CHAPTER 12

Modern Stream Ciphers

Chapter Goals

- To understand the basic principles of modern symmetric ciphers.
- To explain the workings of a modern stream cipher.
- To investigate the properties of linear feedback shift registers (LFSRs).
- To explain how to introduce non-linearity into a stream cipher.

12.1. Stream Ciphers from Pseudo-random Functions

We can interpret a pseudo-random function as a stream cipher. Let $\{F_k\}_K$ be a PRF family with codomain C of bitstrings of length ℓ. The PRF family immediately defines a stream cipher for messages of length ℓ bits. We encrypt a message by setting

$$c = m \oplus F_k(0).$$

We then want to show that this scheme is IND-PASS if the underlying PRF is secure. This result is given in the next theorem.

Theorem 12.1. *If $\{F_k\}_K$ is a PRF family outputting strings of length ℓ bits then the stream cipher Π given by $c = m \oplus F_k(0)$ for $m \in \mathbb{P} = \{0,1\}^\ell$ is* IND-PASS. *In particular*

$$\mathrm{Adv}^{\mathsf{IND\text{-}PASS}}_{\Pi}(A) \leq 2 \cdot \mathrm{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A)$$

Proof. We take the adversary A and consider the game it is playing, as depicted in Figure 12.1. We then change the game slightly, by performing a so-called "game hop". This hop consists of replacing the real PRF function by a completely random function; see Figure 12.2. We call the first game G_0 and the second game G_1. We stress that the adversary (i.e. the algorithm that the adversary runs) in both games is the same; we are just changing the rules of the game.

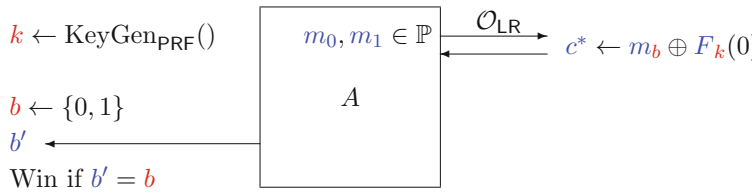

Figure 12.1. Security game G_0 for the scheme $c \leftarrow m \oplus F_k(0)$

Now let b'_0 denote the bit returned by the adversary in game G_0 and b'_1 denote the bit returned by the adversary in game G_1. Similarly let b_0 and b_1 denote the bits chosen by the challenger in games

N.P. Smart, *Cryptography Made Simple*, Information Security and Cryptography, DOI 10.1007/978-3-319-21936-3_12

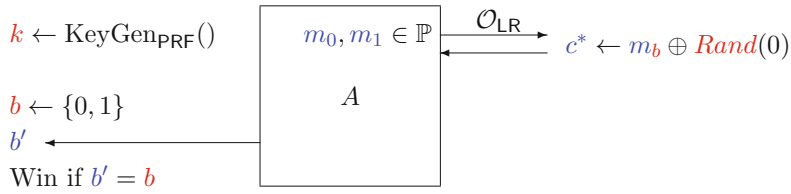

FIGURE 12.2. "Hopped" security game G_1 for the scheme $c \leftarrow m \oplus Rand(0)$

G_0 and G_1 respectively. We have the following relationships between the various probabilities:

$$\left|\Pr[b'_0 = 1 | b_0 = 1] - \Pr[b'_1 = 1 | b_1 = 1]\right| = \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A), \tag{14}$$

and

$$\left|\Pr[b'_0 = 0 | b_0 = 0] - \Pr[b'_1 = 0 | b_1 = 0]\right| = \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A), \tag{15}$$

i.e. for fixed b in both games the difference in the winning probabilities between the two games is the same as the advantage in distinguishing a member of a PRF family from a random function. Also note that

$$\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_1 = 0 | b_1 = 0] = 0 \tag{16}$$

since if we have a random function then an "encryption" of m_0 is a random string, as is an "encryption" of m_1; this is essentially the security of the one-time pad. Thus the probability of the adversary winning in game G_1 is equal to $1/2$. Putting this together we have

$$\begin{aligned}
\text{Adv}^{\mathsf{IND\text{-}PASS}}_{\Pi}(A) &= \left|\Pr[b'_0 = 1 | b_0 = 1] - \Pr[b'_0 = 1 | b_0 = 0]\right| && \text{by definition} \\
&= \Big|\Pr[b'_0 = 1 | b_0 = 1] \\
&\qquad - \big(\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_1 = 1 | b_1 = 1]\big) \\
&\qquad - \Pr[b'_0 = 1 | b_0 = 0]\Big| && \text{adding zero} \\
&\leq \left|\Pr[b'_0 = 1 | b_0 = 1] - \Pr[b'_1 = 1 | b_1 = 1]\right| \\
&\qquad + \left|\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_0 = 1 | b_0 = 0]\right| && \text{triangle inequality} \\
&\leq \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A) + \left|\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_0 = 1 | b_0 = 0]\right| && \text{by equation (14)} \\
&= \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A) + \Big|\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_0 = 1 | b_0 = 0] \\
&\qquad - \big(\Pr[b'_1 = 0 | b_1 = 0] - \Pr[b'_1 = 0 | b_1 = 0]\big)\Big| && \text{adding zero again} \\
&\leq \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A) + \left|\Pr[b'_1 = 1 | b_1 = 1] - \Pr[b'_1 = 0 | b_1 = 0]\right| \\
&\qquad + \left|\Pr[b'_1 = 0 | b_1 = 0] - \Pr[b'_0 = 0 | b_0 = 0]\right| && \text{triangle inequality} \\
&= \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A) + \left|\Pr[b'_1 = 0 | b_1 = 0] - \Pr[b'_0 = 0 | b_0 = 0]\right| && \text{by equation (16)} \\
&\leq 2 \cdot \text{Adv}^{\mathsf{PRF}}_{\{F_k\}_K}(A) && \text{by equation (15).}
\end{aligned}$$

□

So when defining a stream cipher we want to look for a candidate which could possibly be a pseudo-random function. In this chapter we will look at various practical constructions of functions which output what looks like random data.

12.2. Linear Feedback Shift Registers

A standard way of producing a binary stream of data is to use a feedback shift register. These are small circuits containing a number of memory cells, each of which holds one bit of information. The set of such cells forms a register. In each cycle a certain predefined set of cells are "tapped" and their value is passed through a function, called the *feedback function*. The register is then shifted down by one bit, with the output bit of the feedback shift register being the bit that is shifted out of the register. The combination of the tapped bits is then fed into the empty cell at the top of the register. Compare this to how we modelled the Lorenz cipher wheels in Chapter 10: the difference is that the output bit is replaced by a new bit which depends on other bits within the register. This is explained in Figure 12.3.

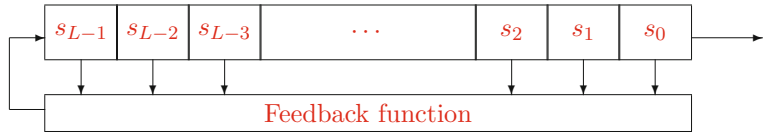

FIGURE 12.3. Feedback shift register

It is desirable, for reasons we shall see later, to use some form of non-linear function as the feedback function. However, this is often hard to do in practice, hence usually one uses a linear feedback shift register, or LFSR for short, where the feedback function is a linear function of the tapped bits. In each cycle a certain predefined set of cells are "tapped" and their value is exclusive-or'ed together. The register is then shifted down by one bit, with the output bit of the LFSR being the bit that is shifted out of the register. Again, the combination of the tapped bits is then fed into the empty cell at the top of the register.

Mathematically this can be defined as follows, where the register is assumed to be of length L. One defines a set of bits $[c_1, \ldots, c_L]$ which are set to one if that cell is tapped and set to zero otherwise. The initial internal state of the register is given by the bit sequence $[s_{L-1}, \ldots, s_1, s_0]$. The output sequence is then defined to be $s_0, s_1, s_2, \ldots, s_{L-1}, s_L, s_{L+1}, \ldots$ where for $j \geq L$ we have

$$s_j = c_1 \cdot s_{j-1} \oplus c_2 \cdot s_{j-2} \oplus \cdots \oplus c_L \cdot s_{j-L}.$$

Note that for an initial state of all zeros the output sequence will be the zero sequence, but for a non-zero initial state the output sequence must eventually be periodic (since we must eventually return to a state we have already been in). The period of a sequence is defined to be the smallest integer N such that

$$s_{N+i} = s_i$$

for all sufficiently large i. In fact there are $2^L - 1$ possible non-zero states and so the most one can hope for is that an LFSR, for all non-zero initial states, produces an output stream whose period is exactly $2^L - 1$.

Each state of the linear feedback shift register can be obtained from the previous state via a matrix multiplication. If we write

$$M = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ c_L & c_{L-1} & c_{L-2} & \dots & c_1 \end{pmatrix}$$

and

$$v = (1, 0, 0, \dots, 0)$$

and we write the internal state as

$$s = (s_1, s_2, \dots, s_L)$$

then the next state can be deduced by computing

$$s \leftarrow M \cdot s$$

and the output bit can be produced by computing the vector product

$$v \cdot s.$$

The properties of the output sequence are closely tied up with the properties of the binary polynomial

$$C(X) = 1 + c_1 \cdot X + c_2 \cdot X^2 + \dots + c_L \cdot X^L \in \mathbb{F}_2[X],$$

called the connection polynomial for the LFSR. The connection polynomial and the matrix are related via

$$C(X) = \det(X \cdot M - I_L).$$

In some textbooks the connection polynomial is written in reverse, i.e. they use

$$G(X) = X^L \cdot C(1/X)$$

as the connection polynomial. One should note that in this case $G(X)$ is the characteristic polynomial of the matrix M.

As examples see Figure 12.4 for an LFSR in which the connection polynomial is given by X^3+X+1 and Figure 12.5 for an LFSR in which the connection polynomial is given by $X^{32}+X^3+1$.

FIGURE 12.4. Linear feedback shift register: $X^3 + X + 1$

FIGURE 12.5. Linear feedback shift register: $X^{32} + X^3 + 1$

Of particular importance is the case in which the connection polynomial is primitive.

Definition 12.2. *A binary polynomial $C(X)$ of degree n is primitive if it is irreducible and a root θ of $C(X)$ generates the multiplicative group of the field $\mathbb{F}_{2^n}$. In other words, since $C(X)$ is irreducible we already have*

$$\mathbb{F}_2[X]/(C(X)) = \mathbb{F}_2(\theta) = \mathbb{F}_{2^n},$$

but we also require $\mathbb{F}_{2^n}^* = \langle\theta\rangle$.

The properties of the output sequence of the LFSR can then be deduced from the following cases.

- $c_L = 0$: i.e. the register is longer than the degree of the connection polynomial.
  In this case the sequence is said to be singular. The output sequence may not be periodic, but it will be eventually periodic.
- $c_L = 1$:
  Such a sequence is called non-singular. The output is always purely periodic, in that it satisfies $s_{N+i} = s_i$ for all i rather than for all sufficiently large values of i. Of the non-singular sequences of particular interest are those satisfying
  - $C(X)$ is irreducible:
    Every non-zero initial state will produce a sequence with period equal to the smallest value of N such that $C(X)$ divides $1 + X^N$. We have that N will divide $2^L - 1$.
  - $C(X)$ is primitive:
    Every non-zero initial state produces an output sequence which is periodic and of exact period $2^L - 1$.

We do not prove these results here, but proofs can be found in any good textbook on the application of finite fields to coding theory, cryptography or communications science. However, we present four examples which show the different behaviours. All examples are on 4-bit registers, i.e. $L = 4$.

Example 1: In this example we use an LFSR with connection polynomial $C(X) = X^3 + X + 1$. We therefore see that $\deg(C) \neq L$, and so the sequence will be singular. The matrix M generating the sequence is given by

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

If we label the states of the LFSR by the number whose binary representation is the state value, i.e. $s_0 = (0, 0, 0, 0)$ and $s_5 = (0, 1, 0, 1)$, then the periods of this LFSR can be represented by the transitions in Figure 12.6. Note that it is not purely periodic.

Example 2: Now let the connection polynomial $C(X) = X^4 + X^3 + X^2 + 1 = (X + 1)(X^3 + X + 1)$, which corresponds to the matrix

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

The state transitions are then given by Figure 12.7. Note that it is purely periodic, but with two different cycle lengths due to the different factorization properties of the connection polynomial modulo 2: Two cycles of length $7 = 2^3 - 1$ corresponding to the factor of degree three, and one of length $1 = 2^1 - 1$ corresponding to the factor of degree one. We ignore the trivial cycle of the zero'th state.

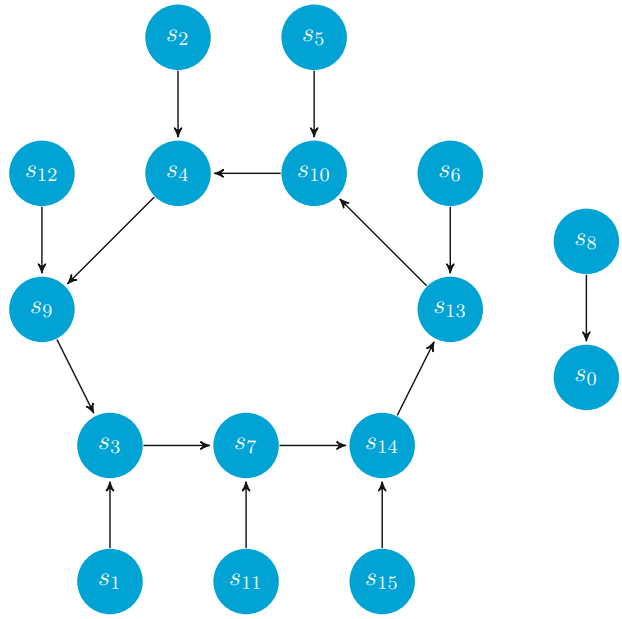

FIGURE 12.6. Transitions of the 4-bit LFSR with connection polynomial $X^3 + X + 1$

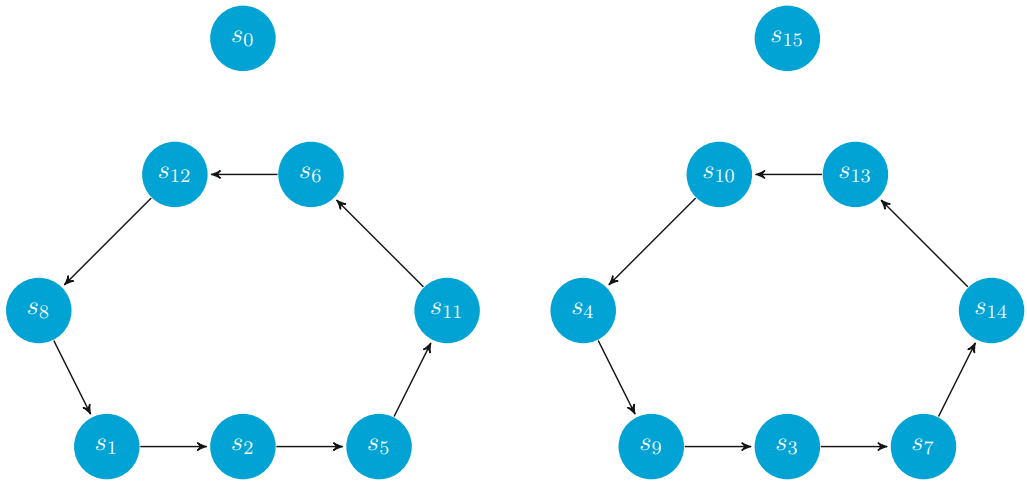

FIGURE 12.7. Transitions of the 4-bit LFSR with connection polynomial $X^4+X^3+X^2+1$

Example 3: Now take the connection polynomial $C(X) = X^4 + X^3 + X^2 + X + 1$, which is irreducible, but not primitive. The matrix is now given by

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

The state transitions are then given by Figure 12.8. Note that it is purely periodic and all cycles have the same length, bar the trivial one.

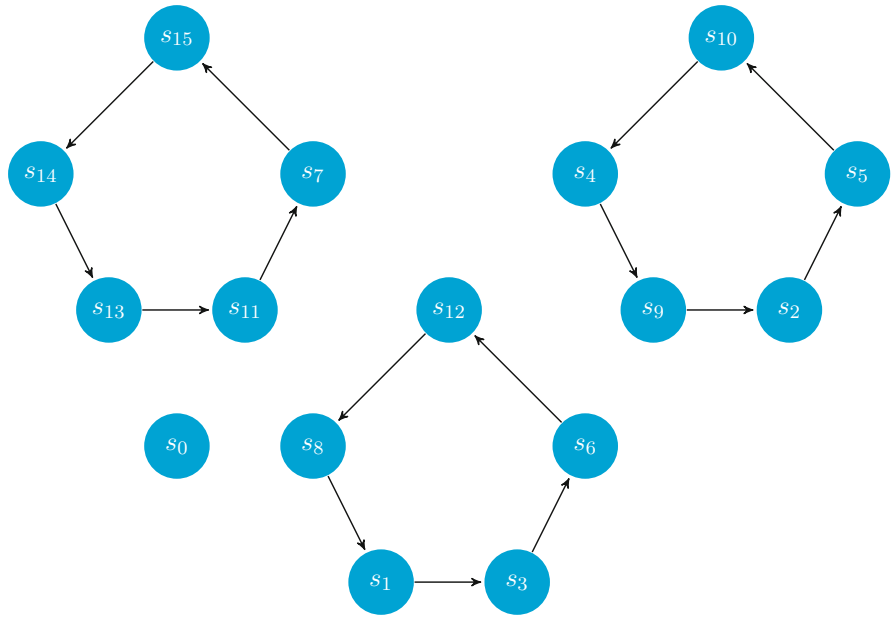

FIGURE 12.8. Transitions of the 4-bit LFSR with connection polynomial $X^4+X^3+X^2+X+1$

Example 4: As our final example we take the connection polynomial $C(X) = X^4 + X + 1$, which is irreducible and primitive. The matrix M is now

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

and the state transitions are given by Figure 12.9.

Whilst there are algorithms to generate primitive polynomials for use in applications we shall not describe them here. The following list gives some examples, all with a small number of taps for efficiency.

$$\begin{array}{ccc} x^{31}+x^3+1, & x^{31}+x^6+1, & x^{31}+x^7+1, \\ x^{39}+x^4+1, & x^{60}+x+1, & x^{63}+x+1, \\ x^{71}+x^6+1, & x^{93}+x^2+1, & x^{137}+x^{21}+1, \\ x^{145}+x^{52}+1, & x^{161}+x^{18}+1, & x^{521}+x^{32}+1. \end{array}$$

Although LFSRs efficiently produce bit streams from a small key, especially when implemented in hardware, they are not usable on their own for cryptographic purposes. This is because they are essentially linear, which is after all why they are efficient.

We shall now show that if we know an LFSR to have L internal registers and we can determine $2 \cdot L$ consecutive bits of the stream then we can determine the whole stream. First notice that

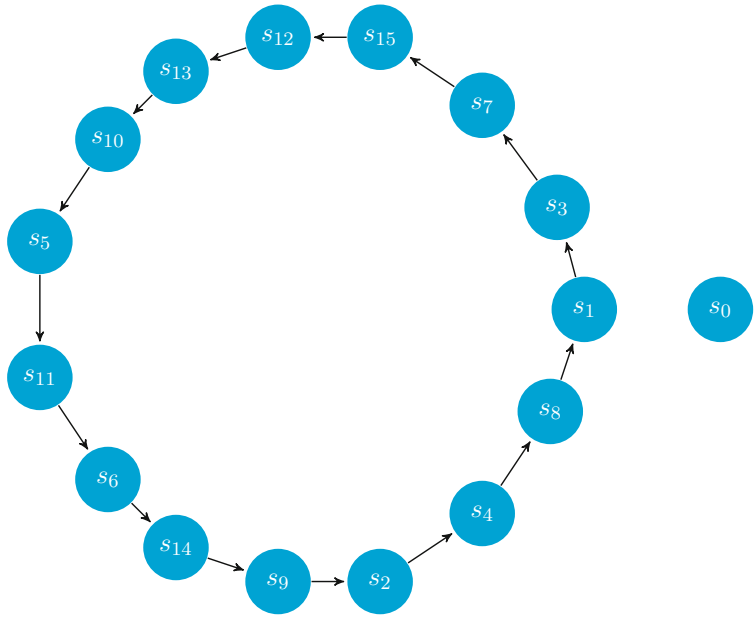

FIGURE 12.9. Transitions of the 4-bit LFSR with connection polynomial $X^4 + X + 1$

we need to determine L unknowns: the L values of the "taps" c_i, since the L values of the initial state $s_0, \ldots, s_{L-1}$ are given to us. This type of data could be available in a known plaintext attack, where we obtain the ciphertext corresponding to a known piece of plaintext; since the encryption operation is simply exclusive-or we can determine as many bits of the keystream as we require. Using the equation

$$s_j = \sum_{i=1}^{L} c_i \cdot s_{j-i} \pmod 2,$$

we obtain $2 \cdot L$ linear equations, which we then solve via standard matrix techniques. We write our matrix equation as

$$\begin{pmatrix} s_{L-1} & s_{L-2} & \cdots & s_1 & s_0 \\ s_L & s_{L-1} & \cdots & s_2 & s_1 \\ \vdots & \vdots & & \vdots & \vdots \\ s_{2L-3} & s_{2L-4} & \cdots & s_{L-1} & s_{L-2} \\ s_{2L-2} & s_{2L-3} & \cdots & s_L & s_{L-1} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{L-1} \\ c_L \end{pmatrix} = \begin{pmatrix} s_L \\ s_{L+1} \\ \vdots \\ s_{2L-2} \\ s_{2L-1} \end{pmatrix}.$$

As an example, suppose we see the output sequence

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

and we are told that this sequence was the output of a four-bit LFSR. Using the above matrix equation, and solving it modulo 2, we would find that the connection polynomial was given by

$$X^4 + X + 1.$$

Hence, if we use an LFSR of size L to generate a keystream for a stream cipher and the adversary obtains at least $2 \cdot L$ bits of this keystream then she can determine the exact LFSR used and so generate as much of the keystream as she wishes. Therefore, we would like to be able to adapt

the use of LFSRs in some non-linear way, which hides their linearity in order to produce output sequences with high linear complexity. We can conclude that a stream cipher based solely on a single LFSR is insecure against a known plaintext attack.

12.2.1. Linear Complexity: An important measure of the cryptographic quality of a sequence is given by the linear complexity of the sequence.

Definition 12.3 (Linear complexity)**.** *For an infinite binary sequence*

$$s = s_0, s_1, s_2, s_3, \ldots,$$

we define the linear complexity of s as $L(s)$ where

- $L(s) = 0$ *if s is the zero sequence,*
- $L(s) = \infty$ *if no LFSR generates s,*
- $L(s)$ *is the length of the shortest LFSR to generate s, otherwise.*

Since we cannot compute the linear complexity of an infinite set of bits we often restrict ourselves to a finite set s^n of the first n bits. The linear complexity satisfies the following properties for any sequence s.

- For all $n \geq 1$ we have $0 \leq L(s^n) \leq n$.
- If s is periodic with period N then $L(s) \leq N$.
- $L(s \oplus t) \leq L(s) + L(t)$.

For a random sequence of bits, which is what we want from a stream cipher's keystream generator, we should have that the expected linear complexity of s^n is approximately just larger than $n/2$. But for a keystream generated by an LFSR we know that we will have $L(s^n) = L$ for all $n \geq L$. Hence, an LFSR produces nothing at all like a random bit string. After all it is produced by a linear function!

We have seen that if we know the length of the LFSR then, from the output bits, we can generate the connection polynomial. To determine the length we use the linear complexity profile, which is defined to be the sequence $L(s^1), L(s^2), L(s^3), \ldots$. There is also an efficient algorithm called the Berlekamp–Massey algorithm which given a finite sequence s^n will compute the linear complexity profile

$$L(s^1), L(s^2), L(s^3), \ldots, L(s^n).$$

In addition the Berlekamp–Massey algorithm will also output the associated connection polynomial, if $n \geq L(s^n)/2$, using a technique more efficient than the prior matrix technique.

12.3. Combining LFSRs

To obtain greater security a common practice is to use a number, say n, of LFSRs, each producing a different output sequence $x_1^{(i)}, \ldots, x_n^{(i)}$. The key is then the initial state of all of the LFSRs and the keystream is produced from these n generators using a non-linear combination function $f(x_1, \ldots, x_n)$, as described in Figure 12.10.

We begin by examining the case where the combination function is a Boolean function of the output bits of the constituent LFSRs. For analysis of this function we write it as a sum of distinct products of variables, e.g.

$$f(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus (x_4 \cdot x_5) \oplus (x_1 \cdot x_2 \cdot x_3 \cdot x_5).$$

However, in practice the Boolean function could be implemented in a different way. When expressed as a sum of products of variables we say that the Boolean function is in algebraic normal form.

Suppose that one uses n LFSRs of maximal length (i.e. all with a primitive connection polynomial) and whose periods $L_1, \ldots, L_n$ are all distinct and greater than two. Then, an amazing fact is that the linear complexity of the keystream generated by $f(x_1, \ldots, x_n)$ is equal to

$$f(L_1, \ldots, L_n)$$

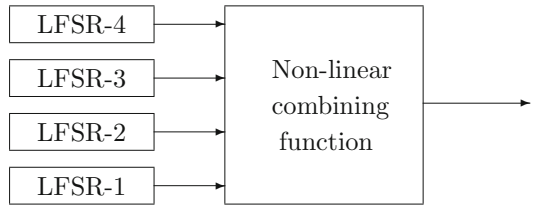

FIGURE 12.10. Combining LFSRs

where we replace $\oplus$ in f with integer addition and multiplication modulo two by integer multiplication, assuming f is expressed in algebraic normal form. The non-linear order of the polynomial f is then defined to be equal to the total degree of f[1].

However, it turns out that creating a non-linear function which results in a high linear complexity is not the whole story. For example, consider the stream cipher produced by the Geffe generator. This generator takes three LFSRs of maximal period and distinct sizes, L_1, L_2 and L_3, and then combines them using the following second-order non-linear function,

$$z = f(x_1, x_2, x_3) = (x_1 \cdot x_2) \oplus (x_2 \cdot x_3) \oplus x_3. \tag{17}$$

This would appear to have very nice properties: its linear complexity is given by

$$L_1 \cdot L_2 + L_2 \cdot L_3 + L_3$$

and its period is given by

$$(2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1).$$

However, it turns out to be cryptographically weak. To understand the weakness of the Geffe generator consider the following table, which presents the outputs x_i of the constituent LFSRs and the resulting output z of the Geffe generator

x_1	x_2	x_3	z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

If the Geffe generator was using a "good" non-linear combining function then the output bits z would not reveal any information about the corresponding output bits of the constituent LFSRs. However, we can easily see that

$$\Pr(z = x_1) = 3/4 \text{ and } \Pr(z = x_3) = 3/4.$$

This means that the output bits of the Geffe generator are correlated with the bits of two of the constituent LFSRs. Hence, we can attack the generator using a correlation attack, as follows. Suppose we know the lengths L_i of the constituent generators, but not the connection polynomials or their initial states. The attack is desribed in Algorithm 12.1.

[1]The total degree of a polynomial in n variables is the maximum sum of the degrees in each monomial term.

Algorithm 12.1: Correlation attack on the Geffe generator

for all primitive connection polynomials of degree L_1 **do**
  for all initial states of the first LFSR **do**
    Compute $2 \cdot L_1$ bits of output of the first LFSR.
    Compute how many are equal to the output of the Geffe generator.
    A large value signals that this is the correct choice of generator and starting state.
Repeat the above for the third LFSR.
Recover the second LFSR by testing possible values using equation (17).

It turns out that there are a total of

$$S = \phi(2^{L_1} - 1) \cdot \phi(2^{L_2} - 1) \cdot \phi(2^{L_3} - 1)/(L_1 \cdot L_2 \cdot L_3)$$

possible connection polynomials for the three LFSRs in the Geffe generator. The total number of initial states of the Geffe generator is

$$T = (2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1) \approx 2^{L_1+L_2+L_3}.$$

This means that the key size of the Geffe generator is

$$S \cdot T \approx S \cdot (2^{L_1+L_2+L_3}).$$

For a secure stream cipher we would like the size of the key space to be about the same as the number of operations needed to break the stream cipher. However, the above correlation attack on the Geffe generator requires roughly

$$S \cdot (2^{L_1} + 2^{L_2} + 2^{L_3})$$

operations. The reason for the reduced complexity is that we can deal with each constituent LFSR in turn.

To combine high linear complexity and resistance to correlation attacks (and other attacks) designers have had to be a little more ingenious in their choice of non-linear combiners for LFSRs. We now outline a small subset of some of the most influential.

12.3.1. Filter Generator: The basic idea here is to take a single primitive LFSR with internal state $s_1, \ldots, s_L$ and then make the output of the stream cipher a non-linear function of the whole state, i.e. $z = F(s_1, \ldots, s_L)$. If F has non-linear order m then the linear complexity of the resulting sequence is given by

$$\sum_{i=1}^{m} \binom{L}{i}.$$

12.3.2. Alternating-Step Generator: This takes three LFSRs of size L_1, L_2 and L_3 which are pairwise coprime and of roughly the same size. Denote the output sequence of the three LFSRs by x_1, x_2 and x_3. The first LFSR is clocked on every iteration; if its output x_1 is equal to one, then the second LFSR is clocked and the output of the third LFSR is repeated from its last value. If the output of x_1 is equal to zero, then the third LFSR is clocked and the output of the second LFSR is repeated from its last value. The output of the generator is the value of $x_2 \oplus x_3$. This operation is described graphically in Figure 12.11, where (as in Chapter 10) we denote a clocking signal by a black dot to the left of the LFSR which is being clocked. The LFSR will clock one step if the wire has a one on it, and will otherwise remain in its current state. The alternating-step generator has period

$$2^{L_1} \cdot (2^{L_2} - 1) \cdot (2^{L_3} - 1)$$

and linear complexity approximately $(L_2 + L_3) \cdot 2^{L_1}$.

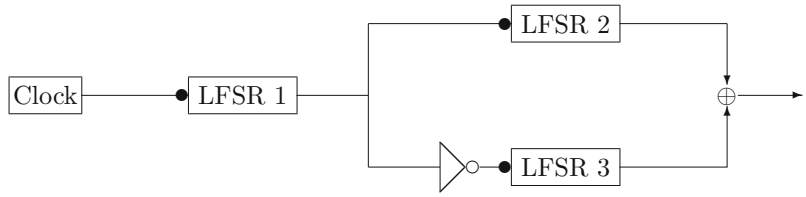

FIGURE 12.11. Graphical representation of the alternating-step generator

12.3.3. Shrinking Generator: Here we take two LFSRs with output sequence x_1 and x_2, and the idea is to throw away some of the x_2 stream under the control of the x_1 stream. Both LFSRs are clocked at the same time, and if x_1 is equal to one then the output of the generator is the value of x_2. If x_1 is equal to zero then the generator just clocks again. Note that, consequently the generator does not produce a bit on each iteration. This operation is described graphically in Figure 12.12. If we assume that the two constituent LFSRs have size L_1 and L_2 with $\gcd(L_1, L_2)$ equal to one, then the period of the shrinking generator is equal to

$$(2^{L_2} - 1) \cdot 2^{L_1 - 1}$$

and its linear complexity is approximately $L_2 \cdot 2^{L_1}$.

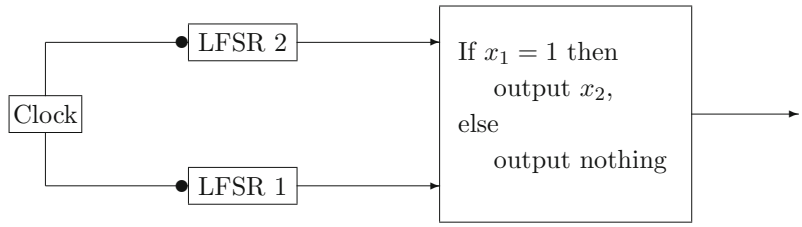

FIGURE 12.12. Graphical representation of the shrinking generator

12.3.4. The A5/1 Generator: Probably the most famous of the LFSR-based stream ciphers is A5/1. This was the stream cipher used to encrypt the on-air traffic in the second generation (a.k.a. GSM) mobile phone networks in Europe and the US. It was developed in 1987, but its design was kept secret until 1999 when it was reverse engineered. There is a weakened version of the algorithm called A5/2 which was designed for use in places to which there were various export restrictions. Various attacks have been published on A5/1 so that it is no longer considered a secure cipher. For example, in 2006 it was shown that one could break into mobile phone conversations which had been protected with A5/1 essentially in real time. In the replacement for GSM, i.e. UMTS (a.k.a. 3G networks) and LTE (a.k.a. 4G networks), the A5/1 cipher has been replaced with the block cipher KASUMI applied in a stream cipher mode of operation.

The stream cipher A5/1 makes use of three LFSRs of lengths 19, 22 and 23. These have characteristic polynomials

$$x^{18} + x^{17} + x^{16} + x^{13} + 1,$$
$$x^{21} + x^{20} + 1,$$
$$x^{22} + x^{21} + x^{20} + x^7 + 1.$$

Alternatively (and equivalently) their connection polynomials are given by

$$x^{18} + x^5 + x^2 + x^1 + 1,$$
$$x^{21} + x^1 + 1,$$
$$x^{22} + x^{15} + x^2 + x^1 + 1.$$

The output of the cipher is the exclusive-or of the three output bits of the three LFSRs.

To clock the registers we associate with each register a "clocking bit". These are in positions 10, 11 and 12 of the LFSRs (assuming bits are ordered with 0 corresponding to the output bit; other books may use a different ordering). We will call these bits c_1, c_2 and c_3. At each clock step the three bits are computed and the "majority bit" is determined via the formulae

$$(c_1 \cdot c_2) \oplus (c_2 \cdot c_3) \oplus (c_1 \cdot c_3).$$

The ith LFSR is then clocked if the majority bit is equal to the bit c_i. Thus clocking occurs subject to the following table.

			Majority	Clock LFSR		
c_1	c_2	c_3	Bit	1	2	3
0	0	0	0	Y	Y	Y
0	0	1	0	Y	Y	N
0	1	0	0	Y	N	Y
0	1	1	1	N	Y	Y
1	0	0	0	N	Y	Y
1	0	1	1	Y	N	Y
1	1	0	1	Y	Y	N
1	1	1	1	Y	Y	Y

We see that in A5/1, each LFSR is clocked with probability 3/4. This operation is described graphically in Figure 12.13, where the "gate" given by an equals sign is the equality-testing gate, and the "gate" labelled by "Maj" is the majority function described above.

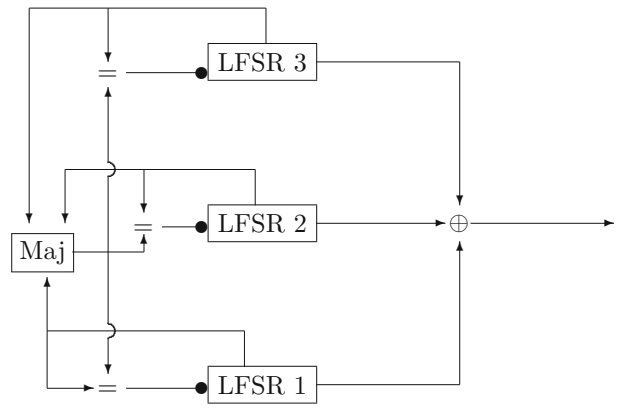

FIGURE 12.13. Graphical representation of the A5/1 generator

12.3.5. Trivium: Trivium is a relatively recent hardware design for a stream cipher, which appears to be more secure than previous designs based on shift registers (although the security of Trivium is not fully guaranteed at this point in time, with some theoretical attacks on it having been presented). The basis of Trivium is a set of three shift registers called a, b and c, of lengths 93, 84 and 111 bits respectively (making 288 bits in total). Once the state has been set up the three shift registers feed into each other via the following equations, over $\mathbb{F}_2$:

$$\begin{aligned} a_i &= c_{i-111} + c_{i-110} + c_{i-109} + c_{i-66} + a_{i-69}, \\ b_i &= a_{i-93} + a_{i-92} + a_{i-91} + a_{i-66} + b_{i-78}, \\ c_i &= b_{i-84} + b_{i-83} + b_{i-82} + b_{i-69} + c_{i-87}. \end{aligned}$$

Notice the regular pattern here: the three top bits of a, b or c are combined with a lower bit (in position 66 or 69) and then with a bit of a second register, to obtain a new bit in the second register. The output bit of Trivium is then obtained from the $\mathbb{F}_2$-equation

$$r_i = c_{i-111} + a_{i-93} + b_{i-84} + c_{i-66} + a_{i-66} + b_{i-84}.$$

To initialize the state an 80-bit key $k_0, \ldots, k_{79}$ and an (up to) 80-bit initial value (IV) $v_0, \ldots, v_{79}$ are fed into the lower bits of the a and b registers, with a getting the key, and b the IV. The rest of the bits of all registers are set to zero, bar the top three bits of the c register. The system is then clocked $4 \cdot 288 = 1152$ times before any keystream is actually used.

Note that this is the first of the stream ciphers we have looked at which explicitly utilizes an IV. We shall see IVs being used in the next chapter on block ciphers, but the basic reason for using them is to move beyond the IND-PASS security of Theorem 12.1. The IV essentially provides a unique input to the keyed PRF that we are trying to produce. So in theoretical terms our cipher becomes $c = m \oplus F_k(\mathrm{IV})$. We do not discuss the theoretical implications here, since much of the discussion on block ciphers in the next chapter will be directly applicable in this situation as well.

12.4. RC4

RC stands for Ron's Cipher after Ron Rivest of MIT. You should not think that the RC4 cipher is a prior version of the block ciphers RC5 and RC6. It is in fact a very, very fast stream cipher. It is easy to remember since it is surprisingly simple. Up until quite recently it was widely deployed in browsers to secure traffic to websites using the TLS protocol. However, recent analysis has shown that the random stream produced by the RC4 algorithm does not behave in a random manner. In particular, each output byte has a particular bias. What is surprising is that the recent analysis is relatively straightforward but the biases had not been discovered in over twenty years of use of the RC4 algorithm. Now that the vulnerability is known, RC4 should no longer be used. However, we present it since it is both historically important and elegantly simple in design.

To describe RC4 we take an array S, indexed from 0 to 255, consisting of the integers $0, \ldots, 255$, permuted in some key-dependent way. The output of the RC4 algorithm is a keystream of bytes K which is exclusive-or'ed with the plaintext byte by byte. Since the algorithm works on bytes and not bits and uses very simple operations, it is particularly fast in software. We start by letting $i = 0$ and $j = 0$. We then repeat the steps in Algorithm 12.2.

Algorithm 12.2: RC4 algorithm

$i \leftarrow (i + 1) \mod 256$.
$j \leftarrow (j + S_i) \mod 256$.
$\mathrm{swap}(S_i, S_j)$.
$t \leftarrow (S_i + S_j) \mod 256$.
$K \leftarrow S_t$.

The security rests on the observation that even if the attacker knows K and i, he can deduce the value of S_t, but this does not allow him to deduce anything about the internal state of the table. This follows from the observation that he cannot deduce the value of t, as he does not know j, S_i or S_j. It is a very tightly designed algorithm as each line of the code needs to be there to make the cipher immune to trivial attacks:

- $i \leftarrow (i+1) \mod 256$**:**
  Makes sure every array element is used once after 256 iterations.
- $j \leftarrow (j+S_i) \mod 256$**:**
  Makes the output depend non-linearly on the array.
- **swap**(S_i, S_j)**:**
  Makes sure the array is evolved and modified as the iteration continues.
- $t \leftarrow (S_i + S_j) \mod 256$**:**
  Makes sure the output sequence reveals little about the internal state of the array.

The initial state of the array S is determined from the key using Algorithm 12.3.

Algorithm 12.3: RC4 key schedule

for $i = 0$ **to** 255 **do** $S_i \leftarrow i$.
Initialize K_i, for $i = 0, \ldots, 255$, with the key, repeating if neccesary.
$j \leftarrow 0$.
for $i = 0$ **to** 255 **do**
  $j \leftarrow (j + S_i + K_i) \mod 256$.
  swap(S_i, S_j).

Chapter Summary

- Many modern stream ciphers can be obtained by combining, in a non-linear way, simple bit generators called LFSRs.
- LFSR-based stream ciphers are very fast ciphers, suitable for implementation in hardware, to encrypt real-time data such as voice or video. But they need to be augmented with a method to produce a form of non-linear output.
- RC4 provides a fast and compact byte oriented stream cipher for use in software, but it is no longer considered secure.

Further Reading

A good introduction to linear recurrence sequences over finite fields is in the book by Lidl and Niederreiter. This book covers all the theory one requires, including examples and a description of the Berlekamp–Massey algorithm. The attacks on the A5/1 algorithm are described in the paper by Barkan et al. The paper by AlFardan et al. covers recent analysis of the RC4 stream cipher.

N.J. AlFardan, D.J. Bernstein, K.G. Paterson, B. Poettering and J.C.N. Schuldt. *On the security of RC4 in TLS*. USENIX Security Symposium, 305–320, USENIX Association, 2013.

E. Barkan, E. Biham and N. Keller. *Instant ciphertext-only cryptanalysis of GSM encrypted communication.* Journal of Cryptology, **21**, 391–429, 2008.

R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, 1986.