

Detection and Exploitation of Small Correlations in Stream Ciphers

Masterthesis

conducted under the guidance of
Prof. Dr. Joachim Rosenthal
and
Dr. Gérard Maze
Institute of Mathematics, University of Zurich

2008

Urs Wagner

Outline

This thesis gives an overview of stream ciphers based on linear feedback shift registers (LFSR) and their vulnerability to correlation attacks. In the first chapter, a short introduction to symmetric key ciphers is given. The main focus hereby is on LFSR based stream ciphers. Further, the principles of LFSR are presented. The chapter is then closed by a stream cipher example, the Geffe Generator. The second chapter treats the general approach of correlation attacks. Moreover a correlation attack is mounted on the Geffe Generator and the practical results are presented.

Boolean functions play an important role in stream cipher designs. The Walsh transform, a tool to analyze the cryptographic properties of Boolean functions, is introduced in chapter 3. Additionally, the cryptographic properties themselves are discussed. In the fourth chapter, an improved kind of correlation attack -the fast correlation attack- is presented. It exploits the same weaknesses in the stream cipher designs as the correlation attack, the mode of operation is however different. In the last chapter, the insights gained in the previous chapters are used to suggest an attack on a stream cipher by Philips, named Hitag 2.

Acknowledgments

This thesis was written in the course of my master's studies at the University of Zurich. I am grateful to Prof. Joachim Rosenthal who gave me the opportunity to write my master thesis in cryptography. Special thanks go to Dr. Gérard Maze who supervised me during the progress of this work.

Contents

Outline	1
Acknowledgments	2
1 Symmetric key ciphers	5
1.1 Block ciphers	6
1.2 Stream ciphers	7
1.2.1 Additive stream ciphers	7
1.3 Keystream generators	8
1.3.1 LFSR	9
1.3.2 LFSR based keystream generators	17
1.3.3 An Example: The Geffe Generator	18
2 Correlation Attack	21
2.1 Idea	21
2.2 A basic correlation attack algorithm	23
2.2.1 Correlation attack on the Geffe Generator	26
2.3 Statistical Analysis of the Correlation Attack	27
2.3.1 Maximum likelihood Test	27
2.3.2 Chebyshev's Inequality	29
2.3.3 Conclusions	32
3 Boolean functions	33
3.1 Walsh Transform	33
3.2 Computing the Walsh-Hadamard Transform	35
3.3 Cryptologic properties of Boolean functions	37
3.3.1 Algebraic Degree	37
3.3.2 Nonlinearity	38
3.3.3 Correlation Immunity	40
3.3.4 Tradeoffs	41
3.4 An Example	42
3.4.1 Distinguishing attack	43
3.5 Cascading of boolean functions	44
4 Fast correlation attack	48
4.1 Block Code model	48
4.2 The original Fast Correlation attack	50
4.2.1 Algorithm A	53
4.2.2 Algorithm B	54

4.3	Improvements of the fast correlation attack	55
4.3.1	Iterative algorithms	55
4.3.2	Non-iterative algorithms	55
5	Hitag 2	61
5.1	The stream generator	61
5.1.1	The initialization process	62
5.1.2	The output function	62
5.1.3	Characteristic polynomial of the LFSR	63
5.2	ML decoding	64
5.3	Attack	65
5.3.1	A distinguishing attack on Hitag 2	65
5.3.2	Suggestions for a fast correlation attack	66
A	Linear Block codes	67
A.0.3	Decoding	68
B	LDPC codes	69

Chapter 1

Symmetric key ciphers

Symmetric key ciphers, also known as secret key ciphers, are an important class of cipher systems. Generally it can be said that they are much faster than Public key ciphers as the RSA [Mol03]. Symmetric key ciphers are characterized by the fact that the same key is used for encryption and decryption. Hence, two communication parties have to share a secret key. The exchange of the secret key between the communication parties is clearly crucial. A third party in possession of the secret key could decrypt all messages exchanged between the communication parties. There are protocols for the key exchange, for example the Diffie Hellmann key exchange protocol [Mol03]. Another way is to use RSA for the secret key exchange. Sometimes however, the exchange of the secret key happens offline and is therefore insensible to eavesdroppers. An easy example is the remote entry system in cars. The manufacturer chooses a secret key and saves it in the car-lock and the according car key. Anybody not in possession of a car key with the correct secret key is thus able to open the car. In the progress of this work, the exchange of the secret key will not be discussed any further. The focus will be on the encryption and decryption algorithms under the assumption that no third party has access to the secret key. Fairly effective encryption and decryption algorithms for symmetric key ciphers allowing fast communication will be presented. Let us start with the basic definitions.

Definition 1. *An alphabet is a set of symbols.*

Definition 2. *Let M and C be sets consisting of finite, nonempty strings of symbols from suitable alphabets. Let K be an arbitrary nonempty set. A symmetric key cipher then consists of two maps*

$$\phi : M \times K \rightarrow C \quad \text{and} \quad \psi : C \times K \rightarrow M,$$

such that

1. *for a fixed $m \in M$, $\phi_m : k \mapsto \phi(m, k)$ is a one-way function, meaning that $\phi_m(k)$ can effectively be computed for all $k \in K$, but it is practically impossible to compute $k \in \phi_m^{-1}(c)$ for almost all $c \in \text{Im } \phi_m$*
2. *For given $k \in K$, $\phi_k : m \mapsto \phi(m, k)$ and $\psi_k : c \mapsto \psi(c, k)$ are bijective and $\psi_k(\phi_k(m)) = m$ for all $m \in M$. Hence for every $k \in K$ we have a one-to-one correspondence between the elements in M and the elements in C .*

M is then called the message space, C is called the cipher space and K is the set of all possible keys.

The bijectivity of ϕ_k and ψ_k in the definition above is not absolutely necessary. Injectivity would be sufficient, but we will stick to this definition so that every codeword has a valid decryption. For convenience we will use the binary alphabet $\{0, 1\}$ for both, the message- and the cipherspace. Note that this is no restriction, since every symbol of an alphabet of size S can be encoded in bitstrings of length $\lceil \log_2(S) \rceil$.

There are two standard approaches for symmetric-key ciphers, the *block ciphers* and the *stream ciphers*.

1.1 Block ciphers

In a block cipher, messages are split into blocks of equal length n . A key $k \in K$ is used to transform n -bit plaintext blocks in n -bit ciphertext blocks. Hence the block cipher defines a permutation in $\{0, 1\}^n$. As direct consequence, two identical plaintext blocks are always encrypted into two identical ciphertext blocks. In cryptography, this is a highly undesirable property. Patterns in the plaintext are observable in the ciphertext. An attacker can analyze these patterns to retrieve some information about the plaintext. That is why different modes of operation have been created.

For example, the Cipher Block Chaining mode can be used (see Figure 1.1). A plaintext block is not directly encrypted anymore. Rather, the preceding ciphertext block is XORed to the plaintext block before encryption. It is this XOR sum that is then encrypted. Decryption is then done in the following way. First, the ciphertext is decrypted. The resulting textblock is then XORed to the preceding ciphertext block, which yields to the original plaintext block again. In this way patterns will not be recognizable in the ciphertext. Clearly if an error occurs during the transmission of a ciphertext block, the ciphertext block is decrypted incorrectly. Since the erroneous ciphertext block is used to retrieve the message from the following ciphertext block, also the following ciphertext block will be decoded incorrectly. The error does not propagate further though. Hence we have only limited *error propagation*.

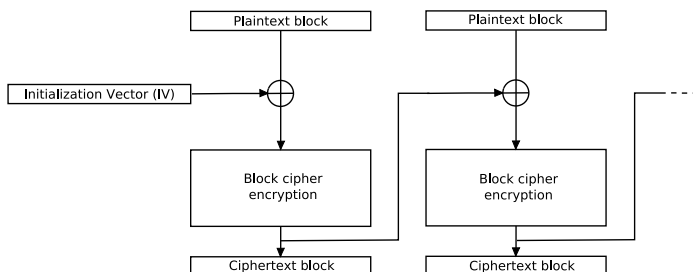


Figure 1.1: Cipher Block Chaining mode encryption

We will not go deeper into block ciphers here. The interested reader is referred to [CBP06] where a nice introduction to block cipher cryptanalysis is given.

1.2 Stream ciphers

In stream ciphers, the block size to be encrypted normally equals the size of one character in the alphabet. As we use the binary alphabet $\{0, 1\}$, the encryption is done bit by bit. Contrary to block ciphers, where the encryption is the same for each block, the encryption in stream ciphers is time-varying, i.e. the encryption varies from bit to bit. Further, every bit is encrypted separately and independently from the previous or following bits. Consequently, patterns in the plaintext are not recognizable in the ciphertext and error propagation is very limited.

An important class of stream ciphers are the additive stream ciphers.

1.2.1 Additive stream ciphers

Let us start with a famous example of an additive streams cipher, the so-called *one-time pad* (OTP). Let $m = (m_1, \dots, m_N) \in \mathbb{F}_2^N$ be the N message-bits to be encrypted. Let further $k = (k_1, \dots, k_N) \in \mathbb{F}_2^N$ be a sample of the random vector (X_1, \dots, X_N) , where X_1, \dots, X_N are identically independently distributed random variables with $\Pr(X_i = 1) = \Pr(X_i = 0) = \frac{1}{2}$. The ciphertext is computed by

$$c_i = m_i \oplus k_i, \quad i = 1, \dots, N,$$

and decrypted by

$$m_i = c_i \oplus k_i \quad i = 1, \dots, N.$$

The security of the OTP comes from the fact that, given the observed ciphertext, every plaintext is equally likely. This can easily be seen since k is a random vector in \mathbb{F}_2^N with uniform distribution. For each transmission a new random key k must be used, the explanation follows.

Assume an attacker knows the plaintext m of a ciphertext c encrypted by the OTP with a secret key k . Obviously he can recover the secret key k by computing

$$k = m \oplus c.$$

This allows him to decrypt all further messages encrypted by the same key k . Hence, before each transmission of a message, a new secret key k has to be exchanged through a secure channel. Since the key has the same length as the message, the message could be sent through this secure channel in the first place.

However, the general approach of the OTP can still be used. If we can, having a secret key k of length l , generate a random looking sequence $r = (r_1, \dots, r_N)$ of length N , called *keystream*, we can encrypt messages of length N in the same way as in the OTP. Encryption is done by just XORing r to the message m and decryption by again XORing r to the ciphertext c . In fact, this is a widely used approach for stream ciphers, called *additive* stream ciphers. The task of generating a random looking keystream $r = (r_i)_{i \geq 0}$ from a short key has to be solved. Algorithms which generate keystreams from a secret key

are called *keystream generators*. From the security standpoint there are two important notes that have to be stated here:

1. Suppose the attacker knows the plaintext m of a ciphertext c , then he can easily compute the keystream used for encryption by computing $r = m \oplus c$. Hence, the keystream r must be generated in a one-way fashion from the secret key, i.e. the attacker must not be able to retrieve the secret key knowing the keystream.
2. As in the OTP, the keystream must differ for every message sent. However, the secret key is normally used several times. Thus the keystream generator must not generate the same keystream twice, although the same secret key is repeatedly used. Often *initialization vectors* (IV) are applied to ensure varying keystreams. An IV is a public known variable that is used together with the secret key to generate the keystream (see Figure 1.2). The IV is hereby chosen differently for each transmission resulting in different keystreams. Clearly 1. must hold also in this case. Knowing the keystream and the IV must not allow the attacker to compute the secret key.

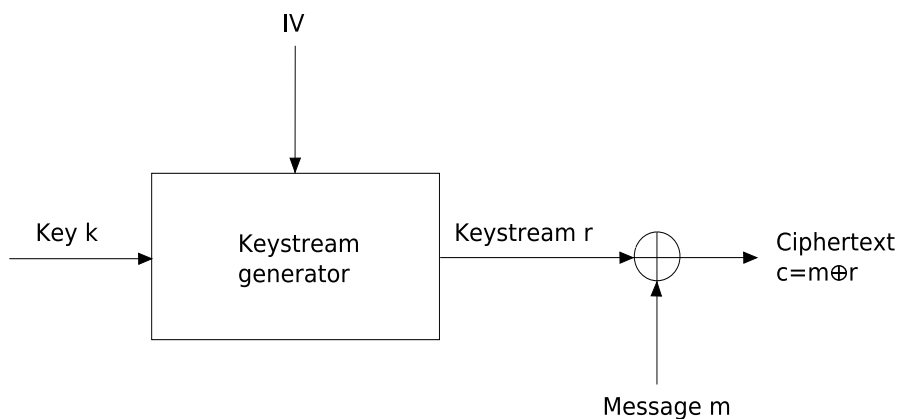


Figure 1.2: An additive stream cipher design

As we have seen, the main challenge in additive stream ciphers is the generating of a suitable keystream of large length from a relatively short key. In the next section we will see how this can be achieved.

1.3 Keystream generators

In this section, we will present common approaches for keystream generators. As already discussed, keystreams can be used to encrypt plaintext streams by just adding the keystream to the the plaintext stream. We have seen in the discussion of the OTP that the ciphertext will be completely random if the keystream is completely random. Since we want to generate a keystream of length N from

a secret key of length l with $N > l$, the resulting keystream is not going to be truly random. It is supposed to look random however. That's why we talk of *pseudorandom sequences*. Pseudorandom sequences have the characteristic that they are not distinguishable from truly random sequences. Additionally the keystreams must be cryptographically secure [Mol03]. This means that the knowledge of the algorithm used to generate the keystream and the knowledge of all previous keystream bits does not allow the prediction of the following bit. This is no easy task and the reader interested in a more thorough discussion of randomness is referred to [Knu98]. In this section, keystream generators based on linear feedback shift registers (LFSR) are discussed. LFSRs generate keystreams in a linear way, which makes its computation very efficient. At the same time the linearity makes them easily predictable which is not desirable in a cryptographic setting. That is why they are not directly used as keystream generators, but as a basis for more advanced keystream generators.

1.3.1 LFSR

We will work in the finite field \mathbb{F}_2 with two elements $\{0,1\}$. The following could however easily be generalized for any finite field \mathbb{F}_q .

Architecture

An LFSR of length l consist of l registers, called *taps*. Each tap has a *feedback coefficient* $c_0, \dots, c_{l-1} \in \mathbb{F}_2$ which defines whether the bit in the according tap is used in the calculation of the feedback sum or not. We call the taps with feedback coefficient equal to 1 as the *feedback taps*. In each round, the assignment of the taps is updated. The elements in the taps are shifted to the left. The element in the leftmost tap is the output of the device in that round. The rightmost tap receives the mod 2 feedback sum of the bits in the feedback taps of the previous round (see Figure 1.3).

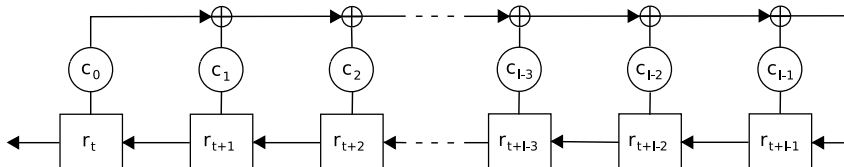


Figure 1.3: An LFSR of length l .

Let $r = (r_i)_{i \geq 0}$ be an output stream of the LFSR. Then r satisfies the linear recurrence relation

$$c_0 r_t + c_1 r_{t+1} + c_2 r_{t+2} + \dots + c_{l-1} r_{t+l-1} + r_{t+l} = 0, \quad t \geq 0.$$

The polynomial

$$f = c_0 + c_1 x + c_2 x^2 + \dots + c_{l-1} x^{l-1} + x^l.$$

in $\mathbb{F}_2[x]$ is called the *feedback polynomial* of the LFSR. A sequence r generated by the LFSR is completely determined by its l initial values $(r_0, \dots, r_{l-1}) =: R_0$,

called *initial state*. Hence the number of different sequences that the LFSR can generate is 2^l . The values of the taps at time step t , $R_t = (r_t, r_{t+1}, \dots, r_{t+l-1})$ is called *state t* of the LFSR.

Note that for $c_0 = 0$ the bits $(r_i)_{i>l}$ in the LFSR sequence r depend only on the $l-1$ preceding bits. Hence the whole sequence $r = (r_i)_{i \geq 0}$ could, except for the first bit r_0 , as well be generated by an LFSR of length $l-1$. That is why we have the convention that

$$c_0 := 1.$$

Due to this convention, the LFSR can be run in both directions, forwards and backwards. The backwards running LFSR can itself be seen as LFSR with the feedback polynomial

$$g = \text{rev}(f) = 1 + c_{l-1}x + c_2x^2 + \dots + c_1x^{l-1} + c_0x^l,$$

of degree l . It is hence clear that if the state $R_t = (r_t, \dots, r_{t+l-1})$ of the LFSR in round t is known, all the following and preceding streambits can be computed.

As already said, there are 2^l different states. The all zero start state gives rise to the all zero sequence. Further, since the LFSR can be run in both directions, the all zero state does only occur in the all zero sequence. Hence, we can state the following lemma.

Lemma 1. *The 2^l LFSR sequences have period at most $2^l - 1$.*

Proof. The all zero sequence produced by the all zero initial state has period 0. For non-zero initial states, the LFSR can run through at most the $2^l - 1$ different non-zero states before repeating itself. \square

Normally the period of an LFSR stream depends on both, the start state and the feedback polynomial. In our cryptographic setting, we desire sequences with large period in order to encrypt long messages. We will see that by choosing the feedback polynomial of degree l correctly, we get sequences of maximum possible period $2^l - 1$, called *m-sequences*. Note that this is equivalent to the fact that the LFSR runs through all possible non-zero states before repeating itself. Hence, the period in this case is independent of the (non-zero) initial state. The choice of the initial state is therefore unproblematic. In the next section we will see how the feedback polynomial must be chosen so that the LFSR generates sequences with maximum possible period.

Linear recurrence sequences

Let

$$f = c_0 + c_1x + c_2x^2 + \dots + c_{l-1}x^{l-1} + x^l,$$

be the feedback polynomial of an LFSR of length l . Let $(r_i)_{i \geq 0}$ be a sequence satisfying the according linear recurrence relation

$$c_0r_t + c_1r_{t+1} + c_2r_{t+2} + \dots + c_{l-1}r_{t+l-1} + r_{t+l} = 0, \quad t \geq 0,$$

then f is called a *characteristic polynomial* of r . We will see that a sequence $(r_i)_{i \geq 0}$ has more than one characteristic polynomial.

Let $\Omega(f)$ be the set of all infinite sequences satisfying the recurrence relation defined by f . Note that $\Omega(f)$ corresponds to the set of sequences that can be

generated by an LFSR with feedback polynomial f . We have already seen that $|\Omega(f)| = 2^l$. Let $r = (r_i)_{i \geq 0}$ be a sequence in $\Omega(f)$. We can write r as a formal power series

$$\hat{r} = \sum_{i \geq 0} r_i x^i = r_0 + r_1 x^1 + r_2 x^2 + \dots \in F[[x]].$$

Let g be the reversal of f ,

$$g := \text{rev}(f) = x^l f(x^{-1}) = c_l + c_{l-1}x + \dots + c_1 x^{l-1} + c_0 x^l.$$

Multiplying \hat{r} and g , we get the infinite sum

$$\begin{array}{cccccccc} g\hat{r} = c_l r_0 & + & c_l r_1 x & + & \dots & + & c_l r_l x^l & + & c_l r_{l+1} x^{l+1} & + & \dots \\ & & + & c_{l-1} r_0 x & + & \dots & + & c_{l-1} r_{l-1} x^l & + & c_{l-1} r_l x^{l+1} & + & \dots \\ & & & & & \ddots & & \vdots & & \vdots & & \\ & & & & & & & + & c_0 r_0 x^l & + & c_0 r_1 x^{l+1} & + & \dots \end{array}$$

By column-wise addition, we get

$$\begin{aligned} g\hat{r} = & c_l r_0 + x \underbrace{\sum_{0 \leq i \leq l-1} c_{i+l-1} r_i + \dots + x^{l-1} \sum_{0 \leq i \leq l-1} c_{i+1} r_i}_{=:h} \\ & + x^l \underbrace{\sum_{0 \leq i \leq l} c_i r_i}_{=:0} + x^{l+1} \underbrace{\sum_{0 \leq i \leq l} c_i r_{i+1}}_{=:0} + \dots \quad (1.1) \end{aligned}$$

We see that we get a polynomial h of degree less than l , since all higher degree terms vanish due to the recurrence relation. In fact, we have proven the following theorem.

Theorem 1. *Let $r = (r_i)_{i \geq 0}$ be linearly recurrent, $\hat{r} = \sum_{i \geq 0} r_i x^i$ the according formal power series, f a polynomial of degree l and $g = \text{rev}(f)$ its reversal. Then the following are equivalent*

1. f is a characteristic polynomial of r ,
2. $\hat{r}g$ is a polynomial of degree less than l ,
3. $\hat{r} = \frac{h}{g}$ for some polynomial $h \in \mathbb{F}_2[x]$ with $\deg h < l$.

From the discussion preceding the theorem, we saw that $h = \hat{r}g$ is completely determined by f and the first l bits (r_0, \dots, r_{l-1}) of the sequence. Conversely, \hat{r} , the sequence $(r_i)_{i \geq 0}$ respectively is uniquely defined by f and the polynomial h . In fact, every polynomial $h \in \mathbb{F}_2[x]$ of degree less than l uniquely defines a sequence satisfying the linear recurrence relation given by f . Since h has degree less than l , \hat{r} is ensured to have infinite degree. We have the following corollary.

Corollary 1. *Let f be a polynomial of degree l . There is a one-to-one correspondence between the sequences in $\Omega(f)$ and the polynomials of degree less than l in $\mathbb{F}_2[x]$.*

By Theorem 1, it is clear that a given recurrence sequence $r = (r_i)_{i \geq 0}$ has more than one characteristic polynomial. In fact, if f is a characteristic polynomial, also every multiple of f is a characteristic polynomial of r . Further if f_1 and f_2 are characteristic polynomials of r , also $f_1 + f_2$ is a characteristic polynomial of r . Hence the set of all characteristic polynomials of r , together with the zero polynomial, is an ideal \mathfrak{S}_r in $\mathbb{F}_2[x]$. We know that every ideal in $\mathbb{F}_2[x]$ is generated by a single element. Thus there is a polynomial m of least degree such that $\mathfrak{S}_r = \{mf : f \in \mathbb{F}_2[x]\}$. This element m is called the *minimal polynomial* of the sequence r .

Lemma 2. *Given two polynomials $f_1, f_2 \in \mathbb{F}_2[x]$ with nonzero constant terms, then*

$$\Omega(f_1) \subseteq \Omega(f_2) \Leftrightarrow f_1 | f_2.$$

In particular

$$\Omega(f_1) = \Omega(f_2) \Leftrightarrow f_1 = f_2.$$

Further

$$\Omega(f_1) \cap \Omega(f_2) \neq 0 \Leftrightarrow f_1 \text{ and } f_2 \text{ have a nontrivial factor in common.}$$

Proof. Let $g_1 := \text{rev}(f_1)$, $g_2 := \text{rev}(f_2)$. Suppose $f_1 | f_2$ and let $qf_1 = f_2$. Let $r = (r_i)_{i \geq 0} \in \Omega(f_1)$, \hat{r} its formal power series. Then $\hat{r} = \frac{h_1}{g_1}$ for some h_1 of degree less than $\deg(f_1)$. Hence

$$\hat{r} = \frac{h_1}{g_1} = \frac{h_1 \text{rev}(q)}{g_1 \text{rev}(q)} = \frac{h_1 \text{rev}(q)}{g_2}.$$

We have

$$\deg(h_1 \text{rev}(q)) = \deg(h_1) + \deg(\text{rev}(q)) < \deg(g_1) + \deg(\text{rev}(q)) = \deg(g_2).$$

Since f_1 has nonzero constant term the degree of g_2 equals the degree of f_2 and hence we get $\deg(h_1 \text{rev}(q)) < \deg(f_2)$. Thus we have $\Omega(f_1) \subseteq \Omega(f_2)$.

Suppose now $\Omega(f_1) \subseteq \Omega(f_2)$. Let us take the sequence $r = (r_i)_{i \geq 0} \in \Omega(f_1)$ defined by $\hat{r} = \frac{1}{g_1}$. From the precondition there exist some h such that $\hat{r} = \frac{h}{g_2}$. Hence we have $\frac{1}{g_1} = \frac{h}{g_2}$ which gives us (taking the reverses) $f_2 = \text{rev}(h)f_1$ and hence $f_1 | f_2$.

Suppose $r = (r_i)_{i \geq 0} \in \Omega(f_1) \cap \Omega(f_2)$. Let m be the minimal polynomial of r . Then m divides both f_1 and f_2 since m generates the ideal of all characteristic polynomials. \square

Corollary 2. *Let $f \in \mathbb{F}_2[x]$ be irreducible with $f(0) \neq 0$. Then f is the minimal polynomial of every sequence in $\Omega(f)$.*

We can now start to investigate the period of linear recurrence sequences. We need the definition of the period of a polynomial $f \in \mathbb{F}_2[x]$

Definition 3. *Let $f \in \mathbb{F}_2[x]$ be of degree l . The period $P(f)$ of f is the least positive integer e such that $f|(x^e - 1)$.*

Lemma 3. *Every polynomial $f \in \mathbb{F}_2[x]$ with nonzero constant term has a period $e < \infty$.*

Proof. We have $f(0) = 1$. Hence x does not divide f and so $\bar{x} = x \pmod{f} \in (\mathbb{F}_2[x]/(f))^*$. Clearly \bar{x} is of finite order b and hence $x^b - x = 0 \pmod{f}$. Hence $f|x^{b-1} - 1$. \square

Recall the convention that feedback polynomials have nonzero constant term. Consequently they have a period. The following lemma gives an important result on the period of the according recurrence sequences.

Lemma 4. *Let $f \in \mathbb{F}_2[x]$ be a polynomial of degree l with nonzero constant term and with $P(f) = e$. Let further $r = (r_i)_{i \geq 0} \in \Omega(f)$. Then the period of r divides e .*

Proof. Let \hat{r} again denote the formal power series of r and g the reverse of f . Hence we have $\hat{r} = \frac{h}{g}$ for some $h \in \mathbb{F}_2[x]$ of degree smaller than l . We further know that $f|x^e - 1$ and hence $fq = x^e - 1$ for some $q \in \mathbb{F}_2[x]$. Note that

$$1 - x^e = \text{rev}(x^e - 1) = \text{rev}(fq) = \text{rev}(f)\text{rev}(q) = \text{grev}(q).$$

Hence we have

$$\hat{r} = \frac{h\text{rev}(q)}{\text{grev}(q)} = \frac{h\text{rev}(q)}{1 - x^e},$$

and by Theorem 1 we have that $1 - x^e = x^e - 1 \in \mathbb{F}_2[x]$ is characteristic polynomial of r and hence $r_{t+e} = r_t$ for all $t \geq 0$. \square

Theorem 2. *Let f be an irreducible polynomial of degree l with nonzero constant term. Let further $P(f) = e$ and $0 \neq r = (r_i)_{i \geq 0} \in \Omega(f)$. Then the period of r equals e .*

Proof. Let p be the period of r . By Lemma 1.3.1 we know that $p|e$. Now suppose $p < e$. Note that $(1 + x^p + x^{2p} + \dots) = \frac{1}{x^p - 1} \in \mathbb{F}_2[[x]]$ and hence we can write

$$\hat{r} = \underbrace{(r_0 + r_1x + \dots + r_{p-1}x^{p-1})}_{=: \rho(x)}(1 + x^p + x^{2p} + \dots) = \frac{\rho(x)}{x^p - 1}.$$

Hence we have

$$\frac{\rho(x)}{x^p - 1} = \frac{h(x)}{g(x)},$$

with $g = \text{rev}(f)$ and $h \in \mathbb{F}_2[x]$ of degree less than l . We get that $\rho g = h(x^p - 1)$ and reversing the whole equation $\text{rev}(\rho)f = \text{rev}(h)(x^p - 1)$ which leads to $f|\text{rev}(h)(x^p - 1)$. Note that h has degree smaller than f and f cannot have a factor in common with $\text{rev}(h)$ due to its irreducibility. Thus $f|(x^p - 1)$ which is a contradiction to the minimality of e . Hence the period of r equals e . \square

Corollary 3. *The period of a sequence r equals the period of its minimal polynomial f .*

Proof. Let p be the period of r . By Lemma 1.3.1 we have that $p|P(f)$. It remains to show that $P(f)|p$. Let h and ρ be as in the proof of Theorem 2 where we come to the conclusion that $f|\text{rev}(h)(x^p - 1)$. Since f is the minimal polynomial it cannot have a factor in common with $\text{rev}(h)$. Else the fraction $\frac{h}{f}$ could be shortened, leading to a characteristic polynomial of lower degree. We get $f|(x^p - 1)$ and hence $P(f)|p$. \square

Remark 1. *It can be shown [Wan03] that the period of an irreducible polynomial equals the order of any of its roots.*

Corollary 4. *All nonzero sequences from an LFSR with irreducible characteristic polynomial have equal period.*

m-sequences

We have seen that LFSRs with irreducible polynomials generate sequences whose periods do not depend on the initial states. In particular, for any nonzero initial state the period of the resulting sequences equals the period of the characteristic polynomial. In fact, the irreducible characteristic polynomial is just the minimal polynomial of all sequences produced by the LFSR, meaning that no LFSR of shorter length could generate any of these sequences. Hence the existence of LFSRs with maximum possible period $2^l - 1$ is equivalent to the existence of irreducible polynomials of degree l and period $2^l - 1$.

In this section we will show that we can find irreducible polynomials in $\mathbb{F}_2[x]$ of degree l period $2^l - 1$. In the discussion of LFSR sequences we have already seen that this is in fact the maximum possible period of sequences generated by an LFSR of length l .

Definition 4. *Let \mathbb{F}_q be a finite field. Generators of the cyclic group \mathbb{F}_q^* are called primitive elements of \mathbb{F}_q .*

Definition 5. *A polynomial $f \in \mathbb{F}_2[x]$ of degree l is called primitive, if it has a primitive element α of \mathbb{F}_{2^l} as one of its roots.*

Lemma 5. *Primitive polynomials are irreducible*

Proof. Let α be a primitive element of \mathbb{F}_{2^l} . Let $f \in \mathbb{F}_2[x]$ be a polynomial of degree l with α as one of its roots. Then also $\alpha^2, \alpha^4, \dots, \alpha^{2^{l-1}}$ are roots of f and they are all different. Hence

$$f = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2^{l-1}}),$$

and f is the polynomial of lowest degree in $\mathbb{F}_2[x]$ having α as one of its roots. Suppose that $f = m_1 m_2$ with $m_1, m_2 \in \mathbb{F}_2[x]$ and $1 \leq \deg(m_1), \deg(m_2) \leq l-1$. Then with $f(\alpha) = 0$ either $m_1(\alpha) = 0$ or $m_2(\alpha) = 0$, what is a contradiction to the minimality of f . \square

Theorem 3. *The period $P(f)$ of an primitive polynomial $f \in \mathbb{F}_2[x]$ of degree l with nonzero constant term equals $2^l - 1$.*

Proof. Let α be a primitive element in \mathbb{F}_{2^l} and a root of f . Clearly then, $\alpha^2, \alpha^4, \alpha^8, \dots, \alpha^{2^{l-1}}$ are the other roots of f and all of them have order $e = 2^l - 1$. Hence $f|(x^e - 1)$ and e is the smallest number with this property, since the roots of f are also roots of $x^e - 1$. \square

Remark 2. *It can even be shown [Wan03] that the period of a polynomial of degree l equals $2^l - 1$ if and only if it is primitive.*

Corollary 5. *Nonzero linear recurrence sequences with primitive characteristic polynomial f of degree l have maximum period $2^l - 1$.*

Proof. Clear with Theorem 3 and 2 since primitive polynomials are irreducible. \square

We see that an LFSR with primitive characteristic polynomial and nonzero initial states generates only m -sequences. Thus the question arises, if we are able to find primitive polynomials of a given degree. In fact, it can be shown [Wan03] that the number of primitive polynomials of degree l over a finite field \mathbb{F}_q equals

$$\frac{\Phi(q^l - 1)}{l},$$

where $\Phi(d)$ denotes the Euler Φ -function and is the sum of all positive divisors of d . This follows from the fact that there are $\Phi(q^l - 1)$ primitive elements in \mathbb{F}_{q^l} and each element together with its l conjugates has a unique minimal polynomial of degree l .

State Transition Matrix

An alternative way to describe an LFSR is by means of its *state transition matrix*. Let again

$$f = c_0 + c_1x + c_2x^2 + \dots + c_{l-1}x^{l-1} + x^l, \quad (1.2)$$

be the feedback polynomial of an LFSR. The according state transition matrix G is then defined by

$$G := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ c_0 & c_1 & c_2 & \dots & c_{l-1} \end{pmatrix}. \quad (1.3)$$

Let $R_0 = (r_0, \dots, r_{l-1})$ be the initial state of the LFSR. The consecutive states $R_i = (r_i, \dots, r_{i+l-1})$, $i \geq 0$, can then be computed by

$$R_i = G^i R_0.$$

It is obvious that if $G^i = I_l$, then $R_i = R_0$ and hence period of r divides the order of G . In fact, we can even show equality. We need to make some observations first.

Up to now we have only considered linear recurrence sequences with elements in \mathbb{F}_2 . It is easy to see that the obtained results hold for any sequence with elements in any vector space V over \mathbb{F}_2 [GG03]. In particular, for

$$A \in V = \mathbb{F}_2^{l \times l}$$

the sequence $a = (A^i)_{i \geq 0}$ is linear recurrent and the characteristic polynomial $X_A = \det(A - xI_l) \in \mathbb{F}_2[x]$ where I_l is the $l \times l$ identity matrix, is a characteristic polynomial of a , by the Caley-Hamilton theorem.

Let f be a feedback polynomial as in (1.2) and G its state transition matrix as in (1.3). It is easy to see that f is a characteristic polynomial of the sequence $(G^i)_{i \geq 0}$. We can even show that it is its minimal polynomial.

Theorem 4. *Let f be a feedback polynomial as in (1.2) and G its state transition matrix as in (1.3). Then f is the minimal polynomial of G , meaning that it is the minimal polynomial of the sequence $(G^i)_{i \geq 0}$.*

Proof. Let $g = g_0 + g_1x^1 + \dots + g_{n-1}x^{n-1} + x^n$ be a polynomial of degree $n < \deg(f) = l$ such that

$$g(G) = g_0I_l + g_1G^1 + \dots + g_{n-1}G^{n-1} + G^n = 0,$$

where I_l is the $l \times l$ unit matrix. It follows that for every $R_0 \in \mathbb{F}_2^l$ we have

$$g_0R_0 + g_1G^1R_0 + \dots + g_{n-1}G^{n-1}R_0 + G^nR_0 = 0,$$

and hence with $R_i = G^iR_0$

$$g_0R_0 + g_1R_1 + \dots + g_{n-1}R_{n-1} + R_n = 0.$$

Thus we have $\Omega(f) \subseteq \Omega(g)$ and hence $f|g$ which is a contradiction. \square

Corollary 6. *Given f and G as above we have*

$$\mathbb{F}_2[G] \cong \mathbb{F}_2[x]/(f).$$

In particular the period of a feedback polynomial f equals the period of its state transition matrix G .

Cryptologic properties of m-sequences

We have seen how the feedback polynomial of an LFSR has to be chosen, so that the LFSR produces sequences with maximum period. The initial state of the LFSR can be used as a secret key to generate a keystream. We will see however, that having access to a long enough keystream-sequence, an attacker can recover the initial state of the keystream generator quite fast. Thus the generation of the keystream is not in a one-way fashion. That's why LFSR-sequences cannot be used directly for encryption in an additive stream cipher.

Definition 6. *The linear complexity of a keystream $r = (r_i)_{i \geq 0}$ is the length of the shortest LFSR which can generate this sequence. It is hence the degree of its minimal polynomial.*

Let $r = (r_i)_{i \geq 0}$ be the sequence with linear complexity l . Suppose an attacker has access to (r_t, \dots, r_{t+2l-1}) , e.g. by a known plaintext attack. He then can recover the feedback coefficients of the LFSR by solving

$$\begin{pmatrix} r_t & r_{t+1} & \dots & r_{t+l-1} \\ r_{t+1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ r_{r+l-1} & \dots & \dots & r_{t+2l-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ c_{l-1} \end{pmatrix} = \begin{pmatrix} r_{t+l} \\ r_{t+l+1} \\ \vdots \\ r_{t+2l-1} \end{pmatrix}. \quad (1.4)$$

This system of linear equations can be solved by Gauss elimination in $O(l^3)$ field operations. Berlekamp and Massey gave an algorithm [GG03] to recover the minimal polynomial in $O(l^2)$ field operations. Clearly knowing the feedback coefficients, an attacker can recover the whole sequence and in particular

the initial state. That's why LFSR sequences should not directly be used as keystreams in an additive stream cipher design.

Still, LFSR sequences serve as a basis for several keystream generators as presented in the next section. The reason for this, apart from the fact that we can efficiently generate sequences of guaranteed large period, lies in the fact that m -sequences look random. A thorough discussion on the randomness of m -sequences is given for example in [BP82]. To sum up it can be said that the fact that the LFSR runs through all possible states in $\mathbb{F}_2^l \setminus \{0\}$ before repeating ensures a "good" pseudorandomness.

1.3.2 LFSR based keystream generators

In the following we will present some commonly used LFSR based keystream generator designs. One is the *nonlinear combination generator* as in Figure 1.4. The output bits of several LFSRs are combined in a nonlinear function F . The output of the generator equals the output of that function F . We will see later how the nonlinear function F dramatically increases the linear complexity of the keystream, so that the Berlekamp-Massey algorithm for finding the minimal polynomial of a keystream becomes infeasible. In the *nonlinear filter generator* as in Figure 1.5, only one LFSR is used. Bits from different taps are combined in a nonlinear function F .

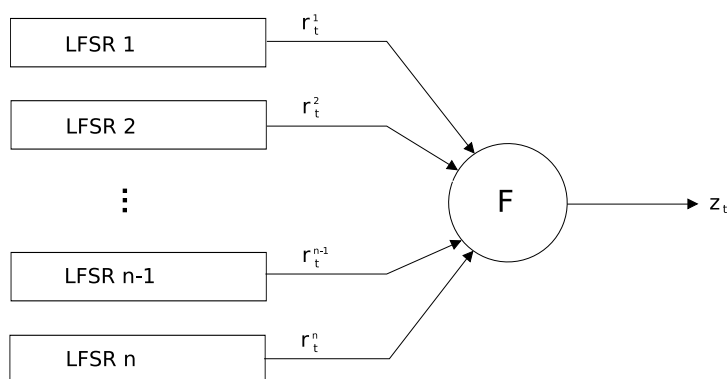


Figure 1.4: The nonlinear combination generator. The output of several LFSRs are combined in a nonlinear boolean function F

The nonlinear combination function F has to be chosen carefully. We will give conditions for F so that the resulting keystream has large linear complexity. This and other crucial properties of F such as the *correlation immunity* and the *nonlinearity* will be discussed in Section 3.3.

There is a third approach of an LFSR based keystream generator, the *clock controlled generator*. Here, the LFSRs are clocked using some irregular signal. The signal is used to determine which LFSR output is used in the calculation

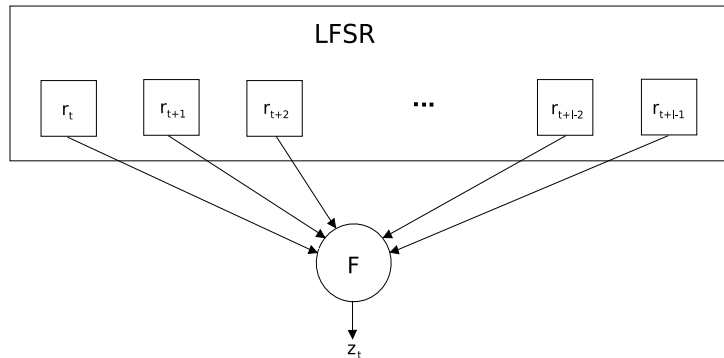


Figure 1.5: The nonlinear filter generator. Several bits from the stages of one LFSR are combined by a nonlinear function F .

of the output of the generator. There are many ways to implement a clock controlled generator. Consequently, it is hard to give general results on the performance of such generators. We will not go deeper into this architecture.

1.3.3 An Example: The Geffe Generator

The Geffe Generator is a well known nonlinear combining generator. It consists of three LFSRs with different periods. In each step, the output of the generator equals the output of the first or the second LFSR, depending on the output of the third LFSR (see Figure 1.6).

As a concrete example we will examine the Geffe Generator with the three LFSRs given by the feedback polynomials

$$LFSR_a : f_a(x) = x^{21} + x^2 + 1 \in \mathbb{F}_2[x],$$

$$LFSR_b : f_b(x) = x^{23} + x^5 + 1 \in \mathbb{F}_2[x],$$

$$LFSR_c : f_c(x) = x^{17} + x^3 + 1 \in \mathbb{F}_2[x],$$

and the initial states

$$A_0 = (a_0, \dots, a_{20}),$$

$$B_0 = (b_0, \dots, b_{22}),$$

$$C_0 = (c_0, \dots, c_{16}).$$

The output r_t of the Geffe Generator is defined by the following selection rule.

$$z_t = \begin{cases} a_t, & \text{if } c_t = 0, \\ b_t, & \text{if } c_t = 1. \end{cases} \quad (1.5)$$

We want to determine the period of the sequence $z = (z_i)_{i \geq 0}$ generated by the Geffe Generator. Polynomials can be tested for irreducibility for example

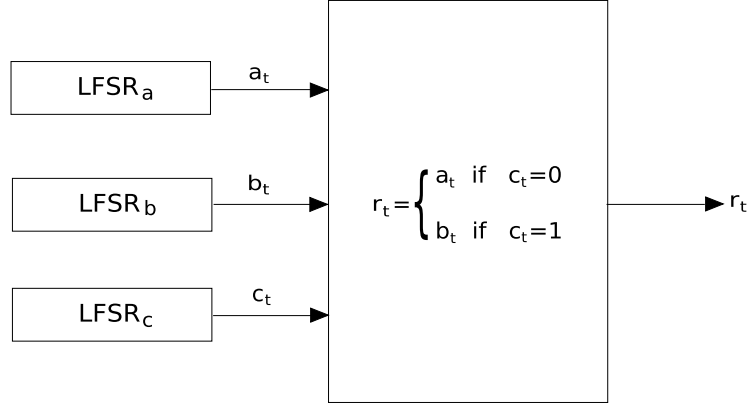


Figure 1.6: The Geffe Generator with three input LFSRs

by the Berlekamp Algorithm [Wan03]. We let Maple do the work for us, and we get that the feedback polynomials of our LFSRs are irreducible.

Fact 1. f_a, f_b and f_c are irreducible.

Hence we can state the following result on the period of the polynomials.

$$P(f_a) | 2^{21} - 1,$$

$$P(f_b) | 2^{23} - 1,$$

$$P(f_c) | 2^{17} - 1.$$

Since $2^{17} - 1$ is a prime, by Remark 2 we have that f_c is primitive. What about f_a and f_b though? Recall the state transition matrix G . Let G_a, G_b respectively, denote the state transition matrix of $LFSR_a, LFSR_b$ respectively. We can determine the period of f_a, f_b respectively, by raising their state transition to the power of the divisors of $2^{21} - 1, 2^{23} - 1$ respectively. This can be done efficiently by consecutive squaring. We get

$$P(f_a) = 2^{21} - 1 = 7^2 \cdot 127 \cdot 337, \quad (1.6)$$

$$P(f_b) = 2^{23} - 1 = 47 \cdot 178481, \quad (1.7)$$

$$P(f_c) = 2^{17} - 1 = 131071. \quad (1.8)$$

Hence all three feedback polynomials are primitive and have maximum period. This allows us to compute the linear complexity of the Geffe Generator. Note that the selection rule in (1.5) can algebraically be described by the formula

$$z_t = F(a_t, b_t, c_t) = (c_t + 1)a_t + c_t b_t = c_t a_t + c_t b_t + a_t \in \mathbb{F}_2.$$

It is known from [RS87] that if LFSR of different length with primitive feedback polynomials are combined in a nonlinear function F , then the resulting sequence

has linear complexity equal to F evaluated in \mathbb{N} at the lengths of the LFSRs, provided that the LFSRs have nonzero initial states. Hence we get the following linear complexity of the Geffe-streams provided nonzero initial states.

$$F(21, 23, 17) = 17 \cdot 21 + 17 \cdot 23 + 21 = 769.$$

Let Ω denote the set of Geffe-streams where the initial states A_0 , B_0 and C_0 are nonzero. These streams have linear complexity 769. For each stream we can determine the according feedback polynomial of degree 769. Note that we do not know yet, whether all sequences in Ω have the same feedback polynomial or not. This is in fact the case though. Note that the periods of the three input LFSRs are coprime (1.8). Thus the Geffe Generator goes through every possible combination of states of the three input LFSRs. Consequently the elements in Ω differ only by a shift in time and all have the same minimal polynomial. Let f_z denote this feedback polynomial of degree 769 that can be found by (1.4).

Claim 1. *All sequences in Ω have the same period equal the period of f_z .*

Proof. Let p be the period of an element $z = (z_i)_{i \geq 0} \in \Omega$ and $p < P(f_z)$. \hat{z} the formal power series of z and $g_z = \text{rev}(f_z)$. We further have $h \in \mathbb{F}_2[x]$ of degree less than 769 such that $\hat{z} = \frac{h}{g_z}$. By following the proof of Theorem 2 we can write

$$\text{rev}(\rho)f_z = \text{rev}(h)(x^p - 1)$$

with h a polynomial of degree less than 769 and $\rho = z_0 + z_1x + \dots + z_{p-1}x^{p-1}$. Hence

$$f_z | \text{rev}(h)(x^p - 1)$$

Since $p < P(f_z)$, f does not divide $(x^p - 1)$ and consequently f_z and $\text{rev}(h)$ have a nontrivial factor in common. Let $q \in \mathbb{F}_2[x]$ be this factor. Then we have

$$\hat{r} = \frac{\frac{h}{\text{rev}(q)}}{\frac{g_z}{\text{rev}(q)}}$$

Since $\deg(\frac{g_z}{\text{rev}(q)}) < \deg(g_z)$ we have a contradiction to the linear complexity. \square

Thus in order to determine the period of the elements in Ω , we need to determine the period of f_z . Clearly we have

$$P(f_z) | \text{lcm}(P(f_a), P(f_b), P(f_c)) = 7^2 \cdot 127 \cdot 337 \cdot 47 \cdot 178481 \cdot 131071$$

So the period of f_z can be determined by raising the state transition matrix G_z of f_z to the power of divisors of $\text{lcm}(P(f_a), P(f_b), P(f_c))$. This can be done efficiently by consecutive squaring and we get the following result.

Fact 2. *All streams in Ω have period equal to $P(f_a)P(f_b)P(f_c) = 7^2 \cdot 127 \cdot 337 \cdot 47 \cdot 178481 \cdot 131071 \approx 2^{61}$.*

We have seen that the Geffe Generator generates keystreams with long period. The feedback polynomials of the LFSRs are trinomials, hence they have only 3 nonzero terms. Due to the low weighted LFSRs and to the easy selection rule (1.5), the Geffe generator is a very efficient keystream generator. We will see however, that one should desist from using the Geffe Generator nowadays. The output of the Geffe Generator leaks information on the LFSR streams which makes it vulnerable to a special class of attacks, the fast correlation attacks.

Chapter 2

Correlation Attack

In this chapter the correlation attack proposed by Siegenthaler [Sie84, Sie85] is presented. The correlation attack can be used to determine the initial states of LFSRs that are combined in a nonlinear combination generator separately. In the first section the general idea of the attack is given. In the second section a basic algorithm for the fast correlation attack is presented. This algorithm is then practically tested on the Geffe Generator. Further a statistical analysis of the capability of the correlation attack is given. Finally a more general algorithm is suggested.

2.1 Idea

Suppose we have n LFSRs $LFSR_1, \dots, LFSR_n$ of length l_1, \dots, l_n that are combined in a nonlinear combination generator. Let $R_0^{(1)}, \dots, R_0^{(n)}$ be the initial states of the LFSRs. Let further $r_t^{(i)}$ be the output of $LFSR_i$ and z_t be the output of the Geffe Generator at time $t \geq 0$ (see Figure 1.4).

Suppose an attacker that has access to the keystream $z = (z_i)_{i \geq 0}$, e.g. through a known plaintext attack. If he wants to recover the initial states of the n LFSRs, he could just test every possible combination of initial states and check which one gives rise to the keystream z . An LFSR of length l has $2^l - 1$ possible non-zero initial states. Hence in order to test every possible combination of initial states he would have to test

$$\prod_{i=1, \dots, n} 2^{l_i} - 1 \approx 2^{\sum_i l_i},$$

different combinations of initial states. There might be a faster way than this brute-force approach, though. Suppose we have the following *correlation probabilities*

$$\Pr(r_t^{(i)} = z_t) = \frac{1}{2} + \epsilon_i \text{ with } \epsilon_i > 0 \text{ for all } i = 1, \dots, n.$$

We assume ϵ_i positive since we want the correlation probability to be bigger than $\frac{1}{2}$. This is by no means a restriction since for negative ϵ_i we can just invert z to get a positive correlation probability. We call ϵ_i the *correlation*.

We will need the following definition.

Definition 7. For $y \in \mathbb{F}_2^n$, the Hamming weight $\text{wt}(y)$ is defined as the number of non-zero coefficients of y . For $x, y \in \mathbb{F}_2^n$, the Hamming distance between x and y is defined as the number of coefficients in which x and y differ. The inverted Hamming distance is the number of coefficients in which x and y agree.

Notation 1. For $x, y \in \mathbb{F}_2^n$, we just write $x \oplus y \oplus 1$ for $x \oplus y \oplus \underbrace{(1, 1, \dots, 1)}_n$.

Remark 3. For $x, y \in \mathbb{F}_2^n$, the inverted Hamming distance is just

$$\text{wt}(x \oplus y \oplus 1).$$

The correlation probabilities can now be used to recover the initial states of the LFSRs separately. Let $z = (z_t)_{1 \leq t \leq N}$ be a length N output stream of the generator. Let $r^{(i)} = (r_t^{(i)})_{1 \leq t \leq N}$ be the according input stream from $LFSR_i$. Clearly then the expected Hamming distance between z and $r^{(i)}$ is

$$\mathbb{E}(\text{wt}(r^{(i)} \oplus z \oplus 1)) = N\left(\frac{1}{2} + \epsilon_i\right) = \frac{N}{2} + \epsilon_i N,$$

whereas for any random sequence $s = (s_t)_{1 \leq t \leq N}$ we have

$$\mathbb{E}(\text{wt}(s \oplus z \oplus 1)) = \frac{N}{2}.$$

If we guess the initial state of $LFSR_i$ we can test the guess for correctness by generating the according $LFSR_i$ -stream s and comparing it to the keystream z . For that purpose we calculate the inverted Hamming distance

$$h_s := \text{wt}(r^{(i)} \oplus s \oplus 1). \quad (2.1)$$

If $s = r^{(i)}$, then h_s is a sample of a $\text{Bi}(N, \frac{1}{2} + \epsilon_i)$ distributed random variable

$$h_s \sim \text{Bi}(N, \frac{1}{2} + \epsilon_i), \quad (2.2)$$

and if $s \neq r^{(i)}$, then h_s is a sample of a $\text{Bi}(N, \frac{1}{2})$ distributed random variable

$$h_s \sim \text{Bi}(N, \frac{1}{2}). \quad (2.3)$$

Ideally the underlying distribution of the sample h_s can be recognized and the correct initial state recovered.

It is clear that for small ϵ_i a large number N might be necessary for a reliable decision. Suppose the conditions for a reliable decision is given, then the initial keys of the different LFSRs can be recovered separately which leads to a complexity of testing

$$\sum_{i=1, \dots, n} 2^{l_i} - 1$$

initial states. This is a significant reduction in complexity compared to the brute-force approach where $\prod_{i=1, \dots, n} 2^{l_i} - 1$ combinations of initial keys have to be tested for correctness. Even if only one LFSR is concerned with the correlation attack, w.L.o.G $LFSR_1$, the complexity of the brute force attack is reduced by a factor 2^{l_1}

$$2^{l_1} + \prod_{i=2, \dots, n} 2^{l_i} - 1 \approx \frac{2^{\sum_{i=1, \dots, n} l_i}}{2^{l_1}}.$$

2.2 A basic correlation attack algorithm

In order to start the correlation attack, we need an algorithm to generate the stream generated by an LFSR. Let

$$f = c_0 + c_1x + c_2x^2 + \cdots + c_{l-1}x^{l-1} + x^l,$$

be the characteristic polynomial of the LFSR. Let further $R_0 = (r_0, \dots, r_{l-1})$ be the initial state of the LFSR. The t -th stream bit r_t , $t \geq l$, can be computed by

$$r_t = c_0r_{t-l} + c_1r_{t-l+1} + c_2r_{t-l+2} + \cdots + c_{l-1}r_{t-1} \text{ for } t \geq l.$$

Let w be the number of feedback taps of the LFSR, i.e. the number of non-zero coefficients in $\{c_0, \dots, c_{l-1}\}$. The number of non-zero coefficients (in the following referred to as the *weight*) of the polynomial f then is $w + 1$. We will need this parameter in the calculation of the complexity of the algorithm.

Algorithm 1. LFSR

INPUT: characteristic polynomial f of degree l , initial state R_0

OUTPUT: Stream $r = (r_t)_{0 \leq t \leq N-1}$ of length N generated by an LFSR with char. pol. f and initial state R_0

- $l \leftarrow \deg(f)$
- $r \leftarrow R_0$
- **for** $t = l, \dots, N - 1$ **do**
 - $r[t] \leftarrow \sum_{0 \leq i \leq l-1} c_i r[t-l+i] \pmod{2}$
- **return** r

Proposition 1. *Let $f \in \mathbb{F}_2[x]$ be a polynomial of degree l and weight $w + 1$. A LFSR stream of length N with characteristic polynomial f can be generated by Algorithm 1 using $(N - l)w$ additions in \mathbb{F}_2 .*

Proof. The for-loop in step 3 is entered $(N - l)$ times. Calculating the sum in this for-loop is the same as adding the w elements in the feedback taps of the corresponding LFSR. \square

We can generate the LFSR stream for an arbitrary initial state $R_0 \in \mathbb{F}_2^l$ and we need to find the correct one. For every $R_0 \in \mathbb{F}_2^l$ we generate the according LFSR stream r of length N . These streams are compared to the keystream z by calculating the inverted Hamming distance as in (2.1). The stream with the highest inverted Hamming distance is automatically the one which is most likely to satisfy (2.2). We will see that it is in fact the one with the highest probability to be correct.

Algorithm 2. Correlation Attack

INPUT: binary keystream $z = (z_i)_{1 \leq i \leq N}$ of length N , LFSR of length l , a correlation $\epsilon > 0$

OUTPUT: A guess on the initial state R_0 of length l of the LFSR and the according inverted Hamming distance to the keystream

- $maxInvDist \leftarrow 0$
- **for** $i = 0, \dots, 2^l - 1$ **do**
 - $R_0 \leftarrow$ binary representation of i
 - produce LFSR stream $s = (s_t)_{0 \leq t \leq N}$ with initial state R_0 using Algorithm 1
 - calculate the inverted Hamming distance $h_s \leftarrow wt(s \oplus z \oplus 1)$
 - **if** $h_s > maxInvDist$ **then**
 - * $maxInvDist \leftarrow h$
 - * $tempClosest \leftarrow R_0$
- **return** $tempClosest, maxInvDist$

Proposition 2. Let l be the length of the target LFSR with correlation $\epsilon > 0$ to the keystream z , $w > 0$ the number of its taps. Algorithm 2 delivers a guess on the initial state of the target LFSR using less than $2^l N(w + 2)$ additions.

Proof. The for-loop is entered 2^l times. We have already seen that producing the stream of length N takes $(N - l)w$ additions in \mathbb{F}_2 . Computing $(s \oplus z)$ takes N additions in \mathbb{F}_2 . Computing the Hamming weight h can be done in at most $(N - 1)$ additions in \mathbb{Z} by just adding all non-zero elements. So we get

$$(2^l)((N - l)w + N + N - 1) \leq 2^l(Nw - lw + 2N) \leq 2^l(Nw + 2N).$$

□

Note that the output of Algorithm 2 is not necessarily correct. We can compute the probability that it is correct however.

Proposition 3. Let R_0 be the initial state of an LFSR sequence. Let $\epsilon > 0$ be the correlation between the LFSR sequence and a keystream z of length N . The probability that the initial state S_0 returned by Algorithm 2 given the corresponding inverted Hamming distance (i.H.d.) h equals

$$\Pr(S_0 = R_0 | Ham. dist. = h) = \frac{\left(\frac{1}{2} + \epsilon\right)^h \left(\frac{1}{2} - \epsilon\right)^{N-h} \left(\frac{1}{2}\right)^l}{\left(\frac{1}{2} + \epsilon\right)^h \left(\frac{1}{2} - \epsilon\right)^{N-h} \left(\frac{1}{2}\right)^l + \left(\frac{1}{2}\right)^N \left(1 - \left(\frac{1}{2}\right)^l\right)}.$$

Proof. By Bayes law we know that

$$\Pr(S_0 = R_0 | i.H.d. = h) = \frac{\Pr(S_0 = R_0 \wedge i.H.d. = h)}{\Pr(i.H.d. = h)},$$

where

$$\Pr(i.H.d. = h) = \Pr(Ham. dist. = h | S_0 = R_0) \Pr(S_0 = R_0) + \Pr(i.H.d. = h | S_0 \neq R_0) \Pr(S_0 \neq R_0),$$

and

$$\Pr(S_0 = R_0 \wedge i.H.d. = h) = \Pr(i.H.d. = h | S_0 = R_0) \Pr(S_0 = R_0).$$

With

$$\Pr(S_0 = R_0) = \left(\frac{1}{2}\right)^l,$$

and

$$\Pr(\text{i.H.d.} = h | S_0 = R_0) = \left(\frac{1}{2} + \epsilon\right)^{N-h} \left(\frac{1}{2} - \epsilon\right)^h,$$

as well as

$$\Pr(\text{i.H.d.} = h | S_0 \neq R_0) = \left(\frac{1}{2}\right)^N,$$

we get the proposition. \square

Remark 4. *For a given output of the algorithm we can determine its probability of correctness. The a-priori probability that the algorithm returns the correct initial state we do not know however. In Section 2.3 we will examine on how the input parameter N must be chosen so that we are likely to get the correct initial state.*

We have seen that the complexity of Algorithm 2 grows exponentially with the length l of the LFSR. This is no surprise, since the bigger l , the more initial states have to be tested. At the same time, l does not remarkably influence the complexity of generating LFSR sequences of large length by Algorithm 1. Rather the number of feedback taps w plays a crucial role here. Hence it would seem natural to choose the characteristic polynomials in the following way:

- Choose primitive characteristic polynomials of low weight and of large degree to have efficient keystream generation and at the same time high complexity in correlation attacks.

Primitive trinomials of high degree clearly satisfy the requirements above. In fact they have gained special interest in nonlinear combination generators. This is not the end of the story however. We will see in Chapter 4 that low weighted characteristic polynomial allow a more advanced type of correlation attacks, the fast correlation attacks. Further on, the main focus in the construction of nonlinear combination generators is not on the choice of the characteristic polynomials, but in the choice of the nonlinear combination function F . We will see that the correlation ϵ plays crucial role in the vulnerability to correlation attacks. The smaller ϵ , the larger the sample length N has to be for a successful attack. In fact we will see that

$$N = O\left(\frac{1}{\epsilon^2}\right).$$

The parameter ϵ is determined by the combination function F . Hence, F should be chosen in a way so that only small correlations occur.

A typical example of a nonlinear combination generator with primitive trinomials as characteristic polynomials is the Geffe Generator presented in Section 1.3.3.

2.2.1 Correlation attack on the Geffe Generator

In this section Algorithm 1 and 2 are applied to the Geffe Generator as presented in Section 1.3.3. The three input LFSRs are denoted by $LFSR_a$, $LFSR_b$ and $LFSR_c$, their output at time step $t \geq 0$ by a_t , b_t and c_t and the according output of the Geffe Generator by z_t .

First, correlations must be detected. Table 2.1 shows the output z_t of the Geffe Generator given the input a_t , b_t and c_t .

a_t	b_t	c_t	z_t
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 2.1: Output of Geffe Generator

We can directly read off the correlation probabilities from this table. Let's start with the correlation between $LFSR_a$ and the keystream z . We have

$$a_t = z_t \Leftrightarrow (a_t, b_t, c_t) = \begin{cases} (0, 0, 0) \\ (0, 0, 1) \\ (0, 1, 0) \\ (1, 0, 0) \\ (1, 1, 0) \\ (1, 1, 1) \end{cases} \quad (2.4)$$

and

$$a_t \neq z_t \Leftrightarrow (a_t, b_t, c_t) = \begin{cases} (0, 1, 1) \\ (1, 0, 1) \end{cases} \quad (2.5)$$

Note that since the characteristic polynomials of the three input LFSRs are primitive, the output of $LFSR_a$, $LFSR_b$ and $LFSR_c$ can be considered balanced, i.e. $\Pr(a_t = 0) = \Pr(b_t = 0) = \Pr(c_t = 0) = \frac{1}{2}$. Thus, since $LFSR_a$, $LFSR_b$ and $LFSR_c$ are independent we have

$$\Pr((a_t, b_t, c_t) = (\alpha, \beta, \gamma)) = \frac{1}{8} \quad \text{for all} \quad (\alpha, \beta, \gamma) \in \mathbb{F}_2^3. \quad (2.6)$$

Hence, with (2.4),(2.5) and (2.6) we get

$$\Pr(a_t = z_t) = \frac{3}{4} = \frac{1}{2} + \frac{1}{4}.$$

Similarly we get

$$\Pr(b_t = z_t) = \frac{1}{2} + \frac{1}{4},$$

and

$$\Pr(c_t = a_t \oplus z_t) = \frac{1}{2} + \frac{1}{4}.$$

Length	Correct (%)	Time (sec)
40	0	7
60	6	10
80	15	12
100	23	15
120	44	18
140	65	20
160	94	23
180	91	26
200	96	28
250	99	35
300	100	41

Table 2.2: Correctness of Output

Having these correlations, we can expect the proposed algorithm to work correctly for sample keystreams of large length N . In fact, the success rate of the algorithm increases with N growing. For different N 's we tested the algorithm 100 times and checked whether the returned key was the correct one. The Geffe Generator hereby was initialized with random keys for $LFSR_a$, $LFSR_b$ and $LFSR_c$. Table 2.2 shows the success rate as well as the time consumption on a standard 2GHz dual-core AMD processor.

2.3 Statistical Analysis of the Correlation Attack

Given the notation as in Section 2.1, we are interested in the following question.

- Under what circumstances is it possible to decide whether h_s is a sample of a $\text{Bi}(N, \frac{1}{2})$ or a sample of a $\text{Bi}(N, \frac{1}{2} - \epsilon)$ distributed random variable?

We will see that for $\epsilon > 0$, the probability of correct decision comes arbitrarily close to 1 if the sample length N can be chosen arbitrarily large.

2.3.1 Maximum likelihood Test

Assume we have a random variable $X = (X_1, X_2, \dots, X_N)$, where X_1, X_2, \dots, X_N are themselves independent identically distributed random variables. Let $D(y) = \Pr[X_i = y]$ be the distribution of these variables and $D^N(\bar{x}) = \Pr^N[X = \bar{x}]$ the distribution of X . Suppose, given a sample $\bar{x} = (x_1, x_2, \dots, x_N)$ in the sample space Ω of X , we have to decide which of the two hypotheses

- $H_0 : D = P_0$, the random variables X_i have distribution P_0 or
- $H_1 : D = P_1$, the random variables X_i have distribution P_1

is true. Let us denote our decision function by $\phi(x) : \Omega \rightarrow \{0, 1\}$ where $\phi(x) = 0$ means that we decide in favor of H_0 and $\phi(x) = 1$ that we decide in favor of H_1 . Two types of error can occur:

- $E_F : \phi(\bar{x}) = 1$, although H_0 ,

- $E_M : \phi(\bar{x}) = 0$, although H_1 .

We want to choose the decision function $\phi(x)$ in a way that the two probabilities of error

- $P_F = P_0^N[\phi(X) = 1]$,
- $P_M = P_1^N[\phi(X) = 0]$,

are minimal. Naturally there is a trade off between these two probabilities. This trade off can be seen easily if we choose $\phi = 1$ constant. Then $P_M = 0$ but of course, P_F is maximal. The classical approach is to give a bound α on either P_M or P_F and then choose ϕ in a way that the other probability of error is minimal in the set of all decision functions with $P_F \leq \alpha$ respectively $P_M \leq \alpha$.

Definition 8. *Given an observation x , the likelihood of H_0 respectively H_1 equals*

$$p_0(x) := \text{Probability of the observation } x \text{ when } H_0 \text{ is true,}$$

respectively

$$p_1(x) := \text{Probability of the observation } x \text{ when } H_1 \text{ is true.}$$

The likelihood-ratio is defined by

$$\Lambda(x) := \frac{p_0(x)}{p_1(x)}.$$

Definition 9. *The overall probability of error*

$$P_e = \pi_0 P_F + \pi_1 P_M$$

where π_0 and π_1 are the prior probabilities of H_0 and H_1 with $\pi_0 + \pi_1 = 1$.

The Newman-Pearson Lemma tells us how we have to choose ϕ in order to get an optimal decision function in the set of all decision functions with $P_M \leq \alpha$.

Proposition 4. *(Newman-Person Lemma) Given the situation described above, the optimal decision function has the form:*

$$\phi(\bar{x}) = \begin{cases} 0, & \text{if } \Lambda(\bar{x}) \geq \kappa, \\ 1, & \text{if } \Lambda(\bar{x}) < \kappa, \end{cases} \quad (2.7)$$

Where $\Lambda(x)$ is the likelihood-ratio of x . The value of the threshold κ is determined by the condition $P_M = \alpha$.

Note, that if we want the probabilities of errors to be equal $P_M = P_F = \alpha$, then we should choose $\kappa = 1$.

It is clear that the performance of a correlation attack depends on the error probabilities of the optimal decision function ϕ . In the correlation attack, the inverted Hamming distance h_s is calculated. It must be decided whether h_s is a sample of a $\text{Bi}(N, \frac{1}{2} + \epsilon)$ or a sample of a $\text{Bi}(N, \frac{1}{2})$ distributed random variable.

Proposition 5. Let z be a binary stream of length N . Let h be its Hamming weight. The likelihood-ratio in the decision function whether the stream is a sample of a random variable Y with distribution $\text{Bi}(N, \frac{1}{2} + \epsilon)$ (Hypothesis H_0) or X with distribution $\text{Bi}(N, \frac{1}{2})$ (Hypothesis H_1) equals:

$$\Lambda(h, N) = \frac{P_0^N(h)}{P_1^N(h)} = \frac{\binom{N}{h} (\frac{1}{2} + \epsilon)^h (\frac{1}{2} - \epsilon)^{N-h}}{\binom{N}{h} (\frac{1}{2})^N} = \frac{(\frac{1}{2} + \epsilon)^h (\frac{1}{2} - \epsilon)^{N-h}}{(\frac{1}{2})^N}.$$

Note that this ratio is growing with h .

We have seen that for a given length N the likelihood-ratio depends only on h . So the condition $\Lambda \geq \kappa$ in the decision function can be replaced by a condition $h \geq H$ with some new threshold $0 \leq H \leq N$ by setting

$$\frac{(\frac{1}{2} + \epsilon)^H (\frac{1}{2} - \epsilon)^{N-H}}{(\frac{1}{2})^N} = \kappa,$$

we get

$$H = \frac{\log \kappa - N \log(\frac{1}{2} - \epsilon)}{\log(\frac{1}{2} + \epsilon) - \log(\frac{1}{2} - \epsilon)}.$$

Proposition 6. Let H_0 and H_1 be the hypotheses as in Proposition 5. Let H be the threshold on the Hamming weight in the decision function ϕ . The probabilities of error then are

$$\begin{aligned} P_F &= P_0^N[\phi(X) = 1] = \sum_{h \geq H} \binom{N}{h} \left(\frac{1}{2}\right)^N \\ P_M &= P_1^N[\phi(X) = 0] = \sum_{h < H} \binom{N}{h} \left(\frac{1}{2} + \epsilon\right)^h \left(\frac{1}{2} - \epsilon\right)^{N-h} \end{aligned} \quad (2.8)$$

As we see, for a given correlation ϵ , the probabilities of error depend on the sample length N and the threshold H . For the correlation attack, the following question is of interest.

Question 1. For given bounds on the error probabilities P_F and P_M , is it possible to choose N and H so that these bounds are reached by the decision function ϕ ?

The answer to this question will be given in the next section.

2.3.2 Chebyshev's Inequality

Consider the following well known result, see for example [GS01].

Proposition 7. (Chebyshefs Inequality) Let X be a random variable with expected value μ and finite variance σ^2 . Then

$$\Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}. \quad (2.9)$$

In words, by taking $k > 1$ the probability that a sample lies inside the radius of k times the standard deviation around the mean is bigger than $1 - \frac{1}{k^2}$.

Proposition 8. Let X and Y be $\text{Bi}(N, \frac{1}{2})$ and $\text{Bi}(N, \frac{1}{2} + \epsilon)$ distributed random variables with means

$$\mu = N\frac{1}{2} \quad \text{and} \quad \mu_\epsilon = N\left(\frac{1}{2} + \epsilon\right)$$

and standard deviations

$$\sigma = \sqrt{\frac{1}{4}N} \quad \text{and} \quad \sigma_\epsilon = \sqrt{N\left(\frac{1}{4} - \epsilon^2\right)}.$$

Let H be the threshold on the Hamming weight h in the decision function ϕ . We get the following bounds on the error probabilities

$$P_F(H) \leq \frac{1}{2} \frac{1}{k^2} \quad \text{and} \quad P_M(H) \leq \frac{1}{2} \frac{1}{k_\epsilon^2},$$

where

$$k = \frac{H - \mu}{\sigma} \quad \text{and} \quad k_\epsilon = \frac{\mu_\epsilon - H}{\sigma_\epsilon}$$

Proof. Note that

$$P_F = \Pr(X \geq H),$$

with $k = \frac{H - \mu}{\sigma}$ we get

$$P_F = \Pr(X \geq k\sigma + \mu) = \Pr(X - \mu \geq k\sigma) \leq \frac{1}{2k^2}$$

with (2.9) and of symmetry reasons we get the result. Similarly for P_M . \square

Proposition 9. Given the conditions as in Proposition 8. The probabilities of error can be made arbitrarily small by increasing N and setting H appropriately.

Proof. For given k and k_ϵ , by the Chebyshefs inequality the probability that a sample of X , respectively a sample of Y lies outside $A := [\mu - k\sigma, \mu + k\sigma]$, respectively $A_\epsilon := [\mu_\epsilon - k_\epsilon\sigma_\epsilon, \mu_\epsilon + k_\epsilon\sigma_\epsilon]$ is smaller than $\frac{1}{k^2}$, respectively $\frac{1}{k_\epsilon^2}$. Hence the probability that a sample of X respectively Y lies in A , respectively in A_ϵ , equals $(1 - \frac{1}{k^2})$, respectively $(1 - \frac{1}{k_\epsilon^2})$. If we can find an H so that

$$\mu + k\sigma < H < \mu_\epsilon - k_\epsilon\sigma_\epsilon,$$

then the probabilities of error satisfy the following bounds

$$P_F < \frac{1}{k^2} \quad \text{and} \quad P_M < \frac{1}{k_\epsilon^2}.$$

We will now show, that for N big enough, this is indeed the case

$$\begin{aligned} \max A < \min A_\epsilon &\Leftrightarrow \mu + k\sigma < \mu_\epsilon - k_\epsilon\sigma_\epsilon \Leftrightarrow k\sigma < N\epsilon - k_\epsilon\sigma_\epsilon \Leftrightarrow \\ \frac{1}{2}k\sqrt{N} < N\epsilon - k_\epsilon\sqrt{N}\sqrt{\frac{1}{4} - \epsilon^2} &\Leftrightarrow \frac{1}{2}k < \sqrt{N}\epsilon - k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2} \Leftrightarrow \\ \frac{\frac{1}{2}k + k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}}{\epsilon} < \sqrt{N} &\Leftrightarrow \frac{\left(\frac{1}{2}k + k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}\right)^2}{\epsilon^2} < N. \end{aligned}$$

Since k and k_ϵ can be chosen and thus the probability of success can hence be made arbitrarily near to 1, we are done. \square

It is obvious now that we can answer Question 1 with "yes". By the proof of Proposition 9 we can even calculate N and H so that given error bounds are reached.

Corollary 7. *Given the situation as above. Let F and M be upper bounds on the error probabilities P_F and P_M . Let*

$$k \leq \sqrt{\frac{1}{2P_F}} \quad \text{and} \quad k_\epsilon \leq \sqrt{\frac{1}{2P_M}}.$$

With

$$N > \frac{\left(\frac{1}{2}k + k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}\right)^2}{\epsilon^2},$$

and

$$\frac{1}{2}N + k\sqrt{\frac{1}{4}N} < H < N\left(\frac{1}{2} + \epsilon\right) - k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2},$$

the decision function ϕ has error probabilities

$$P_F \leq F \quad \text{and} \quad P_M \leq M$$

Corollary 8. *The probability of success of Algorithm 2 with*

$$N > \frac{\left(\frac{1}{2}k + k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}\right)^2}{\epsilon^2},$$

is bigger than

$$\left(1 - \frac{1}{k^2}\right)^{2^l - 2} \left(1 - \frac{1}{k_\epsilon^2}\right)$$

where l is the length of the target LFSR. In particular, for N growing, the probability of success of Algorithm 2 tends to 1.

From the proof of Proposition 9 we have that

$$\frac{\left(\frac{1}{2}k + k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}\right)^2}{\epsilon^2} = \frac{\frac{1}{4}(k^2 + k_\epsilon^2) + kk_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}}{\epsilon^2} - k_\epsilon^2 < N.$$

Since $\sqrt{\frac{1}{4} - \epsilon^2} \in [0, \frac{1}{2}]$ we can say that

$$N = O\left(\frac{1}{\epsilon^2}\right), \tag{2.10}$$

ensures a good success rate, which is confirmed experimentally.

Remark 5. *If $\mu + k\epsilon \leq \mu_\epsilon - k_\epsilon\sigma_\epsilon$ we get the following dependencies between k and k_ϵ*

$$k \leq 2\sqrt{N}\epsilon - 2k_\epsilon\sqrt{\frac{1}{4} - \epsilon^2}, \tag{2.11}$$

$$k_\epsilon \leq \frac{2\sqrt{N}\epsilon - k}{2\sqrt{\frac{1}{4} - \epsilon^2}}. \tag{2.12}$$

2.3.3 Conclusions

We have seen by Corollary 8 that Algorithm 2 returns the correct initial state, given a large enough sample length N . The preceding discussion however allows us to give a more general algorithm for the correlation attack than Algorithm 2. Given a nonlinear combination generator whose output is correlated to an LFSR. Let z be the keystream, ϵ the correlation and M the number of initial states to be tested.

1. For all possible initial states generate the corresponding LFSR stream s^i of length N .
2. Compute the inverted Hamming distance h_{s^i} to the keystream z .
3. Apply an optimal decision function in the form (2.7) to all h_{s^i} 's to decide whether the underlying distribution is $\text{Bi}(N, \frac{1}{2} + \epsilon)$ (Hypothesis H_0) or $\text{Bi}(N, \frac{1}{2})$ (Hypothesis H_1). Let S_a be the set of initial states for which the decision function decided for H_0 and S_d the set for which it decided for H_1 .
4. Discard the set S_d . Increase N and recursively apply this algorithm to the set S_a .

How to parameterize this algorithm? It is clear that the probability that the correct initial state is in the set S_a is

$$\Pr(\text{correct initial state is in } S_a) = 1 - P_M.$$

The expected number of incorrect initial states in the set S_a is

$$MP_F.$$

We have seen in Proposition 9, that P_M and P_F can be made arbitrarily small by increasing N . It is clear that in the case where S_a consist of only the correct initial state, Algorithm 2 would also return the correct result. However this requires a large sample length N . The advantage of the algorithm above is, that improbable initial states can already be filtered with a relatively small sample length, whereas Algorithm 2 tests all initial states with the same large sample length.

Chapter 3

Boolean functions

In a nonlinear combination generator, several LFSRs are combined by a nonlinear function F . The initial states of the LFSRs often correspond to parts of the secret key. Hence, knowing the keystream, the attacker must not be able to recover the initial states. We have seen that if there is a correlation between the keystream and the LFSR streams, the correlation can be exploited in order to recover the initial states of the LFSRs separately. That is why we are interested in nonlinear functions, where no correlation between small sets of the input bits and the output bit occur. Such functions can be used as combining functions.

We will start with giving a tool, called the *Walsh Transform*, to analyze the correlation between linear combinations of the input bits and the output bits of a combining function. We will further give an overview of further cryptographic properties of a nonlinear combination function, such as its *algebraic degree* and its *nonlinearity*. A high algebraic degree is desired, since this usually results in a high linear complexity of the resulting keystream. The nonlinearity is a measure of to what extent the combination function can be approximated by a linear function, and is also calculated by the Walsh Transform. In a cryptographic setting, a high nonlinearity is desired. We will further see that there are trade-offs between these properties. Finally the Walsh Transform is applied as an example to a cipher called "Achterbahn".

3.1 Walsh Transform

Given an integer-valued function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}$, its Walsh-Hadamard Transform is defined as

$$F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{w \cdot x}, \quad w \in \mathbb{Z}_2^n,$$

where $w \cdot x$ denotes the scalar product. For a Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$, we define the Walsh-Hadamard Transform F as the Walsh-Hadamard Transform of the function $f' : \mathbb{Z}_2^n \rightarrow \mathbb{Z}$ with $f'(w) = 1 \in \mathbb{Z}$ if $f(w) = 1 \in \mathbb{Z}_2$ and $f'(w) = 0 \in \mathbb{Z}$ if $f(w) = 0 \in \mathbb{Z}_2$.

Remark 6. For a function $f : \mathbb{Z}_2^n \rightarrow \{0, 1\}$, we have

$$F(w) = |\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 0\}| - |\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 1\}|. \quad (3.1)$$

The Walsh Transform \hat{F} of a Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$ is then defined as the Walsh-Hadamard Transform of the corresponding function $\hat{f} : \mathbb{Z}_2^n \rightarrow \mathbb{Z}$

$$\hat{f}(x) := (-1)^{f(x)},$$

with values in $\{-1, 1\}$. Hence we get

$$\hat{F}(w) = \sum_{x \in \mathbb{Z}_2^n} \hat{f}(x) (-1)^{w \cdot x} = \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x) \oplus w \cdot x}, \quad w \in \mathbb{Z}_2^n.$$

Note that

$$\hat{F}(w) = |\{x \in \mathbb{Z}_2^n : f(x) = w \cdot x\}| - |\{x \in \mathbb{Z}_2^n : f(x) \neq w \cdot x\}|,$$

and with

$$|\{x \in \mathbb{Z}_2^n : f(x) = w \cdot x\}| = (2^n - |\{x \in \mathbb{Z}_2^n : f(x) \neq w \cdot x\}|),$$

we get

$$\hat{F}(w) = 2^n - 2|\{x \in \mathbb{Z}_2^n : f(x) \neq w \cdot x\}|. \quad (3.2)$$

The Walsh Transform of a Boolean function f can easily be obtained from its Walsh-Hadamard Transform [Pom03]:

Lemma 6. *Given a Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$ and its Walsh-Hadamard Transform $F(x)$ and Walsh Transform $\hat{F}(x)$ we have*

$$\hat{F}(w) = -2F(w) + 2^n \cdot \delta(w), \quad (3.3)$$

where $\delta(0) = 1$ and $\delta(w) = 0$ for $w \neq 0$.

Proof. Let $A_{i,j} := \{x \in \mathbb{Z}_2^n : f(x) = i, w \cdot x = j\}$ for $(i, j) \in \mathbb{Z}_2^2$. Then \mathbb{Z}_2^n is the disjoint union of those $A_{i,j}$'s,

$$\mathbb{Z}_2^n = A_{1,0} \cup A_{0,0} \cup A_{1,1} \cup A_{0,1}.$$

For $w \neq 0$, we have: $|\{x \in \mathbb{Z}_2^n : w \cdot x = 0\}| = |\{\mathbb{Z}_2^n : w \cdot x = 1\}|$ and hence

$$|A_{1,0}| + |A_{0,0}| = |A_{1,1}| + |A_{0,1}| = 2^{n-1}.$$

Hence with Equation (3.1) we get

$$F(w) = |A_{1,0}| - |A_{1,1}| = |A_{1,0}| - (2^{n-1} - |A_{0,1}|),$$

which together with Equation (3.2) leads to

$$-2F(w) = 2^n - 2|A_{1,0}| - 2|A_{0,1}| = 2^n - 2|A_{0,1} \cup A_{1,0}| = \hat{F}(w).$$

For $w = 0$, we have

$$|A_{1,1}| + |A_{0,1}| = 0 \text{ and } |A_{1,0}| + |A_{0,0}| = 2^n,$$

and hence

$$F(w) = |A_{1,0}| - |A_{1,1}| = |A_{1,0}| + |A_{0,1}|.$$

Finally

$$-2F(w) = -2|A_{1,0}| - 2|A_{0,1}| = -2|A_{0,1} \cup A_{1,0}| = \hat{F}(w) - 2^n.$$

□

3.2 Computing the Walsh-Hadamard Transform

Given a Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$ its Walsh-Hadamard Transformation $F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{w \cdot x}$ looks very similar to the Discrete Fourier Transformation. In fact, an algorithm very similar to the Fast Fourier Transformation can be used to compute all the Walsh coefficients. Let's start with some preparations.

Define

$$W_0^{(n)} := \{0\} \times \mathbb{Z}_2^{n-1} \subset \mathbb{Z}_2^n \text{ and } W_1^{(n)} := \{1\} \times \mathbb{Z}_2^{n-1} \subset \mathbb{Z}_2^n.$$

Denote with $\dot{\cup}$ the disjoint union of two sets. Then we have:

$$W_0^{(n)} \dot{\cup} W_1^{(n)} = \mathbb{Z}_2^n.$$

Clearly there is a natural 1-1 relation between elements in \mathbb{Z}_2^{n-1} and elements in $W_0^{(n)}$, $W_1^{(n)}$ respectively. So we have bijective maps

$$\begin{aligned} \chi : \mathbb{Z}_2^{n-1} &\xrightarrow{1-1} W_0^{(n)}; x = (x_1, x_2, \dots, x_{n-1}) \leftrightarrow (0, x_1, x_2, \dots, x_{n-1}) \in \mathbb{Z}_2^n, \\ \chi' : \mathbb{Z}_2^{n-1} &\xrightarrow{1-1} W_1^{(n)}; x = (x_1, x_2, \dots, x_{n-1}) \leftrightarrow (1, x_1, x_2, \dots, x_{n-1}) \in \mathbb{Z}_2^n. \end{aligned}$$

We can write

$$\begin{aligned} F(w) &= \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{w \cdot x} = \sum_{x \in W_0^{(n)}} f(x)(-1)^{w \cdot x} + \sum_{x \in W_1^{(n)}} f(x)(-1)^{w \cdot x} \\ &= \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi(x))(-1)^{w \cdot \chi(x)} + \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi'(x))(-1)^{w \cdot \chi'(x)} \quad (3.4) \end{aligned}$$

Note that for $x \in \mathbb{Z}_2^{n-1}$ we have $w \cdot \chi(x) = w \cdot \chi'(x)$ when $w \in W_0^{(n)}$ and $w \cdot \chi(x) + 1 = w \cdot \chi'(x)$ when $w \in W_1^{(n)}$.

$$F(w) = \begin{cases} \sum_{x \in \mathbb{Z}_2^{n-1}} (f(\chi(x)) + f(\chi'(x)))(-1)^{w \cdot \chi(x)} & \text{for } w \in W_0^{(n)}, \\ \sum_{x \in \mathbb{Z}_2^{n-1}} (f(\chi(x)) - f(\chi'(x)))(-1)^{w \cdot \chi(x)} & \text{for } w \in W_1^{(n)}. \end{cases}$$

Define g_f and g'_f as functions from $\mathbb{Z}_2^{n-1} \rightarrow \mathbb{Z}_2$

$$g_f(x) := f(\chi(x)) + f(\chi'(x)) \text{ and } g'_f(x) := f(\chi(x)) - f(\chi'(x)),$$

With $w \cdot \chi(x) = \dot{w} \cdot x$ where \dot{w} is w without first coordinate, we get

$$F(w) = \begin{cases} \sum_{x \in \mathbb{Z}_2^{n-1}} g_f(x)(-1)^{\dot{w} \cdot x} & \text{for } w \in W_0^{(n)}, \\ \sum_{x \in \mathbb{Z}_2^{n-1}} g'_f(x)(-1)^{\dot{w} \cdot x} & \text{for } w \in W_1^{(n)}. \end{cases} \quad (3.5)$$

It seems that specialists are aware of the following algorithm to compute the $F(w)$'s, see e.g. [Yue77], but we could not locate a reference for the actual algorithm.

Algorithm 1. Fast Walsh-Hadamard Transform (FWHT)

INPUT: $F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{w \cdot x}$ and $f(x) \forall x \in \mathbb{Z}_2^n$

OUTPUT: $FWHT(f) = (F(w))_{w \in \mathbb{Z}_2}$

1. if $n = 1$ then return $F(0) = f(0) + f(1)$, $F(1) = f(0) - f(1)$
2. $g \leftarrow \sum_{x \in \mathbb{Z}_2^{n-1}} (f(\chi(x)) + f(\chi'(x)))(-1)^{w \cdot x}$
 $g' \leftarrow \sum_{x \in \mathbb{Z}_2^{n-1}} (f(\chi(x)) - f(\chi'(x)))(-1)^{w \cdot x}$
3. call the algorithm recursively to evaluate g and g' at $0 \leq i < 2^{n-1}$
4. return $g(0_b), g(1_b), \dots, g((2^{n-1} - 1)_b), g'(0_b), g'(1_b), \dots, g'((2^{n-1} - 1)_b)$

where a_b denotes the binary representation of $a \in \mathbb{N}$.

Theorem 5. Algorithm 1 correctly computes the Walsh-Hadamard coefficients of a Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$ in $O(n2^n)$ operations in \mathbb{Z} .

Proof. We prove correctness by induction on n . For $n = 1$, we have $F(w) = f(0)(-1)^{w \cdot 0} + f(1)(-1)^{w \cdot 1}$ and hence the algorithm correctly returns $F(0) = f(0) + f(1)$ and $F(1) = f(0) - f(1)$. If $n > 1$, we have to show that for $0 \leq i < 2^{n-1}$ $g(i_b) = F(i_b)$ and $g'(i_b) = F((2^{n-1} + i)_b)$. So by Equation (3.5)

$$\begin{aligned} g(i_b) &= \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi(x))(-1)^{i_b \cdot x} + \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi'(x))(-1)^{i_b \cdot x} \\ &= \sum_{x \in W_0^{(n)}} f(x)(-1)^{i_b \cdot x} + \sum_{x \in W_1^{(n)}} f(x)(-1)^{i_b \cdot x} = F(i_b), \end{aligned} \quad (3.6)$$

and

$$\begin{aligned} g'(i_b) &= \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi(x))(-1)^{i_b \cdot x} - \sum_{x \in \mathbb{Z}_2^{n-1}} f(\chi'(x))(-1)^{i_b \cdot x} \\ &= \sum_{x \in W_0^{(n)}} f(x)(-1)^{i_b \cdot x} - \sum_{x \in W_1^{(n)}} f(x)(-1)^{i_b \cdot x} = F((2^{n-1} + i)_b) \end{aligned} \quad (3.7)$$

Let $S(n)$ be the number of additions the algorithm uses in \mathbb{Z} to compute the FWHT of a Boolean function on \mathbb{Z}_2^n . Step 1 takes two additions. In the two sums in step 2 we have 2^{n-1} summands each. In the third step we have $2S(n-1)$ additions. Thus, by unfolding the recursion we get $S(1) = 2$ and $S(n) = 2S(n-1) + 2^n = 2(2S(n-2) + 2^{n-1}) + 2^n = \dots = n2^n$. \square

By applying Algorithm 1 the Walsh-Hadamard Transform F of a Boolean function f can be computed by looking at it as an integer-valued function. Equation (3.3) then delivers the Walsh Transform \hat{F} . Note that computing

$$F(w) = \sum_{x \in \mathbb{Z}_2^n} f(x)(-1)^{w \cdot x}, \quad w \in \mathbb{Z}_2^n,$$

takes $2^n - 1$ summations in \mathbb{Z} . Hence computing all Walsh-Hadamard coefficients, $F(w)$ for all $w \in \mathbb{Z}_2^n$, in a straightforward way would lead to a complexity of

$$O(2^{2n})$$

operations in \mathbb{Z} .

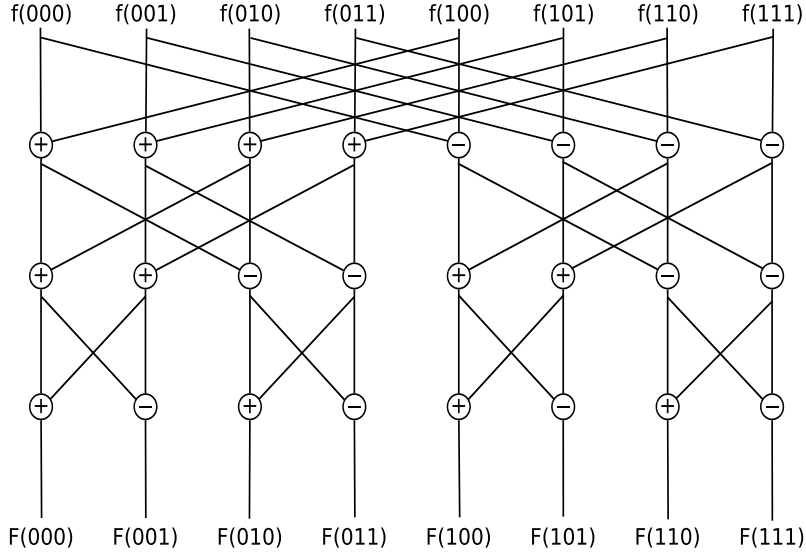


Figure 3.1: The Butterfly structure of the arithmetic circuit computing the FWT for $n=3$.

3.3 Cryptologic properties of Boolean functions

As already said, cryptographic random bit generators are often constructed by combining several maximum-length LFSRs in a nonlinear way. At each step the n output bits of the n LFSRs are combined in a nonlinear Boolean function to get the keystream bit at that step. In order to complicate the predictability of the keystream, this Boolean function should have several properties including high algebraic degree, high nonlinearity and high correlation immunity. In the following, a description of these properties is given.

3.3.1 Algebraic Degree

Every Boolean function $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$ can be expressed in a unique way in its algebraic normal form (ANF), which is the XOR sum of products

$$f_{ANF}(x_1, \dots, x_n) = a_0 \oplus \sum a_i x_i \oplus \sum a_{ij} x_i x_j \oplus \dots \oplus a_{12\dots n} x_1 x_2 \dots x_n,$$

with $a_0, a_i, a_{ij}, \dots \in \mathbb{Z}_2$. The algebraic degree of a function f is its global degree d , i.e. the degree of its highest degree term in the ANF. In [RS87] it was shown that the linear complexity of the resulting keystream when combining n maximum length LFSRs of different lengths l_1, \dots, l_n with a Boolean function f equals $f_{ANF}(l_1, \dots, l_n)$ evaluated over the reals (instead of \mathbb{Z}_2). Clearly if

$$O(l_1) = O(l_2) = \dots = O(l_n) = l,$$

then the linear complexity C of the resulting keystream equals

$$C = O(l^d).$$

Hence the linear complexity of the keystream grows exponentially with the algebraic degree of the combining function. We have seen in Section 1.3.1 that an attacker who has access to a keystream part of length $2C$ can recover the whole keystream. Consequently, the combination function f is desired to have high algebraic degree.

3.3.2 Nonlinearity

Definition 10. A Boolean function f in n variables is called affine function, if it is a function of degree at most one

$$f(x_1, \dots, x_n) = w \cdot x \oplus c = w_1x_1 \oplus \dots \oplus w_nx_n \oplus c$$

with $c \in \mathbb{Z}_2, w \in \mathbb{Z}_2^n$. A linear function is a affine function with zero constant term c .

The nonlinearity N_f of a Boolean function f is its minimum Hamming distance to any affine function g . The Hamming distance between two Boolean functions is hereby defined as the number of bits that differ in their truth tables. By denoting the set of affine functions with n input bits by A_n we get that

$$N_f := \min_{a \in A_n} d_h(f, a) = |\{x : f(x) \neq a(x)\}|.$$

The nonlinearity of a function f is characterized by the flatness of its Walsh Transform \hat{F} [MS90].

Proposition 10.

$$N_f = \frac{1}{2}(2^n - \max_w |\hat{F}(w)|) \quad (3.8)$$

Proof. Remember that by Equation (3.2) we have

$$\hat{F}(w) = 2^n - 2|\{x \in \mathbb{Z}_2^n : f(x) \neq w \cdot x\}| = 2^n - 2d_h(f, w \cdot x),$$

and hence

$$d_h(f, w \cdot x) = \frac{2^n - \hat{F}(w)}{2}.$$

Further we get that

$$d_h(f, w \cdot x \oplus 1) = 2^n - d_h(f, w \cdot x) = \frac{2^n + \hat{F}(w)}{2}.$$

Consider the following cases. If

$$d_h(f, w \cdot x \oplus 1) < d_h(f, w \cdot x),$$

then

$$\frac{2^n + \hat{F}(w)}{2} < \frac{2^n - \hat{F}(w)}{2},$$

and hence $\hat{F}(w)$ negative and

$$d_h(f, w \cdot x \oplus 1) = \frac{2^n - |\hat{F}(w)|}{2}.$$

And if

$$d_h(f, w \cdot x \oplus 1) \geq d_h(f, w \cdot x),$$

then

$$\frac{2^n + \hat{F}(w)}{2} \geq \frac{2^n - \hat{F}(w)}{2},$$

and hence $\hat{F}(w)$ positive or zero and

$$d_h(f, w \cdot x) = \frac{2^n - |\hat{F}(w)|}{2}.$$

It follows that

$$\min(d_h(f, w \cdot x), d_h(f, w \cdot x \oplus 1)) = \frac{2^n - |\hat{F}(w)|}{2}$$

and hence

$$\min_{a \in A_n} d_h(f, a) = \min_w \frac{2^n - |\hat{F}(w)|}{2} = \frac{2^n - \max_w |\hat{F}(w)|}{2}$$

□

The following theorem gives a natural bound on the nonlinearity we can expect to attain.

Proposition 11. (*Paseval's Theorem*)

$$\sum_{w \in \mathbb{Z}_2^n} \hat{F}(w)^2 = 2^{2n} \quad (3.9)$$

Proof. It is easy to show that

$$\begin{aligned} \sum_{w \in \mathbb{Z}_2^n} \hat{F}(w)^2 &= \sum_{w \in \mathbb{Z}_2^n} \hat{F}(w) \hat{F}(w) = \\ &= \sum_{w \in \mathbb{Z}_2^n} \left(\sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x) \oplus w \cdot x} \sum_{y \in \mathbb{Z}_2^n} (-1)^{f(y) \oplus w \cdot y} \right) = \\ &= \sum_{x, y \in \mathbb{Z}_2^n} \left((-1)^{f(x) \oplus f(y)} \sum_{w \in \mathbb{Z}_2^n} (-1)^{(x \oplus y) \cdot w} \right). \end{aligned} \quad (3.10)$$

Note that for $x \oplus y \neq 0$ we have that $(x \oplus y) \cdot w$ is even for one half of the elements in $w \in \mathbb{Z}_2^n$ and odd for the other half. Hence

$$\sum_{w \in \mathbb{Z}_2^n} (-1)^{(x \oplus y) \cdot w} = \begin{cases} 0 & \text{if } x \oplus y \neq 0, \\ 2^n & \text{if } x \oplus y = 0. \end{cases}$$

Consequently we get

$$\sum_x (-1)^{f(x) \oplus f(x)} 2^n = 2^{2n}.$$

□

Corollary 9.

$$\max_w \hat{F}(w) \geq 2^{\frac{n}{2}}$$

Proof. Suppose $\hat{F}(w) < 2^{\frac{n}{2}}$ for all $w \in \mathbb{Z}_2^n$. Then $\sum_{w \in \mathbb{Z}_2^n} \hat{F}(w)^2 < \sum_{w \in \mathbb{Z}_2^n} 2^n = 2^n 2^n = 2^{2n}$ what is a contradiction to Parseval's theorem. \square

Further we have [MS90]

Corollary 10.

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}.$$

Corollary 10 gives a bound on the maximal achievable nonlinearity of a Boolean function f in n variables. A Boolean function f that reaches that bound satisfies the equation in Corollary 9 with equality. By Proposition 11 it is easy to see then that $|\hat{F}(w)| = 2^{\frac{n}{2}}$ for all $w \in \mathbb{Z}_2^n$. Due to their maximal nonlinearity, such functions are of interest.

Definition 11. A Boolean function on an even number n of input bits is called bent function, if and only if

$$|\hat{F}(w)| = 2^{\frac{n}{2}} \text{ for all } w \in \mathbb{Z}_2^n.$$

3.3.3 Correlation Immunity

Definition 12. [Sie84] Let X_1, \dots, X_n be balanced i.i.d. binary random variables. A Boolean function f in n variables is called t -order correlation immune, if the random variable $f(X_1, \dots, X_n)$ is statistically independent of the random vector $(X_{i_1}, \dots, X_{i_t})$ for any choice of indices $1 \leq i_1 < i_2 < \dots < i_t \leq n$.

If at the same time f is balanced (zero and one occur equally often in the output), it is called t -resilient.

It can be shown [XM88] that a discrete random variable Z is independent of m independent binary random variables (X_1, \dots, X_m) if and only if Z is independent of the sum $\lambda_1 X_1 + \dots + \lambda_m X_m$ for every choice $\lambda = (\lambda_1, \dots, \lambda_m) \in \mathbb{Z}_2^m$ not all zero. Let $X = (X_1, \dots, X_m)$ and $|\lambda|$ the Hamming weight of λ .

Corollary 11. A Boolean function f in n variables is t -order correlation immune if and only if $P[f(X) = 1 | \lambda_1 X_1 + \dots + \lambda_n X_n = x] = P[f(X) = 1]$ for all $1 \leq |\lambda| \leq t$, $x \in \mathbb{Z}_2$.

Hence we have f is t -order correlation immune if and only if

$$|\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 1\}| = |\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 0\}| \text{ with } 1 \leq |w| \leq t$$

We get [XM88]:

Proposition 12. A Boolean function f in n variables is t -order correlation immune, $1 \leq t \leq n$, if and only if

$$\hat{F}(w) = 0, \text{ for all } 1 \leq |w| \leq t.$$

Proof. Recall from Remark 3.1 that $F(w) = |\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 0\}| - |\{x \in \mathbb{Z}_2^n : f(x) = 1, w \cdot x = 1\}|$. Also we have by (3.3) that for $w \neq 0$, $\hat{F}(w) = 0 \Leftrightarrow F(w) = 0$. So the Proposition follows with the observations above. \square

$\hat{F}(0) = 0$ means that f is also balanced.

Remark 7. f is t -resilient if $\hat{F}(w) = 0$ for $0 \leq |w| \leq t$.

Corollary 12. Let f be a Boolean function in n variables. Let $w \in \mathbb{Z}_2^n$ and $\hat{F}(w)$ its Walsh Coefficient. Then we have

$$\Pr(f(X) = w \cdot x) = \frac{1}{2} + \frac{\hat{F}(w)}{2^{n+1}}.$$

In particular the Walsh Transform of a Boolean function can be used to compute its correlation probabilities.

Proof. By (3.2) we have $\hat{F}(w) = 2^n - 2|\{x \in \mathbb{Z}_2^n : f(x) \neq w \cdot x\}|$. Hence we get

$$\Pr(f(X) = w \cdot x) = 1 - \Pr(f(X) \neq w \cdot x) = 1 + \frac{\hat{F}(w) - 2^n}{2^{n+1}} = \frac{1}{2} + \frac{\hat{F}(w)}{2^{n+1}}.$$

\square

In order not to allow easy correlation attacks, a combining function f with high resiliency is wanted. We will see however, that there are trade offs between resiliency and nonlinearity as well as resiliency and the algebraic degree.

3.3.4 Tradeoffs

High uncorrelatedness does not allow high order product terms in the ANF [Sie84].

Proposition 13. Let f be a t -resilient Boolean function in n variables with algebraic degree d . The following bounds hold

$$d + t \leq n - 1 \text{ if } 1 \leq t \leq n - 2,$$

$$d + t \leq n \text{ if } t = n - 1.$$

We have seen that for the nonlinearity it would be best if the Walsh Spectrum was flat, i.e. if we have Bent functions. Bent functions are not correlation immune however.

Proposition 14. [KSY97] Given an t -order correlation immune Boolean function f in n variables. Then the following bound on the nonlinearity holds

$$N_f \leq \frac{1}{2} \left(2^n - \frac{2^n}{\sqrt{2^n - \sum_{1 \leq i \leq t} \binom{n}{i}}} \right).$$

Proof. If f is t -order correlation immune, then by Proposition 12 $\hat{F}(w) = 0$ for all $1 \leq |w| \leq t$. This means at least $\sum_{1 \leq i \leq t} \binom{n}{i}$ summands in Equation (3.9) are zero. So

$$\max_w \hat{F}(w) \geq \sqrt{\frac{2^{2n}}{2^n - \sum_{1 \leq i \leq t} \binom{n}{i}}} = \frac{2^n}{\sqrt{2^n - \sum_{1 \leq i \leq t} \binom{n}{i}}}$$

With that, from Equation 3.8 the proposition follows. \square

Example 1. Let f be a Bent function in n variables. Then $|F(w)| = 2^{\frac{n}{2}}$ for all $0 \leq |w| \leq n$. First of all, f is not balanced, since $F(0) \neq 0$. Since $|F(w)| = 2^{\frac{n}{2}}$ for $|w| = 1$ there is the following correlation probability to each input:

$$\Pr[f(X) = X_i] = \frac{\frac{1}{2}(|\{x : f(x) = x_i\}| - |\{x : f(x) \neq x_i\}|)}{2^n} = \frac{\frac{1}{2}|\hat{F}(e_i)|}{2^n} = \frac{1}{2^{\frac{n}{2}} + 1},$$

where e_i is the i -th unit vector. So a classical Siegenthaler attack can be mounted.

3.4 An Example

A distinguishing attack on the cryptosystem Achterbahn, a stream cipher proposed to the eSTREAM competition, was presented in [JMM05]. Basically Achterbahn uses 8 small non-linear registers R_1, \dots, R_8 that are combined in a filtering function f . Let's denote the output bits of the 8 registers at a time step t by $y_1(t), \dots, y_8(t)$. The ANF of the filtering function $f : \mathbb{Z}_2^8 \rightarrow \mathbb{Z}_2$ looks as follows:

$$f(t) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t) \oplus y_5(t)y_7(t) \oplus y_6(t)y_7(t) \oplus y_6(t)y_8(t) \oplus y_5(t)y_6(t)y_7(t) \oplus y_6(t)y_7(t)y_8(t).$$

Computing the Walsh Transform of f , we get a Walsh Spectrum that is mostly flat, but has the following peaks:

$$\hat{F}(w) = 128 \text{ for } w \in A = \{244_b = 11110100, 242_b = 11110010\}, \quad (3.11)$$

and

$$\hat{F}(w) = 64 \text{ for } w \in B = \{244_b = 11111000, 242_b = 11110001\}. \quad (3.12)$$

Furthermore $|\hat{F}(w)| = 64$ for $w \in C = \{243_b, 245_b, 247_b, 250_b, 252_b, 254_b\}$, and $\hat{F}(w) = 0$ for $w \in \mathbb{Z}_2^8 \setminus A \cup B \cup C$.

Every element in $A \cup B \cup C$ gives a linear approximation of f . Note that the elements in A and B have Hamming weight equal to 5, whereas the elements in C have Hamming weight bigger than 5. Hence a linear approximation with elements in A and B can be exploited more easily for an attack. Furthermore since the Walsh coefficient of the elements in A is bigger than the one of the elements in B , the elements in A deliver a better approximation of f than the

elements in B . Let X be a random variable with uniform distribution in \mathbb{Z}_2^8 . Then by Corollary 12 in Section 3.3.3 and (3.11,3.12) we have

$$\Pr[f(X) = w \cdot X] = \frac{10}{16} = \frac{1}{2} + \frac{1}{8} \text{ for } w \in B \cup C,$$

$$\Pr[f(X) = w \cdot X] = \frac{12}{16} = \frac{1}{2} + \frac{1}{4} \text{ for } w \in A.$$

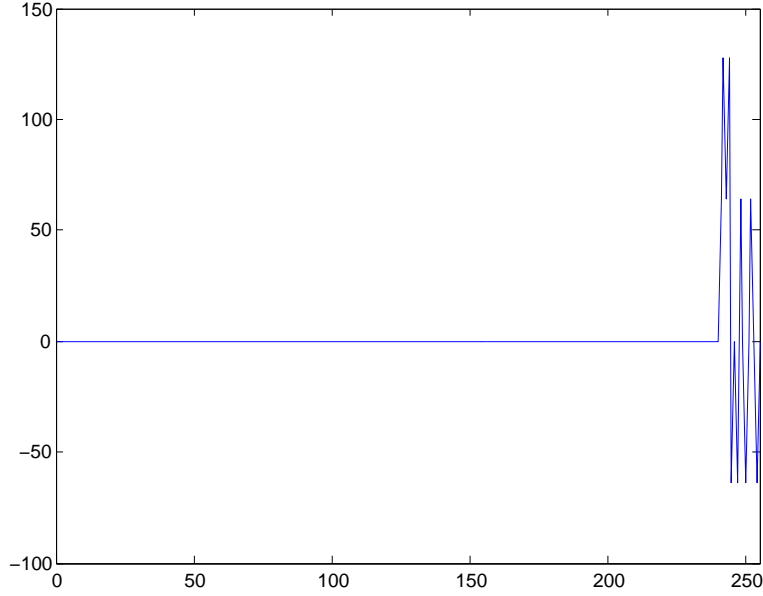


Figure 3.2: Walsh Spectrum of the filtering function f . On the x -axis the decimal representation of the points $x = (x_1, \dots, x_8) \in \mathbb{Z}_2^8$.

3.4.1 Distinguishing attack

The linear approximation

$$l(t) = y_1(t) + y_2(t) + y_3(t) + y_4(t) + y_6(t), \quad (3.13)$$

that corresponds to $w = 11110100 \in A$ is used for a distinguishing attack. Its bias ϵ equals $\frac{1}{4}$. Let us call the periods of the register R_i by T_i , $i = 1, \dots, 8$. Then we have that in

$$l_1(t) := l(t) \oplus l(t + T_1),$$

the output bits $y_1(t)$ and $y_1(t + T_1)$ of R_1 add to zero. Recursively define

$$l_{i+1}(t) := l_i(t) \oplus l_i(t + T_{i+1}) \text{ for } i = 1, \dots, 3.$$

Note that in l_i the output bits $y_j(t)$ and $y_j(t + T_j)$ of R_j add to zero for $1 \leq j \leq i$, $i = 1, \dots, 3$. Define l_5 slightly differently

$$l_5(t) := l_4(t) \oplus l_4(t + T_6). \quad (3.14)$$

We get that l_5 contains no term of y_1, y_2, y_3, y_4, y_6 . Thus clearly the sequence $(l(i))_{i \in \mathbb{N}}$ satisfies the parity check $l_5(t) = 0$ for all $t \in \mathbb{N}$. Writing Equation (3.14) in terms of $l(i)$ is omitted here. However it can be seen that $l_5(t)$ is the XOR of 32 keystream bits of $(l(i))_{i \in \mathbb{N}}$ which have distance

$$T = T_1 + T_2 + T_3 + T_4 + T_6 \approx 2^{28.51}.$$

We need a result that follows directly from the so called piling-up lemma, see for example [Vau06]. Having n independent binary random variables X_1, \dots, X_n with $\Pr(X_i = 0) = \frac{1}{2} + \epsilon_i$ then we have

$$\Pr(X_1 \oplus \dots \oplus X_n = 0) = \frac{1}{2} + 2^{n-1} \prod_{i=1, \dots, n} \epsilon_i. \quad (3.15)$$

Note that this equals the probability that an even number of the X_i 's are 1.

It is clear that the probability that $l_5(t) = 0$ equals the probability that an even number of its 32 summands do not satisfy (3.13). Hence applying this parity check to the output sequence of Achterbahn, it is satisfied with probability

$$\frac{1}{2} + 2^{31} \left(\frac{1}{4}\right)^{32} = \frac{1}{2} + \left(\frac{1}{2}\right)^{33}.$$

Evaluating all the parity checks on a long enough sequence of Achterbahn, it can be distinguished from a truly random sequence.

3.5 Cascading of boolean functions

We have seen that high order correlation immune boolean functions are of interest in a cryptographic setting. At the same time, the functions are supposed to have a high algebraic degree. In this section we investigate how these two properties behave, when two boolean functions are cascaded.

Let $f(Y_0, \dots, Y_n)$ and $g(X_1, \dots, X_m)$ be two balanced boolean functions in $n+1$ respectively m variables. Let f be r -th order correlation immune and g s -th order correlation immune. Define

$$h_{f_g}(X_1, \dots, X_m, Y_1, \dots, Y_n) := f(g(X_1, \dots, X_m), Y_1, \dots, Y_n) \quad (3.16)$$

The following observations on the correlation immunity k of h_{f_g} and the algebraic degree $d(h_{f_g})$ of h_{f_g} can be made immediately:

- A-1 In any case $k \geq r$, since knowing less than r variables in $\{Y_1, \dots, Y_n\}$ does not give any information about f , even if some information about Y_0 is given. Obvious is the case where r variables in $\{Y_1, \dots, Y_n\}$ are known and none in $\{X_1, \dots, X_m\}$. Also obvious is $d(h_{f_g}) \geq d(f)$.
- A-2 If $f = Y_0 \oplus f(0, Y_1, \dots, Y_n)$, then $h_{f_g} = g(X_1, \dots, X_m) \oplus f(0, Y_1, \dots, Y_n)$ is $r+s$ correlation immune [SZZ94] and $d(h_{f_g}) = \max(d(f), d(g))$.
- A-3 If Y_0 appears in the highest degree product term in the ANF of f , then $d(h_{f_g}) = d(g) + d(f) - 1$.

Lemma 7. *Using the notation from above the following upper bounds on the correlation immunity k and algebraic degree $d(h_{f_g})$ of h_{f_g} can be made: $d(h_{f_g}) \leq d(f) + d(g) - 1$ and $k \leq r + s$.*

Proof. The bound on the algebraic degree follows directly from the ANF.

Now since f is correlation immune of order r but not of order $r + 1$, there exist $0 \leq i_1 < \dots < i_{r+1} \leq n$ such that $P(f = 1 | Y_{i_1} = y_{i_1}, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = \frac{1}{2} + \epsilon_1$, $|\epsilon_1| > 0$. If $i_1 \neq 0$, then clearly $k \leq r$, since g is balanced and hence $P(h_{f_g} = 1 | Y_{i_1} = y_{i_1}, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = P(f = 1 | Y_{i_1} = y_{i_1}, \dots, Y_{i_{r+1}} = y_{i_{r+1}})$.

If $i_1 = 0$, let $1 \leq j_1 < \dots < j_{s+1} \leq m$ such that $P(g = 1 | X_{j_1} = x_{j_1}, \dots, X_{j_{s+1}} = x_{j_{s+1}}) = \frac{1}{2} + \epsilon_2$, $|\epsilon_2| > 0$. Note that by fact that $P(Y_{i_1} = 1) = P(Y_{i_1} = 0) = \frac{1}{2}$ and the r -th order correlation immunity of f we receive with with Bayes rule

$$\begin{aligned} \frac{1}{2} &= P(f = 1 | Y_{i_2} = y_{i_2}, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = \\ &= \frac{1}{2} P(f = 1 | Y_{i_1} = 0, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) + \frac{1}{2} P(f = 1 | Y_{i_1} = 1, \dots, Y_{i_{r+1}} = y_{i_{r+1}}), \end{aligned}$$

and hence

$$P(f = 1 | Y_{i_1} = 1, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = 1 - P(f = 1 | Y_{i_1} = 0, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = \frac{1}{2} - \epsilon_1.$$

Again with Bayes the following holds

$$\begin{aligned} &P(h_{f_g} = 1 | X_{j_1} = x_{j_1}, \dots, X_{j_{s+1}} = x_{j_{s+1}}, Y_{i_2} = y_{i_2}, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) = \\ &P(f = 1 | Y_{i_1} = 0, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) P(g = 0 | X_{j_1} = x_{j_1}, \dots, X_{j_{s+1}} = x_{j_{s+1}}) + \\ &P(f = 1 | Y_{i_1} = 1, \dots, Y_{i_{r+1}} = y_{i_{r+1}}) P(g = 1 | X_{j_1} = x_{j_1}, \dots, X_{j_{s+1}} = x_{j_{s+1}}) = \\ &\left(\frac{1}{2} + \epsilon_1\right) \left(\frac{1}{2} + \epsilon_2\right) + \left(\frac{1}{2} - \epsilon_1\right) \left(\frac{1}{2} - \epsilon_2\right) = \frac{1}{2} + 2\epsilon_1\epsilon_2. \end{aligned}$$

So we have that $k < r + s + 1$. \square

We have seen a special case of the function f such that the upper bound on the algebraic degree of h is reached as well as a one where the upper bound on the correlation immunity is reached. Since both are desirable properties in a cryptographic view, a special case of the function f is wanted, such that h has correlation immunity bigger than r and algebraic degree bigger than $\max\{d(f), d(g)\}$.

To make work a bit easier define the sets:

$$Y := \{Y_0, \dots, Y_n\}, \quad Y' := Y \setminus Y_0 = \{Y_1, \dots, Y_n\}, \quad X := \{X_1, \dots, X_m\}.$$

We will write

$$Y_{i_1, \dots, i_k} = \{Y_{i_1}, \dots, Y_{i_k}\},$$

and when assigning values $y_{i_1}, \dots, y_{i_k} \in \mathbb{Z}_2^k$ to the variables Y_{i_1}, \dots, Y_{i_k} we write

$$Y_{i_1, \dots, i_k} = y_{i_1, \dots, i_k}.$$

Lemma 8. *If h_{f_g} is k -resilient with $k \geq r$, then for every choice $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$ with $l \leq \min\{k, n\}$*

$$P(f = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}.$$

Proof. Let $P(f = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2} + \epsilon$, $\epsilon > 0$. Then, since g is balanced we have

$$P(h_{f_g} = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = P(f = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2} + \epsilon.$$

So h_{f_g} has correlation immunity smaller than l . \square

Lemma 9. $k \leq r + s$. $P(f = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}$ for every choice $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$, $l \leq \min\{k, n\}$ if and only if h_{f_g} is k resilient.

Proof. \Rightarrow : Let $Y_{i_1, \dots, i_{l_2}} \subset Y'$, $X_{j_1, \dots, j_{l_1}} \subset X$ and $y_{i_1, \dots, i_{l_2}} \in \mathbb{Z}_2^{l_2}$, $x_{j_1, \dots, j_{l_1}} \in \mathbb{Z}_2^{l_1}$ with $l_1 + l_2 \leq r + s$ and $l_1 \leq m, l_2 \leq n$. We need to show that

$$P(h_{f_g} = 1|X_{j_1, \dots, j_{l_1}} = x_{j_1, \dots, j_{l_1}}, Y_{i_1, \dots, i_{l_2}} = y_{i_1, \dots, i_{l_2}}) = \frac{1}{2}.$$

If $l_1 \leq s$, then

$$\begin{aligned} P(h_{f_g} = 1|X_{j_1, \dots, j_{l_1}} = x_{j_1, \dots, j_{l_1}}, Y_{i_1, \dots, i_{l_2}} = y_{i_1, \dots, i_{l_2}}) &= \\ P(f = 1|Y_{i_1, \dots, i_{l_2}} = y_{i_1, \dots, i_{l_2}}) &= \frac{1}{2}. \end{aligned}$$

If $l_1 > s$, then $l_2 < r$ and hence

$$\begin{aligned} P(h_{f_g} = 1|X_{j_1, \dots, j_{l_1}} = x_{j_1, \dots, j_{l_1}}, Y_{i_1, \dots, i_{l_2}} = y_{i_1, \dots, i_{l_2}}) &= \\ P(f = 1|X_0 = x_0, Y_{i_1, \dots, i_{l_2}} = y_{i_1, \dots, i_{l_2}}) &= \frac{1}{2}, \end{aligned}$$

with $x_0 \in \mathbb{Z}_2$ arbitrary.

\Leftarrow : By Lemma 8. \square

For the next lemma, we write f in the following way

$$f = Y_0 f'(Y_1, \dots, Y_n) \oplus f(0, Y_1, \dots, Y_n). \quad (3.17)$$

So we split the ANF of f in a part that contains Y_0 as a factor and one that does not contain Y_0 .

Lemma 10. Let $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$, $l \in \mathbb{N}$. Then the following are equivalent

1. $P(f = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}$
2. $P(f = 1|Y_0 = y_0, Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}$ or $P(f' = 1|Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}$

with $y_0 \in \{0, 1\}$.

Proof. Given $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$, $l \in \mathbb{N}$. Write

$$P(f = 1|Y_0 = 0, Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = p,$$

and

$$P(f = 1|Y_0 = 1, Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = q.$$

Since

$$P(f = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}p + \frac{1}{2}q = \frac{1}{2}(p + q),$$

we have the condition that

$$P(f = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2} \Leftrightarrow p + q = 1. \quad (3.18)$$

Notice that

$$f(1, Y_1, \dots, Y_n) = f'(Y_1, \dots, Y_n) \oplus f(0, Y_1, \dots, Y_n).$$

Let

$$P(f' = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \rho,$$

then we have

$$p = \rho(\rho - q) + q(1 - \rho) = \rho + q - 2\rho q.$$

We get

$$p + q = 1 \Leftrightarrow \rho + 2q - 2\rho q = 1 \Leftrightarrow \rho(1 - 2p) = 1 - 2p.$$

Using (3.18) we get

$$P(f = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2} \Leftrightarrow \rho(1 - 2p) = 1 - 2p \Leftrightarrow \text{either } \rho = 1 \text{ or } p = \frac{1}{2}.$$

This ends the proof. \square

Theorem 6. h_{f_g} is k -order correlation immune, $k \leq r + s$ if and only if for every choice $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$, $l \leq k$ either

$$P(f = 1 | Y_0 = y_0, Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = \frac{1}{2}, \quad (3.19)$$

or

$$P(f' = 1 | Y_{i_1, \dots, i_l} = y_{i_1, \dots, i_l}) = 1, \quad (3.20)$$

where $y_0 \in \{0, 1\}$ and f' as in (3.17).

Proof. By Lemma (9) and (10). \square

Remark 8. Consider the two cases where either (3.19) or (3.20) in Theorem 6 is satisfied for every choice $Y_{i_1, \dots, i_l} \subset Y'$ and $y_{i_1, \dots, i_l} \in \mathbb{Z}_2^l$.

1. If (3.19) is trivially satisfied, then clearly f is k -order correlation immune. Hence the correlation immunity of the resulting function h_{f_g} is not larger than the one of f . The algebraic degree can be increased if Y_0 appears in a high degree product term of f (compare A-3 at the beginning of the section).
2. If (3.20) is trivially satisfied, then clearly $f' = 1$ and hence f is of the form $f = Y_0 \oplus f(0, Y_1, \dots, Y_n)$. This way, the correlation immunity of h_{f_g} equals $r + s$. Note however that in this case, the algebraic degree of the resulting function h_{f_g} is just equal $\max\{d(f), d(g)\}$ (compare A-2).

We have seen that the cascading of boolean functions could theoretically lead to a simultaneous increase of both, the algebraic degree and the correlation immunity. However it is not clear whether or not functions f that nontrivially satisfy the conditions in Theorem 6 can be found. Consequently cascading of arbitrary boolean functions is not a suitable tool for the construction of high order correlation immune functions of high algebraic degree. However some stream ciphers are based on this construction, see Hitag 2 in Chapter 5.

Chapter 4

Fast correlation attack

We have seen how a correlation attack can be used to retrieve the initial states of LFSRs that are used in a nonlinear combining generator. If there is a correlation between the LFSR streams and the keystream, the initial states of the LFSRs can be determined separately. This reduces the complexity of the attack to $O(2^l)$ where l is the length of the longest LFSR involved. If all LFSRs have large length l however, despite the reduction of complexity in comparison with the brute-force attack, the correlation attack might be infeasible.

In the fast correlation attack, linear dependencies of the keystream bits are used to recover the initial state. We will see that the cryptoanalytic problem of retrieving the initial key of the LFSRs can be seen as a decoding problem of a linear block code. For the reader not familiar with linear block codes, in Appendix A a short introduction is given. It will become obvious that this approach can also be used to attack a nonlinear filter generator and we will see in an example how this can be done.

4.1 Block Code model

Let \mathcal{L} denote the set of all possible LFSR-sequences for a given LFSR of length l . Consider only the first $N \geq l$ bits of each sequence in \mathcal{L} and denote the set of these truncated sequences by \mathcal{L}_N . Clearly then $|\mathcal{L}_N| = 2^l$.

Let

$$G = \begin{pmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & 1 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 1 \\ c_0 & c_1 & c_2 & \cdots & \cdots & c_{N-1} \end{pmatrix},$$

be the state transition matrix. We know that the i -th state of the LFSR equals $G^i k$, where $k = (k_1, \dots, k_l)$ denotes the initial state (zero state) of the LFSR. Hence every bit r_i in an LFSR sequence is the linear combination of the l initial bits, namely $r_i = g_i k^T = k g_i^T$. g_i hereby is the first row of G^{i-1} . Thus \mathcal{L}_N can be seen as a linear $[N, l]$ block code with corresponding generator matrix

$$G_{LFSR} = (g_1 \ g_2 \ \cdots \ g_N).$$

Note that g_1, g_2, \dots, g_l are just the l -dimensional unit vectors, and g_{l+1} the vector consisting of the feedback coefficients of the characteristic polynomial. So G_{LFSR} is in the standard form

$$G_{LFSR} = (I_l \ P)$$

with I_l the $l \times l$ unit matrix and P a $l \times (N - l)$ matrix. The initial states then equal the information bits and the corresponding length N sequences equal the codewords.

Now let $r = (r_t)_{1 \leq t \leq N}$ denote a truncated LFSR-sequence and $z = (z_t)_{1 \leq t \leq N}$ the corresponding output of the keystream generator with correlation probability $p := \Pr(r_t = z_t) = \frac{1}{2} + \epsilon$. We have seen that we can consider r to be a codeword of a linear $[N, l]$ block code. z is the received word after the transmission of r over a BSC with crossover probability $1 - p$. Clearly if we can correctly decode z , we know the initial key of the LFSR.

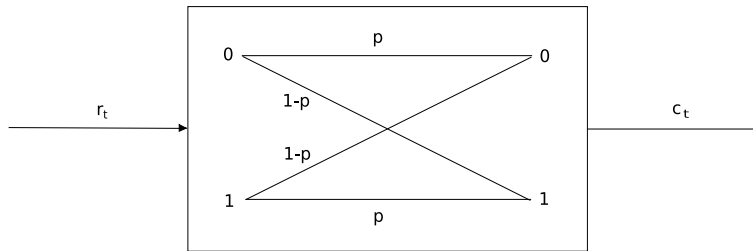


Figure 4.1: BSC with crossover probability $1 - p$

It is clear that we could apply ML-decoding meaning that we just compare all codewords to the received word and look for the closest. ML-decoding technique is optimal in the sense of minimizing error probability, but it is slow. It is the approach corresponding to the standard correlation attack and its complexity is $O(2^L)$. However, there are linear block codes, where suboptimal iterative decoding methods perform very well. A class of such block codes are LDPC codes introduced by Gallager in his PhD thesis [Gal63]. In fact, for large N , the linear block codes in our setting can be seen as LDPC codes. Hence it is no surprise that the algorithms of Meier and Staffelbach [MS88], who in some sense invented the fast correlation attack, resemble the iterative decoding techniques for LDPC codes. A short introduction to LDPC codes is provided in Appendix B.

4.2 The original Fast Correlation attack

In this section we will present the fast correlation attacks proposed by Meier and Staffelbach [MS88]. They used the linear block code model as above and gave the so-called algorithms A and B to decode a received word. Algorithm A can be seen as a one-step decoding algorithm, whose performance for a given situation can be predicted quite well. Algorithm B is an iterative algorithm very similar to the belief propagation algorithm known from LDPC codes. Both are based on the evaluation of the parity checks given by the linear structure of codewords stemming from the LFSR. Let

$$f = a_0 + a_1x + \cdots + a_{l-1}x^{l-1} + a_lx^l$$

be the feedback polynomial of the LFSR of length l . Let w be the number of feedback taps of the LFSR, meaning that the number of non-zero coefficients of f is $w + 1$.

For a LFSR sequence $(r_i)_{i \geq 1}$ we clearly have

$$0 = a_0r_i \oplus a_1r_{i-1} \oplus \cdots \oplus a_l r_{i-l}$$

Thus we get an almost regular (see Appendix B) $(N-l) \times N$ parity check matrix H for the codewords of length N produced by the LFSR.

$$H = \begin{pmatrix} a_l & a_{l-1} & \cdots & a_0 & 0 & \cdots & \cdots & 0 \\ 0 & a_l & a_{l-1} & \cdots & a_0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & a_l & a_{l-1} & \cdots & a_0 \end{pmatrix}$$

For the fast correlation attack a large number of parity checks for every bit is desired. Hence additional parity checks to those given by H are constructed. Note that every multiple of f defines a linear relation for an LFSR sequence r . This means that multiples of f define parity check equations on r . Since we are in the finite field with two elements F_2 , $f^{2^i}(x) = f(x^{2^i})$. Thus, by raising the feedback polynomial f to the power of exponents of 2, we get more parity check equations of weight $w + 1$. The lengths of the parity check obtained in that way then is $2^i l$. An average number J of parity checks for each bit $(r_i)_{1 \leq i \leq N}$ can be computed [MS88]

$$J = \log_2\left(\frac{N}{2l}\right)(w + 1)$$

The parity checks clearly give information on the correctness of a bit. Depending on the number of parity checks a single bit of the received sequence satisfies, its probability of being correct can be computed. In the following we will examine how much the number of parity checks satisfied or not satisfied respectively, tells us about the correctness of a single bit.

Parity check equations of equal weight w on bits with equal probability p of being correct. Suppose we have a parity check equation of weight $w + 1$ on a bit u_1

$$u_1 \oplus u_{i_1} \oplus \cdots \oplus u_{i_w} = 0, \quad 1 < i_1 < \cdots < i_w \quad (4.1)$$

Let $r = (r_i)_{i \geq 1}$ be a sequence satisfying this parity check equation. Let $z = (z_i)_{i \geq 1}$ with $\Pr(z_i = r_i) = \frac{1}{2} + \epsilon$. Let

$$c := z_1 \oplus z_{i_2} \oplus \cdots \oplus z_{i_w}$$

be the parity check equation (4.1) applied to z . We can compute the probability that the parity check provides correct information on z_1 , meaning the probability that it is satisfied if z_1 is correct and not satisfied if z_1 is incorrect.

By (3.15) in Section 3.4.1 we have

$$p_w := \Pr(c = 0 | z_1 = r_1) = \Pr(c \neq 0 | z_1 \neq r_1) = \frac{1}{2} + 2^{w-1} \epsilon^w$$

Note that this is just the probability that an equal number of the bits z_{i_1}, \dots, z_{i_w} is incorrect. Equivalently we have

$$\Pr(c = 0 | z_1 \neq r_1) = \Pr(c \neq 0 | z_1 = r_1) = 1 - p_w = \frac{1}{2} - 2^{w-1} \epsilon^w$$

Let

$$\delta := 2^{w-1} \epsilon^w$$

It can easily be seen that since $|\epsilon| < \frac{1}{2}$, $|\delta|$ decreases with w growing and hence

$$p_w \xrightarrow{w \rightarrow \infty} \frac{1}{2}$$

Suppose we have J parity checks for the first bit and all of them are of weight $w + 1$. Assume further that no two parity checks involve the same bit except for the first one. Then the event of one parity check providing correct information on the first bit is independent of the event of another parity check equation providing correct information on the first bit. Hence if $z_1 = r_1$, the number of satisfied parity checks S is $\text{Bi}(J, \frac{1}{2} + \delta)$ distributed and if $z_1 \neq r_1$ S is $\text{Bi}(J, \frac{1}{2} - \delta)$ distributed. For small δ the expected values of these two distributions is very close. Hence the number of satisfied parity checks for a correct bit is not likely to differ much from the one of an incorrect bit. Generally speaking, the bigger δ , the less parity check equations are needed to decide whether a bit is correct or not.

We can actually calculate the probability of a bit being correct, given the number of satisfied parity checks. We have

$$\Pr(c = 0) = \Pr(z_1 = r_1)p_w + \Pr(z_1 \neq r_1)(1 - p_w) = pp_w + (1 - p)(1 - p_w). \quad (4.2)$$

By Bayes we further know that

$$\Pr(z_1 = r_1 | c = 0) = \frac{\Pr(c = 0 \wedge z_1 = r_1)}{\Pr(c = 0)}. \quad (4.3)$$

Hence, knowing that

$$\Pr(c = 0 \wedge z_1 = r_1) = pp_w,$$

with (4.2) and (4.3) we can calculate

$$\Pr(z_1 = r_1 | c = 0) = \frac{pp_w}{pp_w + (1 - p)(1 - p_w)}. \quad (4.4)$$

Since we assume the parity checks to be independent, the probability of the first bit being correct given that h out of the J parity check equations are satisfied, can easily be computed. Out of the J parity checks let S be the number of satisfied ones. Then

$$p^* = \Pr(z_1 = u_1 | S = h) = \frac{\binom{J}{h} p p_w^h (1 - p_w)^{J-h}}{\binom{J}{h} (p p_w^h (1 - p_w)^{J-h} + (1 - p)(1 - p_w)^h p_w^{J-h})} = \frac{p p_w^h (1 - p_w)^{J-h}}{(p p_w^h (1 - p_w)^{J-h} + (1 - p)(1 - p_w)^h p_w^{J-h})} \quad (4.5)$$

Let's call this probability p^* of a bit being correct after evaluation of the parity checks the *a-posteriori probability*.

Further we can calculate the probability that a bit has at least Λ satisfied parity checks

$$\sum_{\Lambda \leq h \leq J} \binom{J}{h} (p p_w^h (1 - p_w)^{J-h} + (1 - p)(1 - p_w)^h p_w^{J-h}), \quad (4.6)$$

and the probability that it is correct, given it has at least Λ satisfied parity checks equals

$$p_\Lambda = \frac{\sum_{\Lambda \leq h \leq J} \binom{J}{h} p p_w^h (1 - p_w)^{J-h}}{\sum_{\Lambda \leq h \leq J} \binom{J}{h} (p p_w^h (1 - p_w)^{J-h} + (1 - p)(1 - p_w)^h p_w^{J-h})}. \quad (4.7)$$

The situation becomes more complicated if the probabilities of the different bits being correct vary.

Parity check equations of equal weight w on bits with varying probabilities p_i of being correct. Suppose again we have a parity check equation of weight $w + 1$ on a bit u_1

$$u_1 \oplus u_{i_1} \oplus \dots \oplus u_{i_w} = 0, \quad 1 < i_1 < \dots < i_w. \quad (4.8)$$

Let r be as above. Let further $y = (y_i)_{i \geq 1}$ such that $\Pr(y_i = r_i) = \frac{1}{2} + \epsilon_i$. Let

$$c' := y_1 \oplus y_{i_2} \oplus \dots \oplus y_{i_w},$$

be the parity check equation (4.8) applied to y . Again by (3.15) in Section 3.4.1 we have the following probability that the parity check provides correct information on y_1

$$s(p_{j_1}, \dots, p_{j_w}) := \Pr(c' = 0 | y_1 = r_1) = \frac{1}{2} + \prod_{i=j_1, \dots, j_w} \epsilon_i. \quad (4.9)$$

Hence for every parity check equations we can calculate this probability s . Suppose again we have h satisfied out of a total number of J parity checks. Without loss of generality, let c_1, \dots, c_h be the satisfied parity checks and c_{h+1}, \dots, c_J the unsatisfied parity checks. Let s_1, \dots, s_J be the according probabilities. Then

assuming again that the parity checks are independent, the probability that y_1 is correct can be computed.

$$\Pr(z_1 = u_1 | c_1 = \dots = c_h = 0, c_{h+1} = \dots = c_J = 1) = \frac{p_1 \prod_{t=1, \dots, h} s_t \prod_{t=h+1, \dots, J} 1 - s_t}{p_1 \prod_{t=1, \dots, h} s_t \prod_{t=h+1, \dots, J} 1 - s_t + (1 - p_1) \prod_{t=1, \dots, h} 1 - s_t \prod_{t=h+1, \dots, J} s_t}. \quad (4.10)$$

4.2.1 Algorithm A

Given the setting as above, Algorithm A works as follows. First, all parity checks are evaluated. Based on this, for every bit z_1, \dots, z_N the a-posteriori probability p^* is computed. Those bits whose a-posteriori probability p^* exceeds a certain threshold λ are supposed correct. If we have enough correct bits, the initial state can be recovered by solving linear equations.

The main challenge in this algorithm is to make a good choice of the threshold λ . Clearly if λ is chosen too big, we will not have enough bits to recover the initial state. If it is too small, there will be many incorrect bits in the set of bits with a-posteriori probability bigger than λ . Many corrections have then to be made in the set of these bits to recover the initial state, what makes an efficient decoding impossible.

As can be seen from (4.5), in our setting p^* depends only on the number of satisfied parity checks. We can hence determine a threshold Λ on the number h of satisfied parity checks satisfying

$$p^* = \frac{pp_w^h(1-p_w)^{J-h}}{(pp_w^h(1-p_w)^{J-h} + (1-p)(1-p_w)^h p_w^{J-h})} \geq \lambda \iff h \geq \Lambda.$$

Let's denote the set of bits for which at least Λ parity checks are satisfied by Ω_Λ . The expected cardinality $N_{>\Lambda}$ of Ω_Λ can be computed by equation (4.6)

$$N_{>\Lambda} = N \sum_{\Lambda \leq h \leq J} \binom{J}{h} (pp_w^h(1-p_w)^{J-h} + (1-p)(1-p_w)^h p_w^{J-h})$$

And equation (4.7) gives us the probability of a bit in Ω_Λ being correct.

$$p_\Lambda = \frac{\sum_{\Lambda \leq h \leq J} \binom{J}{h} pp_w^h(1-p_w)^{J-h}}{\sum_{\Lambda \leq h \leq J} \binom{J}{h} (pp_w^h(1-p_w)^{J-h} + (1-p)(1-p_w)^h p_w^{J-h})}.$$

We can now state requirements on these parameters for an attack to be successful. Recall that the length of the LFSR is l and we therefore need at least l correct bits to recover the initial state of the LFSR.

- Clearly Λ has to be chosen so that $N_{>\Lambda}$ is bigger than l .
- At the same time, the expected number of incorrect bits in Ω_Λ , calculated by $(1-p_\Lambda)N_{>\Lambda}$, should be small, preferably smaller than 1. The bigger this number, the more modifications of the bits in Ω_Λ have to be tested for correctness.

The choice of a suitable threshold λ is not always possible. In fact, Meier and Staffelbach showed that the complexity of the correlation attack of $O(2^L)$ can be reduced to $O(2^{\beta L})$ where β is dependent on w , p and N/L . For large w ($w \geq 16$), β is estimated to be close to the entropy function $H(p)$, and hence no much improvement in complexity can be expected.

Example 2. Let $p = 0.55$, then $H(p) = -0.55 \log_2 5.5 - 0.45 \log_2 4.5 \approx 0.99$.

4.2.2 Algorithm B

Algorithm B resembles very much the belief propagation algorithm for LDPC codes. Each bit z_i , $i = 1, \dots, N$ has probability p of being correct. By evaluating the parity checks, the a-posteriori probability p^* of being correct can be computed. Hence every bit z_i , $i = 1, \dots, N$ has a new probability of being correct. Using these probabilities as new a-priori probabilities, again new a-posteriori probabilities can be computed. These calculations can iteratively be continued. Let $p_i^{(j)}$ the probability of bit z_i being correct in round j . Clearly $p_i^{(0)} = p$ for all $i = 1, \dots, N$. Then the iterations works as follows.

- Round 0: Evaluate all parity checks. Compute the a-posteriori probability $p_i^{(1)}$ according to (4.5).
- Round $j > 0$: Compute $p_i^{(j+1)}$ according to (4.10).

After a certain number of iterations, those bits with probability below a certain threshold λ are flipped and their probability of correctness is reset to p . After that, the algorithm can be started all over again.

In the following, we will give some analysis on the correction effect to be expected. More precisely we will just estimate the correction effect after round 0. Note that in higher rounds, the probabilities of the different bits are not independent anymore. Then the probabilities computed in the algorithm are not accurate. This is a well known cause of loops in the according Tanner graph. In round 0, the a-posteriori probabilities only depend on the number of satisfied parity checks. Hence we can again replace the probability-threshold λ by a threshold Λ on the number of satisfied parity checks. The expected number of bits for which the number of satisfied parity checks is lower than the threshold can be computed

$$N_{<\Lambda} = N \sum_{0 \leq h \leq \Lambda} \binom{J}{h} (pp_w^h (1-p_w)^{J-h} + (1-p)(1-p_w)^h p_w^{J-h}).$$

If all these bits are flipped, we get an increase I of correct bits, with

$$I = N \sum_{0 \leq h \leq \Lambda} \binom{J}{h} pp_w^h (1-p_w)^{J-h} - N \sum_{0 \leq h \leq \Lambda} \binom{J}{h} (1-p)(1-p_w)^h p_w^{J-h}.$$

Hence the threshold Λ should be chosen as

$$\operatorname{argmax}_{\Lambda} I.$$

Limits of this algorithm can already be seen from the observations for round zero. For p_w close to $\frac{1}{2}$, the expected increase of correct bits is small. By doing

iterations, we might get a bigger increase. But as already mentioned loops in the according Tanner graph slow down the advantages gained by doing iterations. In fact, Meier and Staffelbach showed that in settings where the correlation probability is low ($p \leq 0.75$) and the number of feedback taps large ($t \geq 10$) Algorithms B (as Algorithm A) is infeasible.

In this context it is worth noticing, that one should not use low weighted feedback polynomials for construction of LFSRs in a keystream generator, in order to prevent fast correlation attacks.

4.3 Improvements of the fast correlation attack

4.3.1 Iterative algorithms

We have seen how iterative decoding approaches can be used for fast correlation attacks. The performance of iterative approaches is very hard to predict. The success depends on several parameters as the weight of the parity checks or the girth length of the according Tanner graph. Several improvements of the algorithm by Meier and Staffelbach have been proposed. Most of them are attempts to find parity checks in a way so that a LDPC code with good characteristics is obtained. This means that iterative decoding schemes are likely to succeed. The problem then mostly is, that a huge block length of observed keystream is needed to obtain a helpful amount of parity checks. For an example of an improved iterative fast correlation attack read, e.g. [CT00]

4.3.2 Non-iterative algorithms

In this section we present a one-step decoding approach that has proven to be very powerful in comparison with iterative approaches. It is also based on finding a suitable set of low weighted parity check equations. Since it is non-iterative, the structure of the resulting parity check matrix does not play a very important role. In this approach, and other than in most iterative approaches, not all N bits of the received sequences are decoded. Instead, a certain number B of bits from the initial state of length l are guessed. Suppose the hypothesis (x_1, \dots, x_B) on the first B bits is correct. The remaining $L - B$ bits can then be determined by evaluating the parity checks. Since (x_1, \dots, x_B) is correct and the parity check equations are low-weighted outside these B bits, the probability that they provide correct information about the bit is relatively high. At the same time, by the fact that the parity checks can have arbitrary weight inside the first B bits, we can find quite a lot of them, so that we are likely to correctly determining the remaining $L - B$ bits. Thus we get the initial state under the assumption that the guess on the first B bits is correct. This initial state then is tested for correctness as in the standard correlation attack. If it is not correct, a new guess on the first B bits is made and the whole procedure is repeated.

There are several algorithms following that approach. The algorithm we will present here mostly follows the one from Peizhong Lu and Lianzhen Huang [LH04] and is one of the most sophisticated up to now.

- Pre-processing

1. Construct parity check equations with arbitrary weight in the first B bits, weight zero or one in the following $L - B$ bits and weight w in the remaining $N - L$ bits.

- Processing

1. Set a hypothesis on the first B bits of the initial key (x_1, \dots, x_B) .
2. Evaluate all parity checks involving the guessed bits. If the number of satisfied parity checks is below a certain threshold, discard the hypothesis (x_1, \dots, x_B) and go to processing step 1.
3. Set the remaining $L - B$ bits of the initial state by evaluating the parity checks involving that bit. A candidate initial state (x_1, \dots, x_L) is obtained.
4. Test the candidate: generate the LFSR sequence with candidate (x_1, \dots, x_L) . Compare this sequence to the received one and decide whether it is correct or not as in the standard correlation attack. If it is correct, output (x_1, \dots, x_L) . Else save the hypothesis x_1, \dots, x_B to a set X .

- Post-processing

1. For every saved hypothesis $(x_1, \dots, x_B) \in X$, transform the linear $[N, L]$ code into a linear $[N, L - B]$ code
2. Try to decode the $[N, L - B]$ code. If the decoding succeeds, return (x_1, \dots, x_L)

Note that the post-processing phase equals the processing phase for a $[N, L - B]$ code. Hence its computational complexity can be discarded.

Construction of the Parity check equations

Recall the generator matrix $G_{LFSR} = (g_1 \ g_2 \ \dots \ g_N)$ of the $[N, L]$ code which consists of the length N sequences from the LFSR.

Let $U_0 = (u_1, \dots, u_L)$ be an initial state of the LFSR, $u = (u_1, \dots, u_n)$ the corresponding codeword. Clearly we have

$$u_i = U_0 g_i, i = 1, \dots, n$$

This equation can be seen as parity check equation involving bit u_i and some bits of (u_1, \dots, u_L) depending on g_i . If we choose w different vectors $\{g_{j_1}, \dots, g_{j_w}\}$ out of $\{g_1, \dots, g_N\}$ we get the following parity check equation

$$u_{j_1} \oplus u_{j_2} \oplus \dots \oplus u_{j_w} = U_0 g_{j_1} \oplus U_0 g_{j_2} \oplus \dots \oplus U_0 g_{j_w} = U_0 \underbrace{(g_{j_1} \oplus g_{j_2} \oplus \dots \oplus g_{j_w})}_{=:g}. \quad (4.11)$$

Clearly the parity check equation (4.11) has weight $T = w + wt(g)$.

We want parity check equations with weight w outside the first B bits, and arbitrary weight inside the first B bits. Hence we choose $\{j_1, \dots, j_w\} \subset \{1, \dots, N\}$ so that $g = g_{j_1} \oplus \dots \oplus g_{j_w}$ has arbitrary values in the first B coordinates and value zero in the remaining $L - B$ coordinates. Then g is in the form

$$g = \underbrace{(*, \dots, *)}_B, \underbrace{0, \dots, 0}_{L-B}^T.$$

We get parity check equations of the desired form

$$a_1 u_1 \oplus \cdots \oplus a_B u_B = u_{j_1} \oplus \cdots \oplus u_{j_w} \text{ with } a_i \in \{0, 1\}. \quad (4.12)$$

We will now investigate how many parity checks of that form we obtain. Randomly choosing g_{j_1}, \dots, g_{j_w} we can assume $g = g_{j_1} \oplus \cdots \oplus g_{j_w}$ to be a random vector in F_2^L . Clearly there are 2^B different vectors in F_2^L with zeros in the last $L - B$ bits. Hence having h random vectors in F_2^L the expected number of vectors with zeros in the last $L - B$ bits equals $h \frac{2^B}{|F_2^L|}$ and thus the expected number of parity check equations in the above form is

$$m = \frac{\binom{N}{w}}{2^{L-B}}.$$

We define Φ as the set of parity check equations in the form (4.12). The parity check equations in Φ are then evaluated in the following way to early discard improbable hypotheses (x_1, \dots, x_B) in processing step 2. ses (x_1, \dots, x_B) in processing step 2.

$$a_1 x_1 \oplus \cdots \oplus a_B x_B \oplus u_{j_1} \oplus \cdots \oplus u_{j_w} = 0. \quad (4.13)$$

In processing step 3, we need parity checks to determine the remaining $L - B$ bits of the initial state, assuming the hypothesis is correct. Therefore we construct parity check equations of the form (4.12) but involving exactly one bit u_{B+1}, \dots, u_L

$$a_1 u_1 \oplus \cdots \oplus a_B u_B = u_i \oplus u_{j_1} \oplus \cdots \oplus u_{j_w}. \quad (4.14)$$

By choosing $j_1, \dots, j_w \in L + 1, \dots, N$ so that $g = g_{j_1} \oplus \cdots \oplus g_{j_w}$ has arbitrary values in the first B coordinates and value one at position $B + 1 \leq i \leq L$ and value zero in the remaining $L - B$ coordinates we get parity check equations of the form (4.14). Hence we have equations which we can use to extend the hypothesis to the remaining $L - B$ initial bits

$$a_1 x_1 \oplus \cdots \oplus a_B x_B \oplus u_{j_1} \oplus \cdots \oplus u_{j_w} = x_i \quad (4.15)$$

It is easy to see that for large N

$$|\Phi| \approx |\Phi_i| \approx m.$$

Note that this number does not depend on the weight of the feedback polynomial of the LFSR.

Processing step 2

In Processing step 2 we make a pre-selection on the hypothesis (x_1, \dots, x_B) . This step could, and actually is often, omitted. Therefore only a short explanation of this step will be given here. For further details, please read [LH04].

We have the parity check equations in Φ and Φ_i , $i = B + 1, \dots, L$. Under the assumption that our hypothesis on the first B bits is correct, meaning $(x_1, \dots, x_L) = (u_1, \dots, u_L)$ then the parity check equations in Φ are satisfied with probability

$$p_w = \frac{1}{2} + 2^{w-1} \epsilon^w.$$

It depends on the number w of bits involved.

If the i th bit is correct, then also the probability that a parity check equation from Φ_i is satisfied equals

$$p_w^{(i)} = \frac{1}{2} + 2^{w-1}\epsilon^w.$$

It is easy to see that if the i th bit is not correct, then this probability equals

$$p_w^{(i)} = 1 - p_w^{(i)} = \frac{1}{2} - 2^{w-1}\epsilon^w.$$

Define

$$\delta_w = 2^{w-1}\epsilon^w.$$

Now it can be shown that if our hypothesis on the first B bits is incorrect, meaning $(x_1, \dots, x_L) \neq (u_1, \dots, u_L)$ then the parity check equations in Φ and Φ_i are satisfied with probability $\frac{1}{2}$. Let S be the number of satisfied parity check equations in Φ and S_i the number of satisfied parity check equations in Φ_i . Then, under the assumption that the parity checks are independent, we have

- If $(x_1, \dots, x_B) = (u_1, \dots, u_B)$, then S has distribution $\text{Bi}(m, \frac{1}{2} + \delta_w)$ and S_i has distribution $\text{Bi}(m, \frac{1}{2} + \delta_w)$ or $\text{Bi}(m, \frac{1}{2} - \delta_w)$.
- If $(x_1, \dots, x_B) \neq (u_1, \dots, u_B)$, then S and S_i have distribution $\text{Bi}(m, \frac{1}{2})$.

Note that for big N and small w , the parity check equations are most likely to be independent. The facts above can be used to make a pre-selection on the hypothesis (x_1, \dots, x_B) .

Processing step 3

We have seen that we have $|\Phi_i| \approx m = \frac{\binom{N}{w}}{2^{L-B}}$. Hence, for every bit u_i , $i \in B+1, \dots, L$ we have

$$J = \frac{m}{L-B}$$

parity checks. Again, we have the probability p_w that a parity check equation provides correct information. Let S_i again be the number of passed parity check equations. Let T be the threshold on this number, meaning that if $S_i < T$, the current guess z_i on the i th bit is flipped. Under the independence assumption we get the following probability of correctly determining the i -th bit

$$p' = \sum_{T \leq j \leq J} p p_w^j (1-p_w)^{J-j} + (1-p)(1-p_w)^j p_w^{J-j}.$$

Clearly the bigger p' , in particular $p' > p$, the better the chance of success.

Naturally the question arises on how many parity check equations J are necessary for a successful attack. Again, by defining

$$\delta := 2^{w-1}\epsilon^w,$$

we have a Binomial distribution on the passed parity checks of $\text{Bi}(J, \frac{1}{2} + \delta)$ at correct bits and of $\text{Bi}(J, \frac{1}{2} - \delta)$ at incorrect bits. Bits can be corrected, if these

two distributions can be distinguished by having a sample of J parity checks. It can be shown [FMI07], that the minimum number of parity checks to evaluate each bit at position $B + 1$ to l is well estimated by

$$J_{min} = \frac{1}{4\epsilon^2}.$$

We see here, where the strength of this one-step decoding approach lies. By setting a hypothesis on the first B bits, we can get a useful amount of parity checks for the remaining $L - B$ bits. The number of parity check equations intersecting on the bits $L + 1, \dots, N$ for big N is small compared to J , namely

$$\frac{mw}{N - L}$$

Thus only little information about these bits can be retrieved and hence an iterative approach is not likely to perform well in this setting.

Analysis

The following statement can be made about the complexity of the attack [LH04]. The calculation of the parity check equations is of order $O(N^{w-1} \log N)$. The required space for storage is at most $(L - B + 1)J(B + w \log_2 N)$. The complexity of the processing is

$$O(2^B[(L - B)J + (N - L)\frac{w}{200}]), \quad (4.16)$$

mod 2 operations. We can see that increasing B by one approximately doubles the complexity of the attack. Further it can be shown that the performance of the attack by using parity check equations of weight $w + 1$ is better than the one by only using parity check equations of weight w if and only if

$$2\epsilon\sqrt{\frac{N}{w + 1}} > 1.$$

The algorithm has been practically tested in [LH04] on a LFSR of length 40. It is tested with $w = 2$, correlation probabilities p between 0.6 and 0.7 and B between 18 and 22. Table 4.1 shows some results of their tests. The length N of the keystream is 40000.

corr. ϵ	.17	.16	.15	.14	.13	.12	.11	.10
B=18, J=190	.005	.020	.086	.252	.471	.695	.832	.921
B=19, J=380	.000	.000	.001	.007	.075	.243	.490	.729
B=20, J=761	.005	.000	.000	.000	.000	.007	.076	.292

Table 4.1: Probabilities of incorrect decoding

We can draw the following conclusions from Table 4.1.

- Increasing B gives drastically better results. Note however that by increasing B by one, the complexity of the attack is more than doubled.
- The correlation ϵ plays a very crucial role. Increasing ϵ by approximately 0.02 allows a decrease of B by 1 and hence about halves the complexity of the attack.

Of course, these observations are not accurate. However they give a feeling for the performance of the attack.

Chapter 5

Hitag 2

Hitag 2 is a proprietary algorithm that is part of a security protocol created by Philips. Among other uses it is used for RFID Proximity Access Systems where a card of the size of a credit card can be placed near a sensor to give the owner access to some premises. Hitag 2 has been reverse-engineered¹. Many attacks against Hitag 2 have been discovered since. For example, a standard Time/Memory Tradeoff reduces the key space to 32 bits instead of 48. We will however only focus on the application of the theory studied so far, the correlation attacks.

5.1 The stream generator

Basically Hitag 2 is a nonlinear filter generator consisting of an LFSR of length 48 and a nonlinear boolean function f having 20 taps of the LFSR as input.

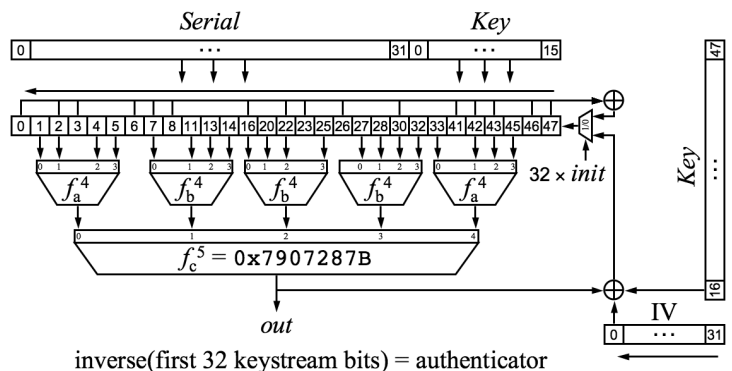


Figure 5.1: Nonlinear filter generator of Hitag 2

¹<http://cryptolib.com/ciphers/hitag2/>

Notation 2. $z_t = \text{output bit at time step } t \geq 0$.
 $r_{t+i} = \text{bit in the } i\text{th tap at time step } t \geq 0, 0 \leq i \leq 47$.

5.1.1 The initialization process

To initialize the generator, the first 32 taps of the LFSR are filled with a Serial Number of length 32 bits. The remaining 16 taps are filled with the first 16 bits of the secret key k . The secret key k has a length of 48 bits in total. The initialization process takes 32 rounds. In each round, the output bit of the generator, the consecutive bits of the initialization vector (IV) and of the rest of the key k are XORed. This bit then is the input for the LFSR. After the 32 initialization rounds, the LFSR works autonomously as usual.

5.1.2 The output function

The combining function f is a composition of Boolean functions

$$f(x_1, x_2, \dots, x_{20}) := f_c^5(f_a^4(x_1, x_2, x_3, x_4), f_b^4(x_5, x_6, x_7, x_8), f_b^4(x_9, x_{10}, x_{11}, x_{12}), f_b^4(x_{13}, x_{14}, x_{15}, x_{16}), f_a^4(x_{17}, x_{18}, x_{19}, x_{20})), \quad (5.1)$$

where

$$f_a^4(x_1, \dots, x_4) := x_1 + x_2 + x_4 + x_1x_3 + x_1x_4 + x_2x_3 + x_1x_2x_3 + 1,$$

$$f_b^4(x_1, \dots, x_4) := x_1 + x_2 + x_4 + x_1x_2 + x_1x_3 + x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + 1,$$

and

$$f_c^5(x_1, \dots, x_5) := x_2 + x_4 + x_1x_2 + x_2x_3 + x_2x_4 + x_2x_5 + x_4x_5 + x_3x_5 + x_1x_3x_4 + x_2x_3x_4 + x_2x_3x_5 + x_1x_4x_5 + x_3x_4x_5 + x_1x_2x_3x_5 + x_1x_2x_4x_5 + 1.$$

Computing the Walsh Transform, we see that none of these functions f_a^4, f_b^4 and f_c^5 is first order correlation immune. By defining $\bar{x} := x + 1 \in \mathbb{Z}_2$ for $x \in \mathbb{Z}_2$ and applying Corollary 12 in Section 3.3.3, we get the following correlation-probabilities.

$$\begin{aligned} \Pr(f_a^4(x_1, x_2, x_3, x_4) = \bar{x}_4) &= \frac{1}{2} + \frac{1}{8}, \\ \Pr(f_b^4(x_1, x_2, x_3, x_4) = \bar{x}_1) &= \frac{1}{2} + \frac{1}{8}, \\ &= \bar{x}_2) = \frac{1}{2} + \frac{1}{8}, \\ &= x_3) = \frac{1}{2} + \frac{1}{8}, \\ \Pr(f_c^5(x_1, x_2, x_3, x_4, x_5) = \bar{x}_2) &= \frac{1}{2} + \frac{1}{16}, \\ &= \bar{x}_3) = \frac{1}{2} + \frac{1}{16}, \\ &= \bar{x}_4) = \frac{1}{2} + \frac{1}{16}. \end{aligned} \quad (5.2)$$

In the stream generator the input of the function f at time $t \geq 0$ is:

$$(r_{t+1}, r_{t+2}, r_{t+4}, r_{t+5}, r_{t+7}, r_{t+11}, r_{t+13}, r_{t+14}, r_{t+16}, r_{t+20}, \\ r_{t+22}, r_{t+25}, r_{t+27}, r_{t+28}, r_{t+30}, r_{t+32}, r_{t+33}, r_{t+42}, r_{t+43}, r_{t+45})$$

By defining

$$\begin{aligned} s_t^{(1)} &:= f_a^4(r_{t+1}, r_{t+2}, r_{t+4}, r_{t+5}), \\ s_t^{(2)} &:= f_b^4(r_{t+7}, r_{t+11}, r_{t+13}, r_{t+14}), \\ s_t^{(3)} &:= f_b^4(r_{t+16}, r_{t+20}, r_{t+22}, r_{t+25}), \\ s_t^{(4)} &:= f_b^4(r_{t+27}, r_{t+28}, r_{t+30}, r_{t+32}), \\ s_t^{(5)} &:= f_a^4(r_{t+33}, r_{t+42}, r_{t+43}, r_{t+45}), \end{aligned}$$

we have that

$$z_t = f_c^5(s_t^{(1)}, s_t^{(2)}, s_t^{(3)}, s_t^{(4)}, s_t^{(5)}). \quad (5.3)$$

From (5.2) and (5.3) we deduce the following correlation in the stream generator.

$$\begin{aligned} \Pr(z_t = r_{t+7}) &= \Pr(z_t = \bar{s}_t^{(2)}) \cdot \Pr(s_t^{(2)} = \bar{r}_{t+7}) + \Pr(z_t \neq \bar{s}_t^{(2)}) \cdot \Pr(s_t^{(2)} \neq \bar{r}_{t+7}) \\ &= \left(\frac{1}{2} + \frac{1}{8}\right)^2 + \left(\frac{1}{2} - \frac{1}{8}\right)^2 = \frac{1}{2} + \frac{1}{32}. \end{aligned}$$

Similarly

$$\begin{aligned} \Pr(z_t = r_{t+11}) &= \frac{1}{2} + \frac{1}{32}, \\ \Pr(z_t = \bar{r}_{t+13}) &= \frac{1}{2} + \frac{1}{32}, \end{aligned}$$

and

$$\begin{aligned} \Pr(z_t = r_{t+16}) &= \frac{1}{2} + \frac{1}{64}, \\ &= r_{t+20}) = \frac{1}{2} + \frac{1}{64}, \\ &= r_{t+27}) = \frac{1}{2} + \frac{1}{64}, \\ &= r_{t+28}) = \frac{1}{2} + \frac{1}{64}, \\ &= \bar{r}_{t+22}) = \frac{1}{2} + \frac{1}{64}, \\ &= \bar{r}_{t+30}) = \frac{1}{2} + \frac{1}{64}. \end{aligned} \quad (5.4)$$

5.1.3 Characteristic polynomial of the LFSR

The characteristic polynomial of the LFSR is

$$\begin{aligned} m(x) &= x^{48} + x^{47} + x^{46} + x^{43} + x^{42} + x^{41} + x^{30} + \\ &\quad x^{26} + x^{23} + x^{22} + x^{16} + x^8 + x^7 + x^6 + x^3 + x^2 + 1. \end{aligned}$$

It can be checked that this polynomial is irreducible in \mathbb{Z}_2 .

5.2 ML decoding

We are interested in the stream $(r_i)_{i \geq 0}$ that -after the initialization process- is generated by the LFSR. Knowing 48 bits of that stream, we can recover the initial state $R_0 := r_0, \dots, r_{47}$ and hence the key k .

We saw that every bit z_t of the keystream is correlated to the nine bits r_{t+i} , $i \in \{7, 11, 13, 16, 20, 22, 27, 28, 30\}$ in the LFSR. Equivalently every bit r_t , $t \geq 30$ in the LFSR is correlated to nine output bits

$$z_{t-i}, i \in \{7, 11, 13, 16, 20, 22, 27, 28, 30\}.$$

It is clear that we can just make a guess \hat{r}_i on each r_i by setting $\hat{r}_i = z_{t-7}$. We then have

$$\Pr(\hat{r}_i = r_i) = \frac{1}{2} + \frac{1}{32}. \quad (5.5)$$

However using all nine bits that give information on a bit r_i simultaneously we can do better. Define

$$a_t := z_{t-7} + z_{t-11} + \bar{z}_{t-13}, \quad b_t := z_{t-16} + z_{t-20} + z_{t-27} + z_{t-28} + \bar{z}_{t-22} + \bar{z}_{t-30}.$$

Let for given $a_t = \hat{a}_t \in \{0, \dots, 3\}$ and $b_t = \hat{b}_t \in \{0, \dots, 6\}$

$$\begin{aligned} \lambda_1(\hat{a}_t, \hat{b}_t) &= \Pr(a_t = \hat{a}_t, b_t = \hat{b}_t | r_t = 1) = \\ &= \binom{3}{\hat{a}_t} \left(\frac{1}{2} + \frac{1}{32}\right)^{\hat{a}_t} \left(\frac{1}{2} - \frac{1}{32}\right)^{3-\hat{a}_t} \binom{6}{\hat{b}_t} \left(\frac{1}{2} + \frac{1}{64}\right)^{\hat{b}_t} \left(\frac{1}{2} - \frac{1}{64}\right)^{6-\hat{b}_t}, \end{aligned} \quad (5.6)$$

and

$$\begin{aligned} \lambda_0(\hat{a}_t, \hat{b}_t) &= \Pr(a_t = \hat{a}_t, b_t = \hat{b}_t | r_t = 0) = \\ &= \binom{3}{\hat{a}_t} \left(\frac{1}{2} + \frac{1}{32}\right)^{3-\hat{a}_t} \left(\frac{1}{2} - \frac{1}{32}\right)^{\hat{a}_t} \binom{6}{\hat{b}_t} \left(\frac{1}{2} + \frac{1}{64}\right)^{6-\hat{b}_t} \left(\frac{1}{2} - \frac{1}{64}\right)^{\hat{b}_t}, \end{aligned} \quad (5.7)$$

be the likelihood coefficients.

Now the likelihood-ratio is

$$\Lambda(\hat{a}_t, \hat{b}_t) = \frac{\lambda_1(\hat{a}_t, \hat{b}_t)}{\lambda_0(\hat{a}_t, \hat{b}_t)}.$$

Maximum likelihood decision is done by setting $\hat{r}_t = 1$ if $\Lambda_t > 1$ and $\hat{r}_t = 0$ if $\Lambda_t < 1$.

By defining $D := \{(a, b) | 0 \leq a \leq 3, 0 \leq b \leq 6\}$, $D_1 := \{(a, b) \in D | \Lambda(a, b) > 1\}$ and $D_0 := \{(a, b) \in D | \Lambda(a, b) < 1\}$ we can give the overall probability of correct decision. It is just the probability that (\hat{a}_t, \hat{b}_t) is in D_1 if $r_t = 1$, the probability that (\hat{a}_t, \hat{b}_t) is in D_0 if $r_t = 0$ respectively

$$\begin{aligned} \Pr(\text{correct decision}) &= \\ &= \sum_{a, b \in D_1} \binom{3}{a} \left(\frac{1}{2} + \frac{1}{32}\right)^a \left(\frac{1}{2} - \frac{1}{32}\right)^{3-a} \binom{6}{b} \left(\frac{1}{2} + \frac{1}{64}\right)^b \left(\frac{1}{2} - \frac{1}{64}\right)^{6-b} = \\ &= \sum_{a, b \in D_0} \binom{3}{a} \left(\frac{1}{2} + \frac{1}{32}\right)^{3-a} \left(\frac{1}{2} - \frac{1}{32}\right)^a \binom{6}{b} \left(\frac{1}{2} + \frac{1}{64}\right)^{6-b} \left(\frac{1}{2} - \frac{1}{64}\right)^b = \\ &= 0.5514. \end{aligned} \quad (5.8)$$

Hence we clearly have a better guess than in 5.5 since $\frac{1}{2} + \frac{1}{32} = 0.53125 < 0.5514$. Note that this is the overall probability, i.e. the probability that for a randomly chosen $t \geq 30$, we decode the bit r_t correctly. For a given $t \geq 30$, the probability that we decode r_t correctly is computed differently. It depends on the parameters a_t and b_t . By Bayes we know that for $x \in \{0, 1\}$

$$\Pr(r_t = x | a_t = \hat{a}_t, b_t = \hat{b}_t) = \frac{\Pr(a_t = \hat{a}_t, b_t = \hat{b}_t | r_t = x) \Pr(r_t = x)}{\Pr(a_t = \hat{a}_t, b_t = \hat{b}_t)}.$$

With $\Pr(r_t = x) = \Pr(r_t = \bar{x}) = \frac{1}{2}$ we have

$$\begin{aligned} \Pr(a_t = \hat{a}_t, b_t = \hat{b}_t) &= \\ & \Pr(a_t = \hat{a}_t, b_t = \hat{b}_t | r_t = x) \Pr(r_t = x) + \\ & \Pr(a_t = \hat{a}_t, b_t = \hat{b}_t | r_t \neq x) \Pr(r_t \neq x) = \\ & \lambda_x(\hat{a}_t, \hat{b}_t) \frac{1}{2} + \lambda_{\bar{x}}(\hat{a}_t, \hat{b}_t) \frac{1}{2}, \end{aligned} \quad (5.9)$$

and hence

$$\Pr(r_t = x | a_t = \hat{a}_t, b_t = \hat{b}_t) = \frac{\lambda_x(\hat{a}_t, \hat{b}_t)}{\lambda_1(\hat{a}_t, \hat{b}_t) + \lambda_0(\hat{a}_t, \hat{b}_t)}. \quad (5.10)$$

Hence if for example we have $a_t = 3$, $b_t = 6$ we set $\hat{r}_t = 1$ and get a probability of correct decoding of $\Pr(\hat{r}_t = r_t) \approx 68\%$.

5.3 Attack

We have seen that we are able to make a guess \hat{r}_t on the value of the bit r_t in the LFSR. The overall error probability however is quite large. Since the bits come from an LFSR they are clearly dependent in the sense that they satisfy some parity checks. We have seen in Chapter 4 how these parity checks can be exploited for a successful attack. Here we will first give an easy distinguishing attack and then give some ideas how Hitag 2 could be attacked using the methods presented in Chapter 4.

5.3.1 A distinguishing attack on Hitag 2

In this section we will shortly describe an easy distinguishing attack on Hitag 2. The aim of a distinguishing attack is not to recover the secret key of the keystream generator, but to distinguish it from a purely random sequence. Recall that for every output bit z_t , $t \geq 0$,

$$\Pr(z_t = r_{t+i}) = \frac{1}{2} + \frac{1}{32} \text{ for } i \in \{7, 11\}.$$

So clearly

$$\begin{aligned} \Pr(z_t \oplus z_{t+4} = 0) &= \Pr(z_t = z_{t+4}) = \\ & \Pr(z_t = r_{t+11}) \Pr(z_{t+4} = r_{t+11}) + \Pr(z_t \neq r_{t+11}) \Pr(z_{t+4} \neq r_{t+11}) = \\ & \left(\frac{1}{2} + \frac{1}{32}\right)^2 + \left(\frac{1}{2} - \frac{1}{32}\right)^2 = \frac{1}{2} + \frac{1}{512}, \end{aligned} \quad (5.11)$$

whereas in a random sequence s_t clearly $\Pr(s_t \oplus s_{t+4} = 0) = \frac{1}{2}$. Thus given a binary sequence $(s_t)_{0 \leq t \leq N}$ of length N , compute the sequence $a = (a_t)_{0 \leq t \leq N-4}$ with $a_t := s_t \oplus s_{t+4}$. Analyze the distribution of zeros and ones in a . If it is $\text{Bi}(N-4, \frac{1}{2})$ -distributed, it is not a Hitag 2 sequence. If it is $\text{Bi}(N-4, \frac{1}{2} + \frac{1}{512})$ -distributed, it is recognized as a Hitag 2 sequence. Having a sample of length $N \geq 512^2 = 2^{18}$ a reliable result can be expected.

5.3.2 Suggestions for a fast correlation attack

The algorithm presented in Section 4.3.2 seems suitable for an attack on Hitag 2. We will refer to that algorithm as the very fast correlation attack (VFCA). Note however that the a-priori correlation probability of about 0.55 is quite small. From the conclusions we draw from Table 4.1 in Section 4.3.2 a higher correlation probability would drastically increase the performance of the attack.

In fact, by making a trade-off with the length of the observed keystream N , a higher correlation can be achieved. We have seen in formula (5.10) how the probability of correct decoding depends on the parameters \hat{a}_t and \hat{b}_t . Clearly the bigger \hat{a}_t and \hat{b}_t the bigger the probability of correct decoding. Table 5.1 shows the cases where the probability of correct decoding is above 0.6 and gives the probability that these cases occur as in 5.9.

	$\hat{a}_t = 3, \hat{b}_t = 6$	$\hat{a}_t = 3, \hat{b}_t = 5$	$\hat{a}_t = 3, \hat{b}_t = 4$	$\hat{a}_t = 2, \hat{b}_t = 6$
$\Lambda(\hat{a}_t, \hat{b}_t)$	0.68	0.65	0.62	0.62
p	0.00205	0.0122	0.02995	0.006

Table 5.1: The probability p that $(a_t, b_t) = (\hat{a}_t, \hat{b}_t)$ the the corresponding probability of correct decoding $\Lambda(\hat{a}_t, \hat{b}_t)$.

The idea now could be, to apply the VFCA only to those bits that have a high probability to be correct, i.e. to look only for parity check equations that involve such bits. Clearly then, a longer observed keystream is necessary. By summing the last row in Table 5.1, we can see that only every 20th bit has probability of being correct bigger than 60%. Consequently the observed keystream must be 20 times longer than in the standard proceeding.

Another approach to improve the VFCA might be to give the parity checks different weights depending on the probabilities of the involved bits. We will not discuss this any further here. However we want to point out, that the correlations in Hitag 2 makes it vulnerable to fast correlation attacks. Therefore it can be said that Hitag 2 should not be used in environments that demand a high security standard.

Appendix A

Linear Block codes

Let \mathbb{F}_2 denote the binary finite field consisting of the elements $\{0,1\}$. Let V_n be \mathbb{F}_2^n as an n dimensional vector space over \mathbb{F}_2 .

Definition 13. A binary linear code C of length n is any linear subspace of V_n . Let k be the dimension of C , then C is called a $[n, k]$ code. The elements c in C are called codewords of C .

Every k -dimensional linear subspace of V_n can be described by k independent basis vectors. Consequently, every code can be described by k independent vectors.

Definition 14. A generator matrix G of an $[n, k]$ code C is a $k \times n$ matrix, where the k rows generate C , meaning that they form a basis of C .

Consequently,

$$C = \{aG | a \in V_k\}$$

Note that $|C| = |V_k| = 2^k$. The encoding of an information vector $a \in V_k$ is simply done by multiplying it with a generator matrix G of C . By encoding $a \in V_k$ we add $n - k$ redundancy bits, which can easily be seen by looking at a generator matrix G_s in standard form

$$G_s = (I_k \ R)$$

where I_k is the $k \times k$ unit matrix and $R \in k \times (n - k)$. The codeword aG_s then simply is the vector c with a in the first k coordinates and the $n - k$ redundancy bits aR .

An alternative way to describe a linear $[n, k]$ code is by means of its parity check matrix. The k dimensional subspace C of V_n can be seen as the solution space of a system of linear equations described by a $(n - k) \times n$ matrix H of rank $n - k$.

Definition 15. A parity check matrix H of an $[n, k]$ code C is an $(n - k) \times n$ matrix satisfying

$$c \in C \Leftrightarrow Hc^T = 0$$

H can be written in the form $H = (-R^T \ I_{n-k})$ where R is the $k \times (n - k)$ redundancy matrix of G in standard form. Then $HG^T = -P^T + P^T = 0$. Clearly H itself is a generator matrix of a $[n, n - k]$ code. This is called the dual code of C and is denoted by C^T . Every pair of codewords of these two codes are orthogonal.

A.0.3 Decoding

When a codeword c is transmitted through a binary symmetric channel, the channel can cause an error pattern e , meaning that the received word r equals $c + e$ for some vector $e \in V_n$. Using the maximum likelihood approach, the codeword closest with respect to the Hamming distance to r is taken. This can be done by just comparing r to all codewords in C . This however has complexity $|C| = 2^k$. The linear structure of the code can for a slightly more sophisticated approach.

The syndrome s of a vector $r \in V_n$ is

$$s = Hr^T = H(c + e)^T = Hc^T + He^T = He^T \in V_{n-k}$$

Thus the syndrome only depends on the error pattern e . Therefore knowing e , decoding is done by calculating $c = r - e$. Note that two vectors $x, y \in V_n$ have the same syndrome if and only if their difference is in C . Hence the set of all vectors with syndrome s is $\{r + c | c \in C\}$, called coset of C . In this set, look for the vector e with lowest weight. e is called the coset leader of the coset. The coset leader is not necessarily unique. If it is not, we can either choose one or give a list of nearest codewords. Clearly

$$H(r - l)^T = Hr^T - Hl^T = 0 \tag{A.1}$$

and hence $c = r - e$ is a codeword. Since e has minimum weight in the set of all vectors that fulfill (A.1), c is the codeword nearest to the received word r .

There are 2^{n-k} different syndromes. Hence if the redundancy is small, meaning $k > n/2$, we can implement a look-up table with all possible syndromes and its respective coset leader. The attack then is faster than the brute force approach of comparing the received word with all 2^k codewords.

Appendix B

LDPC codes

Low density parity check (LDPC) codes are linear codes obtained from sparse bipartite graphs, called Tanner graphs. Bipartite graphs are characterized by the fact that their vertices can be divided into two disjoint sets (e.g. in a 'left' and a 'right' set) and there are no edges in between vertices of the same set. We call the nodes on the left side of the graph message nodes, the nodes on the right side check nodes. At most one edge between any two nodes is allowed.

Having a Tanner graph, its adjacency matrix forms a parity check matrix. Thus a Tanner graph with N message nodes and $N - L$ check nodes can be viewed as a linear $[N, L]$ code, given that the rows of the adjacency matrix are linearly independent. Clearly the sparsity of the graph propagates to the according parity check matrix. Generally it is the sparsity of the parity check matrix that allows efficient algorithms for decoding LDPC codes. We refer to the degree of a message node as the number of directly connected check nodes and the degree of the check nodes as the number of connected message nodes. The degree of the message nodes equals the weight of the columns of the parity check matrix and the degree of the check nodes to the weight of the rows of the parity check matrix. An LDPC code is called regular, if the degree of the message nodes and check nodes is constant.

Belief Propagation

LDPC codes are mainly studied because they can be nearly capacity achieving and there exist polynomial time iterative decoding techniques. The best techniques for LDPC decoding are based on belief propagation, described in [Gal63]. In the following we will shortly explain how the algorithm works. A more detailed analysis is omitted here. The interested reader is referred to e.g. [Gal63] and [Sho04].

The algorithm works in rounds. In each round messages are exchanged between message and check nodes. Let us call the message nodes by m_1, \dots, m_N and the check nodes by c_1, \dots, c_{N-L} . The messages exchanged are probabilities (beliefs). The messages in round $k \geq 1$ are of the following form:

- **Message from message node m_i to check node c_j :** Probability that m_i has a certain value given the observed value of that message node and all the messages sent to m_i in round $k - 1$ from the check nodes other than c_j .

- **Message from check node c_i to message node m_j :** Probability that m_j has a certain value given all the messages sent to c_i in round $k - 1$ from the message nodes other than m_j .

Hard decision decoding

Belief propagation is the best algorithm among message passing decoders [Sho04]. In practice however, it is quite complicated. That is why often discretized versions of the algorithm are used. They clearly do not perform as well as belief propagation, but they are simpler and use less memory. The messages passed are not probabilities anymore, but binary numbers. We will shortly present two hard decision decoding algorithms, known as Gallager A and Gallager B algorithms [Sho04]. Let us start with Gallager A algorithm. In round 0 every message node m_i sends its value $\hat{m}_i^{(0)}$ to all adjacent check nodes. In round $i > 0$, the following messages are sent:

- **Message $\hat{c}_{i,j}^{(i)}$ from check node c_i to message node m_j :** The modulo 2 sum of all incoming messages from the message nodes other than m_j :

$$\hat{c}_{i,j} = \sum_{k \neq j} \hat{m}_{k,i}.$$

- **Message $\hat{m}_{i,j}^{(i)}$ from message node m_i to check node c_j :** If all incoming messages from the check nodes other than c_j have the same value b then $\hat{m}_{i,j} = b$. Else

$$\hat{m}_{i,j}^{(i)} = \begin{cases} b, & \text{if all incom. mes. from the check nodes } c_k, k \neq j, \text{ equal } b, \\ \hat{m}_{i,j}^{(i-1)}, & \text{else.} \end{cases}$$

Gallager B algorithm is slightly more sophisticated. $\hat{m}_{i,j}^{(i)}$ is updated as soon as the number of incoming messages with the same value b exceeds a certain threshold Λ . Λ is determined based on the round and the degree of the message node.

Analysis

The performance of LDPC codes is impossible to describe in general. The success of the iterative decoding methods presented above depend very much on the structure of the graph. It is clear that since the algorithm traverses the edges in the graph, it is of advantage to have a sparse graph. At the same time a good connection is desired in the sense of that message nodes should be connected among each other by preferably short paths. In that way less iterations are needed until the information that one bit carries on the other is transmitted. This leads to another desired property of the graph: the graph should lack small cycles. The girth is the length of the shortest path in the graph. Let G denote the girth length of a graph. Then only $G/4$ iteration steps can be made under the independence assumption, the assumption that the messages passed are statistically independent. The existence of small cycles is therefore undesirable since it can be a drastic constraint to the performance of message passing algorithms.

Bibliography

- [BP82] H. Beker and F. Piper. *Cipher Systems: The Protection of Communications*. Northwood Publications, London, 1982.
- [CBP06] C. de Canniere, A. Biryukov, and B. Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, 2006.
- [CT00] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in cryptology—EUROCRYPT '00*, volume 1807 of *Lecture Notes in Comput. Sci.*, pages 573–588. Springer, Berlin, 2000.
- [FMI07] M. Fossorier, M. Mihaljevic, and H. Imai. Modeling block decoding approaches for the fast correlation attack. *IEEE Trans. Inform. Theory*, 54(12):4728–4737, 2007.
- [Gal63] R.G. Gallager. *Low-Density Parity Check Codes*. M.I.T. Press, Cambridge, MA, 1963. Number 21 in Research monograph series.
- [GG03] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Second Edition. Cambridge University Press, New York, 2003.
- [GS01] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 3rd edition, 2001.
- [JMM05] T. Johansson, W. Meier, and F. Muller. Cryptanalysis of achterbahn, 2005. Available at http://www.staff.uni-mainz.de/pommeren/Kryptologie02/Bitblock/A_Nonlin/nonlin.pdf.
- [Knu98] D.E. Knuth. *The art of computer programming. Vol. 1: Fundamental algorithms*. Third edition. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1998.
- [KSY97] K. Kurosawa, T. Satoh, and K. Yamamoto. Highly nonlinear t-resilient functions. *Journal of Universal Computer Science*, 3:721–729, 1997.
- [LH04] P. Lu and L. Huang. A new correlation attack on lfsr sequences with high error tolerance. In *Coding, Cryptography and Combinatorics*, volume 23 of *Progress in Computer Science and Applied Logic*, pages 67–83. Birkhauser, Basel, 2004.

- [Mol03] R.A. Mollin. *RSA and Public-Key Cryptography*. Chapman & Hall/CRC, Boca Raton, London, New York, Washington, D. C., 2003.
- [MS88] W. Meier and O. Staffelbach. Fast correlation attack on stream ciphers. In *Advances in cryptology—EUROCRYPT '88*, volume 330 of *Lecture Notes in Comput. Sci.*, pages 301–316. Springer, Berlin, 1988.
- [MS90] W. Meier and O. Staffelbach. Nonlinearity criteria for cryptographic functions. In *Advances in cryptology—EUROCRYPT '89*, volume 434 of *Lecture Notes in Comput. Sci.*, pages 549–562. Springer, Berlin, 1990.
- [Pom03] K. Pommerening. Linearitätsmasse für boolesche abbildungen, 2003. Available at <http://www.ecrypt.eu.org/stream/papers.html>.
- [RS87] R.A. Rueppel and O.J. Staffelbach. Products of linear recurring sequences with maximum complexity. *IEEE Trans. Inform. Theory*, 33(1):124–130, 1987.
- [Sho04] A. Shokrollahi. LDPC codes: An introduction. In *Coding, Cryptography and Combinatorics*, volume 23 of *Progress in Computer Science and Applied Logic*, pages 85–110. Birkhauser, Basel, 2004.
- [Sie84] T. Siegenthaler. Correlation-immunity of nonlinear combining function for cryptographic applications. *IEEE Trans. Inform. Theory*, 30(5):776–780, 1984.
- [Sie85] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. on Computers*, 34(1):81–86, 1985.
- [SZZ94] J. Seberry, X. Zhang, and Y. Zheng. On constructions and nonlinearity of correlation immune functions. In *Advances in cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Comput. Sci.*, pages 181–199. Springer, Berlin, 1994.
- [Vau06] S. Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer Verlag, 2006.
- [Wan03] Z.X. Wan. *Lecture on Finite Fields and Galois Rings*. World Scientific Publishing, 2003.
- [XM88] G.Z. Xiao and J.L. Massey. A spectral characterization of correlation - immune combining functions. *IEEE Trans. Inform. Theory*, 34(3):569–571, 1988.
- [Yue77] C.K. Yuen. Testing random number generators by walsh transform. *IEEE Trans. on Computers*, 34(4):329–333, 1977.