UNIVERSITY OF BERGEN

Grein
A New Non-Linear Cryptoprimitive

by

Ole R. Thorsen

Thesis for the degree Master of Science

December 2013

in the

Faculty of Mathematics and Natural Sciences

Department of Informatics

Acknowledgements

I want to thank my supervisor Tor Helleseth for all his help during the writing of this thesis. Further, I wish to thank the Norwegian National Security Authority, for giving me access to their Grein cryptosystem.
I also wish to thank all my colleagues at the Selmer Centre, for all the inspiring discussions. Most of all I wish to thank prof. Matthew Parker for all his input, and my dear friends Stian, Mikal and Jørgen for their spellchecking, and socialising in the breaks.
Finally, I wish to thank my girlfriend, Therese, and my family, for their continuous support during the writing of this thesis. Without you, this would not have been possible.

Contents

Acknowledgements i

List of Figures iv

List of Tables v

1 Introduction 1

2 Cryptography 2
- 2.1 Classical Cryptography . . . 3
- 2.2 Modern Cryptography . . . 4

3 Stream Ciphers 5
- 3.1 Stream Cipher Fundamentals . . . 5
- 3.2 Classification of Stream Ciphers . . . 6
- 3.3 One-Time Pad . . . 7

4 Building Blocks 8
- 4.1 Boolean Functions . . . 8
  - 4.1.1 Cryptographic Properties . . . 10
- 4.2 Linear Feedback Shift Registers . . . 11
  - 4.2.1 The Recurrence Relation . . . 12
  - 4.2.2 The Matrix Method . . . 12
  - 4.2.3 Characteristic Polynomial . . . 13
  - 4.2.4 Period of a Sequence . . . 14
- 4.3 Linear Complexity . . . 16
- 4.4 The Berlekamp-Massey Algorithm . . . 16
- 4.5 Non-Linear Feedback Shift Registers . . . 17
  - 4.5.1 de Bruijn Sequences . . . 18
- 4.6 Filter Generators . . . 20
- 4.7 Irregular Clocking . . . 21
- 4.8 S-Boxes . . . 22

5 The Grain Cryptosystem 24
- 5.1 The eSTREAM Project . . . 24
- 5.2 Description . . . 24
- 5.3 Technical Details . . . 25
- 5.4 Key Initialization . . . 26

5.5 Throughput Rate . . . 27
5.6 Cryptanalysis . . . 28

6 The Grein Cryptosystem **29**
6.1 Description . . . 29
6.2 Paths Through a Tree . . . 29
6.3 The Tree . . . 30
6.4 The Filter Generators . . . 31
6.5 Computing the Path . . . 32
6.6 Irregular Clocking . . . 33
6.7 Feedback . . . 34
6.8 Plugging the Tree into Grein . . . 35
6.9 An Example . . . 36

7 A Modified Version of the Grein Cryptosystem **39**
7.1 de Bruijn Sequences Revisited . . . 39
7.2 Advantages of de Bruijn Sequences . . . 41
7.3 Implementation . . . 41
7.4 Statistical Tests . . . 42
7.5 Path Distributions . . . 43
7.6 S-box Distributions . . . 46
7.7 Feedback . . . 48

8 Conclusions **50**
8.1 Further Work . . . 50

A Tapping Matrices **51**

B Path Distributions **53**

C Statistical Test Results **55**

Bibliography **58**

List of Figures

2.1 The communication model 3

4.1 Feedback shift register with five registers 11
4.2 A Linear Feedback Shift Register 12
4.3 The LFSR from example 4.2.1 12
4.4 Pure cycling register 14
4.5 A non-linear feedback shift register 18
4.6 A filter generator 21
4.7 Irregularly clocked sequence generator 22
4.8 The alternating step generator 22

5.1 The Grain Stream Cipher 26
5.2 Key initialization of Grain 27
5.3 Grain producing two bits of output per clock 28

6.1 The Grein tree structure 30
6.2 The filter function used in Grein 32
6.3 The feedback to the tree 35
6.4 The Grein Stream Cipher 36

7.1 P-values of Grein-L 44
7.2 P-values of Grein-d 44
7.3 Histogram of P-values for Grein-L 45
7.4 Histogram of P-values for Grein-d 45
7.5 Plot of path distributions. Grein-L on top, Grein-d on the bottom. Probabilites along the x-axis, Hamming-weight along the y-axis. 47
7.6 S-box plots. Grein-L on the left, Grein-d on the right 48
7.7 How the feedback function f_2 is integrated 49

List of Tables

4.1 Truth table of function from (4.1.1) 9
4.2 Hamming distances for all affine functions 11
4.3 S-box in four variables 23

6.1 S_2: The PRINCE S-box 36

B.1 Probabilities of path distributions. p_l is the original, LFSR-based Grein, while p_d is the new, de Bruijn-based Grein 53

C.1 Statistical tests on Grein-L 56
C.2 Statistical tests on Grein-d 57
C.3 Statistical tests on LFSR of length 30 57

1

Introduction

In this thesis, we will study a new stream cipher, Grein, and a new cryptoprimitive used in this cipher.
The second chapter gives a brief introduction to cryptography in general.
The third chapter looks at stream ciphers in general, and explains the advantages and disadvantages of stream ciphers compared to block ciphers.
In the fourth chapter the most important building blocks used in stream ciphers are explained. The reader is excepted to know elementary abstract algebra, as much of the results in this chapter depend on it.
In the fifth chapter, the stream cipher Grain is introduced.
In chapter six, the new stream cipher, Grein, is introduced. Here, we look at the different components used in the cipher, and how they operate together.
In chapter seven, we introduce an alteration to the Grein cryptosystem, which hopefully have some advantages.

2

Cryptography

The main goal of cryptography, is to enable two parties, commonly called Alice and Bob, to communicate over an insecure channel, without their adversary, Eve, being able to understand what is being said. This channel could be a telephone line, a military radio, or the internet.

The basic model of cryptography is that the sender, Alice, has some *plaintext* that she wants to send. This is then *encrypted*, and the resulting *ciphertext* is sent to the receiver, Bob, who *decrypts* the ciphertext to get the plaintext.
If Eve is able to evasdrop, she should not be able to learn anything about what is in the message. See figure 2.1 for an illustration.

The following definition is taken from [30, p. 1]:

Definition 1. A *cryptosystem*, is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible plaintexts,
2. $\mathcal{C}$ is a finite set of possible ciphertexts,
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible keys,
4. For each $K \in \mathcal{K}$, there is an *encryption rule* $e_K : \mathcal{P} \to \mathcal{C}$ and $d_K : \mathcal{C} \to \mathcal{P}$ are functions such that $d_K(e_K(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

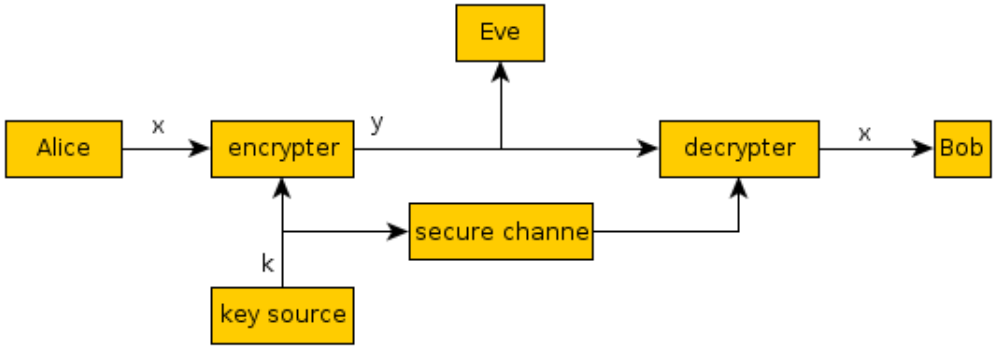

FIGURE 2.1: The communication model

The security of a cryptosystem is measured by how difficult it is to recover the plaintext. If the only way to find this plaintext is to do an exhaustive search of the keyspace, it is commonly agreed upon that the system is secure.

2.1 Classical Cryptography

The need for secure communication is not a new idea. Unreadable Hieroglyphs from ancient Egypt indicate that people have been wanting to hide information for millenia. One of the more well known cryptosystems is named after Julius Caesar, who is known to have sent his military orders using this system.

Definition 2. The Caesar cipher is one of the oldest, known, cryptosystems.
The set of plaintexts and ciphertexts are the same ($\mathcal{P} = \mathcal{C}$), which is all possible strings of characters from the latin alphabeth.
The keyspace is $\mathbb{Z}_+$, the group of all positive integers.
If the plaintext x is a string $x_1x_2 \dots x_n$ of characters, encryption is

$$e_K(x) : (x_i + K) \pmod{26}, 1 \leq i \leq n \tag{2.1}$$

Similarly, decryption is

$$d_K(x) : (x_i - K) \pmod{26}, 1 \leq i \leq n \tag{2.2}$$

Example 2.1.1. If our plaintext is "ATTACK AT DAWN" and our key $K = 4$, the encrypted message becomes "EXXEGO EX HEAR".

Fortunately for Caesar, most of his enemies were illiterate, and would not have been able to read his messages even if they were unencrypted.

There is however one major drawback with the caesar cipher, and that is the size of the keyspace. By having only 26 possible keys, and only 25 of these will actually work as a key (the first key, A, will return the plaintext.), even a novice cryptanalysist is able to test all keys within an hour. By using a computer, this time is reduced to seconds.

Seeing how easy it is to break the Caesar cipher, it is obvious that something more secure is needed.
Most, if not all, of the cryptography invented before 1900 is too easy to break.

2.2 Modern Cryptography

Up until the second world war, cryptography was not a very large research field, and as a result, most of the cryptosystems were easily broken.
In the second world war however, things changed.
All of the major countries involved had their own cryptosystems, with the most known being the German armys Enigma machine.
Some historians speculate that the war ended a couple of years earlier, because of the allied breaking of the cipher.
The major breaktroughs didn't come until the mid '70s, with *public key cryptography*.

Before the discovery of public key cryptography, both sender and receiver needed the same key to encrypt and decrypt messages. This key had to be sent over a secure channel (as seen in figure 2.1).

With public key cryptography, you have two keys, one *private key* and one *public key*. The public key can be distributed widely, so that anyone can send an encrypted message, but only those who know the private key are able to decrypt and read the message.

Public key cryptography depends on that some mathematical problems are hard to solve within reasonable time, such as factoring of large numbers (used in RSA [25]) and the discrete logarithm problem (used in ElGamal [11]). Public key cryptography is sometimes referred to as *asymmetrical* cryptography, because the key used for encryption is different from the key used for decryption.

In *symmetrical* cryptography, the same key is used for both encryption and decryption.

Symmetric ciphers are commonly divided into two further classes; *stream ciphers* and *block ciphers*, a distinction that is sometimes a bit confusing, as block ciphers can operate as stream ciphers.
In the next chapter we will look further into the designs of stream ciphers.

3

Stream Ciphers

Stream ciphers are one of the two major classes of cryptosystems, with block ciphers being the other major class.

Stream ciphers are mainly used in applications where speed of encryption / decryption are a priority. Another area where stream ciphers are prevalent, are systems with low hardware complexity, such as RFID tags. It is simply not possible to implement a block cipher like AES on an RFID tag, due to the constrained memory and power.

The most well known stream ciphers are, in no particular order, A5/1, used in the GSM cellular network, E0, used in Bluetooth and RC4, used in WEP.

Stream ciphers are also commonly used in military encryption, but few, or none details surrounding these stream ciphers exist, for natural reasons.

3.1 Stream Cipher Fundamentals

In July of 1919, Gilbert Vernam patented one of the most fundamental building blocks of stream ciphers, which we know today as the *XOR operation*, often denoted by $\oplus$.

It is usually said to operate on the two elements of $\mathbb{F}_2$, 0 and 1, such that

$$0 \oplus 0 = 1 \oplus 1 = 0$$

and

$$1 \oplus 0 = 0 \oplus 1 = 1.$$

This is clearly the same as addition in $\mathbb{F}_2$.

By representing the letters of the alphabeth as 5-bit groups ($2^5 = 32$ is the smallest sequence of bits that allows us to represent all 26 letters of the alphabeth), this enables us to encrypt messages of arbitrary length.

Example 3.1.1. Representing the letter M as 01101 and F as 00110, and then XORing these together, we get

$$\begin{aligned} &01101 \\ \oplus\, &00110 \\ = &01011, \end{aligned}$$

which is the letter K.

Because XOR is the same as addition over $\mathbb{F}_2$, adding the key to the ciphertext, we get the plaintext.

Example 3.1.2. XORing F with K, we get

$$\begin{aligned} &01011 \\ \oplus\, &00110 \\ = &01101, \end{aligned}$$

which is the letter M, that we started with.

3.2 Classification of Stream Ciphers

Stream ciphers can, in general, be divided into two classes.
These are *synchronous stream ciphers* and *self-synchronizing stream ciphers*.

Synchronous stream ciphers depend only on plain-text and key to produce the cipher-text.
If synchronization is lost, decryption from that point on becomes impossible.

Self-synchronizing stream ciphers on the other hand, depend on both the plain-text and the key, but also N of the previous cipher-text bits.
This enables a self-synchronizing stream cipher to regain synchronization if something should happen to it.

3.3 One-Time Pad

The One-time Pad (OTP), sometimes called the Vernam Cipher, after one of its inventors, was invented in 1917, and is the first example of a stream cipher using the XOR operation.

In the first version of the OTP, the key was read from a punched tape.
However, once the tape had been read, the cipher started again from the start, creating a loop.

A little later, Joseph Mauborgne realized that this was a potential weakness, and together with Vernam, he introduced randomness in to the key.

This was later proven by Shannon [28] to provide *perfect secrecy*.
Perfect secrecy means that, given the ciphertext, a cryptanalysist will gain no information about the plaintext, except its length.

There is, however, a major drawback with the OTP.
In order to guarantee perfect secrecy, a random key of the same length as the plaintext is needed, both for encryption and decryption.
For both sender and receiver to get a copy of the key, they either need to meet in person, or they need a secure channel to exchange the key.
However, if they have access to a secure channel, why not let all communication go through this channel?
Clearly, a more practical cryptosystem is needed.

4

Building Blocks

In order to build a cipher, we need a certain mathematical foundation. In this chapter, we will give a brief introduction into the theory used in creating a stream cipher.

4.1 Boolean Functions

A Boolean function in n variables, is a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$,

The operations in $\mathbb{F}_2$ are addition and multiplication modulo 2.
There are several ways to represent a Boolean function. One that is frequently used in the literature is *Algebraic Normal Form* or ANF.

Definition 3. The ANF of a Boolean function f in n variables is

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = a_0 + a_1 x_1 + \cdots + a_n x_n + \\ a_{1,2} x_1 x_2 + a_{1,3} x_1 x_3 + \cdots a_{1,n} x_1 x_n + \\ a_{1,2,3} x_1 x_2 x_3 + \cdots + a_{n-2,n-1,n} x_{n-2} x_{n-1} x_n \\ \vdots \\ a_{1,2,\ldots,n} x_1 x_2 \cdots x_n \end{aligned} \tag{4.1}$$

where the a_i is either 0 or 1, depending on whether the corresponding x_i is part of the Boolean function.

Example 4.1.1. A Boolean function in 3 three variables is

$$f(x_0, x_1, x_2) = 1 + x_0 + x_1 + x_0 x_2 \tag{4.2}$$

The *algebraic degree*, or just degree, of a Boolean function is equal to the greatest degree of its ANF. The degree of the function in example (4.1.1) is 2.
If the degree of a function is 1, it is a *linear* or *affine* function.

Another common way of representing a Boolean function, is by its *truth table*.
This is a vector $(f(v_0), f(v_1), \ldots, f(v_{2^n-1}))$, where v_i is the binary representation of the number $i, 0 \leq i \leq 2^n - 1$.

Example 4.1.2. The truth table of the function in example (4.1.1) is $(1, 1, 0, 0, 0, 1, 1, 0)$.

The truth table can also be represented as a $2^n \times (n + 1)$ table, where the rows are all possible inputs, sorted lexicographically, the n first columns are the n variables and the last column is the output of the function.

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	1
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	0

TABLE 4.1: Truth table of function from (4.1.1)

A Boolean function $f(x)$ in n variables, is *balanced* if the *Hamming weigth*, denoted $d(f)$,

$$d(f) = \sum_{x \in \mathbb{F}_2^n} f(x),$$

is equal to 2^{n-1}.
The Hamming weigth of the function from example (4.1.1) is 4, so this function is balanced.
The *Hamming distance* between two Boolean functions f and g is the number of occurences where their output differ, or

$$d(f, g) = wt(f \oplus g)$$

where $\oplus$ is the sum modulo 2 of the truth tables of f and g.

4.1.1 Cryptographic Properties

When Boolean functions are used in a cryptographic setting, we want them to be *cryptographically secure*.
Some properties of the Boolean function are more important than others if we wish for them to be cryptographically secure.

Definition 4. A Boolean function $f(x_1, x_2, \ldots, x_n)$ in n variables is *correlation immune of order* k, if every subset of k or fewer variables in $x_1, x_2, \ldots, x_n$ is statistically independent of the value of $f(x_1, x_2, \ldots, x_n)$.

Boolean functions with low correlation immunity are weaker against the *correlation attack* [29], an attack on stream ciphers.
As a result, Boolean functions used in stream ciphers should have as high correlation immunity as possible.

Definition 5. If we have a Boolean function in n variables, with degree d, Siegenthaler showed that the correlation immunity k is upper bounded by $k + d \leq n$, so the higher the degree is, the lower the possible correlation immunity is.

Example 4.1.3. Returning to the Boolean function from example 4.1.1, we first fix the value of x_1 to either 0 or 1.
The resulting truth table has the same distribution as the original.
Next, we do the same with the variable x_2.
We now get a truth table with a different distribution, meaning that this Boolean function is not correlation immune.

Another measure used for Boolean functions, is the *nonlinearity*, denoted $\mathcal{N}_f$.
Nonlinearity is the minimal Hamming distance between the function f and all affine functions, or

$$\mathcal{N}_f = \min_{\phi \in \mathcal{A}_n} d(f, \phi),$$

where $\mathcal{A}_n$ is the set of all affine functions in n variables.

Example 4.1.4. If we again look at the Boolean function from 4.1.1, we list the affine functions in the variables x_1, x_2, x_3, and find the Hamming distances.
By looking at the last row of table 4.2, we see that the function $x_2 + x_3$ only differs in two positions, so the nonlinearity of f is 2.

0	x_1	x_2	x_3	x_1+x_2	x_1+x_3	x_2+x_3	$x_1+x_2+x_3$	f
0	0	0	0	0	0	0	0	1
0	0	0	1	0	1	1	1	1
0	0	1	0	1	0	1	1	0
0	0	1	1	1	1	0	0	0
0	1	0	0	1	1	0	1	0
0	1	0	1	1	0	1	0	1
0	1	1	0	0	1	1	0	1
0	1	1	1	0	0	0	1	0
4	**4**	**6**	**4**	**6**	**4**	**2**	**6**	**0**

TABLE 4.2: Hamming distances for all affine functions

There are many other properties of Boolean functions, but not all of these are important when designing a stream cipher.
For an excellent overview of the different properties related to cryptography, the book by Cusick and Stănică [10] is recommended.

4.2 Linear Feedback Shift Registers

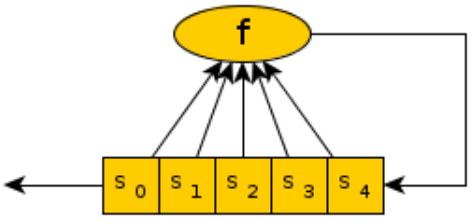

FIGURE 4.1: Feedback shift register with five registers

A *shift register* is a collection of n stages $s_0, s_1, \ldots, s_{n-1}$, each stage s_i containing either the value 0 or 1.

The contents of the register are shifted to the left, so that s_i takes on the value of $s_{i+1}, 0 \leq i < n-1$, and s_{n-1} is updated with the output of some function $f(s_0, s_1, \ldots, s_{n-1})$. The output is taken from the leftmost stage, s_0. This produces the infinite sequence

$$(s_t) = s_0, s_1, s_2, \ldots, s_p, s_0, \ldots$$

with *period* p.

If the function f is linear, it is called a *Linear Feedback Shift Register* or LFSR for short.

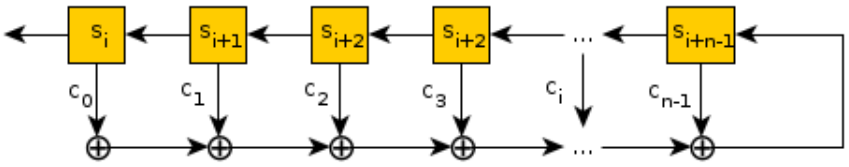

FIGURE 4.2: A Linear Feedback Shift Register

4.2.1 The Recurrence Relation

Any LFSR can be defined as a recurrence relation , with

$$s_{n+i} = c_0 s_i + c_1 s_{i+1} + \cdots + c_{n-1} s_{n+i-1}$$

where the coefficients c_i are either 0 or 1, with the restriction that $c_0 \neq 0$, otherwise it would in effect be a $n-1$-stage register, with the output being delayed.

Another way of saying this is that

Example 4.2.1. The recurrence relation of an LFSR of length 3 is given below:

$$s_{i+3} = s_{i+2} + s_i$$

Here, the last stage is updated with the sum of the first and last register.

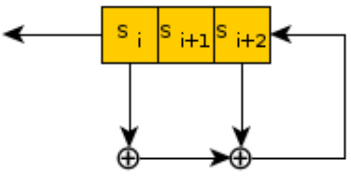

FIGURE 4.3: The LFSR from example 4.2.1

4.2.2 The Matrix Method

Every n-stage LFSR can also be represented by an $n \times n$ matrix.
This matrix takes a special form, as seen below.

$$M = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & c_0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & c_1 \\ 0 & 1 & 0 & \cdots & 0 & 0 & c_2 \\ 0 & 0 & 1 & \cdots & 0 & 0 & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & c_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & 1 & c_{n-1} \end{bmatrix} \tag{4.3}$$

By multiplying the initial vector $(s_0, s_1, \ldots, s_{n-1})$ with the matrix, the subsequent vector $(s_1, s_2, \ldots, s_n)$ is found.
Furthermore, taking the t-th power of the matrix, M^t, and multiplying it with the initial vector, we get the state at time t.

$$(s_0, s_1, \ldots, s_{n-1}) \cdot M^t = (s_t, s_{t+1}, \ldots, s_{n+t-1})$$

Example 4.2.2. If we take the matrix corresponding to the recurrence relation from the previous example, this is the corresponding matrix:

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

If the initial vector (s_0, s_1, s_2) is set to $(1, 0, 0)$, and we multiply this with the matrix M, we get a new vector $(0, 0, 1)$.
Multiplying the initial vector by M^2 we get $(0, 1, 1)$. Repeating this with increasing powers of M, and looking at the first element, s_t at time t of the vector, we get the output sequence

$$\ldots, 1, 0, 0, 1, 1, 1, 0, \ldots$$

with period 7.

4.2.3 Characteristic Polynomial

Alongside the recurrence relation and the matrix, there is one other common way of representing LFSRs, and that is as a polynomial mod 2.
This polynomial can be found by finding the characteristic polynomial of the matrix defined in 4.3, or

$$\det(M - Ix),$$

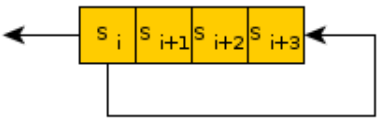

FIGURE 4.4: Pure cycling register

where I is the $n \times n$ identity matrix. This gives a polynomial of degree n that can be used to further analyse the properties of the given LFSR.

Example 4.2.3. Calculating the characteristic polynomial of the matrix from example 4.2.2, we get $x^3 + x^2 + 1$, which is a primitive polynomial of degree 3.

The characteristic polynomial is a very useful method to represent LFSRs, for instance, when we want to determine the periode of the sequence it generates.

4.2.4 Period of a Sequence

Let the period of a sequence (s_t) be denoted by p.
p is upper bounded by $2^n - 1$, and lower bounded by n, where n is the number of stages in the LFSR.

The reason for this upper bound is that a LFSR of maximum period will cycle through all possible states between 1 and $2^n - 1$. The all-zero state is not a possibility, as it will only generate the all-zero sequence.
Similarly, if we look at the *pure cycling register*, as seen in figure 4.4, where the rightmost register is updated with the contents of the leftmost register, we get a period of n.

Definition 6. If the sequence is generated by a recurrence relation with corresponding polynomial $f(x)$, the period is the number p such that $f(x)$ divides $1 - x^p$

This simple definition gives us an easy way of finding the period of a sequence, given that we know the corresponding polynomial.

Example 4.2.4. The polynomial $f(x) = x^4 + x^3 + x^2 + x + 1$ is irreducible. By trying increasing values of p, we find that $f(x)$ divides $1 - x^5$, so the period of the sequence generated by this feedback polynomial is 5.

For a sequence to have maximal period $2^n - 1$ it is a necessary, but not sufficient condition, that the feedback polynomial is *irreducible*. This means that the polynomial can

not be factored into polynomials of lower degree.

There are

$$\Psi_2(n) = \frac{1}{n}\sum_{d|n} 2^d \mu(\frac{n}{d})$$

irreducible polynomials modulo 2 of degree n, where the sum is taken over all d that divides n and $\mu(n)$ is the Möbius μ-function.

Example 4.2.5. The polynomial $f(x) = x^4 + x^3 + 1$ is irreducible modulo 2, and it is also a *primitive polynomial*. Doing the same as in (4.2.4), we find that the period $p = 15$, which is the longest possible for polynomials of degree 4.

If the polynomial is also *primitive*, the corresponding sequence is guaranteed to be of maximal length.

There are

$$\lambda_2 = \frac{\varphi(2^n - 1)}{n}$$

primitive polynomials of degree n. $\varphi(n)$ is Euler's totient function.

The period is also equal to the smallest p such that $M^p = I$, where M is the matrix defined in 4.2.2 and I is the $n \times n$ identity matrix.

Example 4.2.6. The matrix corresponding to the feedback polynomial $x^4 + x^3 + 1$ is

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

Raising this matrix to the 15th power we get

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}^{15} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which is indeed the 4×4 identity matrix, as expected.

4.3 Linear Complexity

The *linear complexity*, or linear span, of a sequence s_t, is defined as the shortest LFSR able to generate the sequence s_t, and is denoted by $L(s_t)$. The following definition is taken from [10, p.20],

1. If s_t is the all-zero sequence, then $L(s_t) = 0$
2. If no LFSR generates s_t, then $L(s_t) = \infty$
3. If there is at least one LFSR that generates s_t, then $L(s_t)$ is the length of the shortest LFSR that generates s_t. Equivalently, $L(s_t)$ is the degree of the minimal polynomial of s_t.

We want the linear complexity to be as high as possible. If the linear complexity of a sequence is n, we only need $2n$ consecutive bits to be able to uniquely recover both the feedback polynomial and initial state. The reason we are able to do this, is the Berlekamp-Massey algorithm.

4.4 The Berlekamp-Massey Algorithm

In 1968, Elwyn Berlekamp designed an algorithm [3], [4] to decode Bose-Chaudhuri-Hocquenghem (BCH) codes.
A year later, James Massey discovered [23] that the same algorithm is very powerful when analysing LFSR sequences.

The algorithm works by viewing the sequence generated by the LFSR as a set of equations.
If we know the size n of the LFSR, and can obtain $2n$ consecutive bits of the output, we know that the bits $s_n, s_{n+1}, \ldots, s_{2n-1}$ are all linear combinations of the n preceding bits.
Using this information, it is fairly easy to solve this as a set of equations.

Example 4.4.1. Given the sequence $\ldots, 1, 0, 0, 1, 1, 1, 0, \ldots$ generated by some LFSR of length three.
We then know that the fourth bit is a linear combination of the three first bits, the fifth bit is a combination of the second, third and fourth bits, and so on. Using this, we

construct the following set of equations

$$\begin{aligned} c_0 \cdot 1 + c_1 \cdot 0 + c_2 \cdot 0 &= 1 \\ c_0 \cdot 0 + c_1 \cdot 0 + c_2 \cdot 1 &= 1 \\ c_0 \cdot 0 + c_1 \cdot 1 + c_2 \cdot 1 &= 1 \end{aligned} \tag{4.4}$$

The matrix system corresponding to the equations from (4.4) is

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T = A \cdot \begin{bmatrix} c_0 & c_1 & c_2 \end{bmatrix}^T \tag{4.5}$$

or

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \tag{4.6}$$

Now, this set of equations have a unique solution if and only if the matrix A from (4.6) has an inverse. The determinant of A, $\det(A) = 1$, so A has an inverse.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Multiplying this inverse with the vector $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$, we get the new vector $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$, corresponding to the recurrence relation

$$s_{t+3} = 1 \cdot s_{t+2} + 0 \cdot s_{t+1} + 1 \cdot s_t,$$

or just

$$s_{t+3} = s_{t+2} + s_t.$$

4.5 Non-Linear Feedback Shift Registers

Non-linear feedback shift registers (NFSRs) operate in the exact same way as the LFSRs defined in section 4.2, except that the update function is non-linear (the degree of the update-function is greater, or equal to, 2).
NFSRs have some advantages, but also some disadvantages.
The main advantage is that non-linearity increases the linear complexity.
However, it is difficult to predict the period, and no general theory exists. There are 2^{2n} Boolean functions in n variables, where only 2^n of these are linear.

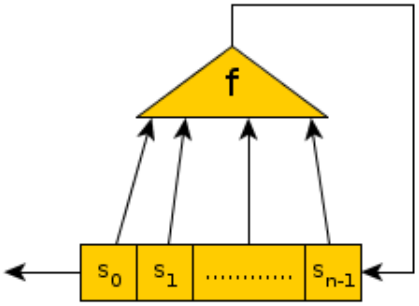

FIGURE 4.5: A non-linear feedback shift register

Example 4.5.1. Given the recurrence relation

$$s_{t+3} = 1 + s_t + s_{t+1}s_{t+2} \tag{4.7}$$

and the initial value $(s_0, s_1, s_2) = (0, 0, 0)$, we get the following sequence

$$\ldots, 0, 0, 1, 1, 0, \ldots \tag{4.8}$$

with period 5 and linear complexity 4. If the initial value is set too $(1, 0, 1)$, the sequence now becomes

$$\ldots, 1, 0, 1, 0, \ldots \tag{4.9}$$

with period 4 and linear complexity 2.

From these two small examples, it is clear that things can change fast with NFSRs. Not all NFSRs are as unpredictable. A class of these will be described in the next section.

4.5.1 de Bruijn Sequences

de Bruijn sequences are a class of non-linear sequences, but their theory are better understood.
They take their name from Nicolaas G. de Bruijn, who rediscovered them in 1946 [2], half a decade after they were proven to exist by C. Flye-Sainte Marie [22].
In his paper, de Bruijn proved that the number of such functions is equal too $2^{2^{n-1}-n}$, which is significantly higher than the number of m-sequences, for a given n.
One of the advantages of de Bruijn sequences, is that they are balanced, the number

of ones is equal to the number of zeroes.

Another advantage is that the linear complexity is significantly higher.
In their paper, Chan, Games and Key [9] prove that the linear complexity is lower bounded by $2^n - 1$ and upper bounded by $2^{n-1} + n$.

Compared with LFSRs, where the linear complexity is upper bounded by n, it is clear that de Bruijn sequences offer some advantages.

Definition 7. A (binary) *de Bruijn sequence* is a sequence of period 2^n, in which each n-bit pattern occurs exactly once in one period of the sequence.
This is referred to as the *span n property*.

Example 4.5.2. A de Bruijn sequence of length is ...,1,0,1,0,0,0,1,1,.... Reading off the subsequences of length 3, we get, in order, 101, 010, 100, 000, 001, 011, 111 and 110.

There exists a wide variety of methods to construct de Bruijn sequences. Fredricksen explains most of them in is survey [12].

Perhaps the easiest way, is to add an extra zero to the longest run of an m-sequence.

Example 4.5.3. If your m-sequence is generated by the recurrence relation

$$s_{t+4} = s_{t+1} + s_t, \tag{4.10}$$

a de Bruijn sequence can be generated by the recurrence relation

$$s_{t+4} = s_{t+1} + s_t + (s_{t+1} + 1)(s_{t+2} + 1)(s_{t+3} + 1)(s_{t+3} + 1) \tag{4.11}$$

Given the recurrence relation (4.10), and initial loading $(s_0, s_1, s_2, s_3) = (1, 1, 0, 0)$, the corresponding m-sequence is

$$\ldots, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, \ldots$$

Now, with the same initial loading, but recurrence relation (4.11), the sequence is

$$\ldots, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, \ldots$$

The linear complexity of the m-sequence is 4, whereas the linear complexity of the de Bruijn sequence is 15, the upper bound for 4-stage NFSRs.

4.6 Filter Generators

A filter generator is a way of generating long sequences using a n-stage LFSR and a Boolean "filtering function".

The inputs to the Boolean function are taken from the different stages of the LFSR.

Example 4.6.1. The sequence generated by update function $s_{i+5} = s_{i+3} + s_i$ generates a sequence with maximal period of 31.
The linear complexity is 5, because the corresponding feedback polynomial, $x^5 + x^3 + 1$, is primitive.

In his paper [18], Edwin L. Key proved that the new linear complexity is now upper bounded by

$${}_rN_m = \sum_{i=1}^{m} \binom{r}{i}, \tag{4.12}$$

where r is the number of stages in the LFSR, and m is the degree of the Boolean function. In our example, where the degree of the feedback polynomial is 3, and the degree of the

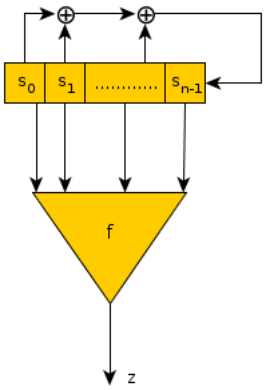

FIGURE 4.6: A filter generator

Boolean function is 2, we get

$$\binom{5}{1} + \binom{5}{2} = 5 + 10 = 15 \tag{4.13}$$

Filter generators are an easy way to increase linear complexity, but due to correlation attacks [29], nonlinear filtering is not enough to have a secure system.

4.7 Irregular Clocking

So far, both the LFSRs and NFSRs have been *regularly clocked*, one bit of keystream have been produced every time the shift register were updated.
With *irregularly clocked* LFSRs, output is not necessarily generated every time the register is clocked.

Example 4.7.1. The easiest example of an irregularly clocked sequence generator, is seen in figure 4.7.
Here we have two LFSR, LFSR1 is run as normal, and its output is $a(t)$, but LFSR2 is clocked once if $a(t) = 0$ and twice if $a(t) = 1$.

This is the step-1/step-2 generator of Gollman and Chambers [14].
If both LFSRs have n stages, it can be shown that the period of $z(t)$ is $(2^n - 1)^2$ and the linear complexity is $n(2^n - 1)$.

Another kind of irregularly clocked stream cipher is the *alternating step generator*, introduced by Günther [16]. This cipher consists of three LFSRs, LFSR1, LFSR2 and LFSR3 (in Günthers paper he used de Bruijn sequences, but it is now common to use

FIGURE 4.7: Irregularly clocked sequence generator

LFSRs instead).
At time t LFSR1 is clocked. If its output $a(t) = 1$ LFSR2 is clocked, if $a(t) = 0$, LFSR3 is clocked. Output is then taken as $z(t) = b(\mu(t)) \oplus c(t - \mu(t) - 1)$.

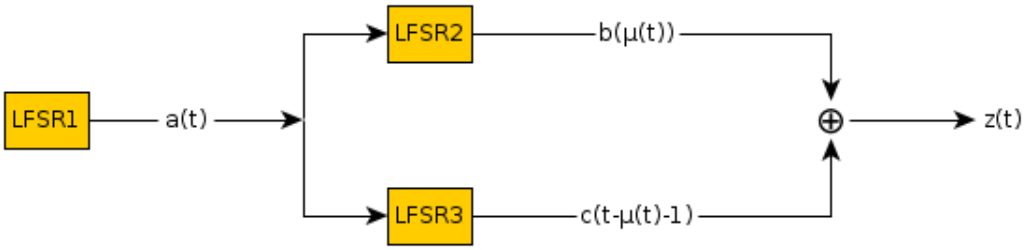

FIGURE 4.8: The alternating step generator

4.8 S-Boxes

A substitution box (S-box), or *vectorial Boolean function*, is a Boolean function

$$f(x_0, \ldots, x_{n-1}) : \mathbb{F}_2^n \to \mathbb{F}_2^m. \tag{4.14}$$

If $m = 1$, these are the same Boolean functions introduced in section 4.1.

S-boxes can be divided into three categories:

- $n > m$
- $n < m$
- $n = m$

When f is an S-box with $\mathbb{F}_2^m$ as its image, it can be visualised as a vector with m coordinates, where each coordinate is a Boolean function. (Sometimes S-boxes are called vectorial Boolean functions in the literature, such as in Carlets book [8]).

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	f	e	b	c	6	d	7	8	0	3	9	a	4	2	1	5
$S_1(x)$	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1
$S_2(x)$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
$S_3(x)$	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
$S_4(x)$	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0

TABLE 4.3: S-box in four variables

In table 4.3, we have an S-box $f : \mathbb{F}_2^4 \to \mathbb{F}_2^4$. It consists of the four Boolean functions seen in (4.15).

$$\begin{aligned} S_1 &= 1 + x_1 + x_3 + x_2x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4 \\ S_2 &= 1 + x_1x_2 + x_1x_3 + x_1x_2x_3 + x_4 + x_1x_4 + x_1x_2x_4 + x_1x_3x_4 \\ S_3 &= 1 + x_2 + x_1x_2 + x_2x_3 + x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4 \\ S_4 &= 1 + x_3 + x_1x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4 \end{aligned} \tag{4.15}$$

5

The Grain Cryptosystem

5.1 The eSTREAM Project

The eSTREAM project was a project to "identify new stream ciphers suitable for widespread adoption". It started with a call for papers, where everyone could submit their own stream cipher proposal, in November 2004, and ended in April 2008.
Submissions were divided into two categories, or profiles,

1. Profile 1: "Stream ciphers for software applications with high throughput requirements"
2. Profile 2: "Stream ciphers for hardware applications with restricted resources such as limited storage, gate count, or power consumption."

The ciphers in profile 1 are HC-128, Rabbit, Salsa20/12 and SOSEMANUK and the ciphers in profile 2 are Trivium, MICKEY and Grain.
In this chapter, we will give a description of the cipher Grain, a stream cipher designed with implementation in hardware in mind.

5.2 Description

Grain is designed primarily for restricted hardware environments, such as RFID tags and other environments where hardware is a limiting factor.
The internal state of Grain is 160 bits, where 80 of these bits are the key, and the remaining 80 bits are the initial value (IV). However, only 64 bits of the IV can be set by the user, the 16 rightmost bits are set to 1 to avoid the all-zero LFSR.

In its normal mode, both the LFSR and NFSR are clocked once for each bit of output that is generated. However, one of the key features of Grain is that the number of output-bits generated can be increased, at a higher cost of implementation in hardware.

5.3 Technical Details

As already mentioned, Grain consists of one 80-bit LFSR, and one 80-bit NFSR.
The LFSR s_t is updated with the recurrence relation

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i \tag{5.1}$$

Its corresponding feedback polynomial is

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}, \tag{5.2}$$

The NFSR is updated with the sum of a non-linear Boolean function, and one bit from the LFSR.

$$\begin{aligned} b_{i+80} =& s_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + \\ & b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + \\ & b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + \\ & b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21} + b_{i+15}b_{i+9} + \\ & b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}. \end{aligned} \tag{5.3}$$

Here, s_i is the left-most stage of the LFSR.
The feedback polynomial, $g(x)$, is defined as

$$\begin{aligned} g(x) = \quad & 1 + x^{18} + x^{20} + x^{28} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{66} + x^{71} + x^{80} + \\ & x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + \\ & x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + \\ & x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}. \end{aligned} \tag{5.4}$$

Four bits from the LFSR and one bit from the NFSR are taken as input to a Boolean function $h(x)$, that is balanced, correlation immune of the first order and with algebraic degree 3. The nonlinearity of $h(x)$ is the highest possible for a function of degree 3, 12.

$$h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4 \quad (5.5)$$

where the variables x_0, x_1, x_2, x_3 and x_4 correspond to the tap positions $s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}$ and b_{i+63} respectively.

The output of Grain is then defined as

$$z_i = \sum_{k \in \mathcal{A}} b_{i+k} + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}) \quad (5.6)$$

where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$.

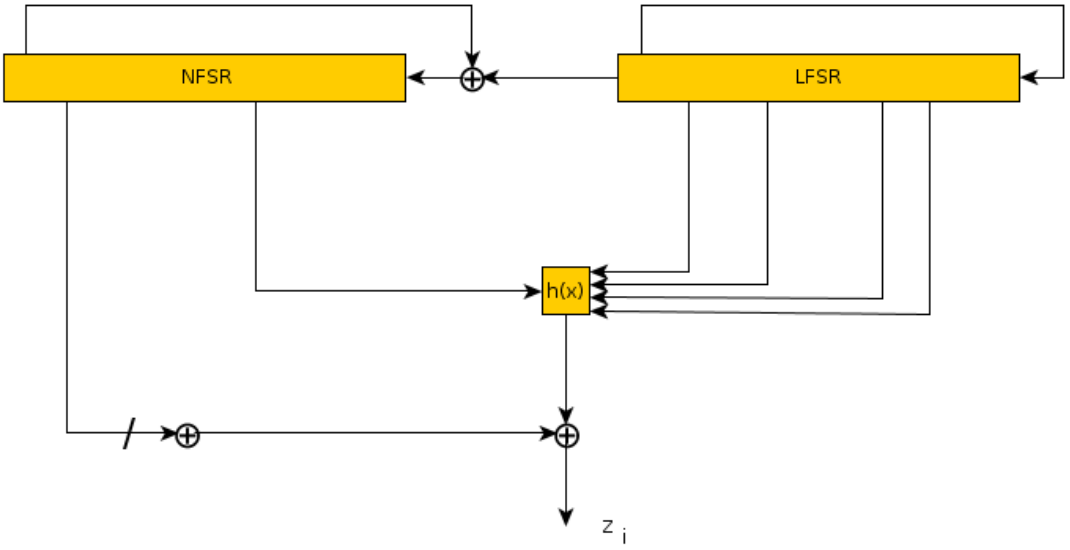

FIGURE 5.1: The Grain Stream Cipher

5.4 Key Initialization

Before any keystream is generated, the key k is loaded into the NFSR,

$$b_i = k_i, 0 \leq i \leq 79 \quad (5.7)$$

and the IV is loaded into the 64 leftmost registers of the LFSR

$$s_i = IV_i, 0 \leq i \leq 63, \quad (5.8)$$

while the remaining 16 bits of the LFSR are set to 1,

$$s_i = 1, 64 \leq i \leq 79.$$

This is done to make sure that the LFSR is not loaded with an all-zero key, which would generate the all-zero sequence.
The cipher is then clocked 160 times, with no output being generated, instead it is fed back into both the LFSR and NFSR.

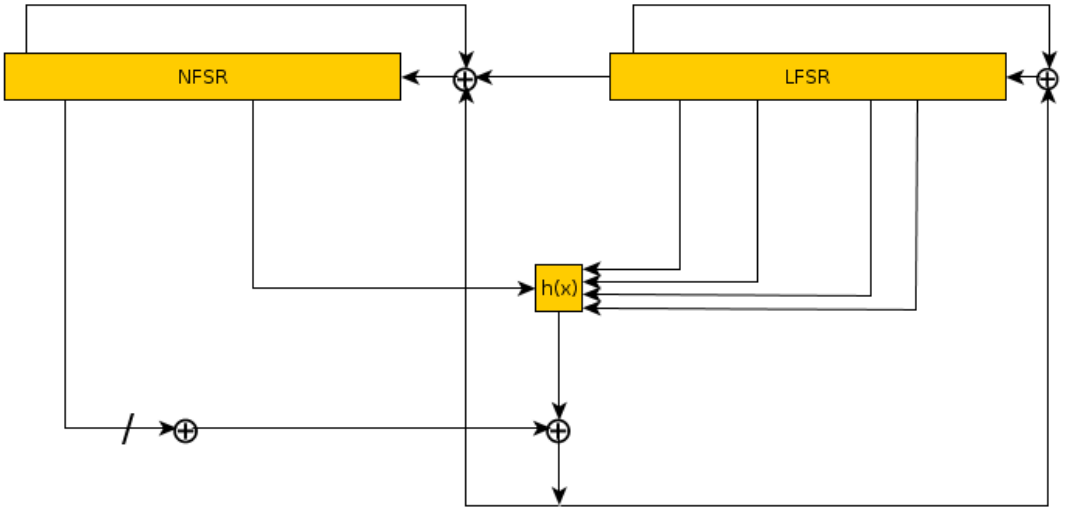

FIGURE 5.2: Key initialization of Grain

5.5 Throughput Rate

As mentioned earlier in this chapter, Grain produces 1 bit of output every time the two registers are clocked.
This can, however, be increased by implementing the functions $f(x)$, $g(x)$ and $h(x)$ several times, at a cost of more hardware.
To make this implementation easier, the last 15 bits of both shift registers, $s_i, 65 \leq i \leq 79$ and $b_i, 65 \leq i \leq 79$ are not used in either the update functions or output function.
In figure 5.3 we show how the throughput can be doubled by implementing the update and output functions twice.
Of course, if Grain is implemented to output more than one bit per clock, it is important to implement the shift registers to shift the corresponding number of bits at each clock.

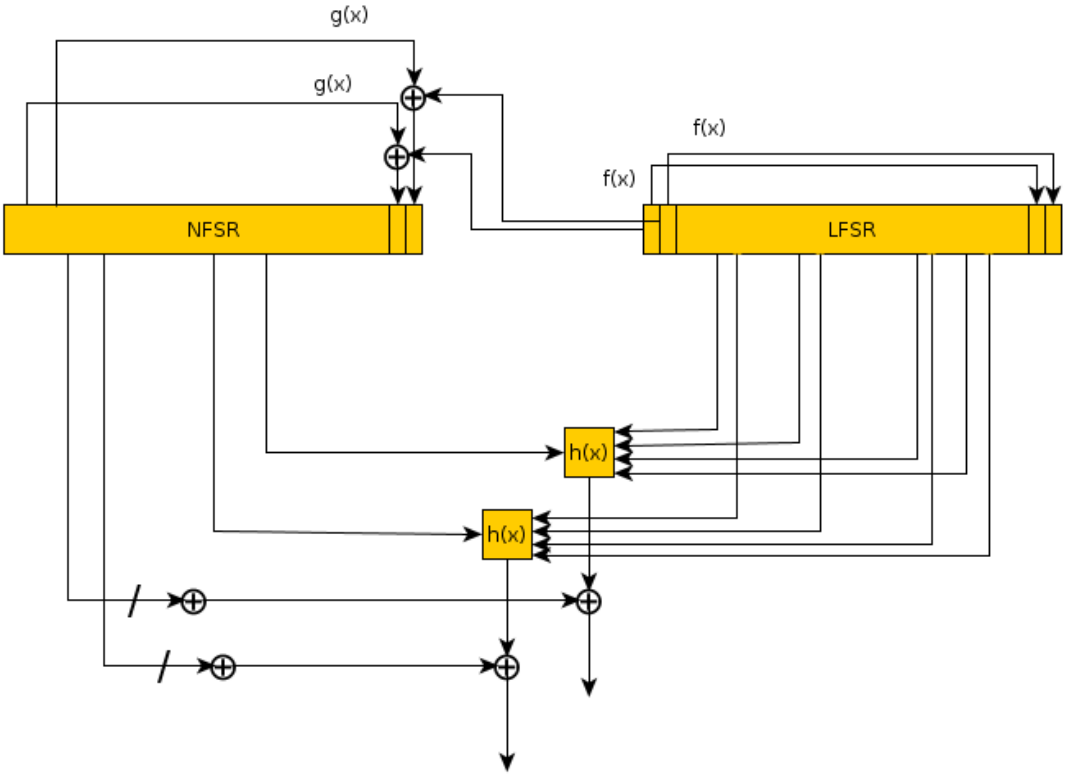

FIGURE 5.3: Grain producing two bits of output per clock

5.6 Cryptanalysis

At the time of writing, there are no known weaknesses in Grein, that are able to find the key in less time than a brute force attack.
In the first version of Grain there were some potential weaknesses, but they were adressed in the second, and final version of the cipher.
For more on cryptanalysis of Grain, see [6] or [19].

6

The Grein Cryptosystem

Grein is a stream cipher developed by the Norwegian National Security Authority.

It takes its design from the Grain cryptosystem explained in the previous chapter, but some alterations have been made.
Most significantly, the NFSR from Grain is replaced by a new binary tree. Some changes to the output function have been made as well.

The specifications can be found in [26].
Grein is norwegian for *branch*, as in *tree branch*.

6.1 Description

The components of Grein are one 80-bit LFSR, update with the same update function as in Grain (see (5.1)).
Instead of the 80-bit NFSR, we have ten 8-bit LFSRs arranged as a binary tree (see figure 6.1).
Each of these LFSRs are updated with their own update function, all corresponding to primitive polynomials of degree 8, to ensure maximal period.

6.2 Paths Through a Tree

Since this is a binary tree, the probability that one goes left should be the same as that one goes right. You can imagine one of those old games, where you insert a coin, and it

will "trickle down" the levels, following one of many paths on its way.
If the tree is balanced, all paths should be equally likely.
Assume a balanced tree, and let $l(i)$ denote the number of nodes at level i. For a node u in the tree, we use the notation $in(u)$ and $out(u)$ to the denote the number of incoming and outgoing edges of u, respectively. Further, denote by

$$\begin{aligned} t(u) &= \sum_{v \in in(u)} \frac{1}{2} t(v) \\ &= \frac{1}{2} \sum_{v \in in(u)} t(v) \end{aligned} \tag{6.1}$$

the probability that a path hits node u, computed recursively. The tree is balanced if $t(u) = \frac{1}{l(i)}$ for all nodes at level i.
To simulate random paths through the tree, it is natural to implement the binary decisions at the nodes with pseudorandom binary sequence generators.
In Grein, this is done with small, irregularly clocked filter generators at the nodes.

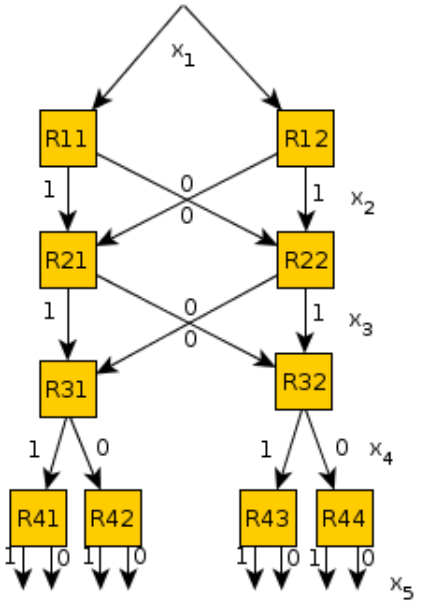

FIGURE 6.1: The Grein tree structure

6.3 The Tree

Since the 80-bit NFSR of Grain is replaced, the total state-size of the equivalent tree must also be 80. The tree that was chosen has structure $(t(1), t(2), t(3), t(4)) = (2, 2, 2, 4)$, so the nodes are implemented with 8-bit registers, to get total state-size of 80 bit.

The tree chosen can be seen in figure 6.1 where the nodes R_{ij} implement 8-bit filter generators with states S_{ij}^n with primitive feedback polynomials g_{ij} filtered through 5-variable Boolean functions f_{ij}.

6.4 The Filter Generators

To simulate random decisions at the nodes, Grein uses small, nonlinearly, filtered LFSRs that are clocked irregularly. Thus, when a path arrives at a node, the next binary path value is given by the output from the filter generator at that particular node. To arrive at 80-bit total state size, the 10 LFSRs all have length 8. The 10 LFSRs in the tree are defined by primitive polynomials of degree 8 over $\mathbb{F}_2$. The minimal polynomials of these LFSRs are all primitive

$$\begin{aligned}
g_{11} &= x^8 + x^4 + x^3 + x^2 + 1 \\
g_{12} &= x^8 + x^5 + x^3 + x + 1 \\
g_{21} &= x^8 + x^5 + x^3 + x^2 + 1 \\
g_{22} &= x^8 + x^6 + x^3 + x^2 + 1 \\
g_{31} &= x^8 + x^6 + x^5 + x + 1 \\
g_{32} &= x^8 + x^6 + x^5 + x^2 + 1 \\
g_{41} &= x^8 + x^6 + x^5 + x^3 + 1 \\
g_{42} &= x^8 + x^6 + x^5 + x^4 + 1 \\
g_{43} &= x^8 + x^7 + x^2 + x + 1 \\
g_{44} &= x^8 + x^7 + x^3 + x^2 + 1
\end{aligned} \tag{6.2}$$

If T_{ij} denotes the characteristic matrix of $g_{ij}(x)$, then the states S_{ij} satisfy

$$S_{ij}^t = S_{ij} \cdot T^t, t = 0, 1, 2, \ldots \tag{6.3}$$

where multiplying the initial state of S_{ij} with powers of T corresponds to shifting the LFSR t times.
For simplicity the same filter function for each of the LFSRs (all f_{ij}s) are identical,

$$f(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 + x_5 + x_2x_3 + x_1x_5 \tag{6.4}$$

This function has optimal nonlinearity for a five variable function, which is 12.
Let M_{ij} be a nonsingular 8×5 binary matrix. Then for each time t the output value

a_{ij}^t of the generator R_{ij} is computed by

$$a_{ij}^t = f((S_{ij} \cdot T_{ij^t}) \cdot M_{ij}), t = 0, 1, 2, \ldots \tag{6.5}$$

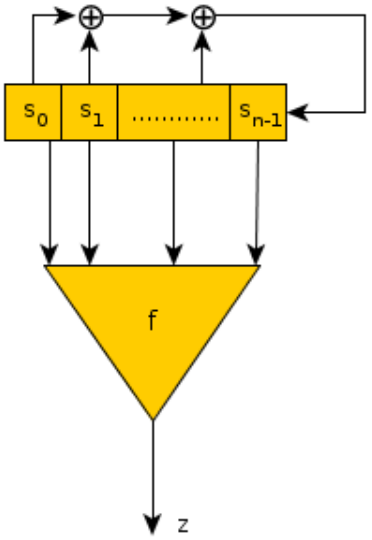

FIGURE 6.2: The filter function used in Grein

6.5 Computing the Path

There are $2^5 = 32$ equally possible paths through the tree, uniquely identified by the binary values $X_1^t, X_2^t, X_3^t, X_4^t$ and X_5^t. The first value X_1^t decides whether to go to R_{11} (if $X_1^t = 1$) or R_{12} (if $X_1^t = 0$). Assume that $X_1^t = 1$ and that the path arrives at the node R_{11}. The next path value X_2^t is then equal to the output value

$$a_{11}^t = f(S_{11}^t \cdot M_{11}) \tag{6.6}$$

of the sequence generator at R_{11}. Or in general

$$X_2^t = X_1^t a_{11}^t + (X_1^t + 1) a_{12}^t \tag{6.7}$$

Thus, if $X_1^t = 1$ and $X_2^t = 0$, it means that the path goes from node R_{11} to R_{22}. Similarly to (6.7), the general path values satisfy

$$\begin{aligned} X_3^t =& (X_1^t X_2^t + (X_1^t + 1)(X_2^t + 1)a_{21}^t + (X_1^t(X_2^t + 1) + (X_1^t + 1)X_2^t)a_{22}^t \\ =& (X_1^t + X_2^t + 1)a_{21}^t + (X_1^t + X_2^t)a_{22}^t \\ X_4^t =& (X_1^t + X_2^t + X_3^t)a_{31}^t + (X_1^t + X_2^t + X_3^t + 1)a_{32}^t \end{aligned} \tag{6.8}$$

while the last path variable X_5^t satisfy

$$\begin{aligned} X_5^t =& (X_1^t + X_2^t + X_3^t)((X_4^t + 1)a_{41}^t + X_4^t(a_{42}^t + 1) + \\ & (X_1^t + X_2^t + X_3^t + 1)(X_4^t(a_{43}^t + 1) + (X_4^t + 1)a_{44}^t) \end{aligned} \tag{6.9}$$

where

$$a_{ij}^t = f_{ij}(S_{ij}^t \cdot M_{ij}). \tag{6.10}$$

So, if for instance $(X_1^t, X_2^t, X_3^t, X_4^t, X_5^t) = (1, 0, 0, 1, 0)$, the path is

$$\overset{1}{\rightarrow} R_{11} \overset{0}{\rightarrow} R_{22} \overset{0}{\rightarrow} R_{31} \overset{1}{\rightarrow} R_{41} \overset{0}{\rightarrow}. \tag{6.11}$$

If we know the path-values, then from the equations (6.7),(6.8),(6.9) we also know the output-bits a_{ij}^t from the four sequence generators in the path.

The whole path is computed sequentially and has complexity roughly equal to that of evaluating the Boolean function in (6.4) four times. The five path values will then be used for the remaining operations:

- Irregular clocking (Section 6.6)
- Feedback (Section 6.7)
- Contribute to keystream (Section 6.8)

6.6 Irregular Clocking

Sequences generated by LFSRs are extremely structured and so it makes sense to add conditional clocking to the LFSRs. Depending on the path through the tree, the LFSRs at node R_{ij} are clocked once or twice depending on the value of their associated Boolean clock-functions h_{ij}. These functions take as input the path-variables X_2^t, X_3^t, X_4^t and X_5^t. We argue the following requirements for the functions h_{ij}. Any LFSR is either clocked once or twice with probability $\frac{1}{2}$. The clock-functions should make sure that the path-variables affect the clocking of all LFSRs in the tree, to maximize dependency between the nodes. In particular, it should not be possible to predict the clocking of

one LFSR by guessing only a subset of the four involved path values.

Let

$$\begin{aligned} g_1(X_2^t, X_3^t, X_4^t, X_5^t) &= X_3^t + X_4^t + X_5^t \\ g_2(X_2^t, X_3^t, X_4^t, X_5^t) &= X_2^t + X_4^t + X_5^t \\ g_3(X_2^t, X_3^t, X_4^t, X_5^t) &= X_2^t + X_3^t + X_5^t \\ g_4(X_2^t, X_3^t, X_4^t, X_5^t) &= X_2^t + X_3^t + X_4^t \end{aligned} \tag{6.12}$$

Then the clock-functions are

$$h_{i,j}(X_2^t, X_3^t, X_4^t, X_5^t) = g_i(X_2^t, X_3^t, X_4^t, X_5^t) + (j \mod 2) \tag{6.13}$$

where $h_{4,1} = h_{4,3}$ and $h_{4,2} = h_{4,4}$. Thus there is a symmetry in the way the LFSRs are clocked, in that we only need to guess four values in order to predict all the 10 registers.

6.7 Feedback

There is a feedback in the tree computed as a four-valued Boolean function in the variables $X_2^t, X_3^t, X_4^t, X_5^t$. The motivation of a feedback is to break up the many symmetries and nice structures introduced by the tree structure; path variables are computations from the past and will have complex effect on computations in the future. For feedback, we use the function

$$f_2(x_1, x_2, x_3, x_4) = x_1 + x_2 + x_3 + x_4 + x_2x_4 \tag{6.14}$$

such that the feedback value becomes

$$w_t = f_2(X_2^t, X_3^t, X_4^t, X_5^t) \tag{6.15}$$

for $t = 0, 1, 2, \ldots$. The value w_t is then XORed with the value starting at the top of the tree, forming the next path value

$$X_1^{t+1} = w_t + s_t \tag{6.16}$$

for $t = 0, 1, 2, \ldots$ where s_t is the value passed from the 80-bit LFSR.

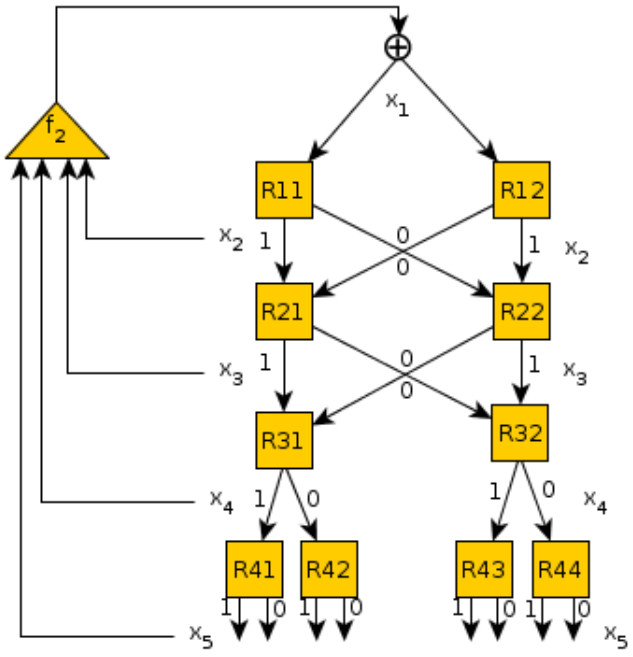

FIGURE 6.3: The feedback to the tree

6.8 Plugging the Tree into Grein

To benefit from analysis already published on Grain[6] [19], the designers of Grein aimed to minimize the distance between Grein and Grain as much as possible.

Let S_2 be a 4-bit S-box and

$$(Y_1^t, Y_2^t, Y_3^t, Y_4^t) = S_2(X_2^t, X_3^t, X_4^t, X_5^t). \tag{6.17}$$

In Grain, one bit b_1^t is fed from the NFSR and in to the 5-bit Boolean filter function h, together with four bits from the LFSR. Then a sum of bits b_2^t from the NFSR is XORed to the output of h again. In Grein, we have

$$b_1^t = Y_1^t + Y_3^t \tag{6.18}$$

form the input to the Grain filter-function h and

$$b_2^t = Y_2^t + Y_4^t \tag{6.19}$$

the value XORed with the output of h.

The function $S_2(x)$ chosen for Grein is the PRINCE [5] S-box.

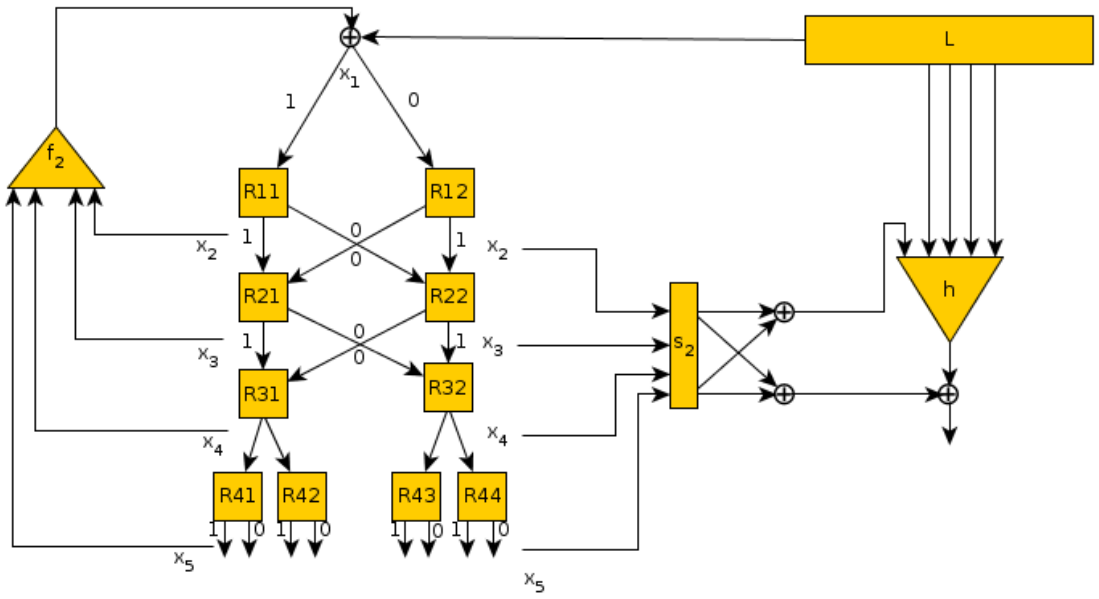

FIGURE 6.4: The Grein Stream Cipher

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_2(x)$	B	F	3	2	A	C	9	1	6	7	8	0	E	5	D	4

TABLE 6.1: S_2: The PRINCE S-box

6.9 An Example

In this section, we will give a full walkthrough of one iteration of Grein.
All eleven registers have been loaded with all ones to simplify the calculations done.

The first thing that needs to be done, is to choose a value for X_1^0. Without loss of generality, we set this to 1.
Next, we calculate the value of X_2^0. Because all the registers are loaded with all ones, the output of the filters will also be 1. Thus we get

$$X_2^0 = X_1^0 a_{11}^0 + (X_1^0 + 1)a_{21}^0 = 1 \cdot 1 + 0 \cdot 1 = 1.$$

With both X_1^0 and X_2^0, we can find X_3^0 in a similar fashion as we found X_2^0.

$$X_3^0 = (X_1^0 + X_2^0 + 1)a_{21}^0 + X_1^0 + X_2^0 a_{22}^0 = (1 + 1 + 1) \cdot 1 + (1 + 1) \cdot 1 = 1$$

Next, we calculate X_4^0,

$$X_4^0 = (X_1^0+X_2^0+X_3^0)a_{31}^0+(X_1^0+X_2^0+X_3^0+1)a_{32}^0 = (1+1+1)\cdot 1+(1+1+1+1)\cdot 1 = 1.$$

Finally, we can calucalte X_5^0,

$$\begin{aligned} X_5^0 =&(X_1^0+X_2^0+X_3^0)((X_4^0+1)a_{41}^0+X_4^t(a_{42}^0+1))+\\ &(X_1^0+X_2^0+X_3^0+1)(X_4^0(a_{43}^0+1)+(X_4^0+1)a_{44}^0)\\ =&(1+1+1)\cdot((1+1)\cdot 1+1\cdot(1+1))+(1+1+1+1)\cdot(1\cdot(1+1)+(1+1)\cdot 1)\\ =&0+0+0+0=0 \end{aligned}$$

We now have $(X_1^0, X_2^0, X_3^0, X_4^0, X_5^0) = (1,1,1,1,0)$. With the five path variables, we can clock the registers of the tree.
To do this, we first need the output of g_1, g_2, g_3 and g_4. Here, we omit the parameters of the functions, as these are all the same.
We get

$$g_1 = X_3^0 + X_4^0 + X_5^0 = 1+1+0 = 0,$$

$$g_2 = X_2^0 + X_4^0 + X_5^0 = 1+1+0 = 0,$$

$$g_3 = X_2^0 + X_3^0 + X_5^0 = 1+1+0 = 0$$

and

$$g_4 = X_2^0 + X_3^0 + X_4^0 = 1+1+1 = 1.$$

We can now find the output of the clock-functions $h_{i,j} = g_i + (j \mod 2)$
,

$$h_{1,1} = g_1 + 1 = 0+1 = 1,$$

$$h_{1,2} = g_1 + 0 = 0+0 = 0,$$

$$h_{2,1} = g_2 + 1 = 0+1 = 1,$$

$$h_{2,2} = g_2 + 0 = 0+0 = 0,$$

$$h_{3,1} = g_3 + 1 = 1+0 = 1,$$

$$h_{3,2} = g_3 + 0 = 0+0 = 0,$$

$$h_{4,1} = g_4 + 1 = 1+1 = 0,$$

$$h_{4,2} = g_4 + 0 = 1+0 = 1,$$

$$h_{4,3} = g_4 + 1 = 1+1 = 0$$

and

$$h_{4,4} = g_4 + 0 = 1 + 0 = 1.$$

With this information, we now clock registers $R_{11}, R_{21}, R_{31}, R_{42}$ and R_{44} twice, and the remaining registers are clocked once. The 80-bit LFSR also gets updated according to its update function.

The next step now, is the S-box S_2 from (6.1),

$$S_2(X_2^0, X_3^0, X_4^0, X_5^0) = S_2(E) = D = (1, 1, 0, 1).$$

We now get

$$b_1^0 = 1 + 0 = 1$$

and

$$b_2^0 = 1 + 1 = 0.$$

We can now find the bit for the keystream,

$$z_0 = h(s_3, s_{25}, s_{46}, s_{64}, b_1^0) + b_2^0 = 0 + 0 = 0.$$

Finally, we find

$$w_0 = f_2(X_2^0, X_3^0, X_4^0, X_5^0) = 1,$$

which is added to the first bit s_0 of the 80-bit LFSR to generate X_1^1,

$$X_1^1 = w_0 + s_0 = 1 + 1 = 0.$$

Continuing this way, we find that the first 10 bits of the keystream are

$$0, 0, 1, 0, 0, 0, 1, 1, 1, 1, \ldots.$$

7

A Modified Version of the Grein Cryptosystem

The Grein cipher, explained in the previous chapter, is updated with a new family of sequences. Hopefully this will be beneficial from a cryptographic perspective.

7.1 de Bruijn Sequences Revisited

As explained in section 4.5.1, de Bruijn sequences are sequences generated by a nonlinear recurrence relation.
In their paper, Mykkeltveit, Siu and Tong [24, Thm 4.1], introduce a new method of constructing de Bruijn sequences with period 2^{n+1} from sequences of period 2^n.

In this section, we will use a sligthly different notation, for example, the nonlinear recurrence relation

$$s_{t+3} = 1 + s_t + s_{t+1} + s_{t+1}s_{t+2},$$

or

$$1 + s_t + s_{t+1} + s_{t+1}s_{t+2} + s_{t+3} = 0$$

is now denoted as a function

$$g(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1x_2 + x_3 = 0$$

Definition 8. Composite Recurrence Relations
Let

$$g(x_0, x_1, \ldots, x_n) = x_0 + G(x_1, \ldots, x_{n-1}) + x_n = 0$$

and

$$f(x_0, x_1, \ldots, x_m) = x_0 + F(x_1, \ldots, x_{m-1}) + x_m = 0$$

be two recurrence relations of n and m stages respectively that generate periodic sequences, where G and F are Boolean functions in $n-1$ and $m-1$ variables, respectively. Then, a *composite recurrence relation*, denoted as $g \circ f$, is defined by [24]

$$g \circ f = g(f(x_0, \ldots, x_m), f(x_1, \ldots, x_{m+1}), \ldots, f(x_n, \ldots, x_{m+n-1})) = 0$$

which is a recurrence relation of $(n+m)$ stages.

Example 7.1.1. Given the recurrence relation

$$g(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1x_2 + x_3 = 0,$$

a recurrence relation in 3 variables, and

$$f(x_0, x_1) = x_0 + x_1,$$

we get

$$g \circ f = g(f(x_0, x_1), f(x_1, x_2), f(x_2, x_3), f(x_3, x_4),$$

which, once calculated, we get

$$g \circ f = 1 + x_0 + x_1x_2 + x_1x_3 + x_2x_3 + x_4 = 0,$$

which is a recurrence relation in 4 variables.

The following definiton, showing how composite recurrence relations are used to construct new de Bruijn sequences, is also from [24]. However, the notation used here is borrowed from [21].

Definition 9. Let $g = x_0 + G(x_1, \ldots, x_{n-1}) + x_n$, which generates a de Bruijn sequence with period 2^n and let $\psi(x_0, x_1) = x_0 + x_1$.
Then both

$$h_1 = g \circ \psi + \prod_{i \in \mathbb{Z}_o^n} x_i \prod_{i \in \mathbb{Z}_e^n} (x_i + 1)$$

and

$$h_2 = g \circ \psi + \prod_{i \in \mathbb{Z}_o^n} (x_i + 1) \prod_{i \in \mathbb{Z}_e^n} x_i$$

will generate a de Bruijn sequence of period 2^{n+1}. Here, $\mathbb{Z}_o^n$ and $\mathbb{Z}_e^n$ is the odd and even numbers between 1 and n, respectively.

Example 7.1.2. Let $g(x_0, x_1, x_2) = 1 + x_0 + x_2 + x_1x_2$, be a Boolean function that generates a de Bruijn sequence of period 8.
We then have

$$h_1 = 1 + x_0 + x_1 + x_1x_2 + x_2x_3 + x_1x_2x_3$$

and

$$h_2 = 1 + x_0 + x_1 + x_2 + x_1x_3 + x_1x_2x_3,$$

which both generate de Bruijn sequences of period 16.

Sequences generated by this method will have high linear complexity, even if the longest gap is reduced by one.

Example 7.1.3. The function h_1 from the previous example generates the sequence

$$\ldots, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, \ldots$$

with linear complexity 15, which is the highest for a recurrence relation in 4 variables (See [9] for proof on this). Removing the first 0 reduces linear complexity to 12.

7.2 Advantages of de Bruijn Sequences

The motivation for substituting linear m-sequences with nonlinear de Bruijn sequences, is the fact that they are balanced.
Hopefully, this will make the distribution of the path variables more uniform, i.e., all paths are equally probable.
If the path variables are more uniformly distributed, this could also affect the output of the S-box, making this more uniform as well.
As an added bonus, de Bruijn sequences have much higher linear complexity, which should make cryptanalysis harder.

7.3 Implementation

To easier differentiate the two versions of Grein, the original, with LFSR-sequences, and the new version, with de Bruijn sequences, we will call the original version Grein-L and the new version Grein-d.

Implementing Grein-d was done exactly the same as when implementing Grein-L, the only difference being the functions used to generate the de Bruijn sequences.

To find 10 different de Bruijn sequences of length 256, we used five different m-sequences, and applied the method explained in Section 7.1. The different functions used can be found in (7.1). The function $G(s)$ is the Boolean function defined Section 7.1.

$$\begin{aligned}
g_{11} &= s_0 + s_1 + s_2 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1) \\
g_{12} &= s_0 + s_1 + s_2 + G(s) + s_2s_4s_6(s_1+1)(s_3+1)(s_5+1)(s_7+1) \\
g_{21} &= s_0 + s_3 + s_4 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1) \\
g_{22} &= s_0 + s_3 + s_4 + G(s) + s_2s_4s_6(s_1+1)(s_3+1)(s_5+1)(s_7+1) \\
g_{31} &= s_0 + s_1 + s_4 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1) \\
g_{32} &= s_0 + s_1 + s_4 + G(s) + s_2s_4s_6(s_1+1)(s_3+1)(s_5+1)(s_7+1) \\
g_{41} &= s_0 + s_4 + s_5 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1) \\
g_{42} &= s_0 + s_4 + s_5 + G(s) + s_2s_4s_6(s_1+1)(s_3+1)(s_5+1)(s_7+1) \\
g_{43} &= s_0 + s_1 + s_2 + s_5 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1) \\
g_{44} &= s_0 + s_1 + s_2 + s_5 + G(s) + s_1s_3s_5s_7(s_2+1)(s_4+1)(s_6+1)
\end{aligned} \tag{7.1}$$

Besides these different functions, no other changes were made.

7.4 Statistical Tests

The National Institute of Standards and Technology have made a suite of statistical tests [1] to test random and pseudorandom number generators. These tests are well suited to test if a stream cipher is also statistically secure.

The test suite is comprised of fifteen tests that should all be passed, for a PRNG to be considered secure.

A list of the test that are done can be found in [1, Chap. 2].
Among the tests are the Frequency Test, where the number of ones and zeroes is counted, and both these numbers should be close to $\frac{N}{2}$.
Other tests are the Runs Test, Frequency within a Block, Fast Fourier Transform and the Linear Complexity Test.

The tests are run several times, and a *P-value* is computed. This value is used as a measure for wether the sequence is random or not. If the P-value is below our *significance level* $\alpha = 0.01$, the sequence is considered non-random. Otherwise, the sequence is considered random.

In our experiments, two sequences of length $1,000,000,000$ were generated, one sequence generated using Grein-L and the other sequence using Grein-d.
All registers were loaded with the all-one state. The test suite from NIST was then run with parameter 1,000,000, and 1,000 bitstreams were generated. In Appendix B, the results for both Grein-L and Grein-d can be found.

For analyising the test results, NIST gives recommendations on how to interpret the results.

First of all, the proportion of sequences passing the tests is found. This is done by computing a *confidence interval*, defined as

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}},$$

where $\hat{p} = 1 - \alpha$ and m is the sample size, 1,000.
In our case, the confidence interval is 0.99 ± 0.00943928. Looking at the P-value plots in figures 7.1 and 7.2, we find that both Grein-L and Grein-d pass all the statistical tests within the confidence interval.

The distribution of P-values should also be uniform. To check for this, the P-values are divided into 10 intervals between 0 and 1, and the P-values that occur within each interval is counted and displayed in a histogram. In figures 7.3 and 7.4 the histograms for Grein-L and Grein-d, it can be seen that both Grein-L and Grein-d have a uniform distribution of the P-values. It could, however, be argued that Grein-L has a more uniform distribution.

Looking at the tables in Appendix C, we find that both Grein-L and Grein-d pass all of the statistical tests.

7.5 Path Distributions

In order to test our theories, a new implementation of Grein is necessary.
The only change, being that the nodes now contain non-linear feedback shift registers producing de Bruijn sequences, instead of linear feedback shift registers.
For the rest of this thesis, we will denote the original version of Grein by Grein-L and the new version by Grein-d, to emphasize that the only difference is the LFSRs used in Grein-L and de Bruijn sequences used in Grein-d.

In order to test the distribution of the path variables, both the original and modified version of Grein was clocked 100000 times.

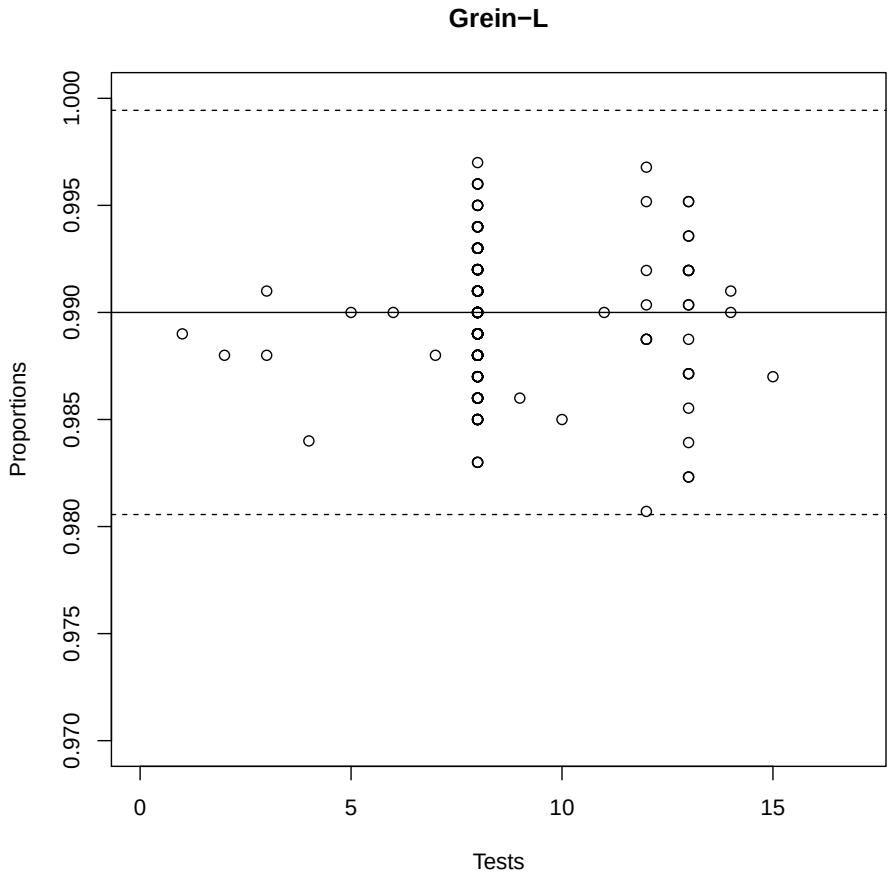

FIGURE 7.1: P-values of Grein-L

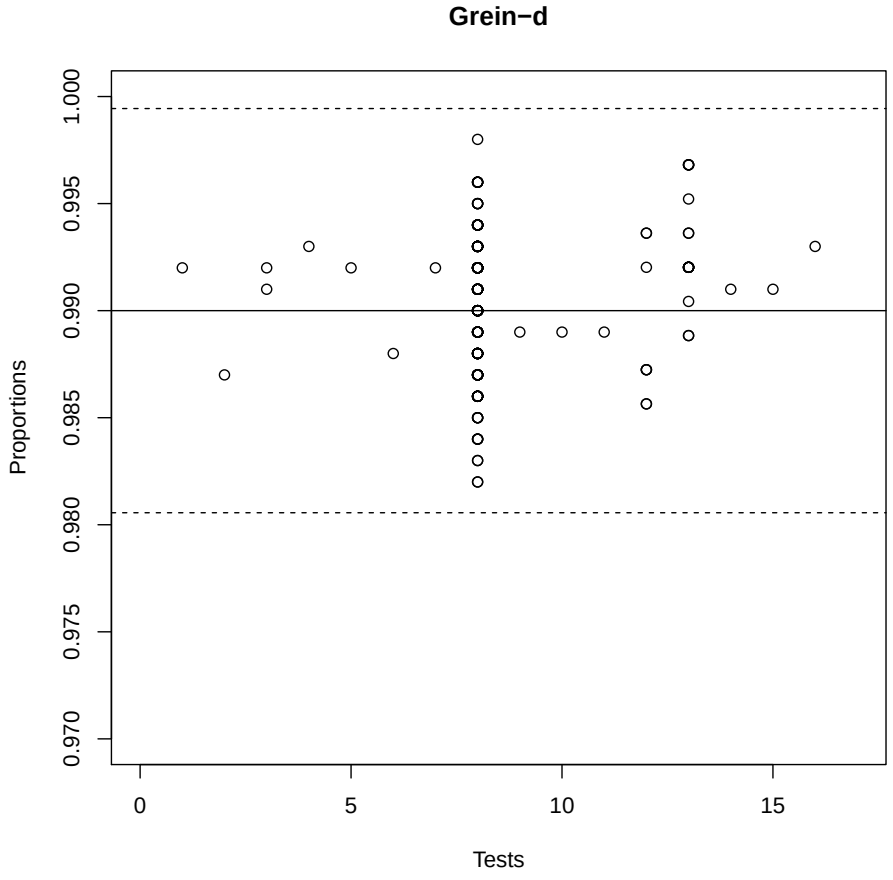

FIGURE 7.2: P-values of Grein-d

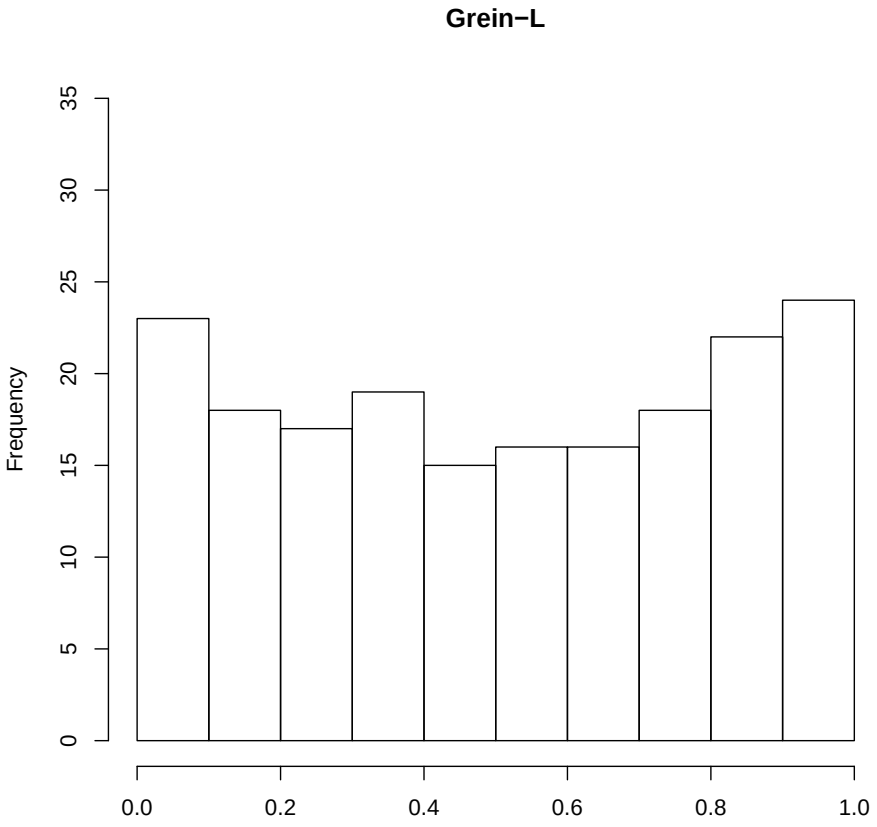

FIGURE 7.3: Histogram of P-values for Grein-L

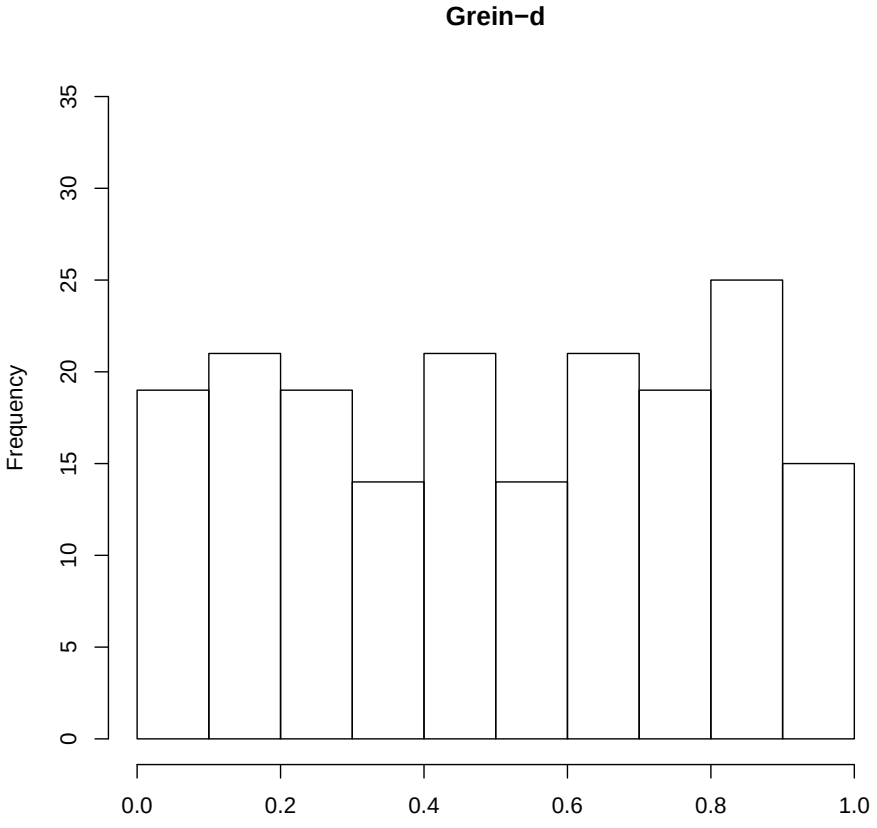

FIGURE 7.4: Histogram of P-values for Grein-d

The path variables x_1, x_2, x_3, x_4 and x_5 were then represented as a 5-bit number,

$$n_x = \sum_{i=0}^{4} x_i 2^{4-i},$$

$0 \leq n_x \leq 31$.
We then count the number of occurences for each number n_x and divide this by 100000. Ideally, this number should be as close as possible to $\frac{1}{32}$, or 0.03125.

Because of the fact that an m-sequence contains 2^{n-1} ones and $2^{n-1} - 1$ zeroes, we expect that paths with higher Hamming weigth occur more often than paths with low Hamming weight. If this is true, it might be exploited by a cryptanalysist.
The modified version of Grein however, should not have this uneven distribution.
The results are listed in full in table B.1 on page 53.
As can be seen in figure 7.5, the path probabilities of Grein-d are centered around 0.03125, while the path probabilities of Grein-L are much more scattered.

Another measure that can give us some insight, is the standard deviation, σ. This shows us how much variation from the sample average there is. Clearly, it should be as close to 0 as possible. In Grein-L, this value is 0.000257 and in Grein-d it is 0.000057. Clearly, both versions have very low standard deviation.

We also compute the correlation between probability of paths taken and the Hamming weight of the path, and find that for Grein-L it is 0.7287, where it is 0.1252 for Grein-d. This is also supported by the plots in figure 7.5.

That some paths are more likely to be chosen in Grein-L is something that can be used by a cryptanalysist, so it seems clear that Grein-d has an advantage in this regard.

7.6 S-box Distributions

As with the paths through the tree, we want the output of the S-box S_2 to be as uniform as possible. This means that it should output the 16 different output values an even number of times, or $\frac{1}{16} = 0.0625$.

By looking at the plots in figure 7.6, we can see that both versions of Grein are centered around $\frac{1}{16}$, but Grein-d has a smaller difference.

Computing the standard deviation, we find that for Grein-L it is 0.00049, and for Grein-d it is 0.000082. Again, both these values are quite close to 0, as we wanted. We can also compute the correlation coefficient of the S-box distributions. Here we find that correlation in Grein-L is -0.2878 and the correlation in Grein-d is -0.0554.

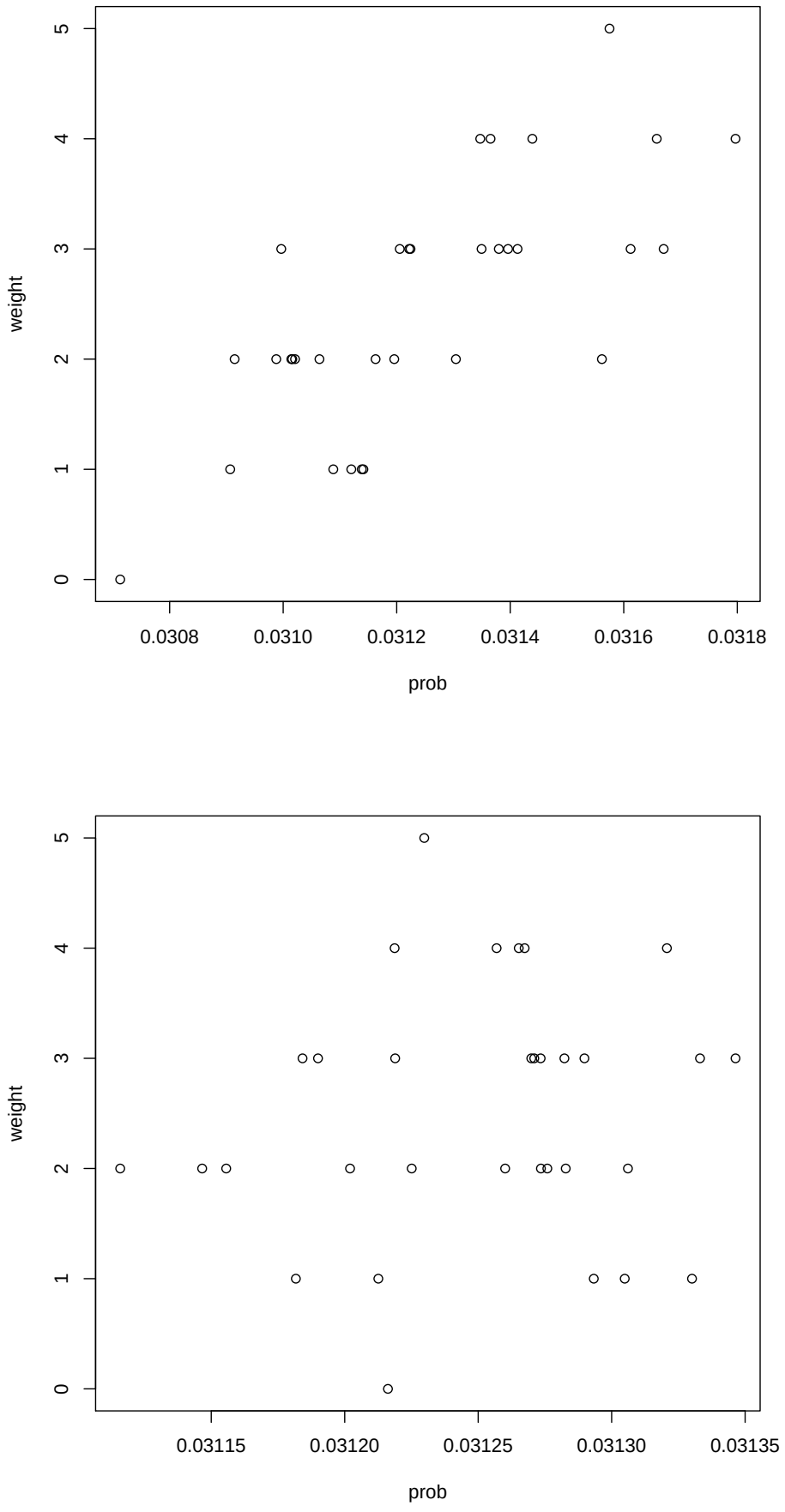

FIGURE 7.5: Plot of path distributions. Grein-L on top, Grein-d on the bottom. Probabilites along the x-axis, Hamming-weight along the y-axis.

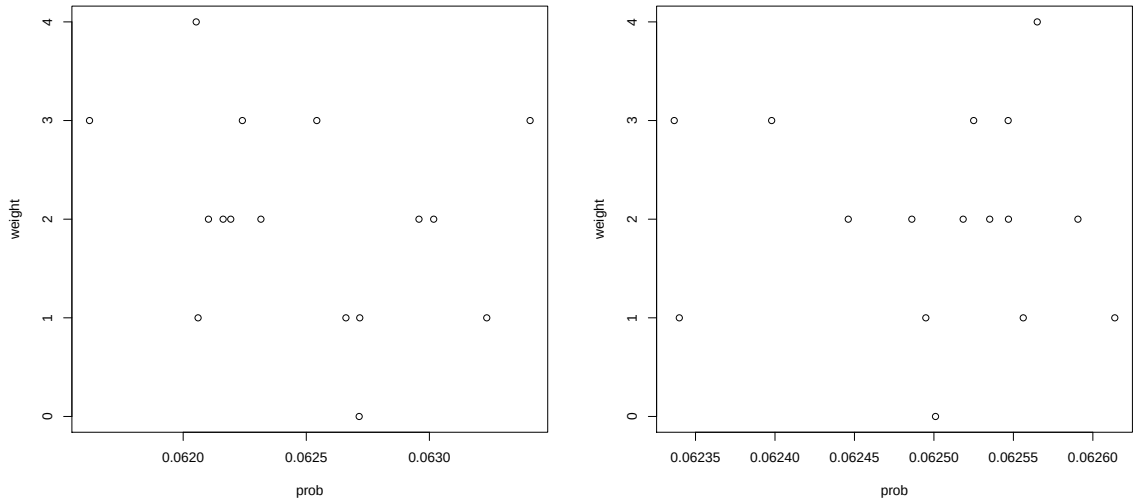

FIGURE 7.6: S-box plots. Grein-L on the left, Grein-d on the right

Looking at the plots in figure 7.6 and these statistical results, it seems that the S-box chosen

7.7 Feedback

The feedback function f_2, as seen in figure 7.7, whose output is combined with the output of the 80-bit LFSR to get the value X_1^{t+1} is one other part of the cipher that could also be exploited by a cryptanalysist. As the image of f_2 is $\mathbb{F}_2$, the function should output 1 as often as it outputs 0. By again running the cipher 10000000 times and counting the occurences of zeroes and ones, this should give a fair measure of how balanced the output is.

First, we check Grein-L, and find that f_2 outputs 0 0.4998166 percent of the time. We then do the same with Grein-d, where we find that it outputs 0 0.4999068 percent of the time.

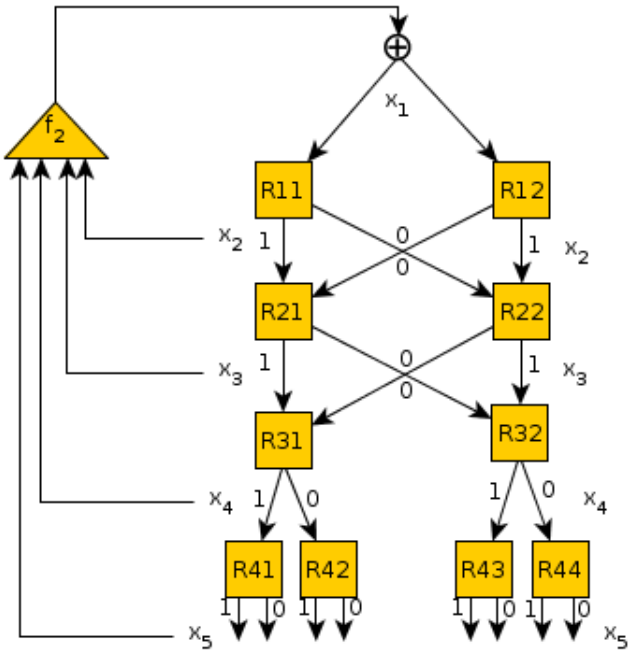

FIGURE 7.7: How the feedback function f_2 is integrated

8

Conclusions

By interpreting the results from the previous chapter, it seems clear that both Grein-L and Grein-d are suitable as stream ciphers, with Grein-d having some slight advantages, due to the more uniform distribution of the paths through the tree.

These advantages seem to be mitigated by the S-box, however, the use of de Bruijn sequences will also have the benefit of increasing the linear complexity.

8.1 Further Work

Some research should be done to test if other configurations of the tree offer more or less security. Shift registers of length 4, 5, 10 and 16 could be used to get a keyspace of length 80.
Alternatively, shift registers of differing lengths can be used, as long as they add up to a keyspace of length 80.

Appendix A

Tapping Matrices

$$M_{11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{21} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad M_{22} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{31} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{32} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M_{41} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad M_{42} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{43} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{44} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Appendix B

Path Distributions

In this appendix, the path distributions from the tree in section 6.3.

n_x	$p_l(n_x)$	$p_d(n_x)$	n_x	$p_l(n_x)$	$p_d(n_x)$
0	0.0307346	0.0312174	16	0.0309148	0.0312376
1	0.0309829	0.0312268	17	0.0308432	0.0312359
2	0.0310657	0.0312097	18	0.0312283	0.0312306
3	0.0311299	0.0312139	19	0.0310167	0.0312817
4	0.0311225	0.0312157	20	0.0309946	0.0312419
5	0.0310453	0.0312125	21	0.0312419	0.0312498
6	0.0315369	0.0312556	22	0.0313602	0.0312689
7	0.0313628	0.0313893	23	0.0314163	0.0312586
8	0.0312049	0.0312717	24	0.0310208	0.031258
9	0.0311095	0.0312768	25	0.0312376	0.0312358
10	0.0314744	0.0312182	26	0.0313357	0.0312535
11	0.0312032	0.0312223	27	0.0313944	0.0312515
12	0.0312619	0.0312562	28	0.0313915	0.0312414
13	0.0314775	0.0312343	29	0.0313004	0.0312246
14	0.031616	0.0312916	30	0.0317637	0.0312401
15	0.0316659	0.0312729	31	0.0315459	0.0313038

TABLE B.1: Probabilities of path distributions. p_l is the original, LFSR-based Grein, while p_d is the new, de Bruijn-based Grein

id	lfsrprob	dbprob
0	0.0627148	0.062501
1	0.0627172	0.0623398
2	0.063233	0.0624949
3	0.0621932	0.0625351
4	0.0620605	0.0625562
5	0.0621027	0.0624861
6	0.0621622	0.0624461
7	0.0625427	0.062525
8	0.062661	0.0626138
9	0.0629577	0.0625469
10	0.0630175	0.0625906
11	0.063409	0.0625467
12	0.0623157	0.0625184
13	0.0616196	0.0623979
14	0.0622404	0.0623366
15	0.0620529	0.062565

Appendix C

Statistical Test Results

In this appendix, the results of the tests introduced in section 7.4 are listed.

The input to the tests were 100,000,000 bits generated by Grein-L and Grein-d, respectively.

These bits were divided into 100 distinct keystreams. The reason for the large number of bits is because some of the tests require a minimum of 1,000,000 bits as input, and these were divided into 100 bitstreams of length 1,000,000.

The same battery of tests were also run on the keystream produced by an LFSR of length 30, with a period of $2^{30} - 1$.

In Table C.1 the results for Grein-L can be seen. The results for Grein-d are found in table C.2. For comparison, the results for the LFSR of length 30 are in table C.3.

P-VALUE	PROPORTION	STATISTICAL TEST
0.911413	10/10	Frequency
0.739918	10/10	BlockFrequency
0.739918	10/10	CumulativeSums
0.739918	10/10	CumulativeSums
0.213309	10/10	Runs
0.017912	10/10	LongestRun
0.213309	10/10	Rank
0.534146	10/10	FFT
0.534146	10/10	NonOverlappingTemplate
0.534146	10/10	OverlappingTemplate
0.739918	10/10	Universal
0.350485	10/10	ApproximateEntropy
0.452799	60/61	RandomExcursions
0.452799	61/61	RandomExcursionsVariant
0.739918	10/10	Serial
0.213309	9/10	Serial
0.350485	10/10	LinearComplexity

TABLE C.1: Statistical tests on Grein-L

P-VALUE	PROPORTION	STATISTICAL TEST
0.213309	10/10	Frequency
0.350485	10/10	BlockFrequency
0.739918	10/10	CumulativeSums
0.534146	10/10	CumulativeSums
0.035174	10/10	Runs
0.534146	10/10	LongestRun
0.739918	10/10	Rank
0.739918	10/10	FFT
0.122325	10/10	NonOverlappingTemplate
0.911413	10/10	OverlappingTemplate
0.911413	10/10	Universal
0.350485	10/10	ApproximateEntropy
0.494392	58/58	RandomExcursions
0.289667	57/58	RandomExcursionsVariant
0.122325	10/10	Serial
0.350485	10/10	Serial
0.350485	10/10	LinearComplexity

TABLE C.2: Statistical tests on Grein-d

P-VALUE	PROPORTION	STATISTICAL TEST
0.637119	96/100	Frequency
0.534146	100/100	BlockFrequency
0.964295	96/100	CumulativeSums
0.191687	97/100	CumulativeSums
0.779188	99/100	Runs
0.971699	100/100	LongestRun
0.000000	0/100	Rank
0.224821	100/100	FFT
0.554420	97/100	NonOverlappingTemplate
0.017912	99/100	OverlappingTemplate
0.574903	99/100	Universal
0.366918	94/100	ApproximateEntropy
0.289667	56/56	RandomExcursions
0.419021	56/56	RandomExcursionsVariant
0.005358	97/100	Serial
0.162606	99/100	Serial
0.000000	0/100	LinearComplexity

TABLE C.3: Statistical tests on LFSR of length 30

Bibliography

[1] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks,Alan Heckert, James Dray, San Vo, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. National Institute of Standards and Technology, Special Publication 800-22 Rev. 1a, 2010

[2] Nicolaas G. de Bruijn, *A Combinatorial Problem*. Proc. Koninklijke Nederlandse Akademie v. Wetenschappen, 1946.

[3] Elwyn Berlekamp, *Nonbinary BCH decoding*, International Symposium on Information Theory, San Remo, Italy, 1967.

[4] Elwyn Berlekamp, *Algebraic Coding Theory* Laguna Hills, CA: Aegean Park Press, 1984.

[5] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, Tolga Yalçın, *PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications* Cryptology ePrint Archive, Report 2012/529, 2012.

[6] Tor E. Bjørstad, *Cryptanalysis of Grain using Time/Memory/Date Tradeoffs*. 2008.

[7] Claude Carlet, *Boolean Function for Cryptography and Error Correcting Codes*. 2006.

[8] Claude Carlet, *Vectorial Boolean Functions for Cryptography*. 2006.

[9] Agnes Hui Chan, Richar A. Games, Edwin L. Key, *On the Complexities of de Bruijn Sequences*. Journal of Combinatorial Theory, Series A 33: 233–246 1982

[10] Thomas W. Cusick, Pantelimon Stănică, *Cryptographic Boolean Functions and Applications*. Academic Press, 2009.

[11] Taher ElGamal, *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE Transactions on Information Theory 31 (4): 469–472, 1985.

[12] Harold Fredricksen, *A Survey of Full Length Nonlinear Shift Register Cycle Algorithms..* SIAM Review, 1982.

[13] Richard A. Games, Agnes H. Chan, *A Fast Algorithm for Determining the Complexity of a Binary Sequence with Period* 2^n. IEEE Transactions on Information Theory 29 (1): 144-146, 1983.

[14] Dieter Gollman and William G. Chambers, *Clock-Controlled Shift Registers: A Review.* IEEE J Selected Areas Communications 1989;7, 1989.

[15] Solomon W. Golomb, *Shift Register Sequences.* 1967.

[16] Christoph Günther, Alternating Step Generators Controlled by de Bruijn Sequences. Advances in Cryptology - EUROCRYPT '87, LNCS 304, (5-14), 1988.

[17] Martin Hell, Thomas Johansson and Willi Meier, *Grain - A Stream Cipher for Constrained Environments.* IJWMC 86-93, 2007.

[18] Edwin L. Key, *An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators.* IEEE Transactions of Information Theory, Vol. IT-22, (732-736), 1976.

[19] Özgül Küçük, *Slide Resyncronization Attack on the Initialization of Grain 1.0.*

[20] Rudolf Lidl and Harald Niederreiter, *Finite Fields.* Cambrigde University Press, 2008.

[21] Kalikinkar Mandal and Guang Gong, *Cryptographically Strong de Bruijn Sequences with Large Periods.* Lecture Notes in Computer Science, Volume 7707, (104-118), 2013.

[22] Camille Flye-Sainte Marie, *Solution to Problem Number 58.* l'Intermediare des Mathematiciens, 1894.

[23] James L. Massey, *Shift-register synthesis and BCH decoding.* IEEE Trans. Information Theory, IT-15, (122-127), 1969.

[24] Johannes Mykkeltveit, M-Keung. Siu and P. Tong, *On the Cycle Structure of Some Nonlinear Shift Register Sequences.* Information and Control, (202-215), 1979.

[25] Ronald Rivest, Adi Shamir, Leonard Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM 21, (120–126), 1978.

[26] Sondre Rønjom, *Grein*. Nasjonal Sikkerhetsmyndighet, 2013.

[27] Ernst S. Selmer, *Linear Recurrence Relations over Finite Fields*. Mathematics Department, University of Bergen, 1966.

[28] Claude E. Shannon, *Communication Theory of Secrecy Systems*. Bell System Technical Journal, 1946.

[29] Thomas Siegenthaler, *Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications*. IEEE Transactions on Information Theory 30, (776–780), 1984.

[30] Douglas R. Stinson, *Cryptography: Theory and Practice, Third Edition*. Chapman and Hall/CRC, 2005.