CHAPTER 10

Historical Stream Ciphers

Chapter Goals

- To introduce the general model of symmetric ciphers.
- To explain the relation between stream ciphers and the Vernam cipher.
- To examine the working and breaking of the Lorenz cipher in detail.

10.1. Introduction to Symmetric Ciphers

A symmetric cipher works using the following two transformations

$$c = e_k(m),$$
$$m = d_k(c)$$

where

- m is the plaintext,
- e is the encryption function,
- d is the decryption function,
- k is the secret key,
- c is the ciphertext.

It is desirable that both the encryption and decryption functions be public knowledge and so the secrecy of the message, given the ciphertext, depends totally on the secrecy of the secret key k. Although this well-established principle, called Kerckhoffs' principle, has been known since the mid-1800s some companies still ignore it and choose to deploy secret proprietary encryption schemes which usually turn out to be insecure as soon as someone leaks the details of the algorithms. The best schemes will be the ones which have been studied by many people for a very long time and which have been found to remain secure. A scheme which is a commercial secret cannot be studied by anyone outside the company.

The above set-up is called a symmetric key system since both parties need access to the secret key. Sometimes symmetric key cryptography is implemented using two keys, one for encryption and one for decryption. However, if this is the case we assume that, given the encryption key, it is easy to compute the decryption key (and vice versa). Later we shall meet public key cryptography where only one key is kept secret, called the private key; the other key, called the public key, is allowed to be published in the clear. In this situation it is assumed to be computationally infeasible for someone to compute the private key given the public key.

Returning to symmetric cryptography, a moment's thought reveals that the number of possible keys must be very large. This is because in designing a cipher we assume the worst-case scenario and give the attacker the benefit of

- full knowledge of the encryption/decryption algorithm,
- a number of plaintext/ciphertext pairs associated with the target key k.

If the number of possible keys is small then an attacker can break the system using an exhaustive search. The attacker encrypts one of the given plaintexts under all possible keys and determines which key produces the given ciphertext. Hence, the key space needs to be large enough to avoid such an attack. It is commonly assumed that a computation taking 2^{128} steps will be infeasible for a number of years to come, hence the key space size should be at least 128 bits to avoid exhaustive search.

The cipher designer must play two roles, that of someone trying to break a cipher, as well as someone trying to create one. These days, although there is a lot of theory behind the design of ciphers, we still rely on symmetric ciphers which are just believed to be strong, rather than ones for which we know a reason why they are strong. All this means is that the best attempts of the most experienced cryptanalysts cannot break them. This should be compared with public key ciphers and modes of operations of block ciphers, where there is now a theory which allows us to reason about how strong a given cipher is (given some explicit computational assumption on the underlying primitive).

Figure 10.1 describes a simple model for enciphering bits which, although simple, is quite suited to practical implementations. The idea of this model is to apply a reversible operation to

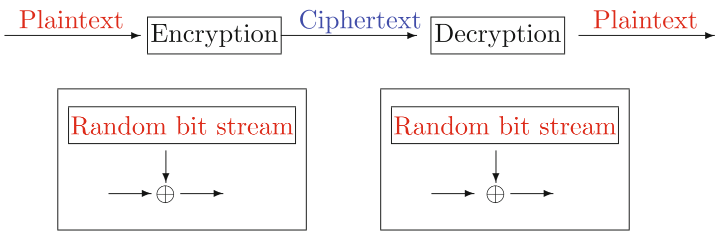

FIGURE 10.1. Simple model for enciphering bits

the plaintext to produce the ciphertext, namely combining the plaintext with a "random stream". The recipient can recreate the original plaintext by applying the inverse operation, in this case by combining the ciphertext with the same random stream.

This is particularly efficient since we can use the simplest operation available on a computer, namely exclusive-or $\oplus$. We saw in Chapter 9 that if the key is different for every message and the key is as long as the message, then such a system can be shown to be perfectly secure, namely we have the one-time pad. However, the one-time pad is not practical in many situations.

- We would like to use a short key to encrypt a long message.
- We would like to reuse keys.

Modern symmetric ciphers allow both of these possibilities, but thereby forfeit the perfect secrecy property. Such a trade-off is worthwhile because using a one-time pad produces horrendous key distribution problems. We shall see that key distribution is still problematic even for short, reusable keys.

There are a number of ways to attack a bulk cipher, some of which we outline below. We divide our discussion into passive and active attacks; a passive attack is generally easier to mount than an active attack.

- **Passive Attacks:** Here the adversary is only allowed to listen to encrypted messages. Then she attempts to break the cryptosystem by either recovering the key or determining some secret that the communicating parties did not want leaked. One common form of passive attack is that of traffic analysis, a technique borrowed from armies in World War

I, where a sudden increase in radio traffic at a certain point on the Western Front would signal an imminent offensive.

- **Active Attacks:** Here the adversary is allowed to insert, delete or replay messages between the two communicating parties. A general requirement is that an undetected insertion attack should require the breaking of the cipher, whilst the cipher needs to allow detection and recovery from deletion or replay attacks.

Bulk symmetric ciphers essentially come in two variants: stream ciphers, which operate on one data item (bit/letter) at a time, and block ciphers, which operate on data in blocks of items (e.g. 64 bits) at a time. In this chapter we look at historical stream ciphers, leaving modern stream ciphers until Chapter 12 and modern block ciphers until Chapter 13.

10.2. Stream Cipher Basics

Figure 10.2 gives a simple explanation of a stream cipher. This is very similar to our previous simple model, except the random bit stream is now produced from a short secret key using a public algorithm, called the keystream generator.

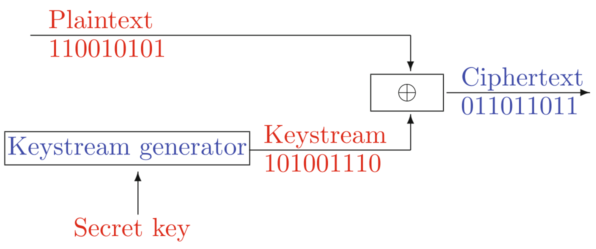

FIGURE 10.2. Stream ciphers

Thus we have $c_i = m_i \oplus k_i$ where

- $m_0, m_1, \ldots$ are the plaintext bits,
- $k_0, k_1, \ldots$ are the keystream bits,
- $c_0, c_1, \ldots$ are the ciphertext bits.

This means

$$m_i = c_i \oplus k_i$$

i.e. decryption is the same operation as encryption.

Stream ciphers such as that described above are simple and fast to implement. They allow very fast encryption of large amounts of data, so they are suited to real-time audio and video signals. In addition there is no error propagation; if a single bit of ciphertext gets mangled during transit (due to an attacker or a poor radio signal) then only one bit of the decrypted plaintext will be affected. They are very similar to the Vernam cipher mentioned earlier, except now the keystream is only pseudo-random as opposed to truly random. Thus whilst similar to the Vernam cipher they are *not* perfectly secure.

Just like the Vernam cipher, stream ciphers suffer from the following problem: the same key used twice gives the same keystream, which can reveal relationships between messages. For example suppose m_1 and m_2 were encrypted under the same key k, then an adversary could work out the exclusive-or of the two plaintexts without knowing what the plaintexts were

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2.$$

Hence, there is a need to change keys frequently either on a per message or on a per session basis. This results in difficult key management and distribution challenges, which, as we shall see later, can be addressed using public key cryptography. A typical strategy is to use public key cryptography to determine session or message keys, and then to rapidly encrypt the actual data using either a stream or block cipher.

The keystream produced by the keystream generator above needs to satisfy a number of properties for the stream cipher to be considered secure. As a bare minimum it should

- Have a long period. Since the keystream k_i is produced via a deterministic process from the key, there will exist a number N such that
$$k_i = k_{i+N}$$
for all values of i. This number N is called the period of the sequence, and should be large for the keystream generator to be considered secure.
- Have pseudo-random properties. The generator should produce a sequence which appears to be random, in other words it should pass a number of statistical random number tests.
- Have large linear complexity (see Chapter 12 for an explanation).

However, these conditions are not sufficient. Generally, determining more of the sequence from a part should be computationally infeasible. Ideally, even if one knows the first one billion bits of the keystream sequence, the probability of guessing the next bit correctly should be no better than one half.

In Chapter 12 we shall discuss how modern stream ciphers can be created using a combination of simple circuits called Linear Feedback Shift Registers. But first we will look at an earlier construction using rotor machines, or in modern nomenclature Shift Registers (i.e. shift registers with no linear feedback).

10.3. The Lorenz Cipher

The Lorenz cipher was a German cipher from World War II which was used for strategic information, as opposed to the tactical and battlefield information encrypted under the Enigma machine. The Lorenz machine was a stream cipher which worked on streams of bits. However it produced not a single stream of bits, but five. The reason was due to the encoding of the teleprinter messages used at the time, namely Baudot code.

10.3.1. Baudot Code: To understand the Lorenz cipher we first need to understand Baudot code. We all are aware of the ASCII encoding for the standard letters on a keyboard, which uses seven bits for the data, plus one bit for error detection. Prior to ASCII, indeed as far back as 1870, Baudot invented an encoding which used five bits of data. This was further developed until, by the 1930s, it was the standard method of communicating via teleprinter. The data was encoded via a tape, consisting of a sequence of five rows of holes/non-holes.

Those of us of a certain age in the United Kingdom can remember the football scores being sent in on a Saturday evening by teleprinter, and those who are even older can maybe recall the ticker-tape parades in New York: the ticker-tape was the remains of messages in Baudot code that had been transmitted between teleprinters. Those who can remember early dial-up modems will recall that the speeds were measured in Baud, or characters per second, in memory of Baudot's invention.

Since five bits does not allow one to encode all the characters that one wants, Baudot code used two possible "states" called *letters shift* and *figures shift*. Movement between the two states was controlled by control characters; a number of other control characters were reserved for things such as space (SP), carriage return (CR), line feed (LF) or a character which rang the teleprinter's bell (BELL) (such control codes still exist in ASCII)[1]. The table for Baudot code in the 1930s is presented in Table 10.1. Thus to transmit the message

[1]A line feed moves one line down, whereas a carriage return moves the cursor to the beginning of a line. These two teleprinter/typewriter codes still cause problems today. In Windows, text files use both codes to move down and

Please, Please Help!

one would need to transmit the encoding, which we give in hexadecimal,

16, 12, 01, 03, 05, 01, 1B, 0C, 1F, 04, 16, 12, 01, 03, 05, 01, 04, 14, 01, 12, 16, 1B, 0D.

Bits in Code					Hex	Letters	Figures
lsb				msb	Code	Shift	Shift
0	0	0	0	0	00	NULL	NULL
1	0	0	0	0	01	E	3
0	1	0	0	0	02	LF	LF
1	1	0	0	0	03	A	-
0	0	1	0	0	04	SP	SP
1	0	1	0	0	05	S	'
0	1	1	0	0	06	I	8
1	1	1	0	0	07	U	7
0	0	0	1	0	08	CR	CR
1	0	0	1	0	09	D	ENQ
0	1	0	1	0	0A	R	4
1	1	0	1	0	0B	J	BELL
0	0	1	1	0	0C	N	,
1	0	1	1	0	0D	F	!
0	1	1	1	0	0E	C	:
1	1	1	1	0	0F	K	(
0	0	0	0	1	10	T	5
1	0	0	0	1	11	Z	+
0	1	0	0	1	12	L	)
1	1	0	0	1	13	W	2
0	0	1	0	1	14	H	£
1	0	1	0	1	15	Y	6
0	1	1	0	1	16	P	0
1	1	1	0	1	17	Q	1
0	0	0	1	1	18	O	9
1	0	0	1	1	19	B	?
0	1	0	1	1	1A	G	&
1	1	0	1	1	1B	Figures	Figures
0	0	1	1	1	1C	M	.
1	0	1	1	1	1D	X	/
0	1	1	1	1	1E	V	=
1	1	1	1	1	1F	Letters	Letters

TABLE 10.1. The Baudot code

start a new line of text, whereas Unix systems achieve the same effect by just using a line feed control code. This causes problems when a text file is moved from one system to another.

10.3.2. Lorenz Operation: The Lorenz cipher encrypted data in Baudot code form by producing a sequence of five random bits which was exclusive-or'd with the bits representing the Baudot code. The actual Lorenz cipher made use of a sequence of wheels, each having a number of pins. The presence, or absence, of a pin signalled a one or a zero signal. As the wheel turned, the position of the pins changed relative to an input signal. In modern parlance each wheel corresponds to a shift register.

Consider a register of length 32 bits or, equivalently, a wheel with circumference 32. At each clock tick the register shifts left by one bit and the leftmost bit is output; equivalently the wheel turns around 1/32 of a revolution and the topmost pin is taken as the output. This is represented in Figure 10.3. In Chapter 12 we shall see shift registers, with more complex feedback functions, being used in modern stream ciphers. However, it is interesting to see how similar ideas were used such a long time ago.

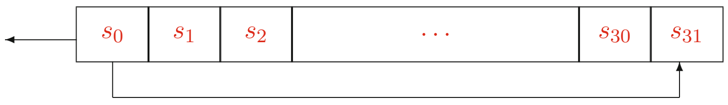

FIGURE 10.3. Shift Register of 32 bits

In Chapter 12 we shall see that the main problem is how to combine the more complex shift registers into a secure cipher. The same problem exists with the Lorenz cipher: namely, how the relatively simple operation of the wheels/shift registers can be combined to produce a cipher which is hard to break. From now on we shall refer to these as shift registers as opposed to wheels.

10.3.3. The Lorenz Cipher's Wheels: A Lorenz cipher uses twelve registers to produce the five streams of random bits. The twelve registers are divided into three subsets. The first set, called the chi-wheels, consists of five shift registers which we denote by $\chi_j^{(i)}$ comprising the output bit of the ith shift register on the jth clocking of the register, for $i = 1, 2, 3, 4, 5$. The five χ registers have lengths 41, 31, 29, 26 and 23, thus

$$\chi_{t+41}^{(1)} = \chi_t^{(1)},\ \chi_{t+31}^{(2)} = \chi_t^{(2)},\ \chi_{t+29}^{(3)} = \chi_t^{(3)},\ \chi_{t+26}^{(4)} = \chi_t^{(4)},\ \chi_{t+23}^{(5)} = \chi_t^{(5)}.$$

for all values of t. The second set of five shift registers, called the psi-wheels, we denote by $\psi_j^{(i)}$ for $i = 1, 2, 3, 4, 5$. These ψ registers have respective lengths 43, 47, 51, 53 and 59, i.e.

$$\psi_{t+43}^{(1)} = \psi_t^{(1)},\ \psi_{t+47}^{(2)} = \psi_t^{(2)},\ \psi_{t+51}^{(3)} = \psi_t^{(3)},\ \psi_{t+53}^{(4)} = \psi_t^{(4)},\ \psi_{t+59}^{(5)} = \psi_t^{(5)}.$$

The other two registers we shall denote by $\mu_j^{(i)}$ for $i = 1, 2$; these are called the motor registers. The lengths of the μ registers are 61 and 37 respectively, and so

$$\mu_{t+61}^{(1)} = \mu_t^{(1)},\ \mu_{t+37}^{(2)} = \mu_t^{(2)}.$$

10.3.4. Lorenz Cipher Operation: To describe how the Lorenz cipher clocks the various registers, we use the variable t to denote a global clock, which will be ticked for every Baudot code character which is encrypted. We also use a variable t_ψ to denote how often the ψ registers have been clocked, and a variable t_μ which denotes how often the second μ register, $\mu^{(2)}$, has been clocked. To start the cipher we set $t = t_\psi = t_\mu = 0$; at a given point we perform the following operations:

(1) Let κ denote the vector $(\chi_t^{(i)} \oplus \psi_{t_\psi}^{(i)})_{i=1}^5$.
(2) $t = t + 1$.

(3) If $\mu_t^{(1)} = 1$ then set $t_\mu = t_\mu + 1$.
(4) If $\mu_{t_\mu}^{(2)} = 1$ then set $t_\psi = t_\psi + 1$.
(5) Output κ.

The first line of the above produces the output keystream, the third line clocks the second μ register if the output of the first μ register is set (once it has been clocked), whilst the fourth line clocks all of the ψ registers if the output of the second μ register is set. From the above it should be deduced that the χ registers and the first μ register are clocked at every time interval. To encrypt a character the output vector κ is exclusive-or'd with the Baudot code representing the character of the plaintext. This is described graphically in Figure 10.4. Each clocking signal is depicted as a line with a circle on the end; each output wire is depicted by an arrow.

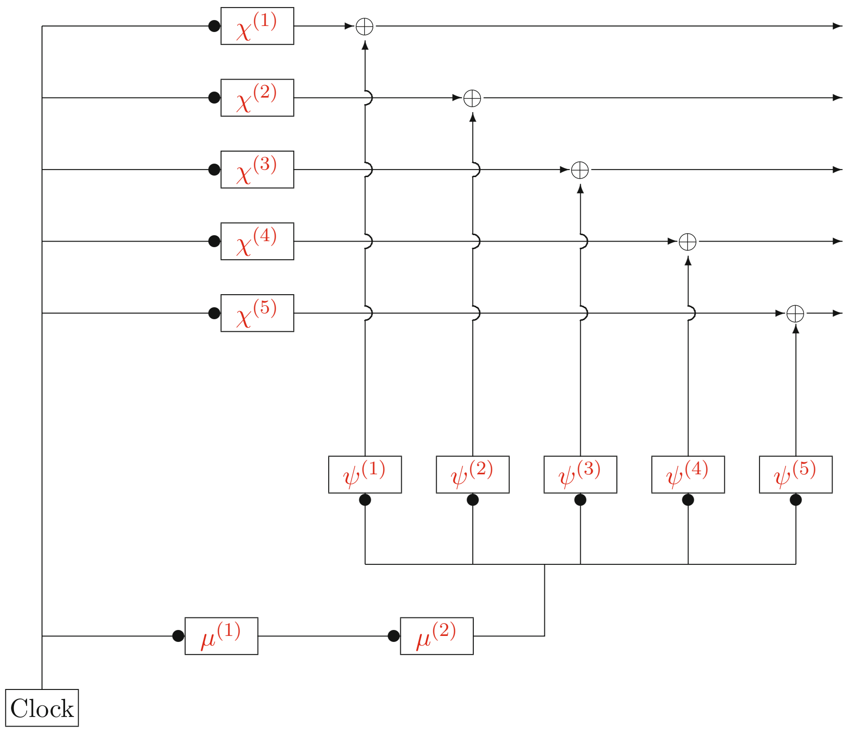

FIGURE 10.4. Graphical representation of the Lorenz cipher

The actual outputs of the ψ and μ motors at each time step are called the extended-ψ and the extended-μ streams. To ease future notation we will let ${\psi'}_t^{(i)}$ denote the output of the ψ registers at time t, whilst ${\mu'}_t^{(2)}$ will denote the output of the second μ register at time t. In other words, for a given tuple (t, t_ψ, t_μ) of valid clock values we have ${\psi'}_t^{(i)} = \psi_{t_\psi}^{(i)}$ and ${\mu'}_t^{(2)} = \mu_{t_\mu}^{(2)}$.

10.3.5. Example: To see this in operation consider the following example, where we describe the state of the cipher with the following notation:

```
Chi:    11111000101011000111100010111010001000111
        1100001101011101101011011001000
        1000100111100110001101111010
        11110001101000100011101001
```

```
          11011110000001010001110
Psi:      10110001101001010011010101010101101010100
          1101010101010101110110101010101100010101010110
          10100001001101010101010001011010111010101010010101001
          0101011010101000010101001101101010011011010110101011001
          0101010101010101011010100100110101001001010101001000100101 0
Motor:    0101111110110101100100011101111000100100111000111110101110100
          01110111000111111111000010101111111111
```

This gives the states of the χ, ψ and μ registers at time $t = t_\psi = t_\mu = 0$. The states will be shifted leftwise, and the output of each register will be the leftmost bit. So executing the above algorithm at time $t = 0$ the output first key vector will be

$$\kappa_0 = \chi_0 \oplus \psi_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Since $\mu_1^{(1)} = 1$ we clock the t_μ value, and since $\mu_1^{(2)} = 1$ we also clock the t_ψ value. Thus at time $t = 1$ the state of the Lorenz cipher becomes

```
Chi:      11110001010110001111000101110100010001111
          1000011010111011010110110010001
          00010011110011000111011110101
          11100011010001000111010011
          10111100000010100011101
Psi:      01100011010010100110101010101011010101010 01
          1010101010101011101101010101011000101010101 1101
          01000010011010101010100010110101110101010100101 0011
          1010110101010000101010011011010100110110101101011 0010
          10101010101010101101010010011010100100101010100100010010100
Motor:    1011111101101011001000111011110001001001110001111101011101000
          11101110001111111110000101011111111110
```

Now we look at what happens at the next clock tick. At time $t = 1$ we now output the vector

$$\kappa_1 = \chi_1 \oplus \psi_0 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

But now since $\mu_{t+1}^{(1)}$ is equal to zero we do not clock t_μ, whilst since $\mu_1^{(2)} = 1$ we still clock the t_ψ value. This process is then repeated, so that we obtain the following sequence for the first 60 output values of the keystream κ_t:

```
010010000101001011101100011011011101110001111111000000001001
000100011101110011111010111110011000011011000111111101110111
001010010110011011101110100001000100111100110010101101010000
101000101101110010011011001011000110100011110001111101010111
100011001000010001001000000101000000010100011100001001101001 1
```

This is produced by exclusive-or'ing the output of the χ registers, which is given by:

```
111110001010110001111000101110100010001111111100010101100011
110000110101110110101101100100011000011010111011010110110010
100010011110011000111011110101000100111100110001110111101010
111100011010001000111010011111000110100010001110100111110001
110111110000000101000111011011111000000010100011101101111000 0001
```

with the values of output of the ψ'_t stream at time t,

```
101100001111111010010100110101111111111111000001101010110101 0
110100101000000101010111011010000000000000011111001010110001 01
101000000100000000110101010101000000000000000000011011010111010
010100110111111101010000101010000000000000000111111101101010011 0
010100101000000101010101101010000000000000000000011001101010010
```

To ease understanding we also present the output $\mu'^{(2)}_t$ which is

```
11110111100000111111111111111000000000000100001011111111111
```

Recall that a one in this stream means that the ψ registers are clocked whilst a zero implies they are not clocked. One can see this effect in the ψ'_t output given earlier.

10.3.6. Lorenz Key Size: Just like the Enigma machine the Lorenz cipher has a long-term key set-up and a short-term per message set-up. The long term key is the state of each register. Thus it appears that there are a total of

$$2^{41+31+29+26+23+43+47+51+53+59+61+37} = 2^{501}$$

states, although the actual number is slightly less than this due to a small constraint which will be introduced in a moment. In the early stages of the war the μ registers were changed on a daily basis, the χ registers were changed on a monthly basis and the ψ registers were changed on a monthly or quarterly basis. Thus, if the month's settings had been broken then the “day” key “only” consisted of at most

$$2^{61+37} = 2^{98}$$

states. As the war progressed the Germans moved to changing all the internal states of the registers every day.

Given these “day” values for the register contents, the per message setting is given by the starting position of each register. Thus the total number of message keys, given a day key, is given by

$$41 \cdot 31 \cdot 29 \cdot 26 \cdot 23 \cdot 43 \cdot 47 \cdot 51 \cdot 53 \cdot 59 \cdot 61 \cdot 37 \approx 2^{64}.$$

The Lorenz cipher has an obvious weakness as defined, which is what eventually led to its breaking, and which the Germans were aware of. The basic technique which we will use throughout the rest of this chapter is to take the “Delta” of a sequence – this is defined as follows, for a sequence $s = (s_i)_{i=0}^{\infty}$:

$$\Delta s = (s_i \oplus s_{i+1})_{i=0}^{\infty}.$$

We shall denote the value of the Δs sequence at time t by $(\Delta s)_t$. The Δ operator is very important in the analysis of the Lorenz cipher because

$$\kappa_t^{(i)} = \chi_t^{(i)} \oplus \psi_{t_\psi}^{(i)}$$

and

$$\kappa_{t+1}^{(i)} = \chi_{t+1}^{(i)} \oplus \left(\mu'^{(2)}_t \cdot \psi_{t_\psi+1}\right) \oplus \left((\mu'^{(2)}_t - 1) \cdot \psi_{t_\psi}\right),$$

so that

$$\begin{aligned}(\Delta\kappa)_t &= (\chi_t \oplus \chi_{t+1}) \oplus \left(\mu'^{(2)}_t \cdot (\psi_{t_\psi} \oplus \psi_{t_\psi+1})\right) \\ &= (\Delta\chi)_t \oplus \left(\mu'^{(2)}_t \cdot (\Delta\psi)_{t_\psi}\right).\end{aligned}$$

Now if $\Pr[\mu'^{(2)}_t = 1] = \Pr[(\Delta_\psi)_{t_\psi} = 1] = 1/2$, as we would have by choosing the register states uniformly at random, then with probability 3/4 the value of the $\Delta\kappa$ stream reveals the value of the $\Delta\chi$ stream, which enables the adversary to recover the state of the χ registers relatively easily. Thus the Germans imposed a restriction on the key values so that

$$\Pr[\mu'^{(2)}_t = 1] \cdot \Pr[(\Delta_\psi)_{t_\psi} = 1] \approx 1/2.$$

In what follows we shall denote these two probabilities by $\delta = \Pr[\mu'^{(2)}_t = 1]$ and $\epsilon = \Pr[(\Delta_\psi)_{t_\psi} = 1]$.

Finally, to fix notation, if we let the Baudot encoding of the message be given by the sequence ϕ of 5-bit vectors, and the ciphertext be given by the sequence γ, then we have

$$\gamma_t = \phi_t \oplus \kappa_t.$$

As the war progressed more complex internal operations of the Lorenz cipher were introduced. These were called "limitations" by Bletchley, and they introduced extra complications into the clocking of the various registers. We shall ignore these extra complications however in our discussion.

Initially the Allies did not know anything about the Lorenz cipher, even that it consisted of twelve wheels, let alone their period. In August 1941 the Germans made a serious mistake: they transmitted almost identical messages, of roughly 4 000 characters in length, using exactly the same key. From this the cryptanalyst John Tiltman managed to reconstruct the key of roughly 4 000 characters that had been output by the Lorenz cipher. From this sequence of apparently random strings of five bits another cryptographer, Bill Tutte, recovered the precise internal workings of the Lorenz cipher. The final confirmation that the internal workings had been deduced correctly did not come until the end of the war, when the Allies captured a Lorenz machine on entering Germany.

10.4. Breaking the Lorenz Cipher's Wheels

Having determined the structure of the Lorenz cipher the problem remained of how to break it. The attack method used was broken into two stages. In the first stage the wheels needed to be broken: this was an involved process which only had to be performed once for each wheel configuration. Then a simpler procedure was produced which recovered the wheel positions for each message.

We now explain how wheel breaking occurred. The first task was to obtain with reasonable certainty the value of the sequence

$$\Delta\kappa^{(i)} \oplus \Delta\kappa^{(j)}$$

for different distinct values of i and j, usually $i = 1$ and $j = 2$. There were various different ways of performing this; below we present a gross simplification of the techniques used by the cryptanalysts at Bletchley. Our goal is simply to show that breaking even a 60 year old stream cipher requires some intricate manipulation of probability estimates, and that even small deviations from randomness in the output stream can cause a catastrophic failure in security.

To do this we first need to consider some characteristics of the plain text. Standard natural language contains a larger sequence of repeated characters than one would normally expect, compared to the case when a message is just random gibberish. If messages were random then one would expect

$$\Pr[(\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t = 0] = 1/2,$$

for any $i, j \in \{1, 2, 3, 4, 5\}$. However, if the plaintext sequence contains slightly more repeated characters than we expect, this probability would be slightly more than 1/2, so we set

$$\Pr[(\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t = 0] = 1/2 + \rho. \tag{11}$$

Due to the nature of German military parlance, and the Baudot encoding method, this was apparently particularly pronounced when one considered the first and second streams of bits, i.e. $i = 1$ and $j = 2$.

There are essentially two situations for wheel breaking. The first (more complex) case is when we do not know the underlying plaintext for a message, i.e. the attacker only has access to the ciphertext. The second case is when the attacker can guess with reasonable certainty the value of the underlying plaintext (a "crib" in the Bletchley jargon), and so can obtain the resulting keystream.

10.4.1. Ciphertext Only Method: The basic idea is that the sequence of ciphertext Deltas,

$$\Delta\gamma^{(i)} \oplus \Delta\gamma^{(j)}$$

will "reveal" the true value of the sequence

$$\Delta\chi^{(i)} \oplus \Delta\chi^{(j)}.$$

Consider the probability that we have

$$(\Delta\gamma^{(i)})_t \oplus (\Delta\gamma^{(j)})_t = (\Delta\chi^{(i)})_t \oplus (\Delta\chi^{(j)})_t. \tag{12}$$

Because of the relationship

$$\begin{aligned}(\Delta\gamma^{(i)})_t \oplus (\Delta\gamma^{(j)})_t &= (\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t \oplus (\Delta\kappa^{(i)})_t \oplus (\Delta\kappa^{(j)})_t \\ &= (\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t \oplus (\Delta\chi^{(i)})_t \oplus (\Delta\chi^{(j)})_t \\ &\quad \oplus \left(\mu'^{(2)}_t \cdot \left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right)\right),\end{aligned}$$

equation (12) can hold in one of two ways:

- Either we have
$$(\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t = 0$$
and
$$\mu'^{(2)}_t \cdot \left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right) = 0.$$
The first of these events occurs with probability $1/2+\rho$ by equation (11), whilst the second occurs with probability
$$(1-\delta) + \delta\cdot(\epsilon^2 + (1-\epsilon)^2) = 1 - 2\cdot\epsilon\cdot\delta + 2\cdot\epsilon^2\cdot\delta.$$
- Or we have
$$(\Delta\phi^{(i)})_t \oplus (\Delta\phi^{(j)})_t = 1$$
and
$$\mu'^{(2)}_t \cdot \left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right) = 1.$$
The first of these events occurs with probability $1/2-\rho$ by equation (11), whilst the second occurs with probability
$$2\cdot\delta\cdot\epsilon\cdot(1-\epsilon).$$

Combining these probabilities together we find that equation (12) holds with probability

$$\begin{aligned}(1/2+\rho)\cdot(1-2\cdot\epsilon\cdot\delta+2\cdot\epsilon^2\cdot\delta) &+ 2\cdot(1/2-\rho)\cdot\delta\cdot\epsilon\cdot(1-\epsilon) \\ &\approx (1/2+\rho)\cdot\epsilon + (1/2-\rho)\cdot(1-\epsilon) \\ &= 1/2+\rho\cdot(2\cdot\epsilon-1),\end{aligned}$$

since $\delta\cdot\epsilon \approx 1/2$ due to the key generation method mentioned earlier. So assuming we have a sequence of n ciphertext characters, if we are trying to determine

$$\sigma_t = (\Delta\chi^{(1)})_t \oplus (\Delta\chi^{(2)})_t,$$

i.e. we have set $i=1$ and $j=2$, then we know that this latter sequence has period $1271 = 41\cdot 31$. Thus each element in this sequence will occur $n/1271$ times. If n is large enough, then taking a majority verdict will determine the value of the sequence σ_t with some certainty.

10.4.2. Known Keystream Method: Now assume that we know the value of $\kappa_t^{(i)}$. We use a similar idea to above, but now we use the sequence of keystream Deltas,

$$\Delta\kappa^{(i)} \oplus \Delta\kappa^{(j)}$$

and hope that this reveals the true value of the sequence

$$\Delta\chi^{(i)} \oplus \Delta\chi^{(j)}.$$

This is likely to happen due to the identity

$$(\Delta\kappa^{(i)})_t \oplus (\Delta\kappa^{(j)})_t = (\Delta\chi^{(i)})_t \oplus (\Delta\chi^{(j)})_t \oplus \left(\mu'^{(2)}_t \cdot \left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right)\right).$$

Hence we will have

$$(\Delta\kappa^{(i)})_t \oplus (\Delta\kappa^{(j)})_t = (\Delta\chi^{(i)})_t \oplus (\Delta\chi^{(j)})_t \tag{13}$$

precisely when

$$\mu'^{(2)}_t \cdot \left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right) = 0.$$

This last equation will hold with probability

$$\begin{aligned}(1-\delta) + \delta \cdot (\epsilon^2 + (1-\epsilon)^2) &= 1 - 2 \cdot \epsilon \cdot \delta + 2 \cdot \epsilon^2 \cdot \delta \\ &\approx 1 - 1 + \epsilon = \epsilon,\end{aligned}$$

since $\delta \cdot \epsilon \approx 1/2$. But since $\delta \cdot \epsilon \approx 1/2$ we usually have $0.6 \le \epsilon \le 0.8$, thus equation (13) holds with a reasonable probability. So as before we try to take a majority verdict to obtain an estimate for each of the 1271 terms of the σ_t sequence.

10.4.3. Both Methods Continued: Whichever of the above methods we use there will still be some errors in our guess for the stream σ_t, which we will now try to correct. In some sense we are not really after the values for the sequence σ_t, what we really want is the exact values of the two shorter sequences

$$(\Delta\chi^{(1)})_t \text{ and } (\Delta\chi^{(2)})_t,$$

since these will allow us to deduce possible values for the first two χ registers in the Lorenz cipher. The approximation for the σ_t sequence of 1271 bits we now write down in a 41×31-bit array, with the first 31 bits in row one, the second 31 bits in row two, and so on. A blank is placed in the array if we cannot determine the value of this bit with any reasonable certainty. For example, assuming the above configuration was used to encrypt the ciphertext, we could obtain an array which looks something like this:

```
0-0---01--1--110--110-10--1-0-1
010---0---10011-1----1-010-1--1
--00-1--111001--1-1--1101--1001
0-00--01----0-------0-10-0--001
-0-110-000-110-1000------1---10
-1--0---1-100110111-01---01---1
01-0-10111-001101-------1-1-0--
-01--0-0--0-100-00001-0---0----
1--1--10000-------0------1--11-
101---1-00-110--0-00--0-0---100
1---11--000---0---0-1--1------1
---1-0----011--1----1---01-01-0
-1---1--1--00110-1-1-110-0-100-
1-------10--10-1---0100-010--1-
0--0--011--0011--11-011------0-
--0001-1-11--11011---1-010110--
```

```
----10---------1000--001-10----
0-00-1-11110----1111-1-01-11-0-
----01-1-----11--111011-1--10--
0-0-01--1---011---11--1-1--1001
10-1---0-1-1-0010--01-0--10--10
----------1-01-0-1--0-10--1-001
010-01-1-110011-11---1-0---1--1
1-1-1-1000-11-010--01-0101-01-0
1---10-00--110---0-0--011-00-10
10-1-010-0----01----1--10-0----
-1--0-011-1001-0----01101011--1
010-010-11-0--101--10--0--1--0-
-01---1-0---1-01000-1---0-001-0
--1-10--0--110-100001-0--10-110
----1--00001-00--00010010----10
011-0101-110-1-011-101101----01
01---1----100--01---01-01--0-01
-011--1-000-1--10-00-0---1001-0
1011-01--0---001---01-01--0----
---0--0-1-1--11----1----1-11---
----0---1--0---0-----11--0--00-
10--101--0----0-0-0--0--010----
-100---1-110-----11-01-0---1---
0100----1-1----01-1--1111--11--
0-0001-1-1-0011011-1---01011-0-
```

Now the goal of the attacker is to fill in this array, a process known at Bletchley as "rectangling", bearing in mind that some of the zeros and ones entered could themselves be incorrect. The point to note is that, when completed, there should be just two distinct rows in the table, each the complement of the other, and similarly for the columns. A reasonable method to follow is to take all rows starting with zero and then count the number of zeros and ones in the second element of those rows. In our example above, we find there are seven ones and no zeros. Doing the same for rows starting with a one we find there are four zeros and no ones. Thus we can deduce that the two types of rows in the table should start with a 10 and a 01. We then fill in the second element in any row which has its first element set. We continue in this way, first looking at rows and then looking at columns, until the whole table is filled in.

The above table was found using a few thousand characters of known keystream, which allows (via the above method) the simple reconstruction of the full table. According to the Bletchley documents, the cryptographers at Bletchley would actually use a few hundred characters of keystream in a known keystream attack, and a few thousand in an unknown keystream attack. Since we are following rather naive methods our results are not as spectacular.

Once completed we can take the first column as the value of the $(\Delta\chi^{(1)})_t$ sequence and the first row as the value of the $(\Delta\chi^{(2)})_t$ sequence. We can then repeat this analysis for different pairs of the χ registers until we determine that we have

$$\begin{aligned}
\Delta\chi^{(1)} &= \texttt{00001001111101001000100111001110011001000},\\
\Delta\chi^{(2)} &= \texttt{0100010111100110111101101011001},\\
\Delta\chi^{(3)} &= \texttt{10011010001010100100110001111},\\
\Delta\chi^{(4)} &= \texttt{00010010111001100100111010},\\
\Delta\chi^{(5)} &= \texttt{01100010000011110010011}.
\end{aligned}$$

From these $\Delta\chi$ sequences we can then determine possible values for the internal state of the χ registers.

10.4.4. Breaking the Other Wheels: So having “broken” the χ wheels of the Lorenz cipher, the task remains to determine the internal state of the other registers. In the ciphertext only attack one now needs to recover the actual keystream, a step which is clearly not needed in the known-keystream scenario. The trick here is to use the statistics of the underlying language again to try to recover the actual $\kappa^{(i)}$ sequence. We first de-χ the ciphertext sequence γ, using the values of the χ registers which we have just determined, to obtain

$$\begin{aligned}\beta_t^{(i)} &= \gamma_t^{(i)} \oplus \chi_t^{(i)}\\ &= \phi_t^{(i)} \oplus {\psi'}_t^{(i)}.\end{aligned}$$

We then take the Delta of this β sequence

$$(\Delta\beta)_t = (\Delta\phi)_t \oplus \left({\mu'}_t^{(2)} \cdot (\Delta\psi)_{t_\psi}\right),$$

and by our previous argument we will see that many values of the $\Delta\phi$ sequence will be “exposed” in the $\Delta\beta$ sequence. Using a priori knowledge of the $\Delta\phi$ sequence, for example that it uses Baudot codes and that natural language has many sequences of bigrams (e.g. a space always follows a full stop), one can eventually recover the sequence ϕ and hence κ. At Bletchley this last step was usually performed by hand.

So in both scenarios we have now determined both the χ and the κ sequences. But what we are really after is the initial values of the registers ψ and μ. To determine these we de-χ the resulting κ sequence to obtain the ${\psi'}_t$ sequence. In our example this would reveal the sequence

```
101100001111111010010100110101111111111110000011010101101010
110100101000000101010111011010000000000001111100101011000101
101000001000000011010101010100000000000000000011011010111010
010100110111111010100001010100000000000001111111011010100110
010100101000000101010101101010000000000000000011001101010010
```

given earlier. From this we can then recover a guess as to the ${\mu'}_t^{(2)}$ sequence.

```
11110111110000011111111111111100000000000100001011111111111 1...
```

Note that it is only a guess; it might occur that $\psi_{t_\psi} = \psi_{t_\psi+1}$, but we shall ignore this possibility. Once we have determined enough of the ${\mu'}_t^{(2)}$ sequence so that we have 59 ones in it, then we will have determined the initial state of the ψ registers. This is because after 59 clock ticks of the ψ registers all outputs have been presented in the ψ' sequence, since the largest ψ register has size 59.

All that remains is to determine the state of the μ registers. To do this we notice that the ${\mu'}_t^{(2)}$ sequence will make a transition from a 0 to a 1, or a 1 to a 0, precisely when $\mu_t^{(1)}$ outputs a one. By constructing enough of the ${\mu'}_t^{(2)}$ stream as above (say a few hundred bits) this allows us to determine the value of the $\mu^{(1)}$ register almost exactly. Having recovered $\mu_t^{(1)}$ we can then deduce the values which must be contained in $\mu_{t_\mu}^{(2)}$ from this sequence and the resulting value of ${\mu'}_t^{(2)}$.

According to various documents, in the early stages of the Lorenz cipher-breaking effort at Bletchley, the entire “Wheel Breaking” operation was performed by hand. However, as time progressed the part which involved determining the $\Delta\chi$ sequences above from the rectangling procedure was eventually performed by the Colossus computer.

10.5. Breaking a Lorenz Cipher Message

The Colossus was the world’s first programmable digital electronic computer, and as such was the precursor of all modern computers. The role of the Colossus was vital to the Allied war effort, and was so secret that its very existence was not divulged until the 1980s. The Colossus computer was originally created not to break the wheels, i.e. to determine the long-term key of the Lorenz cipher, but to determine the per message settings, and hence to help break the individual ciphertexts. Whilst the previous method for breaking the wheels could be used to attack any ciphertext, for it to work efficiently requires a large ciphertext and a lot of luck. However, once the wheels are broken, i.e. we know the bits in the various registers, breaking the next ciphertext becomes easier.

To break a message we again use the trick of de-χ'ing the ciphertext sequence γ, and then applying the Delta method to the resulting sequence β. We assume we know the internal states of all the registers but not their starting positions. We shall let s_i denote the unknown values of the starting positions of the five χ wheels and s_ϕ (resp. s_μ) the global unknown starting position of the set of ϕ (resp. μ) wheels.

$$\begin{aligned}\beta_t &= \gamma_t \oplus \chi_{t+s_p} \\ &= \phi_t \oplus {\psi'}_{t+s_\phi},\end{aligned}$$

and then

$$(\Delta\beta)_t = (\Delta\phi)_{t+s_\phi} \oplus \left({\mu'}^{(2)}_{t+s_\mu} \cdot (\Delta\psi)_{t_\psi}\right).$$

We then take two of the resulting five bit streams and exclusive-or them together as before to obtain

$$\begin{aligned}(\alpha^{(i,j)})_t &= (\Delta\beta^{(i)})_t \oplus (\Delta\beta^{(j)})_t \\ &= (\Delta\phi^{(i)})_{t+s_\phi} \oplus (\Delta\phi^{(j)})_{t+s_\phi} \oplus {\mu'}^{(2)}_{t+t_\mu}\left((\Delta\psi^{(i)})_{t_\psi} \oplus (\Delta\psi^{(j)})_{t_\psi}\right).\end{aligned}$$

Using our prior probability estimates we can determine the following probability estimate

$$\Pr[(\alpha^{(i,j)})_t = 0] \approx 1/2 + \rho \cdot (2 \cdot \epsilon - 1),$$

which is exactly the same probability we had for equation (11) to hold true. In particular we note that $\Pr[\alpha^{(i,j)}{}_t = 0] > 1/2$, which forms the basis of this method of breaking into Lorenz ciphertexts.

Let us fix $i = 1$ and $j = 2$. On assuming we know the values for the registers, all we need do is determine their starting positions s_1, s_2. We simply need to go through all $1271 = 41 \cdot 31$ possible starting positions for the first and second χ registers. For each one of these starting positions we compute the associated $(\alpha^{(1,2)})_t$ sequence and count the number of values which are zero. Since we have $\Pr[\alpha_t^{(i,j)} = 0] > 1/2$ the correct value for the starting positions will correspond to a particularly high value for the count of the number of zeros.

This is a simple statistical test which allows one to determine the start positions of the first and second χ registers. Repeating this for other pairs of registers, or using similar statistical techniques, we can recover the start position of all χ registers. These statistical techniques are what the Colossus computer was designed to perform.

Once the χ register positions have been determined, the determination of the start positions of the ψ and μ registers can then be performed by hand. The techniques for this are very similar to the earlier techniques needed to break the wheels, however once again various simplifications occur since one is assumed to know the state of each register, but not its start position.

Chapter Summary

- We have described the general model for symmetric ciphers, and for stream ciphers in particular.

- We have looked at the Lorenz cipher as a stream cipher, and described its inner workings in terms of shift registers.
- We sketched how the Lorenz cipher was eventually broken, in particular how very tiny deviations from true randomness in the output were exploited by the Blecthley cryptographers.

Further Reading

The paper by Carter provides a more detailed description of the cryptanalysis performed at Bletchley on the Lorenz cipher. The book by Gannon is a very readable account of the entire operation related to the Lorenz cipher, from obtaining the signals through to the construction and operation of the Colossus computer. For the "real" details you should consult the General Report on Tunny.

F.L. Carter. *The Breaking of the Lorenz Cipher: An Introduction to the Theory Behind the Operational Role of "Colossus" at BP.* In Cryptography and Coding – 1997, LNCS 1355, 74–88, Springer, 1997.

P. Gannon *Colossus: Bletchley Park's Greatest Secret.* Atlantic Books, 2007.

J. Good, D. Michie and G. Timms. *General report on Tunny, With Emphasis on Statistical Methods.* Document reference HW 25/4 and HW 25/5, Public Record Office, Kew. Originally written in 1945, declassified in 2000.