: Distribution of 1s in the output sequences**




| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|
| MBSG | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.43 | 0.7 | 0.8 | 0.9 |
| ABSG | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| EBSG | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| SSG | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| BSG | 0.8 | 0.7 | 0.58 | 0.52 | 0.5 | 0.52 | 0.58 | 0.68 | 0.82 |

**Figure 13: Distribution of 0s in the output sequences**

# 6. THE BACKTRACKING ATTACK

EBSG algorithm produces L/2 bits output to the given L-bit evenly distributed input. For this situation, in (Ergüler et al. 2006), authors used an attack, which is called most favorable case attack, which was firstly introduced in (Gouget et al. 2005). This attack works on the EBSG algorithm with the complexity of O ($2^{L/2}$) and requiring O ($\frac{4L}{3} 2^{L/2}$) output bits. The proposed attack is based on the assumption of guessing the first inserted bit and then determining the entire inserted bits to the first guess.

As far as the properties of the EBSG algorithm is concerned, analyzing the inserted bits to the output sequence, starting from the first one to the last one is insufficient. There are different probabilities in the case of odd and even codewords which affects the security of the algorithm. As mentioned in the previous section, the probability of getting even codeword is higher than the probability of getting odd codeword. As described the former section, we can get the even codeword with probability of 2/3 and the getting an odd codeword is also 1/3, if we consider the input sequence as an evenly distributed input. That means we have at most 1/3 of N codewords will be odd in the input stream that is why we are using for the backtracking attack.

In the output sequence, each output bit comes from *bb* or $\overline{b}\, b^i \overline{b}$. The random variables (T and X) form the input sequence of EBSG. The X is the pseudorandom output and the T is the sequence of inserted memory bits. After that, the combination of these two random numbers is used as an input sequence to the EBSG algorithm. These are both evenly distributed sequences, so the occurrences of codeword *bb* and $\overline{b}\, b^i \overline{b}$ , where i>0, will be equal which is another important concept for the attack.

In the backtracking attack, we are starting to analyze the output sequence starting from the last bit. As mentioned in the section 5.4, the insertion bits are dependent to each other. On the average, we have L/2 codeword in the input sequence which is the number output bits. If the last bit in the input sequence is b, this can come from the sequences bb or $\overline{b}\, b^i \overline{b}$ , where i>0. In this type of situation the occurrence of the input

sequence is more likely to *bb* rather than $\bar{b}\,b^{i}\,\bar{b}$. So, if the insertion bit is preserved while generating the last codeword that means the former codeword is even, this situation is being realized with the probability of 2/3. If the (L/2)-1$^{th}$ output is $\bar{b}$, the (L/2)-1$^{th}$ codeword will be $b\,\bar{b}^{2i}b$ type codeword. If we assume that the insertion bit is changed, this situation occurs with the probability of 1/3, and then we will look at the reverse situation. And then the backtracking operation continues to the beginning of the input sequence.

For example, if the (L/2)$^{th}$ output bit is 1 and the most favorable codeword for this output bit is 11, if the (L/2)-1$^{th}$ codeword is even and the (L/2)-1$^{th}$ output bit is 0 then the (L/2)-1$^{th}$ codeword will be $10^{2i}1$, where i>0. That means we will get at least three bits from input sequence and one bit from insertion bits. It is possible to get the codewords like *bb*, which will results with getting one bit from memory bits and one bit from input sequence. If the codeword is $\bar{b}\,b^{i}\,\bar{b}$, we have two situations. The first one is getting even codeword, $\bar{b}\,b^{2i}\,\bar{b}$, this will result with getting one bit from memory bits and at least three bits from input sequence. The latter one is that getting odd codeword, $\bar{b}\,b^{2i+1}\,\bar{b}$. In this situation, we will get one bit from memory bits and at least two bits from input sequence.

If the last bit of the output is b that comes from the bb like codeword in the most probable case. Assume that the value of the insertion bit is conserved in the former codeword. And the former output bit is $\bar{b}$ that means the former codeword is $b\overline{b^{2i}}b$. So the calculation will be as follows:

Pr [$w_{l-1}$ = b, edit bit preserved]

= Pr [$w_{l-1}$ = b | edit bit preserved] · Pr [edit bit preserved]

= X · 2/3

= 3/7*2/3

= 2/7 (probability of getting the correct sequence)

* X: $\sum_{i=1}^{\infty} X\left(\frac{1}{4}\right)^{i} = 1 \Rightarrow X = 3/7$

Another important point of the backtracking attack is that the number of trails of the backtracking operation. As mentioned before the occurrence of even codeword higher than the occurrence of odd codewords. So, in EBSG algorithm probability of getting the odd codewords will be at most 1/3. And also if we have L/2-bit output sequence, the number of output bits that are come from the odd codewords will be at most L/6.

We can represent the distribution of odd codewords in the input sequence as $\sum_{i=0}^{L/6}\binom{L/2}{i}$.

Complexity of an attack is the most important issue in the cryptanalysis of an algorithm. To calculate the complexity of the backtracking attack, we firstly define some concepts that we have already defined before. EBSG has a pseudorandom input with a size of L and produces the L/2-bit output. The number of output bits is also the number of codewords in the input stream, L/2. And we also know that on the average half of the codewords come from the sequence *bb*. So we have L/4 *bb* codewords, that gives L/4 bits from input sequence. The rest of the input sequences will come from $\bar{b}\, b^i \bar{b}$ like codewords. The total number of even sequences is L/3, so we have L/12 more even sequences. According to the definition of the backtracking attack we will have L/4 bits more from input sequences. We have also odd sequences which are L/6 and that give at least L/3 bits from input from input sequences. Totally, we know 5L/6 of input sequence. That means we have to search L/6 bit of input sequence with the complexity of $O(2^{L/6})$ on the average, for each try. The average requiring bits that are used to complete the attack with both the least and the most favorable cases for last codeword is that the $O\left(\sum_{i=0}^{L/6}\binom{L/2}{i}2^{(L-(L/4+3(L/4-i)+2i))+1}\right)$ instead of $O(\frac{4L}{3}2^{L/2})$. The results of the backtracking attack lower than most favorable case attack.

# 7. CONCLUSION AND FUTURE WORK

During this study we have evaluated the probabilistic properties of compression algorithms to the concept of different distributions of input sequences. As a result of the calculations of probabilistic properties and the experimental results, it was shown that the ABSG algorithm is the best in the concepts of security and the output rate for different type of distributions.

We have also analyzed the EBSG algorithm in a cryptanalytic manner; we have proposed an attack for this algorithm called the backtracking attack. This attack is more efficient than most favorable case attack (Ergüler et al. 2006) in the cases of complexity and the required memory bits.

For future work, the results of the probabilistic properties section, there can be a new algorithm that considers the probabilistic distribution of the input sequence and produces the evenly distributed output. The structure of the ABSG algorithm is the most suitable one for this type of situation. The output rate of the ABSG algorithm is 1/3 for every distribution of the input sequence and also it directly reflects the probability distribution of the sequence to the output sequence. As shown in the EBSG algorithm, we can insert or remove bits from input sequence to balance the input distribution.

# REFERENCES


Altuğ, Y., Ayerden, N.A., Mihçak, M.K., Anarım, E., December 2006. *A Note on the Periodicity and the Output Rate of Bit –Search Type Generators, IEEE Transactions on* Information Theory (Submitted).

Armknecht F., 2006. *Algebraic Attacks on Stream Ciphers.* Saint Petersburg : Eurocrypt 2006.

Babbage, S., 2006. *"A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers*". Newbury : Vodafone Ltd.

Boztaş, S., 2007. *Dictators, Influences, Majorities and Tribes and Boolean Functions, Finite Fields and Their Applications* : Melbourne, Australia.

Carlet, C., 2006. *"Boolean Functions for Cryptography and Error Correction Codes"*. Cambridge : Cambridge University Press.

Courtois N., Patarin J., 2003. *About the XL Algorithm over GF(2)* : Cryptographers' Track RSA 2003, San Fransisco 2003, LNCS, Springer.

Ergüler, İ., Anarım, E., 2006. *The Editing Bit-Search Generator*, Boğaziçi University Technical Report.

Goldreich, O., 2001. *Foundations of Cryptography*. Cambridge: Cambridge University Press.

Gouget, A., Sibert, H., 2004. *The Bit Search Generator*. The State of the Art of Stream Ciphers: Workshop Recofrd, ss. 60-68.

Gouget, A., Sibert, H., Berbain, C., Courtois, N., Debbraize, B., Mitchell, C., 2005 *Analysis of Bit-Search Generator and Sequence Compression Technique* : Fast Software Encryption.

Gouget, A., Sibert, H., 2006. *How to Strengthen Pseudo-random Generators by Using Compression* : Springer-Verlag, EUROCRYPT 2006, Lecture Notes in Computer Sciences 4004, pp.129–146.

Hell, M., Johanssoni T., 2005. *Some Attacks on the Bit-Search Generator* : Fast Software Encryption.

Jansen, S.J.A., 2004. *Stream Cipher Design: Make your LFSR jump!* : Workshop Record ECRYPT-State of the Art of Stream Ciphers, pp. 94-108.

Johansson, T., Jönsson, F., 1999 *"Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes"* : Advances in Cryptology - EUROCRYPT '99.

Krause, M., 1999. *BDD Based Cryptanalysis of Keystream Generators* : Springer-Verlag, Lecture Notes In Computer Science, 2002 ISSU 2332, pp 222-237.

Mao, W., 2003. *Modern Cryptography: Theory and Practice* : Prentice Hall PTR,

Menezes, A.J., van Oorschot, P.C., Vanstone, S.A., 1997. *Handbook of Applied Cryptography* : CRC Press.

Mitchel, C.J., 2004. *Some Observations on the Bit Search Generator*. London : Technical Report RHUL–MA–2004–3, Department of Mathematics Royal Holloway, University of London.

Paul, S., Preneel, B., Sekar, G., 2006. *"Distinguishing Attacks on the Stream Cipher Py"* : Fast Software Encryption.

Seren, Ü., 2007. *Analysis of Compression Techniques and Memory Bit Efffects on Compression for Pseudo-Random Generation*. İstanbul : M.S. Thesis, Boğaziçi University, Electrical and Electronics Engineering Department.

Shamir A., Kipnis A., 1999. *Cryptanalysis of the HFE Public Key Cryptosystem* : In Advances in Cryptology, Proceedings of Crypto'99, Springer Verlag.

Soong, T.T., 2004. *Probability and Statistics for Engineers* : John Wiley and Sons,

Stallings, W., 2003. *Cryptogrphy and the Network Security*. New Jersey : Prentice Hall, 3rd Edition

Stamp, M., Low, R.M., 2007. *Applied Cryptanalysis: Breaking Ciphers in the Real World* : John Wiley and Sons,

Stinson, D., 1995. *Cryptoghraphy: Theory and Practice*: CRC Press.

Zenner, E., Krause, M., Lucks, S., 2001. *Improved Cryptanalysis of the Self-Shrinking Generator*. Proceedings of the 6th Australasian Conference on Information Security and Privacy, pp. 21 – 35

**VITA**

Orhan ERMİŞ was born in İstanbul. He received his B.S. degree in Computer Engineering from the Bahçeşehir University in 2005. Since then he has been a research assistant in the Department of Computer Engineering. His main areas of interest are cryptology, complexity theory and networking.