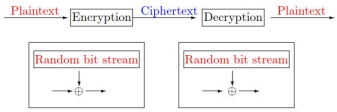
CS 5602 Introduction to Cryptography
Lecture 22
Modern Stream Ciphers

George Markowsky
Computer Science Department
Missouri University of Science & Technology

1

Efficient One-Time Pad?

- We know that the one-time pad is secure, but not practical for daily operations
- What if we could simulate a semi one-time pad?



2

Stream Ciphers

- Think of a black box that emits 0's and 1's
- Imagine if they were at random, we could have a one-time pad that we could use to encode a stream of bits
- There are two immediate problems

1. How do you actually generate "random bits"?
2. How do you save the random bits so they can be used to decode the message

- What if you generated "pseudo-random bits" with a short key?
- How could you do that?

3

Linear Feedback Shift Register

FIGURE 1. Feedback shift register

- The idea is that when you "shift" the register, bit s_0 pops out and can be used
- You create a new bit, s_{L-1}, by having some function of selected bits
- In general, not all the bits are used
- Most commonly the bits used are XORed together and the result becomes the new value of the register, s_{L-1}
- This value is computed first and then the register is shifted

4

```
def LFSR(register,fb):
    # init is the starting pattern and gives the length of the  register
    # fb is the feedback vector -- delivers the register with an extra bit
    # at the beginning. Using strings for compactness.
    eb = False
    for i in range(len(register)):
        if (register[i] == '1') and (fb[i] == '1'):
            eb = not eb
    return str(int(eb)) + register

def stream(init,fb,num):
    register = init
    print register[-1],
    for i in range(num):
        register = LFSR(register,fb)
        print register[-1],
        register = register[:-1]
    print
```

Compute the following by hand for a few terms – what will be the first several terms?
stream('1100','0001',100)

Register	Output Bit fb = 0001
1100	
0110	0
0011	0
1001	1
1100	1
0110	0
0011	0
1001	1
1100	1
cycle	cycle

5

stream('1100','0001',100)

0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

Length = 4

stream('1100','0010',100)

0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1

Length = 3

stream('1100','0011',100)

0 0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 1 0
1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1
1 0 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1

Length = 15

6

Register	Output Bit fb = 0010
1100	
0110	0
1011	0
1101	1
0110	1
1011	0
1101	1
0110	1
1011	0
1101	1
0110	1

Term	Decimal	Register	Output Bit fb = 0011
0	12	1100	
1	6	0110	0
2	11	1011	0
3	5	0101	1
4	10	1010	1
5	13	1101	0
6	14	1110	1
7	15	1111	0
8	7	0111	1
9	3	0011	1
10	1	0001	1
11	8	1000	1
12	4	0100	0
13	2	0010	0
14	9	1001	0
15	12	1100	1

7

Some Quesitons

- What value of the shift register must be avoided at all costs?
- Why?
- Is 15 the best we can do with a 4 bit register?
- Why?
- What is the absolute best that we can hope to do with a k-bit register?
- Why?
- Can we always reach that maximum?
- Sounds like a mathematical question, that perhaps we can answer

8

Period of a Sequence

- Definition: The period of a sequence is defined to be the smallest integer N such that $s_{N+i} = s_i$ for sufficiently large N.
- As noted before you cannot have a period greater that $2^k - 1$ if you have a k-bit shift register since you must avoid 0

9

Matrix Multiplication

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 be eventually periodic (since we must eventually return to a state we have already been in).

10

LFSR and Matrix Multiplication

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

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

and

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

and we write the internal state as

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

then the next state can be deduced by computing

$$s = M \cdot s$$

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

$$v \cdot s.$$

11

The Connection Polynomial

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

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

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

$$C(X) = \det(XM - I_L).$$

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

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

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

As an example see Fig. 2 for an LFSR in which the connection polynomial is given by $X^3 + X + 1$ and Fig. 3 for an LFSR in which the connection polynomial is given by $X^{32} + X^3 + 1$.

12

Examples of Polynomials

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

FIGURE 2. Linear feedback shift register: $X^3 - X - 1$

Of particular importance is when the connection polynomial is primitive.

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

13

Primitive Polynomials

DEFINITION 7.1. *A binary polynomial $C(X)$ of degree L is primitive if it is irreducible and a root θ of $C(X)$ generates the multiplicative group of the field $\mathbb{F}_{2^L}$. In other words, since $C(X)$ is irreducible we already have*

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

but we also require

$$\mathbb{F}_{2^L}^* = \langle \theta \rangle.$$

14

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

- $c_L = 0$:
  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 four bit registers, i.e. $L = 4$.

15

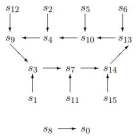
Example 1

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 4. Note, it is not purely periodic.

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



Note that we do not use the connection polynomial to compute the bits!

16

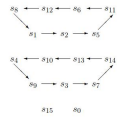
Example 2

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 5. Note, it is purely periodic, but that there are different period lengths due to the different factorization properties of the connection polynomial modulo 2. One 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 period of the zero'th state.

FIGURE 5. Transitions of the four bit LFSR with connection polynomial $X^4+X^3+X^2+1$



17

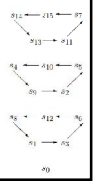
Example 3

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 6. Note, it is purely periodic and all periods have same length, bar the trivial one.

FIGURE 6. Transitions of the four bit LFSR with connection polynomial $X^4+X^3+X^2+X+1$



18

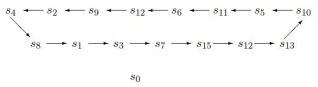
Example 4

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 7.

FIGURE 7. Transitions of the four bit LFSR with connection polynomial $X^4 + X + 1$



19

Primitive Polynomials

Whilst there are algorithms to generate primitive polynomials for use in applications we shall not describe them here. We give some samples in the following list, where we give polynomials with a small number of taps for efficiency.

$$\begin{array}{lll} x^{31} + x^3 + 1 & x^{31} + x^6 + 1 & x^{31} + x^7 + 1 \\ x^{39} + x^4 + 1 & x^{60} + x + 1 & x^{63} + x + 1 \\ x^{71} + x^6 + 1 & x^{93} + x^2 + 1 & x^{127} + x^{21} + 1 \\ x^{145} + x^{52} + 1 & x^{161} + x^{18} + 1 & x^{521} + x^{32} + 1 \end{array}$$

20

Cryptographic Weakness of LFSR

- Although LFSRs efficiently produce bitstreams 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 has L bits, and we can determine 2L consecutive bits of the stream then we can determine the whole stream.
- First notice 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.

21

Finding the Taps

Using the equation

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

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

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

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

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

22

Linear Complexity

An important measure of the cryptographic quality of a sequence is given by the linear complexity of the sequence.

DEFINITION 7.2 (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,*
- $L(s)$ *will be the length of the shortest LFSR to generate s.*

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.

23

Berlekamp-Massey Algorithm

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, this 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.

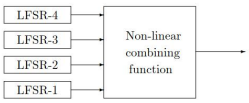
Hence, if we use an LFSR of size L to generate a keystream for a stream cipher and the adversary obtains at least $2L$ bits of this keystream then they can determine the exact LFSR used and so generate as much of the keystream as they wish. Therefore, one needs to find a way of using LFSRs in some non-linear way, which hides the linearity of the LFSRs and produces output sequences with high linear complexity.

24

Combining LFSRs

To use LFSRs in practice it is common for a number of them to be used, producing a set of output sequences $x_1^{(i)}, \ldots, x_n^{(i)}$. 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 Fig. 8.

FIGURE 8. Combining LFSRs



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.

25

Combining LFSRs

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 the linear complexity of the keystream generated by $f(x_1, \ldots, x_n)$ is equal to

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

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 equal to total the degree of f.

However, it turns out that creating a nonlinear 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 non-linear function, of non-linear order 2,

$$z = f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_2 \cdot x_3 \oplus x_3. \tag{12}$$

This would appear to have very nice properties: It's linear complexity is given by

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

and it's period is given by

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

However, it turns out to be cryptographically weak.

26

The Geffe Generator

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. This means that we can attack the generator using a correlation attack.

27

Correlation Attack

This attack proceeds 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 7.1

Algorithm 7.1: Correlation attack on the Geffe generator

```
forall the primitive connection polynomials of degree L1 do
    forall the Initial states of the first LFSR do
        Compute 2L1 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.
    end
end
Repeat the above for the third LFSR
Recover the second LFSR by testing possible values using (12)
```

It turns out that their 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

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

28

Correlation Attack

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 as to how they have produced non-linear combiners for LFSRs. We now outline a small subset of some of the most influential:

29

Filter Generator

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 be 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

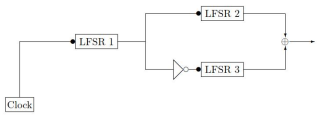
$$\sum_{i=1}^{m} \binom{L}{i}.$$

30

Alternating Step Generator

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. If the output sequence of the three LFSRs is denoted by x_1, x_2 and x_3, then one proceeds as follows: 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 9.

FIGURE 9. Graphical representation of the Alternating step generator



The alternating step generator has period

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

and linear complexity, approximately

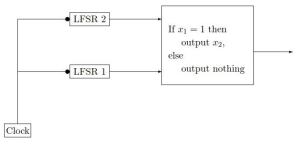
$$(L_2+L_3)\cdot 2^{L_1}.$$

31

The Shrinking Generator

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 this means that the generator does not produce a bit on each iteration. This operation is described graphically in Figure 10.

FIGURE 10. Graphical representation of the Shrinking Generator



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

$$L_2 \cdot 2^{L_1}.$$

32

The A5/1 Generator

The A5/1 Generator: Probably the most famous of the recent LFSR based stream ciphers is A5/1. This is the stream cipher used to encrypt the on-air traffic in the GSM mobile phone networks in Europe and the US. This was developed in 1987, buts 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 where there were various export restrictions. In recent years various attacks have been published on A5/1 which has resulted in it no longer being considered a secure cipher. In the replacement for GSM, i.e. UMTS or 3G networks, the cipher has been replaced with the use of the block cipher KASUMI in a stream cipher mode of operation.

A5/1 makes use of three LFSRs of length 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.

33

The A5/1 Generator

To clock the registers we associate to each register a "clocking bit". These are in positions 10, 11 and 12 of the LFSR's (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

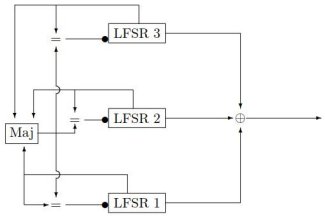
			Majority	Clock LFSR		
c_1	c_2	c_3	Bit	1	2	3
0	0	0	0	Y	Y	Y
0	0	1	0	Y	Y	N
0	1	0	0	Y	N	Y
0	1	1	1	N	Y	Y
1	0	0	0	N	Y	Y
1	0	1	1	Y	N	Y
1	1	0	1	Y	Y	N
1	1	1	1	Y	Y	Y

Thus we see in A5/1 that each LFSR is clocked with probability 3/4. This operation is described graphically in Figure 11.

34

The A5/1 Generator

FIGURE 11. Graphical representation of the A5/1 Generator



35

RC4

- In cryptography, RC4 (Rivest Cipher 4 also known as ARC4 or ARCFOUR meaning Alleged RC4, see below) is a stream cipher.
- While remarkable for its simplicity and speed in software, multiple vulnerabilities have been discovered in RC4, rendering it insecure.
- It is especially vulnerable when the beginning of the output keystream is not discarded, or when nonrandom or related keys are used.
- Particularly problematic uses of RC4 have led to very insecure protocols such as WEP.
- As of 2015, there is speculation that some state cryptologic agencies may possess the capability to break RC4 when used in the TLS protocol.
- IETF has published RFC 7465 to prohibit the use of RC4 in TLS; Mozilla and Microsoft have issued similar recommendations.
- A number of attempts have been made to strengthen RC4, notably Spritz, RC4A, VMPC, and RC4+.

36

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 very easy to remember since it is surprisingly simple.
- Given an array S indexed from 0 to 255 consisting of the integers 0, . . . , 255, permuted in some key-dependent way, the output of the RC4 algorithm is a keystream of bytes K which is XORed 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.

37

RC4

```
def KSA(key): # Generates a permuation based on key
    S = range(256)
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i%len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    return S
def RC4(S,numBits):
    i = 0
    j = 0
    count = 0
    while count < numBits:
        i = (i+1)%256
        j = (j + S[i])%256
        S[i],S[j] = S[j],S[i]
        print S[(S[i]+S[j])%256],
        count += 1

S = KSA([4, 6, 3, 2])
RC4(S,100)
```

222 203 251 191 99 72 30 73 211 162 245 89 242 117 93 151
183 213 163 211 98 124 187 207 55 62 15 233 41 34 171 53 86
169 160 233 7 140 90 114 161 200 93 117 248 254 19 244 78
13 238 79 31 163 102 185 223 156 61 113 33 89 88 206 205
250 128 235 175 196 75 185 246 36 135 2 221 229 113 221 112
64 48 15 238 16 141 131 240 134 84 4 47 44 148 4 124 235 166
208

38

RC4

- It is a very tightly designed algorithm as each line of the code needs to be there to make the cipher secure.
- • i = (i + 1) mod 256 : Makes sure every array element is used once after 256 iterations.
- • j = (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 = (S_i + S_j) mod 256 : Makes sure the output sequence reveals little about the internal state of the array.
- Return S_t
- Although RC4 is very fast for a software based stream cipher, there are some issues with its use.
- In particular both the key schedule and the main algorithm do not produce as random a stream as one might wish.
- Hence, it should be used with care.

39

Chapter Summary

- Modern stream ciphers can be obtained by combining, in a non-linear way, simple bit generators called LFSRs, these stream ciphers are bit oriented.
- LFSR based stream ciphers provide very fast ciphers, suitable for implementation in hardware, which can encrypt real-time data such as voice or video.
- RC4 provides a fast and compact byte oriented stream cipher for use in software. USE with CARE!

40