Correlation attacks on stream ciphers using convolutional codes

by

C.S. Bruwer

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Electronic Engineering)

in the

Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

October 2004

Correlation attacks on stream ciphers using convolutional codes

by

Christian Bruwer

Promoter: Prof. W. T. Penzhorn

Department: Electrical, Electronic and Computer Engineering

Degree: Master of Engineering (Electronic)

Keywords

Stream cipher; Non-linear combining function; Cryptanalysis; Correlation attack; Linear Feedback Shift Register; Viterbi algorithm; Lempel-Ziv complexity; Binary derivative; Binary discriminator.

Summary

This dissertation investigates four methods for attacking stream ciphers that are based on nonlinear combining generators:

- Two exhaustive-search correlation attacks, based on the binary derivative and the Lempel-Ziv complexity measure.
- A fast-correlation attack utilizing the Viterbi algorithm
- A decimation attack, that can be combined with any of the above three attacks

These are ciphertext-only attacks that exploit the correlation that occurs between the ciphertext and an internal linear feedback shift-register (LFSR) of a stream cipher. This leads to a so-called divide and conquer attack that is able to reconstruct the secret initial states of all the internal LFSRs within the stream cipher.

The binary derivative attack and the Lempel-Ziv attack apply an exhaustive search to find the secret key that is used to initialize the LFSRs. The binary derivative and the Lempel-Ziv complexity measures are used to discriminate between correct and incorrect solutions, in order to identify the secret key. Both attacks are ideal for implementation on parallel processors. Experimental results

show that the Lempel-Ziv correlation attack gives successful results for correlation levels of p = 0.482, requiring approximately 62000 ciphertext bits. And the binary derivative attack is successful for correlation levels of p = 0.47, using approximately 24500 ciphertext bits.

The fast-correlation attack, utilizing the Viterbi algorithm, applies principles from convolutional coding theory, to identify an embedded low-rate convolutional code in the pn-sequence that is generated by an internal LFSR. The embedded convolutional code can then be decoded with a low complexity Viterbi algorithm. The algorithm operates in two phases: In the first phase a set of suitable parity check equations is found, based on the feedback taps of the LFSR, which has to be done once only once for a targeted system. In the second phase these parity check equations are utilized in a Viterbi decoding algorithm to recover the transmitted pn-sequence, thereby obtaining the secret initial state of the LFSR. Simulation results for a 19-bit LFSR show that this attack can recover the secret key for correlation levels of p = 0.485, requiring an average of only 153,448 ciphertext bits.

All three attacks investigated in this dissertation are capable of attacking LFSRs with a length of approximately 40 bits. However, these attacks can be extended to attack much longer LFSRs by making use of a decimation attack. The decimation attack is able to reduce (decimate) the size of a targeted LFSR, and can be combined with any of the three above correlation attacks, to attack LFSRs with a length much longer than 40 bits.

Korrelasie aanvalle op stroomsyfers deur die gebruik van konvolusiekodes

deur

Christian Bruwer

Promotor: Prof. W. T. Penzhorn

Departement: Elektriese, Elektroniese en Rekenaar-Ingenieurswese

Graad: Meester in Ingenieurswese (Elektronies)

Sleutelwoorde

Stroomsyfer; Nie liniêre kombineer-funksie; Kritpto-analise; Korrelasie-aanval; Liniêere-terugvoer skuifregister; Viterbi algoritme; Lempel-Ziv kompleksiteit; Binêre afgeleide; Binêre diskrimineerder.

Opsomming

Hierdie verhandeling ondersoek vier metodes om stroomsyfers, gebaseer op nie-liniêre kombinatoriese generators, aan te val:

- Twee korrelasie aanvalle, gebasseer op die binêre differensiaal en die Lempel-Ziv komplexiteit maatstaaf, deur middel van 'n volledige sleutel-soektog
- 'n Vinnige korrelasie-aanval wat gebruik maak van die Viterbi algoritme
- 'n Desimasie aanval wat gekombineer kan word met enige van die drie bogenoemde aanvalle.

Hierdie is syferteks-aanvalle wat die korrelasie tussen die syferteks en 'n interne liniêre terugvoer skuifregister (LFSR) van 'n stroomsyfer benut. Dit lei tot 'n sogenaamde verdeeel-en-heers aanval, wat die geheime begintoestande van die interne LFSRs binne die stroomsyfer kan herwin.

Die binêre afgeleide en die Lempel-Ziv aanvalle vind die geheime sleutel, waarme die LFSR's ge-inisialiseer word, deur middle van 'n volledige sleutel-soektog. Die Lempel-Ziv sekwensie-kompleksiteit en 'n nuwe kompleksiteits-maatstaf vir die binêre afgeleide word gebruik om die korrekte oplossing te identifiseer en die geheime sleutel te vind. Beide aanvalle is ideaal vir implimentering op paralelle verwerkers. Eksperimentele resultate toon dat die Lempel-Ziv korrelasie aanval goeie resultate lewer vir 'n korrelasie van $p = 0.482$ en benodig ongeveer 62000 syferteks bisse

hiervoor. Die binêre afgeleide aanval is suksesvol vir korrelasie vlakke van p = 0.47 en benodig ongeveer 24500 syferteks bisse.

Die vinnige korrelasie-aanval, gebaseer op die Viterbi algoritme, maak gebruik van die teorie van konvolusiekodes. 'n Lae-tempo konvolusie kode word gevind, op grond van die pn-sekwensie wat deur die LFSR genereer is. Hierdie konvolusie kode kan dan met behulp van die Viterbi algoritme gedekodeer word. Die algoritme benodig twee aparte stappe: In die eerste stap moet bruikbare pariteit vergelykings gevind word, gebasseer op die terugvoertappe van die LFSR. Hierdie stap hoef slags eenkeer uitgevoer te word tydens die aanval op 'n sisteem. In die tweede stap word die pariteitsvergelykings in 'n Viterbi dekodeer algoritme gebruik om die pn-sekwensie te herwin en sodoende word die geheime begintoestand van die LFSR gevind. Simulasie resultate vir 'n 19-bis LFSR toon dat hierdie aanval die geheime sleutel kan herwin vir 'n korrelasie van p = 0.485, waarvoor slegs 153,448 syferteks bisse benodig word.

Al drie aanvalle wat in hierdie verhandeling ondersoek word, is in staat om LFSRs met 'n lengte van ongeveer 40 bisse aan te val. Hierdie aanvalle kan egter uitgebrei word na langer LFSRs deur van die desimasie aanval gebruik te maak. Die desimasie aanval wat hier ondersoek word, is in staat om die lengte van 'n LFSR te desimeer en kan gekombineer word met enigeeen van drie bo-genoemde korrelase aanvalle om LFSRs van heelwat langer as 40 bisse aan te val.

I wish to express my special thanks to:

My promoter, Prof. W. T. Penzhorn, for all his help during the researching, writing and editing of this dissertation.

Prof. G. J. Kühn, for all his help and explanations on the theory and background of the fast-correlation attack.

My family and friends for their support.

My employers during this time, Mecalc (Pty) Ltd and Azisa (Pty) Ltd, for the generous amounts of study leave and flexible working hours.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 9

1.1 PROBLEM STATEMENT 9
1.2 OBJECTIVE....10
1.3 CONTRIBUTION 11
1.3.1 CORRELATION ATTACKS....11
1.3.2 FAST CORRELATION ATTACK....11
1.3.3 DECIMATION ATTACK 11
1.4 OUTLINE....11

CHAPTER 2 BACKGROUND ON STREAM CIPHERS....13

2.1 INTRODUCING THE STREAM CIPHER 13
2.2 PRACTICAL RUNNING KEY GENERATORS....17
2.2.1 THE LINEAR FEEDBACK SHIFT REGISTER 17
2.2.2 THE COMBINING FUNCTION FOR THE RUNNING KEY GENERATOR....22
2.3 REVIEW OF THE STATISTICAL MODEL 25

CHAPTER 3 CORRELATION ATTACKS....28

3.1 INTRODUCTION....28
3.2 LEMPEL-ZIV COMPLEXITY OF A BINARY SEQUENCE....29
3.2.1 EXAMPLE:....29
3.3 BINARY DERIVATIVE WITH RUNS TEST....33
3.3.1 BINARY DERIVATIVE OF SEQUENCE 33
3.3.2 RUNS IN A BINARY SEQUENCE....34
3.3.2.1 EXAMPLE....35
3.3.3 GOODNESS-OF-FIT RUN TEST 36
3.3.3.1 ALGORITHM D: χ^2 GOODNESS-OF-FIT RUN TEST 36
3.4 EXPERIMENTAL RESULTS....38
3.4.1 LEMPEL-ZIV ATTACK....38
3.4.2 BINARY DERIVATIVE AND RUNS ATTACK 40
3.5 DISCUSSION....42

CHAPTER 4 FAST CORRELATION ATTACK....43

4.1 INTRODUCTION....43
4.2 REVIEW OF CODING THEORY....44
4.2.1 CONVOLUTIONAL CODES....44
4.2.1.1 POLYNOMIAL DESCRIPTION OF CONVOLUTIONAL CODES 46
4.2.1.2 MATRIX DESCRIPTION OF CONVOLUTIONAL CODES....48
4.2.2 CONVERTING A LFSR TO A BLOCK CODE....50
4.2.2.1 EXAMPLE OF CONVERTING A LFSR TO A BLOCK CODE. 51
4.2.3 FINDING PARITY EQUATIONS WITHIN A BLOCK CODE....53
4.2.3.1 EXAMPLE FOR FINDING PARITY EQUATIONS IN A BLOCK CODE....55
4.2.3.2 VERIFYING A PARITY EQUATION....58
4.2.3.3 THE EXPECTED NUMBER OF PARITY EQUATIONS WITHIN A BLOCK CODE 59
4.2.4 CREATING A CONVOLUTIONAL ENCODER USING PARITY EQUATIONS....61
4.2.4.1 EXAMPLE FOR USING PARITY EQUATIONS TO CREATE A CONVOLUTIONAL ENCODER....63

4.2.5 THE VITERBI DECODING ALGORITHM....63
4.2.5.1 THE TRELLIS DIAGRAM....63
4.2.5.2 THE VITERBI ALGORITHM....67
4.2.5.3 CALCULATING PATH METRICS....68
4.2.5.4 EXAMPLE....70
4.2.5.5 GENERATING THE RECEIVED STREAM....74
4.2.5.6 EXAMPLE FOR GENERATING THE RECEIVED STREAM....75
4.2.5.7 APPLYING THE VITERBI ALGORITHM FOR FAST CORRELATION ATTACKS....75
4.3 INTRODUCING THE ALGORITHM BASED ON A SMALL EXAMPLE....76
4.3.1 OBTAINING A CIPHERTEXT STREAM FOR SIMULATION PURPOSES....76
4.3.2 FIND PARITY EQUATIONS AND GENERATE CONVOLUTIONAL ENCODERS....76
4.3.3 CREATING THE RECEIVED SEQUENCE....79
4.3.4 USING THE VITERBI ALGORITHM FOR A FAST CORRELATION ATTACK....80
4.4 SIMULATION RESULTS AND DISCUSSION....89
4.4.1 SUMMARY OF TOPICS TO BE INVESTIGATED USING SIMULATIONS....89
4.4.2 APPROACH....89
4.4.3 RESULTS....90
4.4.3.1 RESULTS FOR SYSTEMS WITH BSC BELOW $p = 0.47$....91
4.4.3.2 RESULTS FOR SYSTEMS WITH BSC ABOVE $p = 0.47$....97
4.4.4 DISCUSSION....101
4.5 DEVIATIONS FROM METHOD DESCRIBED BY JOHANSSON AND JÖNSSON....104
CHAPTER 5 DECIMATION ATTACK....105
5.1 INTRODUCTION....105
5.2 DECIMATION OF LFSR SEQUENCES....106
5.2.1 EXAMPLE OF FINDING A USEFUL DECIMATION FACTOR D....107
5.2.2 DETERMINING THE FEEDBACK POLYNOMIAL OF THE SIMULATED LFSR....110
5.2.3 THEORETICAL DISCUSSION OF DECIMATION METHOD....111
5.2.3.1 EXAMPLE....111
5.2.4 RESULTS FROM INVESTIGATION....112
CHAPTER 6 CONCLUSION....114
6.1 CORRELATION ATTACKS....115
6.2 FAST-CORRELATION ATTACKS....116
6.3 DECIMATION ATTACK....118
6.4 FUTURE WORK ON FAST CORRELATION ATTACK....119
REFERENCES....120
APPENDIX....122

CHAPTER 1 INTRODUCTION

1.1 Problem Statement

Many practical stream cipher systems are based on binary linear feedback shift registers (LFSRs). A keystream is generated by combining the output of a number of LFSRs using a non-linear combining function *f* as shown in Figure 1.1 below.

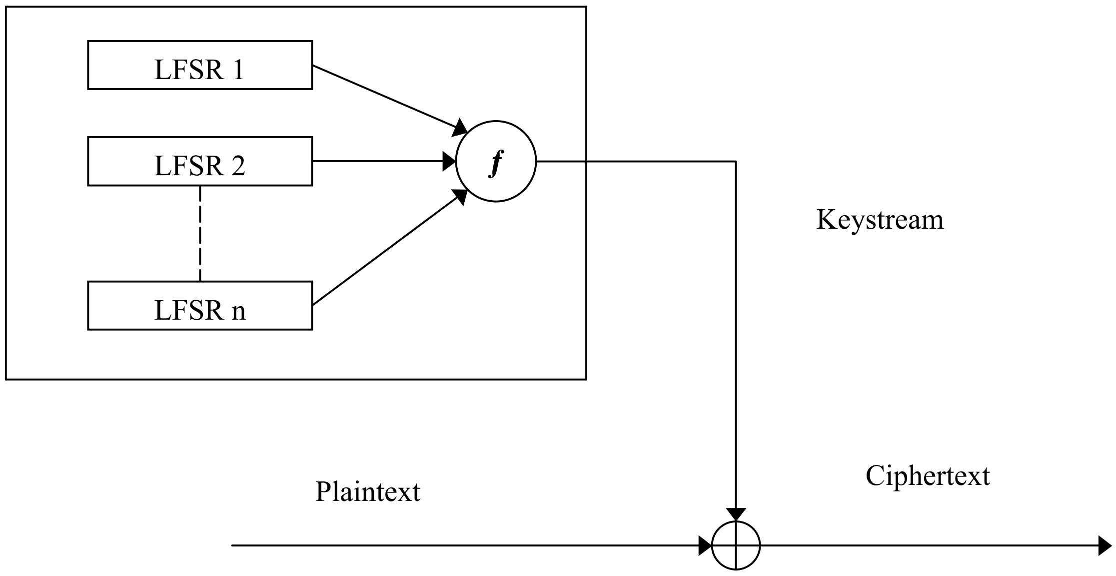

Figure 1.1 Stream cipher based on a nonlinear combining generator

In a stream cipher system the plaintext is encrypted by modulo 2 addition with the keystream, resulting in a ciphertext stream of the same length as the plaintext. The secret key for the stream cipher is used to initialize each of the component LFSRs, and has to be in the possession of both the sender and the receiver. In a brute force attack on such a stream cipher system an attacker would need to test all the possible states of the combined LFSRs, which is computationally infeasible in any contemporary system.

In practical stream cipher systems it is often found that a correlation occurs between the ciphertext and the output of an individual LFSR within the key generator. By exploiting this correlation it is possible to formulate a so-called *divide-and-conquer attack*, thereby attacking the individual LFSRs separately. Such divide-and-conquer correlation attacks radically reduce the effort of finding the secret key, since the initial condition of each LFSR may be reconstructed independently.

Let p denote the amount of correlation occurring between the ciphertext and an individual LFSR within the key generator. For ideal cryptographic applications we would expect that $p = 0.5$. However, in practical systems it is often found that $p < 0.5$, due to correlation weaknesses in the stream cipher. The magnitude of the correlation p has important consequences for divide-and-conquer attacks on stream ciphers. As will be demonstrated in this dissertation, the complexity of correlation attacks generally increases exponentially when the value of p is close to 0.5.

1.2 Objective

This dissertation investigates four different *ciphertext only* correlation attacks on LFSR-based stream cipher systems.

Firstly, two new *correlation attacks* are introduced which target a single LFSR within the key generator:

- The *binary derivative attack*.
- The *Lempel-Ziv attack*.

In these attacks the Lempel-Ziv sequence complexity measure and the Binary Derivative being are used to discriminate between random-looking and systematic binary streams.

Secondly, a *fast-correlation attack,* utilizing the *Viterbi algorithm* is introduced. The attack is quite complex, and its description together with the simulation results, forms the largest part of this dissertation. The attack models he targeted LFSR output as a wireless transmission that was corrupted by noise in a binary symmetric channel. The algorithm is used to reconstructs the LFSR's initial condition from the ciphertext by means of a Viterbi decoder, which is derived using parity equations that are embedded within the structure of any LFSR.

Thirdly, a *decimation attack*, based on an idea proposed by Filiol [1] is investigated. This attack reduces (decimates) the key-space of a targeted LFSR. The attack can be applied in combination with any of the above-mentioned attacks.

1.3 Contribution

1.3.1 Correlation Attacks

Correlation attacks were first introduced by Siegenthaler [2] based on the correlation function. The attacks in this dissertation extend his work, and are able to succeed for values of p close to 0.5, yet with much lower complexity. The Lempel-Ziv correlation attack is successful for correlation levels as low as $p = 0.482$. The binary derivative correlation attack was able to exploit correlation levels of $p = 0.47$. Simulation results show an exponential reduction of the number of ciphertext bits that are needed when the attack applies more derivatives. Simulation results provide information on the relationship between correlation level, number of derivatives, and the amount of ciphertext required for a successful attack.

1.3.2 Fast Correlation Attack

The fast-correlation attack using Viterbi algorithm discussed in this dissertation gives a substantial improvement over previous results, and is successful for correlation levels of only $p = 0.485$. This is in contrast to results obtained by Johansson and Jönsson [3] who required a much greater correlation level of at least at least $p = 0.42$. Numerous simulations in the dissertation give a detailed relationship between the correlation levels, the number of parity equations, the number of required ciphertext bits, the size of the targeted LFSR, as well as the size of the convolutional encoder. It was found that the number of parity equations is the primary factor that determines the likelihood of success for a certain correlation level. These results make it possible to predict beforehand whether an attack is likely to succeed, since this leaves only two more parameters that can be varied. These are the size of the convolutional encoder, and the number of ciphertext bits.

1.3.3 Decimation Attack

In the dissertation a list of all the useful decimation factors for LFSRs bigger than 18 bits and smaller than 64 bits is presented. In many cases it was found that the decimation attack is ineffective because of the unrealistically large number of ciphertext bits required for success attacks.

1.4 Outline

Chapter 2 provides a general introduction to stream ciphers, including a historical overview and the general architecture of such a system. A detailed mathematical model is introduced which is used throughout the remainder of the dissertation.

Chapter 3 investigates two correlation attacks, the Lempel-Ziv attack and the binary derivative attack. A model for the attack is introduced, together with a detailed description of each attack. Simulation results are given for both attacks, which investigate the conditions under which the attacks are likely to succeed, followed by a discussion on the impact of these results.

Chapter 4 investigates a fast correlation attack based on the Viterbi algorithm. A overview of the relevant mathematical background is presented, including a detailed description of the Viterbi algorithm. All steps in the process of the attack are accompanied by a theoretical explanation, followed by a practical example in the same section. These examples give a complete example for performing a fast correlation attack using a small LFSR. Simulation results are presented and discussed, followed by a number of general conclusions.

Chapter 5 investigates the decimation attack. Relevant mathematical theory is reviewed together with examples of finding practical decimation factors. Methods are discussed for applying the decimation attack to the previously introduced correlation attacks, and fast correlation attacks, as well as performing a theoretical mathematical analysis of the feasibility of the attack.

Chapter 6 gives a conclusions of all the methods investigated. This chapter compares and contrasts the various attacks, giving a global discussion of the results obtained and the practical implication thereof.

CHAPTER 2 BACKGROUND ON STREAM CIPHERS

2.1 Introducing the Stream Cipher

Keeping information secret and confidential is an age-old practice. Cipher systems have been used and evolved from the times of the Romans. This evolution has been fueled by the battle between the cryptographer and the cryptanalyst, i.e. the people designing methods to keep information private, and those trying to break these methods. Throughout history there have been times when the cryptographers were holding the upper hand, their ciphers being believed to be unbreakable, and then there were times when no cipher was considered to be safe or unbreakable and the cryptanalyst were in ascendancy.

Two main methods can be identified in a cryptanalyst's armory. The first method involves the guessing of the key by working through every possible combination of the key space and checking the result to see if the guess proved to be correct. The larger the key space, the more difficult this becomes. If the key-space is small enough that an exhaustive search is feasible, the cipher is too weak and can be considered broken. It is therefore important to ensure the key space is large. The second, and by far preferable, method involves identifying of a weakness in the cipher that will save the cryptanalyst the trouble of trying every possible key. Using this method the key can be reconstructed using statistical information embedded in the ciphertext. A simple example of this is the Caesar Shift Cipher.

The Caesar Shift Cipher, used by the Romans, in generalized terms, is a substitution cipher where each letter is substituted with another letter. The key in this case is the map, which tells a person which letter is transformed to which, e.g. every 'a' is substituted with an 'x'. The key space for this example is huge: $26!\text{-}2^{26}+1=403291461126605635516891137$ which even today at a key space of around 2^{88} would be close to impossible to break using automated methods. However, because of the statistical nature of the language this system does not hide the statistical repetition and grouping of letters in the ciphertext, making it easy to break. This weakness in the cipher provides a back door by which one can retrieve the key without trying each possible one.

Until the Second World War most ciphers were based on the substitution of characters, the so-called substitution ciphers, some of which were extremely advanced, e.g. the German Enigma and the Japanese Purple[1] cipher systems. With the development of the modern information age however these systems have changed to ones, which encipher digitally encoded data of any form and are thus not limited to enciphering text-based characters. Modern cipher systems can be loosely grouped into two categories, so-called *stream ciphers* and *block ciphers*.

Block ciphers work on the basis of transforming fixed blocks of data to blocks of ciphertext of equal length (typically 64 bits in size) according to a key as shown in Figure 2.1 below which illustrates the typical functioning of this type of cipher. Examples of block ciphers include DES, Triple-Des, IDEA, Blowfish and RC-5 [4].

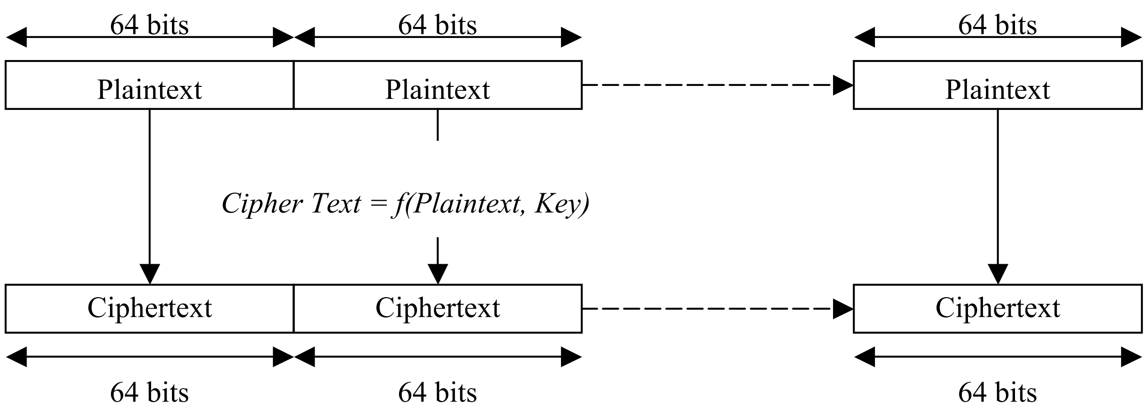

Figure 2.1 Diagram of encryption using a typical block cipher

Although the vast majority of network-based conventional cryptographic applications make use of block ciphers, stream ciphers are also widely used. For example, the A5/1 stream cipher [5], used for encryption in GSM, and RC 4 as well as many military communication systems. As far more effort has gone into the analyzing of block ciphers, the field of stream ciphers presents a big opportunity for further investigation and is the focus of this dissertation.

[1] Both these ciphers made use of electro-mechanical devices for substituting characters based on a session key and are famous for being broken by the Allies [4].

In a stream cipher the plaintext message m to be enciphered, is broken into successive characters $m_1, m_2, \ldots$. Each plaintext character m_j is enciphered by adding a keystream character k_j resulting in a ciphertext character z_j. This type of cipher is also referred to as a Vernam cipher having been introduced by Gilbert Vernam an AT&T engineer in 1918 [4]. In this dissertation only the binary form of the Vernam cipher is considered where all additions are bitwise modulo 2 additions, equivalent to an exclusive-or (XOR) shown in equation (2.1).

$$z_j = m_j + k_j \qquad j = 0,1,2\ldots \tag{2.1}$$

The basic function of the Vernam cipher, illustrated in Figure 2.2 below, is to eliminate any statistical relationship between the plaintext and the ciphertext. This is done with the addition (XOR) of a random keystream with the plaintext. The device used to generate the random keystream, where each bit is equally likely to be 0 or 1 independent of the preceding bits, is called a binary symmetric source (BSS).

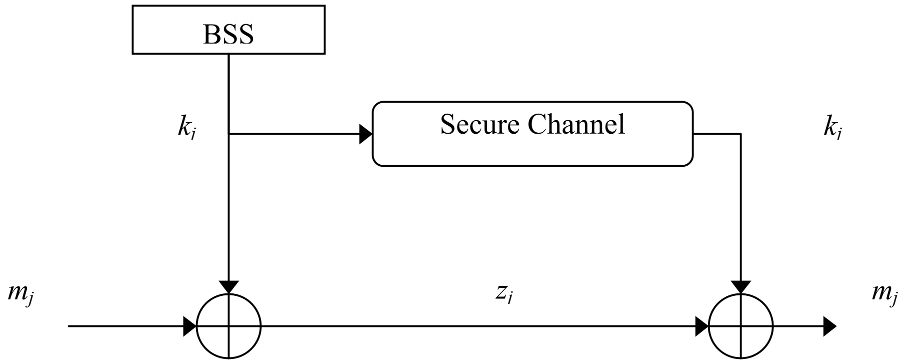

Figure 2.2 Vernam stream cipher model

A special form of the Vernam cipher proposed by Joseph Mauborgne [4] p 41 involves the use of a random keystream that is the same length as the message, without any repetitions. This scheme is known as a one-time-pad and is unbreakable. However, this method is impractical due to the fact that both the sender and the receiver have to be in possession of the same key, which is huge if the data to be encrypted is of any significant size. The key may also never be used again, otherwise there is repetition and the ciphertext is no longer unbreakable.

Using a long random keystream, one can however still attain a very secure encryption system. The key to the one-time-pad strength is the long and completely random keystream. If one can produce a random sequence by seeding a generator with a shorter value, which always produces the same sequence, one only has to communicate the short value used for seeding the generator to the receiver. This generator is referred to as a Running Key Generator (RKG) and to the seeding value as the key (K) which generates the random-looking keystream sequence $k = (k_j)$ as illustrated in Figure 2.3 below.

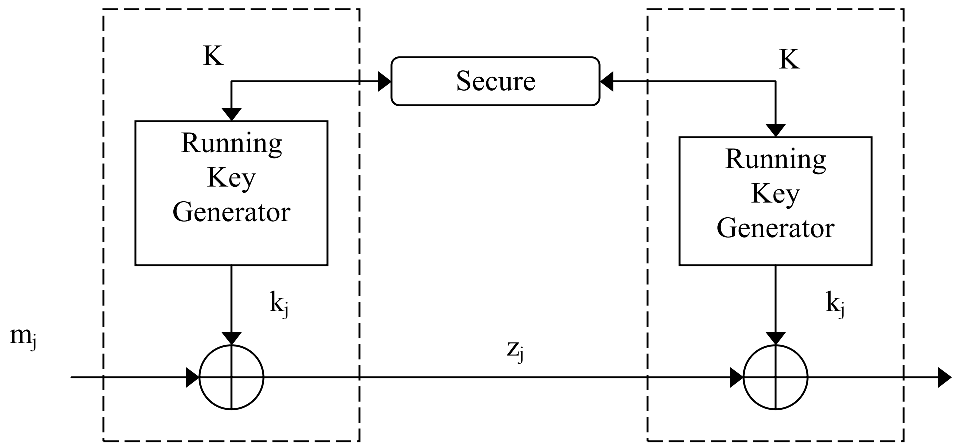

Figure 2.3 Additive stream cipher model

The ciphertext is now the bit-by-bit modulo-2 sum of the plaintext and the keystream, as shown in equation (2.2).

$$z_j = m_j \oplus k_j \qquad j = 0,1,2,\ldots \tag{2.2}$$

Fortunately for the cryptanalysist, the keystream (k_j) is not truly random but deterministic, being determined by the secret key K and the algorithm of the running key generator. Unlike the key for the Vernam cipher, the generator can only generate as many different keystreams as there are key input values. Once the key K is known, the entire keystream sequence can be reconstructed which can be exploited by the cryptanalysist. The main aim of an attacker would thus be to determine K as this allows the reconstruction of the keystream, and hence the secret message. As long as the cipher system is designed to ensure that it is practically impossible to determine K, the system is safe.

2.2 Practical Running Key Generators

2.2.1 The Linear Feedback Shift Register

The running key generator needs to be designed to output a random keystream, which cannot easily be distinguished from a truly random sequence. To make the implementation practical the generator must be able to produce the keystream rapidly without being too complex. One well-known circuit that efficiently produces a random looking sequence is the linear feedback shift register, referred to as a LFSR from now on. The design of LFSRs is based on finite field theory, developed by the French mathematic Évariste Galois apparently shortly before being killed in a dual [6]. In digital circuits, which use binary arithmetic, the operations of LFSRs correspond to operations in a finite field, or Galois Field, with 2^l elements usually denoted as $GF(2^l)$.

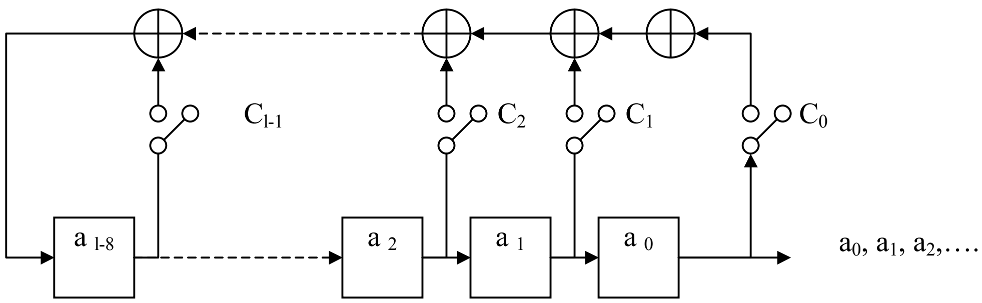

Figure 2.4 Structure of a linear feedback shift register of size l

Figure 2.4 displays the structure of an l-bit LFSR. The shift register's serial input is fed by the modulo 2 addition of previous stages of the register. The connection determining whether a value is fed back or not is represented by the coefficients $c_0, c_1, c_2, \ldots, c_{l-1}$ as shown in Figure 2.4. The next input bit to the LFSR is thus computed as a linear function of the current contents as given in the form of a recurrence relation in (2.3) below, where the initial contents of the shift register is given by the values $a_0, a_1, \ldots a_{l-1}$.

$$a_k = c_0 a_0 + c_1 a_1 + \ldots + c_{l-1} a_{l-1} \tag{2.3}$$

Associated with a LFSR is a characteristic polynomial (often referred to as the *generator polynomial*) $g(x)$, which is also expressed in terms of the feedback coefficients shown in (2.4):

$$g(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{l-1} x^{l-1} + x^l \tag{2.4}$$

The feedback coefficient c_l is equal to 1 by definition and c_0 is always chosen as 1, because the output sequence would otherwise just be a time-shifted version of the LFSR denoted by $x^n \cdot g(x)$. A LFSR of l stages can produce a non-repeating sequence with a maximum length of $L = 2^l - 1$. This so-called pseudo-random bit sequence is of maximum length if the feedback polynomial $g(x)$ is primitive. A primitive polynomial of degree l is an irreducible polynomial that divides $x^{2^l-1} + 1$, but not $x^d + 1$ for any d that divides $2^l - 1$.

The 5-bit LFSR in Figure 2.5 below represents an implementation of the primitive polynomial $g(x) = x^5 + x^2 + 1$ and is used as an example to illustrate the contents of the shift register for each clock cycle when started with the initial condition a_4, a_3, a_2, a_1, a_0 equal to 1,0,0,0,0. The content for the LFSR is shown in Table 2.1 for each clock cycle until the initial state is repeated.

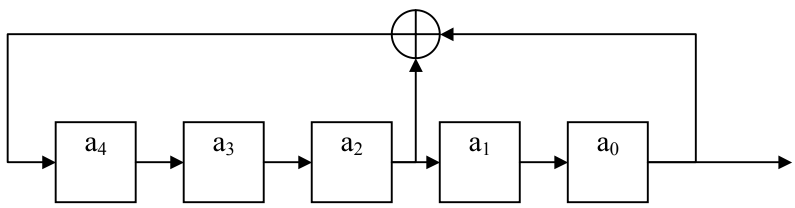

Figure 2.5 Implementation of LFSR for $g(x) = x^6 + x^4 + x^3 + x + 1$

It can be seen in Table 2.1 below that there are 31 unique states ($L = 2^5 - 1$) for the LFSR shown in Figure 2.5, where state 31 is a repeat of state 0. Each consecutive state is a right-shifted version of the previous state, with a_4 being derived by the feedback taps from a_0 and a_2, also from the previous state. An interesting observation that can be made is the fact that the output sequence can be seen in column a_0 and is also $L = 2^5 - 1$ of length before repeating. Each column (each entry in the column represents the contents of a memory cell within the shift register at time j) represents a time-shifted version of the output sequence.

Table 2.1 State of LFSR shown in Figure 2.5 for each clock cycle up to first repeat

State \ Contents	a_4	a_3	a_2	a_1	a_0
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	1	0	0	1	0
4	0	1	0	0	1
5	1	0	1	0	0
6	1	1	0	1	0
7	0	1	1	0	1
8	0	0	1	1	0
9	1	0	0	1	1
10	1	1	0	0	1
11	1	1	1	0	0
12	1	1	1	1	0
13	1	1	1	1	1
14	0	1	1	1	1
15	0	0	1	1	1
16	0	0	0	1	1
17	1	0	0	0	1
18	1	1	0	0	0
19	0	1	1	0	0
20	1	0	1	1	0
21	1	1	0	1	1
22	1	1	1	0	1
23	0	1	1	1	0
24	1	0	1	1	1
25	0	1	0	1	1
26	1	0	1	0	1
27	0	1	0	1	0
28	0	0	1	0	1
29	0	0	0	1	0
30	0	0	0	0	1
31	1	0	0	0	0

A LFSR is usually specified using the polynomial representation, which does not easily map to a hardware implementation. Consider a 8-bit LFSR, with feedback polynomial

$$g(x) = x^8 + x^6 + x^5 + x^3 + 1 \tag{2.5}$$

It is easy to convert the polynomial representation to a recurrence relation representation, which can be readily mapped to the hardware representation of a LFSR, as will be shown with equation (2.5) as an example. Setting $g(x) = 0$ one obtains:

$$0 = x^8 + x^6 + x^5 + x^3 + 1 \tag{2.6}$$

Multiplying by x^n gives:

$$0 = x^{n+8} + x^{n+6} + x^{n+5} + x^{n+3} + x^n \tag{2.7}$$

Multiply by x^{-8} then produces

$$0 = x^n + x^{n-2} + x^{n-3} + x^{n-5} + x^{n-8} \tag{2.8}$$

Replacing x^n with a_n results in:

$$0 = a_n + a_{n-2} + a_{n-3} + a_{n-5} + a_{n-8} \tag{2.9}$$

As these all are GF(2) or modulo 2 operations $a_n = -a_n$; thus

$$a_n = a_{n-8} + a_{n-5} + a_{n-3} + a_{n-2} \tag{2.10}$$

which represents the LFSR shown in Figure 2.6 below where the output sequence is denoted by $u_0, u_1, u_2, \ldots$ and n denotes any relative point in time.

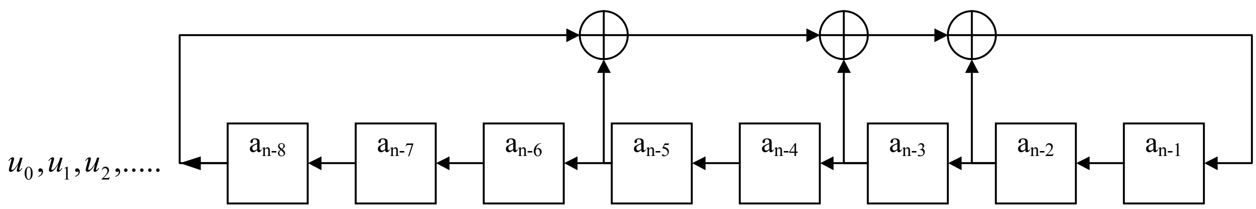

Figure 2.6 LFSR of size 8

Thus, a LFSR generates a random looking bit sequence of length $2^l - 1$ using a short seeding value. However a LFSR cannot be used on its own as a running stream generator. Consider the following case illustrated in Figure 2.7:

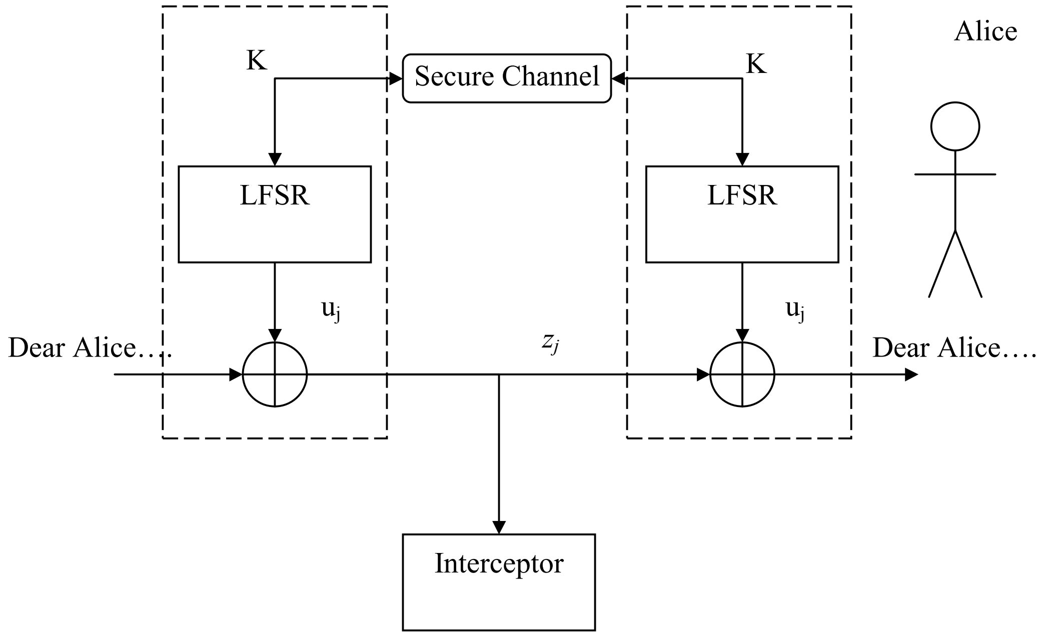

Figure 2.7 Cracking a stream cipher with a weak running key generator

Looking at the viewpoint of the interceptor it is known somebody is writing to Alice, as is the inner working of the cipher system being used and for this special case the keystream is referred to as u_j instead of k_j. By guessing that the message starts with "Dear Alice" provides 10 letters of known plaintext, as the ciphertext z_j produced by $m_j \oplus u_j$ is known. Assuming the guess is correct and the message was written using ASCII letters allows for the retrieval of $10*8=80$ bits of the key sequence as $u_j = m_j \oplus z_j$. As long as the LFSR in the system shown in Figure 2.7 above is shorter than 80 bits, the system has been cracked as one can derive the whole key sequence, forward or backward, from the section retrieved. The reason this particular example of a running key generator can be broken so easily is the lack of *confusion*, a concept that is introduced and elaborated on in the following section.

2.2.2 The Combining Function for the Running Key Generator

The lesson learned from the hypothetical attack described at the end of the previous section is the fact that if one wants to make use of the speed and simplicity of implementation of a LFSR in the construction of the running key generator, measures must be taken to prevent an attacker from retrieving the initial state of the LFSR.

The terms *diffusion* and *confusion* were introduced by Shannon [4], p60 and are fundamental to any practical cryptographic system. In *diffusion* it is attempted to make each bit in the key influence many plaintext bits in order to hide the statistical structure of the plaintext in the ciphertext. *Confusion* attempts to make the statistical relationship between the ciphertext and the key as complex as possible. When using LFSRs in a running key generator, the criteria set by *diffusion* is easily met as changing one bit in the seeding value of the LFSR (which forms part of the key) changes the output sequence of the LFSR, thus also the keystream and as a result the ciphertext.

Several methods are used to introduce *confusion* in stream ciphers for hiding the individual LFSR output bits in order to prevent the reconstruction of the key. The most common methods [7] are *nonlinear filter generators*, *clock-controlled generators*[2] and *nonlinear combining generators*, the latter being the focus of this dissertation. Nonlinear combining generators combine a fixed number of n LFSRs using a nonlinear combining function f as shown in Figure 2.8 below. The individual output streams from the various LFSRs are identified by the superscript $a^{1 \ldots n}$.

[2] A5/1 used for encryption in GSM networks makes use of clock-controlled generators.

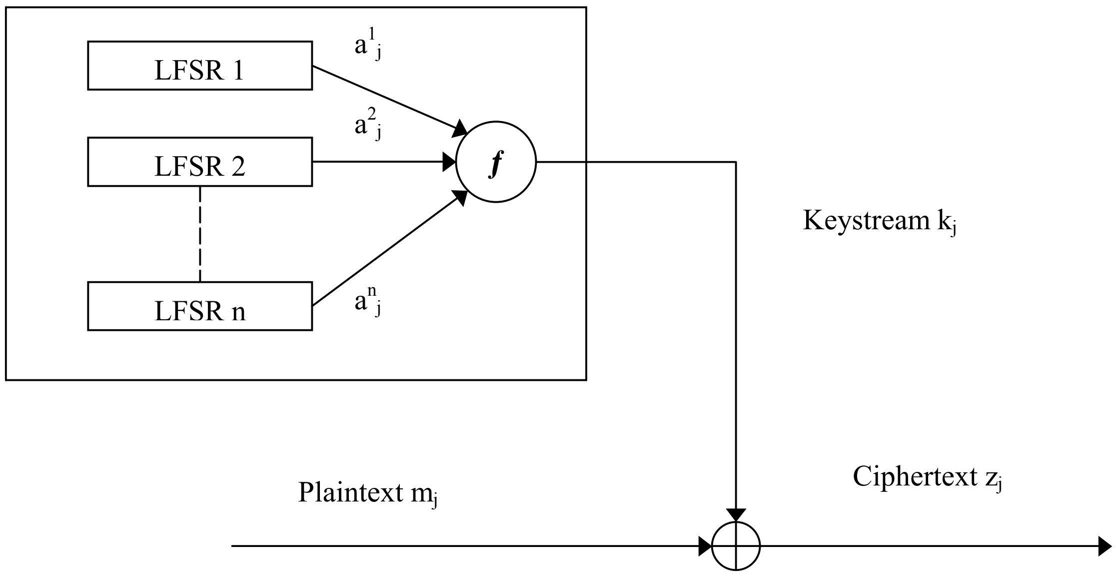

Figure 2.8 Stream cipher based on a nonlinear combining generator

An example of a simple combining function is the Geffe generator, shown in Figure 2.9 below [7]. The key of this generator is the initial conditions of the three component LFSRs.

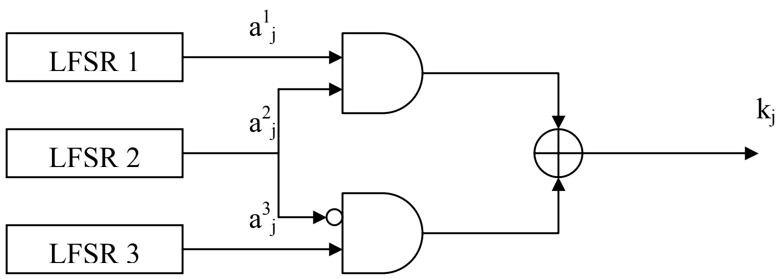

Figure 2.9 The Geffe key generator

To investigate the *confusion* in the keystream introduced by the Geffe generator the truth table of the combining function, shown in Table 2.2 below, is examined.

Table 2.2 Truth table for Geffe combining function

a_j^1	a_j^2	a_j^3	k_j
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Looking at the correlation between the individual LFSR outputs a^i it can be seen that $P(a_j^1 = k_j) = P(a_j^2 = k_j) = P(a_j^3 = k_j) = \frac{6}{8} = \frac{3}{4}$. Thus 75% of the time a bit in the keystream is equal to the contents of a specific component LFSR. Because of this, it is no longer possible to directly deduct the initial condition of a component LFSR from the keystream, however the correlation can be exploited, as the remainder of this dissertation will endeavour to illustrate.

A brute force attack attempts to examine all the possible states of the component LFSRs. Such an attack is however the last resort for a cryptanalysist when all else fails and is unlikely to succeed. Any cipher system would be designed in such a way as to ensure that the key size is orders too large for a brute force attack to succeed. In fact, because of the rapid rate at which computers are increasing in speed, the key-size is usually huge and brute force attacks can typically only be expected to work on very old systems.

Fortunately for the cryptanalysist, Siegenthaler [2][1] has shown that by exploiting the measure of correlation that exists between the running key k and the outputs of individual LFSRs a^i, as shown in the example of the Geffe generator, it is possible to perform a divide and conquer attack on the individual LFSRs thereby reducing the effort of finding the key from $\prod_1^n 2^{l_i}$ to $\sum_1^n 2^{l_i}$. This is possible by performing a brute force attack, targeting the output of only one of the component LFSR's independently from the output of the others. This approach was shown to work for a number of combining functions, e.g. as proposed by Brüer [8], Geffe [9] and Pless [10]. Siegenthaler used the correlation function to discriminate between random-looking binary sequences, resulting from the false initial states, and non-random (deterministic) binary sequences corresponding to the correct state.

To prevent the type of attack introduced by Siegenthaler, one would ideally want to have a combining function, which provides a keystream with a correlation for $P(a^i = k) = 0.5$ that would be completely random and thus the ultimate in *confusion.* In practice however, implementations of combining functions never reach correlation levels that are completely random, and in general it is found that $P(a^i = k) \neq 0.5$. In the following section a mathematical model is introduced that can be used for exploiting this weakness.

2.3 Review of the Statistical Model

In this section a model for representing the statistical relationship between the individual LFSRs within a nonlinear combining generator and the ciphertext is introduced. The model is slightly different for each type of attack described in this text and is refined at the relevant sections. Assume that a segment of N ciphertext bits is being observed by an attacker. From the attacker's viewpoint it is desirable that the value N should be as small as possible; i.e. $N << L = 2^{l_i} - 1$. The fundamental assumption for correlation attacks of stream ciphers is that the ciphertext sequence $k = (k_j)$ is correlated with probability $q' > 0.5$ to the sequence $a = (a_j)$ generated by a particular internal LFSR, i.e.

$$P(k_j = a_j) = q' = (1 - p') > 0.5 \qquad j = \quad 0, \quad 1, \quad 2, \cdots \tag{2.11}$$

The corruption of the internal LFSR sequence (a_j) due to other LFSRs in the stream cipher may be modeled as "errors" in the sequence. In the case of binary-valued digits, the model may be simplified, by setting $k_j = a_j \oplus r_j, \quad j = 0, 1, 2, \cdots$. This is illustrated in Figure 2.10.

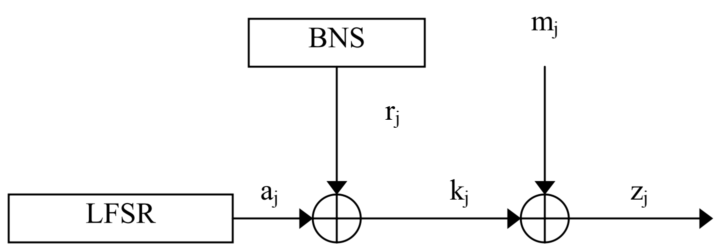

Figure 2.10 Stream cipher model

The simplifying assumption that k_j depends only on the input a_j at time j is made. The corruption of the LFSR sequence a_j due to the other LFSRs in the stream cipher and the addition of the plaintext may be modeled by the addition of "error digits" r_j.

$$P(a_j = k_j) = P(r_j = 0) = q' = 1 - p' \tag{2.12}$$

The assumption is made that the "error bits" (r_j), generated by the memoryless Binary Noise Source (BNS), are identical and independently distributed random variables. In typical applications one further finds that $P(m_j = 0) \neq 0.5$. In fact, the statistical nature of the data being encoded is usually a known factor, for instance for the transmission of voice or the transmission of English ASCII text. Because of this, the effect of the plaintext can be incorporated into the BNS allowing for the simplification of the stream cipher model as shown Figure 2.11 below, where the corruption of the LFSR sequence a_j due to the other LFSRs in the stream cipher and the addition of the plaintext has now been combined in a unified addition of "error digits" e_j.

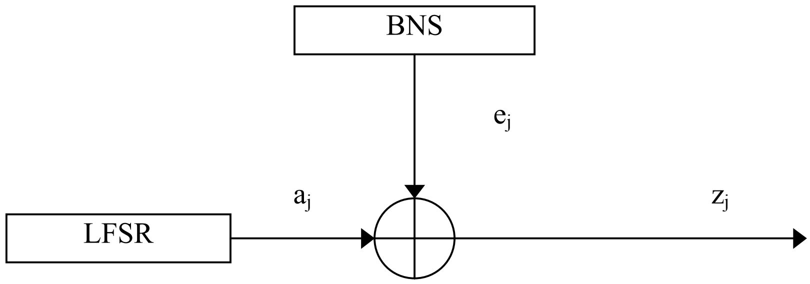

Figure 2.11 Simplified stream cipher model

The simplified model has a lower correlation level between the LFSR output a_j and the ciphertext z_j than exists between a_j and k_j making any attack more difficult. The huge advantage with the simplified model is however that one now is able to perform a *ciphertext-only* attack on the system instead of a known plaintext attack. This correlation level is shown by equation (2.13) where the probabilities q and p now combine the effect of the combining function and the effects of the statistical nature of the plaintext.

$$P(a_j = z_j) = P(e_j = 0) = q = 1 - p \qquad (2.13)$$

The challenge of the cryptanalyst is to restore the unknown LFSR sequence (a_j) from the observed ciphertext sequence (z_j), which may be viewed as a "noisy" version of (a_j) as shown with the equivalent model in Figure 2.12 below.

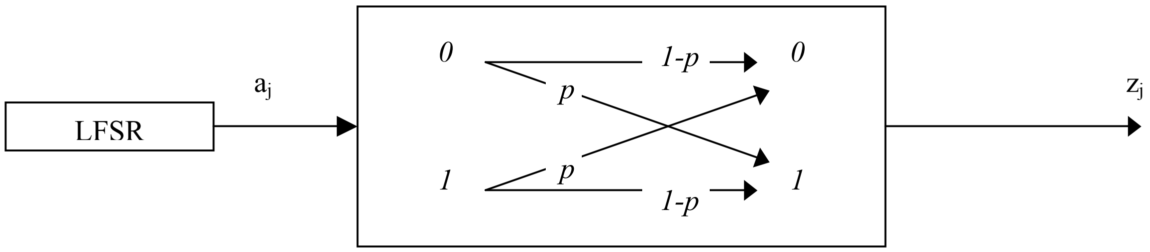

Figure 2.12 BSC equivalent model for stream cipher

The BNS sequence of $P(e_j = 1) = p$ which determines the correlation between z_j and a_j has been replaced by a Binary Symmetric Channel (BSC) with an error probability of p.

CHAPTER 3 CORRELATION ATTACKS

3.1 Introduction

The correlation attack is a divide-and-conquer attack. The goal with this attack is the finding of the initial condition of a targeted LFSR in the stream cipher model presented in section 2.3. To do this a t*est LFS*R, identical to the LFSR under attack is introduced to the simplified model of a stream cipher system presented in Figure 2.10. For the attack, the Test LFSR is stepped through all $2^l - 1$ non-zero initial states, and the output is XOR-ed with the output of the stream cipher model, as shown below.

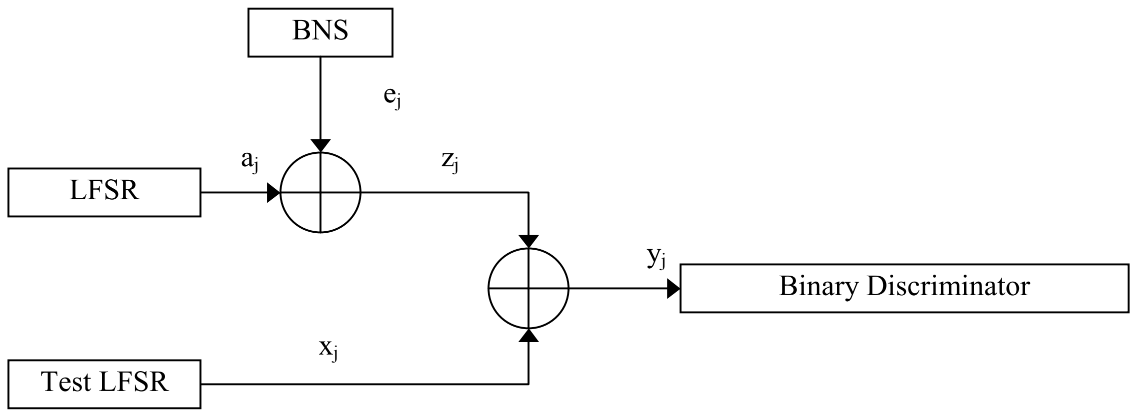

Figure 3.1 Model for the attack

The amount of correlation between the LFSR-sequence and the ciphertext can be adjusted, by changing the probability $p = P(e_j = 1)$ of the BNS emitting a 1. A high level of correlation implies that only very few 1's are injected into the LFSR output sequence by the BNS. In general, the output sequence (y_j) of the model will appear to be "random", since it is the XOR of two out of phase pn-sequences, the number of 0s and 1s in the sequence being roughly equal. However, when the Test LFSR is initialized with the *correct* initial state (identical to the initial state of the LFSR under attack), the output sequence (y_j) will be *unbalance*d, consisting mainly of long runs of 0's, interspersed with a few 1's. Two new methods are introduced for identifying this unbalanced binary sequence from all other "random"-looking sequences. Two new binary discriminators, based on the Lempel-Ziv sequence complexity, and the Binary Derivative combined with the runs test, are introduced.

For the practical application of the attacks, it is important to estimate the number of ciphertext bits that are required for the attacks to be successful. In a practical situation it is very unlikely that high levels of correlation will occur between the ciphertext and any internal LFSR. Realistic correlation levels that may be encountered in practice will lie in the range $0.52 \leq q \leq 0.60$. Such fairly low correlation levels imply that the output sequence of this model will be a very "noisy" version of the LFSR sequence (a_j) under attack.

3.2 Lempel-Ziv Complexity of a Binary Sequence

The Lempel-Ziv (L-Z) algorithm forms the basis of one the most useful and versatile universal, noiseless, data compression algorithms [11]. It is a dictionary-type parsing algorithm that parses a given sequence of digits into consecutive, non-overlapping *phrases* or *codeword*s. The number of parsed phrases, m , serves as a measure of complexity and is commonly referred to as the *Lempel-Ziv complexit*y. The L-Z parsing process may be briefly summarized as follows:

- Search through all parsed codewords for a matching word. Determine the longest possible matching word that serves as the *prefi*x.
- Extend the selected prefix by one new bit from the sequence, i.e. by a *suffi*x, and mark the resulting codeword with a comma. Continue until all the bits in the given sequence have been parsed.

The codewords in the parsed sequence are all unique. The L-Z complexity of the sequence is determined by counting the number of parsed words.

3.2.1 Example:

Given the binary sequence:

$0110010110100101101011 00.$

LZ-parsing gives:

$0,1,10,01,011,010,0101,101,0110,0.$

The sequence is parsed into m codewords, where the last codeword is incomplete. The corresponding codebook entries may be tabulated as follows:

1:	0	6:	010
2:	1	7:	010**1**
3:	10	8:	101
4:	01	9:	011**0**
5:	011	10:	0.

The suffixes are shown in bold print. Note that the prefix of each word corresponds to a previously occurring codeword. Gilbert *et al* have derived exact analytical results for the LZ-parsing of binary sequences [12]. Based on their results, it is possible to use the L-Z algorithm to discriminate between random and deterministic binary sequences.

The searching for the longest possible matching word in the list of all parsed codewords can be a time consuming task. When adding the fact that this needs to be done for each bit parsed in the input string the need arises to speed up this process. Using a hash-table it possible to determine in a single operation whether a codeword is contained in the list or not. This is approached as follows: The codeword is considered as an index into an array of Booleans. If true, the codeword is already contained in the list, if false it is not. The obvious problem with this approach is the fact that although any codeword starting with a '1' is unique however considering the following two codewords: 0001 and 001. Although these are completely different codewords both point to the same index in the hash-table. This can be easily resolved by keeping two separate hash-tables, one for code words starting with a '1' and a different one for codewords starting with '0'. To find an entry in the hash-table containing codewords starting with '0' codeword to be looked up is inverted, thus 0001 becomes 1110 and 001 becomes 110. These inverted codewords do provide unique index positions in the hash-table for codewords starting with '0'.

This approach works well with random sequences as the size of the codeword grows slowly. When working with non-random sequences a simple mind experiment can show that the memory requirement for hash-table would be to big. Consider the sequence $1,11,111,1111,11111,111111,111$. which is the same length as the one used before. The codewords for this sequence are shown below. It is clear from these results that a hash-table approach would not work as the value of the codeword grows exponentially with each bit parsed, thus quickly using up the available memory.

1: 1
2: 11
3: 111
4: 111**1**
5: 1111**1**
6: 11111**1**
7: 111.

However, as one is working with random pn-sequences and using the Lempel-Ziv attack on LFSRs, which are in the size ranges were a codeword is unlikely to exceed the memory available to the cryptanalysist, this does not present a problem.

For the special case of equi-probable binary sequences, the required *average length* $E[x_m]$ of a binary sequence that has been parsed into codewords, is given by the following recursion [12] in (3.1):

$$E[x_m] = m + \left(\frac{1}{2}\right)^{m-1} \sum_{k=0}^{m} \binom{m}{k} E[x_{k-1}] \tag{3.1}$$

with $E[x_1] = 1$ and $E[x_m] = 0$ for $m < 1$

The *second moment* $E[x_m^2]$ is recursively given by

$$E[x_m^2] = \left(\frac{1}{2}\right)^{m-1} \left(\sum_{k=0}^{m} \binom{m}{k} \left(E[x_{k-1}^2] + E[x_{k-1}] \cdot E[x_{m-k-1}] \right) \right) - m^2 + 2m \cdot E[x_m] \tag{3.2}$$

starting with $E[x_0^2] = 0$ and $E[x_1^2] = 1$. The standard deviation σ_m is obtained as the square root of the variance $E[x_m^2] - E[x_m]^2$. In Table 1 selected values are shown for the number of parsed words m and the corresponding average sequence length and standard deviation. These values can be used to discriminate between random and deterministic binary sequences.

Figure 3.2 and Figure 3.3 present graphs of $E[x_m]$ *and* σ for $0 \le m \le 10000$. The exact values can be found in Appendix A. The calculation of $E[x_m]$ and $E[x_m^2]$ present a challenge in determining as the complexity grows directly proportional to m^2. Further it can be seen that the factor $\left(\frac{1}{2}\right)^{m-1}$ becomes minute as m grows while $\binom{m}{k} = \frac{m \cdot (m-1) \cdot (m-2) \cdots (m-(k-1))}{k!}$ is huge for certain values of k as m grows. It is very easy to loose resolution of these values while calculating and special care needs to be taken to continually use the factor $\left(\frac{1}{2}\right)^{m-1}$ to scale $\binom{m}{k}$ as it is not feasible to calculate these two factors separately and only then multiply them with each other.

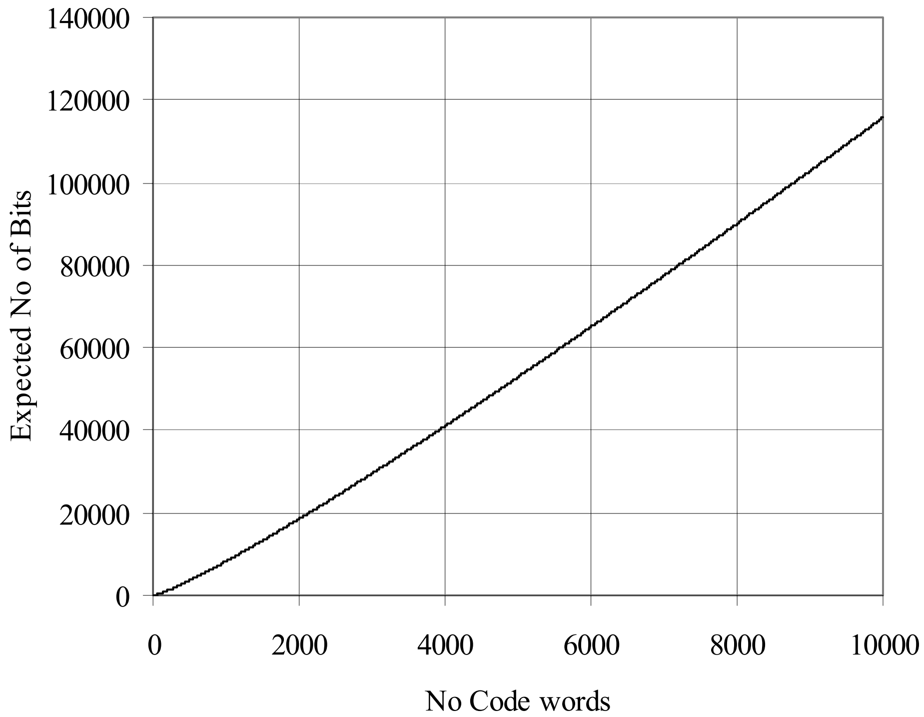

Figure 3.2 $E[x_m]$ *as a function of* m *for* $0 \le m \le 10000$

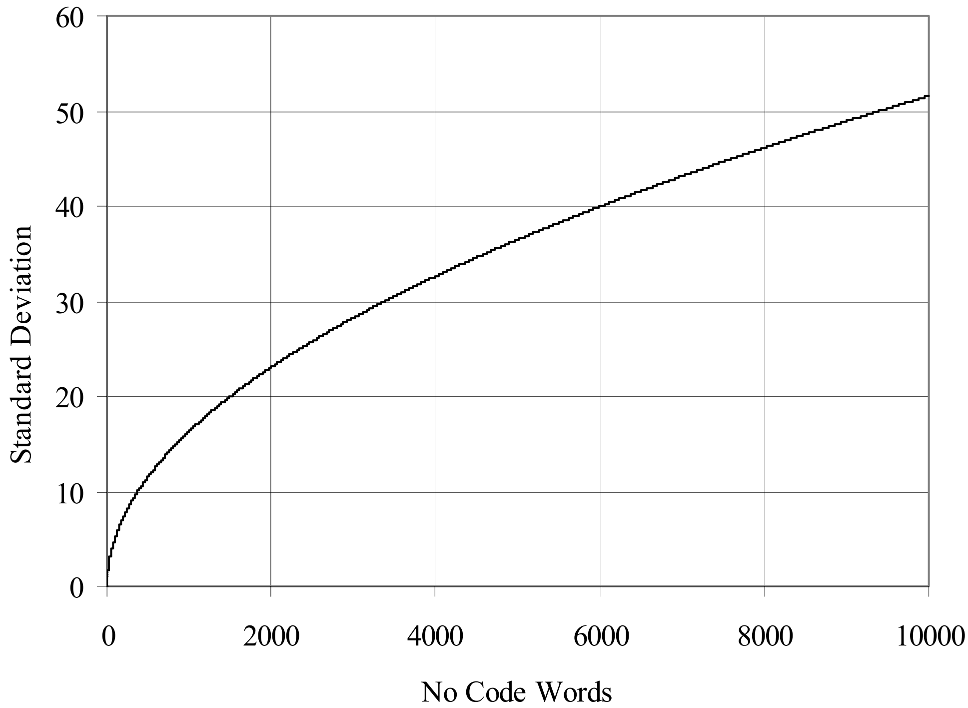

Figure 3.3 σ *as a function of* m *for* $0 \le m \le 10000$

3.3 Binary Derivative with Runs Test

3.3.1 Binary Derivative of Sequence

The binary derivative has been proposed as a test for a binary sequence, to determine if it is random or deterministic [13]. Consider the following binary sequence of length $n = 16$: 1000100011010101.

The binary derivative of the sequence is obtained by computing the XOR of each pair of adjacent bits in the sequence. The derivation process can be repeated recursively any number of times. The initial sequence, as well as the first four derivatives, is shown below. Note also that the sequence length of each derivative is one less than its preceding sequence. The index k denotes the $k-th$ derivative, with $k = 0$ for the initial sequence.

The binary derivative can be rapidly computed by creating a copy of the sequence that is shifted 1 bit to the right and then XOR-ing these two sequences. This can be done very efficiently by using, for example, 32-bit unsigned integers, without the need for bit-operations. Note also, that if the initial sequence is truly random, all subsequent derivatives will also be random.

$k=0$:	1	0	0	0	1	0	0	0	1	1	0	1	0	1	0	1
$k=1$:		1	0	0	1	1	0	0	1	0	1	1	1	1	1	1
$k=2$:			1	0	1	0	1	0	1	1	1	0	0	0	0	0
$k=3$:				1	1	1	1	1	1	0	0	1	0	0	0	0
$k=4$:					0	0	0	0	0	1	0	1	1	0	0	0

Several complexity measures based on the binary derivative have been suggested, to test the randomness properties of a binary sequence as shown in [13],[14],[15] and [16]. These measures comprise the counting of the number of 1's or 0's in a derivative sequence, and then determine the maximum and minimum values thereof. The investigations in this dissertation have shown that these measures are inadequate for the purpose of cryptanalysis considered here. Therefore, a new complexity measure is introduced, based on the distribution of runs in a binary sequence.

3.3.2 Runs in a Binary Sequence

Consider a binary sequence of $n=16$ values: 1000100011010101. A *run* is defined as a sequence of identical observations that is preceded and followed by a different observation, or no observation at all. In this example there are $r=11$ runs in the sequence.

The number of runs that occur in a sequence gives an indication of the randomness properties of the sequence. Specifically, if the sequence may be regarded as random, then the number of runs r in the sequence is approximately normal distributed, with the mean value given by equation (3.3):

$$E[r]=\frac{2n_1n_0}{n}+1 \tag{3.3}$$

Where n_1 is the number of runs in the sequence consisting of 1's, and n_0 the number of runs consisting of 0's. For the special case when the sequence is truly random, it follows that $n_0=n_1=n/2$, and the expected number of runs is given by

$$E[r]=\frac{n}{2}+1 \tag{3.4}$$

The expected number of runs of a binary sequence can be applied as a *non-parametric* test, to evaluate the randomness properties of the sequence [17],[18].

Counting the number of runs in a sequence is a time consuming exercise. To speed this process up a trade-off of processing power versus memory usage is once again used. Instead of looking at two adjacent bits, a hash table is setup for the number of runs contained in a word. The size of the word depends on the amount of memory that is available. It makes sense to use words sizes inherent in the addressing of the computer architecture being used, typically either an 8-, 16-, 32- or 64-bit words. In this implementation a word-size of 16 bits was used (unsigned short). Thus the number of runs in every number contained in a 16-bit word was calculated and entered in the corresponding index position of the array. All that remains to be done is to compare the last bit of one word and the first bit of the next word to see if the next word is also the start of a new run.

3.3.2.1 Example

Consider the sequence $11001111 \quad 11111111 \quad 11100000 \quad 00000000$. This sequence has 4 runs. When using the hash-table method one works as follows:

- runs = ArrayOfAllPossibleRunsIn16BitWord[$11001111 \quad 11111111$]

  thus: $runs = 3$

  (the entry ArrayOfAllPossibleRunsIn16BitWord[$11001111 \quad 11111111$] was calculated once before beginning the attack, as were all other possible index positions for 0 to 65535)

- Compare bit 15 of first word with bit 0 following word. If they mach one knows the last run in the first word continues in the second word. In this case they mach:

  runs += ArrayOfAllPossibleRunsIn16BitWord[$11100000 \quad 00000000$] **– 1**

  thus: $runs = 3 + 2 - 1$

Now consider the sequence $11001111 \quad 11111111 \quad 00011111 \quad 11111111$. This sequence has 5 runs. Again using the hash table:

- runs = ArrayOfAllPossibleRunsIn16BitWord[$11001111 \quad 11111111$]

  thus: $runs = 3$

- If bit 15 of first word and bit 0 following word do not match as is now the case one gets:

  runs += ArrayOfAllPossibleRunsIn16BitWord[$11001111 \quad 11111111$]

  thus: $runs = 3 + 2$

When using this approach the amount of effort for determining the number of runs in a sequence is reduced by a factor of 16 when using a word size of 16. Using hash tables of word-size 32-bits would further reduce the complexity although it has to be remembered that the effort is further reduced only by a factor of 2 while the memory used for the hash-table grows from 65536 bytes to 4294967296 bytes, hardly worth the gain.

3.3.3 Goodness-Of-Fit Run Test

In this section a new non-parametric randomness test is proposed for binary sequences, by combining the Binary Derivative with the runs test. The aim of this test is to discriminate between random and deterministic binary sequences.

Let r^k denote the number of runs in the $k-th$ binary derivative. The expected number of runs for the $k-th$ derivative is given by

$$r_e^k = \frac{n^k}{2} + 1 \tag{3.5}$$

where n^k is the sequence length of the $k-th$ derivative. Let r_0^k denote the *observed* number of runs of the $k-th$ derivative. Next the χ^2 goodness-of-fit test is applied to test the hypothesis that a given sequence is random, if the *observed* number of runs closely follows the (theoretical) *expecte*d number of runs. Thus the χ^2-value for a total of K binary derivatives is calculated as follows:

$$\chi^2 = \sum_{k=0}^{K} \frac{(r_0^k - r_e^k)^2}{r_e^k} \tag{3.6}$$

For each derivate, the difference between the observed and expected number of runs is determined, then the difference is squared and summed. The resulting χ^2-value is approximately normal distributed, with K degrees of freedom.

Based on this discussion, the χ^2 goodness-of-fit run test leads to the following algorithm.

3.3.3.1 Algorithm D: χ^2 Goodness-Of-Fit Run Test

(1) **Initialize:** Set the derivative counter $k = 0$.

(2) **Count runs:** Determine r_0^k, the observed number of runs in the $k-th$ derivative of the given sequence.

(3) **Compute χ^2:** Compute the χ^2-value for the $k-th$ derivative

$$\chi^2 = \sum_{k=0}^{K} \frac{(r_0^k - r_e^k)^2}{r_e^k}$$

(4) **Binary derivative:** Differentiate the sequence, and obtain the next derivative.

(5) **Loop:** Set $k = k+1$. Return to Step 2. Continue until a total of k derivatives have been tested.

(6) **Sum χ^2**: Sum the K χ^2 values.

(7) **Compare χ^2**: Choose a confidence level α and compare the computed χ^2 value to the theoretical limits that can be found in [19]. If the χ^2 value is less than the limit, conclude that the given binary sequence is random. Else the sequence is classed as deterministic.

3.4 Experimental Results

3.4.1 Lempel-Ziv Attack

In Figure 3.4 experimental results are shown where the Lempel-Ziv algorithm is used to test the output of the model shown in Figure 3.1. The probability $p = P(1)$ of the BNS was set to $p = 0.47$ and the Lempel-Ziv algorithm was set to parse $m = 1000$ codewords. The Test LFSR was stepped sequentially through all possible initial states. As can be seen from Figure 3.4, the correct initial state is clearly recognizable as a peak in the otherwise noisy parsed sequence. The lower horizontal line in Figure 3.4 depicts $E[x_{2470}]$ while the upper horizontal line depicts $E[x_m] + 4 \cdot \sigma_m$.

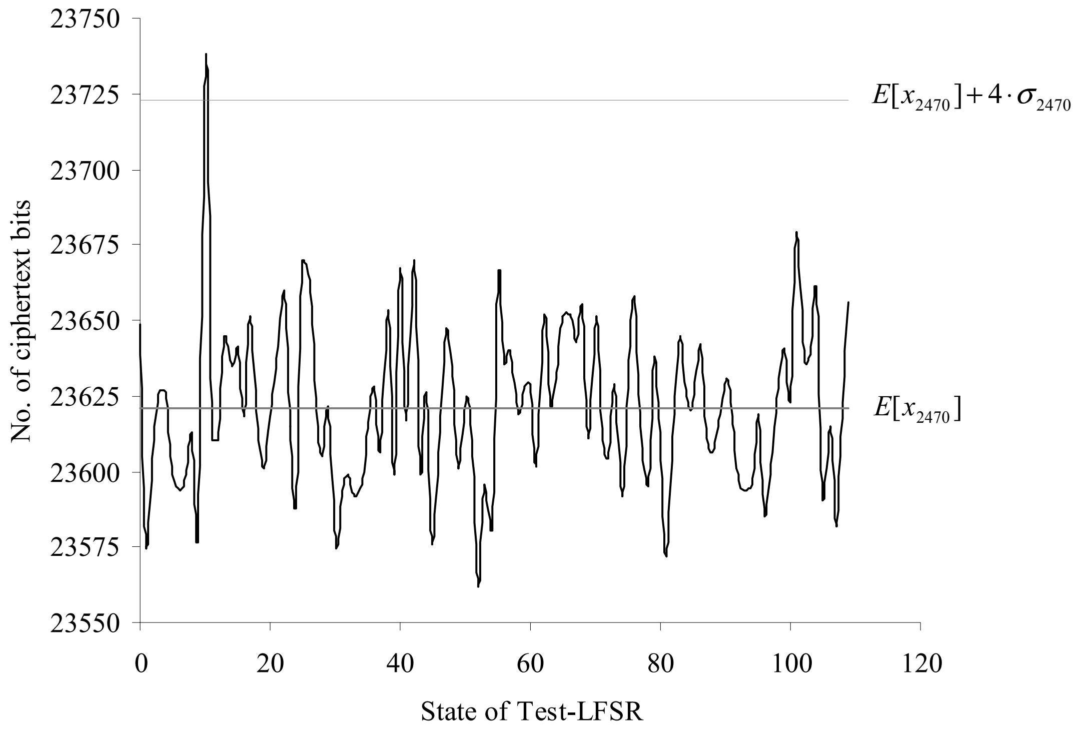

Figure 3.4 Parsing with different initial states ($m = 2470$, $p = 0.47$ *)*

Any peak exceeding the upper horizontal line can safely be considered the correct initial condition, as is the case for relative initial condition 22 in Figure 3.4 above [19]. Figure 3.5 shows the number of ciphertext bits that are needed for correlation values in the range $0.40 \le p \le 0.48$ where the peak of the correct initial condition fulfills the criteria or exceeding $E[x_m] + 4 \cdot \sigma_m$. Note that there is an exponential increase in the number of ciphertext bits, as the correlation between ciphertext and an internal LFSR decreases. The exact values are can be found in Appendix B.

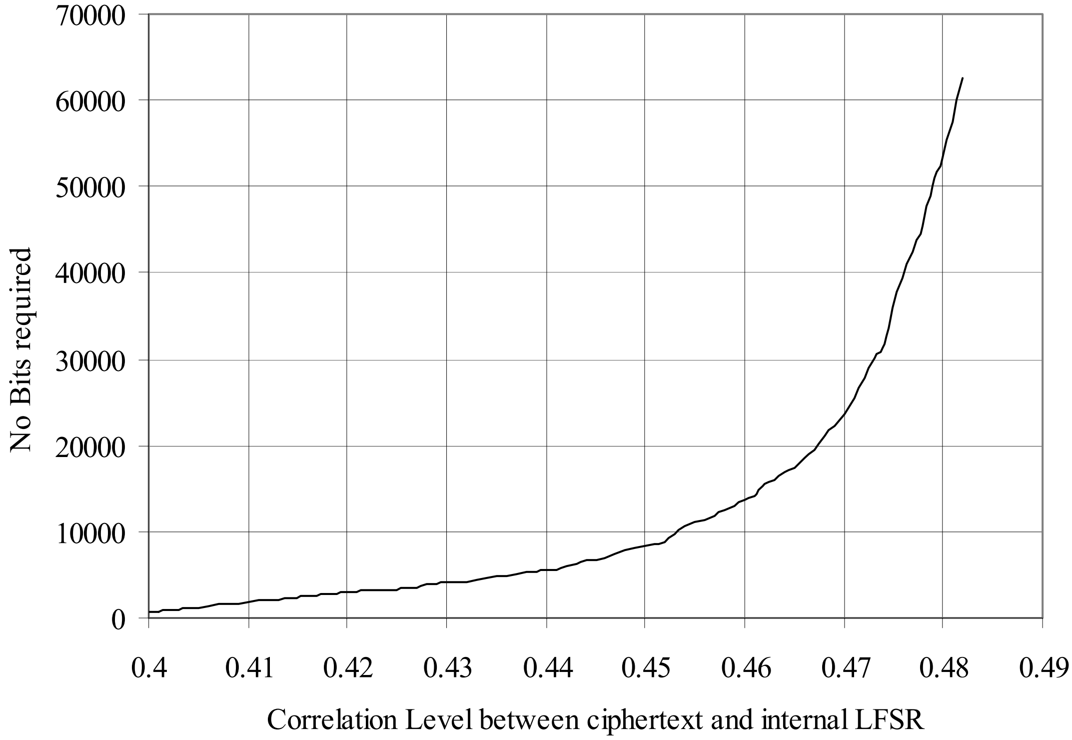

Figure 3.5 Required ciphertext bits for L-Z attack

3.4.2 Binary Derivative and Runs Attack

Figure 3.6 shows the results of the Binary Derivative, combined with the run test, with the probability $p = P(1)$ of the BNS set to $p = 0.45$. The peak, corresponding to the correct initial state of the Test LFSR, is clearly identifiable.

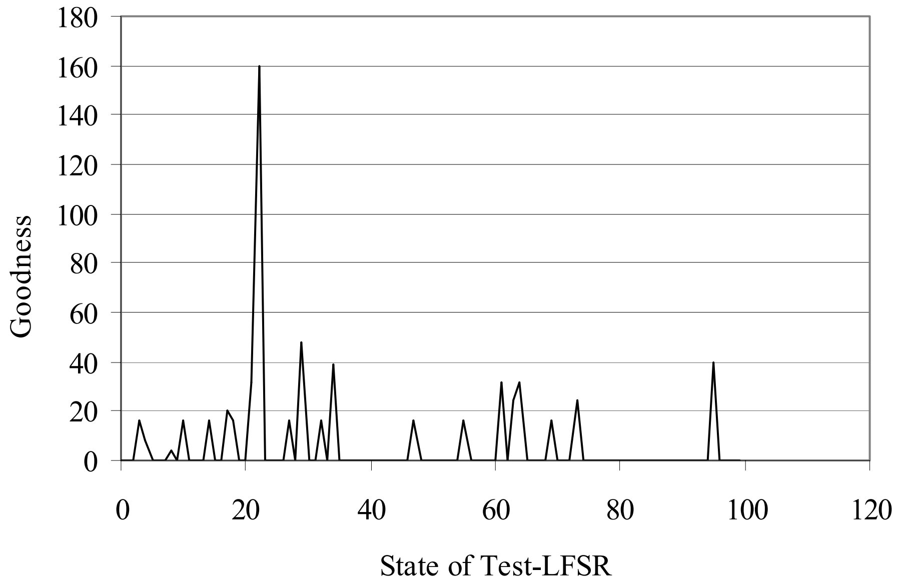

Figure 3.6 Illustration of Binary Derivative attack ($p = 0.45$, $K = 15$, $bits = 8000$)

For the practical application of this attack, two parameters need to be investigated; the number of binary derivatives K and the number of required ciphertext bits.

In Figure 3.7 experimental results for $0.4 \le p \le 0.47$ are shown for various derivatives in the range $0 \le K < 25$. The results indicate that a trade-off exists between the number of derivatives and the required number of ciphertext bits needed for the attack to succeed.

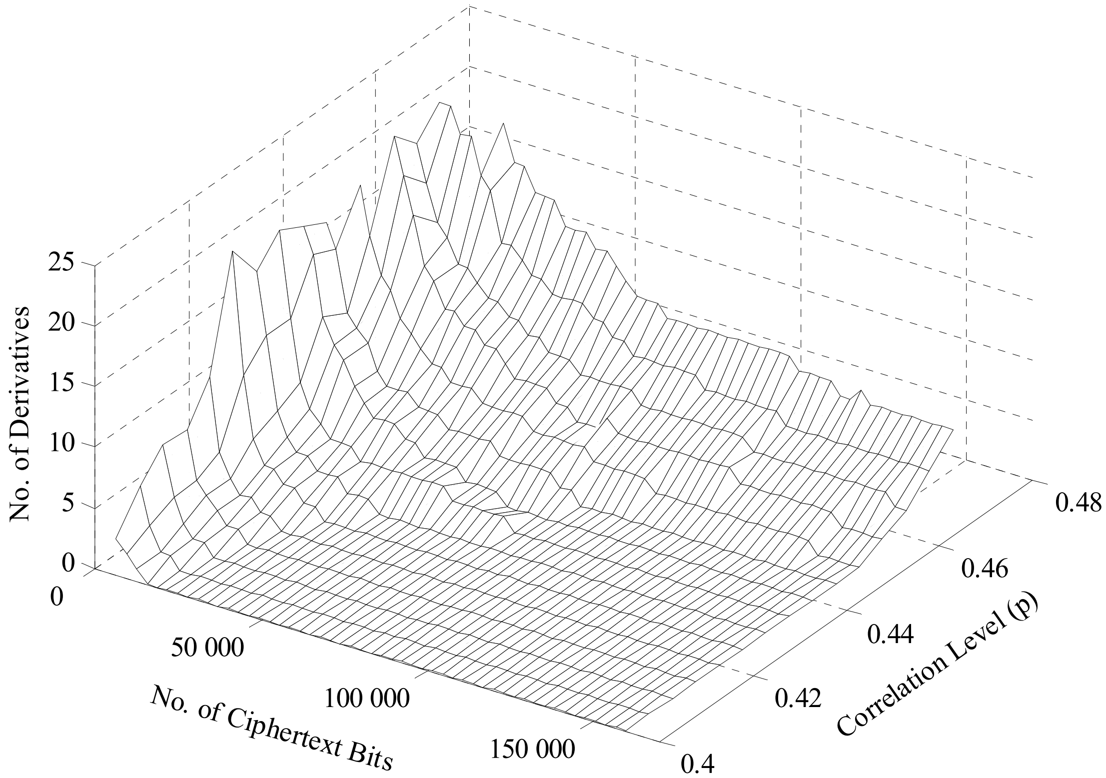

Figure 3.7 Binary Derivative attack as for $0.4 \le p \le 0.47$

The exact values of the data acquired from simulation results for Figure 3.7 are listed in Appendix C.

As the number of available ciphertext bits increases (as can be seen in Figure 3.7), there is an exponential decrease in the number of derivatives that need to be computed. This observation may be of considerable importance in a practical situation where an attacker has a limited number of ciphertext bits available. This can also be clearly observed in Figure 3.7 that the correct initial condition can be obtained for a certain p by using fewer bits but more derivatives K.

A similar result can be seen in Figure 3.7 when looking at the relation between K and p for constant amounts of available ciphertext. Higher values of p can still be broken when having the same amount of bits available by increasing K. The fewer bits are available, the bigger K needs to be. It must be noted however that this procedure cannot be continued indefinitely. Although it has been observed that the correct initial condition could be retrieved when using values of K as big as 60, this could not be reliably repeated. Experimental data would seem to indicate that using values of K in excess of 25 have little or no benefit.

3.5 Discussion

Two new correlation attacks on stream ciphers have been introduced. The first attack utilizes the Lempel-Ziv complexity measure of a binary sequence. The second attack is based on the Binary Derivative of a sequence, combined with the runs test.

Both attacks give very good results, and are able to recover the unknown initial states of a LFSR-based stream cipher, even if a very small correlation of $q = 0.52$ occurs between the observed ciphertext and an internal register of the stream cipher. Experimental results indicate that approximately 60000 ciphertext bits are required for these attacks to succeed in the case of $q = 0.52$.

The memory requirements of the Binary Derivative attack are substantially lower than the Lempel-Ziv attack. This makes the former attack suitable for stream ciphers with longer component LFSRs. Furthermore, it is possible to reduce the computational complexity of both attacks by making use of decimation techniques to reduce the total number of LFSR-states that have to tested.

CHAPTER 4 FAST CORRELATION ATTACK

4.1 Introduction

The obvious problem with the exhaustive approach (used by the correlation attacks in the previous chapter) of finding the correct initial condition of one of the LFSRs is the fact that a LFSR of size l has $2^l - 1$ non-zero initial conditions. By increasing $l>40$ it becomes virtually impossible to find the correct initial state by exhaustively searching for the correct key.

The fast correlation attack, like the correlation attacks described in the previous chapters, is a divide and conquer attack. The model (presented in section 2.3) used for fast correlation attacks was shown in Figure 2.12 and is repeated in Figure 4.1 below for convenience.

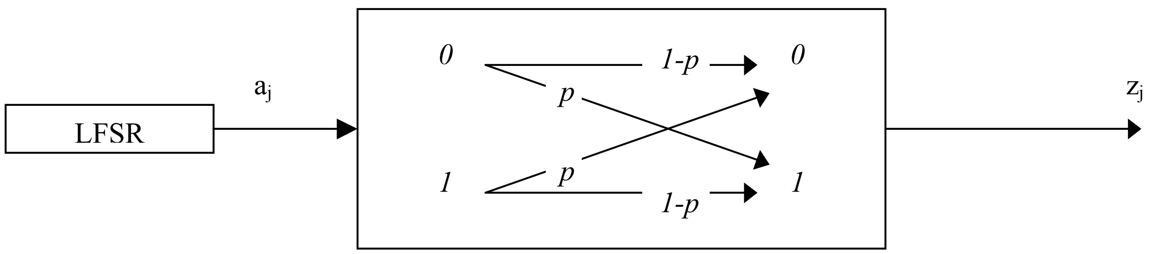

Figure 4.1 BSC equivalent model for a correlation attack

Fast correlation attacks, as described by [3], are based on the same principle used by convolutional codes for correcting errors occurring during transmission of data over a noisy channel. This approach is possible due to the fact that one can identify an embedded low-rate convolutional code in the pn-sequence generated by the LFSR. The embedded convolutional code can then be decoded with low complexity using the Viterbi algorithm. The Viterbi algorithm was chosen for this dissertation as it is one of the most well-known and widely used decoding algorithms for convolutional codes.

All algorithms for fast correlation attacks operate in two phases: In the first phase the algorithms find a set of suitable parity check equations based on the feedback taps from the LFSR, in this case from the LFSR's equivalent block code. The second phase uses these parity check equations in a fast decoding algorithm to recover the transmitted codeword and thus the initial state of the LFSR.

The following aspects are covered in this chapter:

- A review is presented of the theory required by the different elements used for this attack.
- The Viterbi algorithm is introduced using a small example.
- Simulation results are presented and discussed.

4.2 Review of Coding Theory

4.2.1 Convolutional Codes

Convolutional codes are codes where redundancy is introduced into a data stream through the use of a linear shift register [20]. Most codes for computer systems are over $GF(2)$ and $GF(2^b)$. This dissertation will only concentrate on binary codes over $GF(2)$.

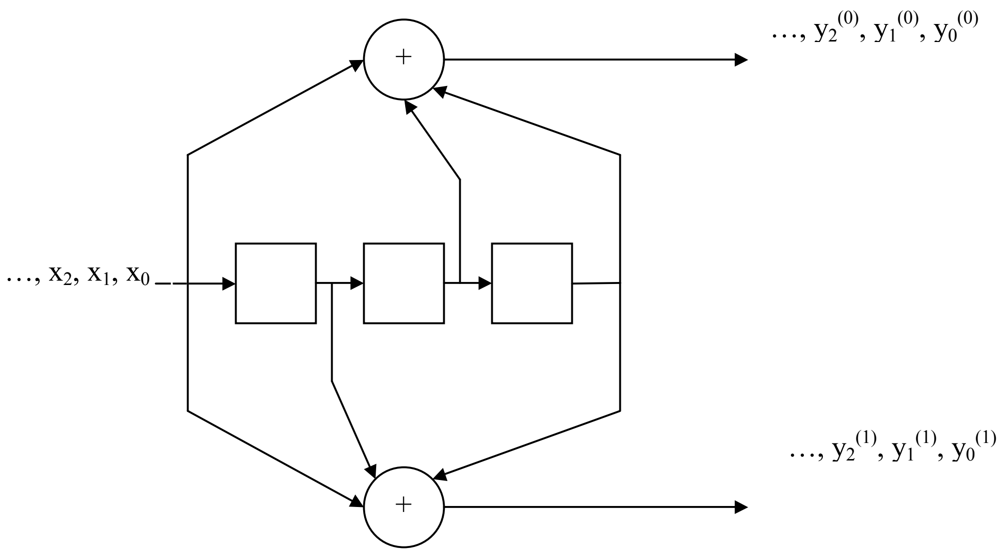

Figure 4.2 Rate 1/2 linear convolutional encoder

Figure 4.2 shows a typical rate-1/2 linear convolutional encoder. The rate of this encoder is established by the fact that the encoder outputs two bits for every input bit. In general, an encoder with *k* inputs and *n* outputs is said to have a rate of $R = \frac{n_0}{k_0}$. With each successive input to the shift register, the values of the memory elements are tapped off and added according to a fixed patter, creating a pair of output coded date streams, $y^{(0)} = (y_0^{(0)}, y_1^{(0)}, y_2^{(0)} ...)$ and $y^{(1)} = (y_0^{(1)}, y_1^{(1)}, y_2^{(1)} ...)$. These output streams can be multiplexed to create a single coded data stream $y = (y_0^{(0)}, y_0^{(1)}, y_1^{(0)}, y_1^{(1)}, y_2^{(0)}, y_2^{(1)} ...)$ where y is the convolutional code word. The infinite set of all infinitely long codewords that one obtains by exciting this encoder with every possible input sequence is called an (n_0, k_0) *tree code* [21], pp 348-350.

The constraint length *v* of a convolutional code is the maximum number of bits in a single output stream that can be affected by any input bit. Although different definitions exist, for this text the constraint length is defined by $v = mk_0$ For convolutional encoders with a single input stream the constraint length *v* will thus always be equal to the length of the shift register [21], pp 348-350.

There are several other length measures for a tree code. Let $k = (m+1) \cdot k_0$. This k is closely related to the constraint length and is called the word length of a convolutional code. The corresponding measure after encoding is called the *blocklength* n given by [21], pp 348-350:

$$n = (m+1)n_0 = k \cdot \frac{n_0}{k_0} \tag{4.1}$$

The convolutional encoder in Figure 4.2 for instance, has $k_0 = 1, n_0 = 2, v = 3, k = 4$ and $n = 8$. An (n_0, k_0) tree code is linear, time invariant, and has finite wordlength $k = (m+1) \cdot k_0$ and is called an (n, k) *systematic convolutional code* [21], pp 361-364. This means one can refer to the same code as an (n_0, k_0) tree code or as an (n, k) convolutional code. Generally k is significantly larger than k_0 which should avoid confusion.

4.2.1.1 Polynomial Description of Convolutional Codes

An $((m+1)n_0,(m+1)k_0)$ convolutional code with constraint length $v = mk_0$ can be encoded by n_0 sets of finite impulse response (FIR) filters, each set consisting of k_0 FIR filters [21], pp 348-350. The input to the decoder is a stream of symbols with a rate of k_0 symbols per unit time and the output to the channel is a stream of n_0 symbols per unit time.

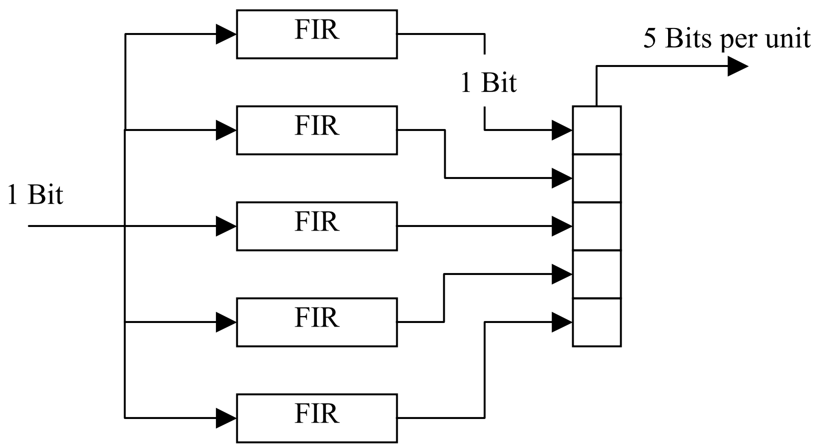

Figure 4.3 A convolutional encoder

Each FIR filter can be represented by a polynomial of degree of at most m. If the input stream is written as a polynomial (possibly of infinite length) the operation of the filter can be written as polynomial multiplication. In this way, the encoder for the convolutional code can be represented by a set of polynomials, and thus the code itself can also be represented by this same set of polynomials. That is, the set of codewords that this set of polynomials will produce. These polynomials are called the generator polynomials of the code.

In contrast to block codes, which are described by a single generator polynomial, a convolutional code requires multiple generator polynomials to describe it, a total of $k_0 \cdot n_0$ polynomials. These can be put together in a generator-polynomial matrix, a k_0 by n_0 matrix of polynomials given by:

$$G(x) = [g_{i,j}(x)] \tag{4.2}$$

For example the matrices of generator polynomials for the encoder in Figure 4.2 is given by:

$$G(x) = [g_{1,1}(x), g_{2,1}(x)] \tag{4.3}$$

with

$$g_{1,1}(x) = x^3 + x^2 + 1 \tag{4.4}$$

and

$$g_{2,1}(x) = x^3 + x + 1 \tag{4.5}$$

Thus

$$G(x) = [x^3 + x^2 + 1, \quad x^3 + x + 1] \tag{4.6}$$

As the output of the convolutional encoder interleaves the two output streams from the two FIR filters the generator matrix can also be shown as:

$$G(x) = [g_{1,1_0} g_{2,1_0} \quad g_{1,1_1} g_{2,1_1} \quad \cdots \quad g_{1,1_m} g_{2,1_m}] \tag{4.7}$$

$$G(x) = [1 \quad 1 \qquad 0 \quad 1 \qquad 1 \quad 0 \qquad 1 \quad 1] \tag{4.8}$$

or for $g_{1,1}(x) = x^3 + x^2 + 1$ and $g_{2,1}(x) = x^3$ one gets

$$G(x) = [1 \quad 0 \qquad 0 \quad 0 \qquad 1 \quad 0 \qquad 1 \quad 1] \tag{4.9}$$

4.2.1.2 Matrix Description of Convolutional Codes

A convolutional code consists of an infinite number of infinitely long codewords. It is linear and can be described by an infinite generator matrix. A large number of generator matrices can be used to describe each code, but only a few of them are convenient to deal with. Even in the best case, a generator matrix for a convolutional code is more cumbersome than a generator matrix for a block code [21], pp 361-364.

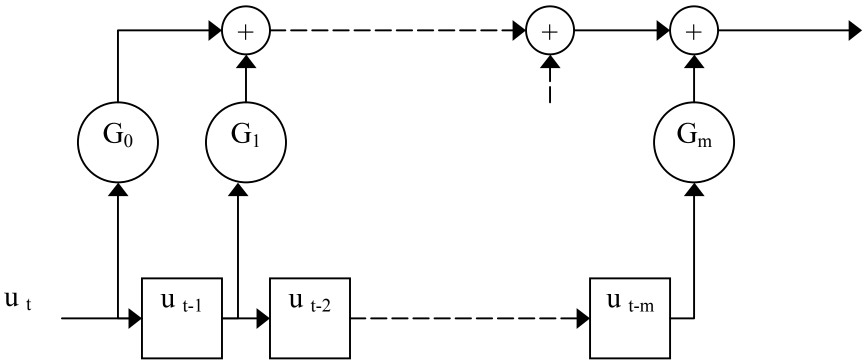

Figure 4.4 A general convolutional encoder (without feedback)

The generator polynomials, indexed by i and j, can be written

$$g_{ij}(x) = \sum_l g_{ijl} x^l \tag{4.10}$$

For each l, let G_l be the k_0 by n_0 matrix $G_l = [g_{ijl}]$

Then the code of blocklength n is

$$G^n = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m \\ 0 & G_0 & G_1 & \cdots & G_{m-1} \\ 0 & 0 & G_0 & \cdots & G_{m-2} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & 0 & G_0 \end{bmatrix} \tag{4.11}$$

where each **0** is a k_0 by n_0 matrix of zeros. The generator matrix for the convolutional code is

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \cdots \\ \vdots & & & & & & & & & \end{bmatrix} \tag{4.12}$$

where the matrix continues indefinitely down and to the right. Equation (4.12) depicts such a a bi-infinite systematic convolutional encoder. Except for the diagonal band of *m* non-zero submatrices, all other entries are equal to zero. For a systematic convolutional code, these two matrices can also be written as:

$$G^{(n)} = \left[\begin{array}{cc:cc:cc:c:cc} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m \\ 0 & 0 & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} \\ 0 & 0 & 0 & 0 & I & P_0 & \cdots & 0 & P_{m-2} \\ & \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & I & P_0 \end{array}\right] \tag{4.13}$$

and

$$G = \left[\begin{array}{cc:cc:cc:c:cc:cc:cc:c} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} & 0 & P_m & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & I & P_0 & \cdots & 0 & P_{m-2} & 0 & P_{m-1} & 0 & P_m & \cdots \\ \vdots & & & & & & & & \vdots & 0 & P_{m-2} & 0 & P_{m-1} & \cdots \\ & & & & & & & & & & \vdots & 0 & P_{m-2} & \cdots \\ & & & & & & & & & & & & \vdots & \end{array}\right] \tag{4.14}$$

where the pattern is repeated, right-shifted in every row, and unspecified matrix entries to the left and right are filled with zeros. Here **I** is a k_0 by n_0 identity matrix, **0** is a k_0 by n_0 matrix of zeros and $P_0,\ldots,P_m$ are k_0 by $(n_0 - k_0)$ matrices. The first row describes the encoding of the first information frame into the first *m* codeword frames. One should interpret this matrix expression in terms of the shift-register description of the encoder.

If the data symbols $d_0, d_1, \ldots, d_{k-1}$ of every message ***d*** are unchanged and appear in the codeword $u = (u_o, \ldots, u_{n-1})$, then the code is said to be a systematic code [22]. The generator matrix **G** in equation (4.15) is in its systematic form.

$$G = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ 0 & 1 & 0 & \cdots & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ & \vdots & & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} = \begin{bmatrix} I_k & P \end{bmatrix} \tag{4.15}$$

The parity equations P introduce redundancy to the data being transmitted based on relations between various data symbols. A decoder of a convolutional code (refer to section 4.2.5 - The Viterbi Decoding Algorithm) exploits this redundancy to correct errors that occurred during transmission. A big part of fast convolutional attacks on stream ciphers is the finding of suitable parity equations within the equivalent block-code description of the LFSR.

4.2.2 Converting a LFSR to a Block Code

There is a corresponding $l \times N$ generator matrix G_{LFSR} which produces the same output as a LFSR, namely $U = u_0 G_{LFSR}$ where u_0 is the initial stare of the LFSR. A LFSR of length l has a set of possible LFSR code vectors U_n denoted by L [3]. Clearly $|L| = 2^l$ and for a fixed length N the truncated sequences from L is also a linear $[N, l]$ block code referred to as C. It can easily be seen that for any code vector, all its cyclic shifts are also in L [22]. Using l linearly independent code vectors or LFSR output sequences U_n from L (which is the also the maximum amount of linearly independent vectors in L) the equivalent linear $[N, l]$ block code is be obtained:

$$\begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_l \end{bmatrix} = \begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} & \cdots & u_{0,N} \\ u_{1,0} & u_{1,1} & u_{1,2} & \cdots & u_{1,N} \\ \vdots & & & & \vdots \\ u_{l,0} & u_{l,1} & u_{l,2} & \cdots & u_{l,N} \end{bmatrix} \tag{4.16}$$

The matrix is now transformed to its systematic form to

$$C = \begin{bmatrix} 1 & 0 & 0 & \cdots & \\ 0 & 1 & 0 & \cdots & \\ \vdots & & & & \\ 0 & 0 & 0 & \cdots & \end{bmatrix} \tag{4.17}$$

A lot of effort can be saved by choosing the starting value for each LFSR sequence or code vector U_n as required for C in its reduced form. Thus for a LFSR of size l choose starting values:

$$\begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & u_{0,l-1} & u_{0,l} \\ u_{1,0} & u_{1,1} & \cdots & u_{1,l-1} & u_{1,l} \\ \vdots & & & & \vdots \\ u_{l,0} & u_{l,1} & \cdots & u_{l,l-1} & u_{l,l} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{4.18}$$

Thus the first code vector is obtained by using the starting value $[u_0 \quad u_1 \quad \cdots \quad u_l] = [1 \quad 0 \quad \cdots \quad 0]$ etc.

4.2.2.1 Example of Converting a LFSR to a Block Code.

The feedback taps of a LFSR are often specified using the polynomial representation. Consider a length 8 LFSR, with feedback polynomial

$$g(x) = x^8 + x^6 + x^5 + x^3 + 1 \tag{4.19}$$

Initially presented in section 2.2.1 and repeated here for convenience, the polynomial representation can be easily converted to a recurrence relation representation, which can be more easily mapped to the hardware representation of a LFSR. Setting $g(x) = 0$ gives:

$$0 = x^8 + x^6 + x^5 + x^3 + 1 \tag{4.20}$$

Multiplying by x^n produces:

$$0 = x^{n+8} + x^{n+6} + x^{n+5} + x^{n+3} + x^n \tag{4.21}$$

Multiply by x^{-8} then gives:

$$0 = x^n + x^{n-2} + x^{n-3} + x^{n-5} + x^{n-8} \tag{4.22}$$

Now replacing x^n with a_n results in:

$$0 = a_n + a_{n-2} + a_{n-3} + a_{n-5} + a_{n-8} \tag{4.23}$$

As these all are GF(2) or modulo 2 operations $a_n = -a_n$; thus

$$a_n = a_{n-8} + a_{n-5} + a_{n-3} + a_{n-2} \tag{4.24}$$

which represents the LFSR shown in Figure 4.5 below.

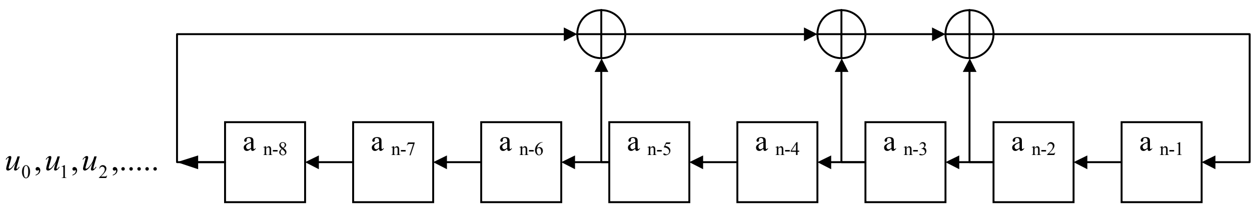

Figure 4.5 LFSR of size 8

Using the starting values

$$a_{n-8} = 1, \quad a_{n-7} = 0, \quad a_{n-6} = 0, \quad a_{n-5} = 0, \quad a_{n-4} = 0, \quad a_{n-3} = 0, \quad a_{n-2} = 0, \quad a_{n-1} = 0 \qquad (4.25)$$

one obtains

$$U_0 = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \cdots] \qquad (4.26)$$

Similarly using the starting value

$$a_{n-8} = 0, \quad a_{n-7} = 1, \quad a_{n-6} = 0, \quad a_{n-5} = 0, \quad a_{n-4} = 0, \quad a_{n-3} = 0, \quad a_{n-2} = 0, \quad a_{n-1} = 0 \qquad (4.27)$$

to obtain

$$U_1 = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \cdots] \qquad (4.28)$$

Continuing along the same lines gives

$$U_2 = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \cdots] \qquad (4.29)$$

$$U_3 = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad \cdots] \qquad (4.30)$$

$$U_4 = [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \cdots] \qquad (4.31)$$

$$U_5 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \cdots] \qquad (4.32)$$

$$U_6 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \cdots] \qquad (4.33)$$

$$U_7 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad \cdots] \qquad (4.34)$$

Which results in $[N, l]$ = [27,8] block code shown below. Obviously N can be made any size by using a longer code vector obtained from the LFSR output.

$$G_{LFSR} = \begin{bmatrix} 1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0 \\ 0&1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1 \\ 0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1 \\ 0&0&0&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&1&0&0&0&1&1&1&1 \\ 0&0&0&0&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&1&0&0&0&1&1&1 \\ 0&0&0&0&0&1&0&0&1&0&0&1&0&1&1&1&0&0&0&0&1&1&0&1&1&0&1 \\ 0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0&0 \end{bmatrix} \tag{4.35}$$

The same result can be achieved by using any initial values to obtain arbitrary code vectors U_n. The Gauss-Jordan reduction method can then be used to obtain the same $[N,l]$ block code C.

4.2.3 Finding Parity Equations within a Block Code

The finding of parity equations [3] in a block code is explained in this section using the equivalent block code obtained for a LFSR as derived in the previous section (section 4.2.2). The generator matrix for a block code is written in its systematic form,

$$G_{LFSR} = [I_l \quad Z] \tag{4.36}$$

which is already the case when using the method described in the previous section (section 4.2.3) used to derive an equivalent block code from a LFSR.

To find these equations one can start by considering the index position $n = B+1$ and introducing the following notation for the generator matrix [3],

$$G_{LFSR} = \begin{bmatrix} I_{B+1} & Z_{B+1} \\ 0_{l-(B+1)} & Z_{l-(B+1)} \end{bmatrix} \tag{4.37}$$

The parameter l is the size of the LFSR and B is a design parameter, which will later be shown to be the size of the convolutional encoder that is in the process of being constructed. Using equation (4.35) as an example and choosing $B = 4$, the form described by equation (4.37) is easily understood:

$$\begin{array}{c} \xleftrightarrow{\quad B+1 \quad} \\ G_{LFSR} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{array}{l} \updownarrow B+1 \\ \updownarrow l-(B+1) \end{array} \updownarrow l \end{array}$$

(4.38)

The aim is to find all parity check equations in G_{LFSR} that involve a current symbol u_n, an arbitrary linear combination of the b previous symbols $u_{n-1},...,u_{n-B}$, together with at most t other symbols. For simplicities sake only $t = 2$ is considered, as shown below:

$$\begin{aligned} u_n + c_{11} \cdot u_{n-1} + c_{21} \cdot u_{n-2} + \cdots + c_{B1} \cdot u_{n-B} + u_{n+i_1} + u_{n+j_1} &= 0 \\ u_n + c_{12} \cdot u_{n-1} + c_{22} \cdot u_{n-2} + \cdots + c_{B2} \cdot u_{n-B} + u_{n+i_2} + u_{n+j_2} &= 0 \\ &\vdots \\ u_n + c_{1m} \cdot u_{n-1} + c_{2m} \cdot u_{n-2} + \cdots + c_{Bm} \cdot u_{n-B} + u_{n+i_m} + u_{n+j_m} &= 0 \end{aligned} \tag{4.39}$$

Thus one tries to find the index positions i and j together with the linear combination of $c_1 = 0 \;\; or \;\; c_1 = 1, \;\; c_2 = 0 \;\; or \;\; c_2 = 1 \;\; \cdots \;\; c_B = 0 \;\; or \;\; c_B = 1$ that satisfy each of the equation above.

The column vectors in G_{LFSR} are numbered as follows:

$$G_{LFSR} = \begin{bmatrix} I_{B+1} & g_1 & \cdots & g_{N-(B+1)} \end{bmatrix} \tag{4.40}$$

or

$$G_{LFSR} = \begin{bmatrix} g_{-B} & g_{-(B-1)} & \cdots & g_0 & g_1 & \cdots & g_{N-(B+1)} \end{bmatrix} \tag{4.41}$$

The parameter B, in the equations above, is a design parameter chosen by the user, which will determine the size of the convolutional decoder to be constructed. To find parity equations in G_{LFSR} one wants to find the columns that satisfy an equation in equation (4.42):

$$c_B \cdot g_{-B} + c_{(B-1)} \cdot g_{-(B-1)} + \cdots + c_1 \cdot g_{-1} + g_0 + g_i + g_j = \vec{0} \tag{4.42}$$

The column pairs g_i, g_j need to satisfy the following:

$$[g_i + g_j]^T = \begin{bmatrix} \overbrace{*,*,\ldots,*,}^{B} & 1, & \overbrace{0,0,\cdots,0}^{l-(B+1)} \end{bmatrix} \tag{4.43}$$

As a column pair satisfying equation (4.43) with index positions i and j has now been found, all that remains to be done is the trivial job of determining $c_1 \cdots c_B$ to determine the column vectors g_{-B} to g_{-1} which will be used in equation (4.42). Terms where $c_y = 0$, with y being any arbitrary index position, are omitted.

4.2.3.1 Example for Finding Parity Equations in a Block Code

Equation (4.38) will now be used as an example with B chosen as $B = 4$. Column pairs in the $Z_{l-(B+1)}$ part need to be found which, when added together, form the zero column vector $0_{l-(B+1)}$.

Using the requirements set by equation (4.43) all pairs of columns g_i, g_j are found such that

$$[g_i + g_j]^T = \begin{bmatrix} \overbrace{*,*,*,*,}^{4} & 1, & \overbrace{0,0,0}^{3} \end{bmatrix} \tag{4.44}$$

$$
\begin{array}{c}
\begin{array}{ccccccccccccccccccccccccccc}
n-4 & n-3 & n-2 & n-1 & n & n+1 & n+2 & n+3 & n+4 & n+5 & n+6 & n+7 & n+8 & n+9 & n+10 & n+11 & n+12 & n+13 & n+14 & n+15 & n+16 & n+17 & n+18 & n+19 & n+20 & n+21 & n+22
\end{array} \\
G_{LFSR} = \begin{bmatrix}
1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0\\
0&1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1\\
0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1\\
0&0&0&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&1&0&0&0&1&1&1&1\\
0&0&0&0&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&1&0&0&0&1&1&1\\
0&0&0&0&0&1&0&0&1&0&0&1&0&1&1&1&0&0&0&0&1&1&0&1&1&0&1\\
0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&0&0&0&0&0&0&1&1&1&0&0
\end{bmatrix}
\end{array}
\tag{4.45}
$$

Assigning the index values as above to G_{LFSR}, the following eight column pairs in equation (4.38) to satisfy equation (4.44) are found:

Table 4.1 Finding parity equations in equation (4.38)

Columns	
g_1	g_{16}
g_4	g_9
g_5	g_6
g_7	g_{10}
g_9	g_{17}
g_{12}	g_{15}
g_{13}	g_{15}
g_{14}	g_{15}

Vectors g_4 and g_9 in Table 4.1 above will now be used to illustrate how the column vectors that were found to satisfy equation (4.44) are used to construct a parity equation. It can be seen that

$$g_4 + g_9 = \begin{bmatrix}1\\0\\0\\1\\0\\1\\1\\0\end{bmatrix} + \begin{bmatrix}1\\1\\1\\0\\1\\1\\1\\0\end{bmatrix} = \begin{bmatrix}0\\1\\1\\1\\1\\0\\0\\0\end{bmatrix} \tag{4.46}$$

satisfies equation (4.44).

The columns with indexes $-B \cdots 0$ that satisfy equation (4.42) now need to be found:

$$g_{-3} + g_{-2} + g_{-1} + g_0 + [g_4 + g_9] = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\1\\1\\1\\1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \tag{4.47}$$

Which results in the parity equation

$$u_{n-3} + u_{n-2} + u_{n-1} + u_n + u_{n+4} + u_{n+9} = 0 \tag{4.48}$$

Re-writing the above gives:

$$u_n + u_{n-1} + u_{n-2} + u_{n-3} + u_{n+4} + u_{n+9} = 0 \tag{4.49}$$

Applying the same process to the other column pairs found in Table 4.1 provides:

Table 4.2 Parity equations found in G_{LFSR}, equation (4.38), with B=4

Equation no	**Columns**		**Parity Equation**
1	g_1	g_{16}	$u_n + u_{n+1} + u_{n+16} = 0$
2	g_4	g_9	$u_n + u_{n-1} + u_{n-2} + u_{n-3} + u_{n+4} + u_{n+9} = 0$

Equation no	Columns		Parity Equation
3	g_5	g_6	$u_n + u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4} + u_{n+5} + u_{n+6} = 0$
4	g_7	g_{10}	$u_n + u_{n-1} + u_{n-2} + u_{n-4} + u_{n+7} + u_{n+10} = 0$
5	g_9	g_{17}	$u_n + u_{n-2} + u_{n-3} + u_{n-4} + u_{n+9} + u_{n+17} = 0$
6	g_{12}	g_{15}	$u_n + u_{n-1} + u_{n-3} + u_{n-4} + u_{n+12} + u_{n+15} = 0$
7	g_{13}	g_{15}	$u_n + u_{n-1} + u_{n-2} + u_{n-3} + u_{n+13} + u_{n+15} = 0$
8	g_{14}	g_{15}	$u_n + u_{n-2} + u_{n+14} + u_{n+15} = 0$

4.2.3.2 Verifying a Parity Equation

A quick check that can be used to verify a parity equation is the requirement that a parity-check polynomial $p(x)$ must be divisible by the generator polynomial $g(x)$.

$$p(x)\bmod\left(g(x)\right) = 0 \text{ in } GF(2) \tag{4.50}$$

4.2.3.2.1 Example

The parity equation $u_n + u_{n+1} + u_{n+16} = 0$ from Table 4.2 will be used as an example:

Replacing $u_n = x^n$ results in

$$x^n + x^{n+1} + x^{n+16} = 0 \tag{4.51}$$

Multiplying with x^{-n} produces

$$1 + x + x^{16} = 0 \tag{4.52}$$

Thus resulting in the polynomial

$$p(x) = x^{16} + x + 1 \tag{4.53}$$

The feedback polynomial for the LFSR in Figure 4.5 (used to generate G_{LFSR}, equation (4.38)) was originally given in example (4.19) and is repeated here for convenience:

$$g(x) = x^8 + x^6 + x^5 + x^3 + 1 \tag{4.54}$$

All calculations are shown below for illustration purposes. Normally one would only be interested in the remainder and would not try to calculate the quotient.

$$
\begin{array}{rl}
 & x^8+x^6+x^5+x^4+x^3+x+1 \\
x^8+x^6+x^5+x^3+1 & \overline{\big)\, x^{16}+x+1} \\
 & \underline{-(x^{16}+x^{14}+x^{13}+x^{11}+x^8)} \\
 & x^{14}+x^{13}+x^{11}+x^8+x+1 \\
 & \underline{-(x^{14}+x^{12}+x^{11}+x^9+x^6)} \\
 & x^{13}+x^{12}+x^9+x^8+x^6+x+1 \\
 & \underline{-(x^{13}+x^{11}+x^{10}+x^8+x^5)} \\
 & x^{12}+x^{11}+x^{10}+x^9+x^6+x^5+x+1 \\
 & \underline{-(x^{12}+x^{10}+x^9+x^7+x^4)} \\
 & x^{11}+x^7+x^6+x^5+x^4+x+1 \\
 & \underline{-(x^{11}+x^9+x^8+x^6+x^3)} \\
 & x^9+x^8+x^7+x^5+x^4+x^3+x+1 \\
 & \underline{-(x^9+x^7+x^6+x^4+x)} \\
 & x^8+x^6+x^5+x^3+1 \\
 & \underline{-(x^8+x^6+x^5+x^3+1)} \\
 & 0
\end{array}
\tag{4.55}
$$

As can be seen from the calculations in equation (4.55) that the requirement set by equation (4.50) has been satisfied.

4.2.3.3 The Expected Number of Parity Equations within a Block Code

To find a parity equation one needs to find columns within G_{LFSR} where the sum of the two columns provides the following result $g_i + g_j = [***\overbrace{1 \quad 0 \quad \cdots \quad 0}^{l-B}]^T$. As the rows of G_{LFSR} are PN-sequences (refer to section 4.2.2) the probability of two corresponding bits in two columns either providing 0 or 1 as required is equal to 0.5. Thus the probability of finding any two columns that satisfy equation (4.43) is equal to 0.5^{l-B}.

In a generator matrix, G_{LFSR} of dimensions $l \times N$ there are N columns to be compared with each other to find possible parity equations. When comparing N different columns vectors with each other, this amounts to $\frac{1}{2} \cdot N \cdot (N-1)$ comparisons.

When combining these two equations it is found that the amount of parity equations that can be expected to be found in a generator matrix, G_{LFSR} of dimensions $l \times N$, is given by equation (4.56) below for a certain value of B. Γ is chosen to be equal to the number of parity equations.

$$E[\Gamma] = \frac{1}{2} \cdot N \cdot (N-1) \cdot \frac{1}{2^e} \tag{4.56}$$

where

$$e = l - B \tag{4.57}$$

The graph shown in Figure 4.6 below gives a graphical representation of equation (4.56) for various values of N and e. Tables of the exact values are given in Appendix D.

To simplify the presentation of Figure 4.6 the new parameter (n) in equation (4.58) below is introduced. N is chosen to be equal to the number of columns to be searched. One can never search more columns than there are ciphertext bits available, thus N is also equal to the minimum amount of required ciphertext bits. This is because of the fact that the parity equations are subsequently applied to the ciphertext to reconstruct the original LFSR output. Because of this there is no use in searching more columns within G_{LFSR} than there are available ciphertext bits as any parity equation containing an index larger than the number of ciphertext bits available, cannot be used.

$$n = \log_2 N \tag{4.58}$$

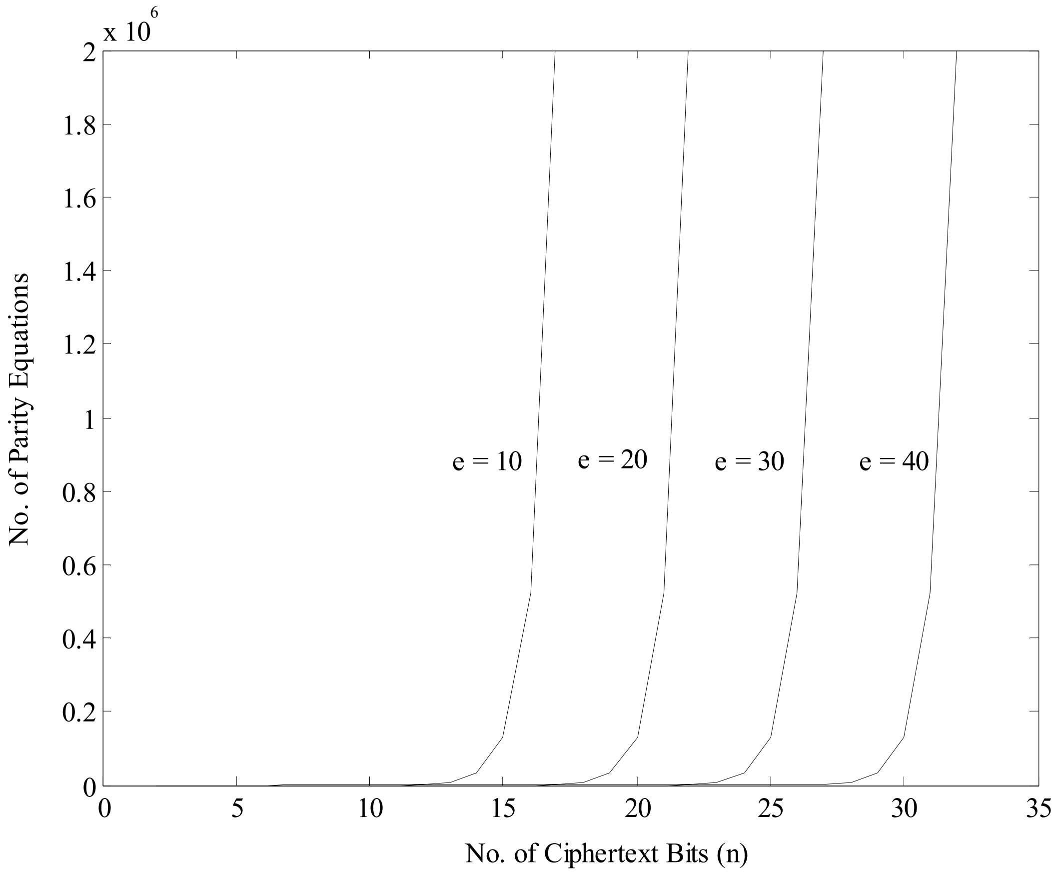

Figure 4.6 The expected number of parity equations to be found for selected values of e

Figure 4.6 represent the number of equations one can expect to find for selected values of *e*, showing a general indication of the minimum required size of B ($e = l - B$) and the minimum amount of ciphertext that is required for finding sufficient parity equations. A number of reference tables showing the number of parity equations that can be found for $2 \leq e \leq 49$ are presented in Appendix D. It can be seen from Figure 4.6 that a trade-off exists between *e* and the amount of ciphertext required, thus by reducing *e*, less ciphertext is needed to find sufficient parity equations. For the case of $l - B = 49$, to find any equations, one needs to search at least $N = 2^{26}$ columns, which amounts to at least $\approx 2^{51}$ operations, a figure close to impossible. The search for parity equations can however be performed in parallel, potentially allowing this number of columns to be searched.

4.2.4 Creating a Convolutional Encoder using Parity Equations

The parity equations found in the previous section (4.2.3) are now used to create a bi-infinite systematic convolutional encoder. The generator matrix for such a code is in the form as shown in equation (4.12) and repeated here for convenience.

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \cdots \\ \vdots & & & & & & & & & \end{bmatrix} \tag{4.59}$$

Identifying the parity check equations from equation (4.39) with the descriptive form of the convolutional code as in equation (4.12) gives [3]

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ \vdots \\ G_B \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & c_{11} & c_{12} & \cdots & c_{1m} \\ 0 & c_{21} & c_{22} & \cdots & c_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_{B1} & c_{B2} & \cdots & c_{Bm} \end{bmatrix} \tag{4.60}$$

Which results in a convolutional encoder with $R = 1/(m+1)$. This is one more than the number of equations found. One of the prerequisites of equation (4.39) is the fact that $c_{00} = 1, c_{01} = 1, c_{02} = 1, \cdots, c_{0m} = 1$ which is why it is never explicitly shown. The extra equation is derived from $u_n + u_n = 0$ and can be seen in the first column (Thus $c_{00} = 1, c_{10} = 0, c_{20} = 0, c_{30} = 0, \cdots, c_{B0} = 0$).

4.2.4.1 Example for Using Parity Equations to Create a Convolutional Encoder

Looking at the equations found in G_{LFSR} and shown in Table 4.2 it is seen that the parameters $c_{i1..m}$ were determined as follows (using the same order as the equations are shown in):

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} & c_{18} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} & c_{28} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} & c_{37} & c_{38} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} & c_{47} & c_{48} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.61}$$

Which gives

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.62}$$

4.2.5 The Viterbi Decoding Algorithm

The Viterbi algorithm is an asymptotical optimum algorithm for the decoding of convolutional codes in memoryless noise. The Viterbi algorithm is introduced here as a maximum likelihood decoding algorithm for convolutional codes.

4.2.5.1 The Trellis Diagram

A trellis diagram is an extension of a convolutional code's state diagram that explicitly shows the passage of time [20]. The rate 1/3 encoder shown in Figure 4.7 has two memory cells, which results in the state diagram with four states as shown in Figure 4.8.

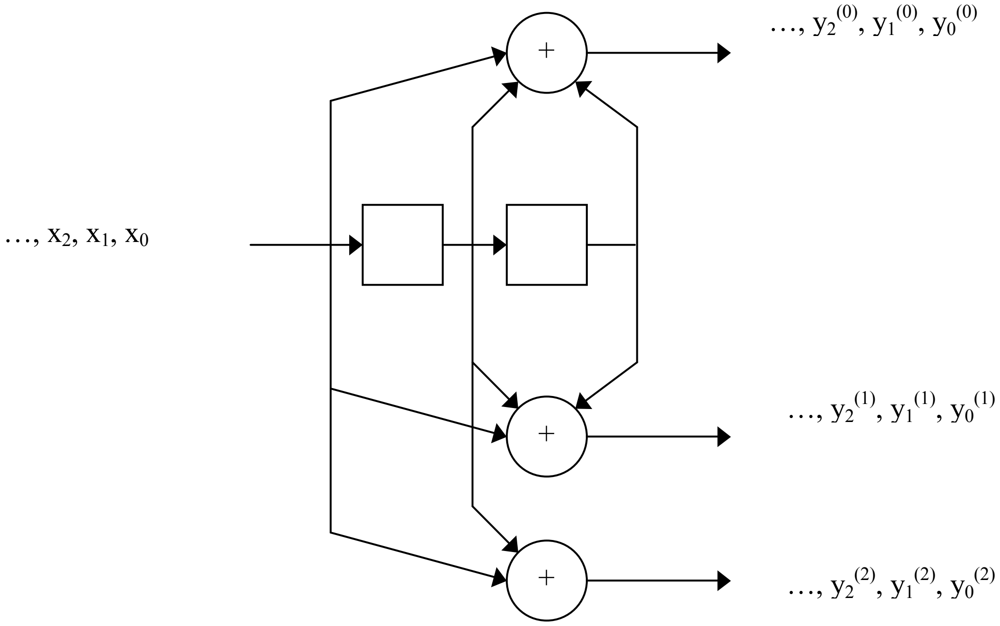

Figure 4.7 Encoder for a rate1/3 convolutional code

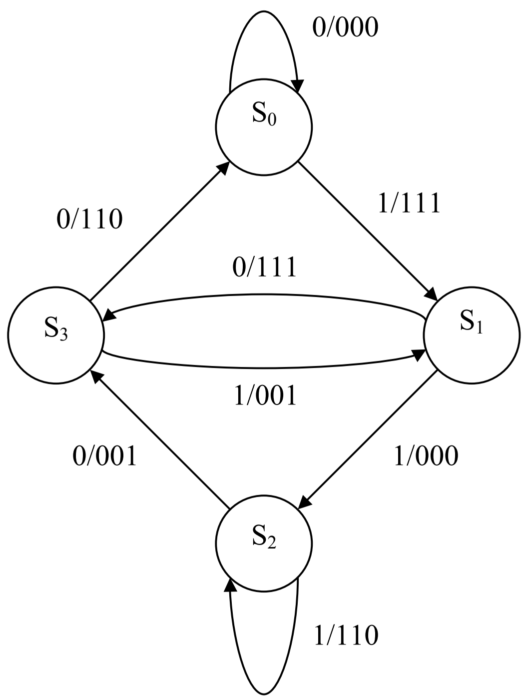

Figure 4.8 State diagram for encoder in Figure 4.7

In Figure 4.9 the state diagram is extended in time to form a trellis diagram. The branches of the trellis diagram are labelled with the output bits corresponding to the associated state transitions. The notation *jk* is used to identify the branch moving from state S_j to state S_k in the trellis. The corresponding output bits are denoted as $y_{jk} = (y_{jk,0}, y_{jk,1}, \cdots, y_{jk,(n-1)})$. The convolutional code is time invariant, thus y_{jk} is always the same.

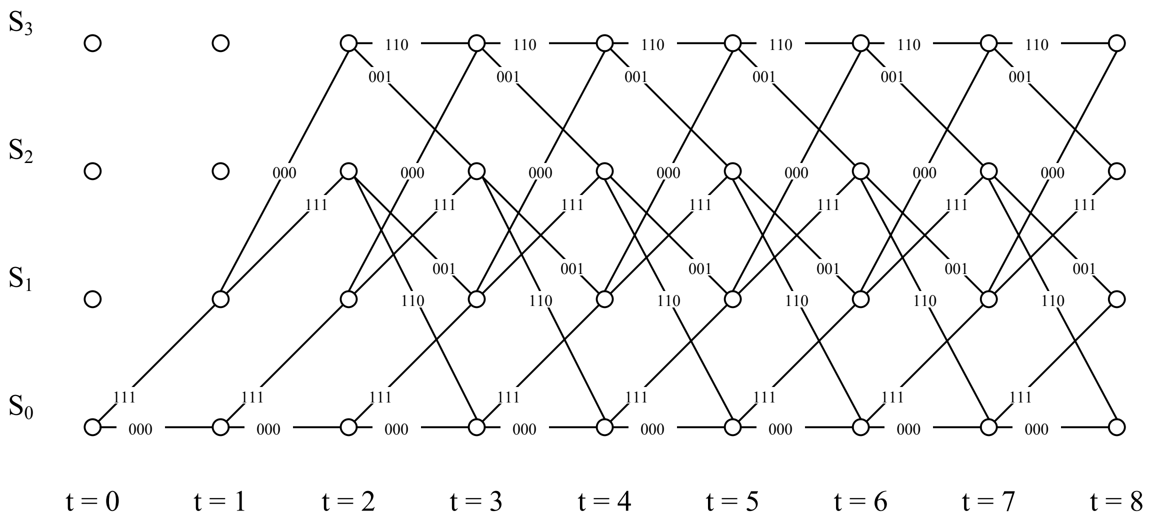

Figure 4.9 Trellis diagram for the encoder shown in Figure 4.7

Every code word in a convolutional code is associated with a unique path, starting and stopping at state S_0, through the associated trellis diagram. The trellis structure enables certain useful observations to be made. A general (n,k) binary convolutional encoder with total memory *M* and maximal memory order *m* will now be considered. The associated trellis diagram has 2^M nodes at each stage, or time increment *t*. There are 2^k branches leaving each node, one branch for each possible combination of input values. After time $t = m$, there are also 2^k branches entering each node. It is assumed that after the input sequence has been entered into the encoder, *m* state transitions are necessary to return the encoder to state S_0. Given an input sequence of $k \cdot L$ bits, the trellis diagram must have $L + m$ stages, the first and last stages starting and stopping respectively in state S_0.

There are thus 2^{kL} distinct paths through the general trellis, each corresponding to a convolutional code word of length $n(L+m)$. For example, the length-3 input sequence $x=(011)$ is shown in Figure 4.10 to correspond to a five-branch path associated with the $3(3+2)=15$-bit convolutional code word $y=(000 \quad 111 \quad 000 \quad 001 \quad 110)$. Note that all paths though the trellis intersect all other possible paths at one or more nodes. The Viterbi algorithm exploits this fact.

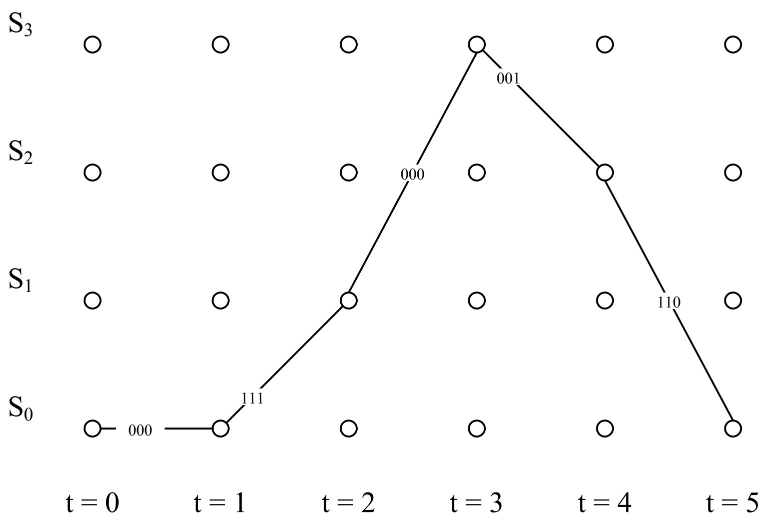

Figure 4.10 Trellis diagram for the input $x=(011)$ *to encoder shown in Figure 4.7*

The Viterbi decoder operates iteratively frame by frame, tracing a path through a trellis identical to that used by the encoder in an attempt to emulate the encoder's behaviour [21], pp 348-350. At any frame time the encoder does not know which node the encoder reached and thus does not try to decode this node immediately. Given the received sequence, the decoder determines the most likely path to every node, and it also determines the distance called the discrepancy of the path. If all paths in the set of most likely paths begin in the same way, the decoder knows how the encoder began.

Then in the next frame, the decoder determines the most likely path to each of the new nodes of that frame. But to get to any one of the new nodes the path must pass through one of the old nodes. One can get the candidate paths to a new node by extending to this new node each of the old paths that can be thus extended. The most likely path is found by adding the incremental discrepancy of each path extension to the discrepancy of the path to the old node. As already mentioned there are 2^k such paths to each new node, and the path with the smallest discrepancy is the most likely path to the new node. In this case as already mentioned k is always equal to 1 as one is always working with rate

$1/(m+1)$ encoders. This means there are always two paths entering each node and two paths leaving each node. At the end of the iteration, the decoder knows the most likely path to each of the nodes in the new frame.

When looking at the set of surviving paths to the set of nodes at the *r*th frame, one or more of the nodes at the first frame time will be crossed by these paths. If all of the paths cross through the same node at the first frame time, then regardless of which node the encoder visits at the *r*th frame time, one knows the most likely node it visited at the first frame time. That is, one knows the first information frame even though one has not yet made a decision for the *r*-th frame.

To implement a Viterbi decoder, one must choose a decoding-window width *b,* usually several times as big as the blocklength. At frame time $t = n$, the decoder examines all surviving paths to see that they agree in the first branch. This branch defines a decoded information frame, which is passed out of the decoder.

Now the decoder drops the first branch and takes in a new frame of the received word for the next iteration. If again all surviving paths pass through the same node of the oldest surviving frame, then this information frame is decoded. The process continues in this way, decoding frames indefinitely.

4.2.5.2 The Viterbi Algorithm

The node corresponding to state S_j at time t is denoted $S_{j,t}$. Each node in the trellis is to be assigned a value $V(S_{j,t})$. The node values are computed as follows:

(1) Set $V(S_{0,0}) = 0$ and $t = 0$.

(2) At time t, compute the partial path metrics for all paths entering each node.

(3) Set $V(S_{k,t})$ equal to the best partial path metric entering the node corresponding to state S_k at time t. Ties can be broken by randomly choosing any one of the two. The non-surviving branches are deleted from the trellis.

(4) If $t < L + m$, increment t and return to step (2).

4.2.5.3 Calculating Path Metrics

Each path in the trellis is assigned a metric [20]. The maximum likelihood (ML) decoder selects, by definition, the estimate y' that maximizes the probability $p(r \mid y')$. If the distribution of the source words is uniform, then the tow decoders are identical and can be related by Bayes' rule:

$$p(r \mid y)p(y) = p(y \mid r)p(r) \tag{4.63}$$

A rate k/n convolutional encoder takes k input bits and generates n output bits with each shift of its internal registers. Suppose that one has an input sequence x composed of L k-bit blocks.

$$x = (x_0^{(0)}, x_0^{(1)}, \cdots, x_0^{(k-1)}, x_1^{(0)}, x_1^{(1)}, \cdots, x_1^{(k-1)}, x_{L-1}^{(k-1)}) \tag{4.64}$$

The output sequence y will consist of L n-bit blocks (one for each input block) as well as m additional blocks, where m is the length of the longest shift register in the encoder.

$$y = (y_0^{(0)}, y_0^{(1)}, \cdots, y_0^{(n-1)}, y_1^{(0)}, y_1^{(1)}, \cdots, y_1^{(n-1)}, y_{L-1+m}^{(n-1)}) \tag{4.65}$$

A noise-corrupted version r of the transmitted code word arrives at the receiver, where the decoder generates a maximum likelihood estimate y' of the transmitted sequence. r and y' have the following form:

$$r = (r_0^{(0)}, r_0^{(1)}, \cdots, r_0^{(n-1)}, r_1^{(0)}, r_1^{(1)}, \cdots, r_1^{(n-1)}, r_{L-1+m}^{(n-1)}) \tag{4.66}$$

$$y' = (y'^{(0)}_0, y'^{(1)}_0, \cdots, y'^{(n-1)}_0, y'^{(0)}_1, y'^{(1)}_1, \cdots, y'^{(n-1)}_1, y'^{(n-1)}_{L-1+m}) \tag{4.67}$$

One assumes that the channel is memoryless meaning that the noise process affecting a given bit in the received word r is independent of the noise process affecting all of the other received bits. Since the probability of joint independent events is simply the product of the probabilities of the individual events [20] it follows that:

$$p(r \mid y') = \prod_{i=0}^{L+m-1} [p(r_i^{(0)} \mid y'^{(0)}_i) p(r_i^{(1)} \mid y'^{1}_i) \cdots p(r_i^{(n-1)} \mid y'^{n-1}_i)] \tag{4.68}$$

$$\Rightarrow p(r \mid y') = \prod_{i=0}^{L+m-1} \left(\prod_{j=0}^{n-1} p(r_i^{(j)} \mid y'^{(j)}_i) \right) \tag{4.69}$$

There are two sets of product indices, one corresponding to the block numbers (subscripts) and the other corresponding to bits within the blocks (superscripts). By taking the logarithm of each side of equation (4.69) one obtains the log likelihood function

$$\log\left(p(r \mid y')\right) = \sum_{i=0}^{L+m-1}\left(\sum_{j=0}^{n-1}\log\left(p(r_i^{(j)} \mid y'^{(j)}_i)\right)\right) \tag{4.70}$$

In hardware implementations of the Viterbi decoder, the summands in equation (4.70) are usually converted to a more easily manipulated form called the bit-metrics

$$M(r_i^{(i)} \mid y'^{(j)}_i) = a\left(\log\left(p(r_i^{(j)} \mid y'^{j}_i)\right) + b\right) \tag{4.71}$$

a and b are chosen such that the bit-metrics are small positive integers that can be easily manipulated by digital logic circuits. In this case the use of integers is preferred, as floating-point operations take significantly longer on a processor. The path metric for a code word y' is then computed as follows.

$$M(r \mid y') = \sum_{i=0}^{L+m-1}\left(\sum_{j=0}^{n-1} M(r_i^{(j)} \mid y'^{(j)}_i)\right) \tag{4.72}$$

If the probability of bit-errors is independent of the value of the transmitted bit, then the channel is said to be a binary symmetric channel as shown in Figure 4.11.

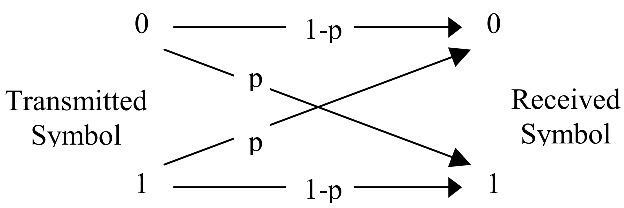

Figure 4.11 Binary symmetric channel model

If a and b in equation (4.71) are set to $a = \left(\log_2(p) - \log(1-p)\right)^{-1}$ and $b = -\log_2(1-p)$, the bit-metrics are independent of the value of the crossover probability p.

$$M(r_i^{(j)} \mid y_i^{(j)}) = \frac{1}{\log_2(p) - \log_2(1-p)}\left(\log_2\left(p(r_i^{(j)} \mid y_i^{(j)})\right) - \log_2(1-p)\right) \tag{4.73}$$

(4.74)

$M(r_i^{(j)} \mid y_i^{(j)})$	$r_i^{(j)} = 0$	$r_i^{(j)} = 1$
$y_i^{(j)} = 0$	0	1
$y_i^{(j)} = 1$	1	0

For the BSC case, the path metric for a code word y given a received word r is simply the Hamming distance $d(r,y)$. The surviving paths are those paths with the minimum partial path metric at each node.

However, setting $a=\log_2(1-p)-\log_2(p)$ and $b=-\log_2(p)$ the bit-metrics shown in equation (4.75) is produced.

$$\begin{array}{c|cc} M(r_i^{(j)} \mid y_i^{(j)}) & r_i^{(j)}=0 & r_i^{(j)}=1 \\ \hline y_i^{(j)}=0 & 1 & 0 \\ y_i^{(j)}=1 & 0 & 1 \end{array} \tag{4.75}$$

4.2.5.4 Example

The encoder in Figure 4.7 encodes the sequence $x=(1\quad 1\quad 0\quad 1\quad 0\quad 1)$, generating the code word $y=(111\quad 000\quad 001\quad 001\quad 111\quad 001\quad 111\quad 110)$.

This codeword, y, is transmitted over a noisy binary symmetric channel and is received as $r=(1\bar{0}1\quad \bar{1}00\quad 001\quad 0\bar{1}1\quad 111\quad \bar{1}01\quad 111\quad 110)$, with the indicated bits having been corrupted.

The bit-metrics as shown in equation (4.75) for calculating the partial path metrics to reconstruct the *transmitted* sequence will be used. All surviving paths are shown in bold print within the trellis diagrams (Figure 4.12 to Figure 4.19) shown in this section.

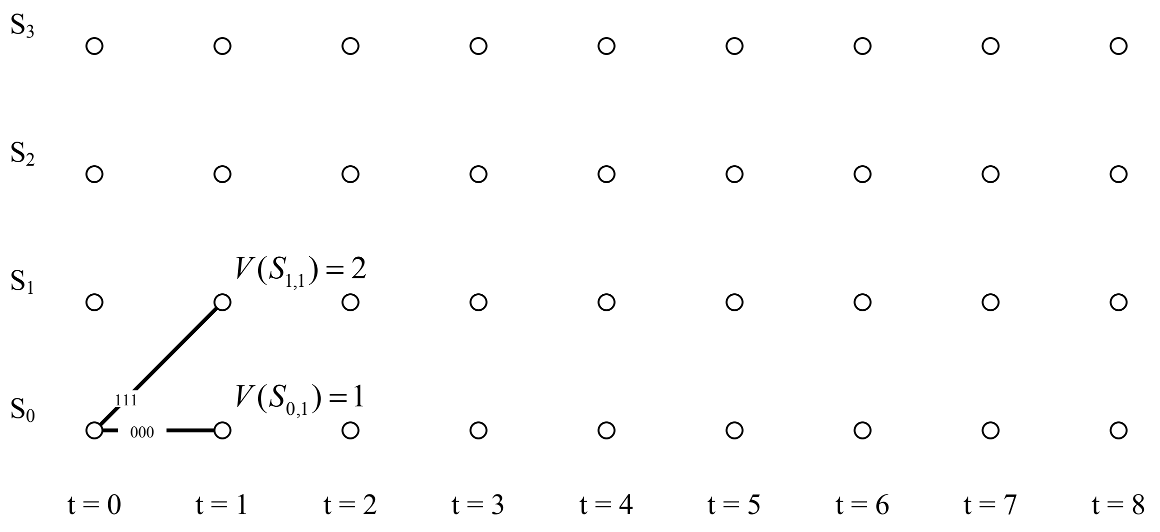

Figure 4.12 Trellis diagram at time $t=1$

The decoder always starts from state S_0 as it is known that this was the case on the encoder side.

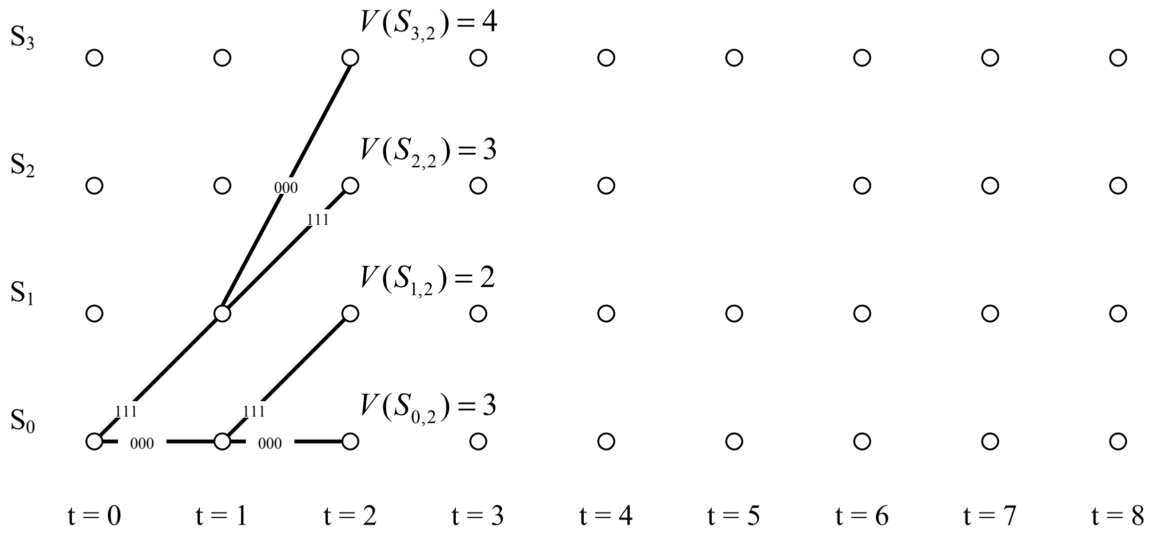

Figure 4.13 Trellis diagram at time $t = 2$

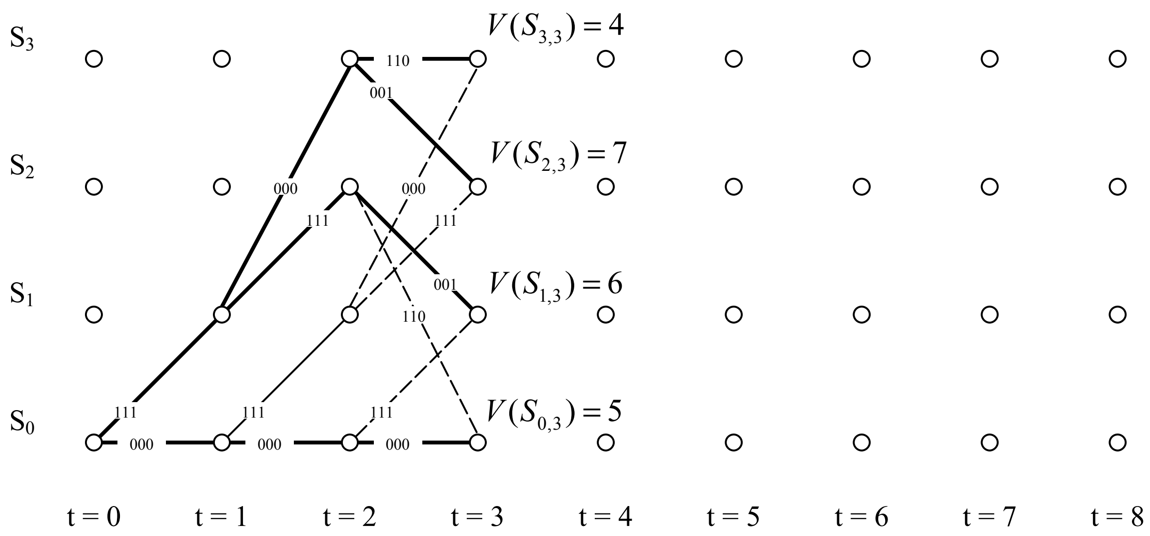

Figure 4.14 Trellis diagram at time $t = 3$

When using a using a $1/n$ rate encoder a maximum of one path may enter each node and a maximum of two paths may leave each node as can be seen for node $S_{3,2}$ in Figure 4.14 above.

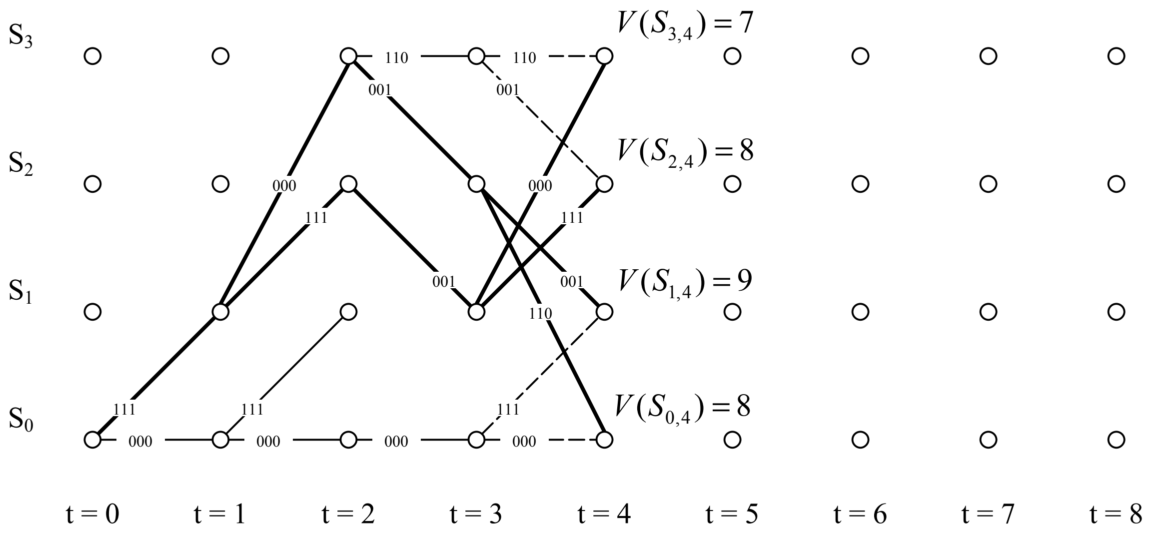

Figure 4.15 Trellis diagram at time $t = 4$

When using the path metrics as determined in equation (4.75) the path with the largest partial path metric is always chosen. Looking at Figure 4.15 the path from $S_{1,3}$ to $S_{3,4}$ survives as it's partial path metric is $V(S_{3,4}) = 7$ while the partial path metric for the path going from $S_{3,3}$ to $S_{3,4}$ would have been $V(S_{3,4}) = 5$.

Sometimes ties occur. Looking at Figure 4.16 it can be seen that the partial path metric from $S_{3,4}$ to $S_{3,5}$ is $V(S_{3,5}) = 9$, while the partial path metric from $S_{1,4}$ to $S_{3,5}$ is also $V(S_{3,5}) = 9$. The partial path that was chosen as shown was chosen randomly.

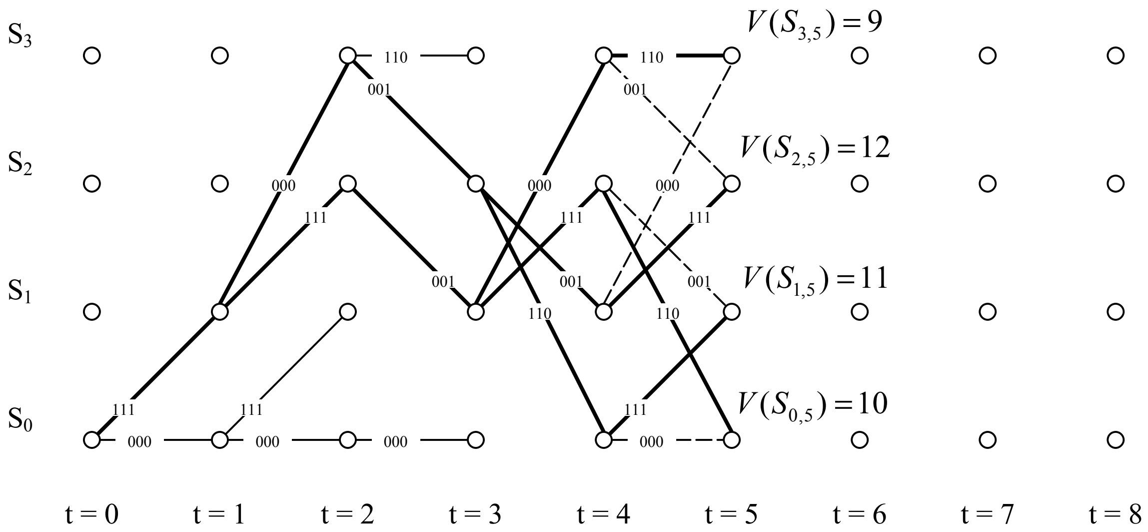

Figure 4.16 Trellis diagram at time $t = 5$

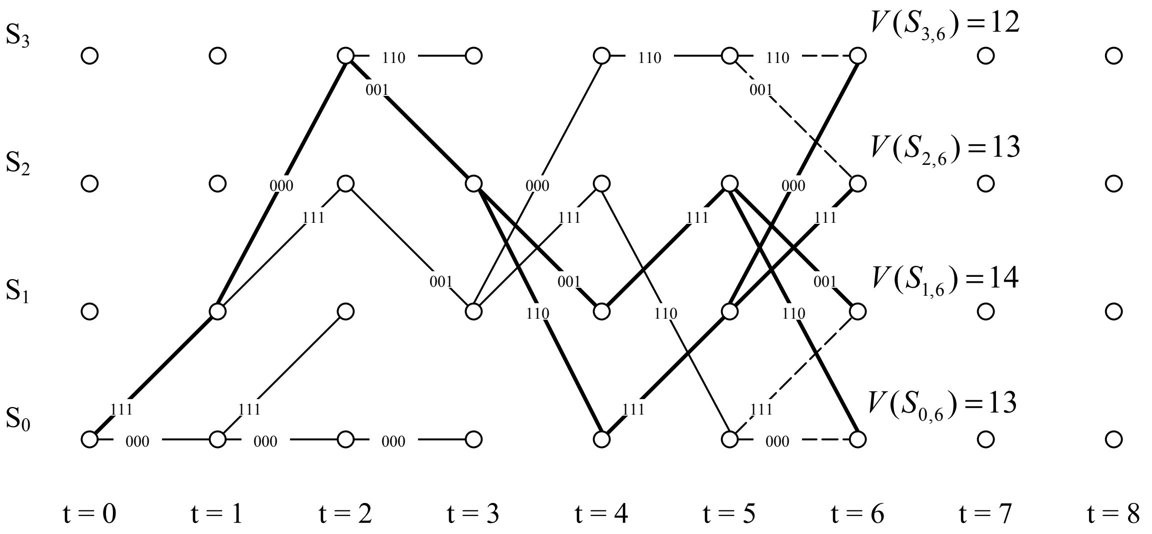

Figure 4.17 Trellis diagram at time $t = 6$

When inserting k input bits into a rate $1/n_0$ convolutional encoder, $n_0 \cdot (k + L)$ output bits are received, where L refers to the length of the LFSR. This means that the last L steps in the encoding process only received 0 as input as the last state of a Viterbi encoder is always S_0. When decoding, knowing this, one does not have to bother about the paths in the trellis diagram that result from inputting 1 into the encoder. Looking at Figure 4.18 and Figure 4.19 it is seen that only the paths for inputting 0 into the decoder are considered.

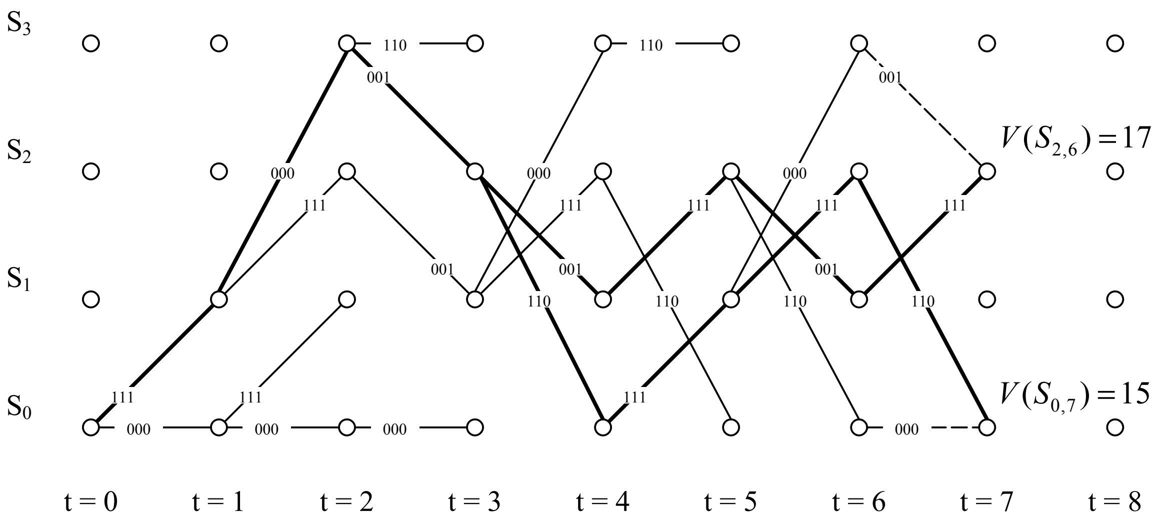

Figure 4.18 Trellis diagram at time $t = 7$

Looking at Figure 4.19 it can be seen that there is now only one surviving path left that can be traced backwards from the final state $S_{0,8}$. Doing this it is found that the estimated transmitted y' word is given by $y' = (111 \quad 000 \quad 001 \quad 001 \quad 111 \quad 001 \quad 111 \quad 110)$. Comparing this with the transmitted word $y = (111 \quad 000 \quad 001 \quad 001 \quad 111 \quad 001 \quad 111 \quad 110)$ it is seen that all errors that occurred during transmission were successfully corrected.

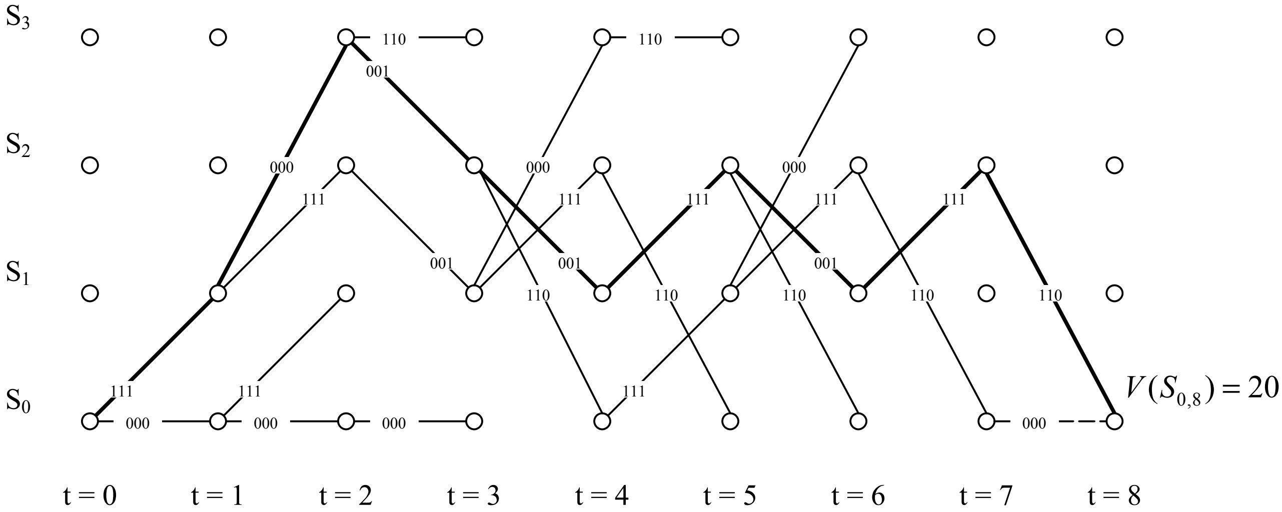

Figure 4.19 Trellis diagram at time $t = 8$

4.2.5.5 Generating the Received Stream

The convolutional encoder in section 4.2.4.1 was derived using a part of each parity equation (index positions $i = n - B$ to $i = n$) shown in Table 4.2. Index positions $i = n$ to $i = n + N - B$ (where N is the number of ciphertext bits available) are now used to transform the ciphertext z_i to the received sequence r_i.

$$
\begin{aligned}
r_n^{(0)} &= z_n \\
r_n^{(1)} &= z_{n+i_i} + z_{n+j_i} \\
r_n^{(2)} &= z_{n+i_2} + z_{n+j_2} \\
&\vdots \\
r_n^{(m)} &= z_{n+i_m} + z_{n+j_m}
\end{aligned}
\tag{4.76}
$$

The index positions $i_1 \cdots i_m$ and $j_1 \cdots j_m$ are the same as specified in equation (4.39). The equations introduced in (4.76) are now used to construct the received stream with a length of at least $(m+1)\cdot l$ (where $l = sizeof(LFSR)$) from the ciphertext stream.

4.2.5.6 Example for Generating the Received Stream

Applying (4.76) to the parity equations found in Table 4.2 (Also refer to Table 4.1) the following equations to generate the received stream are found:

$$\begin{aligned} r_n^{(0)} &= z_0 \\ r_n^{(1)} &= z_1 + z_{16} \\ r_n^{(2)} &= z_4 + z_9 \\ r_n^{(3)} &= z_5 + z_6 \\ r_n^{(4)} &= z_7 + z_{10} \\ r_n^{(5)} &= z_9 + z_{17} \\ r_n^{(6)} &= z_{12} + z_{15} \\ r_n^{(7)} &= z_{13} + z_{15} \\ r_n^{(8)} &= z_{14} + z_{15} \end{aligned} \tag{4.77}$$

4.2.5.7 Applying the Viterbi Algorithm for fast Correlation Attacks

The received stream is now sent through the Viterbi decoder created in section 4.2.4. If the decoding process is successful the output of the LFSR is extracted from the ciphertext. The Viterbi algorithm, as introduced in section 4.2.5, relies in part on the fact that the decoder always starts and ends in the all zero (S_0) state. This does not apply for the case of fast correlation attacks, as this type of attack has neither a fixed starting point nor endpoint. In this case the starting point is in the middle of the trellis diagram and the paths with the best metrics are kept. The initial metric $S_{n,0} = 0$ is assigned to each state.

In the following section 4.3.4 an example of the implementation of the methods introduced in this section is presented.

4.3 Introducing the Algorithm Based on a Small Example

4.3.1 Obtaining a Ciphertext Stream for Simulation Purposes

For this example the LFSR depicted in Figure 4.5 is continued to be used. Using the initial condition

$$IC = (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1) \tag{4.78}$$

a pn-sequence of length 35 shown below is generated.

$$a_{0x05} = (00000101111011000000111000110100000) \tag{4.79}$$

a_{0x05} is now sent through a BSC with an error probability of $p = 0.14$ resulting in a ciphertext sequence of length $N = 32$ (Erroneous bits have been overstruck).

$$z = (0000010\overline{0}\overline{0}1101100\overline{1}00011100011010\overline{1}\,\overline{1}00) \tag{4.80}$$

4.3.2 Find Parity Equations and Generate Convolutional Encoders

To successfully execute the fast correlation attack, a sequence of eight sequential bits need to be decoded correctly for finding the initial condition of an 8-bit LFSR. A ciphertext sequence with a length of 35 bits was generated in the previous section (4.3.1). Thus an equivalent block code that generates a stream of $35-8=27$ bits in length can be used to find a number of parity equations for the pn-sequence generated by the example LFSR.

Choosing $B = 4$ one can re-use the parity equations as found in Table 4.2 and the associated convolutional encoder (equation (4.62)) derived in the example shown in section 4.2.4.1. This convolutional encoder will be used as the Viterbi decoder and is shown here again for convenience:

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.81}$$

This is equivalent to the convolutional encoder, which is shown in Figure 4.20 below together with its state table shown in Table 4.3 below.

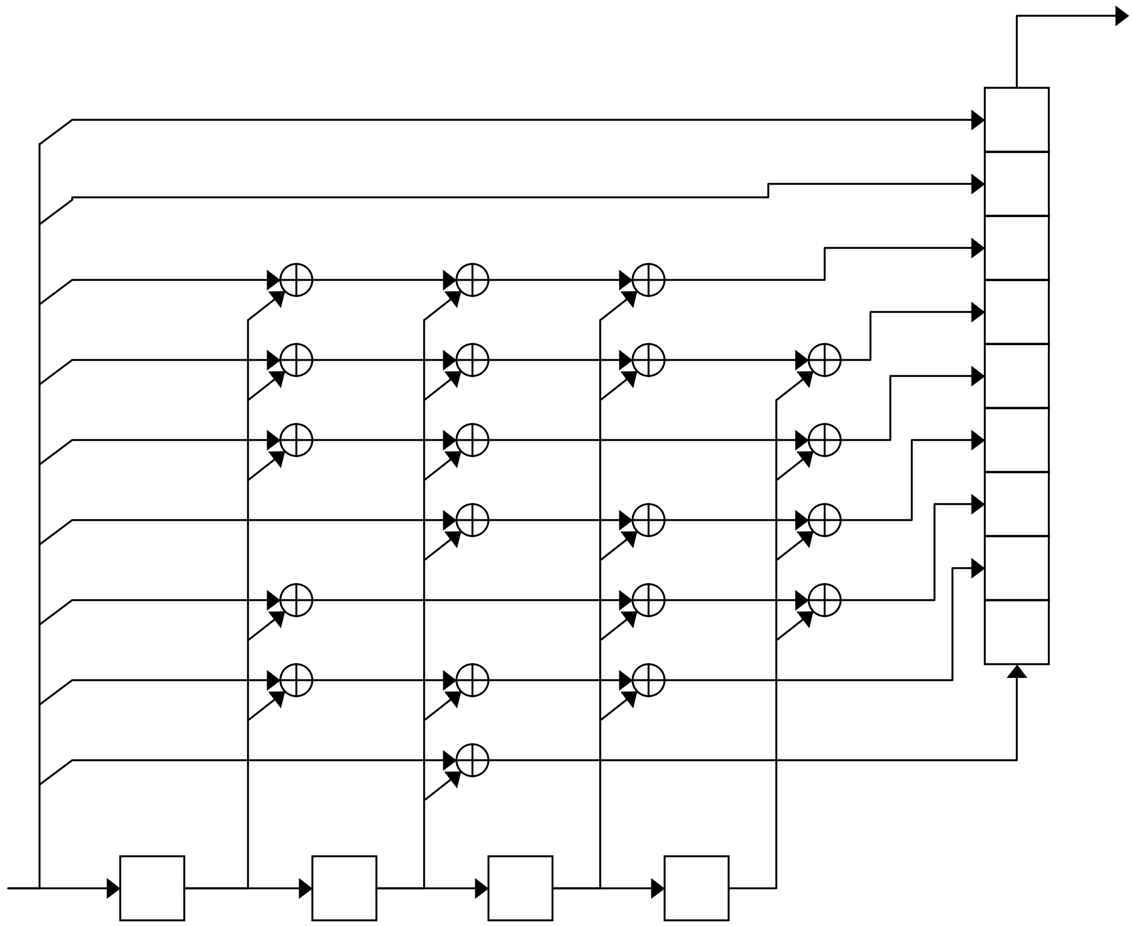

Figure 4.20 Implementation of convolutional encoder represented by equation (4.62)

Table 4.3 State table for convolutional encoder shown in Figure 4.20

Current State	Next State	Input	Output
0	0	0	000000000
0	1	1	111111111
1	2	0	001110110
1	3	1	110001001
2	4	0	001111011
2	5	1	110000100
3	6	0	000001101
3	7	1	111110010
4	8	0	001101110
4	9	1	110010001
5	10	0	000011000
5	11	1	111100111
6	12	0	000010101
6	13	1	111101010
7	14	0	001100011
7	15	1	110011100
8	0	0	000111100
8	1	1	111000011
9	2	0	001001010
9	3	1	110110101
10	4	0	001000111
10	5	1	110111000
11	6	0	000110001
11	7	1	111001110
12	8	0	001010010
12	9	1	110101101
13	10	0	000100100
13	11	1	111011011
14	12	0	000101001
14	13	1	111010110
15	14	0	001011111
15	15	1	110100000

4.3.3 Creating the Received Sequence

To decode eight bits, a received stream of length $8 \cdot R$, where R is the rate of the Viterbi decoder, needs to be generated. In this case a Viterbi encoder with a rate of $R = 9$ was created, thus requiring a received sequence of 72 bits. This received stream is now generated by applying equation (4.77), found in the example given in section 4.2.5.7 on the key sequence z.

$$r_0^{(0)} = z_0 = 0 \tag{4.82}$$

$$r_0^{(1)} = z_1 + z_{16} = 0 + 1 = 1 \tag{4.83}$$

$$r_0^{(2)} = z_4 + z_9 = 0 + 1 = 1 \tag{4.84}$$

$$r_0^{(3)} = z_5 + z_6 = 1 + 0 = 1 \tag{4.85}$$

$$r_0^{(4)} = z_7 + z_{10} = 0 + 1 = 1 \tag{4.86}$$

$$r_0^{(5)} = z_9 + z_{17} = 1 + 0 = 1 \tag{4.87}$$

$$r_0^{(6)} = z_{12} + z_{15} = 1 + 0 = 1 \tag{4.88}$$

$$r_0^{(7)} = z_{13} + z_{15} = 1 + 0 = 1 \tag{4.89}$$

$$r_0^{(8)} = z_{14} + z_{15} = 0 + 0 = 0 \tag{4.90}$$

Thus receiving the sequence

$$\bar{r}_0 = (011111110) \tag{4.91}$$

Applying the same process for $\bar{r}_1$ to r_7 gives:

$$\bar{r}_1 = (000001011) \tag{4.92}$$

$$\bar{r}_2 = (000000001) \tag{4.93}$$

$$\bar{r}_3 = (001100010) \tag{4.94}$$

$$\bar{r}_4 = (001000100) \tag{4.95}$$

$$\bar{r}_5 = (111111111) \tag{4.96}$$

$$\bar{r}_6 = (011100110) \tag{4.97}$$

$$\bar{r}_7 = (001001100) \tag{4.98}$$

4.3.4 Using the Viterbi Algorithm for a Fast Correlation Attack

The following sequence of trellis diagrams (Figure 4.21 to Figure 4.28) depicts the received sequence r being sent through the Viterbi decoder specified by equation (4.81).

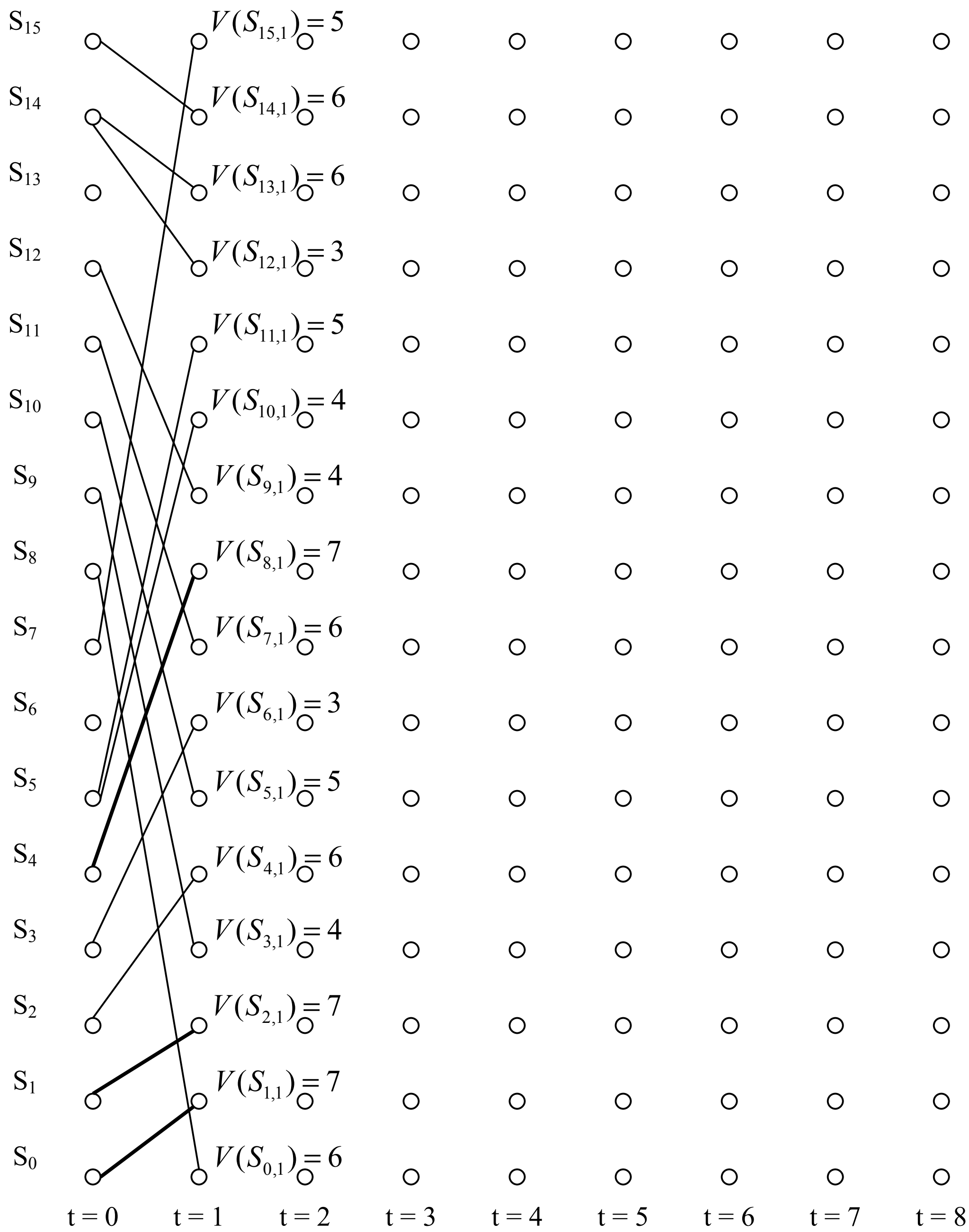

Figure 4.21 Sending r through the Viterbi decoder given by equation (4.81), $t = 1$

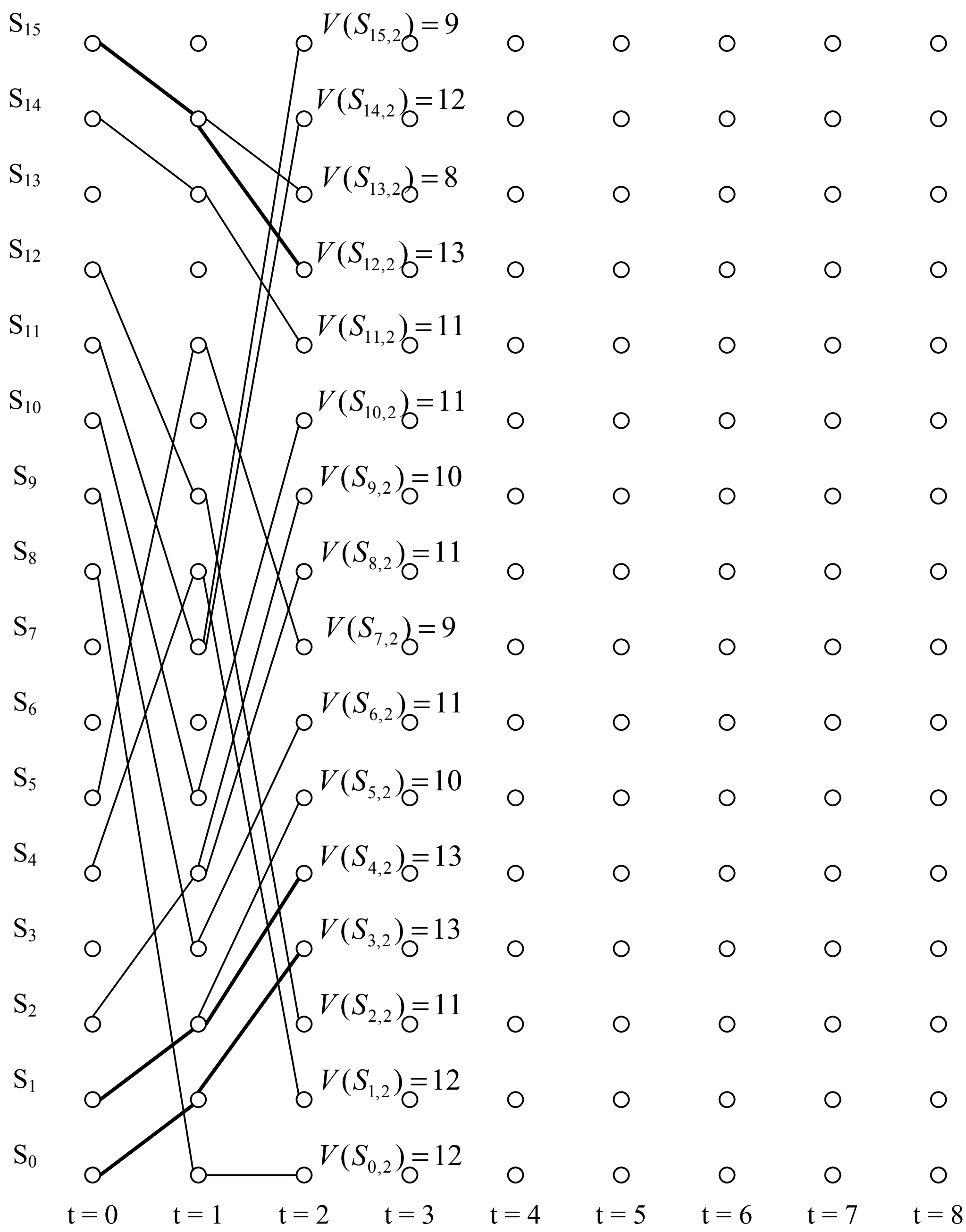

Figure 4.22 Sending r through the Viterbi decoder given by equation (4.81), $t = 2$

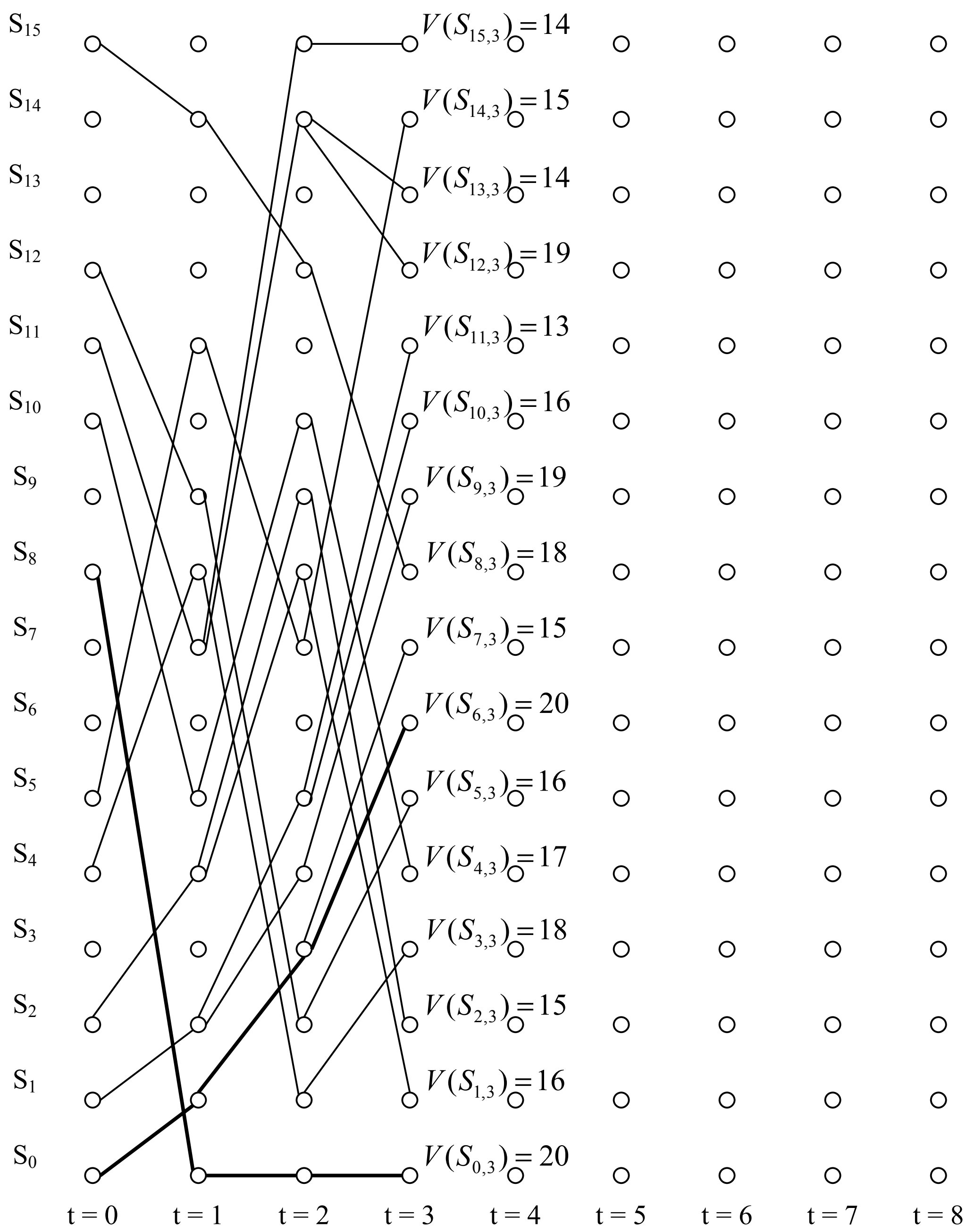

Figure 4.23 Sending r through the Viterbi decoder given by equation (4.81), $t=3$

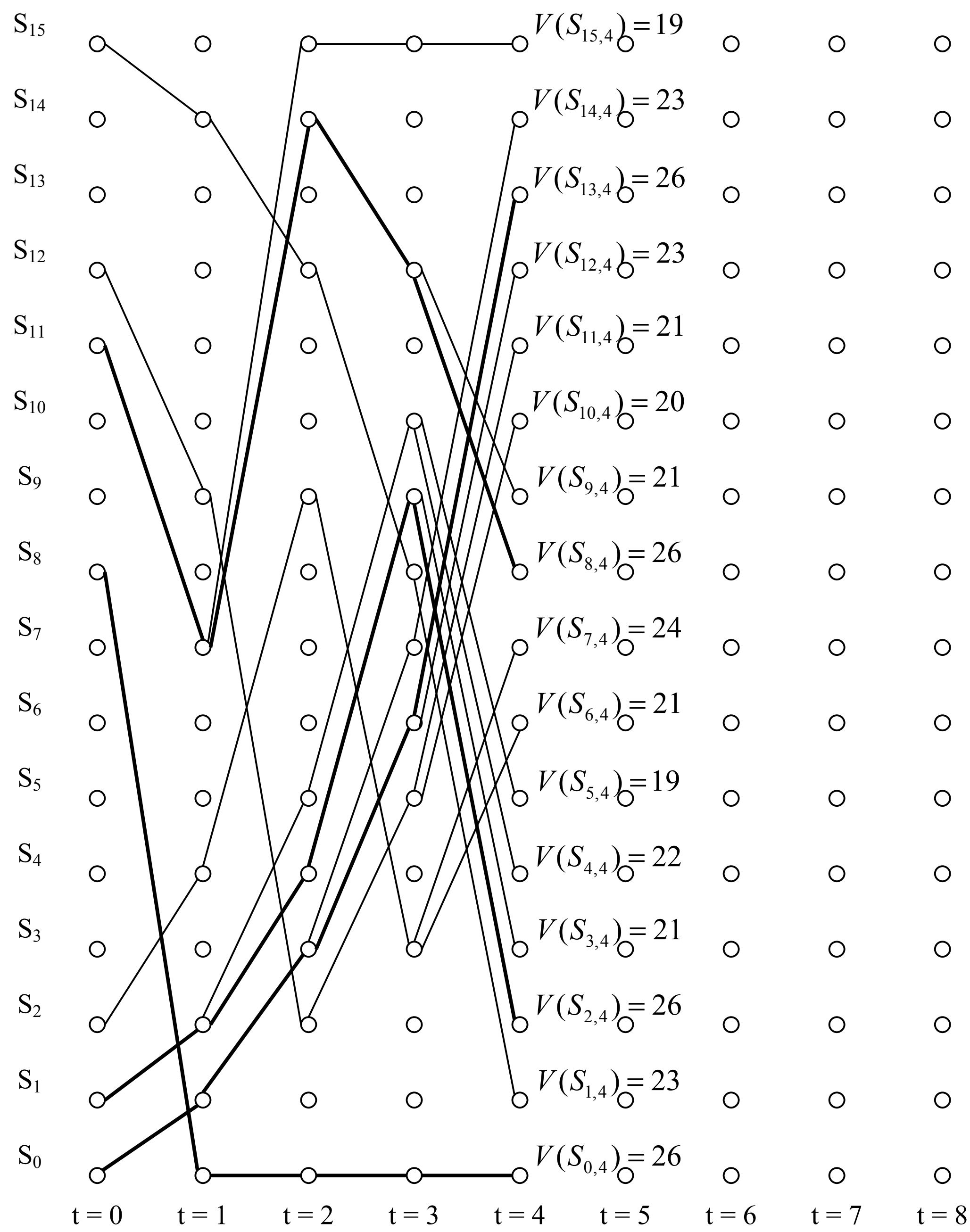

Figure 4.24 Sending r through the Viterbi decoder given by equation (4.81), $t = 4$

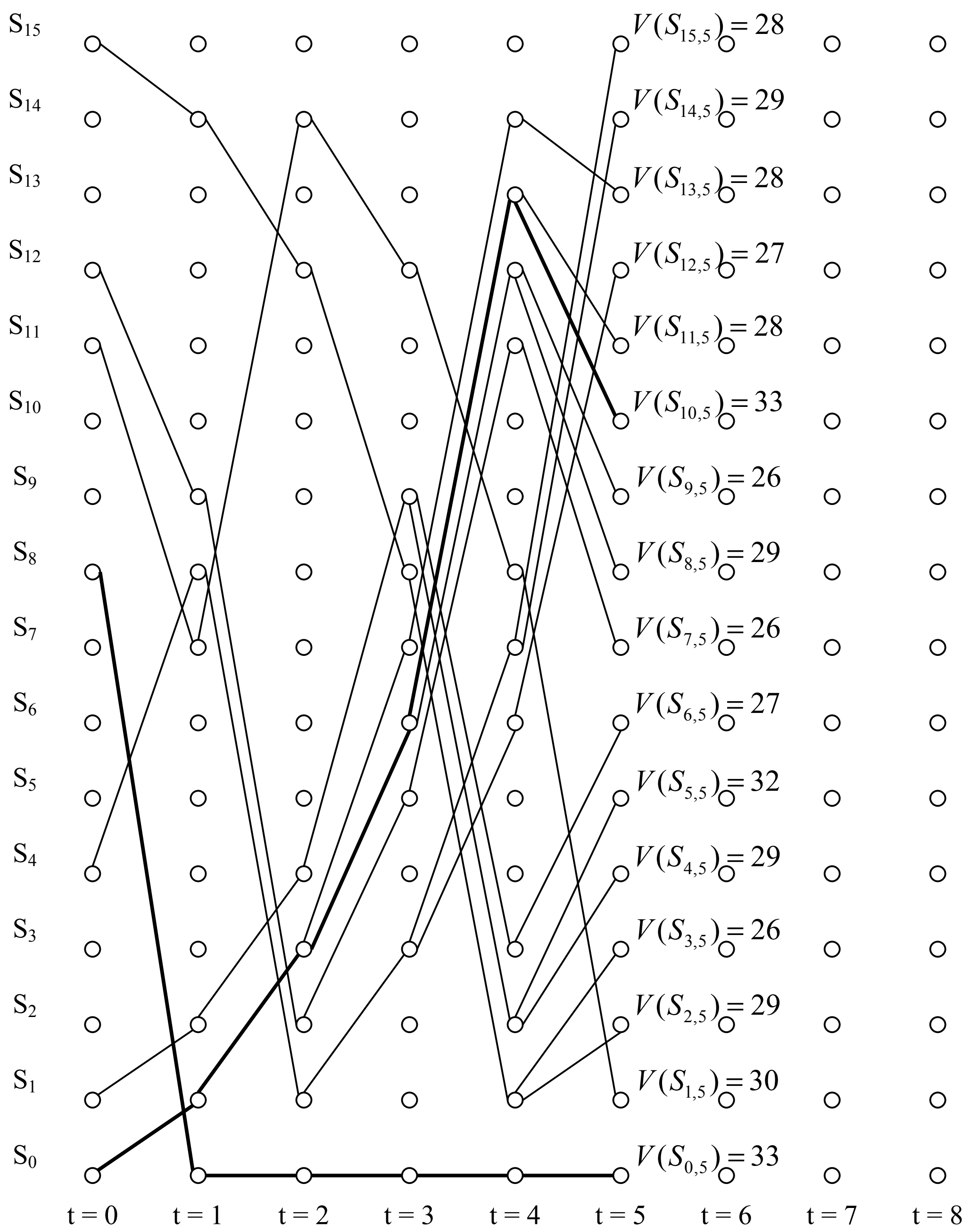

Figure 4.25 Sending r through the Viterbi decoder given by equation (4.81), $t = 5$

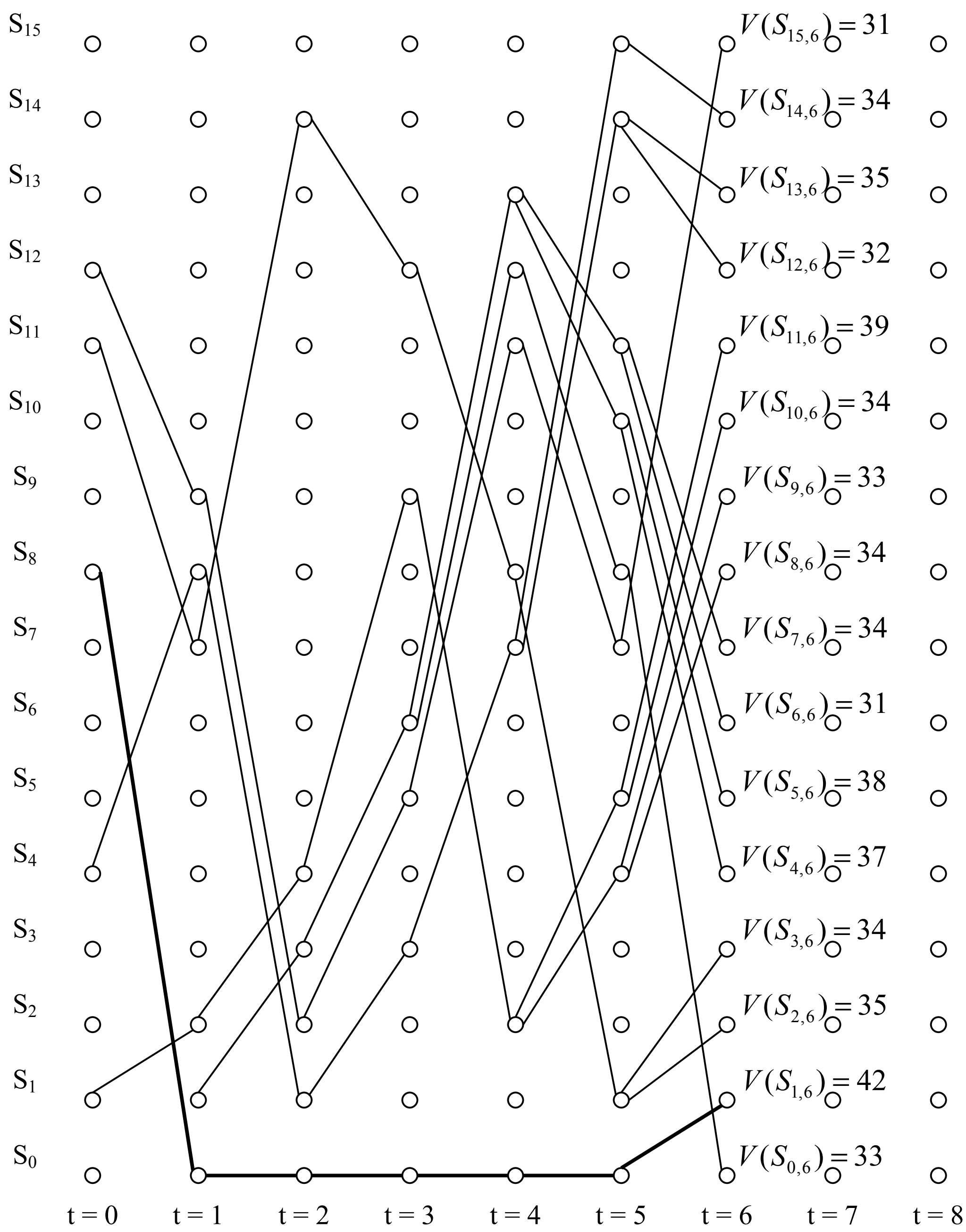

Figure 4.26 Sending r through the Viterbi decoder given by equation (4.81), $t = 6$

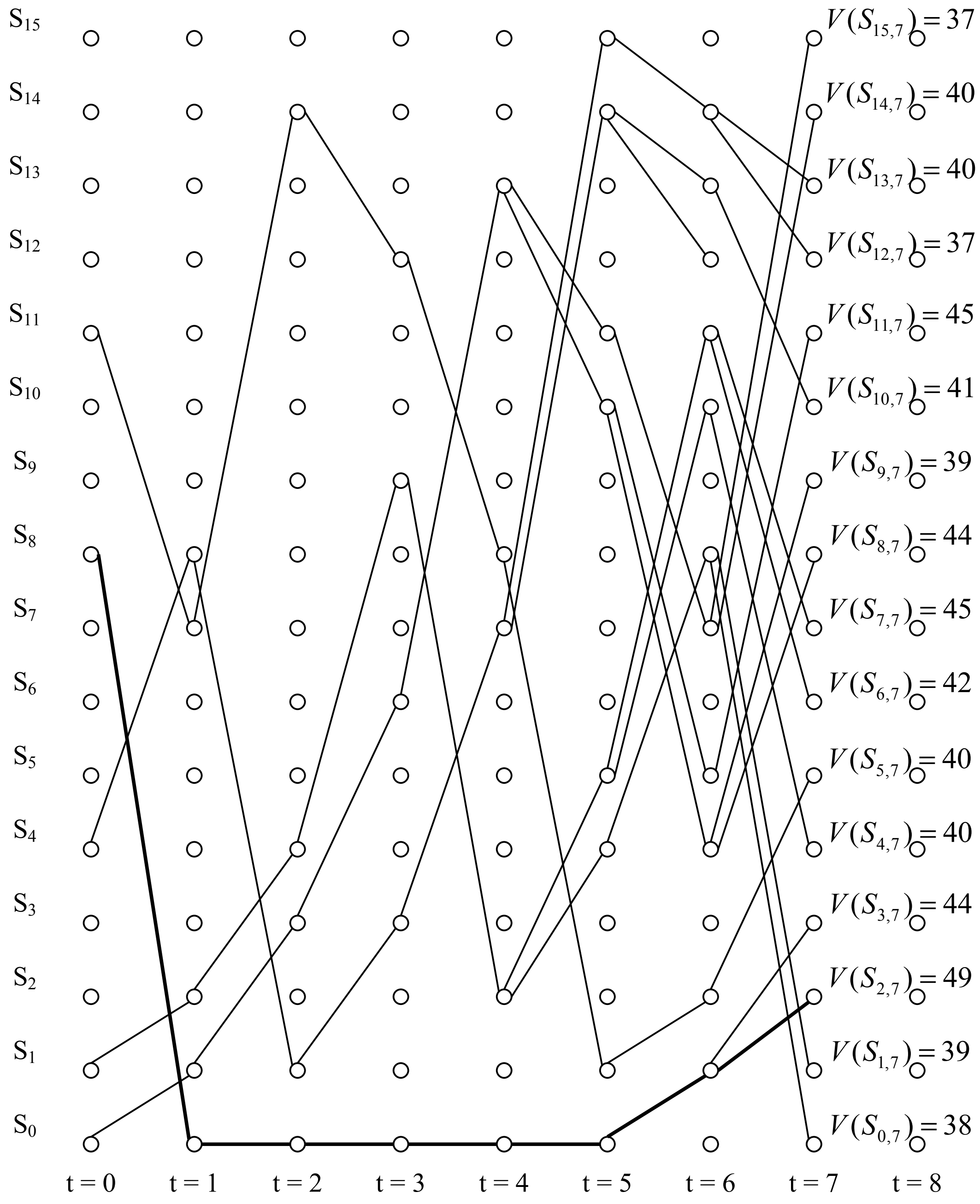

Figure 4.27 Sending r through the Viterbi decoder given by equation (4.81), $t = 7$

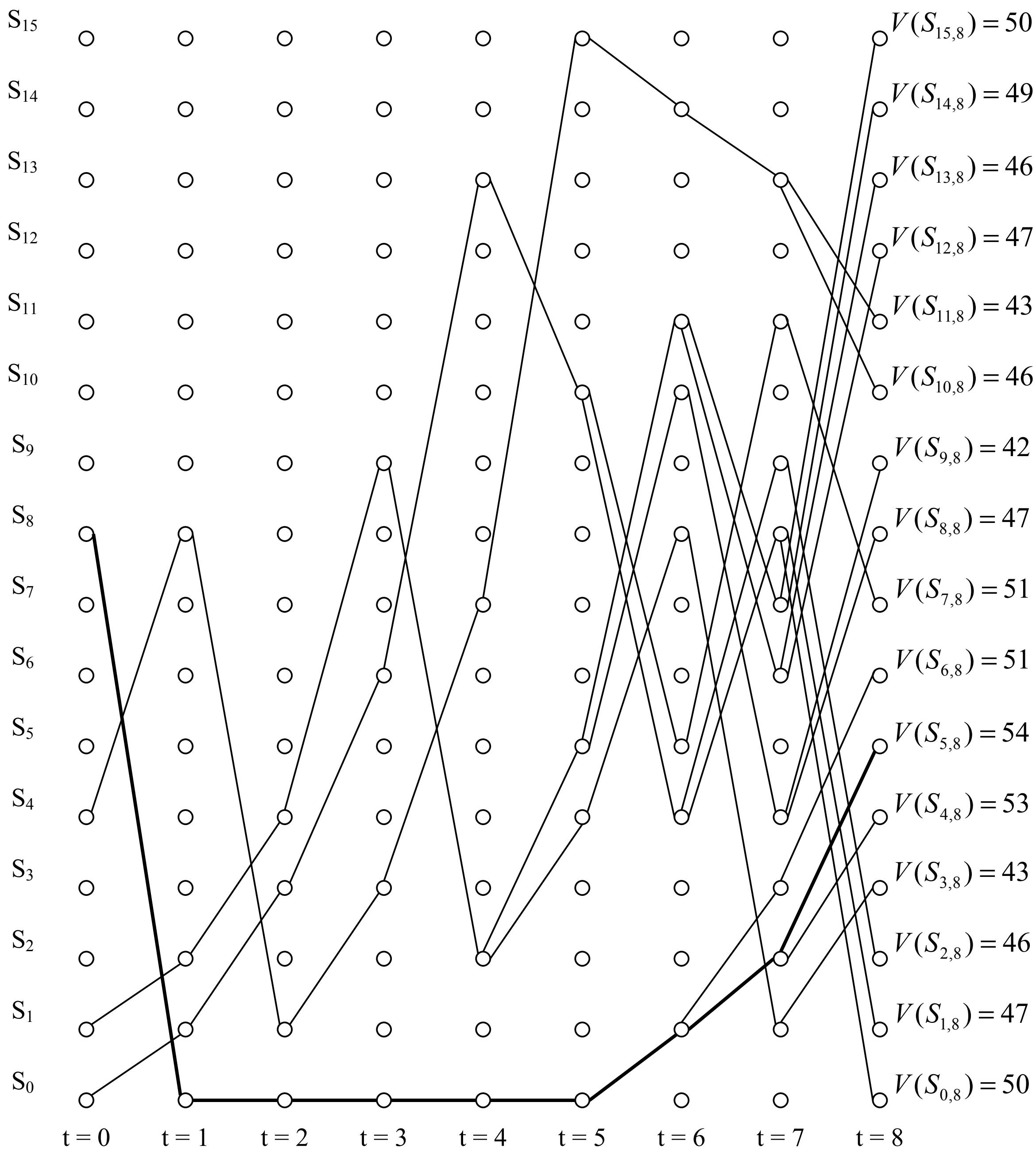

Figure 4.28 Sending r through the Viterbi decoder given by equation (4.81), $t = 8$

The path with the largest metric is used to determine the estimated received sequence, which in this case is the path entering node $S_{5,8}$ with a metric of $V(S_{5,8}) = 54$. Using the Table 4.3 determines the estimated initial state of the LFSR in Table 4.4 below.

Table 4.4 Determining the estimated LFSR initial condition from Figure 4.28 and Table 4.3

State Change	Estimated Transmitted Sequence	Estimated Input
$S_8 \rightarrow S_0$	$\bar{r}_0' = (000111100)$	$a_0 = 0$
$S_0 \rightarrow S_0$	$\bar{r}_1' = (000000000)$	$a_1 = 0$
$S_0 \rightarrow S_0$	$\bar{r}_2' = (000000000)$	$a_2 = 0$
$S_0 \rightarrow S_0$	$\bar{r}_3' = (000000000)$	$a_3 = 0$
$S_0 \rightarrow S_0$	$\bar{r}_4' = (000000000)$	$a_4 = 0$
$S_0 \rightarrow S_1$	$\bar{r}_5' = (111111111)$	$a_5 = 1$
$S_1 \rightarrow S_2$	$\bar{r}_6' = (001110110)$	$a_6 = 0$
$S_2 \rightarrow S_5$	$\bar{r}_7' = (110000100)$	$a_7 = 1$

Looking at Table 4.4 it is seen that the input $a = (00000101)$ to the convolutional encoder, which produced the estimated transmitted sequence r' matches the original initial condition $IC = (00000101)$ of the LFSR used in 4.3.1. The correct initial condition of the LFSR has thus been successfully found.

The strength of the Viterbi algorithm can be seen in the fact that during the first few time intervals ($t = 0$ to $t = 5$) the correct path is not yet identifiable. It takes time for each path to accumulate enough of a history to be able to find the correct one. This initial phase of the algorithm is the most vulnerable to failure as it is at this stage that an incorrect path is most likely to have a higher metric when intersecting the correct path.

4.4 Simulation Results and Discussion

4.4.1 Summary of Topics to be Investigated Using Simulations

(1) Investigate the relationship between B (Viterbi decoder size), N (available ciphertext bits) and p (BSC error probability) as well as the number of parity equations needed to find the correct initial condition

(2) Investigate whether the number of equations required for breaking a cipher system with a certain BSC error probability is constant or also a function of B.

4.4.2 Approach

Although this fast correlation attack was used successfully on LFSRs in excess of 40 bits, the memory requirements are much larger than for smaller LFSRs as the number of bits required to find sufficient parity equations is greater for larger LFSRs and it takes longer to find them (see 4.2.3.3). Because of this it was decided to use a LFSR of size 19 with the polynomial as shown below for the investigation.

$$g(x) = x^{19} + x^{17} + x^{14} + x^{12} + x^{10} + x^{9} + x^{7} + x^{6} + x^{5} + x^{4} + 1 \qquad (4.99)$$

The recurrence relation for $g(x)$ is given as follows:

$$a_n = a_{n-19} + a_{n-15} + a_{n-14} + a_{n-13} + a_{n-12} + a_{n-10} + a_{n-9} + a_{n-7} + a_{n-5} + a_{n-2} \qquad (4.100)$$

Although a LFSR of this size can still be broken by an exhaustive search it however allows one to easily investigate convolutional encoder sizes of $B = 2$ up to $B = 11$ which provides an adequate range to draw conclusions from. If a larger size LFSR is chosen it becomes difficult to find a sufficient number of parity equations (as described in section 4.2.3.3) when using small values of B.

A Viterbi decoder is created for the LFSR. A pn-sequence $\bar{a}$ is generated using a random initial value for the LFSR. This pn-sequence is now corrupted sending it through a BSC of probability p creating the ciphertext-stream $\bar{z}$ from which the Viterbi decoder determines an estimated initial value. This estimation is compared with the actual initial value of the LFSR. The random sequence $\bar{a}$ is generated 15 times using different random seeding values. If the estimated initial value matches the actual initial value 80% of the time (thus allowing 3 incorrect estimates of the initial condition) it is assumed the LFSR has been broken for that specific BSC probability p.

The graphs in the following section indicate the minimum number of equations that are necessary to find the correct initial condition for at least 80% of the time for a specific BSC probability p and convolutional encoder size B.

4.4.3 Results

The results have been divided into two sections, largely due to the huge difference in the number of parity equations required for breaking a system with a BSC probability below $p = 0.47$ and above $p = 0.47$ and the impact this has on the memory requirements for performing these simulations.

As the parameter p increases towards 0.5 the number of ciphertext bits and parity equations required to succeed explode exponentially. Because of this two parameters are used; n originally introduced in equation (4.58), repeated below for convenience

$$n = \log_2 N \tag{4.101}$$

as well as introducing γ defined below in equation (4.102)

$$\gamma = \log_2 \Gamma \tag{4.102}$$

where Γ is the number of parity equations.

4.4.3.1 Results for Systems with BSC below $p = 0.47$

4.4.3.1.1 The number of bits required for finding the correct initial condition

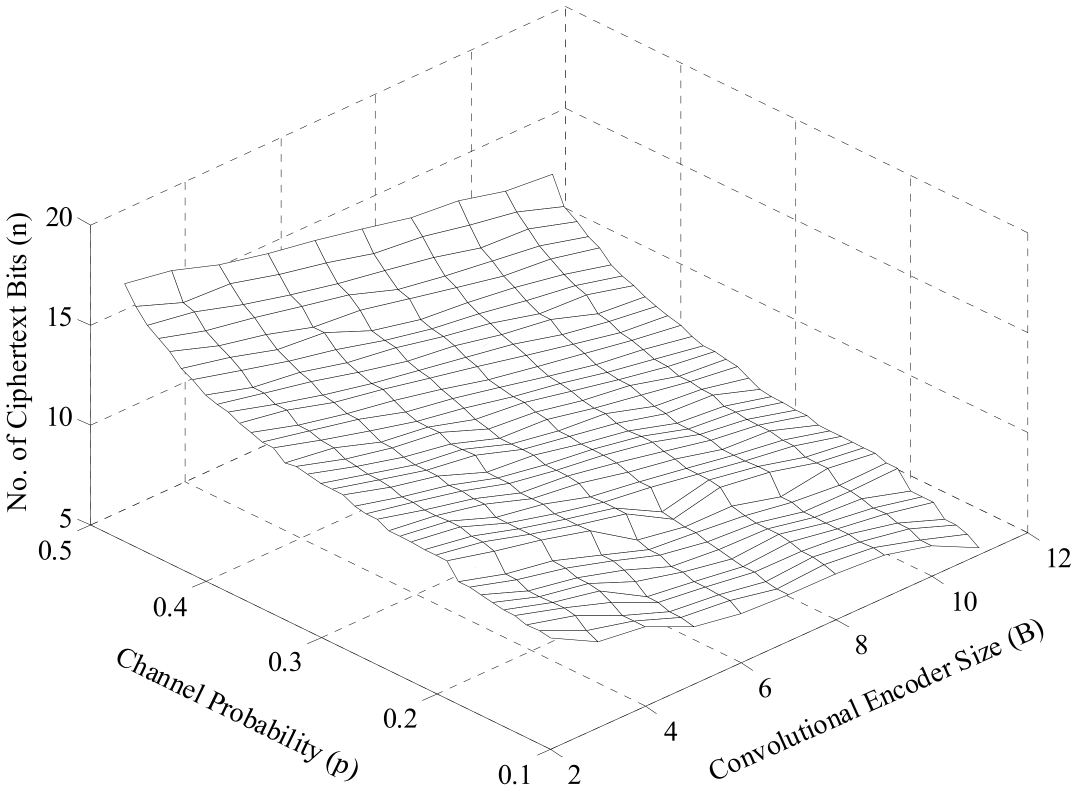

Figure 4.29 No. of ciphertext bits required for a successful attack

The number of ciphertext bits, *N*, (where $n = \log_2 N$), required to find the correct initial condition in Figure 4.29 is given in logarithmic form as this relationship grows exponentially as a function of *B* and *p*. Exact values used for generating this graph can be found in Table 4.5.

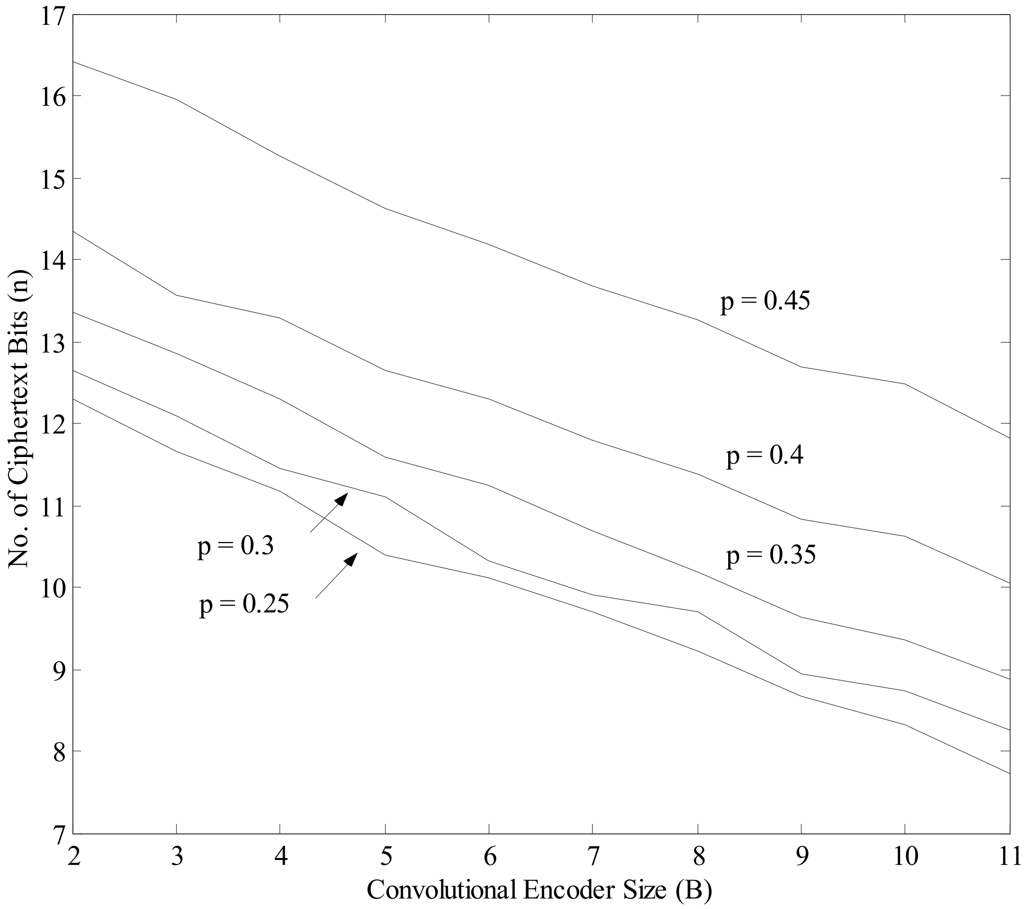

Figure 4.30 No of ciphertext bits (n) required to succeed for selected values of p

Figure 4.30 gives a two-dimensional representation using data from Figure 4.29 of the number of ciphertext bits required, *N*, (where $n = \log_2 N$), as a function of *B* for selected constant values of *p*. The number of bits required to find the initial condition falls exponentially as *B* increases. Furthermore it can also be clearly seen that the amount of ciphertext required to break the system increases dramatically as *p* increases.

Table 4.5 *No. of ciphertext bits required for finding the correct initial condition ($p < .47$)*

p \ B	2	3	4	5	6	7	8	9	10	11
0.1	1491	657	493	242	182	159	159	139	93	42
0.11	1643	796	493	266	231	166	182	152	93	54
0.12	1643	1062	517	279	231	174	191	152	111	60
0.13	1901	1115	657	306	254	200	191	159	116	60
0.14	1901	1115	796	306	254	200	220	166	127	89
0.15	1996	1170	876	448	279	210	231	166	133	89
0.16	2199	1170	876	517	279	242	231	166	152	93
0.17	2308	1420	964	597	353	254	242	166	159	166
0.18	2308	1420	1062	597	353	292	254	182	159	166
0.19	3755	2544	1811	1170	407	292	279	191	242	191
0.2	3755	2944	1811	1170	796	321	448	353	242	191
0.21	3755	2944	1811	1170	835	337	470	353	242	191
0.22	4139	3245	1811	1170	835	657	470	353	292	191
0.23	4139	3245	1811	1353	835	657	470	353	292	191
0.24	4139	3245	2308	1353	1062	759	569	353	321	191
0.25	5029	3245	2308	1353	1115	835	597	407	321	210
0.26	5029	3577	2544	1491	1115	919	657	407	321	220
0.27	5280	3577	2544	1565	1115	919	657	427	353	231
0.28	6417	3942	2804	1565	1289	919	657	427	353	231
0.29	6417	4345	2804	2199	1289	964	723	470	427	242
0.3	6417	4345	2804	2199	1289	964	835	493	427	306
0.31	7426	4562	3091	2308	1289	1170	835	626	470	370
0.32	7797	5029	3407	2671	1491	1228	876	723	470	388
0.33	7797	6112	3407	2671	1643	1420	1012	723	517	427
0.34	10445	6417	4139	2804	2095	1420	1062	759	569	427
0.35	10445	7426	5029	3091	2423	1643	1170	796	657	470
0.36	12090	7426	5280	3755	2671	1811	1289	964	723	569
0.37	13994	9948	6112	4562	3245	2308	1565	1115	796	657
0.38	16198	9948	6737	4790	3407	2671	1901	1289	919	759
0.39	17857	12090	8595	5544	3942	3091	2199	1725	1012	876
0.4	20670	12090	9948	6417	5029	3577	2671	1811	1565	1062
0.41	27602	18654	12358	8568	5406	3845	3086	2247	1808	1226
0.42	34289	23172	14280	12298	6246	4442	3832	2999	2244	1415
0.43	49223	28785	20499	12298	10358	7365	5115	3723	2994	2181
0.44	65734	41320	29425	20398	14867	8510	6353	4970	3996	2707
0.45	87784	63766	39294	25338	18468	13131	9801	6634	5735	3614
0.46	135473	73689	65188	42032	30635	20260	16258	11826	8849	5186
0.47	241543	175503	100601	74957	54632	38837	26969	22667	14676	12346

4.4.3.1.2 The number of equations required for finding the correct initial condition

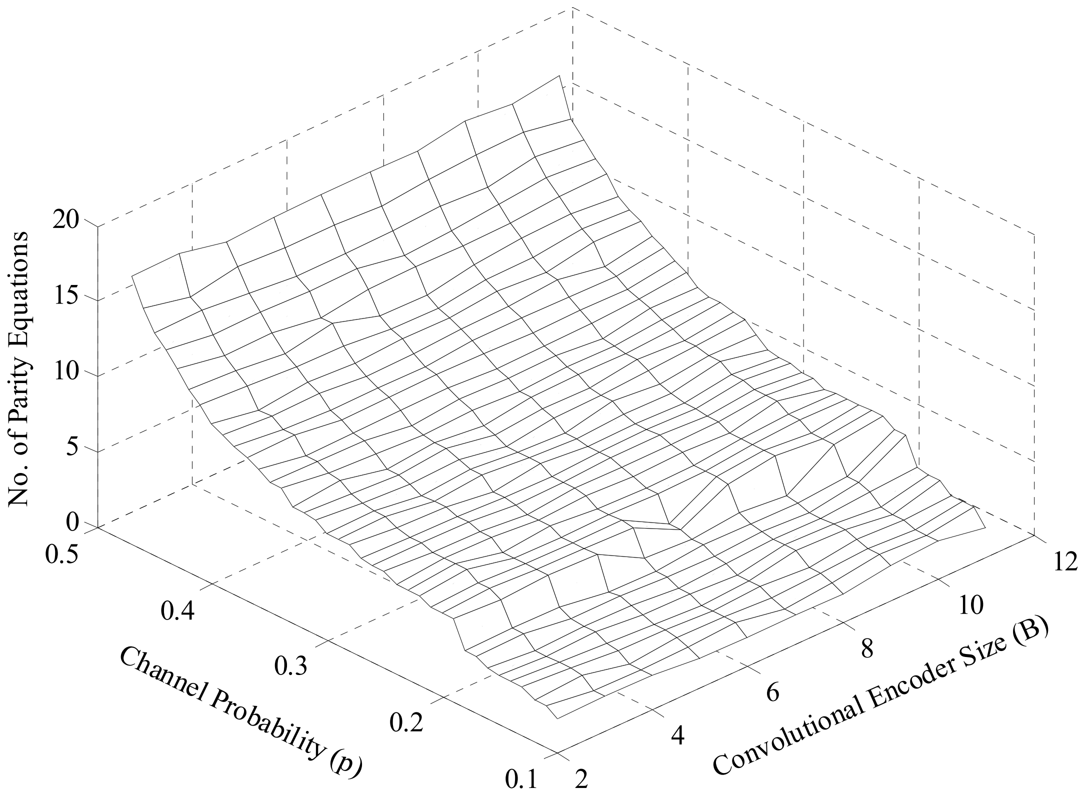

Figure 4.31 No. of parity equations (γ) required for a successful attack

Figure 4.31 shows the relationship between the number of parity equations, *γ*, (where $\gamma = \log_2 \Gamma$), the convolutional encoder size, *B,* and the BSC error probability *p*. The largest error probability of *p*=0.47 shown in this graph, thus requires a convolutional encoder with a rate of up to 296258 for success. Exact values used for generating this graph can be found in Table 4.6.

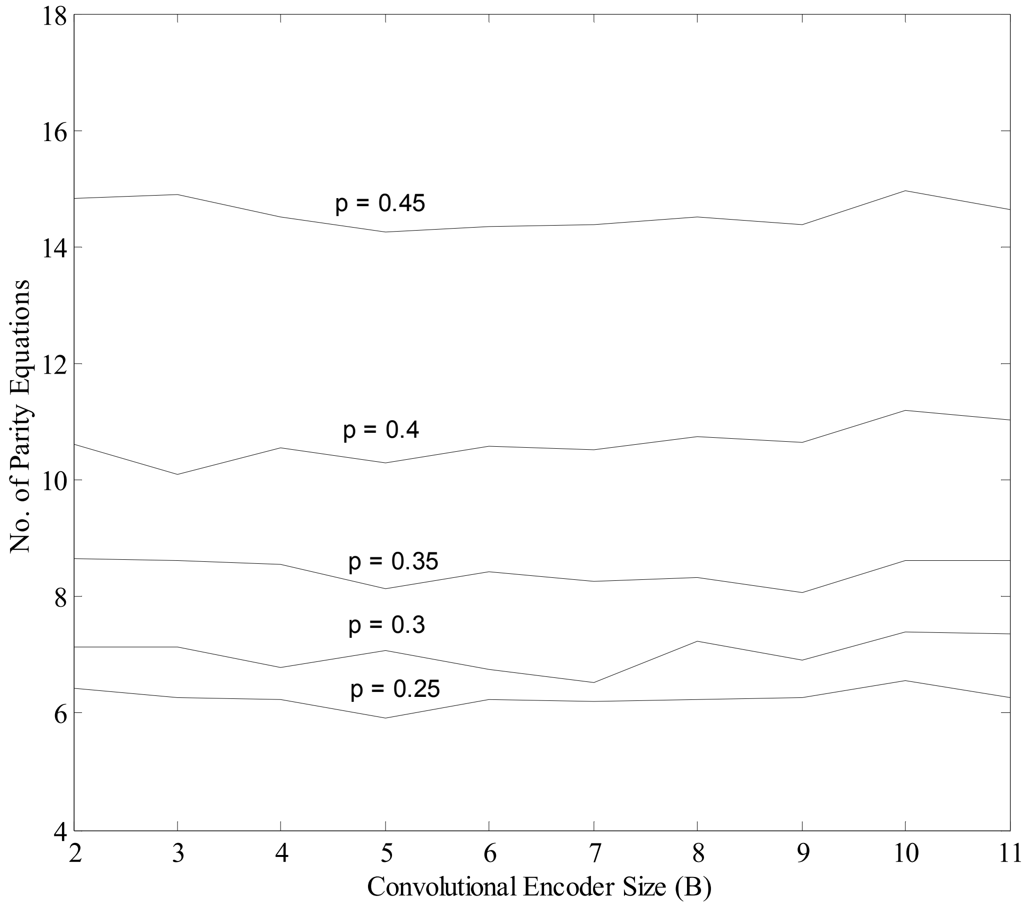

Figure 4.32 No. of parity equations (γ) required to succeed for selected values of p

Figure 4.32 is a two-dimensional representation for selected constant values of p based on data from Figure 4.31. The number of parity equations, γ, (where $\gamma = \log_2 \Gamma$) are shown as a function of B for selected constant values of p. The maximum value of $p = 0.45$ is the graph situated at the top of the figure, while the minimum value of $p = 0.25$ is the bottommost graph. This result is to be expected, as fewer parity equations are required to reconstruct a sequence that has been less corrupted by passing through a BSC noise channel with a lower error probability. The number of parity equations required to find the correct initial condition for a certain BSC value p decreases only slightly as the convolutional encoder size increases. This is in contrast to Figure 4.30 where the amount of ciphertext bits required for success decreases exponentially as the convolutional encoder size B increases.

Table 4.6 No. of parity equations required for finding the correct initial condition ($p < 0.47$)

p \ B	2	3	4	5	6	7	8	9	10	11
0.1	5	5	5	4	4	4	4	5	6	4
0.11	7	5	5	5	6	4	5	7	6	8
0.12	7	8	5	6	6	5	6	7	6	8
0.13	10	9	7	7	6	7	6	8	7	8
0.14	10	9	8	7	6	7	10	10	7	12
0.15	10	9	10	8	8	7	10	10	8	12
0.16	11	9	10	10	8	9	10	10	12	14
0.17	14	15	11	11	10	10	11	10	14	45
0.18	14	15	14	11	10	13	12	12	14	45
0.19	47	43	47	46	14	13	15	13	47	63
0.2	47	57	47	46	43	15	44	62	47	63
0.21	47	57	47	46	47	15	47	62	47	63
0.22	62	77	47	46	47	46	47	62	70	63
0.23	62	77	47	60	47	46	47	62	70	63
0.24	62	77	75	60	68	59	70	62	94	63
0.25	87	77	75	60	75	74	75	77	94	77
0.26	87	92	91	73	75	89	92	77	94	82
0.27	94	92	91	78	75	89	92	87	115	95
0.28	142	112	111	78	108	89	92	87	115	95
0.29	142	140	111	135	108	93	111	105	169	106
0.3	142	140	111	135	108	93	149	120	169	165
0.31	196	154	141	154	108	149	149	175	206	249
0.32	222	188	171	201	141	173	169	234	206	263
0.33	222	268	171	201	171	230	228	234	256	325
0.34	399	297	252	220	264	230	255	252	303	325
0.35	399	397	376	279	343	308	322	271	394	395
0.36	555	397	414	420	418	375	406	426	473	589
0.37	743	738	557	608	618	605	585	586	563	799
0.38	991	738	675	679	683	813	851	805	769	1084
0.39	1193	1099	1093	921	924	1081	1155	1443	940	1432
0.4	1583	1099	1496	1250	1537	1477	1722	1593	2358	2117
0.41	2869	2621	2303	2235	1769	1742	2297	2455	3134	2881
0.42	4388	4082	3113	4615	2393	2340	3555	4342	4861	3874
0.43	9125	6286	6357	4615	6590	6579	6330	6755	8662	9119
0.44	16338	12828	13157	12651	13527	8829	9782	11944	15486	13991
0.45	29248	30756	23244	19496	20837	21099	23464	21302	31627	25215
0.46	70007	41250	64544	53671	57275	50156	64551	68266	76048	51895
0.47	222970	235261	154290	170993	181866	183936	177682	250904	209742	296258

4.4.3.2 Results for Systems with BSC above $p = 0.47$

Due to the fact that the memory requirements for the state table used to implement the Viterbi decoder is directly proportional to the number of equations and exponentially proportional to B as in the relation shown below

$$M_{Memory} \propto 2^{B} \cdot 2 \cdot \Gamma \tag{4.103}$$

the results for finding the correct initial condition for BSC probabilities in excess of $p = 0.47$ these results are only given for $2 \leq B \leq 7$.

4.4.3.2.1 The number of bits required for finding the correct initial condition

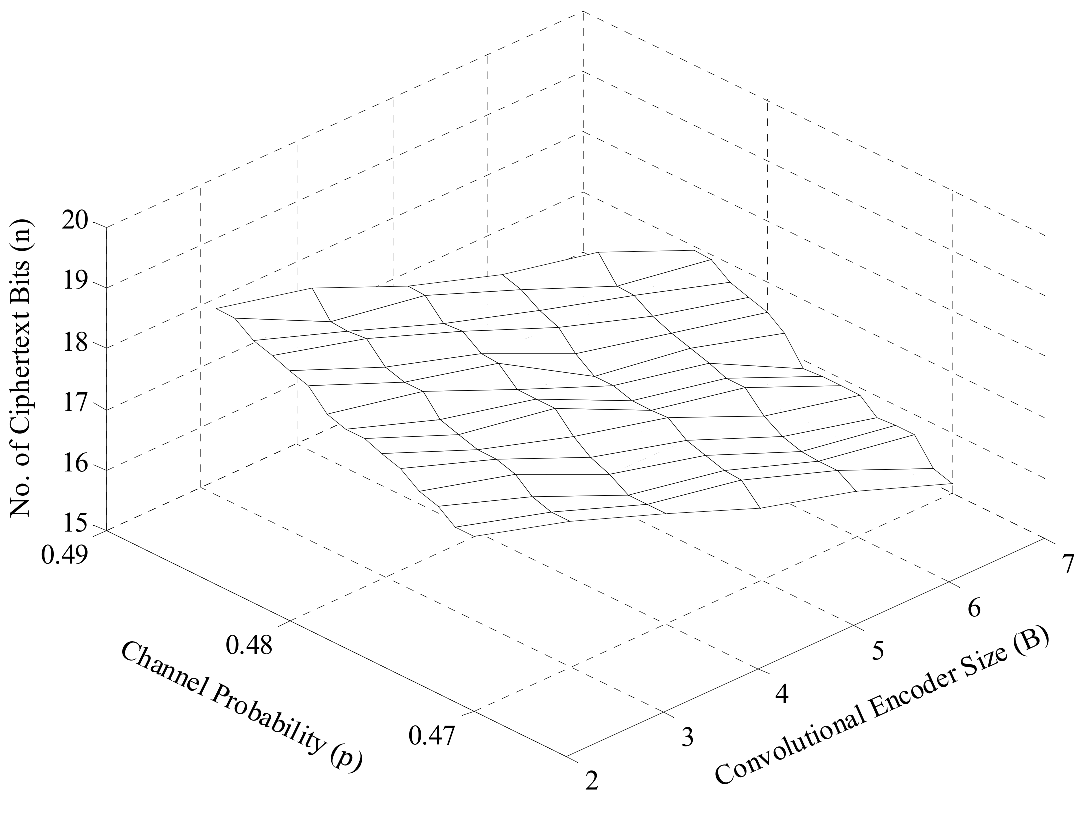

Figure 4.33 No. of ciphertext bits (n)required for a successful attack

Figure 4.33 presents the relationship between the number of bits, N, (where $n = \log_2 N$), required to find the correct initial condition in Figure 4.33 as a function of $2 \leq B \leq 7$ and $0.47 \leq p \leq 0.484$. Exact values used for generating this graph can be found in Table 4.7.

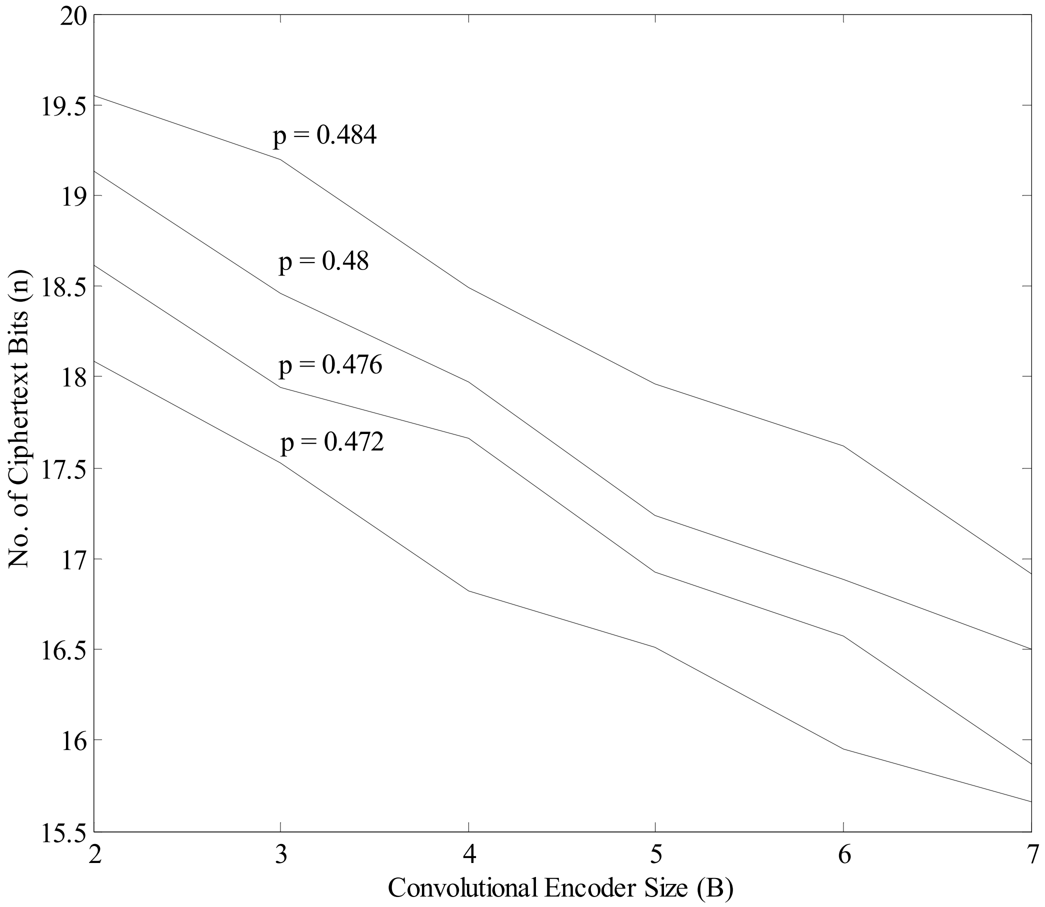

Figure 4.34 No of ciphertext bits (n) required to succeed for selected values of p

Figure 4.34 shows the number of ciphertext bits, *N*, (where $n = \log_2 N$), as a function of *B* for selected constant values of *p*, based on the data also used for Figure 4.33. As can be expected, the higher the BSC probability *p* becomes, the more ciphertext bits are required for success. The amount of ciphertext bits required falls exponentially as *B* increases.

Table 4.7 No. of ciphertext bits required for finding the correct initial condition ($p > 0.47$)

p \ B	2	3	4	5	6	7
0.47	241543	175503	116256	74957	54632	36128
0.471	241543	175503	116256	93117	63133	38837
0.472	279132	188665	116256	93117	63133	51864
0.473	300066	218025	134348	100100	63133	51864
0.474	346762	234376	144424	107607	67867	51864

p \ B	2	3	4	5	6	7
0.475	372769	234376	166899	124352	78428	59934
0.476	400726	251954	207337	124352	97430	59934
0.477	400726	270850	207337	124352	97430	59934
0.478	430780	336475	207337	133678	104737	59934
0.479	535155	336475	257573	133678	112592	80039
0.48	575291	361710	257573	154480	121036	92494
0.481	618437	449350	297656	191909	130113	99431
0.482	664819	449350	297656	206302	150361	106888
0.483	768281	449350	369776	238407	150361	123521
0.484	768281	600090	369776	256287	200801	123521
0.485						153448

4.4.3.2.2 The number of equations required for finding the correct initial condition

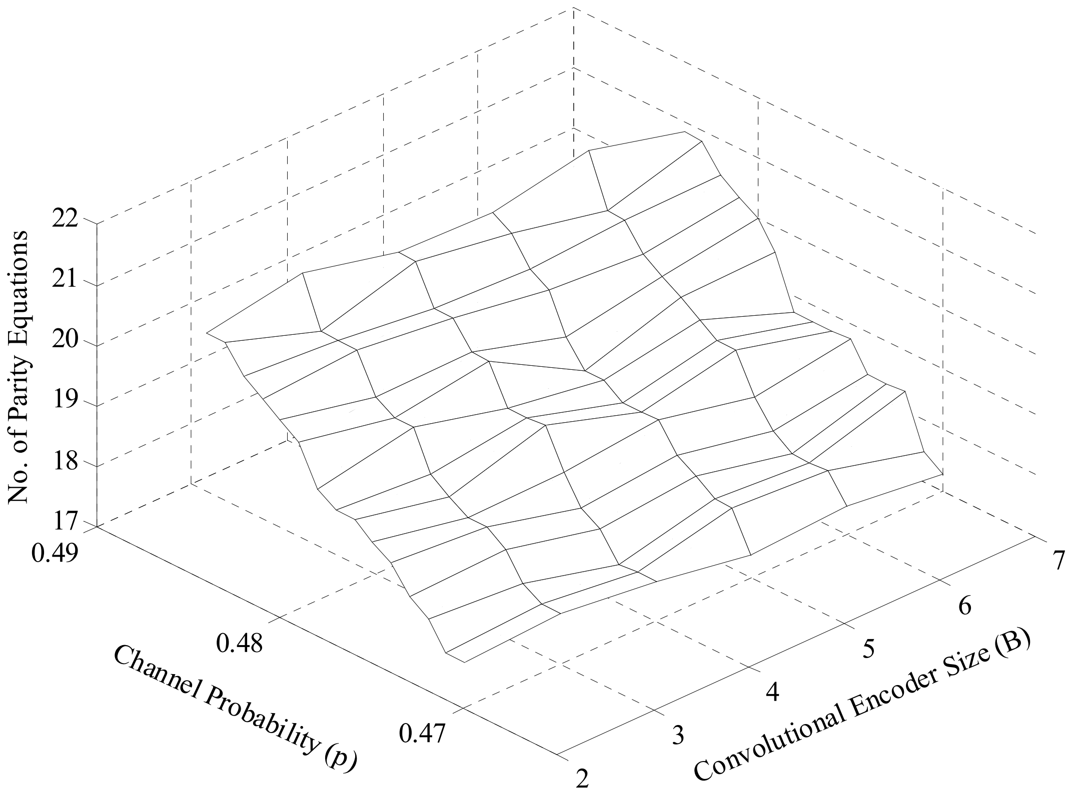

Figure 4.35 No. of parity equations (γ) required for a successful attack

Figure 4.35 represents the number of parity equations, γ, (where $\gamma = \log_2 \Gamma$) required for finding the correct initial condition as a function of the BSC noise channel error probability, *p*, as well as the chosen convolutional encoder size B. The number of parity equations required grows exponentially as a function of B and p. Exact values used for generating this graph can be found in Table 4.8.

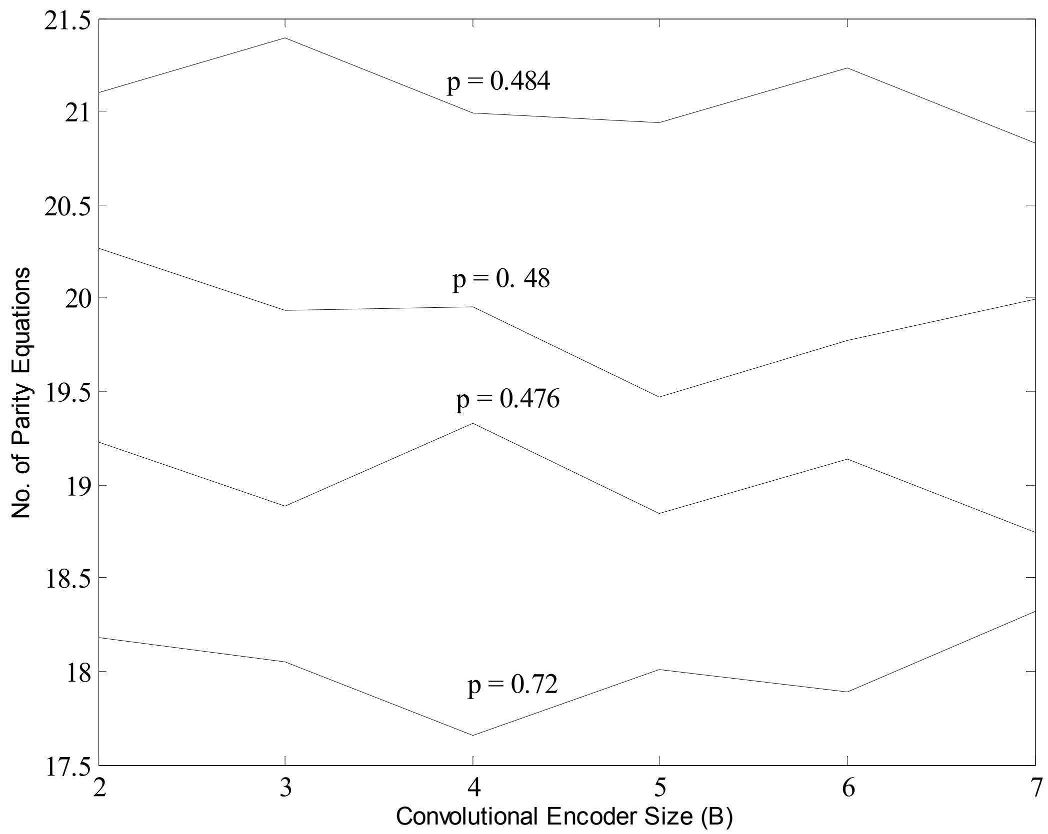

Figure 4.36 No. of parity equations (γ) required to succeed for constant values of p

Figure 4.36 gives a two-dimensional presentation of the number parity equations, γ, (where $\gamma = \log_2 \Gamma$) as a function of B for various constant values of p. Exact values used for generating this graph can be found in Table 4.8. The maximum value of $p = 0.484$ is the line situated at the top of the graph, while the minimum value of $p = 0.47$ is the bottommost line in the graph. In contrast to Figure 4.34 the number of parity equations required to find the correct initial condition for a certain BSC value p does not decrease exponentially as the convolutional encoders size B increases. However, looking at the values contained in Table 4.8, it can be seen that increasing B from 2 to 7 can lower the amount of parity equations required with up to 30%.

Table 4.8 No. of parity equations required for finding the correct initial condition ($p > .47$)

p \ B	2	3	4	5	6	7
0.47	222970	235261	206036	170993	181866	159184
0.471	222970	235261	206036	264045	242813	183936
0.472	297406	272053	206036	264045	242813	328299
0.473	343589	363362	275387	305383	242813	328299
0.474	458711	419547	318088	353095	280347	328299
0.475	530267	419547	425320	472048	374349	438298
0.476	612724	484946	657298	472048	578345	438298
0.477	612724	560081	657298	472048	578345	438298
0.478	708049	863900	657298	545572	668597	438298
0.479	1092476	863900	1013556	545572	773096	781389
0.48	1262373	998125	1013556	728131	893819	1043558
0.481	1458833	1540588	1352677	1125075	1033094	1205921
0.482	1685983	1540588	1352677	1300193	1379039	1393708
0.483	2252082	1540588	2086818	1736191	1379039	1862100
0.484	2252082	2747188	2086818	2006344	2462071	1862100
0.485						2873370

4.4.4 Discussion

It was found that the amount of ciphertext required for finding the correct initial condition drops exponentially as the convolutional encoder size increases, as predicted by equation (4.56). Furthermore, it was also found that the number of parity equations required to find the correct initial condition decreases slightly as the convolutional encoder size B increases, although no definitive figures on the actual improvement can be derived from the data obtained. The slight improvement in performance when using a larger convolutional encoder is to be expected as larger convolutional encoders can correct longer error bursts (as the BSC error probability p increases, the likelihood of more successive bits being corrupted also increases) than smaller convolutional encoders. Thus the maximum number of parity equations required for a successful attack, occurs when the smallest convolutional encoder used, i.e. $B = 2$. This is the worst case and as long as the number of parity equations found for a certain BSC error probability p is equal to this worst case, the system can be broken. The relationship showing the maximum number of parity equations required for success is presented in Figure 4.37. A table of these values is presented in Appendix E.

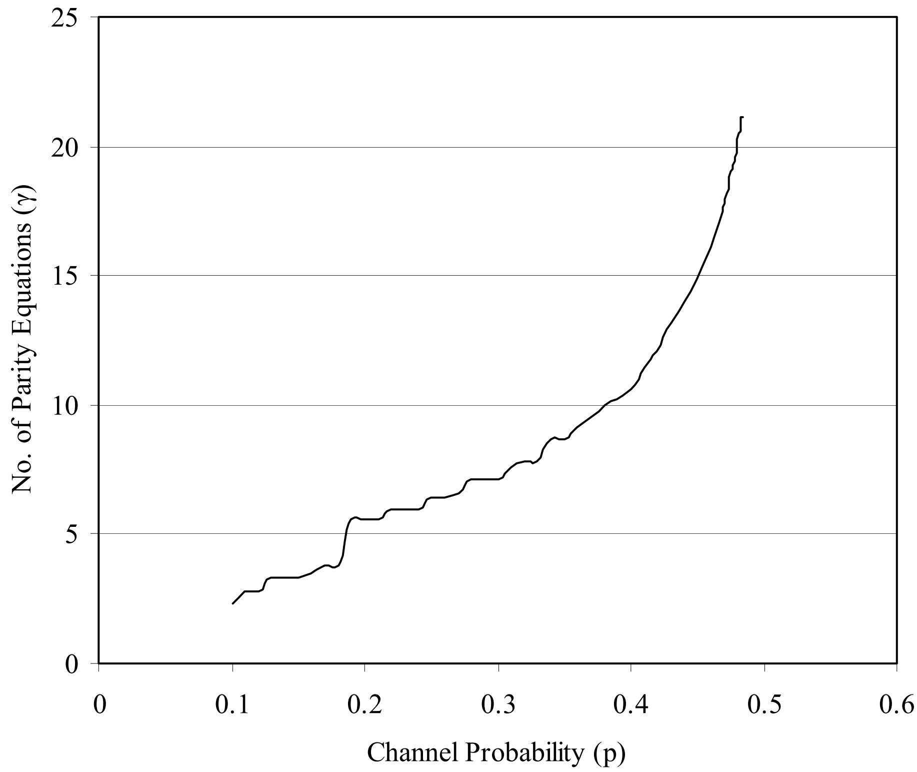

Figure 4.37 Worst case number of parity equations required for success

The number of parity equations found, depend on the size of G_{LFSR} searched through (which amounts to the amount of ciphertext required) as well as the size *B* chosen for the convolutional encoder.

The number of parity equations that are likely to be found grows exponentially with B (see equation (4.50)). Unfortunately, as B grows, the memory requirements (M_{Memory}) and computational complexity ($N_{Compuations}$) also grow exponentially, as can be approximated by equations (4.104) and (4.105) below:

$$M_{Memory} \propto 2^{B+1} \cdot \Gamma \tag{4.104}$$

$$N_{Computations} \propto 2^{B+1} \cdot \Gamma \tag{4.105}$$

Finding parity equations within G_{LFSR} of dimensions $l \times N$ grows directly proportional to the square of N as shown in below in (4.106).

$$N_{Computations} \propto \frac{1}{2} \cdot N \cdot (N-1) \tag{4.106}$$

At a first glance one would guess that searching for equations using more ciphertext should be easier and one can thus reduce the size of the convolutional encoder that is constructed. Unfortunately the amount of parity equations that can be found is reduced exponentially as B decreases, thus using equation (4.56) the above relation can be rewritten as follows.

$$N_{Computations} \propto \Gamma \cdot 2^{l-B} \tag{4.107}$$

The actual number of computations and memory requirements obviously depend a lot on the implementation of the algorithm and the use of more memory can be traded off for fewer operations and vice versa.

The procedure for successfully using the fast correlation attack is summarized in the following points if the attacker is to be successful, or alternatively for designing a cipher system that is safe from a fast correlation attack.

(1) Using Figure 4.37 the worst case (B = 2) number of parity equations that are required, is determined according to the correlation level p of the system.

(2) Equation (4.104) now establishes the maximum size convolutional encoder that can possibly be constructed.

(3) Using equation (4.107) it can be determined if sufficient parity equations can be found in a realistic time while at the same time looking at equation (4.56) it can be determined if sufficient ciphertext is available.

4.5 Deviations from Method Described by Johansson and Jönsson

Johansson and Jönsson [3] suggest running the Viterbi decoding process over a number of dummy information symbols before coming to the *l* information symbols to be decoded (One does not need to correctly decode the initial state of the LFSR, knowing any state, at a given time, is enough as one can derive any previous or future state from this information). Similarly they also suggest running the Viterbi over another set of dummy information symbols after the *l* information symbols to be decoded. It was experimentally found that this did not make identifiable difference to directly decoding the first *l* symbols from the received stream r_n generated by equation (4.76).

They further suggests using the metrics $P\left(v_n^{(0)} = r_n^{(0)}\right) = 1 - p$ and $P\left(v_n^{(i)} = r_n^{(i)}\right) = (1-p)^2 + p^2$ for $B + 1 \leq n \leq l + 10B$. It was found when trying to decode a LFSR of size 40, transmitted through a channel with $p \approx 0.4$, a rate in excess of $R = 10^3$ was necessary, resulting in the fact that only $r_n^{(0)}$ had a different metric while in excess of 10000 ($r_n^{(1)} \cdots r_n^{(m)}$ with $m \geq 10^3$) bits in the stream where assigned the same metric. As only $r_n^{(0)}$ is weighted differently this is insignificant, added to the fact that these metrics complicate the implementation of the Viterbi algorithm it was found that the implementation works adequately when using metrics as specified in equation (4.75).

A further specification not implemented was the assigning of initial metrics $\log\left(P\left(s = (z_1, z_2, \cdots, z_B)\right)\right)$ for each state before starting the Viterbi algorithm.

CHAPTER 5   DECIMATION ATTACK

5.1 Introduction

The same models are used for a decimation attack as for a correlation or fast-correlation attack. The ciphertext is obtained by bitwise addition of the plaintext to a running key. A pseudo-random generator whose initial state constitutes the secret key produces the running key.

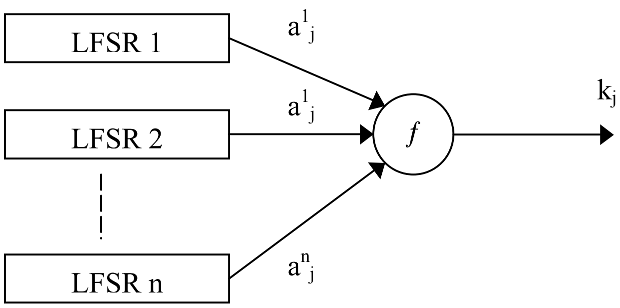

Figure 5.1   Nonlinear combination generator

A decimation attack cannot be used on it's own. The method attempts to reduce the size LFSR being attacked by selectively only using every D-th bit of the ciphertext stream. This approach is used when the LFSR is too large to directly attack. The LFSR has to be decimated to a size where the reduced LFSR can be successfully attacked using a correlation or fast-correlation attack. This method reduces the LFSR size to attack but massively increases the amount of ciphertext that is required. Thus a tradeoff is made of reduced complexity for increased ciphertext amounts.

5.2 Decimation of LFSR Sequences

Consider a sequence $a = a_1, a_2, \cdots$, produced by a LFSR of length L whose feedback polynomial is irreducible in $GF(2)$. By now taking every D-th element in a produces the subsequence $a^* = a_1^*, a_2^*, \cdots$. This is equivalent to the D-decimation of the original sequence. This so-called D-fold clocking of the LFSR causes the original LFSR to behave like a different LFSR called the simulated LFSR. When choosing D correctly the simulated LFSR has properties, which can be exploited to ones advantage.

Let L be the period of the LFSR of size l, thus:

$$L = 2^l - 1 \tag{5.1}$$

Let a^* be the sequence resulting from the D-th decimation of a, thus $a^* = a_{i \cdot d}, d \geq 0$. The simulated LFSR has the following properties of interest:

(1) The period L^* of the simulated LFSR is equal to

$$\frac{L}{\gcd(D, L)} \tag{5.2}$$

(2) The degree l^* of the simulated LFSR is equal to the multiplicative order of q in L^*

All D in C_k, where

$$C_k = \{k, kq, kq^2, \cdots,\} \bmod T \tag{5.3}$$

denotes the cyclotomic set of $k \bmod T$ results in the same simulated LFSR, except for different initial conditions. Every sequence produced by the simulated LFSR is equal to $a_{i \cdot d}$ for some choice of the initial contents of the original LFSR.

The goal of this procedure is finding a decimation factor D which process a sequence L^* where the degree l^* is *lower* than the degree l of the original sequence. The feedback polynomial $P^*(x)$ of the simulated LFSR can be obtained by applying either the Berlekamp-Massey LFSR synthesis algorithm [24] to the sequence a^*, or using the algorithm proposed in section 5.2.2.

5.2.1 Example of Finding a Useful Decimation Factor d

Consider a LFSR of size $l = 18$. The first step with finding an appropriate decimation factor D is the factoring of $L = 2^l - 1, l = 18$ [3]. L has the following prime factors: $L = 3 \cdot 3 \cdot 3 \cdot 7 \cdot 19 \cdot 73$. Thus, in this case, there is a choice of 32 possible values for D. Example: $D = 3, 3 \cdot 3, 3 \cdot 3 \cdot 3, 7, 7 \cdot 3, 7 \cdot 3 \cdot 3, 7 \cdot 3 \cdot 3 \cdot 3, \cdots$. Table 5.2 gives a list of all possible decimation factors D [4] (column $D \cdot 2^0$) with its associated cyclotomic set. The cyclotomic set is formed by the sequence $D \cdot 2^0 \bmod L, D \cdot 2^1 \bmod L, \cdots, D \cdot 2^l \bmod L$.

Any cyclotomic set where the sequence repeats before reaching l elements has a lower degree l^* than l and thus a useful decimation factor D. The degree of l^* is the length of a cyclotomic set before repeating. Looking at Table 5.2 it can be seen that for $D = 513$ the degree l^* is equal to 9.

Looking at Table 5.2 it can be seen that a size 18 LFSR has the 7 useful decimation factors shown in Table 5.1 below.

Table 5.1 Useful decimation factors for LFSR of size 18

D	l^*
513	9
3591	9
4161	6
12483	6
29127	6
37449	3
87381	2

[3] This obviously implies that if L is prime for a certain l the decimation attack will not work.

[4] For $D \in GCD(L^*, D) = D$

Table 5.2 Cyclotomic set of all possible decimation factors in $GF(2^{18})$

$D\cdot 2^0$	$D\cdot 2^1$	$D\cdot 2^2$	$D\cdot 2^3$	$D\cdot 2^4$	$D\cdot 2^5$	$D\cdot 2^6$	$D\cdot 2^7$	$D\cdot 2^8$	$D\cdot 2^9$	$D\cdot 2^{10}$	$D\cdot 2^{11}$	$D\cdot 2^{12}$	$D\cdot 2^{13}$	$D\cdot 2^{14}$	$D\cdot 2^{15}$	$D\cdot 2^{16}$	$D\cdot 2^{17}$
3	6	12	24	48	96	192	384	768	1536	3072	6144	12288	24576	49152	98304	196608	131073
7	14	28	56	112	224	448	896	1792	3584	7168	14336	28672	57344	114688	229376	196609	131075
9	18	36	72	144	288	576	1152	2304	4608	9216	18432	36864	73728	147456	32769	65538	131076
19	38	76	152	304	608	1216	2432	4864	9728	19456	38912	77824	155648	49153	98306	196612	131081
21	42	84	168	336	672	1344	2688	5376	10752	21504	43008	86016	172032	81921	163842	65541	131082
27	54	108	216	432	864	1728	3456	6912	13824	27648	55296	110592	221184	180225	98307	196614	131085
57	114	228	456	912	1824	3648	7296	14592	29184	58368	116736	233472	204801	147459	32775	65550	131100
63	126	252	504	1008	2016	4032	8064	16128	32256	64512	129024	258048	253953	245763	229383	196623	131103
73	146	292	584	1168	2336	4672	9344	18688	37376	74752	149504	36865	73730	147460	32777	65554	131108
133	266	532	1064	2128	4256	8512	17024	34048	68096	136192	10241	20482	40964	81928	163856	65569	131138
171	342	684	1368	2736	5472	10944	21888	43776	87552	175104	88065	176130	90117	180234	98325	196650	131157
189	378	756	1512	3024	6048	12096	24192	48384	96768	193536	124929	249858	237573	213003	163863	65583	131166
219	438	876	1752	3504	7008	14016	28032	56064	112128	224256	186369	110595	221190	180237	98331	196662	131181
399	798	1596	3192	6384	12768	25536	51072	102144	204288	146433	30723	61446	122892	245784	229425	196707	131271
511	1022	2044	4088	8176	16352	32704	65408	130816	261632	261121	260099	258055	253967	245791	229439	196735	131327
513	1026	2052	4104	8208	16416	32832	65664	131328	513	1026	2052	4104	8208	16416	32832	65664	131328
657	1314	2628	5256	10512	21024	42048	84096	168192	74241	148482	34821	69642	139284	16425	32850	65700	131400
1197	2394	4788	9576	19152	38304	76608	153216	44289	88578	177156	92169	184338	106533	213066	163989	65835	131670
1387	2774	5548	11096	22192	44384	88768	177536	92929	185858	109573	219146	176149	90155	180310	98477	196954	131765
1533	3066	6132	12264	24528	49056	98112	196224	130305	260610	259077	256011	249879	237615	213087	164031	65919	131838
1971	3942	7884	15768	31536	63072	126144	252288	242433	222723	183303	104463	208926	155709	49275	98550	197100	132057
3591	7182	14364	28728	57456	114912	229824	197505	132867	3591	7182	14364	28728	57456	114912	229824	197505	132867

$D \cdot 2^0$	$D \cdot 2^1$	$D \cdot 2^2$	$D \cdot 2^3$	$D \cdot 2^4$	$D \cdot 2^5$	$D \cdot 2^6$	$D \cdot 2^7$	$D \cdot 2^8$	$D \cdot 2^9$	$D \cdot 2^{10}$	$D \cdot 2^{11}$	$D \cdot 2^{12}$	$D \cdot 2^{13}$	$D \cdot 2^{14}$	$D \cdot 2^{15}$	$D \cdot 2^{16}$	$D \cdot 2^{17}$
4161	8322	16644	33288	66576	133152	4161	8322	16644	33288	66576	133152	4161	8322	16644	33288	66576	133152
4599	9198	18396	36792	73584	147168	32193	64386	128772	257544	252945	243747	225351	188559	114975	229950	197757	133371
9709	19418	38836	77672	155344	48545	97090	194180	126217	252434	242725	223307	184471	106799	213598	165053	67963	135926
12483	24966	49932	99864	199728	137313	12483	24966	49932	99864	199728	137313	12483	24966	49932	99864	199728	137313
13797	27594	55188	110376	220752	179361	96579	193158	124173	248346	234549	206955	151767	41391	82782	165564	68985	137970
29127	58254	116508	233016	203889	145635	29127	58254	116508	233016	203889	145635	29127	58254	116508	233016	203889	145635
37449	74898	149796	37449	74898	149796	37449	74898	149796	37449	74898	149796	37449	74898	149796	37449	74898	149796
87381	174762	87381	174762	87381	174762	87381	174762	87381	174762	87381	174762	87381	174762	87381	174762	87381	174762

5.2.2 Determining the Feedback Polynomial of the Simulated LFSR

The feedback polynomial, $P^*(x)$, can be obtained by using the equivalent block code (see section 4.2.2) of the LFSR. As the size of the simulated LFSR is already known (refer to section 5.2 point (2)), all that remains to be done is the determining of $P^*(x)$. As has already been discussed previously, the equivalent block code can be written in the following form:

$$G_{LFSR} = [I_l \quad Z] \tag{5.4}$$

A interesting observation that can be made is that the first column vector after the identity matrix I_l is always the recurrence relation of the $P(x)$ from which G_{LFSR} was formed as shown below.

$$G_{LFSR} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & a_{n-l} & z_{0,1} & z_{0,2} & \cdots \\ 0 & 1 & 0 & \cdots & 0 & a_{n-l+1} & z_{1,1} & z_{1,2} & \cdots \\ \vdots & & \ddots & & \vdots & \vdots & \vdots & & \\ 0 & 0 & 0 & \cdots & 1 & a_{n-1} & z_{l-1,1} & z_{l-1,2} & \cdots \end{bmatrix} \tag{5.5}$$

Using this fact every D-th column from G_{LFSR} is taken, using only the first l^* rows. Because each row in G_{LFSR} was linearly independent one knows that l^* rows in the decimated block matrix G^*_{LFSR} are also linearly independent. After performing Gauss-Jordan reduction on G^*_{LFSR} one is left with a matrix in the form $G^*_{LFSR} = [I_{l^*} \quad Z]$.

It is known that the first column after the identity matrix is the recurrence relation of the equivalent LFSR, thus when transforming this column vector from it's recurrence relation to the polynomial form, $P^*(x)$ has been found.

5.2.3 Theoretical Discussion of Decimation Method

The size of the simulated LFSR can also be found by looking at the order of L^*. The order l^* of L^* is the smallest positive integer such that

$$2^{l^*} \bmod L^* = 1 \tag{5.6}$$

To see how effective the decimation attack can be, a new parameter, d, is introduced, which is equal to the degree of D. E.g. $D = x^5 + x^2 + 1$ then $d = 5$.

It is known that $L^* = \frac{L}{D}$, thus

$$L^* \geq \frac{2^l - 1}{2^d} \tag{5.7}$$

The smallest possible value for l^* is achieved for

$$L^* = 2^{l^*} - 1 \tag{5.8}$$

thus when the simulated LFSR itself is also a maximum length pn-sequence.

Combining equation (5.7) and (5.8) and making the assumption that $2^l >> 1$, $2^{l^*} >> 1$ it is found that

$$2^{l^*} \geq \frac{2^l}{2^d} \tag{5.9}$$

thus

$$l^* \geq l - d \tag{5.10}$$

Looking at equation (5.10) it is found that there is a direct trade-off between the size of the decimation factor D and the size of the resulting simulated LFSR size l^*.

5.2.3.1 Example

Looking at a LFSR of size 60, it is far too large to break using a correlation attack, and probably too large (too memory intensive) to break by using a fast-correlation attack. Concentrating on the fast-correlation attack, a LFSR of around 40 bits could be broken. Thus, if one could find a decimation factor of around $D = 1000000 \Rightarrow d \approx 20$, one could produce a simulated LFSR of $l^* = 60 - 20 = 40$, which can be broken using the fast correlation attack within minutes.

The downside of this is if one has a channel probability of $p \approx 0.47$, around 40000 bits are required to break the 40-bit simulated LFSR. As only every 1000000-th bit of the original cipher stream is used after the decimation process, effectively $40000000000 \approx 2^{35}$ bits of ciphertext are required, a tall order.

5.2.4 Results from Investigation

Filiol [1] presents a list of LFSR which are impervious to the decimation attack as L is either prime or does not have a decimation factor D which produces a simulated LFSR with $l^* < l$. The list is repeated below.

Table 5.3 LFSR of size l, immune to decimation attack

	l
Prime L	5, 7, 13, 17, 19, 31, 61, 89, 107, 127
l^* not smaller than l	11, 23, 29, 37, 41, 43, 47, 53, 59, 67, 71, 73, 83, 97, 101, 109, 113, 131, 139, 149, 151, 157, 163, 167, 173, 178, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251

Further all LFSR sizes for $18 \le l \le 64$ were parsed for useful decimation factors $D \le 2^{31} - 1$. What has to be remembered is that the total amount of ciphertext bits required when using the decimation attack amounts to the product of the decimation factor and the number of bits typically required for the attack used after decimating the LFSR for a certain channel probability p.

Appendix F lists all decimation factors smaller than 2^{32} for $18 \le l \le 64$. From this list many more values of l are found which probably can also be considered safe from a decimation attack due to the enormous values of D which would require such a huge amount of ciphertext that a decimation attack would be completely unfeasible.

A new parameter l_{req} is introduced as the maximum LFSR size that can be broken without decimating. This allows for identifying weaknesses of LFSRs with sizes in excess of 64 bits which are not contained in Appendix F. Using equation (5.10) it is found that the degree of the decimation factor D then needs to be at least

$$d = l - l_{req} \tag{5.11}$$

When designing a stream cipher system l must be chosen such that D (if l is not contained in Table 5.3 or Appendix F) is too large to be useful for any known attack that could be successful on a LFSR of size l_{req}.

CHAPTER 6 CONCLUSION

Four methods were investigated for breaking stream ciphers based on nonlinear combining generators. All four methods are ciphertext-only divide and conquer attacks which attempt to reconstruct the initial state of the LFSRs within the running key generator. Three different types of attacks were presented:

- A fast-correlation attack.
- Two correlation attacks:
    - The binary derivative attack.
    - The Lempel-Ziv attack.
- A decimation attack.

The investigation of the different type of attacks aims to give an indication whether a cipher system could be susceptible to the specified attack and the resources that would be required. This information can also be used when designing a new cipher system to choose the parameters so as to ensure that the system is not endangered by the attacks described here.

It has to be remembered that all the attacks investigated, attack one of the LFSRs contained in the running key generator. This means if one can obtain the initial state of a LFSR of size l bits, the system that is attacked actually has a much larger key, which is the sum of all initial conditions of all the component LFSRs contained within the key generator. Thus if the initial condition of a LFSR of 40 bits can be retrieved (as was successfully shown for the two correlation attacks as well as the fast correlation attack), this may not sound very impressive as the key of most cipher systems is much larger (128 bits is the magic number currently used for most block ciphers). However, remembering that one is only talking about one of the component LFSRs here, this means, depending on the system, that keys in excess of 120-bit in strength can be broken (assuming there are at least 3 component LFSRs of similar size) and this is an important result.

6.1 Correlation Attacks

The two new correlation attacks that were investigated, i.e. the Lempel-Ziv method and the binary derivative method, are robust and easy to implement. These attacks where performed with a system based on a Pentium I, 200MHz processor with 112MB of memory on a Linux platform. Both attacks succeed even when only a very small measure of correlation occurs between the ciphertext and one of the component LFSRs. The Lempel-Ziv method succeeds for a correlation of $p = 0.482$ and requires approximately 62000 cipher-bits. The binary derivative method succeeds for $p = 0.47$ and requires only 24500 bits for this when using 20 derivatives.

In the case of the binary derivative there is always a trade-off between speed and the amount of ciphertext required. If no derivative is used, n operations are required. For every additional derivative D the number of operations increases linearly with D, i.e. $D \cdot n$. Generally speaking, the required number of ciphertext bits required grows exponentially as the correlation level p drops. As the amount of ciphertext bits increase, so do the memory requirements as well as the computational load. The big advantage of the binary derivative method is the fact that a trade-off exists between the number of derivatives and ciphertext. Although more derivatives require more processing power, in the process the amount of required ciphertext is drastically reduced. This relationship was presented in Figure 3.7.

The Lempel-Ziv attack is simpler than the binary derivative method in the sense that the success of this attack depends only on one parameter, which is the amount of ciphertext that is required. This relationship was presented in Figure 3.5.

When comparing the two methods using the two figures mentioned it is found that the binary derivative method and the Lempel-Ziv method use approximately the same amount of ciphertext, if the binary derivative method uses a large number of derivatives. The Lempel-Ziv method gives better results at correlation levels $p > 0.47$ than the binary derivative. However, if sufficient ciphertext is available, the binary derivative algorithm is faster than the Lempel-Ziv method when utilizing a small number of derivatives.

The obvious limitation with both these methods is the fact that they need to perform an exhaustive search to find the correct LFSR initial condition. The longer the LFSR that is being attacked, the more time it will take. Since this relationship is exponential, the attacks cannot be expected to be practical for a LFSR of more than about 40 bits. However, the big advantage with these methods is the fact that the amount of ciphertext required is independent of the size of the LFSR and both attacks are ideal for execution on parallel processors.

6.2 Fast-Correlation Attacks

The fast-correlation attack using the Viterbi algorithm is fairly complex in comparison with the correlation attacks although the extra complexity is worthwhile. All simulations for this method where performed on a Pentium IV, 2GHz processor with 256MB memory on a Windows 2000 platform. The fast correlation method was tested for correlation levels as low p = 0.485, using a 7-bit (128-state) convolutional encoder and enough ciphertext to provide 2873370 parity equations; in this case, when attacking a 19-bit LFSR, 153448 bits of ciphertext were required to succeed. The memory required for this was about 160MB.

There are two distinct stages in this algorithm: Firstly, the finding of parity equations in the LFSR structure, and thereafter using the convolutional encoder (constructed using the parity equations) for the extraction of the targeted pn-sequence from the ciphertext. To break a system of a certain correlation level p, sufficient parity equations to construct the convolutional encoder have to be found. The relationship between the correlation level p and the number of required parity equations was shown in Figure 4.37 and is repeated here in Figure 6.1 because of its importance.

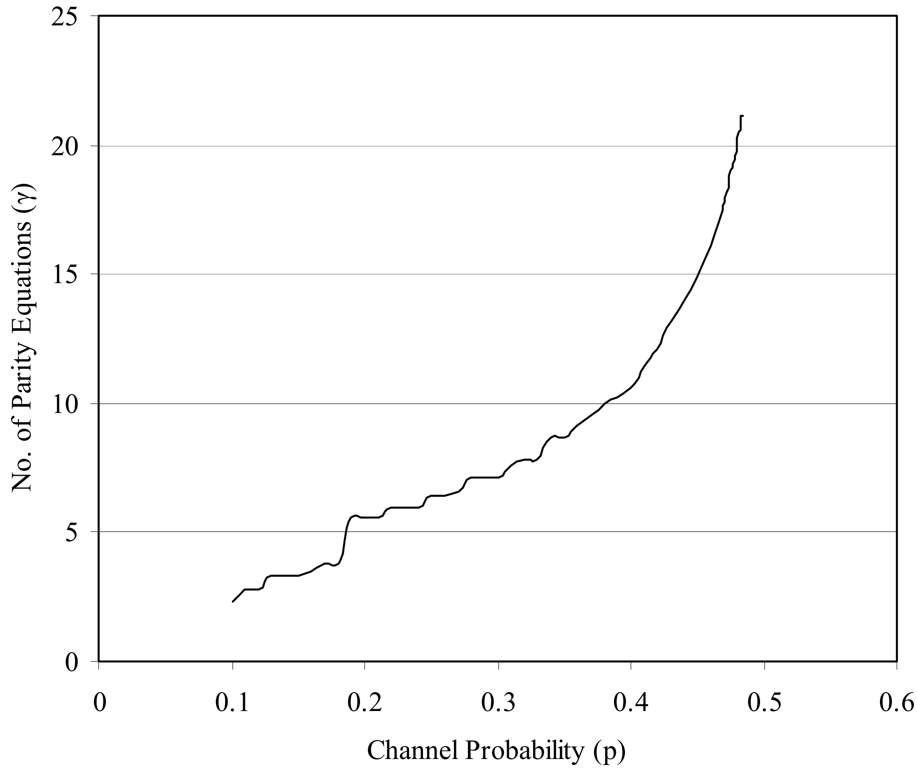

Figure 6.1 No. of parity equations (γ) required for success

The number of computations ($N_{Computations}$) required for finding a certain number of parity equations, Γ, was given by equation (4.107) and is repeated below.

$$N_{Computations} \propto \Gamma \cdot 2^{l-B} \tag{6.1}$$

An attacker can thus see from equation (6.1) whether it is feasible to find a sufficient number of parity equations within the LFSR structure. Note that this has to be done only once for any given cipher system, since finding parity equations is independent of the specific session key being attacked. Therefore extensive resources can be allocated for this task.

From equation (6.1) it can be seen that sufficient parity equations could always be found by increasing the size *B* of the convolutional encoder. Unfortunately, there are two important restraints on the size of the encoder when using it to extract the targeted LFSR output. The larger it becomes, the more operations are required for the decoding process and the larger the memory requirement (M_{Memory}) becomes. This relationship was originally presented in equations (4.104) and (4.105), repeated below.

$$N_{Computations} \propto 2^{B+1} \cdot \Gamma \tag{6.2}$$

$$M_{Memory} \propto 2^{B+1} \cdot \Gamma \tag{6.3}$$

The extraction of the targeted LFSR output has to be performed each time a new session key is being attacked. For this reason it should always be attempted to utilize as many resources as possible to find parity equations in such a way as to minimize *B*.

The big advantage of the fast correlation attack is the fact that it does not perform an exhaustive search for the initial condition of the targeted LFSR. However, the initial condition derived by the fast correlation attack is not necessarily correct. The values of the minimum number of parity equations required for success are average values and should succeed in at least 80% of the time. The results however still need to be verified and for this, the correlation attack methods should be very effective, as it's complexity is not dependant on *l* if the intention is only to verify a LFSR initial condition. Therefore a correlation attack can complement a fast correlation attack and is not obsoleted by it.

6.3 Decimation Attack

The decimation attack differs from the other two types in the sense that it is not a stand-alone attack and still needs a secondary method to find the correct initial state. The purpose of the decimation attack is to reduce the effective size of a LFSR targeted within the key generator by using only every *D*-th bit of the cipher stream. In equation (5.10), repeated here for convenience, the best possible result that can be achieved was shown. In the equation, l^* represents the size of the reduced LFSR, *l*, represents the size of the targeted LFSR and *d* represents the magnitude of the decimation factor *D*, thus $2^d \le D < 2^{d+1}$.

$$l^* \ge l - d \tag{6.4}$$

When applying this attack only every *D*-th bit within the ciphertext is used for the secondary attack on the decimated LFSR. To reduce a LFSR by any significant amount the equation above shows that *D* must be large. A list of decimation factors of $D < 2^{31}$ for $18 \le l \le 64$ is presented in Appendix F. If it is intended to use any of the previously discussed attacks for the secondary attack, it has already been seen that several tens of thousand of bits are required to attack a cipher system which has low correlation levels between the ciphertext and the targeted LFSR. Because of this, the amount of ciphertext required is huge. Added to this it was found that the attack can only be considered if one is lucky enough that the system to be attacked has a LFSR of size *l* for which a decimation factor *D* even exists. Table 5.3 gives a list of for all the sizes *l* of a LFSR for which no decimation factors exist.

However, the decimation attack is still attractive since there is no processing or memory penalty when using this method. If a decimation factor *D* exists, it can be used for attacking smaller LFSRs, which automatically require smaller decimation factors to reduce.

6.4 Future Work on Fast Correlation Attack

Since the Viterbi decoding process is not started with the all-0 state, it is likely that the decoding process may fail at the first stage of the trellis. This could result in the failure of all further decoding stages. It is vital to starting with the correct initial path, so as to exploit the full power of the Viterbi decoding algorithm. Hence it is worthwhile to investigate the adaptation of the Viterbi algorithm to keep all paths (tree code) for the first two or three stages within the trellis diagram. This would allow a longer history of the partial path metrics, which would give a better indication of the wrong paths that may be discarded and also of the correct paths that are kept after completion of the first three stages.

This would increase the memory requirement for this section by at least $2^3 = 8$ times. However, this would not influence the remaining memory usage, but is one of the reasons why it was not further investigated in this dissertation as memory was the prime limitation.

REFERENCES

[1] E. Filiol, “Ciphertext Only Decimation Attack of Stream Ciphers”, INRIA Projet Codes, Le Chesnay Cedex, France, 2000.

[2] T. Siegenthaler, “Decrypting a class of stream ciphers using ciphertext only”, IEEE Trans. Computers, vol. C-34, pp. 81–85, 1985.

[3] T. Johansson, F. Jönsson, “Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes”, Dept. of Information Technology, Lund University, Sweden, 1999.

[4] W. Stallings, “Cryptography and Network Security: Principles and Practice”, Second Edition. Prentice Hall, New Jersey, 1999.

[5] J. Golic, “Cryptanalysis of Alleged A5 Stream Cipher”, School of Electrical Engineering, University of Belgrade, Yugoslavia, 1997.

[6] J. F. Wakerly, “Digital Design, Principles and Practices”, Second Edition, Prentice Hall, New Jersey, USA, p 626, 1994.

[7] A. Menezes, P. van Oorschot, S. Vanstone, “Handbook of Applied Cryptography”, First Edition, CRC Press, Boca Raton, Florida, USA, Chapter 6, 1996.

[8] J. O. Brüer, “On nonlinear combinations of linear shift register sequences”, Proc. IEEE ISIT, les Arcs, France, June 21-25 1982.

[9] P. R. Geffe, “How to protect data with ciphers that are really hard to break”, Electronics, pp. 99–101, January 1973.

[10] V. S. Pless, “Encryption schemes for computer confidentiality”, IEEE Trans. Computers, vol. C-26, pp. 1133–1136, November 1977.

[11] J. Ziv, A. Lempel, “Compression of individual sequences via variable-rate coding”, IEEE Trans. On Information Theory, vol. IT-24, no. 5, pp. 530–536, September 1978.

[12] E. N. Gilbert, T. T. Kadota, “The Lempel-Ziv algorithm and message complexity”, IEEE Trans. on Information Theory, vol. IT-38, no. 6, pp. 1839–1842, November 1992.

[13] J. M. Carrol, L. E. Robbins, “Using binary derivatives to test an enhancement of DES”, Cryptologia, vol. XII, no. 4, pp. 193-208, October 1988.

[14] Barbé A, “Binary random sequences: Derivative sequences and multilevel α-typical randomness”, 8th Benelux Symposium on Information Theory, University of Twente, Belgium, 1986.

[15] J. M. Carrol, “The binary derivative test for the appearance of randomness and its use as a noise filter”, Technical Report No. 221, Dept. of Computer Science, University of Western Ontario, November 1988.

[16] J. W. McNair, “The binary derivative: A new method of testing for the appearance of randomness in a sequence of bits”, M.Sc. Thesis, Dept. of Computer Science, University of Western Ontario, London, Ontario, Canada, May 1989.

[17] J. S. Bendat, A. G. Piersol, “Random Data: Analysis and Measurement Procedures”, Second Edition, Wiley, New York, 1986.

[18] H. R. Neave, P. L. Worthington, “Distribution-free tests”, Unwin Hyman, London, 1988.

[19] W. T. Penzhorn and C. S. Bruwer, "New correlation attacks on stream ciphers", Proc. IEEE AFRICON 2002, 1 - 3 October 2002, George, South Africa, pp. 203-208, 2002.

[20] S. B. Wicker, “Error Control Systems for Digital Communication and Storage”, Prentice-Hall, New Jersey, pp. 264-327, 1995.

[21] R. E. Blahut, “Theory and Practice of Error Control Codes”, Addison-Wesley, New York, pp. 348-350, 1983.

[22] T.R.N. Rao, E. Fujiwara, ”Error-Control Coding for Computer Systems”, Second Editions, Prentice-Hall, New York, p 70, 1989.

[23] W. T. Penzhorn, “Discrimination of Deterministic Binary Sequences”, Internal Report, University of Pretoria, 12 March 1993.

[24] J. L. Massey, “Shift-Register Synthesis and BCH Decoding”, IEEE Transactions on Information Theory, Vol. IT-15, January 1969.

[25] S. B. Wicker, “Error Control Systems for Digital Communication and Storage”, Prentice-Hall, New Jersey, pp. 21-45, 1995.

APPENDIX

TABLE OF CONTENTS

A LEMPEL-ZIV COMPLEXITY FOR RANDOM BINARY SEQUENCES 123
B AMOUNT OF CIPHERTEXT BITS REQUIRED FOR LEMPEL-ZIV ATTACK TO BE SUCCESSFUL 130
C NUMBER OF DERIVATIVES FOR BINARY DERIVATIVE ATTACK TO SUCCEED. 131
D EXPECTED NUMBER OF PARITY EQUATIONS 133
E AVERAGE NUMBER OF PARITY EQUATIONS REQUIRED BY FAST CORRELATION ATTACK 137
F SELECTED DECIMATION FACTORS 138

A LEMPEL-ZIV COMPLEXITY FOR RANDOM BINARY SEQUENCES

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
10	23.73	1.50	3 340	33 387.36	29.80	6 670	73 308.27	42.12
20	61.64	2.20	3 350	33 501.71	29.84	6 680	73 432.58	42.15
30	106.36	2.74	3 360	33 616.10	29.89	6 690	73 556.92	42.18
40	155.67	3.19	3 370	33 730.53	29.93	6 700	73 681.27	42.21
50	208.44	3.58	3 380	33 845.01	29.98	6 710	73 805.64	42.24
60	264.01	3.93	3 390	33 959.53	30.02	6 720	73 930.04	42.27
70	321.92	4.26	3 400	34 074.09	30.07	6 730	74 054.46	42.31
80	381.84	4.56	3 410	34 188.69	30.11	6 740	74 178.90	42.34
90	443.52	4.84	3 420	34 303.34	30.15	6 750	74 303.36	42.37
100	506.77	5.11	3 430	34 418.03	30.20	6 760	74 427.84	42.40
110	571.44	5.36	3 440	34 532.76	30.24	6 770	74 552.34	42.43
120	637.40	5.61	3 450	34 647.53	30.29	6 780	74 676.87	42.46
130	704.54	5.84	3 460	34 762.34	30.33	6 790	74 801.42	42.49
140	772.78	6.06	3 470	34 877.20	30.37	6 800	74 925.98	42.52
150	842.04	6.28	3 480	34 992.09	30.42	6 810	75 050.57	42.56
160	912.24	6.49	3 490	35 107.03	30.46	6 820	75 175.18	42.59
170	983.34	6.69	3 500	35 222.01	30.50	6 830	75 299.81	42.62
180	1 055.27	6.88	3 510	35 337.03	30.55	6 840	75 424.46	42.65
190	1 128.00	7.07	3 520	35 452.09	30.59	6 850	75 549.14	42.68
200	1 201.48	7.26	3 530	35 567.20	30.63	6 860	75 673.83	42.71
210	1 275.67	7.44	3 540	35 682.34	30.68	6 870	75 798.55	42.74
220	1 350.55	7.62	3 550	35 797.52	30.72	6 880	75 923.28	42.77
230	1 426.07	7.79	3 560	35 912.75	30.76	6 890	76 048.04	42.81
240	1 502.22	7.96	3 570	36 028.01	30.81	6 900	76 172.82	42.84
250	1 578.96	8.12	3 580	36 143.32	30.85	6 910	76 297.62	42.87
260	1 656.28	8.29	3 590	36 258.67	30.89	6 920	76 422.44	42.90
270	1 734.14	8.44	3 600	36 374.05	30.94	6 930	76 547.28	42.93
280	1 812.54	8.60	3 610	36 489.48	30.98	6 940	76 672.14	42.96
290	1 891.45	8.75	3 620	36 604.95	31.02	6 950	76 797.02	42.99
300	1 970.85	8.90	3 630	36 720.45	31.07	6 960	76 921.92	43.02
310	2 050.73	9.05	3 640	36 836.00	31.11	6 970	77 046.85	43.05
320	2 131.08	9.20	3 650	36 951.58	31.15	6 980	77 171.79	43.08
330	2 211.87	9.34	3 660	37 067.21	31.19	6 990	77 296.76	43.11
340	2 293.09	9.48	3 670	37 182.87	31.24	7 000	77 421.74	43.15
350	2 374.74	9.62	3 680	37 298.58	31.28	7 010	77 546.75	43.18
360	2 456.80	9.76	3 690	37 414.32	31.32	7 020	77 671.78	43.21
370	2 539.26	9.90	3 700	37 530.10	31.36	7 030	77 796.82	43.24
380	2 622.10	10.03	3 710	37 645.92	31.41	7 040	77 921.89	43.27
390	2 705.32	10.16	3 720	37 761.78	31.45	7 050	78 046.98	43.30
400	2 788.91	10.29	3 730	37 877.68	31.49	7 060	78 172.09	43.33
410	2 872.86	10.42	3 740	37 993.62	31.53	7 070	78 297.22	43.36
420	2 957.16	10.55	3 750	38 109.59	31.58	7 080	78 422.37	43.39
430	3 041.80	10.67	3 760	38 225.61	31.62	7 090	78 547.54	43.42

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
440	3 126.78	10.80	3 770	38 341.66	31.66	7 100	78 672.73	43.45
450	3 212.08	10.92	3 780	38 457.75	31.70	7 110	78 797.94	43.48
460	3 297.70	11.04	3 790	38 573.88	31.74	7 120	78 923.17	43.51
470	3 383.63	11.16	3 800	38 690.05	31.79	7 130	79 048.42	43.54
480	3 469.87	11.28	3 810	38 806.25	31.83	7 140	79 173.69	43.58
490	3 556.41	11.39	3 820	38 922.49	31.87	7 150	79 298.98	43.61
500	3 643.24	11.51	3 830	39 038.77	31.91	7 160	79 424.30	43.64
510	3 730.36	11.63	3 840	39 155.09	31.95	7 170	79 549.63	43.67
520	3 817.76	11.74	3 850	39 271.45	31.99	7 180	79 674.98	43.70
530	3 905.43	11.85	3 860	39 387.84	32.04	7 190	79 800.35	43.73
540	3 993.38	11.96	3 870	39 504.27	32.08	7 200	79 925.75	43.76
550	4 081.60	12.07	3 880	39 620.74	32.12	7 210	80 051.16	43.79
560	4 170.07	12.18	3 890	39 737.24	32.16	7 220	80 176.59	43.82
570	4 258.81	12.29	3 900	39 853.79	32.20	7 230	80 302.04	43.85
580	4 347.79	12.40	3 910	39 970.36	32.24	7 240	80 427.51	43.88
590	4 437.02	12.51	3 920	40 086.98	32.28	7 250	80 553.01	43.91
600	4 526.50	12.61	3 930	40 203.63	32.32	7 260	80 678.52	43.94
610	4 616.21	12.72	3 940	40 320.32	32.37	7 270	80 804.05	43.97
620	4 706.17	12.82	3 950	40 437.05	32.41	7 280	80 929.60	44.00
630	4 796.35	12.93	3 960	40 553.81	32.45	7 290	81 055.17	44.03
640	4 886.76	13.03	3 970	40 670.61	32.49	7 300	81 180.77	44.06
650	4 977.40	13.13	3 980	40 787.44	32.53	7 310	81 306.38	44.09
660	5 068.26	13.23	3 990	40 904.32	32.57	7 320	81 432.01	44.12
670	5 159.33	13.33	4 000	41 021.22	32.61	7 330	81 557.66	44.15
680	5 250.62	13.43	4 010	41 138.17	32.65	7 340	81 683.33	44.18
690	5 342.12	13.53	4 020	41 255.15	32.69	7 350	81 809.02	44.21
700	5 433.83	13.63	4 030	41 372.16	32.73	7 360	81 934.73	44.24
710	5 525.75	13.72	4 040	41 489.21	32.77	7 370	82 060.46	44.27
720	5 617.87	13.82	4 050	41 606.30	32.81	7 380	82 186.21	44.30
730	5 710.19	13.92	4 060	41 723.42	32.86	7 390	82 311.97	44.33
740	5 802.70	14.01	4 070	41 840.58	32.90	7 400	82 437.76	44.36
750	5 895.41	14.11	4 080	41 957.77	32.94	7 410	82 563.57	44.39
760	5 988.32	14.20	4 090	42 075.00	32.98	7 420	82 689.40	44.42
770	6 081.41	14.29	4 100	42 192.27	33.02	7 430	82 815.24	44.45
780	6 174.68	14.39	4 110	42 309.57	33.06	7 440	82 941.11	44.48
790	6 268.15	14.48	4 120	42 426.90	33.10	7 450	83 066.99	44.51
800	6 361.79	14.57	4 130	42 544.27	33.14	7 460	83 192.90	44.54
810	6 455.62	14.66	4 140	42 661.67	33.18	7 470	83 318.82	44.57
820	6 549.62	14.75	4 150	42 779.11	33.22	7 480	83 444.76	44.60
830	6 643.80	14.84	4 160	42 896.59	33.26	7 490	83 570.73	44.63
840	6 738.15	14.93	4 170	43 014.10	33.30	7 500	83 696.71	44.66
850	6 832.67	15.02	4 180	43 131.64	33.34	7 510	83 822.71	44.69
860	6 927.36	15.11	4 190	43 249.22	33.38	7 520	83 948.73	44.72
870	7 022.22	15.20	4 200	43 366.83	33.42	7 530	84 074.77	44.75
880	7 117.25	15.28	4 210	43 484.47	33.46	7 540	84 200.82	44.78
890	7 212.44	15.37	4 220	43 602.15	33.50	7 550	84 326.90	44.81
900	7 307.79	15.46	4 230	43 719.87	33.54	7 560	84 453.00	44.84
910	7 403.30	15.54	4 240	43 837.62	33.58	7 570	84 579.11	44.87
920	7 498.96	15.63	4 250	43 955.40	33.62	7 580	84 705.25	44.90

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
930	7 594.79	15.71	4 260	44 073.22	33.65	7 590	84 831.40	44.93
940	7 690.77	15.80	4 270	44 191.07	33.69	7 600	84 957.58	44.96
950	7 786.90	15.88	4 280	44 308.95	33.73	7 610	85 083.77	44.99
960	7 883.18	15.96	4 290	44 426.87	33.77	7 620	85 209.98	45.02
970	7 979.62	16.05	4 300	44 544.82	33.81	7 630	85 336.21	45.05
980	8 076.20	16.13	4 310	44 662.81	33.85	7 640	85 462.46	45.08
990	8 172.93	16.21	4 320	44 780.83	33.89	7 650	85 588.72	45.10
1 000	8 269.81	16.29	4 330	44 898.88	33.93	7 660	85 715.01	45.13
1 010	8 366.82	16.38	4 340	45 016.96	33.97	7 670	85 841.31	45.16
1 020	8 463.99	16.46	4 350	45 135.08	34.01	7 680	85 967.64	45.19
1 030	8 561.29	16.54	4 360	45 253.23	34.05	7 690	86 093.98	45.22
1 040	8 658.73	16.62	4 370	45 371.42	34.09	7 700	86 220.34	45.25
1 050	8 756.31	16.70	4 380	45 489.64	34.13	7 710	86 346.72	45.28
1 060	8 854.03	16.78	4 390	45 607.89	34.16	7 720	86 473.12	45.31
1 070	8 951.88	16.86	4 400	45 726.17	34.20	7 730	86 599.54	45.34
1 080	9 049.87	16.93	4 410	45 844.49	34.24	7 740	86 725.97	45.37
1 090	9 147.99	17.01	4 420	45 962.84	34.28	7 750	86 852.43	45.40
1 100	9 246.25	17.09	4 430	46 081.22	34.32	7 760	86 978.90	45.43
1 110	9 344.63	17.17	4 440	46 199.63	34.36	7 770	87 105.39	45.46
1 120	9 443.15	17.25	4 450	46 318.08	34.40	7 780	87 231.90	45.49
1 130	9 541.79	17.32	4 460	46 436.56	34.44	7 790	87 358.43	45.52
1 140	9 640.56	17.40	4 470	46 555.07	34.47	7 800	87 484.98	45.54
1 150	9 739.46	17.48	4 480	46 673.61	34.51	7 810	87 611.55	45.57
1 160	9 838.48	17.55	4 490	46 792.19	34.55	7 820	87 738.13	45.60
1 170	9 937.63	17.63	4 500	46 910.80	34.59	7 830	87 864.73	45.63
1 180	10 036.90	17.70	4 510	47 029.44	34.63	7 840	87 991.36	45.66
1 190	10 136.29	17.78	4 520	47 148.11	34.67	7 850	88 118.00	45.69
1 200	10 235.80	17.85	4 530	47 266.81	34.71	7 860	88 244.65	45.72
1 210	10 335.43	17.93	4 540	47 385.55	34.74	7 870	88 371.33	45.75
1 220	10 435.19	18.00	4 550	47 504.32	34.78	7 880	88 498.02	45.78
1 230	10 535.05	18.07	4 560	47 623.12	34.82	7 890	88 624.74	45.81
1 240	10 635.04	18.15	4 570	47 741.95	34.86	7 900	88 751.47	45.84
1 250	10 735.14	18.22	4 580	47 860.81	34.90	7 910	88 878.22	45.86
1 260	10 835.36	18.29	4 590	47 979.70	34.93	7 920	89 004.99	45.89
1 270	10 935.70	18.37	4 600	48 098.63	34.97	7 930	89 131.77	45.92
1 280	11 036.14	18.44	4 610	48 217.58	35.01	7 940	89 258.58	45.95
1 290	11 136.70	18.51	4 620	48 336.57	35.05	7 950	89 385.40	45.98
1 300	11 237.37	18.58	4 630	48 455.59	35.09	7 960	89 512.24	46.01
1 310	11 338.15	18.65	4 640	48 574.64	35.12	7 970	89 639.10	46.04
1 320	11 439.04	18.72	4 650	48 693.72	35.16	7 980	89 765.98	46.07
1 330	11 540.04	18.80	4 660	48 812.84	35.20	7 990	89 892.87	46.10
1 340	11 641.15	18.87	4 670	48 931.98	35.24	8 000	90 019.78	46.12
1 350	11 742.37	18.94	4 680	49 051.15	35.28	8 010	90 146.71	46.15
1 360	11 843.69	19.01	4 690	49 170.36	35.31	8 020	90 273.66	46.18
1 370	11 945.12	19.08	4 700	49 289.59	35.35	8 030	90 400.63	46.21
1 380	12 046.65	19.15	4 710	49 408.86	35.39	8 040	90 527.62	46.24
1 390	12 148.29	19.22	4 720	49 528.16	35.43	8 050	90 654.62	46.27
1 400	12 250.03	19.28	4 730	49 647.48	35.46	8 060	90 781.64	46.30
1 410	12 351.87	19.35	4 740	49 766.84	35.50	8 070	90 908.68	46.33

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
1 420	12 453.82	19.42	4 750	49 886.23	35.54	8 080	91 035.73	46.36
1 430	12 555.87	19.49	4 760	50 005.65	35.58	8 090	91 162.81	46.38
1 440	12 658.02	19.56	4 770	50 125.10	35.61	8 100	91 289.90	46.41
1 450	12 760.27	19.63	4 780	50 244.58	35.65	8 110	91 417.01	46.44
1 460	12 862.62	19.69	4 790	50 364.09	35.69	8 120	91 544.14	46.47
1 470	12 965.06	19.76	4 800	50 483.63	35.73	8 130	91 671.28	46.50
1 480	13 067.61	19.83	4 810	50 603.20	35.76	8 140	91 798.45	46.53
1 490	13 170.25	19.90	4 820	50 722.80	35.80	8 150	91 925.63	46.56
1 500	13 272.99	19.96	4 830	50 842.43	35.84	8 160	92 052.83	46.58
1 510	13 375.82	20.03	4 840	50 962.09	35.87	8 170	92 180.04	46.61
1 520	13 478.75	20.10	4 850	51 081.78	35.91	8 180	92 307.28	46.64
1 530	13 581.78	20.16	4 860	51 201.49	35.95	8 190	92 434.53	46.67
1 540	13 684.90	20.23	4 870	51 321.24	35.98	8 200	92 561.80	46.70
1 550	13 788.11	20.29	4 880	51 441.02	36.02	8 210	92 689.09	46.73
1 560	13 891.42	20.36	4 890	51 560.83	36.06	8 220	92 816.39	46.75
1 570	13 994.82	20.42	4 900	51 680.67	36.10	8 230	92 943.71	46.78
1 580	14 098.31	20.49	4 910	51 800.53	36.13	8 240	93 071.05	46.81
1 590	14 201.89	20.55	4 920	51 920.43	36.17	8 250	93 198.41	46.84
1 600	14 305.56	20.62	4 930	52 040.36	36.21	8 260	93 325.78	46.87
1 610	14 409.32	20.68	4 940	52 160.31	36.24	8 270	93 453.18	46.90
1 620	14 513.17	20.75	4 950	52 280.29	36.28	8 280	93 580.59	46.93
1 630	14 617.11	20.81	4 960	52 400.31	36.32	8 290	93 708.01	46.95
1 640	14 721.14	20.87	4 970	52 520.35	36.35	8 300	93 835.46	46.98
1 650	14 825.26	20.94	4 980	52 640.42	36.39	8 310	93 962.92	47.01
1 660	14 929.46	21.00	4 990	52 760.52	36.43	8 320	94 090.40	47.04
1 670	15 033.75	21.06	5 000	52 880.65	36.46	8 330	94 217.89	47.07
1 680	15 138.13	21.13	5 010	53 000.81	36.50	8 340	94 345.41	47.10
1 690	15 242.59	21.19	5 020	53 120.99	36.53	8 350	94 472.94	47.12
1 700	15 347.14	21.25	5 030	53 241.21	36.57	8 360	94 600.49	47.15
1 710	15 451.78	21.32	5 040	53 361.45	36.61	8 370	94 728.05	47.18
1 720	15 556.49	21.38	5 050	53 481.73	36.64	8 380	94 855.63	47.21
1 730	15 661.29	21.44	5 060	53 602.03	36.68	8 390	94 983.23	47.24
1 740	15 766.18	21.50	5 070	53 722.36	36.72	8 400	95 110.85	47.26
1 750	15 871.15	21.56	5 080	53 842.71	36.75	8 410	95 238.49	47.29
1 760	15 976.19	21.63	5 090	53 963.10	36.79	8 420	95 366.14	47.32
1 770	16 081.33	21.69	5 100	54 083.52	36.83	8 430	95 493.81	47.35
1 780	16 186.54	21.75	5 110	54 203.96	36.86	8 440	95 621.49	47.38
1 790	16 291.83	21.81	5 120	54 324.43	36.90	8 450	95 749.19	47.40
1 800	16 397.21	21.87	5 130	54 444.93	36.93	8 460	95 876.91	47.43
1 810	16 502.66	21.93	5 140	54 565.46	36.97	8 470	96 004.65	47.46
1 820	16 608.19	21.99	5 150	54 686.01	37.01	8 480	96 132.40	47.49
1 830	16 713.81	22.05	5 160	54 806.60	37.04	8 490	96 260.18	47.52
1 840	16 819.50	22.11	5 170	54 927.21	37.08	8 500	96 387.96	47.54
1 850	16 925.27	22.17	5 180	55 047.85	37.11	8 510	96 515.77	47.57
1 860	17 031.12	22.23	5 190	55 168.52	37.15	8 520	96 643.59	47.60
1 870	17 137.04	22.29	5 200	55 289.21	37.18	8 530	96 771.43	47.63
1 880	17 243.05	22.35	5 210	55 409.93	37.22	8 540	96 899.28	47.66
1 890	17 349.13	22.41	5 220	55 530.68	37.26	8 550	97 027.16	47.68
1 900	17 455.28	22.47	5 230	55 651.46	37.29	8 560	97 155.05	47.71

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
1 910	17 561.51	22.53	5 240	55 772.27	37.33	8 570	97 282.95	47.74
1 920	17 667.82	22.59	5 250	55 893.10	37.36	8 580	97 410.88	47.77
1 930	17 774.20	22.65	5 260	56 013.96	37.40	8 590	97 538.82	47.80
1 940	17 880.66	22.71	5 270	56 134.85	37.43	8 600	97 666.77	47.82
1 950	17 987.19	22.76	5 280	56 255.77	37.47	8 610	97 794.75	47.85
1 960	18 093.79	22.82	5 290	56 376.71	37.51	8 620	97 922.74	47.88
1 970	18 200.47	22.88	5 300	56 497.68	37.54	8 630	98 050.74	47.91
1 980	18 307.22	22.94	5 310	56 618.68	37.58	8 640	98 178.77	47.93
1 990	18 414.05	23.00	5 320	56 739.70	37.61	8 650	98 306.81	47.96
2 000	18 520.94	23.05	5 330	56 860.75	37.65	8 660	98 434.86	47.99
2 010	18 627.91	23.11	5 340	56 981.83	37.68	8 670	98 562.94	48.02
2 020	18 734.95	23.17	5 350	57 102.94	37.72	8 680	98 691.03	48.05
2 030	18 842.07	23.23	5 360	57 224.07	37.75	8 690	98 819.14	48.07
2 040	18 949.25	23.28	5 370	57 345.23	37.79	8 700	98 947.26	48.10
2 050	19 056.50	23.34	5 380	57 466.41	37.82	8 710	99 075.40	48.13
2 060	19 163.83	23.40	5 390	57 587.63	37.86	8 720	99 203.56	48.16
2 070	19 271.22	23.45	5 400	57 708.87	37.89	8 730	99 331.73	48.18
2 080	19 378.68	23.51	5 410	57 830.13	37.93	8 740	99 459.92	48.21
2 090	19 486.22	23.57	5 420	57 951.43	37.96	8 750	99 588.13	48.24
2 100	19 593.82	23.62	5 430	58 072.75	38.00	8 760	99 716.35	48.27
2 110	19 701.49	23.68	5 440	58 194.09	38.03	8 770	99 844.59	48.29
2 120	19 809.23	23.74	5 450	58 315.47	38.07	8 780	99 972.84	48.32
2 130	19 917.03	23.79	5 460	58 436.87	38.10	8 790	100 101.11	48.35
2 140	20 024.91	23.85	5 470	58 558.29	38.14	8 800	100 229.40	48.38
2 150	20 132.85	23.90	5 480	58 679.74	38.17	8 810	100 357.71	48.40
2 160	20 240.86	23.96	5 490	58 801.22	38.21	8 820	100 486.03	48.43
2 170	20 348.93	24.01	5 500	58 922.73	38.24	8 830	100 614.37	48.46
2 180	20 457.07	24.07	5 510	59 044.26	38.28	8 840	100 742.72	48.49
2 190	20 565.28	24.13	5 520	59 165.82	38.31	8 850	100 871.09	48.51
2 200	20 673.56	24.18	5 530	59 287.40	38.35	8 860	100 999.48	48.54
2 210	20 781.89	24.24	5 540	59 409.01	38.38	8 870	101 127.88	48.57
2 220	20 890.30	24.29	5 550	59 530.64	38.42	8 880	101 256.30	48.60
2 230	20 998.77	24.34	5 560	59 652.31	38.45	8 890	101 384.73	48.62
2 240	21 107.30	24.40	5 570	59 773.99	38.49	8 900	101 513.19	48.65
2 250	21 215.90	24.45	5 580	59 895.71	38.52	8 910	101 641.65	48.68
2 260	21 324.56	24.51	5 590	60 017.45	38.55	8 920	101 770.14	48.71
2 270	21 433.29	24.56	5 600	60 139.21	38.59	8 930	101 898.64	48.73
2 280	21 542.08	24.62	5 610	60 261.00	38.62	8 940	102 027.15	48.76
2 290	21 650.93	24.67	5 620	60 382.82	38.66	8 950	102 155.69	48.79
2 300	21 759.85	24.72	5 630	60 504.66	38.69	8 960	102 284.24	48.81
2 310	21 868.83	24.78	5 640	60 626.53	38.73	8 970	102 412.80	48.84
2 320	21 977.87	24.83	5 650	60 748.42	38.76	8 980	102 541.38	48.87
2 330	22 086.97	24.89	5 660	60 870.34	38.80	8 990	102 669.98	48.90
2 340	22 196.14	24.94	5 670	60 992.28	38.83	9 000	102 798.59	48.92
2 350	22 305.36	24.99	5 680	61 114.25	38.86	9 010	102 927.22	48.95
2 360	22 414.65	25.04	5 690	61 236.25	38.90	9 020	103 055.86	48.98
2 370	22 524.00	25.10	5 700	61 358.27	38.93	9 030	103 184.52	49.00
2 380	22 633.41	25.15	5 710	61 480.31	38.97	9 040	103 313.20	49.03
2 390	22 742.88	25.20	5 720	61 602.38	39.00	9 050	103 441.89	49.06

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
2 400	22 852.41	25.26	5 730	61 724.48	39.03	9 060	103 570.60	49.09
2 410	22 962.00	25.31	5 740	61 846.60	39.07	9 070	103 699.33	49.11
2 420	23 071.65	25.36	5 750	61 968.75	39.10	9 080	103 828.07	49.14
2 430	23 181.36	25.41	5 760	62 090.92	39.14	9 090	103 956.82	49.17
2 440	23 291.13	25.47	5 770	62 213.12	39.17	9 100	104 085.60	49.19
2 450	23 400.96	25.52	5 780	62 335.34	39.20	9 110	104 214.39	49.22
2 460	23 510.85	25.57	5 790	62 457.58	39.24	9 120	104 343.19	49.25
2 470	23 620.79	25.62	5 800	62 579.86	39.27	9 130	104 472.01	49.28
2 480	23 730.80	25.67	5 810	62 702.15	39.31	9 140	104 600.84	49.30
2 490	23 840.86	25.73	5 820	62 824.47	39.34	9 150	104 729.70	49.33
2 500	23 950.98	25.78	5 830	62 946.82	39.37	9 160	104 858.56	49.36
2 510	24 061.16	25.83	5 840	63 069.19	39.41	9 170	104 987.45	49.38
2 520	24 171.39	25.88	5 850	63 191.59	39.44	9 180	105 116.34	49.41
2 530	24 281.68	25.93	5 860	63 314.01	39.48	9 190	105 245.26	49.44
2 540	24 392.03	25.98	5 870	63 436.45	39.51	9 200	105 374.19	49.46
2 550	24 502.44	26.03	5 880	63 558.92	39.54	9 210	105 503.13	49.49
2 560	24 612.90	26.09	5 890	63 681.41	39.58	9 220	105 632.10	49.52
2 570	24 723.42	26.14	5 900	63 803.93	39.61	9 230	105 761.07	49.54
2 580	24 834.00	26.19	5 910	63 926.47	39.64	9 240	105 890.07	49.57
2 590	24 944.63	26.24	5 920	64 049.04	39.68	9 250	106 019.07	49.60
2 600	25 055.32	26.29	5 930	64 171.63	39.71	9 260	106 148.10	49.62
2 610	25 166.06	26.34	5 940	64 294.25	39.74	9 270	106 277.14	49.65
2 620	25 276.86	26.39	5 950	64 416.89	39.78	9 280	106 406.19	49.68
2 630	25 387.71	26.44	5 960	64 539.55	39.81	9 290	106 535.26	49.71
2 640	25 498.62	26.49	5 970	64 662.24	39.84	9 300	106 664.35	49.73
2 650	25 609.58	26.54	5 980	64 784.95	39.88	9 310	106 793.45	49.76
2 660	25 720.59	26.59	5 990	64 907.69	39.91	9 320	106 922.57	49.79
2 670	25 831.66	26.64	6 000	65 030.45	39.94	9 330	107 051.70	49.81
2 680	25 942.79	26.69	6 010	65 153.23	39.98	9 340	107 180.85	49.84
2 690	26 053.97	26.74	6 020	65 276.04	40.01	9 350	107 310.01	49.87
2 700	26 165.20	26.79	6 030	65 398.88	40.04	9 360	107 439.19	49.89
2 710	26 276.48	26.84	6 040	65 521.73	40.08	9 370	107 568.38	49.92
2 720	26 387.82	26.89	6 050	65 644.61	40.11	9 380	107 697.59	49.95
2 730	26 499.21	26.94	6 060	65 767.52	40.14	9 390	107 826.82	49.97
2 740	26 610.66	26.99	6 070	65 890.45	40.18	9 400	107 956.06	50.00
2 750	26 722.16	27.04	6 080	66 013.40	40.21	9 410	108 085.31	50.03
2 760	26 833.70	27.09	6 090	66 136.37	40.24	9 420	108 214.59	50.05
2 770	26 945.31	27.14	6 100	66 259.37	40.28	9 430	108 343.87	50.08
2 780	27 056.96	27.18	6 110	66 382.40	40.31	9 440	108 473.17	50.10
2 790	27 168.67	27.23	6 120	66 505.44	40.34	9 450	108 602.49	50.13
2 800	27 280.42	27.28	6 130	66 628.51	40.37	9 460	108 731.82	50.16
2 810	27 392.23	27.33	6 140	66 751.61	40.41	9 470	108 861.17	50.18
2 820	27 504.09	27.38	6 150	66 874.72	40.44	9 480	108 990.53	50.21
2 830	27 616.00	27.43	6 160	66 997.86	40.47	9 490	109 119.91	50.24
2 840	27 727.96	27.48	6 170	67 121.03	40.51	9 500	109 249.30	50.26
2 850	27 839.98	27.52	6 180	67 244.22	40.54	9 510	109 378.71	50.29
2 860	27 952.04	27.57	6 190	67 367.43	40.57	9 520	109 508.14	50.32
2 870	28 064.15	27.62	6 200	67 490.66	40.60	9 530	109 637.57	50.34
2 880	28 176.32	27.67	6 210	67 613.92	40.64	9 540	109 767.03	50.37

m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m	m	$E[x_m]$	σ_m
2 890	28 288.53	27.72	6 220	67 737.20	40.67	9 550	109 896.50	50.40
2 900	28 400.80	27.77	6 230	67 860.50	40.70	9 560	110 025.98	50.42
2 910	28 513.11	27.81	6 240	67 983.83	40.74	9 570	110 155.48	50.45
2 920	28 625.47	27.86	6 250	68 107.18	40.77	9 580	110 284.99	50.48
2 930	28 737.89	27.91	6 260	68 230.55	40.80	9 590	110 414.52	50.50
2 940	28 850.35	27.96	6 270	68 353.95	40.83	9 600	110 544.07	50.53
2 950	28 962.86	28.00	6 280	68 477.37	40.87	9 610	110 673.63	50.55
2 960	29 075.42	28.05	6 290	68 600.81	40.90	9 620	110 803.20	50.58
2 970	29 188.03	28.10	6 300	68 724.27	40.93	9 630	110 932.79	50.61
2 980	29 300.68	28.15	6 310	68 847.76	40.96	9 640	111 062.39	50.63
2 990	29 413.39	28.19	6 320	68 971.27	41.00	9 650	111 192.01	50.66
3 000	29 526.14	28.24	6 330	69 094.81	41.03	9 660	111 321.65	50.69
3 010	29 638.94	28.29	6 340	69 218.36	41.06	9 670	111 451.30	50.71
3 020	29 751.79	28.33	6 350	69 341.94	41.09	9 680	111 580.96	50.74
3 030	29 864.69	28.38	6 360	69 465.54	41.13	9 690	111 710.64	50.76
3 040	29 977.63	28.43	6 370	69 589.17	41.16	9 700	111 840.33	50.79
3 050	30 090.63	28.47	6 380	69 712.81	41.19	9 710	111 970.04	50.82
3 060	30 203.67	28.52	6 390	69 836.48	41.22	9 720	112 099.76	50.84
3 070	30 316.75	28.57	6 400	69 960.18	41.25	9 730	112 229.50	50.87
3 080	30 429.89	28.61	6 410	70 083.89	41.29	9 740	112 359.26	50.90
3 090	30 543.07	28.66	6 420	70 207.63	41.32	9 750	112 489.02	50.92
3 100	30 656.29	28.71	6 430	70 331.39	41.35	9 760	112 618.81	50.95
3 110	30 769.57	28.75	6 440	70 455.17	41.38	9 770	112 748.60	50.97
3 120	30 882.89	28.80	6 450	70 578.98	41.42	9 780	112 878.42	51.00
3 130	30 996.25	28.85	6 460	70 702.80	41.45	9 790	113 008.24	51.03
3 140	31 109.67	28.89	6 470	70 826.65	41.48	9 800	113 138.09	51.05
3 150	31 223.12	28.94	6 480	70 950.52	41.51	9 810	113 267.94	51.08
3 160	31 336.63	28.98	6 490	71 074.42	41.54	9 820	113 397.81	51.10
3 170	31 450.18	29.03	6 500	71 198.33	41.58	9 830	113 527.70	51.13
3 180	31 563.77	29.08	6 510	71 322.27	41.61	9 840	113 657.60	51.16
3 190	31 677.41	29.12	6 520	71 446.23	41.64	9 850	113 787.51	51.18
3 200	31 791.10	29.17	6 530	71 570.21	41.67	9 860	113 917.44	51.21
3 210	31 904.83	29.21	6 540	71 694.22	41.70	9 870	114 047.39	51.23
3 220	32 018.60	29.26	6 550	71 818.25	41.74	9 880	114 177.35	51.26
3 230	32 132.42	29.30	6 560	71 942.29	41.77	9 890	114 307.32	51.29
3 240	32 246.29	29.35	6 570	72 066.37	41.80	9 900	114 437.31	51.31
3 250	32 360.20	29.39	6 580	72 190.46	41.83	9 910	114 567.31	51.34
3 260	32 474.15	29.44	6 590	72 314.57	41.86	9 920	114 697.33	51.36
3 270	32 588.15	29.48	6 600	72 438.71	41.89	9 930	114 827.36	51.39
3 280	32 702.19	29.53	6 610	72 562.87	41.93	9 940	114 957.41	51.42
3 290	32 816.28	29.57	6 620	72 687.05	41.96	9 950	115 087.47	51.44
3 300	32 930.41	29.62	6 630	72 811.25	41.99	9 960	115 217.55	51.47
3 310	33 044.58	29.66	6 640	72 935.47	42.02	9 970	115 347.64	51.49
3 320	33 158.80	29.71	6 650	73 059.72	42.05	9 980	115 477.74	51.52
3 330	33 273.06	29.75	6 660	73 183.99	42.08	9 990	115 607.86	51.54
						10 000	115 737.99	51.57

B AMOUNT OF CIPHERTEXT BITS REQUIRED FOR LEMPEL-ZIV ATTACK TO BE SUCCESSFUL

p	Min Bits	p	Min Bits	p	Min Bits
0.4	637	0.43	4097	0.46	13701
0.401	765	0.431	4166	0.461	14076
0.402	919	0.432	4272	0.462	15512
0.403	1022	0.433	4410	0.463	15881
0.404	1180	0.434	4586	0.464	16815
0.405	1236	0.435	4752	0.465	17312
0.406	1345	0.436	4924	0.466	18552
0.407	1568	0.437	5167	0.467	19379
0.408	1619	0.438	5356	0.468	21096
0.409	1723	0.439	5435	0.469	22184
0.41	1856	0.44	5592	0.47	23565
0.411	2013	0.441	5671	0.471	25403
0.412	2104	0.442	6067	0.472	27807
0.413	2196	0.443	6332	0.473	30103
0.414	2299	0.444	6725	0.474	31780
0.415	2359	0.445	6783	0.475	35825
0.416	2459	0.446	7009	0.476	39424
0.417	2602	0.447	7530	0.477	42338
0.418	2711	0.448	7794	0.478	45450
0.419	2755	0.449	8082	0.479	50134
0.42	2959	0.45	8333	0.48	53265
0.421	3079	0.451	8566	0.481	57489
0.422	3145	0.452	8841	0.482	62496
0.423	3203	0.453	9641		
0.424	3251	0.454	10589		
0.425	3334	0.455	11028		
0.426	3445	0.456	11462		
0.427	3524	0.457	11831		
0.428	3899	0.458	12427		
0.429	3997	0.459	13087		

C NUMBER OF DERIVATIVES FOR BINARY DERIVATIVE ATTACK TO SUCCEED

No Bits \ p	0.4	0.405	0.41	0.415	0.42	0.425	0.43	0.435	0.44	0.445	0.45	0.455	0.46	0.465	0.47
3 072	8	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6 144	3	6	8	8	11	20	17	19	18	17	x	x	x	x	x
9 216	2	3	4	4	5	10	12	12	16	15	19	x	x	x	x
12 288	1	1	2	3	3	5	6	7	11	10	15	22	20	22	x
15 360	0	1	1	2	2	3	4	5	9	9	12	19	17	22	x
18 432	0	0	1	1	2	3	3	4	8	8	11	16	14	20	x
21 504	0	0	0	1	1	2	3	3	6	6	9	12	13	20	x
24 576	0	0	0	0	1	2	2	3	5	5	8	11	11	17	20
27 648	0	0	0	0	0	1	2	3	4	5	7	9	11	16	17
30 720	0	0	0	0	0	1	2	2	4	5	6	9	10	12	17
33 792	0	0	0	0	0	0	1	2	4	4	6	8	9	12	15
36 864	0	0	0	0	0	0	1	2	3	4	5	8	9	11	15
39 936	0	0	0	0	0	0	1	2	3	4	5	7	8	10	15
43 008	0	0	0	0	0	0	0	2	2	3	5	7	8	10	13
46 080	0	0	0	0	0	0	0	2	2	3	4	6	7	9	13
49 152	0	0	0	0	0	0	0	2	2	3	4	5	7	9	13
52 224	0	0	0	0	0	0	0	2	2	3	4	5	6	9	12
55 296	0	0	0	0	0	0	0	2	1	3	4	5	6	9	12
58 368	0	0	0	0	0	0	0	1	1	2	4	5	6	8	11
61 440	0	0	0	0	0	0	0	1	1	2	3	4	5	8	10
64 512	0	0	0	0	0	0	0	1	1	2	3	4	5	7	9
67 584	0	0	0	0	0	0	0	1	0	2	3	3	5	7	9
70 656	0	0	0	0	0	0	0	1	0	2	3	3	5	6	9
73 728	0	0	0	0	0	0	0	1	0	2	3	3	5	6	8
76 800	0	0	0	0	0	0	0	0	0	1	3	4	5	6	8
79 872	0	0	0	0	0	0	0	0	0	1	2	3	4	6	8
82 944	0	0	0	0	0	0	0	0	0	0	2	3	4	6	8
86 016	0	0	0	0	0	0	0	0	0	0	2	3	4	6	8
89 088	0	0	0	0	0	0	0	0	0	0	2	3	4	6	8
92 160	0	0	0	0	0	0	0	0	0	0	2	3	4	5	8
95 232	0	0	0	0	0	0	0	0	0	0	2	3	4	5	8
98 304	0	0	0	0	0	0	0	0	0	0	1	3	4	5	8
101 376	0	0	0	0	0	0	0	0	0	0	1	3	4	5	8
104 448	0	0	0	0	0	0	0	0	0	0	1	3	4	5	8
107 520	0	0	0	0	0	0	0	0	0	0	1	3	4	5	8
110 592	0	0	0	0	0	0	0	0	0	0	1	3	4	5	8
113 664	0	0	0	0	0	0	0	0	0	0	1	3	3	5	7
116 736	0	0	0	0	0	0	0	0	0	0	1	2	3	4	7

No Bits \ p	0.4	0.405	0.41	0.415	0.42	0.425	0.43	0.435	0.44	0.445	0.45	0.455	0.46	0.465	0.47
119 808	0	0	0	0	0	0	0	0	0	0	1	2	3	4	7
122 880	0	0	0	0	0	0	0	0	0	0	1	2	3	4	7
125 952	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
129 024	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
132 096	0	0	0	0	0	0	0	0	0	0	1	2	3	4	7
135 168	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
138 240	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
141 312	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
144 384	0	0	0	0	0	0	0	0	0	0	1	2	3	4	6
147 456	0	0	0	0	0	0	0	0	0	0	0	1	3	4	6
150 528	0	0	0	0	0	0	0	0	0	0	0	1	3	4	6
153 600	0	0	0	0	0	0	0	0	0	0	0	1	3	4	6
156 672	0	0	0	0	0	0	0	0	0	0	0	1	2	4	6
159 744	0	0	0	0	0	0	0	0	0	0	0	1	2	4	6

D EXPECTED NUMBER OF PARITY EQUATIONS

Choosing $e = l - B$ the tables (Table 6.1 to Table 1.4) below represent the amount of equations one can expect to find for $2 \le e \le 49$. For the instance of $e = 49$, to find any equations, one needs to search at least $N = 2^{26}$ columns, which amounts to $\approx 2^{51}$ operations, a figure close to impossible. The search for parity equations can however be performed in parallel, potentially allowing this amount of columns to be searched. The values where calculated using (4.56), repeated below.

$$E[equ] = \frac{1}{2} \cdot N \cdot (N-1) \cdot \frac{1}{2^{l-B}} \tag{1.1}$$

Table 6.1 Expected no. equations in $N = 2^1 \ldots 2^{12}$ *for* $e = 2 \ldots 22 \mid e = l - B$

e \ N	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
2	0	1	7	30	124	504	2032	8160	32704	130944	524032	2096640
3	0	0	3	15	62	252	1016	4080	16352	65472	262016	1048320
4	0	0	1	7	31	126	508	2040	8176	32736	131008	524160
5	0	0	0	3	15	63	254	1020	4088	16368	65504	262080
6	0	0	0	1	7	31	127	510	2044	8184	32752	131040
7	0	0	0	0	3	15	63	255	1022	4092	16376	65520
8	0	0	0	0	1	7	31	127	511	2046	8188	32760
9	0	0	0	0	0	3	15	63	255	1023	4094	16380
10	0	0	0	0	0	1	7	31	127	511	2047	8190
11	0	0	0	0	0	0	3	15	63	255	1023	4095
12	0	0	0	0	0	0	1	7	31	127	511	2047
13	0	0	0	0	0	0	0	3	15	63	255	1023
14	0	0	0	0	0	0	0	1	7	31	127	511
15	0	0	0	0	0	0	0	0	3	15	63	255
16	0	0	0	0	0	0	0	0	1	7	31	127
17	0	0	0	0	0	0	0	0	0	3	15	63
18	0	0	0	0	0	0	0	0	0	1	7	31
19	0	0	0	0	0	0	0	0	0	0	3	15
20	0	0	0	0	0	0	0	0	0	0	1	7
21	0	0	0	0	0	0	0	0	0	0	0	3
22	0	0	0	0	0	0	0	0	0	0	0	1

Table 1.2 *Expected no. equations in* $N = 2^{13} \ldots 2^{24}$ *for* $e = 2 \ldots 22 \mid e = l - B$

e \ N	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
3	4193792	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
4	2096896	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5	1048448	4194048	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
6	524224	2097024	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7	262112	1048512	4194176	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
8	131056	524256	2097088	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
9	65528	262128	1048544	4194240	n/a	n/a	n/a	n/a	n/a	n/a	n/a
10	32764	131064	524272	2097120	n/a	n/a	n/a	n/a	n/a	n/a	n/a
11	16382	65532	262136	1048560	4194272	n/a	n/a	n/a	n/a	n/a	n/a
12	8191	32766	131068	524280	2097136	n/a	n/a	n/a	n/a	n/a	n/a
13	4095	16383	65534	262140	1048568	4194288	n/a	n/a	n/a	n/a	n/a
14	2047	8191	32767	131070	524284	2097144	n/a	n/a	n/a	n/a	n/a
15	1023	4095	16383	65535	262142	1048572	4194296	n/a	n/a	n/a	n/a
16	511	2047	8191	32767	131071	524286	2097148	n/a	n/a	n/a	n/a
17	255	1023	4095	16383	65535	262143	1048574	4194300	n/a	n/a	n/a
18	127	511	2047	8191	32767	131071	524287	2097150	n/a	n/a	n/a
19	63	255	1023	4095	16383	65535	262143	1048575	4194302	n/a	n/a
20	31	127	511	2047	8191	32767	131071	524287	2097151	n/a	n/a
21	15	63	255	1023	4095	16383	65535	262143	1048575	4194303	n/a
22	7	31	127	511	2047	8191	32767	131071	524287	2097151	n/a

Table 1.3 *Expected no. equations in* $N = 2^{13} \ldots 2^{24}$ *for* $e = 23 \ldots 49 \mid e = l - B$

e \ N	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
23	3	15	63	255	1023	4095	16383	65535	262143	1048575	4194303	n/a
24	1	7	31	127	511	2047	8191	32767	131071	524287	2097151	n/a
25	0	3	15	63	255	1023	4095	16383	65535	262143	1048575	4194303
26	0	1	7	31	127	511	2047	8191	32767	131071	524287	2097151
27	0	0	3	15	63	255	1023	4095	16383	65535	262143	1048575
28	0	0	1	7	31	127	511	2047	8191	32767	131071	524287
29	0	0	0	3	15	63	255	1023	4095	16383	65535	262143
30	0	0	0	1	7	31	127	511	2047	8191	32767	131071
31	0	0	0	0	3	15	63	255	1023	4095	16383	65535
32	0	0	0	0	1	7	31	127	511	2047	8191	32767
33	0	0	0	0	0	3	15	63	255	1023	4095	16383
34	0	0	0	0	0	1	7	31	127	511	2047	8191
35	0	0	0	0	0	0	3	15	63	255	1023	4095
36	0	0	0	0	0	0	1	7	31	127	511	2047
37	0	0	0	0	0	0	0	3	15	63	255	1023
38	0	0	0	0	0	0	0	1	7	31	127	511

e \ N	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
39	0	0	0	0	0	0	0	0	3	15	63	255
40	0	0	0	0	0	0	0	0	1	7	31	127
41	0	0	0	0	0	0	0	0	0	3	15	63
42	0	0	0	0	0	0	0	0	0	1	7	31
43	0	0	0	0	0	0	0	0	0	0	3	15
44	0	0	0	0	0	0	0	0	0	0	1	7
45	0	0	0	0	0	0	0	0	0	0	0	3
46	0	0	0	0	0	0	0	0	0	0	0	1
47	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0

Table 1.4 *Expected no. equations in* $N = 2^{25}...2^{30}$ *for* $e = 23...49 \mid e = l - B$

e \ N	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}
23	n/a	n/a	n/a	n/a	n/a	n/a
24	n/a	n/a	n/a	n/a	n/a	n/a
25	n/a	n/a	n/a	n/a	n/a	n/a
26	n/a	n/a	n/a	n/a	n/a	n/a
27	4194303	n/a	n/a	n/a	n/a	n/a
28	2097151	n/a	n/a	n/a	n/a	n/a
29	1048575	4194303	n/a	n/a	n/a	n/a
30	524287	2097151	n/a	n/a	n/a	n/a
31	262143	1048575	4194303	n/a	n/a	n/a
32	131071	524287	2097151	n/a	n/a	n/a
33	65535	262143	1048575	4194303	n/a	n/a
34	32767	131071	524287	2097151	n/a	n/a
35	16383	65535	262143	1048575	4194303	n/a
36	8191	32767	131071	524287	2097151	n/a
37	4095	16383	65535	262143	1048575	4194303
38	2047	8191	32767	131071	524287	2097151
39	1023	4095	16383	65535	262143	1048575
40	511	2047	8191	32767	131071	524287
41	255	1023	4095	16383	65535	262143
42	127	511	2047	8191	32767	131071
43	63	255	1023	4095	16383	65535
44	31	127	511	2047	8191	32767
45	15	63	255	1023	4095	16383
46	7	31	127	511	2047	8191
47	3	15	63	255	1023	4095
48	1	7	31	127	511	2047
49	0	3	15	63	255	1023

Table 1.5 *Expected no. equations in* $N = 2^{31} \ldots 2^{36}$ *for* $e = 23 \ldots 49 \mid e = l - B$

e \ N	2^{31}	2^{32}	2^{33}	2^{34}	2^{35}	2^{36}
23	n/a	n/a	n/a	n/a	n/a	n/a
24	n/a	n/a	n/a	n/a	n/a	n/a
25	n/a	n/a	n/a	n/a	n/a	n/a
26	n/a	n/a	n/a	n/a	n/a	n/a
27	n/a	n/a	n/a	n/a	n/a	n/a
28	n/a	n/a	n/a	n/a	n/a	n/a
29	n/a	n/a	n/a	n/a	n/a	n/a
30	n/a	n/a	n/a	n/a	n/a	n/a
31	n/a	n/a	n/a	n/a	n/a	n/a
32	n/a	n/a	n/a	n/a	n/a	n/a
33	n/a	n/a	n/a	n/a	n/a	n/a
34	n/a	n/a	n/a	n/a	n/a	n/a
35	n/a	n/a	n/a	n/a	n/a	n/a
36	n/a	n/a	n/a	n/a	n/a	n/a
37	n/a	n/a	n/a	n/a	n/a	n/a
38	n/a	n/a	n/a	n/a	n/a	n/a
39	4194303	n/a	n/a	n/a	n/a	n/a
40	2097151	n/a	n/a	n/a	n/a	n/a
41	1048575	4194303	n/a	n/a	n/a	n/a
42	524287	2097151	n/a	n/a	n/a	n/a
43	262143	1048575	4194303	n/a	n/a	n/a
44	131071	524287	2097151	n/a	n/a	n/a
45	65535	262143	1048575	4194303	n/a	n/a
46	32767	131071	524287	2097151	n/a	n/a
47	16383	65535	262143	1048575	4194303	n/a
48	8191	32767	131071	524287	2097151	n/a
49	4095	16383	65535	262143	1048575	4194303

E AVERAGE NUMBER OF PARITY EQUATIONS REQUIRED BY FAST CORRELATION ATTACK

Table 1.6 The number of required parity equations as a function of channel probability for B=2

p	*Equ*	*p*	*Equ*	*p*	*Equ*
0.1	5	0.3	142	0.473	343589
0.11	7	0.31	196	0.474	458711
0.12	7	0.32	222	0.475	530267
0.13	10	0.33	222	0.476	612724
0.14	10	0.34	399	0.477	612724
0.15	10	0.35	399	0.478	708049
0.16	11	0.36	555	0.479	1092476
0.17	14	0.37	743	0.48	1262373
0.18	14	0.38	991	0.481	1458833
0.19	47	0.39	1193	0.482	1685983
0.2	47	0.4	1583	0.483	2252082
0.21	47	0.41	2869	0.484	2252082
0.22	62	0.42	4388		
0.23	62	0.43	9125		
0.24	62	0.44	16338		
0.25	87	0.45	29248		
0.26	87	0.46	70007		
0.27	94	0.47	222970		
0.28	142	0.471	222970		
0.29	142	0.472	297406		

F SELECTED DECIMATION FACTORS

Table 1.7 Decimation factors for LFSRs smaller than 64 bits

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
18	3	513	9
	3	3591	9
	3	4161	6
	7	12483	6
	19	29127	6
	73	37449	3
		87381	2
19	prime		
20	3	1025	10
	5	3075	10
	5	11275	10
	11	31775	10
	31	95325	10
	41	33825	5
		69905	4
		209715	4
		349525	2
21		16513	7
		299593	3
22	3	2049	11
	23	47127	11
	89	182361	11
	683	1398101	2
23	47	None	
	178481		
24	3	4097	12
	3	12291	12
	5	20485	12
	7	28679	12
	13	36873	12
	17	53261	12
	241	61455	12
		86037	12
		143395	12
		159783	12
		184365	12
		258111	12
		372827	12
		430185	12
		479349	12

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
		1290555	12
		65793	8
		197379	8
		328965	8
		986895	8
		266305	6
		798915	6
		1864135	6
		1118481	4
		3355443	4
		2396745	3
		5592405	2
25	31	1082401	5
	601		
	1801		
26	8191	8193	13
	3	22369621	2
	2731		
27	7	262657	9
	73	1838599	9
	262657	19173961	3
		134217727	1
28	3	16385	14
	43	49155	14
	127	704555	14
	5	2080895	14
	29	6242685	14
	113	2113665	7
		17895697	4
		53687091	4
		89478485	2
29	233	None	
	1103		
	2089		
30	3	32769	15
	3	229383	15
	7	1015839	15
	11	4948119	15
	31	7110873	15
	151	1049601	10
	331	3148803	10
		11545611	10
		32537631	10
		97612893	10
		17043521	6
		51130563	6
		119304647	6
		34636833	5

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
		153391689	3
		357913941	2
31	prime		
32	3	65537	16
	5	196611	16
	17	327685	16
	257	983055	16
	65537	1114129	16
		3342387	16
		5570645	16
		16711935	16
		16843009	8
		50529027	8
		84215045	8
		252645135	8
		286331153	4
		858993459	4
		1431655765	2
33	7	4196353	11
	23	96516119	11
	89	373475417	11
	599479	1227133513	3
34	3	131073	17
	43691		
	131071		
35	31	270549121	7
	71	1108378657	5
	127		
	122921		
36	3	262145	18
	3	786435	18
	3	1835015	18
	5	2359305	18
	7	4980755	18
	13	5505045	18
	19	7077915	18
	37	14942265	18
	73	16515135	18
	109	19136585	18
		34865285	18
		44826795	18
		57409755	18
		104595855	18
		133956095	18
		172229265	18
		313787565	18
		363595115	18
		401868285	18

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
		516687795	18
		1205604855	18
		16781313	12
		16781313	12
		50343939	12
		83906565	12
		117469191	12
		151031817	12
		218157069	12
		251719695	12
		352407573	12
		587345955	12
		654471207	12
		755159085	12
		1057222719	12
		1527099483	12
		1762037865	12
		1963413621	12
		134480385	9
		941362695	9
		1090785345	6
37	223	None	
	616318177		
38	3	524289	19
	174763		
	524287		
39	7	67117057	13
	79		
	8191		
	121369		
40	3	1048577	20
	5	3145731	20
	5	5242885	20
	11	5242885	20
	31	11534347	20
	41	15728655	20
	17	26214425	20
	61681	32505887	20
		34603041	20
		42991657	20
		57671735	20
		78643275	20
		97517661	20
		128974971	20
		162529435	20
		173015205	20
		214958285	20
		288358675	20

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
		357564757	20
		472908227	20
		487588305	20
		644874855	20
		812647175	20
		865076025	20
		1072694271	20
		1332741367	20
		1418724681	20
		1787823785	20
		1074791425	10
41	13367	None	
	164511353		
42	3	2097153	21
	3	14680071	21
	7	102760497	21
	7	266338431	21
	43	706740561	21
	127	1864369017	21
	337	268451841	14
	5419	805355523	14
43	431	None	
	9719		
	2099863		
44	3	4194305	22
	5	12582915	22
	23	96469015	22
	89	289407045	22
	397	373293145	22
	683	1119879435	22
	2113		
45	7	1073774593	15
	31		
	73		
	151		
	631		
	23311		
46	3	8388609	23
	47	394264623	23
	178481		
	2796203		
47	2351	None	
	4513		
	13264529		
48	3	16777217	24
	3	50331651	24
	5	83886085	24
	7	117440519	24

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
	13	150994953	24
	17	218103821	24
	241	251658255	24
	97	285212689	24
	257	352321557	24
	673	587202595	24
		654311463	24
		754974765	24
		855638067	24
		1056964671	24
		1090519105	24
		1426063445	24
		1526726747	24
		1761607785	24
		1962934389	24
		1996488823	24
49	127	None below 2^31	
	270549121		
50	3	33554433	25
	11	1040187423	25
	31		
	251		
	601		
	1801		
	4051		
51	7	None below 2^31	
	103		
	2143		
	11119		
	131071		
52	3	67108865	26
	5	201326595	26
	53		
	157		
	1613		
	2731		
	8191		
53	6361	None below 2^31	
	69431		
	20394401		
54	3	134217729	27
	3	939524103	27
	3		
	3		
	7		
	19		
	73		
	87211		

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
	262657		
55	23	None below 2^31	
	31		
	89		
	881		
	3191		
	201961		
56	3	268435457	28
	5	805306371	28
	17	1342177285	28
	29		
	43		
	113		
	127		
	15790321		
57	7	None below 2^31	
	32377		
	524287		
	1212847		
58	3	536870913	29
	59		
	233		
	1103		
	2089		
	3033169		
59	179951	None below 2^31	
	3203431780337		
60	3	1073741825	30
	3		
	5		
	5		
	7		
	11		
	13		
	31		
	41		
	61		
	151		
	331		
	1321		
61	prime		
62	3	None below 2^31	
	715827883		
	2147483647		
63	7	None below 2^31	
	7		

LFSR SIZE L	Factors of $2^L - 1$	Decimation Factor D	Decimated LFSR Size L^*
	73		
	127		
	337		
	92737		
	649657		
64	3	None below 2^31	
	5		
	17		
	257		
	641		
	65537		
	6700417		