1 Introduction. Historical Ciphers

First we consider basic terminology and concepts.

1. By A we denote an alphabet. Usually the binary alphabet is used. Another common alphabet in this course is English. Any alphabet can be encoded in terms of binary alphabet. E.g. the English alphabet consists of 26 characters, all of them can be represented by binary strings of length 5, because there are $32 = 2^5$ such strings. One can use the left $6 = 32 - 26$ strings for encoding punctuation marks like comma, full stop,... If one wants to encode more symbols like capital characters along with small ones and more punctuation marks like braces, brackets, dollar sign, asterisk and digits 0,1,...,9, then 5 bits is not enough. One can use 7-bit or 8-bit ASCII codes. The acronym ASCII stands for American Standard Code for Information Interchange.

2. Let M denote the message space. M consists of strings of symbols from A. If A is the binary alphabet, then M is considered as the set of all possible binary strings. Though each message is of finite length, the number of messages may be considered as infinite.

3. C denotes a set called the cipher-text space. Generally, C is a set of symbol strings over an alphabet different from the alphabet of M.

4. Let K denote a set called the key space. The elements of K are called keys.

5. Each element $e \in K$ uniquely determines a bijection(one-to-one and onto mapping) from M to C denoted by E_e. This bijection is called the encryption function. Equally, each element $d \in K$ uniquely determines a bijection from C to M called a decryption function.

Encryption of $m \in M$ is the process of applying E_e to m. Decryption of $c \in C$ is the process of applying D_d to c.

A cipher(an encryption) scheme consists of a set $\{E_e : e \in K\}$ of encryption transformations and a corresponding set $\{D_d : d \in K\}$ of decryption transformations with the property: for each $e \in K$ there exits $d \in K$ such that $D_d = E_e^{-1}$ or $D_d(E_e(m)) = m$.

Let now there are two parties Alice and Bob seeking for a secret communication. They chose an encryption scheme. Then

1. they secretly choose or secretly exchange a key pair (e, d),

2. when afterwards Alice sends a message $m \in M$ to Bob, she computes $c = E_e(m)$ and transmits c to Bob,

3. upon receiving c, Bob computes $D_d(c) = m$.

Quite obvious observation is that Bob need not to know e to decrypt c. So when $e = d = k$, the cipher is called symmetric, and asymmetric if the knowledge of e doesn't easily lead to that of d. In the last case it is not necessary for Alice to have d.

Let us consider a simplified example of the stream cipher. The parties share the same key k. The binary sequence $x_i = x_i(k)$, called key-stream, generated by the parties is the same and this fact makes the encryption-decryption work. The cipher-text is computed by the rule: $c_i = m_i \oplus x_i$.

The natural question is why keys are necessary. In a good cipher changing the key makes the encryption-decryption transformations work different. So the key should be changed from time to time because the cipher may be vulnerable if too many messages are encrypted on the same key. Let N messages $m_1, m_2, \ldots, m_N$ be encrypted with the key e. So

$$E_e(m_1) = c_1,\ E_e(m_2) = c_2, \ldots, E_e(m_N) = c_N$$

and assume the adversary knows plain-text, cipher-text pairs m_i, c_i. This is a system of equations in one unknown e. The bigger N the more possibilities are to find e, as the number of equations grows. So the number of messages N should be bounded.

When Alice and Bob want to communicate secretly, they should only keep in secret their secret keys. One can gain additional security by keeping secret all encryption-decryption transformations, that is the cipher they are using. But they should not base the secrecy of the transmitted information on this only approach. This was indicated by Kerckhoffs in 1883 along with some other principles. Maintaining the secrecy of the cipher is expensive. But for some cases, especially for governmental purposes, it is probably worth doing. Humans operating the key may be corrupted and if they are able to copy a particular key, they may, at least in theory, sell it to an adversary. It might be more difficult to obtain the cipher as a physical device if it is not the case of a war. During the war the device may be captured in a combat.

An encryption scheme can be broken by trying all possible keys. This is called an exhaustive search on the key space or the brute force attack. The symmetric ciphers are designed such that the brute force attack would the best possible approach for breaking the cipher. But this is not generally right for asymmetric ciphers.

We'll consider symmetric ciphers now. There are two classes of symmetric encryption schemes: block ciphers and stream ciphers. For a block cipher the plain-text to be transmitted is represented by blocks of a fixed length t over the alphabet A. Basicaly, the blocks are encrypted one block at a time. So that for the plain-text

$$m = m_1, m_2, \ldots$$

the cipher-text is

$$c = c_1, c_2, \ldots = E_e(m_1), E_e(m_2), \ldots.$$

Next time the same plain-text block m_i will be encrypted by the same cipher-text block. In stream ciphers the encryption of a block does depend on its place in the stream of the plain-text:

$$c = c_1, c_2, \ldots = E_{e,1}(m_1), E_{e,2}(m_2), \ldots .$$

In some operation modes a block cipher may work as a stream cipher as well.

In this course we look into some basic crypt-analytic approaches especially in stream ciphers which are less complicated in comparison with modern block ciphers. Why to study the cryptanalysis? Any cipher is a response to the information security problem and it does not provide the security if it is not strong enough. To decide requires special attention of skilled professionals. It is not possible to prove that or this cipher is secure like a theorem in mathematics. So cryptanalysts try to break the cipher by using their brain, knowledge of crypt-analytic attacks and experience. In case they fail for quite a while, the cipher may be considered secure. For important ciphers this work should continue on a permanent basis as some new cryptanalytic methods appear and the computational power is constantly increasing. Moreover the cipher may be tested by implementation in software and hardware to figure out whether it is practical in usage.

The prerequisites for this course include some basic probability theory, polynomials over binary field, linear algebra and common sense. We will try to give proofs to almost all mathematical statements we are using. Non of them is difficult. Studying proofs will create intuition and significantly facilitate the applications.

1.1 Substitution Cipher

The example of the block cipher is the substitution cipher, where A is the English alphabet and $t = 1$. The encryption is defined by a substitution on A. For example, let substitution be

A	B	C	D	E	F	G	H	I	J	K	L	M
x	n	y	a	h	p	o	g	z	q	w	b	t

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
s	f	l	r	c	v	m	u	e	k	j	d	$i.$

The text "THE DATA ENCRYPTION STANDARD" taken without gaps is encrypted to "mghaxmxhsycdlmzfsvmxsaxca". The cipher looks secure if one uses a random substitution e for the key. There are $26! \approx 4 \times 10^{25}$ such substitutions. So the exhaustive search can't break the cipher in reasonable time. The problem here is that the alphabet is too small and a frequency analysis is applicable. Let us consider the cipher-text:

xlivi	*evixa*	*somrh*	*wsjgv*	*ctxsw*	*cwxiq*
wwcqq	*ixvmg*	*erhew*	*cqqix*	*vmgwc*	*qqixv*
mggvc	*txswc*	*wxiqw*	*ywixl*	*iweqi*	*oicxl*
iwigv	*ixoic*	*xsirg*	*vctxe*	*rhhig*	*vctxe*
qiwwe	*kierh*	*ewcqq*	*ixvmg*	*gvctx*	*swcwx*
iqwyw	*isrio*	*icxli*	*tyfpm*	*goicx*	*sirgv*
ctxeq	*iwwek*	*ierhe*	*hmjji*	*virxo*	*icxli*
tvmze	*xioic*	*xshig*	*vctxm*	*xewwc*	*qixvm*
ggvct	*xswcw*	*xiqwe*	*viepw*	*sgepp*	*ihtyf*
pmgoi	*cgvct*	*xswcw*	*xiqw*	–	–

We write down the frequency(the number of occurrences divided by the cipher-text length) of characters in the cipher-text:

i	0.148	t	0.041	f	0.006
x	0.11	m	0.038	k	0.006
w	0.107	h	0.031	a	0.003
c	0.089	r	0.031	z	0.003
v	0.069	o	0.027	b	0.0
e	0.062	l	0.017	d	0.0
g	0.062	p	0.017	n	0.0
q	0.058	y	0.013	u	0.0
s	0.041	j	0.01	–	–

Also we take into account some of the most frequent digrams:

wc	0.034	xs	0.027
gv	0.034	ix	0.027
vc	0.031	ic	0.024
ct	0.031	mg	0.024
tx	0.031	oi	0.024
qi	0.027	xi	0.02

It is easy to see that the substitution cipher preserves the frequencies of the characters. We compare our data with the average frequencies of characters in English texts. It is worth mentioning that as the latter were derived by observing long English texts, our conclusions would be more precise if we deal with longer cipher-texts.

E	0.1231	L	0.0403	B	0.0162
T	0.0959	D	0.0365	G	0.0161
A	0.0805	C	0.032	V	0.0093
O	0.0794	U	0.031	K	0.0052
N	0.0719	P	0.0229	Q	0.002
I	0.0718	F	0.0228	X	0.002
S	0.0659	M	0.0225	J	0.001
R	0.0603	W	0.0203	Z	0.0009
H	0.0514	Y	0.0188	–	–

The most frequent digrams of the English texts are

TH	0.0315	ES	0.0145
HE	0.0251	ON	0.0145
AN	0.0172	EA	0.0131
IN	0.0169	TI	0.0128
ER	0.0154	AT	0.0124
RE	0.0148	ST	0.0121

The comparison of the above data reveals the substitution. That is simply the cyclic shift of the alphabet characters to 3 positions to the right. In so doing we get the plain-text:

$THERE$	$ARETW$	$OKIND$	$SOFCR$	$YPTOS$	$YSTEM$
$SSYMM$	$ETRIC$	$ANDAS$	$YMMET$	$RICSY$	$MMETR$
$ICCRY$	$PTOSY$	$STEMS$	$USETH$	$ESAME$	$KEYTH$
$ESECR$	$ETKEY$	$TOENC$	$RYPTA$	$NDDEC$	$RYPTA$
$MESSA$	$GEAND$	$ASYMM$	$ETRIC$	$CRYPT$	$OSYST$
$EMSUS$	$EONEK$	$EYTHE$	$PUBLI$	$CKEYT$	$OENCR$
$YPTAM$	$ESSAG$	$EANDA$	$DIFFE$	$RENTK$	$EYTHE$
$PRIVA$	$TEKEY$	$TODEC$	$RYPTI$	$TASSY$	$METRI$
$CCRYP$	$TOSYS$	$TEMSA$	$REALS$	$OCALL$	$EDPUB$
$LICKE$	$YCRYP$	$TOSYS$	$TEMS$	–	–

We put down the obtained plain-text with gaps and punctuation marks now:

THERE ARE TWO KINDS OF CRYPTOSYSTEMS SYMMETRIC AND ASYMMETRIC. SYMMETRIC CRYPTOSYSTEMS USE THE SAME KEY, THE SECRET KEY, TO ENCRYPT AND DECRYPT A MESSAGE AND ASYMMETRIC CRYPTOSYSTEMS USE ONE KEY, THE PUBLIC KEY, TO ENCRYPT A MESSAGE AND A DIFFERENT KEY, THE PRIVATE KEY, TO DECRYPT IT. ASSYMETRIC CRYPTOSYSTEMS ARE ALSO CALLED PUBLICKEY CRYPTOSYSTEMS.

We summarize the method:

1. Find the frequencies of characters(digrams and trigrams for a short text) in the ciphertext,
2. Range the frequencies from the most significant to the least significant,
3. Compare with ranged frequencies of the characters(digrams and trigrams) of English.
4. Make a guess on the substitution or a part of it.
5. Check the guess by partially decrypting the cipher-text. The guess is correct if the result is a sensible English text.

1.2 Unicity Distance

Let us consider a substitution cipher given by a substitution S. We saw that the longer the cipher-text the easier is the problem of finding the plain-text, such that if we have several hundreds of characters it is very easy to determine the plain-text. We will consider the question:

What is the minimum amount of cipher-text such that it is still possible to determine the plain-text uniquely on the average? We call this number the unicity distance. For short cipher-texts it is relatively easy to bump into a cipher-text that can be decrypted into different plain-texts. For instance, the cipher-text "vgfuba" may be decrypted to "remain" to "should" with two different substitutions. This example is rather trivial because these two words include the same number of characters and they don't repeat. An interesting problem is to find two English words(phrases) as long as possible that may be encrypted into one cipher-text by applying two different substitutions. It is clear the longer are phrases the lesser are chances to find them. They might not even exist. Return to the unicity distance. Let

$$S = S_1, S_2 \ldots, S_t$$

be all possible substitutions, so $t = 26!$. Let c be a particular cipher-text of length n. Consider the decryption of c by using all the above substitutions S_i:

$$m = S^{-1}(c), m_1 = S_2^{-1}(c), \ldots, m_t = S_t^{-1}(c).$$

We want to estimate the minimal n such that between $m_1, \ldots, m_t$ there is at most one English text. We see that m_i are n-strings of English characters. On the whole there are just 26^n such strings, because for each of n places in the string there are 26 possibilities.

We estimate the number of n-strings that are English texts. One sees that this number should be much less than 26^n. The estimation was given by Shannon, who empirically showed this to be about

$$2^{H_E n},$$

where H_E is the entropy of English texts per character and $1 \leq H_E \leq 1.5$. The problem of estimation H_E looks very difficult. We take $H_E \approx 1.5$. So we assume there are $2^{1.5n}$ English texts of length n when n is big.

The probability that a particular n-string is an English text is $p \approx 2^{1.5n}/26^n$ and the probability that an n-string is not an English text is $1 - p$. We can consider $m_1, \ldots, m_t$ as random n-strings. We say now m_i is success if it is a sensible English text. Let γ_i be random variable that has 2 values: $\gamma_i = 1$ if m_i is a success and $\gamma_i = 0$ otherwise. Then the number of sensible English texts in $m_1, \ldots, m_t$ is

$$\gamma_1 + \gamma_2 + \ldots + \gamma_t.$$

Compute now the expectation of this random variable. We have

$$\mathbf{E}(\gamma_1 + \gamma_2 + \ldots + \gamma_t) = \mathbf{E}\gamma_1 + \mathbf{E}\gamma_2 + \ldots + \mathbf{E}\gamma_t = tp$$

because $\mathbf{E}\gamma_i = p$. We want to find minimal n such that this value is at most 1, in other words, at most one sensible decryption. That is $tp \leq 1$. This is equivalent to

$$26!(2^{H_E}/26)^n \leq 1$$

or

$$n \geq \frac{\log_2(26!)}{\log_2 26 - H_E} \approx 27$$

It follows that for $H_E = 1.5$ the unicity distance can be taken ≈ 27. This agrees with empirical evidence, that it is about 25 or so cipher-text characters defines the plain-text uniquely. Though it may be a rather hard computational task to find it. In the same fashion for any cipher that deals with encrypting English text it can be derived that the unicity distance is approximately

$$\frac{\log_2 |K|}{\log_2 26 - H_E}.$$

For other languages the alphabet size and the entropy may be different.

In above $\mathbf{E}$ is the expectation operator. That gives an average value of a random variable. For instance, let $\gamma = x$ with probability p, and $\gamma = y$ with probability $1 - p$, where $0 \leq p \leq 1$. By definition $\mathbf{E}\gamma = xp + y(1 - p)$. Useful property of $\mathbf{E}$ is that it is linear: expectation of the sum of random variables is the sum of their expectations.

Properties of substitution cipher:

1. very simple in usage, great advantage,

2. the cipher-text preserves the statistics of the plain-text, so the cipher is breakable by using only cipher-text, great disadvantage.

The further development of block ciphers went in the direction to avoid preserving plain-text statistics in cipher-text and to keep the usage simplicity.

1.3 Homophonic Ciphers

For homophonic ciphers the block size is still $t = 1$. Let A be the alphabet of English and A^2 be the set of all digrams

$$A^2 = \{aa, ab, \ldots, zz\}.$$

The size of A^2 is $26^2 = 676$. We partition the set A^2 into 26 subsets of generally different size:

$$A^2 = H_a \cup H_b \cup \ldots \cup H_z, \tag{1}$$

where subsets are encoded by characters A. A homophonic cipher is defined by

1. key space is all possible partitions (1),
2. encryption. Let the message be represented by characters $m_1, m_2, \ldots$. To encrypt m_1 the sender Alice takes a digram $m_{11}m_{12} \in H_{m_1}$ at random. To encrypt m_2 the sender Alice takes a digram $m_{21}m_{22} \in H_{m_2}$ at random and so on. The cipher-text is $c = m_{11}m_{12}m_{21}m_{22}, \ldots$
3. decryption. To decrypt c the receiver Bob takes the first digram $m_{11}m_{12}$ and search it through all H_α. When $m_{11}m_{12} \in H_{m_1}$ he decrypts the digram as m_1 and so on.

Properties of the homophonic cipher:

1. encryption-decryption is complicated, especially decryption,
2. the cipher-text is twice longer the plain-text,
3. the size of the key space is much bigger than that of the substitution cipher. For given $|H_a|, \ldots, |H_z|$ the size of the key space is

$$\binom{26^2}{|H_a|}\binom{26^2 - |H_a|}{|H_b|}\ldots,$$

   where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the number of k-subsets of an n-set.
4. statistics. The homophonic cipher may be used to make the frequencies of cipher-text characters uniform(at the expense of longer cipher-texts). Let $q_a, q_b, \ldots, q_z$ be the frequencies of characters in English. One takes the size of $H_a, H_b, \ldots, H_z$ such that $q_\alpha/|H_\alpha| \approx 1/26^2$, that is

$$\begin{aligned} |H_a| &\approx 26^2 q_a = 26^2 \times 0.0805 \approx 54 \\ |H_b| &\approx 26^2 q_b = 26^2 \times 0.0162 \approx 11 \\ &\ldots \\ |H_z| &\approx 26^2 q_z = 26^2 \times 0.0009 \approx 1, \end{aligned}$$

   and $|H_a| + |H_b| + \ldots + |H_z| = 26^2$.

We look now at the frequencies(probabilities) of digrams, encrypting characters, in the cipher-text. Let $m_1m_2 \in H_\alpha$ be any such digram. Then the probability of the digram $c = m_1m_2$ in the cipher-text is

$$\begin{aligned}\mathbf{Pr}(c = m_1m_2) &= \mathbf{Pr}(x = \alpha)\,\mathbf{Pr}(c = m_1m_2/x = \alpha) \\ &= \frac{\mathbf{Pr}(x = \alpha)}{|H_\alpha|} = \frac{q_\alpha}{|H_\alpha|} \approx \frac{1}{26^2}\end{aligned}$$

by the choice of $|H_\alpha|$. This means the probabilities of digrams are approximately the same.

So the homophonic cipher is a good one in sense that no way is known to find a plain-text from the cipher-text. However from this is a weak cipher. The modern requirement is the strength against known plain-text attack. In the case of a homophonic cipher an adversary is obviously able to recover the key, that is sets H_α, if he collects a number of plaintext-ciphertext pairs.

The conditional probability formula

$$\mathbf{Pr}(AB) = \mathbf{Pr}(B)\,\mathbf{Pr}(A/B)$$

for two events A, B was used. Here AB denotes an event, where A and B occur simultaneously. In above $A : c = m_1m_2$, implies $B : x = \alpha$, so AB is equivalent to A.

Problem 1 *Prove that the distribution of cipher-text characters is uniform. Hint: fix a character α and count the number of digrams encrypting plain-text characters and including α*

1.4 Polyalphabetic Substitution Ciphers

In this ciphers the block size(called also period) is $t > 1$.

1. The key space consists of some ordered set of substitutions $(p_1, p_2, \ldots, p_t)$, where p_i are defined on the alphabet A and t is secret as well.

2. Encryption of the block of characters $m = m_1, \ldots, m_t$ under the key $e = (p_1, p_2, \ldots, p_t)$ is given by

$$E_e(m) = p_1(m_1), p_2(m_2), \ldots, p_t(m_t).$$

3. The decryption key is $d = (p_1^{-1}, p_2^{-1}, \ldots, p_t^{-1})$ and the decryption procedure is obvious.

The plain-text block m may be encoded such that m_i are characters from different alphabets A_i. So p_i are substitutions on A_i. This implies the name polyalphabetic. When $A_1 = \ldots = A_t = A$ the cipher is called Vigenère cipher.

Example. Let A be the English alphabet and $t = 3$. Choose $e = (p_1, p_2, p_3)$, where p_1 maps each character to three positions to the right, p_2 to seven position and p_3 to ten positions, one can write this as $e = (3, 7, 10)$. So the phrase "THE-DAT—AEN—CRY—PTI—ONS—TAN—DAR—D" is encrypted to the cipher-text "woo—ghd—dlx—fyi—sas—ruc—whx—ghb—g". The character "A" was encrypted to the characters "h,d" and the character "T" to "d,a,w" depending on the position in the plain-text.

From the example we see that the cipher generally doesn't preserve the frequencies of the plain-text. So to analyze a Vigenère cipher it is important to define t and then use frequency analysis to get the key substitutions and the plain-text. Two ways to determine t are known.

1. Kasiski test. It is based on the observation that two identical segments of plain-text are encrypted to the same cipher-text whenever they occur in the plain-text at a distance multiple to t. In our example the digram da occurs twice in the plain-text at the 4th and the 22nd positions. Here $22 - 4 \equiv 0(mod \quad 3)$. This digram is encrypted to the same digram gh.

   If we observe two identical segments of cipher-text, the length of which is at least three, it is likely that they correspond to identical segments of plain-text and the difference of their positions is a multiple of t. The chance is bigger if segments are longer. The Kasiski test works as follows

   - search the cipher-text for pairs of identical segments of length at least three and record the distance d_i between the starting positions of the two segments.
   - If we obtain several such distances, say $d_1, d_2, \ldots$, then it is very probable that t divides their gcd. Very often t is just this gcd. An error is possible: two identical cipher-text segments are not encryptions of identical plain-text segments. Therefore they are not

at a distance multiple of t. E.g. let $d_1, d_2, \ldots, d_r$ be right distances, that is $\gcd(d_1, d_2, \ldots, d_r) = t$, but d is a wrong one. So $\gcd(d_1, d_2, \ldots, d_r, d) < t$ and is usually very small. In so doing one detects the wrong d.

2. The second method is called cipher-text auto-correlation. Let $c_1, c_2, \ldots, c_L$ be a cipher-text. Count the number of occurrences $c_i = c_{i+s}$ for all $1 \leq i \leq L-s$. Begin with $s = 1$ and continue for $s = 2, 3 \ldots$ until some natural bound for t. Tabulate the counts or counts divided by $L-s$, which are the probabilities of coincidences like $c_i = c_{i+s}$ in this case. Then the actual period t is computed as follows. Take the first s with a relatively higher probability and put $t = s$.

We justify the auto-correlation method now. We fix s and compute the probability that two random characters in the cipher-text at the distance s are identical. The plain-text may be considered random, so it is enough to compute $\mathbf{Pr}(c_i = c_{i+s})$ for some i. We will do this under two different conditions.

Remark that $\mathbf{Pr}(c_i = c_{i+s}) = \mathbf{Pr}(p_u(m_i) = p_v(m_{i+s})) = \mathbf{Pr}(m_i = h(m_{i+s}))$, where $u \equiv i$, $v \equiv i + s \pmod{t}$ and $h = p_u^{-1} p_v$. By complete probability formula,

$$\begin{aligned}
\mathbf{Pr}(m_i = h(m_{i+s})) &= \sum_{\alpha} \mathbf{Pr}(m_{i+s} = \alpha)\mathbf{Pr}(m_i = h(m_{i+s}) | m_{i+s} = \alpha) \\
&= \sum_{\alpha} \mathbf{Pr}(m_{i+s} = \alpha)\mathbf{Pr}(m_i = h(\alpha)) \\
&= \sum_{\alpha} q_\alpha q_{h(\alpha)} \leq \sum_{\alpha} q_\alpha^2.
\end{aligned}$$

The latter by Cauchy inequality. Firstly, let A_0 be the event s is a multiple of t. If A_0, then h is identity and

$$\mathbf{Pr}(c_i = c_{i+s} | A_0) = \sum_{\alpha} q_\alpha^2 \approx 0.065.$$

Therefore, the probability of coincidence at distance s is about 0.065 and does not depend on i. Secondly, let A_1 denote the event s is not a multiple of t. Then h is, generally, not identity and one can prove

$$\mathbf{Pr}(c_i = c_{i+s} | A_1) \approx \frac{1}{26} = 0.038...$$

on the average. Therefore, the probability of coincidence at distance s is about 0.038. We are able to distinct which event A_0 or A_1 have occurred by computing the probability of coincidence at distance s. If it is close to 0.065 we conclude that s is a multiple of t, otherwise, and it is close to 0.038, we conclude that s is not.

Having found the correct value of t we determine the key $p_1, p_2, \ldots, p_t$ and the plain-text. The way to proceed depends on if the substitutions $p_1, p_2, \ldots, p_t$ are randomly chosen (this variant of the cipher is called full Vigenère cipher) or are right shifts by $k_1, k_2, \ldots, k_t$ positions. We will consider the both.

1. Full Vigenère. Write down the cipher-text as an array of dimension $t \times (\approx n/t)$, where n is the cipher-text length. The rows of the array are substrings

$$\begin{aligned} C(1) &= c_1, c_{1+t}, c_{1+2t}, \dots, \\ C(2) &= c_2, c_{2+t}, c_{2+2t}, \dots, \\ &\dots, \\ C(t) &= c_t, c_{2t}, c_{3t}, \dots. \end{aligned}$$

We see that $C(i) = c_i, c_{i+t}, c_{i+2t}, \dots$ is produced from the plain-text $m(i) = m_i, m_{i+t}, m_{i+2t}, \dots$ with the substitution p_i. Therefore, $C(i)$ preserves the frequencies of the characters in $m(i)$, which is a substring of the plain-text of m. So the frequencies of characters in $C(i)$ and hence in $m(i)$ should be like frequencies of English. We can't say that about digrams and trigrams in $C(i)$. Anyway if n/t is big enough one is able to reveal p_i from the character frequencies only.

- Write down the cipher-text as an array.
- Compute character frequencies in each $C(i)$
- Compare the latter with character frequencies in English. Make a guess on p_i.
- Check the guess by partially decrypting the cipher-text. The guess is correct if the cipher-text is decrypting to a sensible text.

2. Simple Vigenère. Let $r_0, r_1, \dots, r_{25}$ be the character frequencies(probability distribution) in $C(i)$, which was obtained by the shift to k_i positions from a substring of the plain-text. As α was encrypted to $\alpha + k_i$ there should be $q_\alpha \approx r_{\alpha+k_i}$. So the hope is that the probability distribution shifted to k_i positions

$$r_{k_i}, r_{1+k_i}, \dots, r_{25+k_i},$$

is close to the character distribution in English. Here indexes are taken modulo 26. We introduce a measure of the closeness

$$M_s = \sum_{\alpha \in A} q_\alpha r_{\alpha+s}.$$

For $s = k_i$

$$M_s = \sum_{\alpha \in A} q_\alpha^2 \approx 0.065.$$

But when $s \neq k_i$, the number M_s is strictly smaller, it is about 0.038 on the average. We summarize the method now.

- Write down the cipher-text as an array of strings $C(i)$.
- For each $C(i)$ compute the vector of frequencies $r_0, r_1, \dots, r_{25}$.

- For each $s = 0, 1, \ldots, 25$ compute M_s. Make the guess $k_i = s$ for the maximal M_s.
- Check the guess by partially decrypting the cipher-text.

Although the attack is rather simple, it takes some time. We estimate the number of steps. For each i one looks through the substring $C(i)$ of size $\approx n/t$ and computes the sum M_s for each s modulo 26. The sum involves 26 summands, so the algorithm runs in time

$$t(n/t + 26^2) \approx n + t26^2$$

some simple steps. The number of keys is 26^t. The above algorithm is much better than an exhaustive search. But it only works when the number of cipher-text characters n is quite big.

Complete probability formula: let $H_1, \ldots, H_k$ be disjoint events, that is $\mathbf{Pr}(H_i H_j) = 0$, and such that $\sum_{i=1}^{k} \mathbf{Pr}(H_i) = 1$. Then for any event A

$$\mathbf{Pr}(A) = \sum_{i=1}^{k} \mathbf{Pr}(H_i)\, \mathbf{Pr}(A|H_i).$$

Cauchy inequality:

$$\left(\sum_{i=1}^{k} a_i b_i \right)^2 \leq \left(\sum_{i=1}^{k} a_i^2 \right) \left(\sum_{i=1}^{k} b_i^2 \right).$$

Consider the cipher-text.

kphpw	*qzxrg*	*rhlvp*	*zthlt*	*fzyum*	*ujeyb*
glrki	*upgpi*	*lesbl*	*pfwau*	*hnvvl*	*vphix*
izvlm	*kpxyt*	*yppsx*	*uelyh*	*xrlsb*	*ipgvf*
hdxvt	*qlgjx*	*seeuv*	*hzjyx*	*dwmar*	*lemzt*
qtpsn	*vtsum*	*klxfh*	*xelpl*	*kltwr*	*dymse*
xdmvg	*rqxoh*	*vpaoh*	*klzle*	*rdxpm*	*efxaa*
hjsbg	*jvrvp*	*wsift*	*upayx*	*wnllw*	*izvaa*
hjeyx	*ifpsh*	*iellm*	*ufxoe*	*hdwpw*	*hlpzp*
ktgoa	*dgiix*	*hymul*	*wtpsx*	*gtrah*	*wsitt*
qoihv	*kemtx*	*wsifv*	*rxipg*	*fzrat*	*feapm*
kellk	*hlpaa*	*hjeyx*	*ecypl*	*hoeuw*	*zzyuw*
ho					

The frequency analysis shows that it was not produced with a simple substitution cipher as the characters frequencies are different from that in English.

l	0.081	w	0.042	k	0.030
p	0.078	r	0.036	d	0.021
h	0.069	y	0.039	j	0.021
x	0.066	m	0.036	o	0.021
e	0.057	s	0.039	q	0.015
t	0.048	u	0.036	b	0.012
v	0.048	z	0.036	n	0.012
i	0.045	f	0.033	c	0.003
a	0.042	g	0.033		

So assume that was a Vigenère cipher. We try the Kasiski test to find the period. It appears that the substring of 7 characters "aahjeyx" occurs from the positions 309 and 209, that is at distance 100. Then "psx" starts in the positions 258 and 73, at distance 185. There are several other trigrams occurring twice:"izv","wsif" and "hlp" at distance multiple of 5. So likely $t = 5$.

We arrange now the cipher-text as an array of rows, where the row $C(1)$ is

kqrzfugulphvikyuxihqshdlqvkxkdxrvkrehjwuwihiiuhhkdhwgwqkwrffkhhehzh.

The frequencies of characters are entries from "a" to "z" of the vector:

$$r = (0, 0, 0, 3/67, 2/67, 3/67, 2/67, 12/67, 5/67, 1/67, 8/67, 2/67, 0,$$
$$0, 0, 1/67, 4/67, 4/67, 1/67, 0, 5/67, 3/67, 5/67, 3/67, 1/67, 2/67).$$

We compute now the dot-product of the vector q representing the frequencies of characters in English with the vector r cyclically shifted to $0, 1, \ldots, 25$ positions to the left. We get a vector of dot-products arranged according to shifts.

$$(0.032, 0.031, 0.041, \mathbf{0.065}, 0.043, 0.034, 0.041, 0.041,$$
$$0.031, 0.031, 0.032, 0.026, 0.0341, 0.0341, 0.044, 0.041, 0.048,$$
$$0.044, 0.042, 0.042, 0.038, 0.030, 0.0381, 0.035, 0.027, 0.040)$$

The dot-product related to the shift to 3 positions is maximal. So the first entry in the encryption key is 3. In so doing we reveal the whole encryption key $(3, 11, 4, 7, 19)$ and decrypt the cipher-text. That is "HE DID NOT KNOW HOW WIDE A COUNTRY ARID AND PRECIPITOUS MUST BE CROSSED BEFORE THE TRAVELLER THROUGH LIFE COMES TO AN ACCEPTANCE OF REALITY IT IS AN ILLUSION THAT YOUTH IS HAPPY AN ILLUSION OF THOSE WHO HAVE LOST IT BUT THE YOUNG KNOW THEY ARE WRETCHED FOR THEY ARE FULL OF THE TRUTHLESS IDEALS WHICH HAVE BEEN INSTILLED INTO THEM AND EACH TIME THEY COME IN CONTACT WITH THE REAL THEY ARE BRUISED AND WOUNDED"

Index of coincidence. We will define the index of coincidence as a tool to distinct English text from strings of characters. That may be important when a cryptanalysis decrypts(may be partly) the cipher-text and decides how good his guess was.

Definition 1 *Suppose $x = x_1, x_2, \ldots, x_n$ be a string of n alphabetic characters. The index of coincidence of x, denoted $I_c(x)$, is defined to be the probability that two random elements of x are identical.*

We compute the index of coincidence for a random n-string of English characters, then for a random English text of length n and finally we provide with a short formula to compute the index for a given n-string.

Let $x = x_R$ be a random n-string of English characters. We fix two indices $1 < i < j < n$ and find the probability of the coincidence

$$\mathbf{Pr}(x_i = x_j) = \sum_{\alpha \in A} \mathbf{Pr}(x_i = x_j = \alpha).$$

As x_i and x_j are independently chosen we can write

$$\mathbf{Pr}(x_i = x_j) = \sum_{\alpha \in A} \mathbf{Pr}(x_i = \alpha)\mathbf{Pr}(x_j = \alpha) = \sum_{\alpha \in A} 1/26^2 = \frac{1}{26}.$$

This number doesn't depend on the indices $i < j$ so $I_c(x) = \frac{1}{26} \approx 0.038$ too. Really,

$$I_c(x) = \sum_{i<j} \frac{1}{\binom{n}{2}} \mathbf{Pr}(x_i = x_j) = \frac{1}{26},$$

where the sum is over all possible indices $i < j$.

Let $m = x_E$ be a random English text of n characters. Again we fix a pair of indices $1 < i < j < n$ and find the probability of two characters m_i and m_j in the above positions to be identical. By complete probability formula,

$$\mathbf{Pr}(m_i = m_j) = \sum_{\alpha} \mathbf{Pr}(m_i = \alpha)\mathbf{Pr}(m_i = m_j | m_i = \alpha) = \sum_{\alpha} q_\alpha^2 \approx 0.065.$$

The number $\mathbf{Pr}(m_i = m_j)$ doesn't depend on the indices. Therefore, $I_c(x_E) = \sum_{\alpha \in A} q_\alpha^2 \approx 0.065$.

Let x be a fixed n-string of English characters and f_α denote the number of appearances of α in x. We see that the number of characters x_i, x_j in different positions such that $x_i = x_j$ is $\sum_{\alpha \in A} \binom{f_\alpha}{2}$. The total number of such pairs is $\binom{n}{2}$. So

$$I_c(x) = \frac{\sum_{\alpha \in A} \binom{f_\alpha}{2}}{\binom{n}{2}} = \frac{\sum_{\alpha \in A} f_\alpha(f_\alpha - 1)}{n(n-1)}.$$

E.g. let $x =$ "$THEDATAENCRYPTIONSTANDARD$". We compute numbers f_α. For $\alpha = A, T$ it is 4, for $\alpha = D, N$ it equals 3, for $\alpha = E, R$ we have $f_\alpha = 2$ and $f_\alpha = 1$ when $\alpha = C, H, I, O, P, S, Y$. All other f_α are zeros. So

$$I_c(x) = \frac{4 \times 3 + 4 \times 3 + 3 \times 2 + 3 \times 2 + 2 \times 1 + 2 \times 1}{25 \times 24} = 40/600 \approx 0.066$$

is close to the I_c of a random English text.

The index of coincidence may be used to distinct English texts from random string of characters in automated decryptions. Being close to 0.066 it indicates correct or partially correct decryption. A stronger tool based on the distribution of digrams and trigrams may be defined too. Remark, that for some primitive ciphers as substitutions and transpositions the index may not distinct cipher-texts from plain-texts.

1.5 Transposition cipher

Transposition cipher is a symmetric block cipher with block length t over an alphabet A.

1. The key space consists of all permutations on $1, 2, \ldots, t$. The encryption key is a permutation e and $d = e^{-1}$ is a decryption key.

2. In order to encrypt a block $m = m_1, m_2, \ldots, m_t$, one permutes block characters according to e. So that the cipher-text block is

$$c = c_1, c_2, \ldots, c_t = m_{e(1)}, m_{e(2)}, \ldots, m_{e(t)}$$

   If the plain-text length is not a multiple of t, the last block should be padded, with random characters or in some other way.

3. The decryption works as follows

$$m = c_{d(1)}, c_{d(2)}, \ldots, c_{d(t)}$$

The encryption-decryption works because $c_{d(i)} = m_{e(d(i))} = m_i$.

Let the plain-text be given

$$m_{11}m_{12}\ldots m_{1t}\, m_{21}m_{22}\ldots m_{2t}\ldots,$$

as a sequence of blocks. So the cipher-text is

$$m_{1e(1)}m_{1e(2)}\ldots m_{1e(t)}\, m_{2e(1)}m_{2e(2)}\ldots m_{2e(t)}\ldots,$$

We see that the characters in the same positions in the plain-text blocks are encrypted in the same way. Rewrite now blocks as an array

$$\begin{aligned} &m_{1e(1)}m_{1e(2)}\ldots m_{1e(t)},\\ &m_{2e(1)}m_{2e(2)}\ldots m_{2e(t)},\\ &\qquad\qquad\ldots \end{aligned} \tag{2}$$

The transposition cipher keeps the frequency of the characters but not diagrams. It therefore can be easily recognisable. Below we show a cipher-text only attack.

The number t may be a part of the key. To recover t one tries all numbers from some region known to be correct. However a method similar to the Kasiski test is possible here. Having guessed the period, how to proceed further in order to break the cipher? First, try to find in one of the above rows(e.g i-th row)a permuted common word or a part of it(syllabus), permute back the columns to get this word. Then look at another rows to check the guess. Namely, look at the positions in other rows that compose the word in the i-th row. If all these combinations are not sensible, try another i or another word.

Example. Let we have the cipher-text *hatencaetaynostaiprtdskonxdanr*. Assume that it was produced with a transposition cipher, where $t = 10$. So we

can proceed as above. But can't believe much in the last block, which may be padded.

1	2	3	4	5	6	7	8	9	10
h	a	t	e	n	c	a	e	t	d
y	n	o	s	t	a	i	p	r	t
d	s	k	o	n	x	d	a	n	r

3	1	4	3	1	8	9	1	4	9	1	8
t	h	e	t	h	e	t	h	e	t	h	e
o	y	s	o	y	p	r	y	s	r	y	p
k	d	o	k	d	a	n	d	o	n	d	a

5	7	3	2	10	7	3	2
n	a	t	a	d	a	t	a
t	i	o	n	t	i	o	n
n	d	k	s	r	d	k	s

9	1	8	10	7	3	2	4	5	6
t	h	e	d	a	t	a	e	n	c
r	y	p	t	i	o	n	s	t	a
n	d	a	r	d	k	s	o	n	x

In so doing we have revealed the decryption key given by the permutation $(9, 1, 8, 10, 7, 3, 2, 4, 5, 6)$.

Although a transposition cipher is not strong against cipher-text only attack, a superposition of two transpositions with periods t_1 and t_2 is much stronger. This is equivalent to a single transposition cipher with period $lcm(t_1, t_2)$, the least common multiple of t_1 and t_2. Every of them is broken with known plain-text attack anyway for a long enough known plain-text.

Problem 2 *Does the transposition cipher preserve the frequency of plain-text characters? What about digrams?*

Problem 3 *Is the transposition cipher more secure if the block size is bigger?*

1.6 Rotors Machines

Rotor machines were the most important ciphers of the World War II and remained in use at least until the late of fifties. The most famous one was German "Enigma". It was broken by the Poles and then by the British.

The central component of a rotor machine is the rotor or wired wheel. This a disk serving to implement a simple substitution. Around the perimeter of the each face there are 26(for the Latin alphabet) evenly spaced electrical contacts. Each contact on the front face is wired to exactly one contact on the rear face. The contacts are marked with alphabet characters from "A" to "Z" following clock direction. Assume input and output plugs marked with alphabet characters. In the initial position the rotor implements a substitution R. Let it be for example

$$\begin{array}{cccccc} A & \dots & L & M & \dots & Z \\ S & \dots & Y & A & \dots & D. \end{array}$$

So, in the initial position, the signal from the character M on the input plug goes to the character M on the front face, then to the character A on the rear face and finally to the character A on the output plug:

$$M \to M \to A \to A.$$

The rotor is on a bar and is able to rotate while the plugs remain still. When the rotor rotates to one position following clock, the whole device implements another substitution. That is after one rotation the path for the character M is $M \to L \to Y \to Z$ as in the next Figure. Mathematically, after one rotation

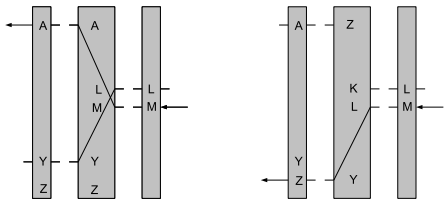

Figure 1: Rotor before and after rotation.

the substitution is CRC^{-1} and after j rotations that is C^jRC^{-j}. Here C is a cyclic shift to one alphabet character back(that is e.g. $L \to M$ and $Y \to Z$.) So that

$$CRC^{-1}(M) = CR(L) = C(Y) = Z.$$

Rotors are connected one after the other in a bank such that a signal entering at one end of the bank is permuted by all rotors in the bank. To make the cipher stronger a plugboard substitution S is applicable. The substitution implemented by the cipher is

$$C^{j_1}R_1C^{-j_1}C^{j_2}R_2C^{-j_2}C^{j_3}R_3C^{-j_3}S \tag{3}$$

after j_i shifts of the i-th rotor. When the rotors rotate the bank implements a large variety of different substitutions, at most $26^3 = 17576$. To decrypt one is to implement an inversion of (3) and it is not very convenient to have different devices for encryption and decryption. Mathematically, for the current state j_1, j_2, j_3, "Enigma" implements the substitution:

$$S(j_1, j_2, j_3) =$$
$$S^{-1}C^{j_3}R_3^{-1}C^{j_2-j_3}R_2^{-1}C^{j_1-j_2}R_1^{-1}C^{j_1}TC^{j_1}R_1C^{j_2-j_1}R_2C^{j_3-j_2}R_3C^{-j_3}S,$$

where T is a substitution implemented by the reflector. As for the reflector $T = T^{-1}$, the similar is true for the substitution implemented by the Enigma:

$$S(j_1, j_2, j_3) = S(j_1, j_2, j_3)^{-1}.$$

Therefore, the same device is used both for the encryption and decryption.

The sequence of shifts (j_1, j_2, j_3) are called states. In order to build a strong cryptographic system the state should change from one character to the other. So the rotor machine implements a Vigenèr cipher with a large period, up to 26^3. It consists of a bank of rotors and a mechanism for changing the state each time a character is encrypted along with some other auxiliary devices like plugs, bulbs.

There are two requirements on the way how the rotors rotate:

1. the period of states should be large, otherwise frequency analysis is applicable as for a Vigenère cipher with short period.

2. each change of state should be large. That is the state

   $$(j_1(t+1), j_2(t+1), j_3(t+1))$$

   at the time $t+1$ should differ much from the state $(j_1(t), j_2(t), j_3(t))$ at the time t.

Theoretically it is easy to provide with these properties. For example, one fixes t(the number of rotors in the bank) pairwise coprime integers $k_1, \ldots, k_t$ coprime also to 26. Then the motion is described by the rule:

$$j_i(t+1) \equiv j_i(t) + k_i (mod\, 26).$$

The period of the states is 26^t. But it may be difficult to implement it mechanically. In "Enigma" an odometer like motion of rotors was accepted. The rightmost wheel rotates once each time a character is encrypted. The next rotor

rotates once after the first makes the whole round of 26 shifts. The leftmost rotates once after 26^2 rotations of the first two wheels. So the sequence of states is

$$(0,0,0),\ (0,0,1),\ldots,(0,0,25),\ (0,1,0),(0,1,1),\ldots \tag{4}$$

The Enigma machine was intent to both encrypt and decrypt. So it looks a bit mere complicated, see the Figure. The cipher key consisted of the two parts:

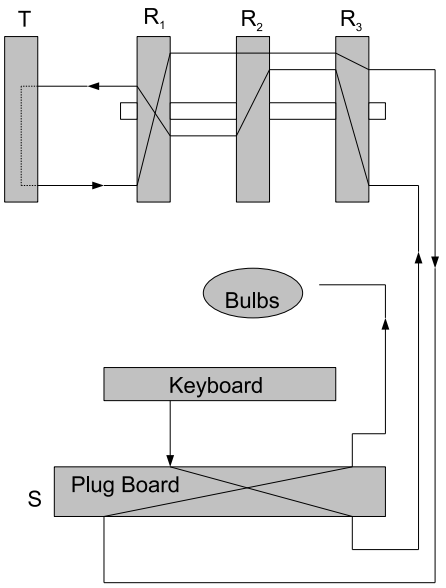

Figure 2: A model of the Enigma machine.

1. daily key, describing the order of rotors and their initial positions. This daily key was used by all Enigmas,
2. message key chosen by an operator and defining the initial state of the rotors. As there were three rotors, the state was defined by a triple of alphabet characters.

Several factors made the cipher break possible:

1. The French managed to bought tables of daily keys for two consecutive months September and October, 1932.

2. German operators tended to select simple predictable patterns as message keys. E.g. three identical characters or characters corresponding to three neighboring keys on the keyboard.

3. They encrypted this message key with the daily key before encrypting the message and sent it with the cipher-text as a prefix.

1.7 G-Schreiber

G-Schreiber was another cipher-machine used by the Germans during World War II. As its period was much bigger than that of Enigma it was supposed to be even more secure. For encrypting-decrypting and transmitting messages a teleprinter was used. That is, roughly speaking, before encrypting the plain-text was represented as a sequence of 5-bit groups. The international teleprinter code , CCITT2, was used:

Letter shift	5-bit group	Figure shift
A	11000	-
B	10011	?
C	01110	:
D	10010	"Who's there?"
E	10000	3
F	10110	$\ddot{U}$
G	01011	$\ddot{A}$
H	00101	$\ddot{O}$
I	01100	8
J	11010	Bell
K	11110	(
L	01001	)
M	00111	.
N	00110	,
O	00011	9
P	01101	0
Q	11101	1
R	01010	4
S	10100	'
T	00001	5
U	11100	7
V	01111	=
W	11001	2
X	10111	/
Y	10101	6
Z	10001	+
CR	00010	CR
NL	01000	NL
LS	11111	LS
FS	11011	FS
SP	00100	SP
BL	00000	BL

The special characters are

CR	Carriage return
NL	New line
LS	Letter shift
FS	Figure shift
SP	Space
BL	Empty character

5-bit groups are enough to represent 32 characters but in order to use digits and a number of punctuation marks something else should be done. In CCITT2 the same 5-bit combination may represent two different characters depending on whether it is preceded by one of two controlling characters denoted by LS(Letter shift) and FS(Figure shift). Once LS(FS) appears, all subsequent 5-bit groups will be interpreted as if from the Letter shift(Figure shift) column of the code table until an FS(LS) appears. For instance, the plain-text "HELLO12AND.." got the following encoding by 5-bit groups:

11111 00101 10000 01001 01001 00011 11011
11101 11001 11111 11000 00110 10010..

The G-Schreiber was an electromechanical device with ten wheels. Each wheel had a number of equally spaced positions around the rim, each position representing 0 or 1. The number of the wheel positions are:

$$47, 53, 59, 61, 64, 65, 67, 69, 71, 73.$$

These are pairwise coprime numbers so the number of different positions of the whole set of ten wheels is

$$47 \times 53 \times 59 \times 61 \times 64 \times 65 \times 67 \times 69 \times 71 \times 73 \approx 9 \times 10^{17}$$

in case they rotate simultaneously.

The wheels feelers were connected to the rest of the machine via cables which could be permuted arbitrarily. The permutation is a part of cipher key setting. All wheels have numbering on their rims according to the number of positions starting from zero. That is the first wheel's positions are numbered by $0, 1, \ldots, 46$.

The encryption was done in two steps. First the plain-text character in its teleprinter code representation was XOR-ed with the 5-bit key-stream character produced by five of the wheels. In the example the five wheels generate the combination 01001. Assume the keyboard character R was pressed, then 5-bit combination 01010 is generated and XOR-ed with 01001 to produce $c = 00011$. The remaining five wheels generate a permutation on the latter five bits. The permutation is performed by five relays which are opened and closed according to whether the controlling pulse from the wheels were zero or one in Fig.3. That is the first relay switches the first and the fifth bits of c, then the second relay

Figure 3: How a switch works.

switches the fourth and the fifth, then the third switches third and the fourth, then the fourth relay switches the second and the third and then finally the last relay switches the first and the second bits. So after such permutation c is transformed to 11000, that is character A. Therefore, A is the encryption of R. But next time R should be encrypted to another character defined by the position of the wheels. The key-system consists of a

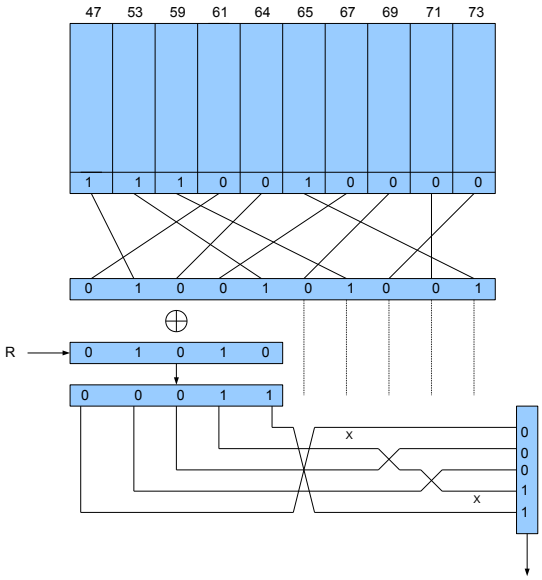

Figure 4: A model of the G-Schreiber machine.

1. long-term key: $0, 1$-distribution on wheels and cabling,
2. daily key: QEK numbers initial state on some 5 wheels ,

3. message key: QEP numbers initial state on the rest 5 wheels.

The initial position of the wheels was determined by two sets of numbers, QEP and QEK. The QEK numbers were the same for all telegrams during a 24-hour period. The QEP numbers were supposed to be different from telegram to telegram and were chosen by operators themselves.

Date,Wheel	1	2	3	4	5	6	7	8	9	10
May 2	12	32	...	...	...	...	...	15	06	44
May 3	42	11	58	02	...	...	...	...	...	68
May 4	22	...	...	...	...	...	67	30	58	62
May 5	37	15	27	26	29	...	...	...	...	...

The dots in the table represent the positions of the QEP numbers to be chosen by the operator. Thus, if on May 4 the operator chose the QEP numbers $31, 45, 13, 19, 4$, then the initial wheel positions would be

$$22, 31, 45, 13, 19, 4, 67, 30, 58, 62.$$

The cipher was broken by the Swedes during the war by cipher-text only attack. They reconstructed the machine and were able to read much of the German Navy communication, where G-Schreiber was in use. One thing helped the Swedes a lot. Germans operators were tending not to change QEP numbers from one telegram to another. So the Swedes collected a lot of cipher-text produced with the same key-stream.

From the modern point of view G-Schreiber is a weak cipher. It is vulnerable to the known plain-text attack. With some amount of the plain-text and related cipher-text produced with one key, it is easy to reveal all cipher's settings and decrypt the rest of the cipher-texts produced with the same key. We briefly describe a known-plaintext attack:

Stage 1. In the cipher-text find positions of 5-bit groups: 00000 and 11111, that is of weight 0 or 5. Reconstruct first 5 bits on the plug for those positions.

Stage 2. Reconstruct cabling to the first 5 plug positions and $0, 1$-distribution on relevant wheels.

Stage 3. In the cipher-text find positions of 5-bit groups of weight 1 or 4. Reconstruct the last 5 bits on the plug for those positions.

Stage 4. Reconstruct cabling to the last 5 plug positions and $0, 1$-distribution on relevant wheels.

The attack works when the cipher-text is long enough.

1.8 Hagelin Machine

Hagelin was a Swede, he improved a previously known cipher machine and established a Swiss company producing several types of his cipher-machine. One of them under military designation M-209 was widely used as an American field cipher during the World War II and for some time afterwards. We briefly consider a mathematical model for M-209. The cipher produces a sequence of alphabet characters x_i called key-stream. The cipher-text character c_i encrypting the plain-text character m_i is determined by $c_i \equiv x_i - m_i \mod 26$, where alphabet characters are represented as residues modulo 26. This makes encrypting and decrypting processes identical.

Two major components in the key-stream generation process are the key-wheels generating a long pseudo-random sequence of six-bit groups and the cage which converts the six-bit groups into characters. There are 6 key-wheels. They are gears with $26, 25, 23, 21, 19, 17$ teeth. The teeth are marked with alphabet characters: $A, B, \ldots, Z$ for the first gear, $A, B, \ldots, Y$ for the second, ... and $A, B, \ldots, Q$ for the last one. Next to each tooth is a pin. The pin may be in two positions: extended and retracted. The pin position is determined by the key such that an extended pin corresponds to a key bit equal to 1 and retracted to 0. So there are at least $26 + 25 + 23 + 21 + 19 + 17 = 131$ key-bits in the key.

At the beginning of the encryption process the key-wheel state is $AAAAAA$. The 6-bit group corresponding to the A teeth on each gear determines the first character of the key-stream. After the first character of the plain-text is encrypted all wheels are rotated to one position, so that the state becomes $BBBBBB$ and so on. This holds true for the first 17 characters. At that point the state is $QQQQQQ$. Then the 18-th character of the key-stream is determined by the state $RRRRRA$ since the last wheel has finished one complete revolution. So the 18-th character of the key-stream is correlated with the first character, the 19-th is correlated with the second. The 20-th is correlated with the third and the first key-stream characters, because the state at this point is $TTTTAC$. Because the numbers $26, 25, 23, 21, 19, 17$ are coprime it is easy to understand that the period of the key-stream is $26 \times 25 \times 23 \times 21 \times 19 \times 17 \approx 10^8$.

One can imagine cage as a read-only memory holding 64 characters of the Latin alphabet. Each 6-bit group produced by the key-wheel designates a memory location of a character which is the current key-stream character. Some observations on M-209.

1. the sequence of 6-bit groups may be considered as uniformly distributed,

2. the distribution of the key-stream characters can't be uniform, because there are 26 alphabet characters and 64 memory locations for them. For example, in M-209 12 alphabet characters may occur 3 times whilst the left 14 characters occur 2 times.

3. as the character distribution in the plain-text is not uniform, the cipher-text characters distributions is not uniform either,

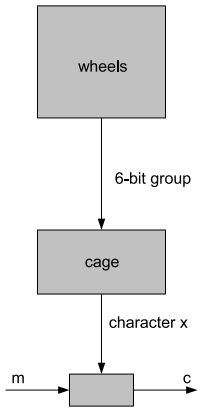

Figure 5: A model of the Hageline machine.

4. when one of the six bits entered the cage is fixed the non uniformity of the key-stream tends to increase. The distribution of the key-stream depends on which bit was fixed.

We will use the above observations for cryptanalysis. Let C be a cipher-text. We represent C as an array with 17 rows.

$$\begin{array}{rcccc} C(1) = & c_{1+17} & c_{1+2\times 17} & \dots \\ C(2) = & c_{2+17} & c_{2+2\times 17} & \dots \\ \dots & & & \\ C(17) = & c_{17+17} & c_{17+2\times 17} & \dots \end{array}$$

So the cipher-text characters $c_i, c_{i+17}, c_{i+2\times 17}, \dots$ were produced with key-stream characters where the last key-wheel is in the same position. For this position let A_1 designate the event "the pin is extended" and A_0 designate "the pin is retracted". The cage property is that the distribution of key-stream characters $x_{i+17j}, j = 0, 1, \dots$ under conditions A_1 and A_0 are different. So do the distributions of the cipher-text characters $c_{i+17j}, j = 0, 1, \dots$. By observing the above distribution we are able to distinct A_0 and A_1 for a fixed position of the last wheel.

Example. We demonstrate here that the distribution of the cipher-text is correlated with whether the last wheel pin was extended(value 1 in the right most position of the 6-bit group) or retracted(value 0).

Let the cage hold the following 64 characters in its locations:

$$A, A, A, B, B, B, \ldots, L, L, L, M, M, N, N, \ldots, Z, Z.$$

So that the first 12 alphabet characters $A, B, \ldots, L$ occur 3 times and the other 14 characters $M, N, \ldots, Z$ occur 2 times. So the distribution of the key-stream characters x is

$$X = (3/64, 3/64, \ldots, 3/64, 2/64, 2/64, \ldots, 2/64),$$

where $3/64$ is the probability of $A, B, \ldots, L$ and $2/64 = 0.031$ is the probability of $M, N, \ldots, Z$. We compute the cipher-text characters distribution now

$$(\ .039,\ .037,\ .037,\ .037,\ .037,\ .037,\ .036,\ .038,\ .038,\ .038, .038,\ .038, .038,$$
$$.038,\ .038,\ .040,\ .040,\ .040,\ .039,\ .038,\ .037,\ .037,\ .039,\ .039,\ .039,\ .037\ )$$

This is the dot-product of $X = (\mathbf{Pr}(x = 0), \ldots, \mathbf{Pr}(x = 25))$ and shifted English characters distribution $q_0, q_1, \ldots, q_{25}$ for all 26 shifts. That is

$$\begin{aligned} \mathbf{Pr}(c = j) &= \sum_{i=0}^{25} \mathbf{Pr}(x = i)\mathbf{Pr}(c = j | x = i) \\ &= \sum_{i=0}^{25} \mathbf{Pr}(x = i)\mathbf{Pr}(m = i - j) = \sum_{i=0}^{25} \mathbf{Pr}(x = i) q_{i-j} \end{aligned}$$

because $c = x - m$. Let now 17-teeth wheel be in a fixed position, where "the pin is retracted". So the distribution of the key-stream characters is

$$(3/32, 3/32, \ldots, 3/32, 2/32, 0, \ldots, 0),$$

where $3/32$ is the probability of $A, B, \ldots, J$ and $2/32$ is the probability of K, and all other characters have zero probabilities. So the cipher-text characters distribution is

$$(\ .042,\ .042,\ .042,\ .035,\ .033,\ .032,\ .030,\ .031,\ .034,\ .037, .034,\ .031, .037,$$
$$.043,\ .044,\ .047,\ .045,\ .040,\ .039,\ .038,\ .035,\ .036,\ .044,\ .040,\ .037,\ .036).$$

Similarly, if the 17-teeth wheel is in a fixed position, where "the pin is extended", then the distribution of the key-stream characters is

$$X = (0, \ldots, 0, 1/32, 3/32, 2/32 \ldots, 2/32),$$

where the probability of $A, B, \ldots, J$ is zero, the probability of K is $1/32$, the probability of L is $3/32$ and the probability of $M, N, \ldots, Z$ is $2/32$. So the cipher-text characters distribution is

$$(.035,\ .034,\ .035,\ .040,\ .041,\ .040,\ .042,\ .044,\ .040,\ .036, .040,\ .043, .037,$$
$$.033,\ .032,\ .031,\ .032,\ .037,\ .039,\ .039,\ .040,\ .038,\ .033,\ .037,\ .039,\ .038).$$

One can easily distinct the last two distributions given some amount of the cipher-text.

To split the distributions the following rule may be used. Assume the two distributions $R = (r_0, r_1, \ldots, r_{25})$ and $S = (s_0, s_1, \ldots, s_{25})$ are given and the distribution $T = (t_0, t_1, \ldots, t_{25})$ of characters in some sample sequence is observed. The Euclidean lengths

$$|R - T| = (\sum_{i=0}^{25} (r_i - t_i)^2)^{1/2} \quad \text{and} \quad |S - T| = (\sum_{i=0}^{25} (s_i - t_i)^2)^{1/2}$$

of the vectors $R - T$ and $S - T$ are computed. If $|R - T|$ is close to 0, then it is accepted that the sample was produced with R-distribution. If $|S - T|$ is close to 0 , then it is accepted that the sample was produced with S-distribution. Some errors are possible but their probabilities are decreasing as the amount of the available characters is growing.

By using Euclidean distances in $\mathbf{R}^{26}$, we split the 17 distributions defined by $C(1), \ldots, C(17)$ into two groups. Therefore, the last wheel positions are split into two groups, where the pin was extended(retracted) for all positions in each group. One does the same for all wheels, and there are only two variants for pin distribution on each wheel. We come up with $2^6 = 64$ guesses on the state of pins in all 6 wheels.

How do we check the variants? For each guess we produce a sequence of 6-bit groups. We fix one group $y = y_1, \ldots, y_6$. Let y occur in the positions $i_1, \ldots, i_j, \ldots$ in the stream of 6-bit groups. If the guess was right, then the key-stream character in these positions is the same and equal x, though it is unknown.

The cipher-text characters are there defined as $c_{i_j} \equiv x - m_{i_j} \mod 26$. Therefore if our guess was right, then the distribution of c_{i_j} is a shift of the distribution of the characters $-m_{i_j} \mod 26$, where m_{i_j} are plain-text characters. If the latter is wrong for every shift, then the guess was wrong. If this is right for some shift x, then the guess was right and one simultaneously finds the cage value at the address y.

We use this as a criterion for rejecting wrong guesses. It should work well if the cipher-text is long enough. Having determined the key-wheels, similarly we determine the cage. As above one should simply find correct shift $x = x(y)$ in $c_{i_j} \equiv x - m_{i_j} \mod 26$ for each 6-bit group y.

On the whole it requires no more than 2000 cipher-text characters by experiments. The known plain-text attack is much easier. It succeeds with only about 100 characters.

1.9 Cryptanalysis at Depth and Running Key Cipher

Consider a stream cipher, like a Hagelin machine, generating a key-stream $\{x_i\}$, where the cipher-text $\{c_i\}$ is being produced as

$$c_i \equiv x_i - m_i \mod 26 \tag{5}$$

for the plain-text $\{m_i\}$. Let several plain-texts

$$\{m_i^1\}, \{m_i^2\}, \ldots, \{m_i^s\}$$

be encrypted with the same key-stream $\{x_i\}$. Then so called at-depth cryptanalysis is applicable. The depth is the number of the plain-texts produced with the same key-stream. The key-stream characters are eliminated using (5) to get new equations

$$c_i^k - c_i^l = m_i^l - m_i^k$$

for any $1 \leq k < l \leq s$. These equations don't comprise unknown key-stream characters. Then the knowledge of one plain-text in some positions reveals characters of other plain-texts in the same positions. The other characters of the latter are recovered by guessing sensible extensions. The guess is used to find new portions of plain-texts and a particular extension is rejected if they are not sensible. But how do we find the initial portion of a plain-text? One can try a probable word, one which probably occur in the plain-text. In military communications for example the words "BATTALION","COLONEL", "GENERALSTAFF" etc. are probable. Lacking even this, the cryptanalyst can use common words or groups of characters "OFTHE","TION","WHICH". In order to proceed one tries a probable word in all possible locations. If the word is present, the method is a success when tried at the proper location. Some mistakes may occur, especially with short probable words. But they will be detected as analysis proceeds since they won't have sensible extensions. The cryptanalysis may be complicated if the depth is small, like 2. The practice shows that the three cipher-texts are well enough to recover the plain-texts.

We consider an example. Let M_1, M_2 and M_3 be three plain-texts of length $363, 249$ and 234 encrypted with the same key-stream X of length 363:

gcjjqhxjhzycswnjqhgipksirmcyotpnhkbnuhwrbyiarczhzzoolirespk
jvcagrsifurxjjvghhacrlpkuunulwlkwizasawwsufggqqjoecdhytimwi
mctcvdnptvknohzhmdwewzxbrtqzffeltifduyqqdvsqcvprvovjabkrizu
uzprvmjpztgkpwtrawalijrluauznqlehhqfhbrkjppolhtmokjuqtgcham
clpyfqbzzefkgqirufzyxmlcsclauldedgoerzxvnzmgybwutzdlsiferip
yhuwptqhqywfulllhfuuypdhvmruigenjxthfssyvhwquxpbxmhaaiqplae
lexfxdgpv

rgqoxvadfmlmlmeuvwwahlhirawfugfgxztgxwwdoplbvrdxqyfgygnepsv
fwhrexkcustmkuxcdltlvvfqtqvibwloleptacpqwptgaijopfylezrpnvv
pqtfmuvxunozdjeudjqbhvyqpbttkuabglizjmudbicqyhzucsvzkfvzqv
xckqxlvbpgtvvprzwpaicmdrwwojcesorrkisrddianmphpfpykrzqmlqcn
hryjkuhwbqufxs

$hfojmautbncxdendfpomwusfrwrxjfpwxooktegzoystvjtofvkrengnmboz$
$knvftepdaptyuijpobnzfmxsgygfohjvetlpkyzhliggrqtgokmfzwfabzag$
$vnzzmiotuoqmudeufmlwqfsbuwiowezbalzmbutwmaqoqlsbnzqdrvysyrr$
$ltctadukrahxaxnmgzhinprysqmzthmrreisizzxlatxfkqahqzigd$

For simplicity we use the notation $C_i \equiv X - M_i$. So one computes

$$M_{31} \equiv C_1 - C_3 \equiv M_3 - M_1$$
$$M_{32} \equiv C_2 - C_3 \equiv M_3 - M_2$$

Then M_{31} of 234 characters and M_{32} of 249 characters are

$zxvaehdqgmwfpsaglsswtqadaqlbfoarkwndbdqsnaqhwtgtueexhvlrgowklpf$
$byotcucelpnxstzpsgdncopogiebbegpdqyxljxaazaqiqsrczxdmvjmwyaieodlhr$
$tarnnjzczkoblmbpfjxqppjedjzepjcwecfbbkwuwblhanbcwiygckqpwzwtppw$
$rdjofjcjyfjwhxemlavzodtzckqsdlowheujnauywe$

$kbcflvgkezjpiirrqhiolrpdaefilbqkalfwesqeartiaikjldvputhrdrhgmuwzegnr$
$setmaptoxsywqttbkxcwiefqawilsrrpelaurtvjrozzavknuwpkydzvwlmzvfpts$
$aafwwvkcakjszommfcmhzynichpgeyrleraumlpsivxfetzmstipdzlepkcwsnw$
$wnwjjhxlkogpykyrbnvlabopehwikoursandkz$

We know that the messages are something introductory on cryptography. So there should be patterns like "CRYPTOSYSTEM","THE DISCRETE LOGARITHM PROBLEM", "PUBLIC KEY CRYPTOGRAPHY", or "SYMMETRIC KEY CRYPTOGRAPHY" etc. Let we try G="THE DISCRETE LOGARITHM PROBLEM" as a probable word in the third message. At the first place of the latter we get

$$\begin{aligned} &THEDISCRETELOGARITHMPROBLEM \\ -&zxvaehdqgmwfpsaglsswtqadaql \equiv \\ &ukjdelzbyhigzoalxbpqwboylob \end{aligned}$$

for M_1 and

$$\begin{aligned} &THEDISCRETELOGARITHMPROBLEM \\ -&kbcflvgkezjpiirrqhiolrpdaef \equiv \\ &jgcyxxwhauvwgyjasmzyeazylah \end{aligned}$$

for M_2. They are not sensible. One should try another places in the third message. Finally, at the 35-th place we find M_1, M_2 and M_3 are

$GEDASAPROMISINGNEWAREAINPUB$
$OLALSOCALLEDEXPONENTIALKEYA$
$THEDISCRETELOGARITHMPROBLEM.$

The first plain-text is extended to get the word "PUBLIC" and the pattern "AGRE" in the second plain-text, which obviously is extended to "AGREEMENT". That is a right guess because the third plain-text gets a sensible extension $ANDISREL$ and the first gets $LICKEYCR$ which continues to $LICKEYCRYPTOGRAPHY$. In so doing we find out the rest of the plain-texts to the right. The extension to the left is usually more complicated, but still possible. We guess that OL in the second plain-text is the end of $PROTOCOL$. This looks as a right guess for the third plain-text extends by $ASEDON$ which is a part of $BASEDON$ and the first plain-text gets an extension $AVEEMER$, a part of a sensible pattern $HAVEEMER$ etc. If the process stops one needs more guesses or probable words.

Hagelin machine is a device for generating character sequence, called key-stream, with long periods. But such sequences may be difficult to implement by hand which is sometimes important for spies. Hand cipher using long aperiodic characters sequences are historically called running key ciphers. The key for such a cipher is typically the name of a readily available book, together with a page, line and column number. To encrypt a message one writes it and the text from the book one above the other without spaces, punctuation marks and adds(subtracts) them modulo 26. Decryption is carried out by subtracting(adding) the key from the cipher-text modulo 26.

The cipher was generally solved in the late 1890's. Similar to the at-depth cryptanalysis the solution is based on probable words. These words may not only come from the general content of transmitted information but also from the book used in production of the key-stream. Really, the cryptanalyst has a sum(difference) of two plain-texts. Then trying a probable word in all possible locations he gets some extension of the word or reject the current location. In practice it might be sometimes complicated to make good extensions, as the depth is only 2. Then more guesses or probable words are necessary.

Anyway, the cipher was broken is due to the fact that English is about 75-percent redundant. That is 4.7 bits are on the average necessary to represent one character. But one character only represents 1.2 bits of English text information. The fact that running key ciphers can be solved proves that English is at least 50-percent redundant, since two texts(the plain-text and the running key) can be recovered from a single cipher-text of the same length. This implies that the cipher may be got stronger after several successive encryptions with different running keys. Four encryptions would be secure against all attacks. It would be impossible to recover the plain-text since, by symmetry, it could also be possible to recover the four running keys, which would imply that English is at least 80-percent redundant.

2 Stream Ciphers

There are two main types of stream ciphers. Synchronous and self-synchronizing. The encryption-decryption process of a synchronous stream cipher is described by the picture where k is a key, $S_0, S_1, \ldots$ are the sequence of internal states of

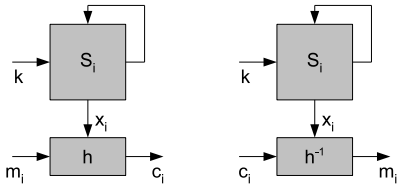

Figure 6: Synchronous stream cipher encryption-decryption.

the cipher, $x_0, x_1, \ldots$ the key-stream sequence and

$$S_{i+1} = f(S_i, k), \quad x_i = g(S_i, k), \quad c_i = h(x_i, m_i)$$

and $c_i = h^{-1}(x_i, m_i)$ for the decryption. In order to make the cipher work an initial state S_0 should be define. There are two possibilities:

1. S_0 is public,
2. S_0 is secret, e.g. it depends on the key k.

In order to avoid at-depth cryptanalysis a new initial state S_0 should be taken for a new message. The goal is to make the key-stream significantly different. Then S_0 is sent before the cipher-text in clear. The initial state is usually called the initial value and denoted IV. When S_0 is secret, that is depends on the key, it is better to make S_0 be dependent on some public IV, which changes from one message to another and is transmitted before the cipher-text. Common practical solution is that an IV and the key are introduced into the state. The cipher does then a number of clocks without producing any key-stream. All bits of S_0 now depend on all key bits and the bits of the IV.

We consider some properties of synchronous stream cipher.

1. for proper decryption the sender and receiver should be synchronized. This means that they should have the same key as well as the same initial state. The decryption fails if the synchronization is lost, which is possible when some digits(characters) of the cipher-text were deleted or inserted. Then the encryption-decryption is to be re-initialized, that is the plain-text should be encrypted with a new IV. The fact that the synchronization is lost can be detected when the message is naturally redundant, as a text in some language. Otherwise, special markers could be placed in the cipher-text in order to detect insertions and deletions.

2. The cipher is without error propagation. If a cipher-text digit(character)was modified during transmission, it does not affect the decryption or other cipher-text digits.

3. Some additional measures should be taken in order to protect data integrity, because an active adversary is able to change some digits without being detected.

4. If the adversary knows the plain-text for a given cipher-text, he finds the key-stream, which doesn't depend on the plain-text. In the case of some stream ciphers it may be used to find out the key with some correlation or algebraic attacks.

   Most of synchronous stream ciphers are binary additive stream ciphers in which key-stream, plain-text and cipher-text are binary digits and the output function h is an XOR function.

A self-synchronizing stream cipher is one, where the key-stream is generated as a function of the key and a fixed number of previous digits of the cipher-text, see Figure 7. The internal state is $S_i = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1})$, and $x_i =$

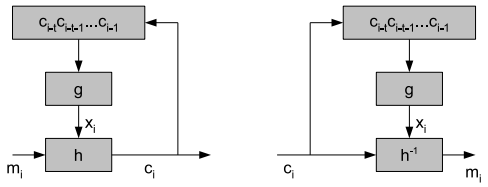

Figure 7: Self-synchronizing stream cipher encryption-decryption.

$g(S_i, k)$, $c_i = h(x_i, m_i)$, and x_i is a key-stream and c_i are cipher-text digits, where $m_i = h^{-1}(x_i, c_i)$ for decryption. Properties of self-synchronizing stream ciphers are:

1. Self-synchronization is restoring if even some ciphertext digits have been inserted or deleted. After such alterations of the cipher-text the next t digits won't be decrypted properly. But the next digits will be decrypted correctly.

2. One error in the channel results in improper decrypting only t next digits.

3. One should use some special technique to provide data integrity, because an active adversary is able to replay some segments of cipher-text and this can't be detected.

4. The key-stream depends on the plain-text. It makes it difficult to apply correlation attacks.

2.1 Time-Memory TradeOff

We consider a synchronous stream cipher with binary keystream, where the functions f and h don't depend on the key k while the initial state S_0 is determined by k as well as by an initial value IV. Let the bit-size of S_i be t and the key bit-size is n. Assume also that the sequence of the states is cyclic of period $\approx 2^t$. Any state is uniquely determined by first t bits of the key-stream produced with starting with that state. Finally, assume the cryptanalyst knows a segment of the key-stream bits

$$x_0, x_1, \ldots, x_{T-1}$$

of length $T \geq m + t - 1$, where $m = \lceil \sqrt{T} \rceil$. We will show that he finds the initial state in $O(t2^{t/2})$ operations with binary t-strings.

1. Define t-strings $X_j = (x_j, x_{j+1}, \ldots, x_{j+t-1})$ for $0 \leq j \leq m-1$ and put into the memory pairs (j, X_j).

2. Sort the pairs to the second coordinate.

3. For a randomly chosen initial state S, compute the key-stream segment X of length t. In the sorted set of pairs find (j, X_j), where $X_j = X$. Then $S = S_j$, so S_0 is found by moving back from S_j to S_0. If such a pair can't be found, try the step again.

The complexity of sorting is $O(m \log m)$ operations with binary t-strings. The probability that $X = X_j$ for some $j = 0, \ldots, m-1$ is $m/2^t$. So the average number of trials before we find such a pair is $2^t/m$. One should compute at most m states to go from S_j to S_0. On the whole the complexity of the method is

$$O(m \log m + (2^t/m) \log m + m) = O(t2^{t/2})$$

operations with binary t-strings. The cryptanalyst is now able to generate more key-stream $x_T, x_{T+1}, \ldots$ and so decrypt some new cipher-text. Alternatively, one tries all 2^n keys one after the other and decrypts some cipher-text too. If $n > t/2$, the first approach is preferable. This is the reason why in stream ciphers the key length n is being taken $\leq t/2$. For instance, in the popular cipher "Trivium" the state length is 288, while the key size is 80 bits. In many cases, when the state size is big, this tradeoff is impractical as the cryptanalyst is supposed to have exceedingly large amount of the key-stream.

2.2 Linear Feedback Shift Registers

In order to construct strong stream ciphers there is a need for sequences of long periods, otherwise, if short-period sequences(key-streams) are used, at-depth cryptanalysis is applicable. Linear Feedback Shift Registers are able to produce long-period sequences at a very low cost.

Let $c_1, c_2, \ldots, c_L$ be a binary L-string. Given an initial vector $s_{L-1}, s_{L-2}, \ldots, s_0$ one produces a recurrent sequence $s = s_0, s_1, \ldots$ by the rule

$$s_j \equiv c_1 s_{j-1} + c_2 s_{j-2} + \ldots + c_L s_{j-L} \bmod 2$$

for all $j \geq L$. To simplify notation we will use $=$ instead of $\equiv$ and omit $\bmod 2$. In other notation the latter congruence may be represented by

$$s_j = (c_1, c_2, \ldots, c_L)(s_{j-1}, s_{j-2}, \ldots, s_{j-L})^T.$$

For example, let $L = 4$ and

$$\begin{aligned} s_3, s_2, s_1, s_0 &= 0, 1, 1, 0, \\ c_1, c_2, c_3, c_4 &= 1, 0, 0, 1. \end{aligned}$$

Then

$$\begin{aligned} s_4 &= (1, 0, 0, 1)(0, 1, 1, 0)^T = 0, \\ s_5 &= (1, 0, 0, 1)(0, 0, 1, 1)^T = 1, \\ &\ldots \end{aligned}$$

That is

$$s = 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, \ldots$$

The process of generating this sequence may be implemented by using a linear feedback shift register. This is a device comprising L delay elements numbered $0, 1, 2, \ldots, L-1$ and capable of storing one bit. They are connected in a chain with a linear feedback. For example, the above sequence s is generated by the LFSR in Figure 2.2.2 where $(0, 1, 1, 0), (0, 0, 1, 1), (1, 0, 0, 1), \ldots$ is a sequence of

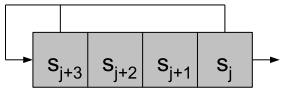

Figure 8: LFSR of length 4.

the internal states of the register.

The LFSR is denoted $< L, C(X) >$, where $C(X) = 1 + c_1 X + \ldots + c_L X^L$ is the connection polynomial. The LFSR is called nonsingular if the degree of the connection polynomial is just L, or equally $c_L = 1$, and $s_{L-1}, s_{L-2}, \ldots, s_0$ is called the initial state of the LFSR. We will mostly characterize the LFSR by a characteristic polynomial $f(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ which related to the connection polynomial as $f(X) = C(1/X)X^L$. So the LFSR is singular if $c_L = 0$.

Figure 9: LFSR.

Example of a singular LFSR, that is $\deg C(X) < L$ or $f(0) = 0$. Let us take $f(X) = X^4 + X^3 + X^2$ and the string 1100 as an initial state, then the output sequence $s = 0011011011\ldots$ is aperiodic. It is of cause ultimately periodic with the minimal period 3 and the tail of length 1. The sequence of states

$$(1100), (0110), (1101), (1011), (0110), \ldots$$

is aperiodic too with the same minimal period and the tail length. This is a general fact. Namely,

Problem 4 *Let $s = s_0, \ldots, s_{l-1}, s_l, \ldots, s_{l+t-1}, s_l, \ldots$ be a sequence generated by a LFSR, where l is the tail length and t is its minimal period. Let $S = S_0, \ldots, S_{l_1-1}, S_{l_1}, \ldots, S_{l_1+t_1-1}, S_{l_1}, \ldots$ be a sequence of the related LFSR states, where l_1 is the tail length and t_1 is the minimal period. Prove that $l = l_1$ and $t = t_1$*

Problem 4 implies

Problem 5
1. *Let an aperiodic sequence s be generated by a LFSR. Prove that the LFSR is singular.*
2. *Let LFSR be singular. Prove that the LFSR can generate an aperiodic sequence for some initial state.*

In other words, an LFSR is nonsingular if and only if any generated infinite sequence is pure periodic.

Sometimes it is more convenient to work with matrices instead LFSRs. Let

$f(X) = X^L + c_1X^{L-1} + \ldots + c_L$ be a polynomial, then the matrix

$$\begin{pmatrix} 0 & 0 & 0 & c_L \\ 1 & 0 & 0 & c_{L-1} \\ 0 & 1 & 0 & c_{L-2} \\ \ldots & & & \\ 0 & 0 & 1 & c_1 \end{pmatrix} \tag{6}$$

is called a companion matrix to $f(X)$. Let

$$z_0 = (s_0, s_1, \ldots, s_{L-1}), \ldots, z_i = (s_i, s_{i+1}, \ldots, s_{L+i-1}), \ldots,$$

be vectors produced from the sequence $s = s_0, s_1, \ldots$ generated by a LFSR of length L with the characteristic polynomial $f(X)$. They are the states of the register written backwards. Then $z_iA = z_{i+1}$ that is

$$(s_i, s_{i+1}, \ldots, s_{L+i-1}) \begin{pmatrix} 0 & 0 & 0 & c_L \\ 1 & 0 & 0 & c_{L-1} \\ 0 & 1 & 0 & c_{L-2} \\ \ldots & & & \\ 0 & 0 & 1 & c_1 \end{pmatrix} = (s_{i+1}, s_{i+2}, \ldots, s_{L+i}).$$

We now know that the period of a sequence generated by a LFSR is equal to the period of its states. So s is pure periodic of period N if and only if $z_0A^N = z_0$.

We will remind some notion from Linear Algebra.

1. The characteristic polynomial $f_A(X)$ of a square matrix A is

   $$f_A(X) = \det(XE + A) = X^L + c_1X^{L-1} + \ldots + c_L,$$

   where E is the unity matrix of the same size as A. That is $f_A = f$ is the characteristic polynomial of A. The characteristic polynomial has the property $f(A) = 0$. That is

   $$A^L + c_1A^{L-1} + \ldots + c_LE = 0,$$

   where E is the unity matrix of size $L \times L$.

2. The minimal polynomial of any square matrix A is a nonzero polynomial $m_A(X)$ of smallest degree such that $m_A(A) = 0$. The main property of the minimal polynomial is if $g(A) = 0$ for some polynomial $g(X)$, then $m_A(X)$ divides $g(X)$. In particular m_A divides f_A.

3. Let z be any nonzero L-string. By $m_{z,A}(X)$ we denote a nonzero polynomial of smallest degree such that

   $$z\, m_{z,A}(A) = 0.$$

   This polynomial is called minimal polynomial of z in respect with A. One sees $m_{z,A}$ divides any polynomial $g(X)$ such that $zg(A) = 0$. In particular, $m_{z,A}$ divides m_A and therefore f_A.

Example. Let $A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. Then $f_A = \det(XE + A) =$

$$\det\left(\begin{pmatrix} X & 0 \\ 0 & X \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\right) = \det\left(\begin{pmatrix} X+1 & 0 \\ 1 & X+1 \end{pmatrix}\right) = (X+1)^2.$$

There are 3 possibilities for the minimal polynomial: $1, X+1, (X+1)^2$. It is easy to see $m_A = (X+1)^2$. Take $z = (1,0)$, then $m_{z,A} = X+1$ because

$$z(A+E) = (1,0)\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = (0,0).$$

Let $f(X)$ be a polynomial in $GF(2)[X]$. Then $f(X)$ is called reducible if $f(X) = f_1(X)f_2(X)$, where $\deg f_1, \deg f_2 < \deg f$. Otherwise it is called irreducible. The minimal natural N such that an irreducible polynomial $f(X)$ divides X^N+1 is called the period of $f(X)$.

Theorem 1 *Let A be any matrix of size $L \times L$. Then*

1. *If f_A irreducible, then $m_{z,A} = m_A = f_A$ for any non-zero z.*
2. *Let $f(X)$ be an irreducible polynomial of degree L and of period N. Then the LFSR with generating polynomial f and any nonzero initial state generates a pure periodic sequence of the minimal period N.*
3. *Let $f(X)$ be an irreducible polynomial of degree L and of period N. Then N divides $2^L - 1$ and $f(X)$ divides $X^{2^L-1} + 1$.*

Proof. The proof of the first statement is obvious, because $m_{z,A}$ divides m_A and m_A divides f_A. A non-zero z generates a pure periodic sequence of period t if and only if $zA^t = z$, where A stands for the companion matrix to f. Equivalently, $z(A^t + E) = 0$. That is true if and only if $m_{z,A} = f_A$ divides $X^t + 1$. Therefore, t is a minimal period if and only if $t = N$.

In order to prove the last statement we realize that there are $2^L - 1$ nonzero strings z. This set is partitioned into subsets of size N:

$$\{z_1, z_1A, \ldots, z_1A^{N-1}\}, \{z_2, z_2A, \ldots, z_2A^{N-1}\}, \ldots.$$

If some of these sets intersect, then they coincide. We have partitioned the set of size $2^L - 1$ into equal subsets of size N each. So N divides $2^L - 1$. Therefore, $X^N + 1$ divides $X^{2^L-1} + 1$ because

$$X^{2^L-1} + 1 = (X^{(m-1)N} + X^{(m-2)N} + \ldots + 1)(X^N + 1),$$

where $2^L - 1 = mN$. Then f divides $X^{2^L-1} + 1$. The Theorem is proved.

Definition 2 *An irreducible polynomial f of degree L is called primitive if its period is $2^L - 1$.*

2.2.1 Irreducible and Primitive polynomials

In order to construct a sequence of a maximal period with an LFSR of length L one should use a primitive polynomial of degree L as generating. We will answer how to construct an irreducible polynomial or, more generally, to factor polynomials and then how to find a primitive polynomial.

Theorem 2 *Let $f(X)$ be a polynomial of degree L. Then $f(X)$ is irreducible if and only if*

$$\gcd(f, X^{2^s} + X) = 1$$

for all s such that $1 \leq s \leq L/2$.

Proof. We will formulate a lemma first.

Lemma 1 *The polynomial $X^{2^s} + X$ is a product of all irreducible polynomials of degree t, where $1 \leq t \leq s$ and t divides s.*

We won't prove the Lemma but we give an example.

$$\begin{aligned} X^2 + X &= X(X+1), \\ X^{2^2} + X &= X(X+1)(X^2+X+1), \\ X^{2^3} + X &= X(X+1)(X^3+X+1)(X^3+X^2+1). \end{aligned}$$

In order to prove the Theorem we assume f irreducible and

$$\gcd(f, X^{2^s} + X) = g(X) \neq 1,$$

where $1 \leq s \leq L/2$. That is g of degree ≥ 1. So g has an irreducible factor $h(X)$, where $\deg h \leq L/2$. Therefore, h divides f, which is a contradiction. Vice verse, let

$$\gcd(f, X^{2^s} + X) = 1$$

for all s such that $1 \leq s \leq L/2$ and f is reducible. Then $f = g_1(X)g_2(X)$ is a decomposition of f, where the degree of g_1 and g_2 are less than L. At least one of these degrees, for example $\deg g_1$, should be less or equal to $L/2$. Let $h(X)$ be its irreducible factor. So $1 \leq \deg h = s \leq L/2$. Then h divides $X^{2^s} + X$. As h also divides f, we get a contradiction with $\gcd(f, X^{2^s} + X) = 1$. This proves the Theorem.

Obviously, the method may be used for factoring some polynomials. For example, let $f(X) = X^5 + X + 1$. We find

$$\begin{aligned} \gcd(f, X^2 + X) &= 1, \\ \gcd(f, X^{2^2} + X) &= X^2 + X + 1. \end{aligned}$$

That is $f = (X^2 + X + 1)(X^3 + X^2 + 1)$. But the method does not generally work. For example, $f(X) = X^6 + X^5 + X^4 + X^3 + X^2 + X + 1$ is a reducible

polynomial because

$$\gcd(f, X^2 + X) = 1,$$
$$\gcd(f, X^{2^2} + X) = 1,$$
$$\gcd(f, X^{2^3} + X) = f,$$

but the gcds don't produce any of its factors. For a large degree L the computations may be tedious as the degree of $X^{2^s} + X$ can be about $2^{L/2}$. In this case we use $\gcd(f, g) = \gcd(f, g \mod f)$ and $X^{2^s} + X \equiv (X^{2^{s-1}} + X)^2 + (X^2 + X) \mod f$. E.g.

$$\gcd(X^6 + X + 1, X^{2^3} + X) = \gcd(X^6 + X + 1, X^3 + X^2 + X) = 1$$

The Theorem implies a simple randomized method to construct an irreducible polynomial of degree L.

1. Generate random polynomial $f(X) = X^L + c_1 X^{L-1} \ldots + c_L$. There should be $c_L = 1$ and $c_1 + \ldots + c_L = 0$, otherwise X or $X + 1$ are its factors.

2. For s from 1 to $L/2$ compute $g_s \equiv X^{2^s} + X \mod f$ and $\gcd(f, g_s)$. If all these gcds are 1, then f is irreducible.

There is another method for checking irreducibility, which may also be used in order to factor polynomials. We consider how to construct an irreducible polynomial first. It is based on Berlekamp test.

1. Generate random polynomial $f(X) = X^L + c_1 X^{L-1} \ldots + c_L$. There should be $c_L = 1$ and $c_1 + \ldots + c_L = 0$, otherwise X or $X + 1$ are its factors.

2. Compute $\gcd(f', f)$(gcd is 1 if and only if f is without multiple roots), where f' is a formal derivative of f, that is

$$f' = LX^{L-1} + c_1(L-1)X^{L-2} \ldots + c_{L-1}.$$

   If the *gcd* is not 1, then the polynomial is reducible and one repeats from the first stage. Otherwise go on to

3. (Berlekamp test.) Consider the equation $g^2 + g = 0 \mod f$ in polynomials g of degree $< \deg f$. We prove that is equivalent to a system of linear equations. Let $g(X) = \sum_{i=0}^{L-1} y_i X^i$, where $y_0, y_1, \ldots, y_{L-1}$ are variables taking $0, 1$ values. Let

$$X^{2i} = \sum_{j=0}^{L-1} b_{ij} X^j \mod f(X).$$

Then

$$\begin{aligned} g^2+g &= \left(\sum_{i=0}^{L-1} y_i X^i\right)^2 + \sum_{i=0}^{L-1} y_i X^i \\ &= \sum_{i=0}^{L-1} y_i X^{2i} + \sum_{i=0}^{L-1} y_i X^i = \sum_{j=0}^{L-1} (\sum_{i=0}^{L-1} y_i b_{ij}) X^j + \sum_{j=0}^{L-1} y_j X^j \\ &= \sum_{j=0}^{L-1} (y_j + \sum_{i=0}^{L-1} y_i b_{ij}) X^j = 0 \mod f(X) \end{aligned}$$

if and only if $y_j + \sum_{i=0}^{L-1} y_i b_{ij} = 0$ for $0 \le j \le L-1$. In matrix form:

$$(y_0, \ldots, y_{L-1})\,(B+E) = (0, \ldots, 0),$$

where $B = (b_{ij})_{0 \le i,j \le L-1}$ is a matrix of size $L \times L$. The system has at least two solution $1, 0, \ldots, 0$ and $0, 0, \ldots, 0$. The polynomial f is irreducible if and only if the system has exactly two solutions, that is $\mathrm{rank}(B+E) = L-1$.

Example. Let $f = X^4 + X^3 + 1$ be a random polynomial of degree 4. Compute $f' = X^2$ and $\gcd(f, f') = 1$. So the polynomial is without multiple roots. The following congruences modulo f are true:

$$\begin{aligned} (X^0)^2 &\equiv 1, \\ (X^1)^2 &\equiv X^2, \\ (X^2)^2 &\equiv X^3 + 1, \\ (X^3)^2 &\equiv X^3 + X^2 + X + 1. \end{aligned}$$

The rows of the matrix B are the coefficients of the above polynomials. So the matrix

$$B+E = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} + \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0. \end{matrix}$$

is the matrix of the system of equations (7). It is of rank 3, the system has exactly two solutions. Therefore, the polynomial is irreducible. The method is based on the following Theorem

Theorem 3 *Let $f(X)$ be a polynomial of degree L without multiple roots. Consider the equation*

$$g^2 + g \equiv 0 (\mod f) \tag{7}$$

in polynomials $g(X)$ of degree $< L$. Then

1. *The polynomial f is irreducible if and only if $g = 0, 1$ are the only solutions to this equation.*

2. *If $g \neq 0, 1$ is a solution to (7), then $h = \gcd(f, g)$ is a nontrivial factor of f.*

Proof. Let f be reducible. As f is without multiple roots, $f = f_1 f_2$ is a product of its coprime factors $f_1 = f_1(X)$ and $f_2 = f_2(X)$ with $1 \leq f_i < L$. There exist two polynomials $u = u(X)$ and $v = v(X)$ such that $uf_1 + vf_2 = 1$ by a property of coprime polynomials. We put $g \equiv uf_1 \bmod f$. If $g = 0$ then f_2 divides u which is impossible. If $g = 1$ then f_1 divides v which is also impossible. Thus $g \neq 0, 1$. We see

$$g^2 + g = g(g+1) \equiv uf_1 vf_2 \equiv 0 \mod f$$

So we have constructed a nontrivial solution to (7).

Let $g \neq 0, 1$ be a solution to (7). Then $h = \gcd(f, g)$ differs from 1 and f. Really, if $h = 1$ then the fact that f divides $g(g+1)$ implies that f divides $g + 1$ and so $g = 1$ as $\deg g < \deg f$. If $h = f$ then f divides g and so $g = 0$ as $\deg g < \deg f$. The both cases are not possible by assumptions. So f is reducible and h is its factor.

The Theorem is used to factor any polynomials. Let $f(X) = X^6 + X^5 + X^4 + X^3 + X^2 + X + 1$. We represent the equation (7) as a system of linear equations as above. In order to write it down one computes

$$1^2 \equiv 1, X^2 \equiv X^2, X^4 \equiv X^4, X^6 \equiv X^5 + X^4 + X^3 + X^2 + X + 1,$$
$$X^8 \equiv X, X^{10} \equiv X^3$$

The coefficients of these polynomials represent rows of the matrix B. The matrix of the system is $B + E$. The system itself is

$$(y_0, y_1, \ldots, y_5) \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} = 0$$

One finds a nontrivial solution $(y_0, y_1, \ldots, y_5) = (0, 1, 1, 0, 1, 0)$ or $g = y_5 X^5 + y_4 X^4 + y_3 X^3 + y_2 X^2 + y_1 X + y_0 = X^4 + X^2 + X$ in a polynomial form. Then $\gcd(f, g) = X^3 + X + 1$. Thus the sought factorization is

$$X^6 + X^5 + X^4 + X^3 + X^2 + X + 1 = (X^3 + X + 1)(X^3 + X^2 + 1).$$

We will now answer how to construct a primitive polynomial of degree L. We should accept without proof here that it really exists.

1. Factor the number $2^L - 1 = \prod_i q_i^{l_i}$, where q_i are different prime numbers.
2. Take randomly an irreducible polynomial $f(X)$ of degree L.

3. For all i compute
$$X^{(2^L-1)/q_i} \mod f.$$
If all these residues don't equal 1, then f is primitive. Otherwise take another random irreducible f.

Problem 6 *Let $f(X)$ be an irreducible polynomial of degree L. Then f is primitive if and only if*
$$X^{(2^L-1)/q_i} \neq 1 \mod f$$
for all prime factors q_i of $2^L - 1$.

We will prove that $X^4 + X^3 + 1$ is primitive. We factor $2^4 - 1 = 3 \times 5$.
$$X^{\frac{2^4-1}{3}} = X^5 \equiv X^4 + X \equiv X^3 + X + 1 \neq 1,$$
$$X^{\frac{2^4-1}{5}} = X^3 \neq 1.$$
So f is primitive. For a large L the exponent $n = (2^L - 1)/q_i$ may be very big. So in order to compute $X^n \, mod \, f$ one uses the binary method.
$$n = 2^l + n_{l-1}2^{l-1} + \ldots + n_0,$$
$$X^n = (\ldots(X^2 X^{n_{l-1}})^2 \ldots)^2 X^{n_0},$$
where squarings and multiplications are done modulo f.

In case $2^L - 1$ is a prime number, called prime Mersenne number, any irreducible polynomial of degree L is automatically primitive. There are presently known 44 Mersenne prime numbers: $3, 7, 31, 127, \ldots$. The 39-digit number $2^{127} - 1$ is known to be prime since 1876. Therefore, for instance, the irreducible polynomial $X^{127} + X + 1$ is primitive. This is a big challenge in Number Theory to prove the conjecture that the number of prime Mersenne numbers is infinite.

Let the sequence s be produced with an LFSR of length L and of minimal period $2^L - 1$. So
$$s^{2^L-1} = s_0, s_1, \ldots, s_{2^L-2}$$
composes a period of s. Let N_0 be the number of 0 in s^{2^L-1} and N_1 be the number of 1.

Lemma 2 *$N_1 = 2^{L-1}$ and $N_0 = 2^{L-1} - 1$.*

Proof. Let's consider the sequence of internal states of the register:
$$z_i = (s_i, s_{i+1}, \ldots, s_{L+i-1}),$$
where i runs through $0, 1, \ldots, 2^L - 2$. As the register period is $2^L - 1$, they are all nonzero L-bit vectors. We count the number of 1 in all of them. That is
$$\sum_{i=1}^{L} i \binom{L}{i},$$

where $i\binom{L}{i}$ is the number of 1 in all vectors of weight i. Due to the identity

$$\sum_{i=1}^{L} i\binom{L}{i} x^{i-1} = L(1+x)^{L-1},$$

we have $\sum_{i=1}^{L} i\binom{L}{i} = L2^{L-1}$. Each digit of the sequence occurs L times in vectors z_i, so each 1 was counted L times in $L2^{L-1}$. Therefore, $N_1 = 2^{L-1}$ and $N_0 = 2^L - 1 - N_1 = 2^{L-1} - 1$. The Lemma is proved.

2.2.2 Linear Complexity

Let s^n denote a finite binary sequence of length n whose terms are $s_0, s_1, \ldots, s_{n-1}$.

Definition 3 *The linear complexity of a finite binary sequence $s^n \neq 0$(infinite ultimately periodic sequence s), denoted $L(s^n)$ ($L(s)$), is the length of the shortest LFSR that generates an infinite sequence having s^n as its first terms (generates s).*

If $s^n = 0, \ldots, 0$ then we put $L(s^n) = 0$ with a generating polynomial 1, equally the linear complexity of an infinite zero sequence is 0. In order to make simple the usage of Theorem 6 we also put $s^0 = \emptyset$ is generated by a constant polynomial 1 and its linear complexity is 0. Examples. We consider some simplest sequences

$$\begin{aligned} s^n &= 1, 0, 0, \ldots, 0, \text{then} \quad L(s^n) = 1, \\ s^n &= 1, 1, 0, \ldots, 0, \text{then} \quad L(s^n) = 2, \\ s^n &= 0, 1, 0, \ldots, 0, \text{then} \quad L(s^n) = 2, \end{aligned}$$

Problem 7 *Prove that $L(s^n) = n$ if and only if $s^n = 0, \ldots, 0, 1$.*

Let s^n be any binary sequence of length n and of linear complexity L, and $f(X)$ be a generating polynomial of s^n, where $\deg f = L$ is the length of the related LFSR. One uses f to produce infinite sequence s the first n bits of which are s^n:

$$s_0, s_1, \ldots, s_{n-1}, s_n = c_1 s_{n-1} + \ldots + c_L s_{n-L}, \ldots.$$

This infinite sequence has f as its generating polynomial.

We will answer the problem. Given $L(s^n)$ and $L(t^n)$ estimate $L(s^n + t^n)$ for two sequences s^n and t^n. For an infinite sequence s, consider shift subsequences: $s^{(o)} = s, s^{(1)}, s^{(2)}, \ldots$, where

$$\begin{aligned} s^{(0)} &= s_0, s_1, s_2, \ldots \\ s^{(1)} &= s_1, s_2, s_3, \ldots \\ &\ldots \\ s^{(i)} &= s_i, s_{i+1}, s_{i+2}, \ldots \\ &\ldots \end{aligned}$$

Let $h(X) = d_0 X^M + d_1 X^{M-1} + \ldots + d_M$ be any polynomial. One defines the sequence

$$h(s) = \sum_{i=0}^{M} d_{M-i} s^{(i)}.$$

In other words, $h(s) =$

$$d_M s_0 + d_{M-1} s_1 + \ldots + d_0 s_M, \ldots, d_M s_i + d_{M-1} s_{i+1} + \ldots + d_0 s_{i+M}, \ldots$$

For example, if $h = X$ then $X(s) = s_1, s_2, \ldots = s^{(1)}$ and $X(s^{(i)}) = s^{(i+1)}$. Therefore, the following Lemma holds

Lemma 3 *Let s be an infinite sequence. Then $h(X) \neq 0$ is a generating polynomial for s if and only if $h(s) = 0$, that is a zero sequence.*

Theorem 4 *1. Let s and t be any infinite binary sequences. Then $h(s+t) = h(s) + h(t)$ for any polynomial h.*

2. Let h and g be two polynomials. Then $(h + g)(s) = h(s) + g(s)$.

3. $(Xh)(s) = X(h(s))$ for any polynomial h.

4. $(hg)(s) = h(g(s))$.

Proof The proof of the first two statements is obvious. To prove the third statement one observes that

$$Xh(X) = \sum_{i=0}^{M} d_{M-i} X^{i+1}$$

and

$$(Xh)(s) = \sum_{i=0}^{M} d_{M-i} s^{(i+1)}.$$

On the other hand,

$$X(h(s)) = X(\sum_{i=0}^{M} d_{M-i} s^{(i)}) =$$
$$\sum_{i=0}^{M} d_{M-i} X(s^{(i)}) = \sum_{i=0}^{M} d_{M-i} s^{(i+1)}.$$

So $(Xh)(s) = X(h(s))$. The latter implies $(X^i h)(s) = X^i(h(s))$. To prove the last statement one sees

$$hg = d_0 X^M g + d_1 X^{M-1} g + \ldots + d_M g.$$

Therefore,

$$(hg)(s) = (\sum_{i=0}^{M} d_{M-i} X^i g)(s) =$$
$$\sum_{i=0}^{M} d_{M-i} (X^i g)(s) = (\sum_{i=0}^{M} d_{M-i} X^i) g(s) = h(g(s))$$

Corollary 1 *Let s and t be two infinite binary sequences with generating polynomials $g(X)$ and $f(X)$. Then a generating polynomial for $s + t$ is fg.*

Proof One sees that

$$(gf)(t+s) = (gf)(t) + (gf)(s) =$$
$$g(f(t)) + f(g(s)) = g(0) + f(0) = 0$$

The polynomial gf annihilates $t+s$, so it is a generating polynomial for $t+s$ by Lemma 3. For instance, we have two LFSRs with generating polynomials X^2+X+1 and X^3+X+1, and initial states 01 and 001 respectfully. The output of them is xored to get the sequence $1,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,1\ldots$. Its generating polynomial is

$$(X^2+X+1)(X^3+X+1) = X^5+X^4+1$$

and the following two devices produce the same sequence, see in Fig.10.

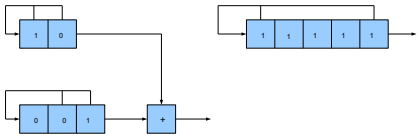

Figure 10: Two LFSRs with output xored

Corollary 2 *Let s^n and t^n be two binary sequences, then $L(s^n+t^n) \le L(s^n)+L(t^n)$.*

Proof Let $g(X)$ and $f(X)$ be generating polynomials for s^n and t^n, such that $\deg g = L(s^n)$ and $\deg f = L(t^n)$. Using these polynomials we expand s^n and t^n to infinite sequences s and t. From Corollary 1 one sees that gf is a generating polynomial for $s+t$ and so for s^n+t^n. Therefore, $L(s^n+t^n) \le \deg gf = L(s^n)+L(t^n)$.

Problem 8 *Given integer numbers L_1 and L_2, construct two LFSRs of length L_1, L_2 the sum of infinite output sequences of which is of linear complexity just L_1+L_2.*

Corollary 3 *Let s be an infinite binary sequences and $f(X)$ be its generating polynomial of a smallest degree $L = L(s) \ge 1$. Prove that f divides any other generating polynomial of s.*

Proof Let $g(X)$ be any generating polynomial of s. Let us divide $g = hf + g_1$, where $g_1 \ne 0$ and $0 \le \deg g_1 < L$ or $g_1 = 0$. In the first case $g_1(s) = (g + hf)(s) = g(s) + h(f(s)) = 0 + h(0) = 0$ which contradicts to the fact that f is of the smallest degree generating polynomial.

Definition 4 *Let $s = s_0, s_1, \ldots$ be a binary sequence and L_N denote the linear complexity of a subsequence $s = s_0, s_1, \ldots, s_{N-1}$. The sequence $L_1, L_2, \ldots$ is called the linear complexity profile of s.*

2.2.3 Berlekamp-Massey Algorithm

Example. Find the linear complexity profile of the sequence $s^{10} = 1001001111$. By trials we compute:

N	s^N	$f(X)$	L_N
0	$\emptyset$	1	0
1	1	$X, X+1$	1
2	10	X	1
3	100	X	1
4	1001	$X^3+1, X^3+X^2+1,$	3
		X^3+X+1, X^3+X^2+X+1	
5	10010	X^3+1, X^3+X+1	3
6	100100	X^3+1	3
7	1001001	X^3+1	3
8	10010011		
9	100100111		
10	1001001111		

We will need some theory in order to find the last three L_8, L_9, L_{10} as trials are not now very efficient.

Theorem 5 *1. Let $L(s^{2L-1}) = L$, then the matrix $S =$*

$$\begin{pmatrix} s_{L-1} & s_L & \dots & s_{2L-2} \\ \dots & & & \\ s_1 & s_2 & \dots & s_L \\ s_0 & s_1 & \dots & s_{L-1} \end{pmatrix}$$

is nonsingular(with non-zero determinant).

2. If $j > i$, then $L_j \geq L_i$.

3. If $L_N > N/2$, then $L_{N+1} = L_N$.

4. If $L_N \leq N/2$, then $L_{N+1} = N + 1 - L_N$.

Proof. We prove the first statement. Let $f(X) = X^L + c_1 X^{L-1} \dots + c_L$ be a generating polynomial for s^{2L-1} and the matrix S is singular. Then there is a nonzero solution a to the equation $aS = 0$, of shape $a = (0, \dots, 0, 1, a_1, \dots, a_t)$ for some binary $a_1, \dots, a_t$. So

$$\begin{aligned} s_t &= a_1 s_{t-1} + \dots + a_t s_0, \\ s_{t+1} &= a_1 s_t + \dots + a_t s_1, \\ &\dots \\ s_{t+L-1} &= a_1 s_{t+L-2} + \dots + a_t s_{L-1}, \end{aligned}$$

We multiply the first equation by c_L, the second by c_{L-1},... the last one by c_1 and sum the results. We get $s_{t+L} = a_1 s_{t+L-1} + \dots + a_t s_L$. Then we repeat the

step beginning with the second equation and so on. Finally, we get

$$\begin{aligned} s_{t+L} &= a_1 s_{t+L-1} + \ldots + a_t s_L \\ &\ldots \\ s_{2L-2} &= a_1 s_{2L-3} + \ldots + a_t s_{2L-t-2}. \end{aligned}$$

So $X^t + a_1 X^{t-1} + \ldots + a_t$ is a generating polynomial for s^{2L-1} and its degree is less than L. The contradiction proves the first statement. The second statement is obvious. The last two follow from the Berlekamp-Massey algorithm below.

Lemma 4 *Let $L_{N+1} + L_N \leq N$. Then $L_{N+1} = L_N$ and s^{N+1} is generated by f_N, so one puts $f_{N+1} = f_N$.*

Proof. Let $L = L_N$ and $L_1 = L_{N+1}$. Let s^N be generated by $f_N(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ and s^{N+1} be generated by $f_{N+1}(X) = X^{L_1} + c'_1 X^{L_1-1} + \ldots + c'_{L_1}$. Then

$$\begin{gathered} (c_1, \ldots, c_L)(s_{N-1}, \ldots, s_{N-L})^T = \\ (c_1, \ldots, c_L) \begin{pmatrix} s_{N-2} & \ldots & s_{N-L_1-1} \\ \ldots & & \\ s_{N-L-1} & \ldots & s_{N-L-L_1} \end{pmatrix} (c'_1, \ldots, c'_{L_1})^T = \\ (s_{N-1}, \ldots, s_{N-L})(c'_1, \ldots, c'_{L_1})^T = s_N, \end{gathered}$$

where the matrix is of size $L \times L_1$ and well defined as $N - L - L_1 \geq 0$. So we get $s_N = c_1 s_{N-1} + \ldots + c_L s_{N-L}$. That is f_N generates s^{N+1}.

Let s^N be generated by $f_N(X) = X^L + c_1 X^{L-1} + \ldots + c_L$. Then $f_N(X)$ generates s^{N+1} if and only if the N-th discrepancy

$$d_N = s_N + c_1 s_{N-1} + \ldots + c_L s_{N-L}$$

is zero.

Theorem 6 *Let $s^{N+1} = s_0, s_1, \ldots, s_N$ be given. Then*

1. *If $s^{N+1} = 0^{N+1}$, then $L_{N+1} = 0$ and $f_{N+1} = 1$. If $s^{N+1} = 0^N 1$, then $L_{N+1} = N + 1$ and f_{N+1} is any polynomial of degree $N + 1$. Otherwise:*
2. *let $d_N = 0$, then $L_{N+1} = L$(we denote $L = L_N$) and $f_{N+1} = f_N$.*
3. *Let $d_N = 1$, then*

$$L_{N+1} = \begin{cases} L, & L > N/2; \\ N + 1 - L, & L \leq N/2. \end{cases}$$

Let m be the largest integer $< N$ such that $L(s^m) < L = L(s^N)$. Then

$$f_{N+1} = \begin{cases} f_N + X^{2L-N-1} f_m, & \text{if } L > N/2; \\ X^{N-2L+1} f_N + f_m, & \text{if } L \leq N/2. \end{cases}$$

Remark 1 *Let s^0 be a sequence without members, $L(s^0) = 0$ and $f_0(X) = 1$.*

Proof. If $s^{N+1} = 0^{N+1}$ or $s^{N+1} = 0^N 1$, then the Theorem is true by the definition of linear complexity and Problem 7. We prove the left statements 2 and 3 by induction. Let's check them for $N = 1$. So we should examine two cases $s^2 = 10$ and 11. So

1. let $s^2 = 10$, then as $L_1 = 1$ and $f_1 = X$, one finds that $d_1 = 0 + 0 \times 1 = 0$, so $L_2 = 1$ and $f_2 = X$.
2. let $s^2 = 11$, then as $L_1 = 1$ and $f_1 = X$, one finds that $d_1 = 1 + 0 \times 1 = 1$, so $L_2 = 1$ and

$$f_2 = f_1 + X^{2\times 1-1-1} f_m = f_1 + 1 = X + 1,$$

where $m = 0$ and $f_0 = 1$, and $L_0 = 0$.

So the Theorem is true for all sequences of length 2. Assume the statements are true for all sequences s^l, where $l < N + 1$ and prove them for s^{N+1}.

Statement 2 is obviously true. Let $d_N = 1$. Let s^N be generated by $f_N(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ and s^m be generated by $f_m(X) = X^{L_m} + b_1 X^{L_m - 1} + \ldots + b_{L_m}$. One writes

$$s_{N-i} + c_1 s_{N-i-1} + \ldots + c_L s_{N-i-L} = \begin{cases} d_N = 1, & i = 0; \\ 0, & N - L \geq i > 0 \end{cases}$$

and

$$s_{m-i} + b_1 s_{m-i-1} + \ldots + b_{L_m} s_{m-i-L_m} = \begin{cases} d_m = 1, & i = 0; \\ 0, & m - L_m \geq i > 0. \end{cases} ,$$

We sum up the above two equations. So when $\min\{N - L, m - L_m\} \geq i \geq 0$ we get a generating recurrence for s^{N+1}:

$$\begin{gathered} s_{N-i} + c_1 s_{N-i-1} + \ldots + c_L s_{N-i-L} + \\ s_{m-i} + b_1 s_{m-i-1} + \ldots + b_{L_m} s_{m-i-L_m} = 0. \end{gathered}$$

Consider two cases: $\min\{N - L, m - L_m\}$ equals $m - L_m$ or $N - L$. We will have two different polynomials.

In the first case $N - L > m - L_m = L - 1$, because $L_m + L = m + 1$ by induction. Therefore $N - L > m - L_m$ is equivalent to $L \leq \frac{N}{2}$. To get the generating polynomials we put $i = m - L_m = L - 1$ and the first recurrent relation

$$s_{N-L+1} = c_1 s_{N-L} + \ldots + c_L s_{N-2L+1} + s_{m-L+1} + b_1 s_{m-L} + \ldots + b_{L_m} s_0.$$

All other relations are its shifts. Therefore, the generating polynomial for s^{N+1} is $X^{N-2L+1} f_N + f_m$. Its degree is $N - L + 1$. We already know that $L_{N+1} \geq N - L + 1$, otherwise $d_N = 0$ by Lemma 4. So $L_{N+1} = N - L + 1$ and we put $f_{N+1} = X^{N-2L+1} f_N + f_m$.

In the second case $N - L \leq m - L_m = L - 1$ is equivalent to $L > \frac{N}{2}$. We put $i = N - L$ and get the first relation

$$s_L = c_1 s_{L-1} + \ldots + c_L s_0 + s_{m-N+L} + b_1 s_{m-N+L-1} + \ldots + b_{L_m} s_{2L-N-1},$$

as $(m - L_m) - (N - L) = 2L - N - 1$. All other relations are its shifts. Therefore, the generating polynomial is $f_N + X^{2L-N-1} f_m$. Its degree is L, because

$$\deg X^{2L-N-1} f_m = 2L - N - 1 + L_m = L - N + m < L = \deg f_N,$$

we put $f_{N+1} = f_N + X^{2L-N-1} f_m$ and $L_{N+1} = L$. This finishes the proof of the Theorem.

Corollary 4 *Let $s = s_0, s_1, \ldots$ be binary sequence of linear complexity L and $s^t = s_0, s_1, \ldots, s_{t-1}$ be an interval of s of length $t \geq 2L - 1$. Then $L(s^t) = L$.*

Proof. Let, by contrary, $L(s^t) < L$. Then there exists a natural $T > t$ such that

$$L(s^t) = L(s^{t+1}) = \ldots = L(s^{T-1}),$$

but $L(s^T) > L(s^t)$. The number T does really exist. By the third statement of the Theorem $L(s^T) + L(s^t) = T$, where $T \geq 2L$ by assumption. On the other hand, $L(s^T) + L(s^t) < 2L$. The contradiction proves the corollary.

Corollary 5 *Let $s^{2L} = s_0, s_1, \ldots, s_{2L-1}$ be a binary sequence of linear complexity L and $f(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ be its generating polynomial. Then the coefficient vector of f is uniquely defined by the system of linear equations*

$$(c_1, \ldots, c_L) \begin{pmatrix} s_{L-1} & s_L & \ldots & s_{2L-2} \\ \ldots & & & \\ s_1 & s_2 & \ldots & s_L \\ s_0 & s_1 & \ldots & s_{L-1} \end{pmatrix} = (s_L, s_{L+1}, \ldots, s_{2L-1}).$$

Proof. By the previous statement we get that $L(s^{2L-1}) = L$. So the matrix of the system is nonsingular by Theorem . Therefore, the solution and, hence, the polynomial f is determined uniquely.

We return to the example now and compute f_8 first. As $d_7 = 1$ and $L_7 = 3 < 7/2$ we get

$$f_8 = X^{7-2\times 3+1} f_7 + f_3 = X^2(X^3 + 1) + X = X^5 + X^2 + X.$$

Let us compute $f_9(X)$. One sees that $d_8 = 1$ and $L_8 = 5 > 8/2$. So

$$f_9 = f_8 + X^{2\times 5-8-1} f_7 = (X^5 + X^2 + X) + X(X^3 + 1) = X^5 + X^4 + X^2.$$

Similarly one computes f_{10} as

$$f_9 + X^{2\times 5-9-1} f_7 = (X^5 + X^4 + X^2) + (X^3 + 1) = X^5 + X^4 + X^3 + X^2 + 1.$$

Berlekamp-Massey algorithm computes the linear complexity profile and generating polynomials for a given binary sequence by using Theorem 6 in $O(L^2)$ bit operations. Important application of the Berlekamp-Massey algorithm is in the following. Let s be a binary sequence of linear complexity L and let a subsequence t of s of length at least $2L$ be given. Then Berlekamp-Massey algorithm given t finds a generating polynomial for s.

2.3 Boolean Functions

Definition 5 *Let n be a natural number and V_n denote a set of all binary n-strings. A Boolean functon is a mapping $f : V_n \to \{0, 1\}$, where n is called the number of variables of f.*

The function may be defined by a table of its values, called truth table. For example, for $n = 3$ consider the function defined by

x	x_1	x_2	x_3	f_0
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

.

There are some other ways to represent Boolean functions. We will define the algebraic normal form(ANF) of a Boolean function in n variables. It is a polynomial in $x_1, x_2, \ldots, x_n$ with coefficients $\{0, 1\}$, sum modulo 2 of monomials $x_{i_1} x_{i_2} \ldots x_{i_t}$, where $\{i_1, i_2, \ldots, i_t\}$ run over all subsets of indices $1, 2, \ldots, n$. That is

$$f = \sum_{\{i_1, i_2, \ldots, i_t\}} c_{\{i_1, i_2, \ldots, i_t\}} x_{i_1} x_{i_2} \ldots x_{i_t},$$

where $c_{\{i_1, i_2, \ldots, i_t\}} = 0, 1$ including the coefficient $c_\emptyset$ at 1. We will prove that each Boolean function has such a representation and explain a fast way to compute the ANF given function's truth table. We will find that the above Boolean function is represented by a polynomial

$$x_1 x_2 x_3 + x_1 x_2 + x_1 x_3 + x_2 x_3 + x_1.$$

We will introduce some useful notation first. To each binary n-string $x = (x_1, x_2, \ldots, x_n)$ we relate an integer number by the rule:

$$x = x_1 2^{n-1} + x_2 2^{n-2} + \ldots + x_n,$$

so that $0 = (0, 0, \ldots, 0), \ldots, 2^n - 1 = (1, 1, \ldots, 1)$. That gives a natural lexicographic ordering on the set of all binary n-strings. So the function f is represented by a 2^n-string of its values:

$$f = (f(0), f(1), \ldots, f(2^n - 1)).$$

For our example function $f = (0, 0, 0, 1, 1, 0, 0, 1)$. Similarly, we represent a coefficient vector for the polynomial representing f:

$$C = (c_0, c_1, \ldots, c_{2^n - 1}),$$

where $c_a = c_{\{i_1,i_2,\ldots,i_t\}}$, $a = 2^{n-i_1}+2^{n-i_2}+\ldots+2^{n-i_t}$ and $c_0 = c_\emptyset$. For instance, we have

$$C = (0,0,0,1,1,1,1,1)$$

for the example function f. We will relate vectors f and C.

Theorem 7

$$fA_n = C,$$

where

$$A_n = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{[n]}.$$

That is A_n is a tensor power of the matrix $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, or in other notation,

$$A_n = \begin{pmatrix} A_{n-1} & A_{n-1} \\ 0 & A_{n-1} \end{pmatrix}.$$

We prove the Theorem by induction on n. Let $n = 1$. We take any Boolean function in one variable $(f(0), f(1))$ and apply the linear operator A_1. We get

$$(f(0), f(1)) \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = (f(0), f(0) + f(1)).$$

On the other hand, it is obvious that

$$f(x_1) = f(0) + (f(0) + f(1))x_1.$$

So

$$C = (f(0), f(0) + f(1))$$

and therefore $fA_1 = C$. Let this be true when the number of variables is less than n and prove the statement when the number of variables is just n. We put

$$f_0(x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n),$$
$$f_1(x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n).$$

So that $f_0 = (f_0(0), \ldots, f_0(2^{n-1}-1))$ and $f_1 = (f_1(0), \ldots, f_1(2^{n-1}-1))$. Then, obviously, $f = (f_0, f_1)$ is the concatenation of f_0 and f_1. Let C_0 and C_1 be related coefficient vectors for f_0 and f_1. Then by induction

$$C_0 = f_0 A_{n-1},$$
$$C_1 = f_1 A_{n-1}.$$

Now

$$f = f_0 + (f_0 + f_1)x_1$$

and, therefore, $C = (C_0, C_0 + C_1)$. Then

$$C = (C_0, C_0 + C_1) = (f_0 A_{n-1}, f_0 A_{n-1} + f_1 A_{n-1}) = (f_0, f_1) \begin{pmatrix} A_{n-1} & A_{n-1} \\ 0 & A_{n-1} \end{pmatrix}.$$

Therefore, $C = fA_n$. This finishes the proof.

The Theorem implies that given f we find C by application A_n to f. Generally, it takes about 2^{2n} binary operations. We will provide with an algorithm able to accomplish the task in $O(n2^n)$ operations.

input Boolean function f in n variables.

output Coefficients vector defining the ANF of f.

For any $0 \le k \le n$ and $0 \le a \le 2^{n-k} - 1$ we define auxiliary vectors $f_{k,a}$ of length 2^k.

1. Initialize $f_{0,a} = f(a)$ for $0 \le a \le 2^n - 1$.
2. For k from 0 to $n - 1$ do
$$f_{k+1,b} = (f_{k,2b}, f_{k,2b+1} + f_{k,2b})$$
for all $0 \le b \le 2^{n-k-1} - 1$.
3. Put $C = f_{n,0}$.

The proof that the algorithm really computes the ANF repeats the proof of Theorem 7. Let us estimate the complexity. At each stage one does 2^{n-k-1} additions modulo 2 (XORs) of vectors of length 2^k. This makes $2^k 2^{n-k-1} = 2^{n-1}$ XORs. On the whole $n2^{n-1}$ XORs of binary digits are necessary. Then one does $O(2^n)$ rewritings of bits. The resulting complexity is $O(n2^n)$ bit operations. For example,

f	0	0	0	1	1	0	0	1
	0	0	0	1	1	1	0	1
	0	0	0	1	1	1	1	0
C	0	0	0	1	1	1	1	1

We look at the positions of 1 in C. We see

$$\begin{array}{ll} 3 = 2 + 1 & x_2 x_3 \\ 4 = 2^2 & x_1 \\ 5 = 2^2 + 1 & x_1 x_3 \\ 6 = 2^2 + 2 & x_1 x_2 \\ 7 = 2^2 + 2 + 1 & x_1 x_2 x_3. \end{array}$$

So the ANF for the example function f is $x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_1$ as stated.

Definition 6 *1. Algebraic degree of f is the degree of a polynomial representing the ANF of f. It is denoted* $\deg f$.

2. *If* $\deg f = 1$ *the function is called affine. Any affine function looks as*

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n + b,$$

where $a_1, a_2, \ldots, a_n, b$ *are fixed binary digits.If* $\deg f = 2$ *the function is called quadratic and when* $\deg f = 3$ *it is called cubic.*

The function from the last example is cubic, that is $\deg f = 3$.

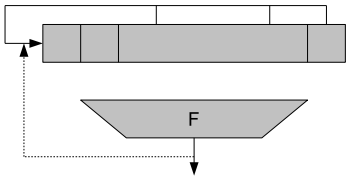

Figure 11: LFSR with a filter

2.4 A common stream cipher

A common stream cipher is being constructed with two components: a LFSR of length L of maximal period $2^L - 1$, defined by a primitive polynomial $X^L + c_1 X^{L-1} + \ldots + c_L$, and a Boolean function $F(x_1, x_2, \ldots, x_m)$. The device is depicted in Figure 11.

When m is not so big the Boolean function F being defined by its values, that is 2^m bits, is kept in some memory locations. But often to complicate matters for a cryptanalyst a Boolean function of many variables is used. In this case it is possible to achieve a larger algebraic degree. To this end F is a composition of some Boolean functions in small number of variables. For instance,

$$F = F_1(G_1(x_{1,1}, \ldots, x_{1,k}), \ldots, G_k(x_{k,1}, \ldots, x_{k,k}).$$

Such a function depends on k^2 variables, that is it has 2^{k^2} values, but one only keeps $(k+1)2^k$ values of $F_1, G_1, \ldots, G_k$. The device depicted in Figure 11 is not completely a stream cipher. It should be initialized. The goal is to introduce a key k, an initial value IV and determine an initial state of the register. So r bits of the key k and s bits of IV along with $L - r - s$ fixed bits, like 0, are used to set up a preinitial state. Then the register should work a fixed number of clocks, e.g. $5L$. To complicate the cryptanalysis feedback in the initialization process is taken nonlinear. E.g., some output bits may be used to form the feedback. After the initialization the content of the register $s_{L-1}, s_{L-2}, \ldots, s_0$ nonlinearly depends on the keybits and IV bits. So the encryphtion-decryption process is presented in Figure 12. IV is sent before the cipher-text in clear. So the receiver becomes able to synchronize the cipher in order to decrypt the cipher-text correctly. Consider cryptanalyst's assumptions and tasks. The cryptanalyst is supposed to have a physical device implementing the cipher but not the key. Then he has the whole cipher-text and some segments of the related plain-text. Therefore, some related segments of the key-stream are supposed to be known too. The task is to reveal some new segments of the plain-text or the whole plain-text. In order to accomplish the tasks the cryptanalyst can try three approaches:

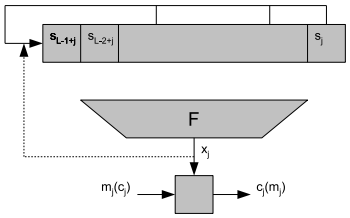

Figure 12: A common stream cipher

1. Find the key.
2. Find the initial state without finding the key.
3. Construct a device modeling the key-stream generator with finding neither the key nor the initial state.

As an example of the second approach we consider a Time-Memory Trade off. This is a method of finding an initial state of the cipher given a very large segment of the plain-text(key-stream). Let S_0 be an initial state of the LFSR of length L with a primitive polynomial that we are looking for. Let m denote $\lceil\sqrt{2^L-1}\rceil$ the smallest integer number bigger or equal to $\sqrt{2^L-1}$. The task is given $m+L-1$ key-stream digits, find S_0. Let S be any state, then we denote

$$\bar{F}(S) = (F(S), F(SA), \ldots, F(SA^{L-1}),$$

where A is the matrix implementing the state change, it is a kind of companion matrix to the generating polynomial. Our assumption is that $\bar{F}(S)$ uniquely defines the state S. That is $\bar{F}(S_1) = \bar{F}(S_2)$ if and only if $S_1 = S_2$.

Let S_1 be any fixed nonzero state of our choice. Then $S_1 = S_0A^y$ for some $0 \le y < 2^L - 1$, because all nonzero states occur at the period as the generating polynomial is primitive. We will show how to find y. One divides y by m with remainder: $y = sm + r$, where $0 \le r < m$. Then $s = (y-r)/m \le y/m < m$, because $2^L - 1 < m^2$.

1. For all $0 \le r < m$ compute pairs $(r, \bar{F}(S_0A^r))$. One needs $m+L-1$ key-stream bits.
2. Order the set of pairs by the second coordinate.
3. For $0 \le s < m$ compute $\bar{F}(S_1A^{-sm})$ and look up the string as the second coordinate of a pair in the ordered set of pairs. Our theory guarantees that such a match always occurs. So $\bar{F}(S_1A^{-sm}) = \bar{F}(S_0A^r)$ and $y = sm + r$.

The complexity is $O(Lm) = O(L2^{L/2})$ operations with L-strings. Memory requirement is of the same order.

2.5 Berlekamp-Massey Attack

Consider a synchronous stream cipher as depicted in Figure 12. The goal is given a cipher-text $c_0, c_1, \ldots, c_{N-1}, c_N, \ldots$ and a segment $m_0, m_1, \ldots, m_{N-1}$ of the plain-text, reveal the rest of the plain-text.

We find the related key-stream $x^N = x_0, x_1, \ldots, x_{N-1}$ and construct a LFSR generating x^N by a Berlekamp-Massey algorithm application. We will see that if N is big enough the LFSR generates all key-stream bits. In so doing one finds $x_N, x_{N+1}, \ldots$ and $m_N, m_{N+1}, \ldots$. The main question is how big N should be taken in order to make the method work. The following Theorem is basic.

Theorem 8 *Let a LFSR be given and F is of algebraic degree d. Then the linear complexity of the key-stream $\{x_i\}$ is no more than $D = \sum_{i=0}^{d} \binom{L}{i}$.*

Proof. Let $(s_{L-1}(t), \ldots, s_1(t), s_0(t))$ be the LFSR state at time t. The following vector is called a monomial state of degree d.

$$S(t) = (1, s_{L-1}, \ldots, s_1, s_0, s_{L-1}s_0, \ldots, s_1 s_0, \ldots, s_{L-1}s_{L-2}\ldots s_{L-d}).$$

This is a vector of all products of order $\leq d$ of the state bits at time t. $S(t)$ is of length $D = \sum_{i=0}^{d} \binom{L}{i}$. Let

$$F(X_{L-1}, \ldots, X_0) = \sum_{i_1..i_t} c_{\{i_1..i_t\}} X_{i_1} \ldots X_{i_t}.$$

We define

$$C_F = (c_\emptyset, c_{\{L-1\}}, \ldots, c_{\{0\}}, c_{\{L-1,0\}}, \ldots, c_{\{1,0\}}, \ldots, c_{\{L-1,L-2,\ldots,L-d\}}),$$

where $c_\emptyset$ is a free term coefficient in F. Therefore, $x_t = S(t)C_F$, the dot-product of the two vectors. The next monomial state $S(t+1)$ is a linear transform of $S(t)$ as

$$s_i(t+1) = s_{i+1}(t), \quad 0 \leq i \leq L-2$$

and

$$s_{L-1}(t+1) = c_1 s_{L-1}(t) + \ldots + c_L s_0(t),$$

where $X^L + c_1 X^{L-1} + \ldots + c_L$ is a LFSR generating polynomial. So $S(t+1) = S(t)R = \ldots = S(1)R^t$ for a matrix R of size $D \times D$ and so $S(t+1) = S(1)R^t$. We finally have

$$x_t = S(1)R^t C_F.$$

Let $G(X) = b_0 X^D + b_1 X^{D-1} + \ldots + b_D$, $b_0 = 1$ be a characteristic polynomial of the matrix R. So

$$\sum_{i=0}^{D} b_i \, x_{t-i} = \sum_{i=0}^{D} b_i \, S(1)R^{t-i}C_F = S(1)R^{t-D}(\sum_{i=0}^{D} b_i R^{D-i})C_F = 0$$

Hence $G(X)$ is a generating polynomial for the key-stream $\{x_i\}$. Its linear complexity is bounded by D as stated.

Experiments show that the linear complexity is about this number. Then our theory on linear complexity states that $\leq 2D$ digits of the key-stream is enough to determine a generating polynomial uniquely. We summarize the method.

1. Having got a segment of the plain-text, compute the related segment of the key-stream x^N of length $N \geq 2D$.

2. Find the linear complexity profile for x^N along with polynomials. We only need a final one. Let $L(x^N) = L_1$ and $f_N(X)$ be a polynomial of degree L_1 generating x^N.

3. Construct a LFSR of length L_1 and with polynomial $f_N(X)$. Use $x_0, \ldots, x_{L_1-1}$ as an initial state of the LFSR to produce the whole key-stream $x_0, x_1, \ldots, x_{N-1}, x_N, \ldots$.

4. Compute the rest of the plain-text $m_i = c_i + x_i$ for $i = N, N+1, \ldots$.

The complexity of the method is $O(D^2)$ bit operations and at most $2D$ bits of the key-stream are necessary.

2.6 Berlekamp-Massey Algorithm and the GCD

In this Section we will show that it is possible to compute a generating polynomial faster than with Berlekamp-Massey algorithm. Let $s^N = s_0, s_1, \ldots, s_{N-1}$ be a binary sequence of length N and $f(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ be any polynomial of degree $L < N$. Denote

$$C(X) = X^L f(1/X) = 1 + c_1 X + \ldots + c_L X^L$$
$$S(X) = 1 + s_0 X + \ldots + s_{N-1} X^N.$$

Lemma 5 *There exists a polynomial $D(X)$ of degree $\leq L$ such that*

$$S(X)C(X) \equiv D(X) \mod X^{N+1}.$$

if and only if $f(X)$ is a generating polynomial for s^N.

Proof. We find

$$\begin{aligned} S(X)C(X) &= (1 + s_0 X + \ldots + s_{N-1} X^N)(1 + c_1 X + \ldots + c_L X^L) \\ &= D(X) + (s_L + c_1 s_{L-1} + \ldots + c_{L-1} s_1 + c_L s_0) X^{L+1} + \ldots \\ &+ (s_{N-1} + c_1 s_{N-2} + \ldots + c_{L-1} s_{N-L} + c_L s_{N-L-1}) X^N \mod X^{N+1}, \end{aligned}$$

where

$$\begin{aligned} D(X) &= 1 + (s_0 + c_1) X + (s_1 + c_1 s_0 + c_2) X^2 + \ldots \\ &+ (s_{L-1} + c_1 s_{L-2} + \ldots + c_{L-1} s_0 + c_L) X^L. \end{aligned}$$

So $S(X)C(X) \equiv D(X) \mod X^{N+1}$ if and only if

$$\begin{aligned} & s_L + c_1 s_{L-1} + \ldots + c_{L-1} s_1 + c_L s_0 = 0, \\ & \ldots \\ & s_{N-1} + c_1 s_{N-2} + \ldots + c_{L-1} s_{N-L} + c_L s_{N-L-1} = 0. \end{aligned}$$

That is if and only if the polynomial $f(X)$ generates s^N. The Lemma is proved.

Let $s^{2L} = s_0, s_1, \ldots, s_{2L-1}$ be a binary sequence of linear complexity L and $f(X)$ be a minimum degree generating polynomial for s^{2L}. That is f is of degree L. Put again

$$\begin{aligned} C(X) &= X^L f(1/X) = 1 + c_1 X + \ldots + c_L X^L, \\ S(X) &= 1 + s_0 X + \ldots + s_{2L-1} X^{2L}. \end{aligned}$$

Lemma 6 *1. There exist nonzero polynomial $D(X)$ such that $\deg D(X) \leq L$ and*

$$S(X)C(X) \equiv D(X) \mod X^{2L+1}.$$

2. The polynomials $C(X)$ and $D(X)$ are coprime and $\deg C(X) = L$ or $\deg D(X) = L$.

3. *The polynomials $C(X)$ and $D(X)$ are the only polynomials of degree $\leq L$ satisfying the above congruence.*

Proof. $D(X)$ exists by Lemma 5. Assume $\deg C(X), \deg D(X)$ are at most $L_1 < L$. By Lemma 5, the polynomial $f_1(X) = X^{L_1}C(1/X)$ of degree L_1 generates s^{2L}. That is a contradiction as s^{2L} is of linear complexity L.

We prove $C(X)$ and $D(X)$ are coprime. Let $\gcd(C, D) = h$ for some nonconstant polynomial $h(X)$. Because $S(X)C(X) = D(X) + X^{2L+1}T(X)$ for some polynomial $T(X)$, we consider two cases:

1. $h(X)$ has a common nonconstant divisor with X^{2L+1}, then X divides $h(X)$ and so X divides $C(X)$. That is a contradiction.

2. $h(X)$ is coprime to X^{2L+1}. Then h divides T. One defines

$$C_1(X) = C(X)/h(X), \quad D_1(X) = D(X)/h(X).$$

Therefore,

$$S(X)C_1(X) \equiv D_1(X) \bmod X^{2L+1},$$

where $C_1(X), D_1(X)$ of degree $< L$, say $\deg C_1(X) = L_1$. Then by Lemma 5, the polynomial $X^{L_1}C_1(1/X)$ of degree L_1 generates s^{2L}. That is a contradiction as s^{2L} is of linear complexity L.

To prove the uniqueness of the pair is unique we assume another pair $C_1(X)$ and $D_1(X)$ of polynomials of degree $\leq L$ such that

$$S(X)C_1(X) \equiv D_1(X) \bmod X^{2L+1}.$$

Then we get

$$C_1(X)D(X) \equiv C(X)D_1(X) \bmod X^{2L+1}.$$

In other words, X^{2L+1} divides the polynomial $C_1(X)D(X) - C(X)D_1(X)$ which is of degree $\leq 2L$. This is only possible when $C_1(X)D(X) = C(X)D_1(X)$. As $C(X)$ and $D(X)$ are coprime, $C(X)$ divides $C_1(X)$, that is $C_1(X) = C(X)h(X)$ and so $D_1(X) = D(X)h(X)$. Hence $h(X) = 1$ because $C(X)$ or $D(X)$ is of degree L. Then $C_1(X) = C(X)$ and $D_1(X) = D(X)$. This proves the last statement.

2.6.1 Euclidean Algorithm

We will remind the Euclidean Algorithm.

input. Polynomials $a_0(X)$ and $a_1(X)$ such that $\deg a_0 > \deg a_1$.

output. Polynomial $d(X)$ a gcd of polynomials $a_0(X)$ and $a_1(X)$.

$$\begin{aligned} a_0(X) &= q_1(X)a_1(X) + a_2(X), \\ a_1(X) &= q_2(X)a_2(X) + a_3(X), \\ &\dots \\ a_{s-1}(X) &= q_s(X)a_s(X) + a_{s+1}(X), \end{aligned}$$

This a sequence of polynomial divisions with remainder, that is

$$\deg a_0 > \deg a_1 > \ldots > \deg a_s$$

and for the first time $a_{s+1}(X) = 0$. So $d(X) = a_s(X)$. The polynomials $a_0, a_1, \ldots, a_s$ are called remainders and $q_1, q_2, \ldots, q_{s-1}$ quotients.

Lemma 7 $\deg q_i = \deg a_{i-1} - \deg a_i$.

To estimate the complexity one first finds that $s \leq \deg a_0$. At each step one spends $O((1 + \deg q_i) \deg a_i)$ bit operations for division with remainder. So the complexity of the whole Euclidean Algorithm is

$$O(\sum_{i=1}^{s}(1 + \deg q_i) \deg a_i)) = O(\sum_{i=1}^{s}(1 + \deg a_{i-1} - \deg a_i) \deg a_1) = O((s + \deg a_0 - \deg a_s) \deg a_1) = O(\deg a_0 \deg a_1)$$

We need a variant of the Euclidean Algorithm called Extended Euclidean Algorithm. At each step of the algorithm three polynomials u_i, v_i, a_i are computed. They satisfy the equality $u_i a_0 + v_i a_1 = a_i$. The algorithm is usually depicted with a table

q_i	u_i	v_i	a_i
	1	0	a_0
$q_1 = \lfloor a_0/a_1 \rfloor$	0	1	a_1
$q_2 = \lfloor a_1/a_2 \rfloor$	1	q_1	$a_2 = a_0 + q_1 a_1$
$\dots$	q_2	$1 + q_1 q_2$	$a_3 = a_1 + q_2 a_2$
$\dots$			
$q_s = \lfloor a_{s-1}/a_s \rfloor$	u_s	v_s	$a_s = a_{s-2} + q_{s-1} a_{s-1}$
	u_{s+1}	v_{s+1}	0.

The i-th row(in the last three columns) is a sum of the $i-1$-th row multiplied by q_{i-1} and the $i-2$-th row for all $2 \leq i \leq s+1$.

$$(u_i, v_i, a_i) = q_{i-1}(u_{i-1}, v_{i-1}, a_{i-1}) + (u_{i-2}, v_{i-2}, a_{i-2})$$

The algorithm stops when $a_{s+1} = 0$. The previous remainder a_s is the sought gcd. The complexity of the Extended Euclidean Algorithm is again $O(\deg a_0 \deg a_1)$.

As an example we apply the Extended Euclidean Algorithm to the polynomials $X^5 + X^2 + X + 1$ and X^2:

q_i	u_i	v_i	a_i
	1	0	$X^5 + X^2 + X + 1$
$X^3 + 1$	0	1	X^2
$X + 1$	1	$X^3 + 1$	$X + 1$
$X + 1$	$X + 1$	$X^4 + X^3 + X$	1
	X^2	$X^5 + X^2 + X + 1$	0.

Lemma 8 *Let* $\deg a_0 > \deg a_1$. *Then*

1. $\deg u_i < \deg v_i$ *for all* $i = 2, 3, \ldots, s + 1$.
2. $\deg v_i = \deg a_0 - \deg a_{i-1}$ *for all* $i = 2, 3, \ldots, s + 1$.
3. *Let* l *be a natural number such that*

$$\deg a_l > \frac{\deg a_0}{2} \geq \deg a_{l+1}$$

   Then $\deg v_{l+1} < \frac{\deg a_0}{2}$.

4. $u_i v_{i+1} + v_i u_{i+1} = 1$ *for all* $i = 0, \ldots, s$. *In particular,* u_i, v_i *are coprime.*

Proof. In order to prove the first statement we realize that $v_i = q_1 q_2 \ldots q_i + \ldots$ and $u_i = q_2 \ldots q_i + \ldots$, they are only represented by the most significant terms. Due to $\deg a_0 > \deg a_1$ the degree of q_1 is at least 1. So $\deg u_i < \deg v_i$. For the second statement we see that $v_1 = 1$ and $\deg v_1 = \deg a_0 - \deg a_0 = 0$. So the statement is true for $i = 1$. Then $v_2 = q_1$ and so $\deg v_2 = \deg a_0 - \deg a_1$, so the statement holds for $i = 2$. We find that $v_i = v_{i-2} + q_{i-1} v_{i-1}$, so by induction

$$\deg v_i = \deg q_{i-1} + \deg v_{i-1} =$$
$$\deg a_{i-2} - \deg a_{i-1} + \deg a_0 - \deg a_{i-2} = \deg a_0 - \deg a_{i-1}$$

Therefore the second statement of the Lemma is true in general. In order to prove the third statement we compute

$$\deg v_{l+1} = \deg a_0 - \deg a_l < \deg a_0 - \frac{\deg a_0}{2} \leq \frac{\deg a_0}{2}.$$

We realise that

$$\begin{pmatrix} 0 & 1 \\ 1 & q_i \end{pmatrix} \begin{pmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{pmatrix} = \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix}.$$

By induction this implies that the determinant of the last matrix $u_i v_{i+1} + v_i u_{i+1} = 1$. This finishes the proof of the Lemma.

We return to the Berlekamp-Massey Algorithm. We put $a_0 = X^{2L+1}$ and $a_1 = S(X)$. Then for a natural l such that $\deg a_l > L \geq \deg a_{l+1}$ one gets the equation

$$u_{l+1}X^{2L+1} + v_{l+1}S(X) = a_{l+1}.$$

After taking it modulo X^{2L+1} we obtain the congruence

$$v_{l+1}S(X) \equiv a_{l+1} \, mod \, X^{2l+1},$$

where $\deg v_{l+1}, \deg a_{l+1} \leq L$. Then we put by Lemma that

$$C(X) = v_{l+1} \quad \text{and} \quad D(X) = a_{l+1}.$$

So we get a generating polynomial for s^{2L}, that is $f(X) = X^L C(\frac{1}{X})$.

Example. Let

$$S^{22} = 00110111010100011011 10.$$

Then we put

$$S(X) = 1+X^3+X^4+X^6+X^7+X^8+X^{10}+X^{12}+X^{16}+X^{17}+X^{19}+X^{20}+X^{21}$$

and apply the Extended Euclidean Algorithm to $S(X)$ and X^{23}. In so doing we produce a sequence of remainders:

$$a_0 = X^{23},\ a_1 = S(X), \ldots, a_9 = X + X^4 + X^5 + X^8 + X^{12},$$
$$a_{10} = 1 + X^3 + X^4 + X^6 + X^7 + X^{10}.$$

We have stopped at this point because $\deg a_9 > 11 \geq \deg a_{10}$. Along with remainders we have produced a sequence of related polynomials v_i, in particular, $v_{10} = 1 + X^8 + X^{11}$. So $C(X) = 1 + X^8 + X^{11}$ and $f(X) = X^{11} + X^3 + 1$.

Let the sequence $s^N = s_0, s_1, \ldots, s_{N-1}$ of linear complexity L be given and $N \geq 2L$, though the number L remains unknown. In this case the same approach to reveal a generating polynomial for s^N is applicable. We apply the Extended Euclidean Algorithm to the polynomials X^{N+1} and $S(X) = 1 + s_0X + \ldots + s_{N-1}X^N$, stop when $\deg a_l > \frac{N+1}{2} \geq \deg a_{l+1}$ and get v_{l+1}. Then $v_{l+1} = X^r C(X)$ for $r \geq 0$, where r is easy to find as $C(0) = 1$. Really, as

$$\begin{aligned} SC &= D \mod X^{N+1} \\ Sv_{l+1} &= a_{l+1} \mod X^{N+1} \end{aligned}$$

we have $Ca_{l+1} = Dv_{l+1}$ because $\deg C, \deg D \leq L$ and $\deg a_{l+1}, \deg v_{l+1} \leq \frac{N+1}{2}$ and $N \geq 2L$. As C and D are coprime, then $v_{l+1} = Ch$ and $a_{l+1} = Dh$ for some polynomial h. From

$$u_{l+1}X^{N+1} + v_{l+1}S = a_{l+1}$$

we see h divides X^{N+1} as u_{l+1}, v_{l+1} are coprime. So $h = X^r$ for some $r \geq 0$.

In so doing we find C. The generating polynomial is $f(X) = X^L C(\frac{1}{X})$, where $L \geq \deg C(X)$ is found by trials. There should be only few trials in practice. For instance, if s is pure periodic, then no trials at all as $\deg C = L$.

The complexity of the computations is $O(N^2)$ as with the plain Berlekamp-Massey Algorithm. But we have represented our task in a form suitable for the application of the asymptotically fast GCD Algorithm based on a procedure called Half GCD. It enables computing a generating polynomial in $O(N \log^2 N)$ bit operations.

First, we will describe the Extended Euclidean Algorithm with matrix notation. That is

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & q_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix},$$

$$\begin{pmatrix} a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & q_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & q_2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & q_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}, \ldots.$$

Generally,

$$\begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & q_i \end{pmatrix} \ldots \begin{pmatrix} 0 & 1 \\ 1 & q_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}.$$

Then we denote

$$R_i^{(a_0,a_1)} = \begin{pmatrix} 0 & 1 \\ 1 & q_i \end{pmatrix} \ldots \begin{pmatrix} 0 & 1 \\ 1 & q_1 \end{pmatrix} \begin{pmatrix} u_0 & v_0 \\ u_1 & v_1 \end{pmatrix},$$

where $\begin{pmatrix} u_0 & v_0 \\ u_1 & v_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. So $R_i^{(a_0,a_1)} = \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix}$. For any natural i by $l(i)$ we denote the unique integer number such that $\deg a_{l(i)} > i \geq \deg a_{l(i)+1}$.

Theorem 9 *Let $f(X) = f_1(X)X^k + f_2(X)$ and $g(X) = g_1(X)X^k + g_2(X)$ be two polynomials such that*

1. $\deg f_2, \deg g_2 < k$,
2. $\deg f_1 > \deg g_1 \geq \frac{\deg f_1}{2}$,
3. $\deg f = n$.

Then

$$R^{(f,g)}_{l(\lceil \frac{n+k}{2} \rceil)} = R^{(f_1,g_1)}_{l(\lceil \frac{n-k}{2} \rceil)}.$$

Proof. Let $a_0 = f, a_1 = g$ and $b_0 = f_1, b_1 = g_1, b_2, \ldots$ be a sequence of remainders produced by the Euclidean Algorithm for the polynomials f_1 and g_1. Then

$$b_{i+1} = b_{i-1} + q_i b_i,$$

where q_i is a quotient and $\deg q_i = \deg b_{i-1} - \deg b_i$. We consider a sequence of polynomials defined by the rule

$$a_{i+1} = a_{i-1} + q_i a_i.$$

We will prove that if

$$\deg b_i > \lceil \deg f_1/2 \rceil = \lceil \deg b_0/2 \rceil = \lceil (n-k)/2 \rceil,$$

then $\deg a_{i+1} < \deg a_i$ and $\deg a_i > k + \frac{\deg f_1}{2}$. This would mean that a_{i+1} is the remainder and q_i is the quotient in the division of a_{i-1} by a_i. So q_i, for all such i, would be a quotient in the application of the Euclidean Algorithm to $a_0 = f, a_1 = g$. Really,

$$\begin{aligned} a_2 = a_0 + q_1 a_1 = (f_1 X^k + f_2) + q_1(g_1 X^k + g_2) = \\ (b_0 + q_1 b_1) X^k + (f_2 + q_1 g_2) = b_2 X^k + c_2, \end{aligned}$$

where $\deg c_2 < k + \deg q_1$. Then

$$\begin{aligned} a_3 = a_1 + q_2 a_2 = (g_1 X^k + g_2) + q_2(b_2 X^k + c_2) = \\ (g_1 + q_2 b_2) X^k + (g_2 + q_2 c_2) = b_3 X^k + c_3, \end{aligned}$$

where $\deg c_3 < k + \deg q_1 + \deg q_2$ and so on. Generally,

$$\begin{aligned} a_{i+1} = a_{i-1} + q_i a_i = (b_{i-1} X^k + c_{i-1}) + q_i(b_i X^k + c_i) = \\ (b_{i-1} + q_i b_i) X^k + (c_{i-1} + q_i c_i) = b_{i+1} X^k + c_{i+1}, \end{aligned}$$

where

$$\begin{aligned} \deg c_{i+1} &< k + \deg q_1 + \ldots + \deg q_i \\ &= \deg b_0 - \deg b_i + k \\ &< \deg b_i + k < \deg b_{i-1} + k. \end{aligned}$$

Therefore,

$$\begin{aligned} \deg a_{i+1} &= \deg(b_{i+1} X^k + c_{i+1}) < \deg b_i + k \\ &\leq \deg b_i X^k + c_i = \deg a_i \end{aligned}$$

and $\deg a_i > k + \lceil \frac{\deg f_1}{2} \rceil = \lceil \frac{n+k}{2} \rceil$. Let l be the maximal number such that $\deg b_l > \lceil \frac{n-k}{2} \rceil$ then $q_1, \ldots, q_l$ are quotients related to the sequence of remainders $a_0, a_1, \ldots, a_l$ and still $\deg a_l > \lceil \frac{n+k}{2} \rceil$.

Similarly, let l_1 be the maximal number such that $\deg a_{l_1} > \lceil \frac{n+k}{2} \rceil$. Then one proves that, starting with a_0, a_1, one produces a sequence of quotients $q'_1, \ldots, q'_{l_1}$ and remainders $a_0, a_1, \ldots, a_{l_1}$. These remainders satisfy the property: if

$$\deg a_i > k + \lceil \frac{\deg f_1}{2} \rceil = \lceil \frac{n+k}{2} \rceil$$

then $\deg(b_{i-1} + q'_i b_i) < \deg b_i$. So $b_0, b_1, \ldots, b_{l_1}$ is the sequence of remainders for the polynomials b_0, b_1 and still $\deg b_{l_1} > \lceil \frac{n-k}{2} \rceil$. Then $l = l_1$ and $q_i = q'_i$ for $i = 1, \ldots, l$. Because

$$R^{(f_1, g_1)}_{l(\lceil \frac{n-k}{2} \rceil)} = \begin{pmatrix} 0 & 1 \\ 1 & q_l \end{pmatrix} \ldots \begin{pmatrix} 0 & 1 \\ 1 & q_1 \end{pmatrix}$$

and

$$R^{(f,g)}_{l(\lceil \frac{n+k}{2} \rceil} = \begin{pmatrix} 0 & 1 \\ 1 & q'_l \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & q'_1 \end{pmatrix}$$

one gets $R^{(f,g)}_{l(\lceil \frac{n+k}{2} \rceil} = R^{(f_1,g_1)}_{l(\lceil \frac{n-k}{2} \rceil)}$ as stated. This proves the Theorem.

2.6.2 Half GCD

We will formulate a procedure called HGCD(HalfGCD). Given two polynomials a_0 and a_1, where $\deg a_0 = n \geq \deg a_1$, HGCD returns the matrix

$$R^{(a_0,a_1)}_{l(\lceil \frac{n}{2} \rceil)} = \begin{pmatrix} u_l & v_l \\ u_{l+1} & v_{l+1} \end{pmatrix}.$$

That is

$$\begin{pmatrix} u_l & v_l \\ u_{l+1} & v_{l+1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} a_l \\ a_{l+1} \end{pmatrix},$$

where $\deg a_l > \lceil \frac{n}{2} \rceil \geq \deg a_{l+1}$. This matrix is about what we are looking for when constructing a generating polynomial for a segment of linear recurrent sequence. In most cases $C(X) = v_{l+1}$.

HGCD

1. Denote $n = \deg a_0$. If $\deg a_1 \leq \frac{n}{2}$, then return

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

2. Let $\deg a_1 > \frac{n}{2}$, then put $m = \lceil \frac{n}{2} \rceil$. Represent

$$a_0 = b_0 X^m + c_0,$$
$$a_1 = b_0 X^m + c_1,$$

   where $\deg c_0, \deg c_1 < m$ and $\deg b_0 = n - m < m$.

3. (a) Set $R \leftarrow HGCD(b_0, b_1)$.
   (b) Compute

$$\begin{pmatrix} d \\ e \end{pmatrix} = R \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$$

   If $\deg e \leq m$, then return R and stop.
   (c) Otherwise, divide d by e with remainder. That is $d = qe + f$, where $\deg f < \deg e$.

4. Set $k = n - \deg e$ and represent

$$e = g_0 X^k + h_0,$$
$$f = g_1 X^k + h_1,$$

   with $\deg h_0, \deg h_1 < k$.

5. Compute $S \leftarrow HGCD(g_0, g_1)$ and return

$$S \begin{pmatrix} 0 & 1 \\ 1 & q \end{pmatrix} R.$$

Then stop.

Lemma 9 *1. The degrees of f and e are at most $\lceil \frac{n+m}{2} \rceil$ and the degrees of g_0 and g_1 are at most $m+1$.*

2.

$$R^{(a_0,a_1)}_{l(\lceil \frac{n}{2} \rceil)} = S \begin{pmatrix} 0 & 1 \\ 1 & q \end{pmatrix} R.$$

Proof. By Theorem

$$R = R^{(a_0,a_1)}_{l(\lceil \frac{n+m}{2} \rceil)}.$$

So denoting $l(\lceil \frac{n+m}{2} \rceil) = l_1$ one gets

$$\begin{pmatrix} a_{l_1} \\ a_{l_1+1} \end{pmatrix} = R \begin{pmatrix} a_0 \\ a_1 \end{pmatrix},$$

where $a_{l_1+1} = e$. By the definition of the HGCD we see that

$$\deg a_{l_1} > \lceil \frac{n+m}{2} \rceil \geq \deg a_{l_1+1}.$$

That is $\deg f < \deg e \leq \lceil \frac{n+m}{2} \rceil$.

Now $\deg g_0 = \deg e - k = 2 \deg e - n \leq 2 \lceil \frac{n+m}{2} \rceil - n \leq m+1$. This proves the first statement of the Lemma.

In order to prove the second statement we put $l = l(\lceil \frac{n}{2} \rceil)$ and summarize main steps of the Algorithm:

$$\begin{aligned} R \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} d \\ e \end{pmatrix} &= \begin{pmatrix} a_{l_1} \\ a_{l_1+1} \end{pmatrix}, \\ \begin{pmatrix} 0 & 1 \\ 1 & q \end{pmatrix} \begin{pmatrix} d \\ e \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix} &= \begin{pmatrix} a_{l_1+1} \\ a_{l_1+2} \end{pmatrix}, \\ S \begin{pmatrix} e \\ f \end{pmatrix} &= \begin{pmatrix} a_l \\ a_{l+1} \end{pmatrix}, \end{aligned}$$

because

$$S = R^{(g_0,g_1)}_{l(\lceil \frac{\deg e - k}{2} \rceil)} = R^{(e,f)}_{l(\lceil \frac{n}{2} \rceil)}$$

by Theorem. This proves the Lemma because

$$a_0, a_1, \ldots, d, e, f, \ldots, a_l, a_{l+1}$$

is the sequence of remainders in the Euclidean Algorithm applied to a_0 and a_1.

An application of the HGCD is enough to produce a generating polynomial for the sequence s^{2L} of linear complexity L. By definition

$$HGCD(X^{2L+1}, S(X)) = \begin{pmatrix} u_l & v_l \\ u_{l+1} & v_{l+1} \end{pmatrix},$$

where $l = l(\lceil \frac{2L+1}{2} \rceil) = l(L+1)$ so that

$$\deg a_l > L + 1 \geq \deg a_{l+1}.$$

One computes a_l and a_{l+1}. There are two cases.

1. Let $\deg a_{l+1} = L + 1$, then one should compute a_{l+2} and v_{l+2}. Then $\deg a_{l+2} \leq L$ and

   $$\deg v_{l+2} \leq 2L + 1 - \deg a_{l+1} = L,$$

   so one puts $C(X) = v_{l+2}$.

2. If $\deg a_{l+1} < L + 1$ then $C(X) = v_{l+1}$.

Theorem 10 *Let $M(n)$ be the time required to multiply(divide) two polynomials of degree $\leq n$. Them the HGCD on them takes $O(M(n) \log n)$ bit operations.*

Proof. Let $T(n)$ be the time required by the HGCD. For simplicity, we only consider $n = 2^k$. Then

$$\begin{aligned} T(n) \leq 2T(\frac{n}{2}) + cM(n) \leq \\ 2(2T(\frac{n}{4}) + cM(\frac{n}{2})) + cM(n) \leq \dots \\ = O(n + \sum_{i=0}^{k} 2^i M(\frac{n}{2^i})). \end{aligned}$$

One sees $2^i M(\frac{n}{2^i}) = O(M(n))$ for a big n and any $i \geq 1$. So $T(n) = O(M(n) \log n)$ and the Theorem is proved.

With the fast Fourier Transform $M(n) = O(n \log n)$ bit operations, so the HGCD works in time $O(n \log^2 n)$.

Example. We use the previous example, that is $a_0 = X^{23}$ and $a_1 = S(X) =$

$1 + X^3 + X^4 + X^6 + X^7 + X^8 + X^{10} + X^{12} + X^{16} + X^{17} + X^{19} + X^{20} + X^{21}$.

We set $m = 12$ and represent

$$\begin{aligned} a_0 &= X^{11} X^{12}, \\ a_1 &= (1 + X^4 + X^5 + X^7 + X^8 + X^9) X^{12} \\ &+ 1 + X^3 + X^4 + X^6 + X^7 + X^8 + X^{10}. \end{aligned}$$

So $b_0 = X^{11}$ and $b_1 = 1 + X^4 + X^5 + X^7 + X^8 + X^9$. One computes

$$R = R_4^{(b_0,b_1)} = \begin{pmatrix} X & X^3+X^2+1 \\ X^2+1 & X^4+X^3+X^2 \end{pmatrix},$$

because $l(6) = 3$ as $b_3 = X^7+X^6+X^5+X^4+X^3+X^2+1$ and $b_4 = X^6+X^4+X^3+X^2$. Then

$$R \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} d \\ e \end{pmatrix},$$

where

$$\begin{aligned} d &= X^{19}+X^{18}+X^{17}+X^{16}+X^{15}+X^{13} \\ &+ X^{11}+X^{10}+X^6+X^5+X^4+X^2+1, \\ e &= X^{18}+X^{16}+X^{15}+X^{13}+X^{10}+X^5+X^4+X^3+X^2. \end{aligned}$$

Compute $q = \lfloor d/e \rfloor = X+1$ and $k = 23 - \deg e = 5$. Then represent

$$\begin{aligned} e &= X^{13}+X^{11}+X^{10}+X^8+X^5+1)X^5+X^4+X^3+X^2 \\ f &= d+qe = (X^{11}+1)X^5+X^4+1 \end{aligned}$$

Set $g_0 = X^{13}+X^{11}+X^{10}+X^8+X^5+1$ and $g_1 = X^{11}+1$ and compute

$$S = R_4^{(g_0,g_1)} = \begin{pmatrix} X^2+1 & X^4+X+1 \\ X^3+X^2+1 & X^5+X^4+X^3+X^2+X \end{pmatrix},$$

because $l(7) = 4$ as $g_4 = X^8+X^7+X^5+X^4+X^2+X$ and $g_5 = X^7+X^4+X+1$ for remainders in a sequence generated from g_0 and g_1. Then one computes

$$T = R_{l(12)}^{(a_0,a_1)} = S \begin{pmatrix} 0 & 1 \\ 1 & q \end{pmatrix} R =$$
$$\begin{pmatrix} X^7+X^6+X^4+X^2+X & X^9+X^7+X^2+X+1 \\ X^8+X^3+X^2+X+1 & X^{10}+X^9+X^7+X \end{pmatrix}.$$

Apply

$$T \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} X^{13}+X^{12}+X^{10}+X^9+X^8+X^7+X^3+X^2+X+1 \\ X^{12}+X^8+X^5+X^4+X \end{pmatrix}.$$

As $a_{l(12)+1}$ is of degree just 12, one should do another step. That is compute the quotient $X+1 = \lfloor \frac{a_{l(12)}}{a_{l(12)+1}} \rfloor$ and $v_{l(12)+2} = X^{11}+X^8+1$. So the generating polynomial for s^{22} is $X^{11}+X^3+1$.

2.7 Affine Approximation Attack

This is a kind of known plain-text attack. A synchronous stream cipher depicted in Figure 12 is considered. The Boolean function F and the $LFSR$ are assumed to be public knowledge, as well as other particularities about the cipher like how initialization works except the key. The Affine Approximation Attack is against the initial state of the cipher. A segment $x^N = x_0, x_1, \ldots, x_{N-1}$ of the key-stream is known for $N \geq L$. One obtains the segment from a known segment of the plain-text and related cipher-text. First, we will study the idea of the attack. Assume we have found a Boolean function g which is the best affine approximation to F. That is $g = a_1X_1 \oplus \ldots \oplus a_LX_L \oplus b$ and $\mathbf{Pr}(F = g) = p$ is maximal in all affine functions in variables $X_1, \ldots, X_L$. By definition,

$$\mathbf{Pr}(F = g) = \frac{\text{the number of } L\text{-tuples } y = y_1, \ldots, y_L \text{ such that F(y)=g(y)}}{2^L}.$$

In practice, the function F doesn't depend on all L variables, but the last equality holds anyway. One can use any affine approximation, but the result is better if the probability is higher.

Example.

X_1	X_2	X_3	F	X_1	$X_2 \oplus X_3$	$X_2 \oplus X_3 \oplus 1$
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	1	0	1	0
0	1	1	1	0	0	1
1	0	0	1	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	0	1
Pr				1/2	1/4	3/4

So $Pr(F = X_2 \oplus X_3 \oplus 1) = 3/4$ and $X_2 \oplus X_3 \oplus 1$ is the best among $X_1, X_2 \oplus X_3, X_2 \oplus X_3 \oplus 1$. In fact, this is the best among all affine functions in three variables as we will see. Consider now an output sequence u_j of the LFSR with the Boolean function g instead of F. That is

$$u_j = a_1s_{L+j-1} \oplus a_2s_{L+j-2} \oplus \ldots \oplus a_Ls_j \oplus b \tag{8}$$

and we write

$$v_j = u_j \oplus x_j, \quad 0 \leq j \leq L-1.$$

We show how to solve those equations, find u_j and therefore the state by solving equations (8). We now count probabilities

$$\begin{aligned} \mathbf{Pr}(v_j = 0) &= \mathbf{Pr}(u_j = x_j) \approx p, \\ \mathbf{Pr}(v_j = 1) &= \mathbf{Pr}(u_j \neq x_j) \approx q = 1 - p. \end{aligned}$$

We write the equation for u_j in matrix form:

$$u_j = \begin{pmatrix} s_j & s_{j+1} & \dots & s_{j+L-1} \end{pmatrix} \begin{pmatrix} a_L \\ a_{L-1} \\ \dots \\ a_1 \end{pmatrix} + b = \tag{9}$$

$$\begin{pmatrix} s_0 & s_1 & \dots & s_{L-1} \end{pmatrix} A^j \begin{pmatrix} a_L \\ a_{L-1} \\ \dots \\ a_1 \end{pmatrix} + b,$$

where A is a matrix which implements changing of LFSR states, the companion matrix of the generating polynomial. That is

$$A = \begin{pmatrix} 0 & \dots & 0 & c_L \\ 1 & \dots & 0 & c_{L-1} \\ \dots & & & \dots \\ 0 & \dots & 1 & c_1 \end{pmatrix}.$$

Generally, the system $v_j = u_j + x_j$ has 2^L solutions. Fortunately, one knows that $\mathbf{Pr}(v_j = 1) = q$ is a relatively small number, as p is relatively big. So the weight of the vector

$$v = (v_0, v_1, \dots, v_{L-1})$$

is about qL on the average, see Theorem 12, and this number is relatively small. Therefore, the idea of the method is in trying all vectors v of weight approximately qL, find u_j and solve the system (9) for $s_0, s_1, \dots, s_{L-1}$, then form a guess on the initial state and then check it with other bits of the key-stream or decrypt a segment of the cipher-text and look if it is sensible or not. Formally,

1. Fix a number d the choice of which we discuss below. Try all vectors v of weight r for $|r - qL| \le d$. If d is small the probability of a mistake(to miss a true v) increases while for a big d the number of trials may be big.

2. Solve the system for $s_0, s_1, \dots, s_{L-1}$. Form a guess.

3. Check the guess by producing a sequence with $s_0, s_1, \dots, s_{L-1}$ as an initial state of the LFSR and compare with other bits of the key-stream.

Remark 2 *1. In order to make checking faster one may take $L_1 > L$ equations and try vectors v of length L_1 and weight about qL_1.*

2. Complexity of the attack equals the number of trials times the complexity of solving the linear system in L(or L_1) variables.

3. One misses the true vector v if $|weight(v) - qL| > d$.

Therefore, our plan is to address the following problems.

1. Find the best affine approximation for a given Boolean function.
2. Consider an example.
3. Estimate the probability of missing the correct v.
4. Estimate the number of trials.

2.7.1 Walsh-Hadamard Coefficients

We denote $X = X_1 2^{n-1} + X_2 2^{n-2} + \ldots + X_n$ for a set of Boolean variables $X_1, X_2, \ldots, X_n$ and $a = a_1 2^{n-1} + a_2 2^{n-2} + \ldots + a_n$ for a binary n-string $a_1, a_2, \ldots, a_n$. Then let

$$(X, a) = a_1 X_1 + a_2 X_2 + \ldots + a_n X_n$$

be a dot product of two vectors a and X. Let F be a Boolean function in variables $X_1, X_2, \ldots, X_n$. By $N_{0,a}$ we denote the number of $x \in V_n$ such that $F(x) = (x, a)$ and $N_{1,a}$ denotes the number of $x \in V_n$ such that $F(x) = (x, a) \oplus 1$. Then

$$p_a = Pr(F(x) = (x, a)) = \frac{N_{0,a}}{2^n},$$
$$q_a = Pr(F(x) = (x, a) \oplus 1) = \frac{N_{1,a}}{2^n} = 1 - \frac{N_{0,a}}{2^n},$$

because $N_{0,a} + N_{1,a} = 2^n$ for all a. In order to calculate the probabilities one finds

$$W_a = \frac{1}{2^n} \sum_{x \in V_n} (-1)^{F(x)+(x,a)} =$$
$$\frac{N_{0,a} - N_{1,a}}{2^n} = \frac{2N_{0,a} - 2^n}{2^n} = 2p_a - 1.$$

This is because

$$(-1)^{F(x)+(x,a)} = \begin{cases} 1, & F(x) = (x, a); \\ -1, & F(x) \neq (x, a). \end{cases}$$

In other words,

$$p_a = \frac{1 + W_a}{2} \quad \text{and} \quad q_a = 1 - p_a = \frac{1 - W_a}{2}.$$

The numbers $W_0, W_1, \ldots, W_{2^{n-1}}$ are called Walsh-Hadamard coefficients for the Boolean function F. For convenience we rewrite the expression for W_a in a matrix form

$$W = \frac{(-1)^F H_n}{2^n},$$

where $W = (W_0, W_1, \ldots, W_{2^{n-1}})$ and

$$(-1)^F = ((-1)^{F(0)}, (-1)^{F(1)}, \ldots, (-1)^{F(2^n-1)}).$$

Then $H_n = ((-1)^{(x,a)})_{x,a}$ is a matrix of size $2^n \times 2^n$ with rows and columns numbered by x and a ordered lexicographically and with entries $(-1)^{(x,a)}$. For example, $H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Theorem 11

$$H_n = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{[n]}.$$

Proof. We will prove the statement by induction on n. For $n = 1$ the Theorem is true. Assume that it is true when the number of variables is less than n and prove it correctness for n variables. One writes $x = (x_1, \bar{x})$, where $\bar{x} = (x_2, x_2, \ldots, x_n)$, and $a = (a_1, \bar{a})$, where $\bar{a} = (a_2, a_2, \ldots, a_n)$. Then we represent the matrix H_n with its sub-matrices:

$$H_n = \begin{pmatrix} (-1)^{(\bar{x},\bar{a})+0\times 0} & (-1)^{(\bar{x},\bar{a})+0\times 1} \\ (-1)^{(\bar{x},\bar{a})+1\times 0} & (-1)^{(\bar{x},\bar{a})+1\times 1} \end{pmatrix} = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes H_{n-1} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{[n]}.$$

This finishes the proof.

From the above we see that in order to compute the vector W one applies the linear operator H_n to the vector $\frac{(-1)^F}{2^n}$. Generally, it takes $O(2^{2n})$ arithmetic operations with integer numbers bounded by 2^n in modulus. We will show that it can be done in $O(n2^n)$ additions and subtractions with integer numbers $\leq 2^n$ in absolute value.

Fast Algorithm to compute W

input Boolean function F defined by the vector of its values.

output Vector W of Walsh-Hadamard coefficients for F.

We introduce first some auxiliary vectors $W_{k,a}$ to be integer 2^k-strings, where $0 \leq a \leq 2^{n-k} - 1$ for all $0 \leq k \leq n$.

1. Initialize $W_{0,a} = (-1)^{F(a)}$ for $0 \leq a \leq 2^n - 1$.
2. For $0 \leq k \leq n-1$ compute
$$W_{k+1,b} = (W_{k,2b} + W_{k,2b+1}, W_{k,2b} - W_{k,2b+1})$$
where $0 \leq b \leq 2^{n-k-1} - 1$.
3. Return $W = \frac{W_{n,0}}{2^n}$.

The proof of the correctness of the algorithm repeats the proof of Theorem 11. To accomplish the algorithm takes $O(2^n)$ arithmetic operations at each step, because computing $W_{k+1,b}$ costs $O(2^{k+1})$ operations and one should compute 2^{n-k-1} of them. So the complexity of the whole algorithm is $O(n2^n)$ operations.

Example. Let a Boolean function F in three variables be given by a string of its values $0, 0, 1, 1, 1, 0, 0, 1$. The application of the above algorithm is represented by the table:

W_0	1	1	−1	−1	−1	1	1	−1
W_1	2	0	−2	0	0	−2	0	2
W_2	0	0	4	0	0	0	0	−4
W_3	0	0	4	−4	0	0	4	4
W	0	0	1/2	−1/2	0	0	1/2	1/2.

Here W_k stands for a concatenation of vectors $W_{k,a}$. We now find the probabilities vector

$$\frac{\bar{1}+W}{2} = (\frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}).$$

Entries of this vector numbered $2, 6, 7$ are $\frac{3}{4}$, entry at 3 is $\frac{1}{4}$ and other entries are $\frac{1}{2}$. This implies that Boolean functions $X_2, X_1 \oplus X_2, X_1 \oplus X_2 \oplus X_3$ have the probability $\frac{3}{4}$ of coincidence with F, Boolean function $X_2 \oplus X_3$ has the probability $\frac{1}{4}$ and the probability is $\frac{1}{2}$ for all other linear functions.

Finding the best affine approximation. To find the best affine approximation to F, compute the vector W of its Walsh-Hadamard coefficients. Take a such that W_a is maximal in absolute value. Then

1. $W_a > 0$, put $g = (a, X)$.
2. $W_a < 0$, put $g = (a, X) \oplus 1$.

The probability of approximation is computed as

$$Pr(g = F) = \frac{1 + |W_a|}{2}.$$

In the above example $|W_2| = |W_3| = |W_6| = |W_7| = \frac{1}{2}$. So 4 functions are the best affine approximations:

$$X_2,\ X_2 + X_3 + 1,\ X_1 + X_2,\ X_1 + X_2 + X_3$$

and the probability is $\frac{3}{4}$.

2.7.2 Piling-up Lemma

The algorithm for finding the best affine approximation is fine when the number of variables doesn't exceed some bound determined by our computational power, e.g. $n \leq 30$. But there is often a need to handle functions with a larger number of variables. Look at the key-stream generator.

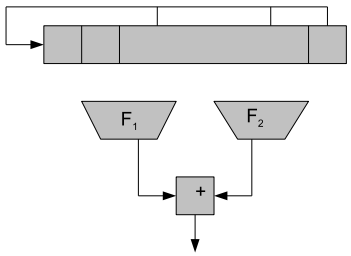

Figure 13: Example

We will answer how to find the best affine approximation to $F_1 \oplus F_2$, minding that F_1 and F_2 depend on different variables and can be handled separately. First, we will prove a so called piling-up Lemma. Let X_1 and X_2 be two Boolean variables such that

1. $Pr(X_i = 0) = p_i = \frac{1}{2} + \delta_i$, where $0 \leq |\delta_i| \leq 1/2$.
2. Variables X_1 and X_2 are independent. This means that

$$Pr(X_1 = a, X_2 = b) = Pr(X_1 = a)Pr(X_2 = b)$$

for all binary a, b. E.g. variables X_1 and $X_2 = X_1 + 1$ are dependent. Really, one computes $Pr(X_1 = 0, X_2 = 0) = 0$ but $Pr(X_1 = 0)Pr(X_2 = 0) = p_1(1 - p_1)$.

Lemma 10 *$Pr(X_1 \oplus X_2 = 0) = \frac{1}{2} + \delta$, where $\delta = 2\delta_1\delta_2$.*

Proof. We count

$$\begin{aligned} Pr(X_1 \oplus X_2 = 0) &= Pr(X_1 = 0, X_2 = 0) + Pr(X_1 = 1, X_2 = 1) \\ &= Pr(X_1 = 0)Pr(X_2 = 0) + Pr(X_1 = 1)Pr(X_2 = 1) \\ &= p_1p_2 + (1 - p_1)(1 - p_2) \\ &= (\frac{1}{2} + \delta_1)(\frac{1}{2} + \delta_2) + (\frac{1}{2} - \delta_1)(\frac{1}{2} - \delta_2) = \frac{1}{2} + 2\delta_1\delta_2. \end{aligned}$$

Let $x_1, \ldots, x_k, x_{k+1}, \ldots, x_n$ be Boolean variables and

$$F_1 = F_1(x_1, \ldots, x_k), \quad F_2 = F_2(x_{k+1}, \ldots, x_n).$$

Lemma 11 *If variables x_i have independent distributions then*

$$X_1 = F_1(x_1, \ldots, x_k), \quad X_2 = F_2(x_{k+1}, \ldots, x_n)$$

are independent variables.

The proof is obvious.

Corollary 6 *Let l_1, l_2 be any functions in variables $x_1, \ldots, x_k$ and $x_{k+1}, \ldots, x_n$ such that $Pr(F_i = l_i) = \frac{1}{2} + \delta_i$. Then $Pr(F_1 \oplus F_2 = l_1 \oplus l_2) = \frac{1}{2} + 2\delta_1\delta_2$.*

Corollary 7 *The function l_i or $l_i \oplus 1$ is the best affine approximation to F_i if and only if $l = l_1 \oplus l_2$ is the best affine approximation to $F = F_1 \oplus F_2$.*

Proof. We know that $Pr(F_i = l_i) = \frac{1}{2} + \delta_i$. If l_i is the best affine approximation the number δ_i is positive and maximal in the set of all affine functions in variables of F_i. So $\delta_1\delta_2$ is positive and maximal in the set of all affine functions in variables of F. This proves the statement.

2.8 Example.

We present an example of the Affine Approximation Attack. The key-stream generator is depicted in Figure 14. The function F is as in the above exam-

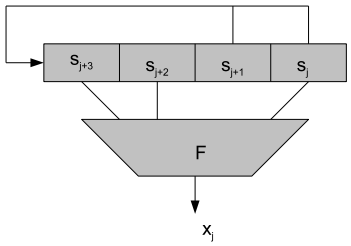

Figure 14: Example.

ple. The segment of the key-stream: $0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, \ldots$ is given. We already know that the best affine approximation to F is x_2 with $3/4$ as the probability of approximation. So the key-stream x_j is correlated with the output u_j of the generator in Figure 15. So

$$v_j = x_j \oplus u_j = \begin{cases} 0, & \text{Pr=3/4;} \\ 1, & \text{Pr=1/4.} \end{cases}$$

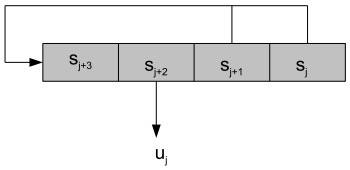

Figure 15: Example.

Then one takes $L_1 = 5$ and defines that

$$\begin{aligned} u_0 &= s_2, \\ u_1 &= s_3, \\ u_2 &= s_0 \oplus s_1, \\ u_3 &= s_1 \oplus s_2, \\ u_4 &= s_2 \oplus s_3. \end{aligned}$$

The related system of linear equations is

$$\begin{aligned} v_0 &= s_2 \oplus x_0, \\ v_1 &= s_3 \oplus x_1, \\ v_2 &= s_0 \oplus s_1 \oplus x_2, \\ v_3 &= s_1 \oplus s_2 \oplus x_3, \\ v_4 &= s_2 \oplus s_3 \oplus x_4. \end{aligned}$$

The weight of the vector $(v_0, \ldots, v_4)$ should be approximately $5/4$, so one tries vectors of weight $0, 1, 2$. First, $v = (0, 0, 0, 0, 0)$, then is

$$\begin{aligned} 0 &= s_2, \\ 0 &= s_3 \oplus 1, \\ 0 &= s_0 \oplus s_1 \oplus 1, \\ 0 &= s_1 \oplus s_2, \\ 0 &= s_2 \oplus s_3. \end{aligned}$$

The system has no any solution. Hence the guess was wrong. One tries $v = (1, 0, 0, 0, 0)$ and finds the only solution $(s_0, s_1, s_2, s_3) = (0, 1, 1, 1)$. By using this as an initial state we produce a key-stream $0, 1, 1, 0, 0, 1, 1, 0, 0, 0, \ldots$ which differs from what we were given, so this guess was wrong. Finally, we try

$v = (0, 1, 0, 0, 0)$ and solve the system of equations

$$\begin{aligned} 0 &= s_2, \\ 1 &= s_3 \oplus 1, \\ 0 &= s_0 \oplus s_1 \oplus 1, \\ 0 &= s_1 \oplus s_2, \\ 0 &= s_2 \oplus s_3. \end{aligned}$$

to find the only solution $(s_0, s_1, s_2, s_3) = (1, 0, 0, 0)$. This is a true initial state because the produced key-stream $0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, \ldots$ is identical to what was given.

The algorithm is based on the fact that the weight of the true vector v is about $5/4$. But it may be much bigger or less with some probability and it would result in increasing of the computational cost to find it. We will develop some theory necessary for estimating this probability.

2.8.1 Bernoulli trials

Let $v_1, v_2, \ldots, v_n, \ldots$ be a sequence of independent random variables equally distributed such that

$$v_i = \begin{cases} 0, & \text{with the probability } p; \\ 1, & \text{with the probability } q. \end{cases}$$

So $p + q = 1$. The independence means that

$$Pr(v_1 = a_1, \ldots, v_n = a_n) = Pr(v_1 = a_1) \ldots Pr(v_n = a_n)$$

for any n and any binary n-string $a_1, \ldots, a_n$. This sequence is called a sequence of Bernoulli trials. The outcome of the sequence of Bernoulli trials is a sequence of 0 and 1. Historically, an outcome of v_i is called "success" if $v_i = 1$ and "failure" if $v_i = 0$. We are interested in the number of successes after n trials $S_n = \sum_{i=1}^{n} v_i$. In other words, S_n is the weight of the vector $v = (v_1, \ldots, v_n)$. S_n is a random variable.

Theorem 12 *1. The mean(expected) value of S_n is qn.*

2. (de Moivre-Laplace Limit Theorem) For every fixed $a \leq b$

$$Pr(a \leq \frac{S_n - nq}{\sqrt{npq}} \leq b) \to \Phi(b) - \Phi(a),$$

as $n \to \infty$. Here $\Phi(b)$ is the normal distribution function.

3. (Chernoff's inequalities)

$$Pr(|S_n - nq| > a) \leq 2e^{-2a^2/n},$$

$$Pr(S_n - nq > a) \leq e^{-2a^2/n}.$$

We will prove the first statement of the Theorem by using two different approaches.

1. The probability of one particular string $v_1, \ldots, v_n$ of weight i is $q^i p^{n-i}$ and the number of such strings is $\binom{n}{i}$. So

$$Pr(S_n = i) = \binom{n}{i} q^i p^{n-i}.$$

By the definition of expectation,

$$ES_n = \sum_{i=0}^{n} i \binom{n}{i} q^i p^{n-i} =$$

$$= p^n \frac{q}{p} \sum_{i=0}^{n} i \binom{n}{i} (\frac{q}{p})^{i-1}.$$

One easily checks that

$$n(1+x)^{n-1} = \sum_{i=0}^{n} i\binom{n}{i}x^{i-1}$$

for any x. So

$$ES_n = p^n \frac{q}{p} n(1+\frac{q}{p})^{n-1} = nq.$$

2. The second proof is based on the equality $E(u+v) = Eu + Ev$ which holds for any random variables u and v. We have

$$ES_n = E(\sum_{i=1}^{n} v_i) = \sum_{i=1}^{n} Ev_i = nq,$$

because $Ev_i = 1 \times q + 0 \times p = q$.

Definition 7 *The function*

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$$

is called the normal density function. Its integral

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\frac{y^2}{2})dy$$

is the normal distribution function.

The graph of $\phi(x)$ is a symmetric bell-shaped curve. The domain bounded by

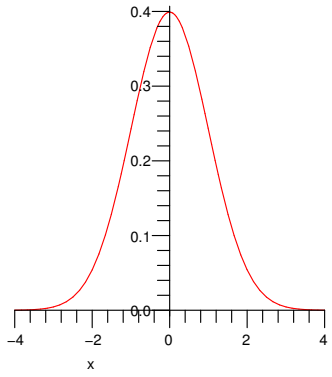

Figure 16: Normal density function.

$\phi(x)$ and the x-axis has unit area, so $\Phi(x)$ is a probability distribution function. Because $\phi(x)$ is symmetric, the equality $\Phi(-x) = 1 - \Phi(x)$ holds for any real number x.

We return to the affine approximation attack. Strictly speaking we want to solve the following problem. Given $0 < \epsilon < 1$ find $d = d_\epsilon$ such that

$$Pr(|\sum_{i=0}^{L-1} v_i - qL| > d) \leq \epsilon. \tag{10}$$

First we give the solution holding for large L with de Moivre-Laplace Theorem. We put $d = x\sqrt{Lpq}$ for some $x = x_\epsilon$ we are to determine. One rewrites

$$Pr(|\sum_{i=0}^{L-1} v_i - qL| > d) =$$
$$Pr(\frac{|\sum_{i=0}^{L-1} v_i - qL|}{\sqrt{Lpq}} > x).$$

The last probability tends to

$$1 - (\Phi(x) - \Phi(-x)) = 2(1 - \Phi(x))$$

as L tends to infinity by de Moivre-Laplace Theorem. We now find x such that $2(1 - \Phi(x)) \leq \epsilon$. If we take a bigger x the probability $Pr(|\sum_{i=0}^{L-1} v_i - qL| > d)$ will be smaller, but the number of trials will increase. So we should take x as small as possible that is x should be the solution of the equation $2(1-\Phi(x)) = \epsilon$. In other words x is a root of the equation $\Phi(x) = 1-\epsilon/2$. One uses a table from the book W.Feller, An Introduction to Probability Theory, volume 1 presenting explicit values of the normal distribution function. E.g.

ϵ	$1 - \epsilon/2$	x
0.1	0.95	1.65
0.05	0.975	1.95
0.02	0.99	2.35
0.01	0.995	2.6
0.005	0.9975	2.82
...		.

For $\epsilon = 0.01$ one takes $x \approx 2.613$. That is if we want the failure probability ϵ of the algorithm to be ≤ 0.01, then we should try all vectors $v = (v_j, \ldots, v_{j+L-1})$ of weight r, where

$$qL - 2.6\sqrt{Lpq} \leq r \leq qL + 2.6\sqrt{Lpq}. \tag{11}$$

That is $d = 2.6\sqrt{Lpq}$. This interval is correct for all large enough L and fixed reasonably small q. E.g. for $L = 200$ this is true for all $q \geq 1/70$.

We now use Chernoff's bound. For $2e^{-2d^2/L} < \epsilon$ and therefore

$$d > \sqrt{L \ln(\frac{2}{\epsilon})/2}$$

Theorem 12 guarantees (10). E.g. for $\epsilon = 0.01$ we get $d \geq 1.62\sqrt{L}$ and therefore

$$qL - 1.62\sqrt{L} \leq r \leq qL + 1.62\sqrt{L}. \tag{12}$$

This is true for any L and q but the interval (11) is sharper. This is because $qp \leq 1/4$ for any $q \geq 0$, $p \geq 0$ and $p + q = 1$.

So that for $L = 200$ and $q = 1/20$ we get $1 \leq r \leq 19$ by (11) with the number of trials

$$\sum_{i=1}^{19} \binom{200}{i} \approx 1.9 \times 10^{26},$$

whilst the brute force attack takes time $2^{200} \approx 1.6 \times 10^{60}$. Asymptotically as L tends to ∞ and fixed x,

$$\sum_{i=0}^{qL+x\sqrt{Lpq}} \binom{L}{i} = \sum_{i=0}^{qL(1+o(1))} \binom{L}{i} = 2^{H(q)L(1+o(1))},$$

where $H(q) = -q \log_2 q - p \log_2 p$ is the binary entropy function, and because

$$\binom{L}{qL(1+o(1))} = 2^{H(q)L(1+o(1))}$$

with the Stirling approximation of the factorial function. For any $q < 1/2$ we have $H(q) < 1$. So the affine approximation attack with a small probability of failure is always better than the brute force algorithm in finding the initial state.

2.9 Bent Functions

We have seen the bigger some Walsh-Hadamard coefficients of F the less the security of the key-stream generator. So when designing a cipher it looks natural to take F with Walsh-Hadamard coefficients as small as possible. We will investigate how small they all may be.

Lemma 12

$$\sum_{x \in V_n} (-1)^{(x,a)} = \begin{cases} 0, & a \neq 0; \\ 2^n, & a = 0. \end{cases}$$

The proof is obvious as when $a \neq 0$ the function (a, x) has the same number 1 and 0 values. We compute

$$\sum_{a \in V_n} W_a^2 = \sum_{a \in V_n} (\frac{1}{2^n} \sum_{x \in V_n} (-1)^{F(x)+(x,a)})^2 =$$
$$\sum_{a \in V_n} \frac{1}{2^{2n}} \sum_{x,y \in V_n} (-1)^{F(x)+(x,a)+F(y)+(y,a)} =$$
$$\sum_{x,y \in V_n} \frac{1}{2^{2n}} (-1)^{F(x)+F(y)} \sum_{a \in V_n} (-1)^{(x+y,a)} =$$
$$\sum_{x=y} \frac{1}{2^{2n}} (-1)^{F(x)+F(y)} 2^n = \sum_{x=y} \frac{1}{2^{2n}} 2^n = 1.$$

Lemma 13

$$\max_a |W_a| \geq \frac{1}{2^{n/2}}.$$

Proof. Let on the contrary $\max_a |W_a| < \frac{1}{2^{n/2}}$. We then compute

$$\sum_{a \in V_n} W_a^2 < \sum_{a \in V_n} (\frac{1}{2^{n/2}})^2 = \sum_{a \in V_n} \frac{1}{2^n} = 1.$$

This contradicts the above identity.

The last Lemma implies that for any Boolean function F in n variables there exists an affine function g such that

$$Pr(F = g) \geq \frac{1 + \frac{1}{2^{n/2}}}{2} = \frac{1}{2} + \frac{1}{2^{n/2+1}}.$$

Definition 8 *A Boolean function F in n variables is called bent if $|W_a| = \frac{1}{2^{n/2}}$ for all its Walsh-Hadamard coefficients W_a.*

Properties of bent-functions. Let Boolean function F in n variables be bent. Then

1. n is even. Really, by the definition of Walsh-Hadamard coefficients, W_a are rational numbers, on the other hand $|W_a| = \frac{1}{2^{n/2}}$. It may only be possible when n is even.

2. The number of 0 values of F is $2^{n-1} \pm 2^{n/2-1}$ and the number of 1 values is $2^{n-1} \mp 2^{n/2-1}$. As $N_{0,0}$ is the number of 0 values and

$$W_0 = \frac{2N_{0,0}}{2^n} - 1 = \pm\frac{1}{2^{n/2}},$$

one gets $N_{0,0} = 2^{n-1} \pm 2^{n/2-1}$.

We will continue with bent-functions for a while. Our goal is to determine some classes of bent-functions for any even number of variables n and in so doing prove their existence.

Lemma 14 *Let F be a Boolean function in n variables and W_a be its Walsh-Hadamard coefficients. Then*

$$(-1)^{F(x)} = \sum_{a \in V_n} W_a (-1)^{(a,x)}.$$

Proof. We know that

$$W_a = \frac{1}{2^n} \sum_{y \in V_n} (-1)^{F(y)+(y,a)}.$$

Now compute

$$\sum_{a \in V_n} W_a (-1)^{(x,a)} = \sum_{a \in V_n} (\frac{1}{2^n} \sum_{y \in V_n} (-1)^{F(y)+(y,a)})(-1)^{(x,a)} =$$
$$\sum_{y \in V_n} \frac{1}{2^n} (-1)^{F(y)} \sum_{a \in V_n} (-1)^{(x,a)+(y,a)} = \frac{1}{2^n}(-1)^{F(x)} 2^n = (-1)^{F(x)},$$

by Lemma 12. This proves the statement.

Theorem 13 *A Boolean function F is bent if and only if for any nonzero $a = (a_1, \ldots, a_n)$ the Boolean function*

$$F(X) + F(X + a) = F(X_1, \ldots, X_n) + F(X_1 + a_1, \ldots, X_n + a_2)$$

is balanced(has the equal number of 1 and 0 values).

Proof. We observe that $F(X) + F(X + a)$ is balanced if and only if

$$\sum_x (-1)^{F(X)+F(X+a)} = 0.$$

As $(-1)^{F(x)} = \sum_a W_a (-1)^{(a,x)}$, one gets

$$\begin{aligned}
\sum_x (-1)^{F(X)+F(X+a)} &= \sum_x (\sum_b W_b (-1)^{(b,x)})(\sum_c W_c (-1)^{(c,x+a)}) \\
&= \sum_{b,c} W_b W_c (-1)^{(c,a)} \sum_x (-1)^{(b+c,x)} \\
&= 2^n \sum_b W_b^2 (-1)^{(b,a)}.
\end{aligned}$$

Therefore $F(X)+F(X+a)$ is balanced for all non zero a if and only if

$$\sum_x (-1)^{F(X)+F(X+a)} = 0$$

the latter if and only if

$$\begin{aligned} 2^n \sum_{b\in V_n} W_b^2 (-1)^{(b,a)} &= 0, \quad a \neq 0 \\ 2^n \sum_{b\in V_n} W_b^2 &= 2^n. \end{aligned} \tag{13}$$

This is a system of linear equations in $W_0^2, \ldots, W_{2^n-1}^2$. The matrix of the system is $H_n = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{[n]}$ of nonzero determinant. The system has only one solution $W_b^2 = \frac{1}{2^n}$. Therefore the function $F(x)+F(x+a)$ for every nonzero a is balanced if and only if $W_b^2 = \frac{1}{2^n}$ for all b, that is F is bent. The theorem is proved.

Theorem 14 *Let n be an even natural number. Then the Boolean function*

$$F = X_1X_2 + X_2X_3 + \ldots X_{n-1}X_n$$

is bent.

Proof. One computes

$$\begin{aligned} & F(X)+F(X+a) = X_1X_2 + X_2X_3 + \ldots X_{n-1}X_n \\ + \quad & (X_1+a_1)(X_2+a_2) + (X_2+a_2)(X_3+a_3) + \ldots + (X_{n-1}+a_{n-1})(X_n+a_n) \\ = \quad & a_2X_1 + (a_1+a_3)X_2 + (a_2+a_4)X_3 + \ldots + (a_{n-2}+a_n)X_{n-1} + a_{n-1}X_n + b, \end{aligned}$$

where b is a constant. Let $a \neq 0$. Then all coefficients of the last expression in X_i can't be zeros. Really,

$$a_2 = 0, \quad a_1 + a_3 = 0, \quad a_2 + a_4 = 0, \ldots, a_{n-2} + a_n = 0, a_{n-1} = 0$$

would imply $a_1 = a_2 = \ldots a_n = 0$, because n is even. So $F(X)+F(X+a)$ is a non-constant affine function, it is balanced. This proves the statement.

Example. Let $F = X_1X_2 + X_2X_3 + X_3X_4$. The function F may be given by a string of its values $0,0,0,1,0,0,1,0,0,0,0,1,1,1,0,1$. Then one applies the algorithm computing the Walsh-Hadamard coefficients and gets

$$\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}.$$

So this is really a bent function.

Theorem 15 *Let $n = 2k$. Then the Boolean function*

$$\begin{aligned} & F(X_1, \ldots, X_k, X_{k+1}, \ldots, X_n) \\ = \quad & X_1X_{k+1} + X_2X_{k+2} + \ldots + X_kX_n + h(X_{k+1}, \ldots, X_n) \end{aligned}$$

is bent for any Boolean function h.

Proof. Let $X = (X_1, \ldots, X_k)$ and $Y = (X_{k+1}, \ldots, X_n)$. Then for a nonzero vector $(a, b) = (a_1, \ldots, a_k, b_{k+1}, \ldots, b_n)$ in variables (X, Y) we find

$$\begin{aligned} F(X+a, Y+b) + F(X,Y) = \\ (X_1+a_1)(X_{k+1}+b_1) + \ldots + (X_k+a_k)(X_n+b_n) + h(Y+b) + \\ X_1X_{k+1} + X_2X_{k+2} + \ldots + X_kX_n + h(Y) = \\ a_1X_{k+1} + b_1X_1 + \ldots + a_kX_n + b_kX_k + h(Y) + h(Y+b) + d, \end{aligned}$$

where d is a binary digit. There are two cases

1. Let $b \neq 0$, for instance, $b_1 \neq 0$. Then $F(X+a, Y+b) + F(X,Y) = X_1 + g(X_2, \ldots, X_n)$ is balanced. Really, $|g|$ be the weight of g, the number of 1 values. Then the weight of $X_1 + g$ equals $|g| + |g+1| = 2^{n-1}$.

2. Let $b = 0$ now. Then $F(X+a, Y+b) + F(X,Y) = a_1X_{k+1} + \ldots + a_kX_n$. When $a \neq 0$ this is a nonconstant linear function which is balanced.

This finishes the proof.

The following theorem, taken without proof here, gives a bound on the algebraic degree of bent functions.

Theorem 16 *Let F be a Boolean bent function in n variables. Then the algebraic degree of F doesn't exceed $\frac{n}{2}$.*

Problem 9 *Is it a good idea to use a bent function in a stream cipher presented in Figure 12? Give pro and contra.*

2.10 Fast Correlation Attack

This is a variant of the Affine Approximation Attack. We consider a key-stream generator like that in Figure 12. The generating polynomial of the LFSR $X^L + c_1X^{L-1} + \ldots + c_L$ and the Boolean function $F(X_1, \ldots, X_L)$ are supposed to be known. The task is given a segment of the key-stream $x^N = x_0, x_1, \ldots, x_{N-1}$, find the initial state $s_{L-1}, \ldots, s_1, s_0$. One obtains the segment from a known segment of the plain-text and related cipher-text. First, we will study the idea of the Fast Correlation Attack. Assume we have found a good(probably the best) affine approximation $g = a_1X_1 + \ldots + a_LX_L + b$ to the function F. Let $Pr(F = g) = p > \frac{1}{2}$. Let

$$u_j = a_1s_{j+L-1} + \ldots + a_Ls_j + b.$$

The assumption implies $Pr(x_k = u_k) = p$. This is a basis for an approximation attack.

We can assume that $b = 0$, because instead of F one takes $F + b$ and instead of x^N one considers $x_0 + b, x_1 + b, \ldots, x_{N-1} + b$. Therefore, g can be taken a linear Boolean function and so $u_j = a_1s_{j+L-1} + \ldots + a_Ls_j$.

Idea of the Attack. Initially, we have $p = Pr(x_k = u_k)$. We will observe that given quite a long key-stream new probability $p_k = Pr(x_k = u_k|\text{key-stream})$ may be computed, and this probability depends on the index k and may be $p_k \geq p$. One takes $L_1 \geq L$ of indexes k, say $k \in \{j_1, \ldots, j_{L_1}\}$, with the largest probabilities p_k. A system of linear equations coming from

$$\begin{aligned} v_{j_1} &= x_{j_1} + u_{j_1}, \\ &\ldots \\ v_{j_{L_1}} &= x_{j_{L_1}} + u_{j_{L_1}} \end{aligned}$$

is considered. Let $p^* = \min_{k \in \{j_1, \ldots, j_{L_1}\}} p_k$ and $q^* = 1 - p^*$. Then the probability that $v_{j_i} = 1$ is $\leq q^*$ which implies that the weight of the vector $v = (v_{j_1}, \ldots, v_{j_{L_1}})$ is q^*L_1 on the average. Therefore, one should try about

$$\sum_{i=0}^{q^*L_1(1+o(1))} \binom{L_1}{i} = 2^{H(q^*)L_1(1+o(1))}$$

such vectors. The rest as in the Affine Approximation attack above. As $q^* \leq q$, one finds $H(q^*) \leq H(q)$ and the new method generally works faster. We will see that it is really faster if the number of the LFSR taps (non principle nonzero terms of the generating polynomial) is relatively small. So this kind of attack is the most efficient for generating polynomials like $X^L + X^l + 1$. Our plan is to study

1. Sparse linear relations for LFSR's output digits.
2. Example of the method application.
3. A formula for computing new probabilities.
4. Summary of the attack.

2.10.1 Sparse Linear Relations

Let $s_0, s_1, \ldots, s_i, \ldots$ be produced by a LFSR with a generating polynomial $f(X) = X^L + c_1 X^{L-1} + \ldots + c_L$ and let t be the number of nonzero c_i. In other words, t is the number of taps of the LFSR. For instance, $t = 2$ for $X^L + X^l + 1$. We are looking for sparse linear relations between bits $s^N = s_0, s_1, \ldots, s_{N-1}$ as

$$s_{j+M} \oplus d_1 s_{j+M-1} \oplus \ldots \oplus d_M s_j = 0, \quad 0 \le j \le N - M - 1 \tag{14}$$

for some natural M and binary $d_1, \ldots, d_M$. If u_j is a linear function in $s_j, s_{j+1}, \ldots, s_{j+L-1}$, then

$$u_{j+M} \oplus d_1 u_{j+M-1} \oplus \ldots \oplus d_M u_j = 0, \quad 0 \le j \le N - M - 1 \tag{15}$$

as well. Sparsity means the number of nonzero $d_1, \ldots, d_M$ is low. From the previous theory we know that $f(X)$ should divide the polynomial $X^M + d_1 X^{M-1} + \ldots + d_M$ and vice verse any polynomial which is a multiple of $f(X)$ will produce such relations provided its degree doesn't exceed $N - 1$. We have already got one. This is $f(X)$, so $M = L$ and $d_i = c_i$ with $t + 1$ nonzero summands in all relations (14). But we need them more.

We observe that the polynomials $f(X^{2^r}) = f(X)^{2^r}$ are multiple of $f(X)$. Therefore, as long as $L2^r \le N - 1$(or $r \le \log_2 \frac{N-1}{L}$) we get useful relations of the kind (14) and (15) with $t + 1$ nonzero summands.

Example. Let a LFSR be of length 4 with the generating polynomial $f(x) = X^4 + X + 1$. So

$$\begin{aligned} s_4 \oplus s_1 \oplus s_0 &= 0, \\ s_5 \oplus s_2 \oplus s_1 &= 0, \\ s_6 \oplus s_3 \oplus s_2 &= 0, \\ s_7 \oplus s_4 \oplus s_3 &= 0, \\ s_8 \oplus s_5 \oplus s_4 &= 0, \\ s_9 \oplus s_6 \oplus s_5 &= 0, \\ &\ldots. \end{aligned}$$

Also as $f(X)^2 = X^8 + X^2 + 1$ we have

$$\begin{aligned} s_8 \oplus s_2 \oplus s_0 &= 0, \\ s_9 \oplus s_3 \oplus s_1 &= 0, \\ s_{10} \oplus s_4 \oplus s_2 &= 0, \\ s_{11} \oplus s_5 \oplus s_3 &= 0, \\ s_{12} \oplus s_6 \oplus s_4 &= 0, \\ s_{13} \oplus s_7 \oplus s_5 &= 0, \\ &\ldots. \end{aligned}$$

2.10.2 Application of the Method. Example.

We exploit the same example as above when have been studying Affine Approximation Attack, see Fig.14 and Fig.15. The segment of the key-stream $x^{12} = 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0$ is given. As previously, $v_j = x_j \oplus u_j$. We will use relations coming from two polynomials $X^4 + X + 1$ and $X^8 + X^2 + 1$. They are generating polynomials for the sequence u_j as well because u_j is a linear function of the LFSR output. For this example it is obvious because $u_j = s_{j+2}$. So with $X^4 + X + 1$ we have

$$\begin{aligned} v_4 \oplus v_1 \oplus v_0 &= x_4 \oplus x_1 \oplus x_0 \oplus u_4 \oplus u_1 \oplus u_0, \\ v_5 \oplus v_2 \oplus v_1 &= x_5 \oplus x_2 \oplus x_1 \oplus u_5 \oplus u_2 \oplus u_1, \\ &\dots . \end{aligned}$$

As $u_4 \oplus u_1 \oplus u_0 = 0, u_5 \oplus u_2 \oplus u_1 = 0, \dots$ we have

$$\begin{aligned} v_4 \oplus v_1 \oplus v_0 &= x_4 \oplus x_1 \oplus x_0, \\ v_5 \oplus v_2 \oplus v_1 &= x_5 \oplus x_2 \oplus x_1, \\ &\dots . \end{aligned}$$

Similarly, by using $X^8 + X^2 + 1$,

$$\begin{aligned} v_8 \oplus v_2 \oplus v_0 &= x_8 \oplus x_2 \oplus x_0, \\ v_9 \oplus v_3 \oplus v_1 &= x_9 \oplus x_3 \oplus x_1, \\ &\dots . \end{aligned}$$

We compute the right hand sides of the above relations for known key-stream digits and get

$$\begin{aligned} v_4 \oplus v_1 \oplus v_0 &= 1, \\ v_5 \oplus v_2 \oplus v_1 &= 0, \\ v_6 \oplus v_3 \oplus v_2 &= 0, \\ v_7 \oplus v_4 \oplus v_3 &= 0, \\ v_8 \oplus v_5 \oplus v_4 &= 1, \\ v_9 \oplus v_6 \oplus v_5 &= 0, \\ v_{10} \oplus v_7 \oplus v_6 &= 0, \\ v_{11} \oplus v_8 \oplus v_7 &= 1. \end{aligned}$$

Similarly, with $X^8 + X^2 + 1$:

$$\begin{aligned} v_8 \oplus v_2 \oplus v_0 &= 0, \\ v_9 \oplus v_3 \oplus v_1 &= 0, \\ v_{10} \oplus v_4 \oplus v_2 &= 0, \\ v_{11} \oplus v_5 \oplus v_3 &= 0. \end{aligned}$$

For each k we count the number m_k of all relations above where v_k occurs and the number h_k of them the right-hand side of which is 0. For instance, v_1 occurs in $m_1 = 3$ relations and $h_1 = 2$ of them are zero. We use then a magic formula to be proved later to compute new probabilities p_k. That is

1. Compute $s = s(p,t)$ by the recurrent formula

$$\begin{aligned} s(p,1) &= p, \\ s(p,t) &= p\,s(p,t-1) + (1-p)\,(1-s(p,t-1)). \end{aligned}$$

2. Compute finally $p_k =$

$$Pr(v_k = 0) = \frac{ps^{h_k}(1-s)^{m_k-h_k}}{ps^{h_k}(1-s)^{m_k-h_k} + (1-p)(1-s)^{h_k}s^{m_k-h_k}}. \qquad (16)$$

Then we calculate the probabilities p_k(here $t = 2$) for our example.

k	0	1	2	3	4	5	6	7	8	9	10	11
m_k	2	3	4	4	4	4	3	3	3	2	2	2
h_k	1	2	4	4	2	3	3	2	1	2	2	1
p_k	.75	.83	.95	.95	.75	.89	.93	.83	.64	.89	.89	.75

We take now indexes k with the highest probabilities p_k. They are $2, 3, 5, 6, 10$ with $p^* = \min\{p_2, p_3, p_5, p_6, p_{10}\} = 0.89$. Then $q^* = 1 - p^* = 0.11$. That is the average weight of $v_2, v_3, v_5, v_6, v_{10}$ is about $q^*L_1 = 0.11 \times 5 = 0.55$. Then one uses the equation

$$u_j = (s_3, s_2, s_1, s_0) \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}^j \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

to determine

$$\begin{aligned} u_2 &= s_0 \oplus s_1, \\ u_3 &= s_1 \oplus s_2, \\ u_5 &= s_0 \oplus s_1 \oplus s_3, \\ u_6 &= s_0 \oplus s_2, \\ u_{10} &= s_0 \oplus s_1 \oplus s_2 \oplus s_3. \end{aligned}$$

Then $u_j \oplus x_j = v_j$ imply a system of equations

$$\begin{aligned} v_2 &= s_0 \oplus s_1 \oplus 1, \\ v_3 &= s_1 \oplus s_2 \oplus 0, \\ v_5 &= s_0 \oplus s_1 \oplus s_3 \oplus 0, \\ v_6 &= s_0 \oplus s_2 \oplus 1, \\ v_{10} &= s_0 \oplus s_1 \oplus s_2 \oplus s_3 \oplus 1. \end{aligned}$$

Now we try vector $v = (v_2, v_3, v_5, v_6, v_{10})$ with a small weight and solve the above system. For $|v| = 0$ the solution is $(s_3, s_2, s_1, s_0) = (1, 1, 1, 0)$. Using the latter as an initial state one produces the key-stream $0, 1, 1, 1, 0, 0, 0, \ldots$. So the guess was wrong. When trying weight 1 vectors we realize that they are not equally probable. E.g. vectors $(0, 0, 1, 0, 0)$ and $(0, 0, 0, 0, 1)$ are more probable than other vectors of weight 1. Trying $v = (0, 0, 1, 0, 0)$ we get the true solution $(s_3, s_2, s_1, s_0) = (0, 0, 0, 1)$.

2.10.3 How to Compute New Probabilities

Let u_k occur in a sparse relation

$$u_k \oplus u_{k_1} \oplus \ldots \oplus u_{k_t} = 0.$$

Then

$$R = x_k \oplus x_{k_1} \oplus \ldots \oplus x_{k_t} = v_k \oplus v_{k_1} \oplus \ldots \oplus v_{k_t}.$$

One is able to compute R as the key-stream digits are known. So

$$R = v_k \oplus V,$$

where $V = v_{k_1} \oplus \ldots \oplus v_{k_t}$ and v_k doesn't occur in V. We now find $Pr(V = 0)$. Assume v_j are independent random variables equally distributed such that $Pr(v_j = 0) = p$. By induction one finds

$$\begin{aligned} s(p,t) &= Pr(V = 0) \\ &= Pr(v_{k_1} = 0)Pr(v_{k_2} \oplus \ldots \oplus v_{k_t} = 0) \\ &= Pr(v_{k_1} = 1)Pr(v_{k_2} \oplus \ldots \oplus v_{k_t} = 1) \\ &= p\, s(p,t-1) + (1-p)\,(1 - s(p,t-1)), \end{aligned}$$

where $s(p,1) = p$.

Because x_k appears in several above relations, one gets a system of $m = m_k$ relations only involving v_k and some other v_j:

$$\begin{aligned} R_1 &= v_k \oplus v_{k_{1,1}} \oplus \ldots \oplus v_{k_{1,t}} = v_k \oplus V_1, \\ R_2 &= v_k \oplus v_{k_{2,1}} \oplus \ldots \oplus v_{k_{2,t}} = v_k \oplus V_2, \\ \ldots, & \\ R_m &= v_k \oplus v_{k_{m,1}} \oplus \ldots \oplus v_{k_{m,t}} = v_k \oplus V_m. \end{aligned}$$

After pre-computation we know that some h of $R_1, R_2, \ldots, R_m$ are zeros. This is an additional information we want. We compute now the probability p_k that $v_k = 0$ under this condition. By symmetry this probability doesn't depend on which of R_i are zeros. So we can assume $R_1 = \ldots = R_h = 0$. Let us denote the event $\{R_1 = \ldots = R_h = 0, R_{h+1} = \ldots = R_m = 1\}$ by A. Then one finds

$$p_k = Pr(v_k = 0|A) = \frac{Pr(v_k = 0, A)}{Pr(A)} = \frac{Pr(v_k = 0)Pr(A|v_k = 0)}{Pr(A)}.$$

We compute $Pr(A) =$

$$Pr(v_k = 0)Pr(A|v_k = 0) + Pr(v_k = 1)Pr(A|v_k = 1),$$

where $Pr(A|v_k = 0) =$

$$Pr(V_1 = \ldots = V_h = 0, V_{h+1} = \ldots = V_m = 1) = s^h(1-s)^{m-h},$$

and $Pr(A/v_k = 1) =$

$$Pr(V_1 = \ldots = V_h = 1, V_{h+1} = \ldots = V_m = 0) = s^{m-h}(1-s)^h.$$

Therefore,

$$Pr(A) = ps^h(1-s)^{m-h} + (1-p)(1-s)^h s^{m-h}$$

and

$$Pr(v_k = 0)\, Pr(A|v_k = 0) = ps^h(1-s)^{m-h}.$$

Then on the whole

$$p_k = \frac{ps^h(1-s)^{m-h}}{ps^h(1-s)^{m-h} + (1-p)(1-s)^h s^{m-h}}.$$

Hence (16) is true.

2.10.4 Summary of the Method.Complexity

1. Compute an affine approximation $g = a_1X_1 + \ldots + a_LX_L + b$ to F and the probability of approximation p.

2. Given a segment of key-stream $x^N = x_0, x_1, \ldots, x_{N-1}$ compute $x^N \oplus b$, denoted x^N again for simplicity.

3. For each $0 \le k \le N-1$ compute the number m_k of possible relations R_k, where v_k occurs and the number h_k of zero-relations. We need relations with small number of summands. So they are produced from the generating polynomial $f(X)$ of the LFSR and its powers $f(X^{2^r})$ for $L2^r \le N-1$.

4. For each $0 \le k \le N-1$ compute new probabilities p_k. Take L_1 indexes k with the largest probabilities $p_{k_1}, \ldots, p_{k_{L_1}}$. Compute $q^* = 1 - \min p_{k_i}$.

5. Try all vectors $v = (v_{k_1}, \ldots, v_{k_{L_1}})$ of weight about q^*L_1. Define margins from the probability of the algorithm failure as in the Affine Approximation Attack.

6. Solve the system of linear equations coming from $x_{k_i} = u_{k_i} \oplus v_{k_i}$. Form a guess $s_{L-1}, \ldots, s_1, s_0$.

7. Check the guess by decrypting the cipher-text.

The complexity(the number of trials) of the Attack is about($L_1 = L$)

$$\sum_{i=0}^{q^*L(1+o(1))} \binom{L}{i} = 2^{H(q^*)L(1+o(1))}.$$

Because $q^* \le q$ this number is generally less than the number of trials in the Affine Approximation Attack.

Example. Let $L = 100$, $t = 2$ and $p = 3/4$. The length of the key-stream $N = 5000$. So on the average one has $m = 12$ relations per digit of the key-stream. One can prove that approximately 109 digits will give $h \ge 11$ zero-relations. The new probabilities for these digits are at least 0.9978. Then one should only try few vectors of size 109 with a very low weight.

2.11 Algebraic Attacks

This is a kind of a known plain-text attack usually against initial state. A stream cipher as depicted in Figure 12 is considered. A segment of the key-stream $x^N = x_0, x_1, \ldots, x_{N-1}$ of length N is known. So this key-stream and the initial state $s_0, s_1, \ldots, s_{L-1}$ are related as

$$x_j = F((s_{L-1}, s_{L-2}, \ldots, s_0)A^j) \tag{17}$$

for $j = 0, \ldots, N-1$, where A is a matrix implementing the change of the LFSR states, it is similar to the companion matrix for the generating polynomial of the LFSR. One represents the right-hand sides of the above equations with their algebraic normal forms in variables $s_0, s_1, \ldots, s_{L-1}$. A nice property of Boolean functions $F((s_0, s_1, \ldots, s_{L-1})A^j)$ is that their algebraic degrees are equal.

Lemma 15 *Let the matrix A be nonsingular. Then*

$$\deg F((s_{L-1}, s_{L-2}, \ldots, s_0)A^j)$$

doesn't depend on j.

Proof. From the definition of the algebraic degree we see that $\deg F(xA) \leq \deg F(x)$ for any matrix A. But when A is nonsingular we get $\deg F(x) = \deg F((xA^{-1})A) \leq \deg F(xA)$. Then $\deg F(xA) = \deg F(x)$. This finishes the proof.

Algebraic Attack is in solving the system of nonlinear Boolean equations (17). The problem is that we can only handle efficiently linear equation systems due to Gaussian elimination. After each step of elimination one has linear equations again. This is not true for nonlinear equations, that is the algebraic degree may increase when a variable is being eliminated. E.g. in the system

$$X_1X_2 \oplus X_3 = 1,$$
$$X_4X_3 \oplus X_1 \oplus X_2 = 0$$

we want to eliminate X_3 from the second equation using the first one. We get a new system

$$X_1X_2 \oplus X_3 = 1,$$
$$X_4X_1X_2 \oplus X_4 \oplus X_1 \oplus X_2 = 0$$

the algebraic degree of which is 3 instead of 2 as in the initial system.

2.11.1 Linearization Attack

Consider another example of a system describing the stream cipher depicted in Fig.14. Boolean function $F = X_1 \oplus X_2 \oplus X_1X_3$ and a segment of the key-stream

$x^{10} = 0, 1, 1, 0, 0, 0, 1, 0, 1, 1$ of length 10 is given. The system of equations in unknown digits s_3, s_2, s_1, s_0 of the initial state is

$$\begin{aligned}
x_0 &= s_3 \oplus s_2 \oplus s_3 s_0, \\
x_1 &= s_0 \oplus s_1 \oplus s_3 \oplus (s_0 \oplus s_1) s_1, \\
x_2 &= s_2 \oplus s_0 \oplus (s_1 \oplus s_2) s_2, \\
x_3 &= s_3 \oplus s_1 \oplus (s_2 \oplus s_3) s_3, \\
x_4 &= s_0 \oplus s_1 \oplus s_2 \oplus (s_0 \oplus s_1 \oplus s_3)(s_0 \oplus s_1), \\
x_5 &= s_2 \oplus s_1 \oplus s_3 \oplus (s_0 \oplus s_2)(s_1 \oplus s_2), \\
x_6 &= s_1 \oplus s_3 \oplus s_0 \oplus s_2 \oplus (s_1 \oplus s_3)(s_2 \oplus s_3), \\
x_7 &= s_0 \oplus s_2 \oplus s_3 \oplus (s_0 \oplus s_1 \oplus s_2)(s_0 \oplus s_1 \oplus s_3), \\
x_8 &= s_3 \oplus s_0 \oplus (s_1 \oplus s_2 \oplus s_3)(s_0 \oplus s_2), \\
x_9 &= s_0 \oplus (s_0 \oplus s_1 \oplus s_2 \oplus s_3)(s_1 \oplus s_3).
\end{aligned}$$

Then one expands brackets, simplifies equations with identities $s_i^2 = s_i$ because we want solutions $s_i = 0, 1$, substitute known digits of the key-stream, introduce new variables $s_0 s_1 = T, s_0 s_2 = U, s_0 s_3 = V, s_1 s_2 = W, s_1 s_3 = R, s_2 s_3 = Z$ and gets a new system this time linear:

$$\begin{aligned}
0 &= s_3 \oplus s_2 \oplus V, \\
1 &= s_0 \oplus s_3 \oplus T, \\
1 &= s_0 \oplus W, \\
0 &= s_1 \oplus Z, \\
0 &= s_2 \oplus V \oplus R, \\
0 &= s_1 \oplus s_3 \oplus U \oplus W \oplus T, \\
1 &= s_1 \oplus s_0 \oplus s_2 \oplus W \oplus R \oplus Z, \\
0 &= s_2 \oplus s_3 \oplus s_1 \oplus V \oplus U \oplus W \oplus R \oplus Z, \\
1 &= s_3 \oplus s_0 \oplus s_2 \oplus V \oplus U \oplus W \oplus T \oplus Z, \\
1 &= s_0 \oplus s_1 \oplus s_3 \oplus V \oplus W \oplus T \oplus Z.
\end{aligned}$$

Elimination is now applicable. The last system has the unique solution

$$T = U = V = W = R = Z = s_3 = s_2 = s_1 = 0, s_0 = 1.$$

In particular, this defines the initial state of the LFSR.

We estimate the method application to general equations (17). As the algebraic degree of all the equations (17) is d the number of nonzero terms in all of them doesn't exceed

$$D = \sum_{i=1}^{d} \binom{L}{i}.$$

So the system of linear equations will be of no more than D variables. In order to ensure a unique solution there should be at least D equations. So it is necessary

$N \geq D$. Let N be a bit less than D. Then the true initial state is revealed with finding $\geq 2^{D-N}$ solutions to the equation system and check which of them, used as an initial state, able to decrypt the rest of the cipher-text to somewhat sensible. The complexity of the attack is approximately $D^3/3$ bit operations with Gaussian elimination with about D bits of the key-stream. In the above example the algebraic degree $d = 2$ and $D = \binom{4}{1} + \binom{4}{2} = 10$. So 10 digits of the key-stream is generally enough to find the true initial state uniquely.

2.11.2 Extended Linearization Method

We consider the system of Boolean equations whose left-hand sides are polynomials in Algebraic Normal Form. By $\deg f_j$ we denote the total algebraic degree of the polynomial f_j.

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ \ldots & \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned} \tag{18}$$

One first construct a new system

$$x_1^{i_1} \ldots x_n^{i_n} f_j = 0 \tag{19}$$

for all $i_1, \ldots, i_n \in \{0, 1\}$ and $1 \leq j \leq m$, where $i_1 + \ldots + i_n + \deg f_j \leq c$, where c is a parameter. We then solve the equations (19) by Linearization as above. Let N_c be the number of different non constant terms in all (19). Also let R_c be the rank of (19) after the linearisation, that is writing non linear terms as new variables. One can prove in the interesting case when the number of initial equation system solutions is low there exists c such that $N_c - R_c$ is relatively small. For any c the complexity of the Extended Linearisation is proportional to

$$N_c^3 + 2^{N_c - R_c}$$

some simple steps. One can change N_c^3 to $N_c^{2+\epsilon}$ if the system is sparse enough by using some special linear algebra algorithms.
Example. Let

$$x_1x_2 + x_3 = 0,\ x_2x_3 + x_3 + x_1 = 1,\ x_1x_3 + x_2 = 0.$$

One multiplies the equations by x_1, x_2 and by x_3 and get

$$\begin{gathered} x_1x_2 + x_1x_3 = 0,\ x_1x_2x_3 + x_1x_3 + x_1 = x_1,\ x_1x_3 + x_1x_2 = 0, \\ x_1x_2 + x_2x_3 = 0,\ x_2x_3 + x_2x_3 + x_2x_1 = x_2,\ x_1x_2x_3 + x_2 = 0, \\ x_1x_2x_3 + x_3 = 0,\ x_2x_3 + x_3 + x_1x_3 = x_3,\ x_1x_3 + x_2x_3 = 0. \end{gathered}$$

By solving the linearised equations, we get

$$x_1x_2x_3 = x_1x_2 = x_1x_3 = x_2x_3 = x_2 = x_3,\ x_1 = 1.$$

So there are two solutions $x_1 = 1, x_2 = 0, x_3 = 0$ and $x_1 = 1, x_2 = 1, x_3 = 1$.

2.11.3 Annihilator Attack.

This is a kind of Algebraic Attack. The idea is to reduce somehow the algebraic degree of equations one solves to find the initial state.

Definition 9 *Let $F = F(X_1, X_2, \ldots, X_n)$ be a Boolean function in n variables. A nonzero Boolean function $G = G(X_1, X_2, \ldots, X_n)$ is called annihilator for F if $FG = 0$.*

Look at the truth table for F and G.

X_1	$\ldots$	X_n	F	G
$\ldots$				
a_1	$\ldots$	a_n	1	0
$\ldots$				
b_1	$\ldots$	b_n	0	$*$
$\ldots$				

When F has 1 value, G should have 0 value. But when $F = 0$ the function G may have the both 1 and 0 values.

Return to the stream cipher described by equations (17).

Assumption. Let F have a nonzero annihilator G_0 of degree d_0 and $F \oplus 1$ have an annihilator G_1 of degree d_1. One can have $d_0, d_1 \leq d$ anyway as $F(F \oplus 1) = 0$. But sometimes one of them or the both may be much less than d.

Example.

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
F	0	1	1	0	1	1	1	0	1	1	0	0	0	0	0	1
G_0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	0
G_1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0

.

The Boolean function F is of degree 3 and is represented by the polynomial

$$X_4 \oplus X_3 \oplus X_2 \oplus X_2X_4 \oplus X_2X_3 \oplus X_2X_3X_4 \oplus X_1 \oplus X_1X_4 \oplus X_1X_2X_4.$$

But G_0 and G_1 are of degree 2:

$$G_0 = X_1X_2 \oplus X_3X_4, \quad \text{and} \quad G_1 = X_1X_4 \oplus X_1X_3 \oplus X_3 \oplus X_4.$$

Theorem 17 *Let $F = F(X_1, \ldots, X_n)$ be a Boolean function in n variables. There exist two Boolean functions $G = G(X_1, \ldots, X_n)$ and $H = H(X_1, \ldots, X_n)$ of degrees $\leq \lceil \frac{n}{2} \rceil$ such that $GF = H$ and the functions G and H are not zero simultaneously.*

Proof. Let m denote the monomial $X_{i_1} \dots X_{i_r}$ so r is the degree(length) of m. We want to find coefficients c_m and d_m not all zero such that

$$\sum_m c_m mF + \sum_m d_m m = 0,$$

where the sum is over all the monomials m of degree $r \leq \lceil \frac{n}{2} \rceil$. This is a system of 2^n linear equations(because mF and m are represented by their coefficient vectors of length 2^n). We count now the number of variables occurring in the system. The number is 2 times the number of monomials of degree $\leq \lceil \frac{n}{2} \rceil$: $2\sum_{i=0}^{\lceil \frac{n}{2} \rceil} \binom{n}{i} > 2^n$. As the number of equations $2^n = \sum_{i=0}^{n} \binom{n}{i}$ is strictly less than the number of variables, the homogeneous system has at least one nonzero solution. Then $G = \sum_m c_m m$ and $H = \sum_m d_m m$ and $GF = H$. The polynomials can't be zero simultaneously. This proves the Theorem.

Corollary 8 *The Boolean function F or $F \oplus 1$ has an annihilator of degree $\leq \lceil \frac{n}{2} \rceil$.*

Proof. Consider two cases:

1. The polynomial $H \neq 0$. Multiply the equation $GF = H$ by F and get $GF^2 = HF$. As $F^2 = F$, we get $H = HF$ and therefore H is an annihilator of $F \oplus 1$ of degree $\leq \lceil \frac{n}{2} \rceil$.

2. The polynomial $H = 0$. Then $GF = 0$, so G is an annihilator of F of degree $\leq \lceil \frac{n}{2} \rceil$.

This proves the Corollary.

Given F of degree d, we wonder how to find an annihilator for F of degree at most d_0. To this end one produces the algebraic normal form for $mF = X_{i_1} X_{i_2} \dots X_{i_t} F$ for all possible monomials $m = X_{i_1} X_{i_2} \dots X_{i_t}$ of degree $t \leq d_0$. The polynomial mF is also representable as a binary string of its coefficients of length $\sum_{i=0}^{d'} \binom{n}{i}$ where $d' = \min\{d + d_0, n\}$. Then one finds a zero linear combination of these strings $\sum_m c_m mF = 0$ if there exist any. Then put $G = \sum_m c_m m$.

We apply annihilators to reduce the algebraic degree of equations (17). There are two cases:

1. Let $x_j = 0$ then multiply both the sides of $0 = F((s_{L-1}, s_{L-2}, \dots, s_0)A^j) = F(SA^j)$ by $G_1(SA^j)$. Because $(F \oplus 1)G_1 = 0$ one gets a new equation

   $$G_1(SA^j) = 0$$

   of degree d_1.

2. Let $x_j = 1$ then multiply the both sides of $1 = F((s_{L-1}, s_{L-2}, \dots, s_0)A^j)$ by $G_0((s_{L-1}, s_{L-2}, \dots, s_0)A^j)$. Because $FG_0 = 0$ one gets a new equation

   $$G_0((s_{L-1}, s_{L-2}, \dots, s_0)A^j) = 0$$

   of degree d_0.

The new equations system is of smaller degree than that of F. Managed to construct annihilators of smaller degrees we will get an advantage in the number of necessary key-stream digits and the complexity.

Example. Consider the stream cipher presented in Fig. 14 with the Boolean function F as in the previous example. The key-stream is $x^{10} = 1, 1, 1, 1, 1, 0, 1, 0, 0, 1$. The goal is to find the initial state. Because the algebraic degree of F is 3 a simple linearization requires at least

$$\binom{4}{1} + \binom{4}{2} + \binom{4}{3} = 14$$

key-stream bits to ensure an unique solution. Then solving a system of linear equations in 14 variables is necessary. With annihilators G_0 and G_1 we reduce the algebraic degree of the equations to solve. The new equation system is represented as

$$G_0(S) = 0, \quad G_0(SA) = 0, \quad G_0(SA^2) = 0, \quad G_0(SA^3) = 0, \quad G_0(SA^4) = 0,$$
$$G_1(SA^5) = 0, \quad G_0(SA^6) = 0, \quad G_1(SA^7) = 0, \quad G_1(SA^8) = 0, \quad G_0(SA^9) = 0.$$

That is

$$\begin{aligned}
s_0s_1 \oplus s_2s_3 &= 0,\\
s_2s_1 \oplus s_3(s_0 \oplus s_1) &= 0,\\
s_2s_3 \oplus (s_0 \oplus s_1)(s_1 \oplus s_2) &= 0,\\
s_3(s_0 \oplus s_1) \oplus (s_1 \oplus s_2)(s_2 \oplus s_3) &= 0,\\
(s_0 \oplus s_1)(s_1 \oplus s_2) \oplus (s_2 \oplus s_3)(s_0 \oplus s_1 \oplus s_3) &= 0,\\
(s_1 \oplus s_2)(s_0 \oplus s_2) \oplus (s_1 \oplus s_2)(s_0 \oplus s_1 \oplus s_3) \oplus s_1 \oplus s_3 \oplus s_2 &= 0,\\
(s_2 \oplus s_3)(s_0 \oplus s_1 \oplus s_3) \oplus (s_0 \oplus s_2)(s_1 \oplus s_3) &= 0\\
(s_0 \oplus s_1 \oplus s_3)(s_0 \oplus s_1 \oplus s_2) \oplus (s_0 \oplus s_1 \oplus s_3)(s_1 \oplus s_3) \oplus s_1 \oplus s_3 \oplus s_2 &= 0,\\
(s_0 \oplus s_2)(s_1 \oplus s_2 \oplus s_3) \oplus (s_0 \oplus s_2)(s_0 \oplus s_1 \oplus s_2) \oplus s_1 \oplus s_3 &= 0,\\
(s_1 \oplus s_3)(s_0 \oplus s_1 \oplus s_2) \oplus (s_1 \oplus s_2 \oplus s_3)(s_0 \oplus s_1 \oplus s_2 \oplus s_3) &= 0.
\end{aligned}$$

Then we expand brackets, simplify equations and introduce new variables $s_0s_1 = T, s_0s_2 = U, s_0s_3 = V, s_1s_2 = W, s_1s_3 = R, s_2s_3 = Z$ and get a new system:

$$\begin{aligned}
T \oplus Z &= 0,\\
W \oplus R \oplus V &= 0,\\
s_1 \oplus Z \oplus W \oplus U \oplus T &= 0,\\
s_2 \oplus Z \oplus W \oplus V &= 0,\\
s_1 \oplus s_3 \oplus Z \oplus R \oplus V \oplus T &= 0,\\
s_3 \oplus R \oplus Z &= 0,\\
s_3 \oplus T \oplus R \oplus U &= 0,\\
s_2 \oplus W \oplus T \oplus R \oplus U \oplus Z &= 0,\\
s_3 \oplus U \oplus Z \oplus V &= 0,\\
s_2 \oplus s_3 \oplus W \oplus R \oplus U \oplus Z &= 0.
\end{aligned}$$

The system has two solutions:

$$s_1 = s_2 = s_3 = T = U = V = W = R = Z = 0$$

and $s_0 = 0, 1$. As $S \neq 0$ we get $S = (0, 0, 0, 1)$.

2.12 Combining LFSR.

Let n LFSRs be combined with a Boolean function $F = F(X_1, \ldots, X_n)$:

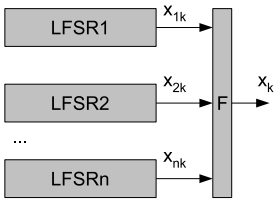

Figure 17: LFSRs Combiner.

The key-stream is computed as $x_k = F(x_{1,k}, x_{2,k}, \ldots, x_{n,k})$. We will answer two questions

1. How to guarantee the period of the key-stream?
2. What is the linear complexity of the key-stream?
3. How to break the generator with an improper Boolean function F?

To answer the first question we assume the sequence $\{x_{i,k}\}$ has the minimal period $T_i > 1$ and the periods are pairwise co-prime.

Lemma 16 *The string $(x_{1,k}, x_{2,k}, \ldots, x_{n,k})$ runs over all possible binary n-strings as k tends to ∞.*

We will prove the statement by induction on n. When $n = 1$ the statement is true as the sequence $\{x_{1,k}\}$ is not constant due to $T_1 > 1$. Assume the statement to be true when the number of sequences $< n$ and prove it for just n sequences. We fix any vector $(x_1^0, x_2^0, \ldots, x_n^0)$. By induction there is a number s such that

$$(x_{2,s}, \ldots, x_{n,s}) = (x_2^0, \ldots, x_n^0).$$

Let $T_2 \ldots T_n = T'$. By assumption $\gcd(T_1, T') = 1$. Consider the sequence of vectors

$$(x_{1,s+T'r}, x_{2,s+T'r}, \ldots, x_{n,s+T'r}) = (x_{1,s+T'r}, x_2^0, \ldots, x_n^0),$$

where $r = 0, 1, 2, \ldots$. The number $s + T'r$ runs over all residues modulo T_1. That is $x_{1,s+T'r}$ runs over the whole sequence $\{x_{1,k}\}$. In particular, $x_{1,s+T'r}$ may have 0 and 1 values. So there exists r_0 such that $x_{1,s+T'r_0} = x_1^0$. Then

$$(x_{1,s+T'r_0}, x_{2,s+T'r_0}, \ldots, x_{n,s+T'r_0}) = (x_1^0, x_2^0, \ldots, x_n^0).$$

This finishes the proof.

Definition 10 *The variable X_1 is called relevant for a Boolean function $F = F(X_1, \ldots, X_n)$ if there exists an $n-1$-string $x_2^0, \ldots, x_n^0$ such that the function $F(X_1, x_2^0, \ldots, x_n^0)$ in one variable X_1 is not constant.*

Theorem 18 *Let all variables $X_1, \ldots, X_n$ be relevant to the Boolean function $F = F(X_1, \ldots, X_n)$. Then the minimal period of $\{x_k\}$ is equal to the product $T_1 T_2 \ldots T_n$.*

Proof. Let t be the minimal period of the key-stream $\{x_k\}$. We will prove that each T_i divides t. As X_1 is relevant for $F = F(X_1, \ldots, X_n)$ there is a string $(x_2^0, \ldots, x_n^0)$ such that $F(X_1, x_2^0, \ldots, x_n^0)$ is not constant, that is equals $X_1 \oplus b$ for some binary digit b. By the previous Lemma one finds s such that

$$(x_{2,s}, \ldots, x_{n,s}) = (x_2^0, \ldots, x_n^0).$$

Then $x_{s+T'r} =$

$$F(x_{1,s+T'r}, x_{2,s+T'r}, \ldots, x_{n,s+T'r}) = F(x_{1,s+T'r}, x_2^0, \ldots, x_n^0),$$

which equals $x_{1,s+T'r} \oplus b$ for all $r = 0, 1, 2, \ldots$. As $\gcd(T', T_1) = 1$ the minimal period of $\{x_{1,s+T'r}\}$ is T_1. On the other hand, t, being a period of $\{x_r\}$, is a period of $\{x_{1,s+T'r}\}$. Therefore, T_1 divides t. Similarly, one proves that other T_i all divide t. So the product $T_1 \ldots T_n$ divides t, as the terms of the product are pairwise co-prime. Because $T_1 \ldots T_n$ is by itself a period of $\{x_k\}$ then $T_1 \ldots T_n$ is the minimal period.

Choose now LFSRs as generators for $\{x_{i,s}\}$. Let they be of maximal periods $T_i = 2^{L_i} - 1$, where L_i stand for LFSR's lengths. In order to use Theorem 18 one should guarantee that $\gcd(T_i, T_j) = 1$.

Lemma 17

$$\gcd(2^{L_i} - 1, 2^{L_j} - 1) = 2^{\gcd(L_i, L_j)} - 1.$$

Proof. We apply the Euclidean Algorithm to the natural numbers L_i and L_j. That is $L_i = a_1 L_j + b_1$, where $0 \le b_1 < L_j$ and so on. Then

$$2^{L_i} - 1 (\mod 2^{L_j} - 1) \equiv 2^{a_1 L_j + b_1} - 1 \equiv (2^{L_j})^{a_1} 2^{b_1} - 1 \equiv 2^{b_1} - 1. \tag{20}$$

This implies

$$\gcd(2^{L_i} - 1, 2^{L_j} - 1) = \gcd(2^{L_j} - 1, 2^{b_1} - 1).$$

Similarly, let $b_1, b_2, \ldots, b_s = 0$ be the sequence of remainders in the Euclidean Algorithm application. Then

$$\gcd(2^{L_j} - 1, 2^{b_1} - 1) = \gcd(2^{b_1} - 1, 2^{b_2} - 1) = \ldots = 2^{\gcd(L_i, L_j)} - 1,$$

because generally

$$2^{b_{s-1}} - 1 \equiv 2^{a_{s+1} b_s + b_{s+1}} - 1 \equiv 2^{b_{s-1}} - 1 \mod 2^{b_{s+1}} - 1.$$

This proves the statement. As an example, $\gcd(2^{11}-1, 2^{17}-1) = 2^{\gcd(11,17)}-1 = 1$. To secure a large period of the combiner one takes LFSRs with primitive polynomials and pairwise coprime lengths.

The linear complexity of the combiner output is estimated with The Theorem.

Theorem 19 *Let LFSRs are of full periods and of pair-wise co-prime lengths $L_1, \ldots, L_n$. Then the linear complexity of the key-stream is bounded by*

$$F(L_1, \ldots, L_n),$$

where F is taken over integers, that is with changing $\oplus$ by $+$.

2.12.1 Geffe generator.

The picture represents so called Geffe generator. The combining function is

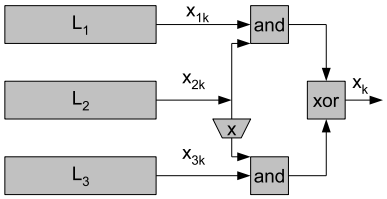

Figure 18: Geffe Generator.

$F(X_1, X_2, X_3) = X_1X_2 \oplus X_3(X_2 \oplus 1)$. All variables are relevant in F. So for $L_1 = 11, L_2 = 13, L_3 = 17$ the period of the output sequence $\{x_k\}$ is

$$(2^{11} - 1)(2^{13} - 1)(2^{17} - 1) \approx 2.19 \times 10^{12}$$

Let S_i^0 be an initial state of the LFSRi. We suppose them unknown while a sequence of the key-stream $x^N = x_0, \ldots, x_{N-1}$ is known. We briefly consider some approaches to find all initial states or predict other digits of the key-stream.

1. One tries 13 bits of S_2^0, produces a segment of the sequence $\{x_{2,k}\}$ and finds that $x_{3,k} = x_k$ when $x_{2,k} = 0$ and $x_{1,k} = x_k$ when $x_{2,k} = 1$. It is necessary to have a bit more than 11 digits of $\{x_{1,k}\}$ to reveal S_1^0 and a bit more than 17 digits of $\{x_{3,k}\}$ to find S_3^0. On the whole one should have approximately(or a bit more) 34 digits of the key-stream to break the generator with 2^{13} trials.

2. As the linear complexity of the key-stream is at most $11 \times 13 + 17(13 + 1) = 381$, the Berlakamp-Massey attack requires at most 762 key-stream digits.

3. About $\binom{41}{2} + \binom{41}{1} = 861$(in fact $11 \times 13 + 13 \times 17 + 41 = 405$ because not all possible monomials occur in equations) key-stream bits are necessary for the Algebraic Attack via linearization as equations are quadratic.

4. For an Affine Approximation Attack we realize that

$$\begin{aligned} Pr(F = X_1) = Pr(X_2 = 1)Pr(F = X_1/\{X_2 = 1\}) + \\ Pr(X_2 = 0)Pr(F = X_1/\{X_2 = 0\}) = \\ \frac{1}{2} + \frac{1}{2}Pr(X_3 = X_1) = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}. \end{aligned}$$

   One needs 11 key-stream digits and the number of trials is approximately $\sum_{i=0}^{4} \binom{11}{i}$. This is actually the number of guesses for S_1^0.

We will consider now a kind of correlation attack which has not been studied before. The idea of the attack is in the following. Try binary 11-string S_1 as the initial state of the LFSR1 and compute a segment of the output sequence: $x_{1,0}, x_{1,1}, \ldots x_{1,n-1}$. Then compute the sequence $v_k = x_{1,k} \oplus x_k$, for all $0 \leq k \leq N-1$. There are two possibilities:

- The current S_1 is S_1^0, the true initial state of LFSR1. Then $Pr(x_k = x_{1,k}) = \frac{3}{4}$ and the random variables v_k have the distribution:

$$v_k = \begin{cases} 1, & Pr = \frac{1}{4}; \\ 0, & Pr = \frac{3}{4}. \end{cases}$$

- If $S_1 \neq S_1^0$ then $x_{1,k}, x_k$ are assumed independent. So $Pr(x_{1,k} = x_k) = \frac{1}{2}$. Therefore, the random variables v_k have the distribution:

$$v_k = \begin{cases} 1, & Pr = \frac{1}{2}; \\ 0, & Pr = \frac{1}{2}. \end{cases}$$

One observes a sample sequence $v_0, v_1, \ldots, v_{n-1}$ and decides which distribution was used. If it was $\frac{1}{2}$-distribution, then one accepts $S_1 \neq S_1^0$, otherwise, if it was $\frac{1}{4}$-distribution, one accepts $S_1 = S_1^0$. Two kind of errors are possible. First, one accepts $S_1 = S_1^0$ while in reality $S_1 \neq S_1^0$. We denote the probability of the event by α_n, we will later see that it depends on n. The cryptanalytic sense of this probability is that $\alpha_n 2^{L_1}$ is the average number of S_1, survivors after the test. Another error is that one accepts $S_1 \neq S_1^0$ while in reality $S_1 = S_1^0$. We denote the probability of this event by β_n. This is the probability of rejecting the true initial state S_1^0. So if we distinct these two cases, we will find he true initial state S_1^0.

2.12.2 Distinguishing two distributions.

We put the problem generally. A sequence of Bernoulli trials $x_1, x_2, \ldots$, is considered, where x_i are equally distributed independent variables such that

$$v_k = \begin{cases} 1, & \Pr = p \ ; \\ 0, & \Pr = q = 1 - p \ , \end{cases}$$

where p is an unknown number $0 < p < 1$. However it is known $p = p_0 = \frac{1}{2} + \delta$ for some $0 < \delta < \frac{1}{2}$ or $p = \frac{1}{2}$.

We observe a sample sequence $x = x_1, x_2, \ldots$, an outcome of the Bernoulli trials. A question to answer is whether x was produced with Bernoulli trials for $p = p_0$ or $p = \frac{1}{2}$. The average number of successes in a sequence of N Bernoulli trials is Np. That is

$$V_N = \sum_{i=1}^{N} x_i \approx Np.$$

This means that the number $\frac{V_N}{N}$ should be close to p. This implies a criterion. Let δ be positive. If $\frac{V_N}{N} > \frac{1}{2} + \frac{\delta}{2}$, then accept $p = p_0$. Otherwise, if $\frac{V_N}{N} \leq \frac{1}{2} + \frac{\delta}{2}$, then accept $p = \frac{1}{2}$.

When δ is negative, the criterion becomes: If $\frac{V_N}{N} \leq \frac{1}{2} - \frac{\delta}{2}$, then accept $p = p_0$. Otherwise, if $\frac{V_N}{N} > \frac{1}{2} - \frac{\delta}{2}$, then accept $p = \frac{1}{2}$.

Some errors are possible. That is in reality $p = p_0$ but we accept $p = \frac{1}{2}$ and vise verse. De Moivre-Laplace Limit Theorem implies the bigger n the closer $\frac{V_N}{N}$ to p. So when the number of trials is growing the probability of the errors is decreasing. We will put this observation in a mathematical form. We fix positive real numbers $0 < \alpha < 1$ and $0 < \beta < 1$ and find a minimal N such that

$$\alpha_N = Pr(\text{accept}\, p = p_0 | p = \frac{1}{2}) \leq \alpha,$$

and

$$\beta_N = Pr(\text{accept}\, p = \frac{1}{2} | p = p_0) \leq \beta.$$

For Geffe generator, one says that $\alpha_n 2^{L_1}$ is the number of survivors after the test and β_n is the probability of rejecting the true S_1^0.

We only consider the case $\delta > 0$. Let t_α be a real number such that $\Phi(-t_\alpha) = \alpha$. E.g. $\Phi(-1.15) \approx \frac{1}{8}$. We will justify that if the number of trials is $n \geq \frac{t_\alpha^2}{\delta^2}$ then $Pr(\text{accept}\, p = p_0 | p = \frac{1}{2}) \leq \alpha$. Really,

$$\begin{aligned} Pr(\text{accept} \quad p = p_0 | p = 1/2) &= Pr(\frac{V_N}{N} > \frac{1}{2} + \frac{\delta}{2}) \\ &= Pr(\frac{V_N - \frac{N}{2}}{\sqrt{\frac{N}{4}}} > \delta\sqrt{N}) \\ &\leq Pr(\frac{V_N - \frac{N}{2}}{\sqrt{\frac{N}{4}}} > t_\alpha) \approx \Phi(-t_\alpha) = \alpha, \end{aligned}$$

because $\delta\sqrt{N} \geq t_\alpha$. The smaller δ the above approximation based on de Moivre-Laplace Theorem is sharper. By direct computation one finds that the bound

$$N \geq \frac{t_\alpha^2}{\delta^2} \tag{21}$$

is true for all $\delta \leq 0.05$ and reasonable α like 0.1 and 0.01. With Chernoff's inequality $n \geq \frac{2\ln(\alpha^{-1})}{\delta^2}$ which is always worse, in sense n bigger, than (21). Consider a numeric example. Let we want the probability of the error $\leq \frac{1}{8}$. Then one should take the number of trials $n \geq \frac{1.32}{\delta^2}$. For $\delta = \frac{1}{100}$ the number of trials N should be bigger than 13200. We consider another kind of error now. We will prove that if the number of trials is

$$N \geq \frac{4p_0q_0t_\beta^2}{\delta^2} \approx \frac{t_\beta^2}{\delta^2}$$

then $Pr(p = \frac{1}{2}|p = p_0) \leq \beta$. This is because $4p_0q_0 = 4(\frac{1}{2}+\delta)(\frac{1}{2}-\delta) = 1-4\delta^2 \approx 1$. Really, one finds, under condition $p = p_0$, that

$$\begin{aligned}
\beta_N = Pr(\text{accept } p = \frac{1}{2}|p = p_0) &= Pr(\frac{V_N}{N} \leq \frac{1}{2} + \frac{\delta}{2}) \\
&= Pr(\frac{V_N}{N} - p_0 + \delta \leq \frac{\delta}{2}) = Pr(\frac{V_N}{N} - p_0 \leq -\frac{\delta}{2}) \\
&= Pr(\frac{V_N - Np_0}{\sqrt{Np_0q_0}} \leq -\frac{\delta N}{2\sqrt{Np_0q_0}}) \\
&\leq Pr(\frac{V_N - Np_0}{\sqrt{Np_0q_0}} \leq -t_\beta) \approx \Phi(-t_\beta) = \beta,
\end{aligned}$$

because

$$\frac{\delta N}{2\sqrt{Np_0q_0}} \geq t_\beta.$$

For instance, when $\beta = \frac{1}{8}$ and $\delta = \frac{1}{100}$ one should have $N \geq 13200$ again.

Example. For Geffe generator, depicted in Figure 19, let $L_1 = 4$,$L_2 = 5$,$L_3 = 7$ and the LFSRs be defined with primitive polynomials $X^4 + X + 1$, $X^5 + X^2 + 1$ and $X^7 + X + 1$. A segment of the key-stream of length 20 is given

$$x^{20} = 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1.$$

According to the attack description one tries all nonzero $S_1 = (s_{1,3}, s_{1,2}, s_{1,1}, s_{1,0})$ and computes a related segment of the first LFSR's output $x_{1,k}$ of the same size 20 and a segment of $v_k = x_{1,k} \oplus x_k$. Then one computes

$$\frac{V_N}{N} = \frac{\sum_{k=0}^{N-1} v_k}{N}$$

and uses the above criterion to decide whether the current S_1 survives($\frac{V_N}{N} \leq 0.375$) or not. E.g. for $S_1 = (1, 0, 0, 0)$ the first LFSR's output is

$$0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0$$

so $v^{20} = 1,1,0,1,1,1,0,0,0,1,0,0,1,1,0,1,0,1,1,1$. We compute $\frac{V_N}{N}$ for $N = 10, 20$. That is $\frac{V_{10}}{10} = \frac{6}{10} = 0.6$ and $\frac{V_{20}}{20} = \frac{12}{20} = 0.6$ and so on. Then we tabulate this data for all nonzero 4-strings S_1.

S_1	$\frac{V_{10}}{10}$	$\frac{V_{20}}{20}$
1000	0.6	0.6
0100	0.6	0.6
1100	0.8	0.65
0010	0.3	0.45
1010	0.3	0.45
0110	0.5	0.5
1110	0.7	0.55
0001	0.4	0.45
1001	0.6	0.5
0101	0.2	0.1
1101	0.6	0.55
0011	0.5	0.45
1011	0.3	0.5
0111	0.5	0.5
1111	0.5	0.55

For $N = 10$ there are 4 surviving 4-strings:

$$(0,0,1,0),\ (1,0,1,0), (0,1,0,1), (1,0,1,1),$$

while for $N = 20$ only $(0,1,0,1)$ survives. The probability of missing the true S_1^0 is $\beta_{20} \approx 0.098$ when $n = 20$.

Let S_1^0 have been computed. What to do next? We know that $F = X_1$ under the condition $X_2 = 0$. So we try for S_2 all possible nonzero L_2-strings and compute the sequence $x_{2,k}$. If $S_2^0 = S_2$, then for $x_{2,k} = 0$ we have $x_{1,k} = x_k$. Otherwise, if $S_2^0 \neq S_2$, then digits $x_{1,k}$ and x_k are assumed to be independent. That is the probability of $x_{1,k} = x_k$ is $\frac{1}{2}$. One uses this observation as an criterion for finding S_2^0. One should have a bit more than $2L_2$ digits of the key-stream in order to accomplish the test. Then the initial state of the third register is computed by solving the system of linear equations. To this end one should have at least $2L_3$ key-stream digits. Also we realise

$$Pr(F = X_2/X_1 = 0) = Pr(X_3(X_2 \oplus 1) \oplus X_2 = 0) = \frac{1}{4}.$$

So

$$Pr(x_k = x_{2,k}) = \frac{1}{4}$$

under the condition $x_{1,k} = 0$. This implies a method of finding S_2^0. Given S_1^0, one produces a segment of $x_{2,k}$ and looks for indexes k, where $x_{2,k} = 0$

2.13 Correlation Attack

We will apply now the above correlation attack to a more general combiner in Fig.17. For a segment of the key-stream $x^N = x_0, x_1, \ldots, x_{N-1}$, the task is to find the initial states $S_1^0, S_2^0, \ldots, S_n^0$ of the LFSRs. Let there be a Boolean function $\phi(X_1, \ldots, X_k)$, where $k < n$, such that

$$Pr(\phi(X_1, \ldots, X_k) = F(X_1, \ldots, X_n)) = q_0 = \frac{1}{2} - \delta$$

for some $0 < |\delta| < \frac{1}{2}$. Let $x_i' = \phi(x_{1i}, \ldots, x_{ki})$. Then $Pr(x_i' = x_i) = q_0$.

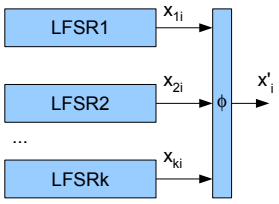

Figure 19: Auxiliary Combiner.

To break the generator in Fig.17 we try all k-tuples of nonzero initial states $S_1, S_2, \ldots, S_k$. For each tuple the sequence $v_i = x_i \oplus x_i'$ is computed. There are two possibilities

1. $(S_1, S_2, \ldots, S_k) = (S_1^0, S_2^0, \ldots, S_k^0)$. Then

$$v_i = \begin{cases} 0, & Pr = q_0; \\ 1, & Pr = p_0 = 1 - q_0. \end{cases}$$

2. $(S_1, S_2, \ldots, S_k) \neq (S_1^0, S_2^0, \ldots, S_k^0)$. Then

$$v_i = \begin{cases} 0, & Pr = \frac{1}{2}; \\ 1, & Pr = \frac{1}{2}, \end{cases}$$

as x_i and x_k' are assumed to be independent.

One counts the number of 1 in this sequence $\sum_{i=0}^{N-1} v_i = V_N$ and computes $\frac{V_N}{N}$. For $\delta > 0$ a criterion is constructed(when $\delta < 0$ this is similar): one accepts

$$(S_1, S_2, \ldots, S_k) = (S_1^0, S_2^0, \ldots, S_k^0)$$

if $\frac{V_N}{N} \geq \frac{1}{2} + \frac{\delta}{2}$ and one accepts

$$(S_1, S_2, \ldots, S_k) \neq (S_1^0, S_2^0, \ldots, S_k^0)$$

if $\frac{V_N}{N} < \frac{1}{2} + \frac{\delta}{2}$. By this test one finds a number of variants for the first k initial states. The number of trials is about $2^{L_1+\ldots+L_k}$ and the number of survivors is $\alpha_N 2^{L_1+\ldots+L_k}$, where α_N is the probability to take a wrong initial state $(S_1, S_2, \ldots, S_k)$ as true. To compute the rest of the states, one tries $2^{L_{k+1}+\ldots+L_n}$ initial states of the rest LFSRs for each the survivor above. The complexity of the attack is about

$$2^{L_1+\ldots+L_k} + \alpha_N 2^{L_1+\ldots+L_n}$$

trials. The probability of rejecting the true $(S_1^0, S_2^0, \ldots, S_k^0)$ is β_N.

2.14 Correlation Immune Boolean Functions

Definition 11 *A Boolean function $F(X_1, \ldots, X_n)$ is called m-balanced ($0 \leq m < n$) if for each subset of m variables $X_{i_1}, \ldots, X_{i_m}$ and any binary m-string $a_{i_1}, \ldots, a_{i_m}$ the Boolean function*

$$F(X_1, \ldots, a_{i_1}, \ldots, a_{i_m}, \ldots, X_n)$$

is balanced. One can say that F is m-th order correlation immune.

For instance, the Boolean function $F(X_1, \ldots, X_n) = X_1 \oplus \ldots \oplus X_n$ is $n-1$-balanced, because any fixation of $n-1$ variables, e.g. $X_1 = a_1, \ldots, X_{n-1} = a_{n-1}$, produces a balanced function, e.g.

$$F(a_1, \ldots, a_{n-1}, X_n) = a_1 \oplus \ldots \oplus a_{n-1} \oplus X_n.$$

Theorem 20 *Let $F(X_1, \ldots, X_n)$ be m-balanced, then*

1. *$F(X_1, \ldots, X_n)$ is k-balanced for any $0 \leq k < m$.*
2. *The algebraic degree of F is $\leq n - m - 1$ or F is of algebraic degree 1.*
3. $$Pr(\phi(X_{i_1}, \ldots, X_{i_k}) = F(X_1, \ldots, X_n)) = \frac{1}{2}$$
   for any subset of $k \leq m$ variables $X_{i_1}, \ldots, X_{i_k}$ and any Boolean function $\phi(X_{i_1}, \ldots, X_{i_k})$. The reverse statement is also true.

Proof.

Lemma 18 *Let $F(X_1, \ldots, X_n)$ not be balanced, then for any $0 \leq k < n$ and any subset of k variables there exist their fixations such that the function F is not balanced after these fixations.*

Proof. Let the subset of variables be $X_1, \ldots, X_k$ and $F(a_1, \ldots, a_k, X_{k+1}, \ldots, X_n)$ be balanced for all possible binary k-strings $a_1, \ldots, a_k$. Then $Pr(F = 1) =$

$$\sum_{a_1, \ldots, a_k} Pr((X_1, \ldots, X_k) = (a_1, \ldots, a_k)) Pr(F(a_1, \ldots, a_k, X_{k+1}, \ldots, X_n) = 1) =$$

$$\sum_{a_1, \ldots, a_k} \frac{1}{2^k} \frac{1}{2} = \frac{1}{2}.$$

Contradiction. Then there exist $a_1, \ldots, a_k$ such that $F(a_1, \ldots, a_k, X_{k+1}, \ldots, X_n)$ is not balanced. This proves the Lemma. In order to prove the first statement of the Theorem we assume that $F(X_1, \ldots, X_n)$ is not k-balanced. So one finds a fixation of some its variables, e.g. $X_1, \ldots, X_k$, and gets a non-balanced function $F(a_1, \ldots, a_k, X_{k+1}, \ldots, X_n)$. Then the Lemma guaranties that there is a fixation of the other $m-k$ variables, e.g. $X_{k+1}, \ldots, X_m$, such that one gets non-balanced Boolean function. This contradicts with the Theorem assumption.

We will now prove the statement on the algebraic degree of a m-balanced Boolean function.

Lemma 19 *Let $F(X_1, \ldots, X_n)$, $n \geq 2$, be balanced, then* $\deg F \leq n-1$.

Proof. Let $C = (c_0, c_1, \ldots, c_{2^n-1})$ be the coefficient vector of the algebraic normal form $F = c_0 + c_1 X_n + \ldots + c_{2^n-1} X_1 X_2 \ldots X_n$, introduced in Section 2.3. Theorem 7 states that

$$C = F \left(\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right)^{[n]},$$

where F denotes the values vector of F. We observe that the last column of the matrix $\left(\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right)^{[n]}$ only has entries 1. This implies

$$c_{2^n-1} = \sum_{a=0}^{2^n-1} F(a) = 2^{n-1} \equiv 0 \mod 2,$$

as $n \geq 2$ and because F is balanced. Take now into account that c_{2^n-1} is the coefficient at $X_1 X_2 \ldots X_n$ of the algebraic normal form for F and all other monomials are of smaller degrees. This proves the statement of the Lemma.

Let F be of algebraic degree $k \geq 2$. In order to prove the second statement of the Theorem assume, by contrary, that

$$F(X_1, \ldots, X_k, X_{k+1}, \ldots, X_n) = X_1 \ldots X_k + \ldots + X_{j_1} \ldots X_{j_l} + \ldots$$

is of algebraic degree $k \geq n-m$. That is for all monomials $X_{j_1} \ldots X_{j_l}$ occurring in the ANF for F, except $X_1 \ldots X_k$, we have $l < k$ or $l = k$ and at least one of the variables $X_{j_1}, \ldots, X_{j_l}$ is in $\{X_{k+1}, \ldots, X_n\}$. We fix the latter variables somehow and get the Boolean function

$$G(X_1, \ldots, X_k) = F(X_1, \ldots, X_k, a_{k+1}, \ldots, a_n).$$

The monomial $X_1 \ldots X_k$ is the largest in the ANF for G. By assumption, $k \geq 2$ and G is balanced because we have fixed $n - k \leq m$ variables. This is a contradiction with the above Lemma and the statement is proved.

We will prove the last statement of the Theorem. Let F be m-balanced, then F is k-balanced. One fixes $i_1, \ldots, i_k = 1, \ldots, k$. So

$$Pr(\phi(X_1, \ldots, X_k) = F(X_1, \ldots, X_n)) =$$
$$\frac{1}{2^k} \sum_{a_1, \ldots, a_k} Pr(\phi(a_1, \ldots, a_k) = F(a_1, \ldots, a_k, X_{k+1}, \ldots, X_n)) =$$
$$\frac{1}{2^k} \sum_{a_1, \ldots, a_k} \frac{1}{2} = \frac{1}{2}$$

for any subset $i_1, \ldots, i_k$ and any Boolean function $\phi(X_{i_1}, \ldots, X_{i_k})$.

Let us prove the reverse statement, that is if

$$Pr(\phi(X_{i_1}, \ldots, X_{i_k}) = F(X_1, \ldots, X_n)) = \frac{1}{2}$$

for any subset of $k \leq m$ variables $X_{i_1}, \ldots, X_{i_k}$ and any Boolean functions $\phi(X_{i_1}, \ldots, X_{i_k})$, then F is m-balanced. Assuming the converse, one fixes some m variables, e.g. $(X_1, \ldots, X_m) = (a_1, \ldots, a_m)$, one gets a function

$$F(a_1, \ldots, a_m, X_{m+1}, \ldots, X_n)$$

which is not balanced. We now define the Boolean function $\phi(X_1, \ldots, X_m)$ by the rule $\phi(b_1, \ldots, b_m) =$

$$\begin{cases} 1, & \text{if } Pr(F(b_1, \ldots, b_m, X_{m+1}, \ldots, X_n) = 1) \geq \frac{1}{2}; \\ 0, & \text{if } Pr(F(b_1, \ldots, b_m, X_{m+1}, \ldots, X_n) = 0) > \frac{1}{2}. \end{cases}$$

for all binary string $b_1, \ldots, b_m$. We realize that

$$Pr(\phi(b_1, \ldots, b_m) = F(b_1, \ldots, b_m, X_{m+1}, \ldots, X_n)) \geq \frac{1}{2}$$

and

$$Pr(\phi(a_1, \ldots, a_m) = F(a_1, \ldots, a_m, X_{m+1}, \ldots, X_n)) > \frac{1}{2}.$$

This implies that

$$Pr(\phi(X_1, \ldots, X_m) = F(X_1, \ldots, X_n)) =$$
$$\frac{1}{2^m} \sum_{b_1, \ldots, b_m} Pr(\phi(b_1, \ldots, b_m) = F(b_1, \ldots, b_m, X_{m+1}, \ldots, X_n)) > \frac{1}{2}.$$

This contradicts with the assumption on F. The Theorem is proved.

2.15 Nonlinear feedback shift registers

We start with a definition of a more general device than LFSR. A feedback shift register of length L consists of L delay elements numbered $0, 1, \ldots, L-1$, storing one bit and having one input and one output. The sequence $s_{L-1}, s_{L-2}, \ldots, s_0$ is the initial state of the register. During each moment of time the content of stage 0 is output and forms output sequence, the content of stage i is moved to stage $i-1$ for each $1 \le i \le L-1$, the new content of stage $L-1$ is the feedback bit $s_{j+L} = f(s_{j+L-1}, s_{j+L-2}, \ldots, s_j)$, where f is a Boolean function in L variables, see Fig.20. The output sequence of the register is $s = s_0, s_1, \ldots$.

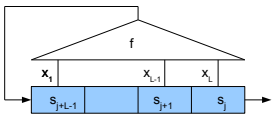

Figure 20: NFSR.

Also we consider the sequence of the NFSR internal states:

$$S_0 = (s_{L-1}, s_{L-2}, \ldots, s_0), \quad S_1 = (s_L, s_{L-1}, \ldots, s_1), \quad S_2 = (s_{L+1}, s_L, \ldots, s_2), \ldots$$

A property similar to LFSR holds for NFSR:

Lemma 20 *Let l be the tail length of the sequence s and t is its minimal period. Let $S = S_0, S_1, S_2, \ldots$ be the sequence of the related NFSR states, where l_1 is the tail length and t_1 is its minimal period. Then $l = l_1$ and $t = t_1$.*

A NFSR is called nonsingular if for each initial state $S_0 = (s_{L-1}, s_{L-2}, \ldots, s_0)$ the sequence $s = s_0, s_1, \ldots$ is pure periodic.

Lemma 21 *A NFSR with the feedback function $f(x_1, x_2, \ldots, x_L)$ is non-singular if and only if*

$$f(x_1, x_2, \ldots, x_L) = x_L \oplus g(x_1, x_2, \ldots, x_{L-1}).$$

Proof A NFSR is nonsingular if and only if it is reversible. That is for any state $v = (v_{L-1}, v_{L-2}, \ldots, v_0)$ there is only one state $u = (u_{L-1}, u_{L-2}, \ldots, u_0)$ such that $u \to v$ in the graph of the register states. In other words,

$$(f(u_{L-1}, \ldots, u_0), u_{L-1}, \ldots, u_1) = (v_{L-1}, v_{L-2}, \ldots, v_0).$$

Consider the decomposition

$$f(x_1, x_2, \ldots, x_L) = x_L h(x_1, x_2, \ldots, x_{L-1}) \oplus g(x_1, x_2, \ldots, x_{L-1}),$$

where h, g are Boolean functions in $x_1, x_2, \ldots, x_{L-1}$. Two cases:

1. $h = 1$, then given v one constructs only one previous state

$$(v_{L-2}, \ldots, v_0, v_{L-1} \oplus g(v_{L-2}, \ldots, v_0)).$$

2. $h \neq 1$, then there is $(v_{L-2}, \ldots, v_1)$ such that $h(v_{L-2}, \ldots, v_0) = 0$ and one constructs two previous states for

$$(v_{L-1}, v_{L-2}, \ldots, v_0), \quad v_{L-1} = g(v_{L-2}, \ldots, v_0).$$

They are $(v_{L-2}, \ldots, v_1, 0)$ and $(v_{L-2}, \ldots, v_1, 1)$. This proves the statement.

If the period of the output sequence is 2^L then the NFSR is called a de Bruijn FSR and the sequence is called a de Bruijn sequence. As an example, consider the FSR of length 3 with nonlinear function $f(x_1, x_2, x_3) = 1 \oplus x_2 \oplus x_3 \oplus x_1x_2$ and its deployment from the initial state $(0, 0, 0)$.

$$\begin{aligned}(0,0,0) \to (1,0,0) \to (1,1,0) \to (1,1,1) \to (0,1,1) \\ \to (1,0,1) \to (0,1,0) \to (0,0,1) \to (0,0,0).\end{aligned}$$

Because the period of a de Bruijn sequence is 2^L and states are L-strings, all L-strings occur once on the period. So de Bruijn sequence has nice statistical properties. That is each sequence of length k appears exactly 2^{L-k} times as a subsequence of a de Bruijn sequence of period 2^L and length $2^L + k - 1$.

We will address the question how to construct a de Bruijn sequence. Assume a LFSR of length L and of period $2^L - 1$, and with linear feedback function $f(x_1, x_2, \ldots x_L) = c_1x_1 + \ldots + c_{L-1}x_{L-1} + x_L$. Then an NFSR with the feedback function

$$f'(x_1, x_2, \ldots x_L) = f(x_1, x_2, \ldots x_L) + (x_1 + 1) \ldots (x_{L-1} + 1)$$

is a de Bruijn NFSR. This is because

$$f'(x_1, x_2, \ldots x_L) = \begin{cases} f(x_1, x_2, \ldots x_L), & x_1, \ldots, x_{L-1} \neq 0, \ldots, 0; \\ f(x_1, x_2, \ldots x_L) + 1, & x_1, \ldots, x_{L-1} = 0, \ldots, 0. \end{cases}$$

The graph of the internal states of the FSR is in Figure 21

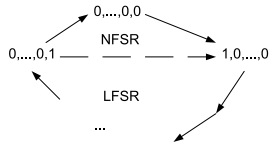

Figure 21: de Bruijn sequence construction.

We see that all edges except two are produced with a LFSR, so this NFSR acts as a LFSR for most states. This property isn't desirable for cryptographic applications. Consider a more general construction. Let a nonsingular NFSR states graph comprises edges:

$$\begin{aligned} x = (s_{L-1}, \ldots, s_1, s_0) &\rightarrow y = (s_L, s_{L-1}, \ldots, s_1) \\ \bar{x} = (s_{L-1}, \ldots, s_1, s_0 \oplus 1) &\rightarrow \bar{y} = (s_L \oplus 1, s_{L-1}, \ldots, s_1). \end{aligned}$$

When changing the feedback function to

$$f'(x_1, x_2, \ldots x_L) = f(x_1, x_2, \ldots x_L) + (x_1 + s_{L-1} + 1) \ldots (x_{L-1} + s_1 + 1)$$

the above edges change to

$$x \rightarrow \bar{y}, \quad \bar{x} \rightarrow y.$$

All other edges remain intact. The operation results in producing one cycle from some two different cycles if x and $\bar{x}$ are on different cycles. If x and $\bar{x}$ are on the same cycle, then this cycle decomposes into two. It is interesting question if a de Bruijn NFSR is produced from any nonsingular NFSR with a sequence of such operations. Even yes it is difficult in practice to find this sequence for a large register as one should know its ciclic structure. Also it is an open problem to construct an explicit de Bruijn NFSR of large length with a simple feedback function, e.g. of low algebraic degree like quadratic or qubic.

2.16 Clock-Controlled Generators

In regularly clocked LFSRs like combiners all LFSRs clock regularly, the movement of data is controlled by the same clock. A nonlinearity is introduced by using nonlinear Boolean functions as filters. The main idea behind a clock-controlled generator is in introducing nonlinearity by having the output of one LFSR controls the clocking of the other LFSRs. We will consider Alternating Step Generator. The generator works as follows. When the $LFSR_1$ is clocked

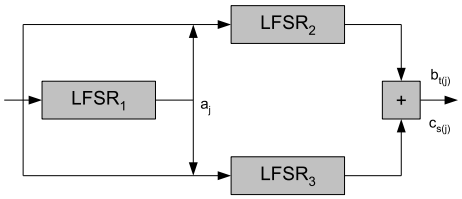

Figure 22: Alternating Step Generator.

and its output is 1, then $LFSR_2$ is clocked. $LFSR_3$ is not clocked but its previous output bit is repeated, for the first clock the previous bit is taken 0. When the output bit of $LFSR_1$ is 0, then $LFSR_3$ is clocked. $LFSR_2$ is not clocked but its previous output bit is repeated, for the first clock the previous bit is taken 0. The output of $LFSR_2$ and $LFSR_3$ are XORed and the result is a key-stream digit. Let

$$a_0, a_1, a_2, \ldots$$
$$b_0, b_1, b_2, \ldots$$
$$c_0, c_1, c_2, \ldots$$

be the output sequences of $LFSR_1$,$LFSR_2$ and $LFSR_3$ respectively. At clock j the output of the second register is $b_{t(j)}$, where $t(j) = \sum_{i=0}^{j} a_i - 1$, that is the number of clocks when the second register moves diminished by -1. Really, when $a_{i_1} = 1$ for the first time the output of $LFSR_2$ is b_0, when $a_{i_2} = 1$ for the second time the output of $LFSR_2$ is b_1,... and when $a_{i_r} = 1$ for the j-th time the output of $LFSR_2$ is b_{r-1}. So that $r = \sum_{i=0}^{j} a_i$ and the above formula is true. Similarly, at clock j the output of the third register is $c_{s(j)}$, where $s(j) = \sum_{i=0}^{j} \bar{a}_i - 1$. Then

$$t(j) + s(j) = \sum_{i=0}^{j} a_i - 1 + \sum_{i=0}^{j} \bar{a}_i - 1 = j - 1.$$

So $s(j) = j - t(j) - 1$.

Suppose that $LFSR_1$ produces a de Bruijn sequence of period 2^{L_1}, that is one obtains a de Bruijn sequence from a maximal period $LFSR_1$, and $LFSR_2$ and $LFSR_3$ are maximum-period LFSRs of co-prime lengths L_2 and L_3. Then the output sequence of the Alternating Step Generator is $2^{L_1}(2^{L_2}-1)(2^{L_3}-1)$ and its linear complexity L satisfies

$$(L_2+L_3)2^{L_1-1} < L < (L_2+L_3)2^{L_1}.$$

This fact implies that one needs at least $2(L_2+L_3)2^{L_1}$ key-stream digits in order to predict all other of them and the complexity of the attack is $O((L_2+L_3)^2 2^{2L_1})$ binary operation with the Berlekamp-Massey algorithm. But there is a simpler method. This is an attack against the initial states S_1^0, S_2^0, S_3^0 of the registers, given a segment of the key-stream $x_0, x_1, x_2, \ldots$ of size N a bit more than L_2+L_3.

1. Try all nonzero binary L_1-strings for the initial state of the first register and produce the output sequence $a_0, a_1, a_2, \ldots$.

2. Construct the system of linear equations

   $$x_j = b_{t(j)} \oplus c_{j-t(j)-1}$$

   because the attacker knows $t(j) = \sum_{i=0}^{j} a_i - 1$. In the above equation $b_{t(j)}$ is a known linear function in S_2^0 and $c_{j-t(j)-1}$ is a known linear function in S_3^0. So one needs about L_2+L_3 such equations to find unique solution.

3. Solve the system of linear equations with Gaussian elimination and get a guess on S_1^0, S_2^0, S_3^0.

4. Check the guess by using the rest of the key-stream or by decrypting cipher-text to somewhat sensible.

Example. Let $L_1 = 3$ and $LFSR_1$ is defined by the polynomial X^3+X+1 and the initial state $0, 0, 1$. Then the key-stream generation is as follows.

a_j	1	0	0	1	0	1	1	1
$b_{t(j)}$	b_0	b_0	b_0	b_1	b_1	b_2	b_3	b_4
$c_{j-t(j)-1}$	0	c_0	c_1	c_1	c_2	c_2	c_2	c_2
x_j	b_0	$b_0 \oplus c_0$	$b_0 \oplus c_1$	$b_1 \oplus c_1$	$b_1 \oplus c_2$	$b_2 \oplus c_2$	$b_3 \oplus c_2$	$b_4 \oplus c_2$

2.17 Shrinking Generator

In shrinking generator one LFSR controls the output of another.

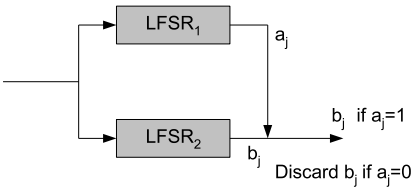

Figure 23: Shrinking Generator.

Example.

a_j	1	1	0	0	0	1	0	1	1	1	...
b_j	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	...
x_j	b_0	b_1				b_5		b_7	b_8	b_9	...

Let $LFSR_i$ produce maximum period sequences and their lengths L_1 and L_2 are co-prime, then the output sequence x_j has the period $(2^{L_2}-1)2^{L_1-1}$. The linear complexity L of the key-stream satisfies

$$L_2 2^{L_1-2} \leq L \leq L_2 2^{L_1-1}.$$

The latter fact implies that the Berlekamp-Massey algorithm requires at least $2L_2 2^{L_1-2}$ digits of the key-stream in order to predict all other of them with complexity about $O(L_2^2 2^{2(L_1-2)})$ bit operations. But there is a faster algorithm to find the initial state of the LFSRs only requiring a bit more than $2L_2$ key-stream digits.

Problem 10 *Find the above mentioned method.*

2.18 Arithmetic Summators

In this Section we will see that usual arithmetic operations are source of infinite binary sequences with high linear complexity and that combining sequences with arithmetic operations highly increases the linear complexity. We will introduce some new notions.

Let an infinite binary sequence be given:

$$a_0, a_1, a_2, \ldots,$$

where $a_i \in \{0, 1\}$. We relate with this sequence a formal number:

$$a = a_0 + a_1 2 + a_2 2^2 + \ldots.$$

When a_i is an eventually zero sequence, then a is well defined natural number or zero. But generally, a is a number of a new kind called 2-adic number. The set of all so defined 2-adic numbers is commonly denoted by Z_2. One says that it is a set of all integer 2-adic numbers because there are non-integer 2-adic numbers as well.

Any 2-adic numbers may be added, subtracted, multiplied and in many cases divided by each other like ordinary integer numbers, that is with carries.

1. Addition. Let $c = a + b$, where

   $$a = a_0 + a_1 2 + a_2 2^2 + \ldots \quad \text{and} \quad b = b_0 + b_1 2 + b_2 2^2 + \ldots,$$

   so that $c = c_0 + c_1 2 + c_2 2^2 + \ldots$. Let u_n be a carry digit. That is $u_0 = 0$ and

   $$\begin{aligned} c_n &= a_n \oplus b_n \oplus u_n, \\ u_{n+1} &= a_n b_n \oplus a_n u_n \oplus b_n u_n. \end{aligned}$$

   One sees that if a and b are natural numbers(finite sequences), then $c = a + b$ is a finite sequence but may be of one bit longer.

2. Subtraction. Let $c = a - b$, where v_n be a carry digit. That is $v_0 = 0$ and

   $$\begin{aligned} c_n &= a_n \oplus b_n \oplus v_n, \\ v_{n+1} &= a_n b_n \oplus a_n v_n \oplus b_n v_n \oplus b_n \oplus v_n. \end{aligned}$$

   In order to prove the last formula one writes $a = c + b$ and

   $$\begin{aligned} a_n &= c_n \oplus b_n \oplus u_n, \\ u_{n+1} &= c_n b_n \oplus c_n u_n \oplus b_n u_n \\ &= (a_n \oplus b_n \oplus u_n) b_n \oplus (a_n \oplus b_n \oplus u_n) u_n \oplus b_n u_n \\ &= a_n b_n \oplus a_n u_n \oplus b_n u_n \oplus b_n \oplus u_n. \end{aligned}$$

   One now puts $v_n = u_n$. The formula is proved.

Example. It is easy to see that

$$\begin{aligned} -1 &= 1+2+2^2+2^3+2^4\ldots, \\ -2 &= 2+2^2+2^3+2^4\ldots, \\ -3 &= -1+(-2)=1+(-2)2 \\ &= 1+2^2+2^3+2^4\ldots \end{aligned}$$

3. Multiplication. Let $c = ab$ and w_n be a carry digit, where $w_0 = 0$. Therefore,

$$\begin{aligned} c_n &= a_n b_0 \oplus a_{n-1}b_1 \oplus \ldots a_0 b_n \oplus w_n, \\ w_{n+1} &= \frac{a_n b_0 + a_{n-1}b_1 + \ldots a_0 b_n + w_n - c_n}{2}. \end{aligned}$$

So w_{n+1} is a natural number.

4. Division. Let $c = a/b$. This is only possible if b is an odd number, that is $b = 1 + b_1 2 + b_2 2^2 + \ldots$. One writes $a = cb$ and as previously

$$\begin{aligned} a_n &= c_n \oplus c_{n-1}b_1 \oplus \ldots c_0 b_n \oplus w_n, \\ w_{n+1} &= \frac{c_n + c_{n-1}b_1 + \ldots c_0 b_n + w_n - a_n}{2} \end{aligned}$$

In other words,

$$\begin{aligned} c_n &= a_n \oplus c_{n-1}b_1 \oplus \ldots c_0 b_n \oplus w_n, \\ w_{n+1} &= \lceil \frac{c_{n-1}b_1 + \ldots c_0 b_n + w_n - a_n}{2} \rceil \end{aligned}$$

For $b = 1 + b_1 2 + b_2 2^2 + \ldots + b_r 2^r$ we have

$$\begin{aligned} c_n &= a_n \oplus c_{n-1}b_1 \oplus \ldots c_{n-r} b_r \oplus w_n, \\ w_{n+1} &= \lceil \frac{c_{n-1}b_1 + \ldots c_{n-r} b_r + w_n - a_n}{2} \rceil. \end{aligned}$$

This looks a recurrence to produce c_i. It is implemented with a Feedback with Carry Shift Register. The initial state of the device is

$$(c_{-1}, \ldots, c_{-r+1}, c_{-r}) = (0, \ldots, 0, 0)$$

and $w_0 = 0$. We discard the first r digits of the output sequence and get $c_0, c_1, \ldots$. An important question is the size of the carry-function. We will show that $0 \le w_n \le r + 2$. Really,

$$\begin{aligned} 0 \le w_n \le \frac{\sum_{i=1}^{r} c_{n-1-i}b_i + 2 + w_{n-1}}{2} &\le \frac{r + 2 + w_{n-1}}{2} \\ &\le (r+2)(\frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^{n-1}}) < r + 2. \end{aligned}$$

Feedback with Carry Shift registers enable producing binary sequences from rational numbers with odd denominators. Two cases:

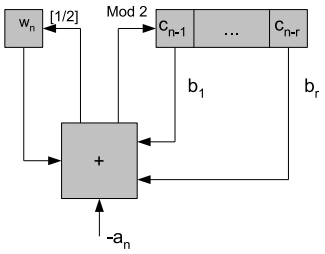

Figure 24: Feedback with Carry Shift Register.

1. $a \geq 0$, so $a = a_0 + a_1 2 + \ldots + a_s 2^s$. For initialization one inputs $-a_0, -a_1, \ldots - a_s$ during the first $s+1$ clocks. Then the device works autonomously. We expand $\frac{3}{5}$ with the device Fig.24 for $r = 2$:

n	0	1	2	3	4	5	6	7	8	9	10	11
c_{n-1}	0	1	1	1	0	0	1	1	0	0	1	1
c_{n-2}	0	0	1	1	1	0	0	1	1	0	0	1
$-a_n$	-1	-1	0	0	0	0	0	0	0	0	0	0
w_n	0	0	0	1	1	1	1	1	1	1	1	1

.

The output sequence is eventually periodic with the minimal period 4 and the tail of length 1:

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

2. $a < 0$, so $a = -a_0 - a_1 2 - \ldots - a_s 2^s$ for nonnegative a_i. Then for initialization one inputs $a_0, a_1, \ldots a_s$ during the first $s+1$ clocks. Then the device works autonomously. We expand $\frac{-3}{5}$ with the device Figure 24 for $r = 2$:

n	0	1	2	3	4	5	6	7	8	9	10	11
c_{n-1}	0	1	0	0	1	1	0	0	1	1	0	0
c_{n-2}	0	0	1	0	0	1	1	0	0	1	1	0
$-a_n$	1	1	0	0	0	0	0	0	0	0	0	0
w_n	0	1	1	1	1	1	1	1	1	1	1	1

.

The output sequence is pure periodic with the minimal period 4 :

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

Let $\frac{a}{b}$ be any rational number with an odd denominator and we expand it with a FCSR to get a 2-adic representation:

$$\frac{a}{b} = c_0 + c_1 2 + c_2 2^2 + \ldots$$

Is the sequence periodic or not? If yes, what its minimal period is. To answer the first question we realise the FCSR has a finite number of states. Really, the internal state is represented by numbers $w_n, c_{n-1}, \ldots, c_{n-r}$, where $0 \leq w_n \leq r+2$ and $c_i \in \{0,1\}$. So the number of states is at most $(r+2)2^r$. When the FCSR works autonomously the output sequence is at least eventually periodic, as when the internal state repeats the output digit is the same. This implies that the period is at most $(r+2)2^r$. We will prove a reverse statement that any eventually periodic sequence is the expansion of a rational number.

Lemma 22 *Let $c_0, c_1, c_2, \ldots$ be any eventually periodic binary sequence. Then there exists a rational number $\frac{a}{b}$ with odd denominator b such that $\frac{a}{b} = c_0 + c_1 2 + c_2 2^2 + \ldots$ is its 2-adic expansion.*

Proof. Let t be a period of $c = c_0, c_1, c_2, \ldots$. Then

$$c = c_0, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{i+t-1}, c_{i+t}, c_{t+i+1}, \ldots, c_{i+2t-1}, \ldots,$$

where $c_i, c_{i+1}, \ldots, c_{i+t-1}$ is a period. We denote

$$\begin{aligned} X_0 &= c_0 + c_1 2 + \ldots + c_{i-1} 2^{i-1}, \\ X_1 &= c_i + c_{i+1} 2 + \ldots + c_{i+t-1} 2^{t-1}. \end{aligned}$$

These are natural numbers or zeros. So 2-adic number c is represented

$$\begin{aligned} c &= X_0 + 2^i (X_1 + X_1 2^t + \ldots + X_1 2^{2t} + \ldots) \\ &= X_0 + 2^i X_1 (1 + 2^t + 2^{2t} + \ldots) \\ &= X_0 + 2^i \frac{X_1}{1-2^t} = \frac{X_0(1-2^t) + 2^i X_1}{1-2^t}. \end{aligned}$$

This is a rational number. The Lemma is proved.

Example. Let $c = 11001001001\ldots$, then

$$c = \frac{(1-2^3)+2}{1-2^3} = \frac{5}{7}.$$

Lemma 23 *Let $b > 0$ and $\gcd(a,b) = 1$. The rational number $\frac{a}{b}$ expands to a pure periodic sequence if and only if $-b \leq a \leq 0$. The minimal period t is the minimal natural number such that $2^t \equiv 1 (\mod b)$.*

Proof. Let $-b \leq a \leq 0$ and $2^t \equiv 1 (\mod b)$. Let us denote $c = \frac{2^t-1}{b}$. Then

$$\frac{a}{b} = \frac{ac}{bc} = \frac{ac}{2^t - 1} = \frac{d}{2^t - 1}.$$

So $-(2^t - 1) \leq d \leq 0$. Therefore $-d = d_0 + d_1 2 + \ldots + d_{t-1} 2^{t-1}$ and

$$\begin{aligned} \frac{a}{b} &= \frac{-d}{1-2^t} = -d(1 + 2^t + 2^{2t} + \ldots) \\ &= d_0 + d_1 2 + \ldots + d_{t-1} 2^{t-1} + d_0 2^t + d_1 2^{t+1} + \ldots + d_{t-1} 2^{2t-1} + \ldots \end{aligned}$$

To prove the last statement let t_0 be the minimal period for $\frac{a}{b}$. Assume $t_0 < t$. Then by Lemma 22

$$\frac{a}{b} = \frac{e}{2^{t_0} - 1}.$$

Therefore $a(2^{t_0} - 1) = be$. So b divides $2^{t_0} - 1$ which contradicts the minimality of t.

It is very plausible the linear complexity of the sequence generated from $\frac{a}{b}$ is about the period t.

Using FCSRs is a good way to produce long period sequences of high linear complexity. For instance, take the prime number $2^{67} + 3$. One checks that $t = 2^{67} + 2$ is the minimal natural number such that $2^t \equiv 1(\mod 2^{67} + 3)$. Let us expand the rational number $\frac{-1}{2^{67}+3}$. This results in a pure periodic binary sequence of period $2^{67} + 2$.

Remark 3 *One must not use any output of a FCSR as a key-stream, because in spite of its hight linear complexity the sequence is predictable. Given a segment of the output of length r, one uses this as an internal state of the register. One doesn't know carry which is a number from the interval $0 \leq w_n \leq r + 2$. So one guesses the carry in order to get the full internal state of the FCSR.*

2.19 Summation Generator.

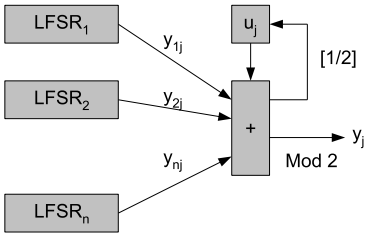

Figure 25: Summation Generator.

The number u_j is called carry, it changes at each clock. Initially, $u_0 = 0$. Then

$$y_j = y_{1j} \oplus \ldots \oplus y_{nj} \oplus u_j$$

and

$$u_{j+1} = \frac{y_{1j} + \ldots + y_{nj} + u_j - y_j}{2} = \lfloor \frac{y_{1j} + \ldots + y_{nj} + u_j}{2} \rfloor.$$

We will answer what the period of the output sequence is.

Lemma 24 *Let the minimal period of the sequence y_{ik} be T_i and the numbers $T_1, \ldots, T_2$ are pairwise coprime. Then the minimal period of the output sequence y_k is $T_1 T_2 \ldots T_n$.*

Proof. One sees that $\{y_k\}$ is the sum of $\{y_{ik}\}$ as 2-adic numbers. Let $\{y_{ik}\}$ be represented by the numbers $\frac{a_i}{b_i}$, where $\gcd(a_i, b_i) = 1$. So b_i divides $2^{T_i} - 1$, where T_i is the minimal such the number. So $\{y_k\}$ is the expansion of the number

$$\frac{a_1}{b_1} + \ldots + \frac{a_n}{b_n} = \frac{a}{b},$$

where $b = b_1 b_2 \ldots b_n$ and $a = a_1 b_2 \ldots b_n + a_2 b_1 b_3 \ldots b_n + \ldots + b_1 \ldots b_{n-1} a_n$ are coprime. Really, let q be a prime number which is a common divisor of a and b. Let q divide b_1, so q divides $a_1 b_2 \ldots b_n$. As q does not divide a_1, so q divides $b_2 \ldots b_n$ and q divides some of the $b_2, \ldots, b_n$, e.g. q divides b_2. Then as b_i divides $2^{T_i} - 1$, the number q is a common divisor of $2^{T_1} - 1$ and $2^{T_2} - 1$ which are coprime as

$$\gcd(2^{T_1} - 1, 2^{T_2} - 1) = \gcd(2^{\gcd(T_1, T_2)} - 1) = 1$$

because $\gcd(T_1, T_2) = 1$. So $q = 1$ and the numbers a and b are coprime.

The minimal period of $\frac{a}{b} = y_0, y_1, \ldots$ is the minimal natural number T such that $2^T \equiv 1 (\mod b)$. As b_1 divides b, we have b_1 divides $2^T - 1$. Let us represent $T = cT_1 + d$, where $0 \leq d < T_1$. So

$$2^T - 1 = 2^{cT_1 + d} - 1 \equiv 2^d - 1 (\mod b_1) \equiv 0.$$

So b_1 divides $2^d - 1$. By the definition of T_1,$d = 0$ and, therefore, T_1 divides T. Similarly, all other T_i divide T. As they are pairwise coprime, $T_1T_2 \dots T_n$ divides T.

On the other hand,

$$b_i | 2^{T_i} - 1 | 2^{T_1T_2 \dots T_n} - 1.$$

As b_i are pairwise coprime, $b = b_1 \dots b_n$ divides $2^{T_1T_2 \dots T_n} - 1$. Therefore, $T_1T_2 \dots T_n$ is a period of $\{y_k\}$ and hence T divides $T_1T_2 \dots T_n$. So $T = T_1T_2 \dots T_n$.

Example. 30 digits of the output and intermediate sequences are in the following table.

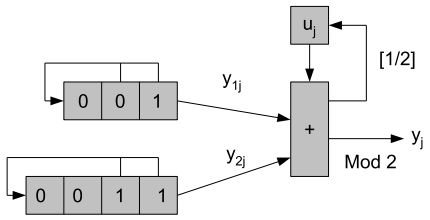

Figure 26: Summation Generator. Example.

$\{y_{1k}\}$	1001011100101110010111001011 10
$\{y_{2k}\}$	1100010011010111100010011010 11
$\{y_k\}$	0011000000000110001100110100 10

The period of $\{y_{1k}\}$ is $2^3 - 1 = 7$ and the period of $\{y_{2k}\}$ is $2^4 - 1 = 15$. They are coprime, so the output period equals 105.

The summation generators provide with sequences of large minimal periods and high linear complexity, about the period of the sequence by conjecture. This implies that to predict the output sequence with the Berlekamp-Massey one needs about the whole period of the sequence. So this is not sensible. As we will see soon there exists an attack based on 2-adic approximation. Another advantage of the summation generator is that the output sequence $\{y_k\}$ is not correlated with any sequence $\{u_k = \varphi(y_{1k}, \dots, y_{nk})\}$, where φ is any Boolean function. That is this combiner with memory can't be approximated by any memoryless combiner.

2.19.1 2-adic approximation

Consider the problem. Given a summation generator with periods T_i of pure periodic sequences $\{y_{ik}\}$ and a number of the output digits of the key-stream $y^s = y_0, y_1, \ldots, y_{s-1}$, predict all other digits. We know that $\{y_k\}$ is the expansion of the number

$$\frac{a_1}{b_1} + \ldots + \frac{a_n}{b_n} = \frac{a}{b},$$

where $-b_i < a_i < 0$, as $\{y_{ik}\}$ are pure periodic. Then $\gcd(a, b) = 1$ and

$$1 \le b \le b_1 b_2 \ldots b_n = B \le \prod_{i=1}^{n} (2^{T_i} - 1)$$

and $1 \le |a| < nB$. Surprisingly, to compute the rational number $\frac{a}{b}$ one needs much less bits of the output sequence than its tail and the period. Consider

$$\frac{a}{b} = y_0 + y_1 2 + \ldots + y_{s-1} 2^{s-1} + y_s 2^s + \ldots$$

modulo 2^s. Denote $y_0 + y_1 2 + \ldots + y_{s-1} 2^{s-1} = Y_s$. We get $\frac{a}{b} \equiv Y_s (\mod 2^s)$ or

$$a \equiv b Y_s (\mod 2^s). \tag{22}$$

Given Y_s, we will show how to find a and b. Generally, this can't be solved uniquely, but the additional information $|a| < nB$ and $|b| \le B$ helps once s is big enough. Let

$$L_s = \{(u, v) \in \mathbf{Z}^{[2]} | u \equiv v Y_s (\mod 2^s)\}$$

denote the lattice of the integer solutions to the congruence (22). So $(a, b) \in L_s$ for any s. The basis of L_s is

$$\begin{aligned} A_1 &= (Y_s, 1) \\ A_2 &= (2^s, 0). \end{aligned}$$

Really, if $(u, v) \in L_s$, then $u = v Y_s + 2^s h$ for some integer number h. Then

$$(u, v) = v(Y_s, 1) + h(2^s, 0).$$

The measure of the vector is its length $|(u, v)| = (u^2 + v^2)^{\frac{1}{2}}$. E.g. the length of (a, b) is bounded by $(n^2 B^2 + B^2)^{\frac{1}{2}}$. The measure(determinant) of the lattice is the area of the parallelogram whose sides are determined by the basis vectors. The determinant of L_s is the absolute value of the determinant of the matrix $\begin{pmatrix} Y_s & 1 \\ 2^s & 0 \end{pmatrix}$ which is 2^s. The idea of the 2-adic approximation is in the following. Given more and more bits of the output, one constructs the bigger and bigger lattice L_s. All these lattices comprise the vector (a, b) of bounded length. Then for some s the vector (a, b) becomes the shortest nonzero vector of L_s and it can be produced with the Lagrange-Gauss reduction algorithm.

2.20 Some facts about lattices

Let

$$(u_1, v_1), \quad (u_2, v_2)$$

be two integer vectors linearly independent over reals. The lattice generated by these vectors is defined as

$$L = \{n(u_1, v_1) + m(u_2, v_2) | n, m \in \mathbf{Z}\},$$

where $\mathbf{Z}$ is the set of integer numbers. The determinant of L is

$$\Delta(L) = |\det \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix}|,$$

the absolute value of the determinant of the matrix $\begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix}$. This number doesn't depend on the basis. Really, if there is another basis

$$(a_1, b_1) = n_1(u_1, v_1) + m_1(u_2, v_2),$$
$$(a_2, b_2) = n_2(u_1, v_1) + m_2(u_2, v_2),$$

then the matrix $\begin{pmatrix} n_1 & m_1 \\ n_2 & m_2 \end{pmatrix}$ is invertible over $\mathbf{Z}$. Therefore, its determinant is ± 1. So as

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} = \begin{pmatrix} n_1 & m_1 \\ n_2 & m_2 \end{pmatrix} \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix}$$

we have

$$det \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} = \pm det \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix}.$$

Theorem 21 *Let (a, b) be the shortest nonzero vector of a lattice L. Then*

$$|(a, b)| \leq \left(\frac{4}{3}\right)^{1/4} \Delta(L)^{1/2}.$$

By definition, the basis A_1, A_2 of the lattice L is reduced if

$$|A_1| \leq |A_2| \leq |A_2 + xA_1|$$

for any integer x.

Lemma 25 *$|A_2| \leq |A_2 + xA_1|$ for any integer x if and only if $\frac{|(A_1, A_2)|}{|A_1|^2} \leq \frac{1}{2}$.*

Proof. We have $|A_2 + xA_1| = |A_2|^2 + 2x(A_1, A_2) + x^2|A_1|^2$. So $|A_2| \leq |A_2 + xA_1|$, for integers x if and only if

$$2x(A_1, A_2) + x^2|A_1|^2 \geq 0$$

for integers x. The latter is true if and only if

$$\frac{|(A_1, A_2)|}{|A_1|^2} \leq \frac{1}{2}.$$

Lemma 26 *Let A_1, A_2 be a reduced basis of a lattice L. Then A_1 is the shortest nonzero vector of L.*

Proof. By assumption

$$|A_1| \le |A_2| \le |A_2 + xA_1|$$

for all integer x. Let $C = nA_1 + mA_2$ be any nonzero vector of L. We will prove $|C| \ge |A_1|$. Consider two cases

1. $m = 0$, then $|C| = |nA_1| = n|A_1|$. Therefore, $|C| \ge |A_1|$.
2. $m \ne 0$, then we divide n by m with remainder:

$$n = am + b, \quad 0 \le b < |m|.$$

Then

$$|C| = |(am + b)A_1 + mA_2| = |m(aA_1 + A_2) + bA_1| \ge$$
$$|m(aA_1 + A_2)| - b|A_1| \ge |m||A_1| - b|A_1| = (|m| - b)|A_1| \ge |A_1|.$$

The Lemma is proved.

Lagrange-Gauss reduction algorithm.
input. The lattice L given by the basis A_1, A_2, where $|A_2| \ge |A_1|$.
output. A reduced basis of L.

1. Compute $r = \lfloor \frac{(A_1, A_2)}{|A_1|^2} \rceil$, where (A_1, A_2) denotes the dot product of A_1 and A_2. Put $a \leftarrow A_2 - rA_1$.
2. If $|a| < |A_1|$, then $A_2 \leftarrow A_1, A_1 \leftarrow a$, go to 1, else $A_2 \leftarrow a$ and terminate.

Theorem 22 *The algorithm terminates. Let A_1, A_2 be the output of the reductions algorithm. Then A_1, A_2 is a reduced basis.*

Proof. We realize that at each step of the algorithm

$$|A_2 - rA_1| \le |A_2|,$$

because $|A_2 - rA_1| = \min_x |A_2 - xA_1|$ over integers x. If $|A_2 - rA_1| < |A_2|$, then the algorithm does some steps and if $|A_2 - rA_1| = |A_2|$, then it terminates. Therefore, the length of A_2 is reducing with each step of the algorithm. Now as any lattice contains only finite number of vectors of bounded length, the algorithm should terminate at some point. We see that at each step of the algorithm A_1 and A_2 is a basis of L. Really, each step of the algorithm consists of

$$(A_1, A_2) \leftarrow (A_2 - rA_1, A_1) = (A_1, A_2) \begin{pmatrix} -r & 1 \\ 1 & 0 \end{pmatrix},$$

or

$$(A_1, A_2) \leftarrow (A_1, A_2 - rA_1) = (A_1, A_2) \begin{pmatrix} 1 & -r \\ 0 & 1 \end{pmatrix},$$

for some integer number r, where the matrices in the right hand sides are invertible over integer numbers. These means that A_1 and A_2 is a basis of L at each step of the algorithm. We will prove the final basis is reduced. The algorithm stops when $|A_2 - rA_1| \geq |A_1|$, where $r = \lfloor \frac{(A_1,A_2)}{|A_1|^2} \rceil$. One finds that $|A_2 - rA_1| = \min_x |A_2 - xA_1|$ over integers x. Therefore, $A_1, A_2 - rA_1$ is a reduced basis of L. The Theorem is proved.

The Theorem implies that the result of the Lagrange-Gauss reduction algorithm is a reduced basis. Therefore by Lemma 26 one of these vectors is the shortest nonzero vector of the lattice.

One also proves that the complexity of the Gaussian algorithm is $O(s^2)$ bit operations, where s is the number of bits to represent initial basis.

Example. Let $L = \{(a,b) | a \equiv 15y \pmod{23}\}$. The initial basis of the lattice is $A_1 = (15,1), A_2 = (23,0)$. We see that $|A_1| < |A_2|$. We compute

1. $r = \lfloor \frac{(A_1,A_2)}{|A_1|^2} \rceil = \lfloor \frac{15 \times 23}{15^2+1^2} \rceil = 2$. Then

$$a \leftarrow A_2 - 2A_1 = (-7,-2).$$

   As $|a| < |A_1|$ we put $A_1 \leftarrow (-7,-2), A_2 \leftarrow (15,1)$.

2. $r = \lfloor \frac{-15 \times 7 - 2}{7^2+2^2} \rceil = -2$. Then

$$a \leftarrow A_2 + 2A_1 = (1,-3).$$

   As $|a| < |A_1|$ we put $A_1 \leftarrow (1,-3), A_2 \leftarrow (-7,-2)$.

3. $r = \lfloor \frac{-7+6}{3^2+1^2} \rceil = 0$. Then $a \leftarrow A_2$. As $|a| \geq |A_1|$ we put $A_1 \leftarrow (1,-3), A_2 \leftarrow (-7,-2)$. The algorithm stops.

A reduced basis is found, where $(1,-3)$ is the shortest nonzero vector of L.

We will return to the 2-adic approximation attack. Let s be a natural number such that $B^2(n^2+1) < 2^s$ and in the lattice L_s there is another vector (a_1, b_1) such that

$$|(a_1,b_1)| \leq B(n^2+1)^{\frac{1}{2}}$$

and therefore $|a_1|, |b_1| \leq B(n^2+1)^{\frac{1}{2}}$. Then we have the system of equations:

$$a \equiv bY_s(\pmod{2^s}), \quad a_1 \equiv b_1 Y_s(\pmod{2^s}).$$

By eliminating Y_s, we get

$$b_1 a - a_1 b \equiv 0(\pmod{2^s}).$$

We estimate

$$|b_1 a - a_1 b| \leq |b_1 a| + |a_1 b| \leq B(n^2+1)^{\frac{1}{2}}(Bn + B).$$

So $|b_1 a - a_1 b| \leq (B(n+1))^2 < 2^s$. This implies that $b_1 a - a_1 b = 0$ and so (a_1, b_1) is a multiple of (a,b). Therefore (a,b), up to a multiple of -1, is the

unique nonzero vector of the lattice of length $\leq |(a,b)|$. So (a,b) is the unique, up to a multiple of -1, shortest nonzero vector of the lattice and it is computed with the reduction algorithm.

As $B = \prod_{i=1}^{n}(2^{T_i}-1)$, one sees that

$$s \geq 2(T_1 + T_2 + \ldots + T_n + \log_2(n+1)) \tag{23}$$

implies $(B(n+1))^2 < 2^s$. So s digits of the key-stream is enough to compute a and b. Then one expands the number $\frac{a}{b}$ with a FCSR to get the rest of the key-stream.

Example 1. Given the summation generator in Fig.26, where the LFSR's initial states are unknown, and the segment of the key-stream

$$y^{48} = 001100000000011000110011010010011110100001000100,$$

one predicts the other digits. We put $s = 48$ to provide $s \geq 2(7 + 15 + \log_2 3)$ and compute

$$Y_{48} = y_0 + y_1 2 + \ldots + y_{47}2^{47} = 37484642459660.$$

We apply the reduction algorithm to the lattice generated by $(Y_s, 1)$ and $(2^{48}, 0)$. The reduced basis is

$$A_1 = (-6923764, 4161409), \quad A_2 = (-70590152, -120186854).$$

The shortest of the two is the shortest nonzero vector in the lattice, it is (a,b). We expand

$$\frac{-6923764}{4161409} = 0011000000000110001100110101001\ldots,$$

this is our key-stream. So 48 bits of the key-stream is enough to predict the rest $57 = 105 - 48$ digits.

Example 2. Let we combine 6 sequences each of the maximal period $2^n - 1$, where $n = 7, 11, 13, 15, 16, 17$. So the minimal period of the output is about

$$(2^7-1)(2^{11}-1)(2^{13}-1)(2^{15}-1)(2^{16}-1)(2^{17}-1) \approx 5.99 \times 10^{23}.$$

We now take

$$s \geq 2(2^7 - 1 + 2^{11} - 1 + 2^{13} - 1 + 2^{15} - 1 + 2^{16} - 1 + 2^{17} - 1 + \log_2 7) \approx 5 \times 10^5.$$

So $\approx 5 \times 10^5$ digits of the key-stream is enough to predict all other $\approx 5.99 \times 10^{23}$ digits. The cost of the attack is $\approx (5 \times 10^5)^2 = 2.5 \times 10^{11}$ bit operations. As the length of the FCSR should be $\approx 5 \times 10^5$, predicting one bit takes the same number of bit operations.

For the above reasons the summation generator is not reliable by itself. It can't be used as a key-stream generator.