

COMSM2004 : Random Streams of Bits

B. Warinschi and N.P. Smart

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB
United Kingdom.

January 30, 2009

Outline

Random Numbers

Linear Feedback Shift Registers

Stream Ciphers

Random Numbers

Almost all cryptographic systems require random numbers

Randomization Essential to Encryption

- ▶ Semantic security of public key systems
 - ▶ RSA-OAEP, ElGamal etc

Signature Schemes Require Randomization

- ▶ DSA, RSA-PSS etc

A good random number generator is crucial

- ▶ Security usually rests on the quality of the random numbers used

Random Number Generators

Can use **physical** random number generation

- ▶ Decaying of radioactive isotopes
- ▶ Hardware characteristics
 - ▶ Disk spin down/up times etc
- ▶ OS state

Or **pseudo**-random number generators

- ▶ Software/hardware which expands a small seed into a long random sequence
 - ▶ Often uses cryptographic hash functions

Or combination of both

- ▶ `Linux /dev/random`

Quality of Random Numbers

How do we know a random number generator (RNG) is good ?

Apply statistical tests

The sequences generated should **pass** certain statistical tests as genuinely random sequences would

For RNGs generating bit sequences one usually uses the following tests

- ▶ Frequency Test
- ▶ Serial Test
- ▶ Poker Test
- ▶ Runs Test
- ▶ Autocorrelation Test

Statistical Test

A statistical test usually works as follows

Null Hypothesis

This is essentially what we think is true

- ▶ i.e. the given bit stream looks like a random sequence would

Collect some data

i.e. A sample of the bitstream

Compute some “test statistic”

We **reject** the **Null Hypothesis** if data does not support the hypothesis

- ▶ i.e. test statistic is too small or large

Statistical Testing

If the **null hypothesis** is **true** then the computed test statistic should follow a given statistical distribution

- ▶ Normal distribution
- ▶ χ^2 distribution

So we know whether the test statistic is in the right range

See Handbook of Applied Crypto Chapter 5 for more theoretical treatment

- ▶ We simply now give the tests usually used

Let $s = s_0, s_1, s_2, \dots, s_{n-1}$ be a binary sequence of length n

- ▶ This is the **sample** we have taken

Frequency Test

Basic Idea:

We expect there to be a roughly equal number of ones and zero's

Let

- ▶ n_0 be the number of zero's in the sequence
- ▶ n_1 be the number of ones in the sequence

Compute the **test statistic**

$$X = \frac{(n_0 - n_1)^2}{n}.$$

Frequency Test

The test statistic

$$X = \frac{(n_0 - n_1)^2}{n}.$$

Would expect X to be close to zero if sequence is truly random

Formally X should follow a χ^2 distribution with one degree of freedom

- ▶ Expect 5 percent of **random** sequences to have a value of X greater than 3.84

This test is sometimes called the **Monobit Test**

Frequency Test: Example

Consider the (non-random) sequence s of length $n = 160$ obtained by replicating the following sequence four times

▶ 11100 01100 01000 10100 11101 11100 10010 01001

We compute

▶ $n_0 = 84$

▶ $n_1 = 76$

The value of the statistic X is 0.4.

Therefore have no reason to assume it is not random given the Frequency Test

▶ Formally we **accept the null hypothesis**

Serial Test

Basic Idea:

Tests whether the frequencies of 00, 01, 10 and 11 are approximately equal

Let

- ▶ n_0, n_1 denote the number of 0's and 1's in s
- ▶ $n_{00}, n_{01}, n_{10}, n_{11}$ denote the number of occurrences of 00, 01, 10, 11 in s

Compute the **test statistic**

$$X = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1.$$

Formally X should follow χ^2 distribution with two degrees of freedom

- ▶ Expect 5 percent of **random** sequences to have a value of X greater than 5.99

Serial Test: Example

Again consider the (non-random) sequence s of length $n = 160$ obtained by replicating the following sequence four times

▶ 11100 01100 01000 10100 11101 11100 10010 01001

We compute

▶ $n_{00} = 44$

▶ $n_{01} = 40$

▶ $n_{10} = 40$

▶ $n_{11} = 35$

The value of the statistic X is 0.6252

Therefore have no reason to assume it is not random given the Serial Test

▶ Formally we **accept the null hypothesis**

Poker Test

Basic Idea:

Splits sequence into “poker hands” and tests whether hands are equally likely

Let

- ▶ m be a positive integer such that $\lfloor n/m \rfloor \geq 5 \cdot 2^m$
- ▶ $k = \lfloor n/m \rfloor$

Divide the sequence s into k non-overlapping parts each of length m ,

- ▶ Let n_i be the number of occurrences of the i th type of sequence of length m , $1 \leq i \leq 2^m$.

Poker Test

Each of the k parts correspond to a different “poker hand”

Compute the **test statistic**

$$X = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

Formally X should follow χ^2 distribution with $2^m - 1$ degrees of freedom

If $m = 1$ the poker test reduces to the frequency test

Poker Test: Example

Again consider the (non-random) sequence s of length $n = 160$ obtained by replicating the following sequence four times

▶ 11100 01100 01000 10100 11101 11100 10010 01001

Take

- ▶ $m = 3$
- ▶ $k = 53$

The blocks

▶ 000, 001, 010, 011, 100, 101, 110, 111

appear 5, 10, 6, 4, 12, 3, 6 and 7 times, respectively

Poker Test: Example

With this data...

The value of the statistic X is 9.6415

- ▶ Threshold value for 7 degrees of freedom is 14.06

Therefore have no reason to assume it is not random given the Poker Test

- ▶ Formally we **accept the null hypothesis**

Runs Test

Basic Idea:

Tests whether runs of 0s and 1s occur with expected frequency

The expected number of gaps (or blocks) of length i in a random binary sequence of length n is

- ▶ $e_i = (n - i + 3)/2^{i+2}$.

Let

- ▶ k be equal to the largest integer i for which $e_i \geq 5$.
- ▶ B_i, G_i be the number of blocks and gaps of length i in s .

Runs Test

Given k , B_i and G_i

Compute the **test statistic**

$$X = \sum_{i=1}^k \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^k \frac{(G_i - e_i)^2}{e_i}$$

Formally X should follow χ^2 distribution with $2k - 2$ degrees of freedom

Runs Test: Example

Again consider the (non-random) sequence s of length $n = 160$ obtained by replicating the following sequence four times

- ▶ 11100 01100 01000 10100 11101 11100 10010 01001

Compute

- ▶ $k = 3$
- ▶ $e_1 = 20.25, e_2 = 10.0625, e_3 = 5$
- ▶ There are 25, 4, 5 blocks of lengths 1, 2, 3, respectively
- ▶ There are 8, 20, 12 gaps of lengths 1, 2, 3, respectively.

The value of the statistic X is 31.7913

- ▶ Threshold value for 4 degrees of freedom is 9.49

Therefore we expect it is not random given the Runs Test

Formally we **reject** the **null hypothesis**

- ▶ Conclude sequence is probably not random

Autocorrelation Test

Basic Idea:

sequence s and (non-cyclic) shifted versions of it.

Let d be a fixed integer, $1 \leq d \leq \lfloor n/2 \rfloor$.

The number of bits in s not equal to their d -shifts is

$$\blacktriangleright A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$$

Compute the **test statistic**

$$X = \left(A(d) - \frac{n-d}{2} \right) / \sqrt{n-d}$$

Formally X should follow **normal** distribution $N(0, 1)$

\blacktriangleright Threshold value is ± 1.96

Autocorrelation Test: Example

Again consider the (non-random) sequence s of length $n = 160$ obtained by replicating the following sequence four times

- ▶ 11100 01100 01000 10100 11101 11100 10010 01001

Let

- ▶ $d = 8$

then

- ▶ $A(8) = 100$

The value of the statistic X is 3.8933

Therefore we expect it is not random given the Autocorrelation Test

Formally we **reject** the **null hypothesis**

- ▶ Conclude sequence is probably not random

FIPS 140-1

When testing cryptographic hardware and software various agencies spend a long time looking at the random number generator

- ▶ Since this is the most likely place for problems to occur

US Federal standard FIPS 140-1 specifies the following as its test

Take a single output bit sequence of 20000 bits

Then do four tests

- ▶ If any test fails then the RNG fails

FIPS 140-1

Given 20000 bits of output

Frequency Test

- ▶ The number of ones n_1 should satisfy $9654 \leq n_1 \leq 10346$

Poker Test

- ▶ The poker test statistic X is computed for $m = 4$
- ▶ Pass if $1.03 < X < 57.4$

Runs Test

- ▶ For $1 \leq i \leq 6$ compute B_i and G_i , which should satisfy
- ▶ $G_1, B_1 \in [2267, 2733]$, $G_2, B_2 \in [1079, 1421]$,
- ▶ $G_3, B_3 \in [502, 748]$, $G_4, B_4 \in [223, 402]$,
- ▶ $G_5, B_5 \in [90, 223]$, $G_6, B_6 \in [90, 223]$.

Long Run Test

- ▶ Passes if no runs of length 34 or more

Statistical Testing Caveat

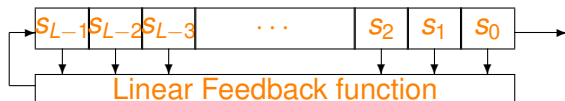
Passing a set of statistical tests is a **necessary** condition for security

Not a sufficient condition!

There are many weak proposals that pass statistical testing!

Linear Feedback Shift Registers

A popular way of generating a random sequence of bits in hardware/software is to use **Linear Feedback Shift Registers** or **LFSR's**



Finite State Machines

An LFSR is an example of a **finite state machine**

This is a machine which consists of

- ▶ A finite set $S = \{S_1, \dots, S_n\}$ of states
- ▶ A finite input alphabet $A = \{A_1, \dots, A_a\}$
- ▶ A finite output alphabet $B = \{B_1, \dots, B_b\}$
- ▶ A state transition function $F : S \times A \longrightarrow S$
- ▶ An output function $Y : S \times A \longrightarrow B$

In state S_i with input A_j

- ▶ Machine moves to state $F(S_i, A_j)$
- ▶ Outputs $Y(S_i, A_j)$

Linear Finite State Machine

A **Linear Finite State Machine** is a Finite State Machine with

- ▶ No-input alphabet
- ▶ Output alphabets given by $\mathbb{F}_2 = \{0, 1\}$
- ▶ Space State given by $S = \mathbb{F}_2^L$, i.e. an L -dimensional vector s

Let M be an $L \times L$ matrix over \mathbb{F}_2

Next state function (**matrix multiplication**)

$$F(s) = M \cdot s$$

Output function (**vector product**)

$$Y(s) = v \cdot s$$

for some fixed vector v

LFSRs

All linear finite state machines can be written in the following form

Matrix M

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

Vector v

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

LFSRs

Note

$$\begin{aligned}C(X) &= \det(XM - I_L) \\ &= 1 + c_1X + c_2X^2 + \dots + c_LX^L\end{aligned}$$

This is called the connection polynomial of the LFSR

- ▶ Sometimes this is written in reverse
 - ▶ $G(X) = X^n C(1/X)$
 - ▶ Then $G(X)$ is **characteristic polynomial** of the matrix M

The c_i are called taps

- ▶ Tap c_i corresponds to state bit s_{L-i}
- ▶ Output bit is s_0

Math Magic

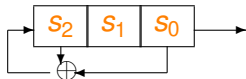
Cayley-Hamilton Theorem says that a matrix satisfies its own characteristic polynomial, i.e.

$$G(M) = 0$$

LFSR Example 1

An LFSR with connection polynomial $X^3 + X + 1$

- ▶ Characteristic polynomial $X^3 + X^2 + 1$



LFSR Example 2

An LFSR with connection polynomial $X^{32} + X^3 + 1$

- ▶ Characteristic polynomial $X^{32} + X^{29} + 1$



LFSRs

Alternatively

Given

- ▶ A register of length L , s_0, s_1, \dots, s_{L-1}
- ▶ A set of bits c_1, \dots, c_L

If output of register is given by

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

then this LFSR is the same as the matrix representation above

LFSR Periods

Since state space is finite the output must eventually become periodic

Period of sequence Smallest N such that for sufficiently large i we have

- ▶ $s_{N+i} = s_i$

There are $2^L - 1$ possible non-zero states so maximum possible period is

- ▶ $2^L - 1$

If $C(X)$ has degree **less than** L

- ▶ Sequence is eventually periodic, but **not necessarily** periodic

From now on assume $\deg C(X) = L$

LFSR Periods

If $C(X)$ is **irreducible** (as polynomial over $\mathbb{F}_2[x]$)

- ▶ All output sequences of period N where
- ▶ N is least positive integer such that
- ▶ $C(X)$ divides $1 + X^N$

If $C(X)$ is **primitive**, i.e. a root generates $\mathbb{F}_{2^L}^*$, then output has **maximal** period $2^L - 1$

Hence usually use LFSR's with primitive connection polynomials $C(X)$.

Intuition

Suppose $C(X)$ is irreducible and the common period is N

Each state s_i corresponds to an element in $\mathbb{F}_{2^L} = \mathbb{F}_2[X]/C(X)$

The matrix M corresponds to the root of $C(X)$

If $C(X)$ is primitive every state can be written as a power of θ

i.e. every state can be written as a power of M

$$s^{(i)} = M^i s^{(0)}.$$

The above can be made mathematically rigorous, but we will not bother

LFSR Period Examples

Suppose we take a four bit register, i.e. $L = 4$

Connection polynomial $C(X) = X^3 + X + 1$,

▶ i.e. $\deg C(X) \neq L$

Matrix M is

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

LFSR Period Examples

State transitions are then given by (s_0 in left most position)

- ▶ $[1, 0, 0, 0] \longrightarrow [0, 0, 0, 0]$
- ▶ $[0, 0, 1, 0] \longrightarrow [0, 1, 0, 0] \longrightarrow [1, 0, 0, 1] \longrightarrow [0, 0, 1, 1] \longrightarrow$
 $[0, 1, 1, 1] \longrightarrow [1, 1, 1, 0] \longrightarrow [1, 1, 0, 1] \longrightarrow [1, 0, 1, 0] \longrightarrow$
 $[0, 1, 0, 0]$
- ▶ $[1, 1, 0, 0] \longrightarrow [1, 0, 0, 1]$
- ▶ $[0, 1, 1, 0] \longrightarrow [1, 1, 0, 1]$
- ▶ $[0, 0, 0, 1] \longrightarrow [0, 0, 1, 1]$
- ▶ $[0, 1, 0, 1] \longrightarrow [1, 0, 1, 0]$
- ▶ $[1, 0, 1, 1] \longrightarrow [0, 1, 1, 1]$
- ▶ $[1, 1, 1, 1] \longrightarrow [1, 1, 1, 0]$

Note, **not** purely periodic

- ▶ Since $\deg C(X) \neq 4$

LFSR Period Examples

Again take four bit register

Connection polynomial

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

Matrix M is

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

LFSR Period Examples

State transitions are then given by (s_0 in left most position)

- ▶ $[0, 0, 0, 0]$
- ▶ $[1, 0, 0, 0] \longrightarrow [0, 0, 0, 1] \longrightarrow [0, 0, 1, 0] \longrightarrow [0, 1, 0, 1] \longrightarrow [1, 0, 1, 1] \longrightarrow [0, 1, 1, 0] \longrightarrow [1, 1, 0, 0] \longrightarrow [1, 0, 0, 0]$
- ▶ $[0, 1, 0, 0] \longrightarrow [1, 0, 0, 1] \longrightarrow [0, 0, 1, 1] \longrightarrow [0, 1, 1, 1] \longrightarrow [1, 1, 1, 0] \longrightarrow [1, 1, 0, 1] \longrightarrow [1, 0, 1, 0] \longrightarrow [0, 1, 0, 0]$
- ▶ $[1, 1, 1, 1]$

Note, purely periodic and different period lengths

- ▶ Since $\deg C(X) = 4$ and $\deg C(X)$ is reducible

LFSR Period Examples

Again take four bit register

Connection polynomial $C(X) = X^4 + X^3 + X^2 + X + 1$

- ▶ Irreducible, not primitive

Matrix M is

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

LFSR Period Examples

State transitions are then given by (s_0 in left most position)

- ▶ $[0, 0, 0, 0]$
- ▶ $[1, 0, 0, 0] \rightarrow [0, 0, 0, 1] \rightarrow [0, 0, 1, 1] \rightarrow [0, 1, 1, 0] \rightarrow [1, 1, 0, 0] \rightarrow [1, 0, 0, 0]$
- ▶ $[0, 1, 0, 0] \rightarrow [1, 0, 0, 1] \rightarrow [0, 0, 1, 0] \rightarrow [0, 1, 0, 1] \rightarrow [1, 0, 1, 0] \rightarrow [0, 1, 0, 0]$
- ▶ $[1, 1, 1, 0] \rightarrow [1, 1, 0, 1] \rightarrow [1, 0, 1, 1] \rightarrow [0, 1, 1, 1] \rightarrow [1, 1, 1, 1] \rightarrow [1, 1, 1, 0]$

Note, purely periodic and all periods have same length

- ▶ Bar the trivial one
- ▶ Since $\deg C(X) = 4$ and $C(X)$ is irreducible

LFSR Period Examples

Again take four bit register

Connection polynomial $C(X) = X^4 + X + 1$

- ▶ irreducible and primitive

Matrix M is

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

LFSR Period Examples

State transitions are then given by (s_0 in left most position)

- ▶ $[0, 0, 0, 0]$
- ▶ $[1, 0, 0, 0] \longrightarrow [0, 0, 0, 1] \longrightarrow [0, 0, 1, 1] \longrightarrow [0, 1, 1, 1] \longrightarrow [1, 1, 1, 1] \longrightarrow [1, 1, 1, 0] \longrightarrow [1, 1, 0, 1] \longrightarrow [1, 0, 1, 0] \longrightarrow [0, 1, 0, 1] \longrightarrow [1, 0, 1, 1] \longrightarrow [0, 1, 1, 0] \longrightarrow [1, 1, 0, 0] \longrightarrow [1, 0, 0, 1] \longrightarrow [0, 0, 1, 0] \longrightarrow [0, 1, 0, 0] \longrightarrow [1, 0, 0, 0]$

Note, purely periodic of maximal length period

- ▶ Since $\deg C(X) = 4$ and $C(X)$ is primitive

m-sequences

A sequence which is the output of an primitive LFSR is called an **m-sequence**

In a cycle of an m-sequence from an L-stage LFSR

- ▶ There are 2^{L-1} ones and $2^{L-1} - 1$ zeros
- ▶ Every block of length r , for $1 \leq r \leq L$, occurs 2^{L-r} times
 - ▶ except the all zeros block which occurs $2^{L-r} - 1$ times

m-sequences

Provided the first bit and the last bit are different

- ▶ There are 2^{L-1} runs
 - ▶ of which 2^{L-2} are runs of zeros and 2^{L-2} are runs of ones
- ▶ There are
 - ▶ 2^{L-r-2} runs of zeros of length r ,
 - ▶ 2^{L-r-2} runs of 1s of length r , for $1 \leq r \leq L - 2$
- ▶ There is one run of zeros of length $L - 1$ and one run of ones of length L

Result is m -sequences pass the random number generator tests

LFSR Software Implementation

Let `ulong` be an unsigned type of bitlength at least $L \leq 32$

- ▶ Generalisation to greater than 32 bits is obvious

Let connection polynomial be a pentanomial (say)

$$C(X) = X^L + X^r + X^s + X^t + 1$$

Compute the mask

```
ulong mask = 1 + (1 << (L-r)) + (1 << (L-s)) + (1 << (L-t));
```

Let the internal state be given by the variable

```
ulong state = initial_key
```

LFSR Software Implementation

To compute next state and output bit we perform

```
int output=state&1;

ulong t=state&mask;

t=(t>>16)^(t&0xFFFF);
t=(t>>8)^(t&0xFF);
t=(t>>4)^(t&0xF);
t=(t>>2)^(t&0x3);
t=(t>>1)^(t&0x1);
state>>=1;
state^=(t<<(L-1));

return output;
```

LFSR Properties

LFSRs have a number of desirable properties

- ▶ **Fast** to implement, especially in hardware
- ▶ **Good** statistical properties
- ▶ **Easy** to determine the period length
 - ▶ Use properties of the connection polynomial

However, **not suitable** on their own for cryptographic stream cipher

- ▶ We usually assume the taps, i.e. the c_j are unknown to the attacker

Suppose we can determine the output stream s_0, s_1, s_2, \dots

- ▶ e.g. from a chosen plaintext attack

We will show we can determine the state of the cipher

LFSR Attack

Assume

- ▶ We know the size of the LFSR, namely L
- ▶ Want to determine the taps c_i

We have the equation, for all j ,

$$s_j - \sum_{i=1}^L c_i s_{j-i} = 0 \pmod{2}$$

So, if we know $2L$ consecutive values of the s_i we can determine c_i via (equation is mod 2)

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

LFSR Attack

The above works once we have computed the size of the LFSR L

- ▶ Need to compute L

This is done using the **Berlekamp–Massey** algorithm

- ▶ This computes the linear complexity of a sequence $\{s_i\}_{i=1}^n$

The linear complexity $L(s, n)$ of a sequence $\{s_i\}_{i=1}^n$ is

- ▶ The **size** of the **smallest** LFSR which generates the sequence
- ▶ Berlekamp–Massey will compute the connection polynomial as well
 - ▶ If $n \geq L(s, n)/2$

Linear Complexity

The **linear complexity** satisfies

- ▶ $L(s, n) = 0$ if s is the zero sequence
- ▶ $0 \leq L(s, n) \leq n$
- ▶ $L(s \oplus t, n) \leq L(s, n) + L(t, n)$

If s is a truly random sequence then

- ▶ $L(s, n) \approx n/2$

If s is output from an LFSR of length L then

- ▶ $L(s, n) = L$ for all $n \geq L$

LFSRs and Stream Ciphers

Hence, we **cannot** use LFSRs **on their own** to create a stream cipher

- ▶ But we want to use them since they are fast and efficient

Need to combine them using a non-linear function

- ▶ Removes the linearity properties

Simple example is to use a **Combination Generator**

- ▶ Take n LFSR's of size L_i and maximal period $2^{L_i} - 1$
 - ▶ i.e. with primitive connection polynomials
- ▶ Take a non-linear boolean function $F(x_1, \dots, x_n)$
 - ▶ e.g. $F(x_1, x_2, x_3, x_4) = (x_1 \cdot x_2) \oplus (x_3 \cdot x_4)$
- ▶ Output $z = F(x_1, \dots, x_n)$

Non-Linear Functions

Let $F(x_1, \dots, x_n)$ denote a non-linear function

Write as a sum of products, **algebraic normal form**

When writing a non-linear function in algebraic normal form the **non-linear order** is the multiplicative term of largest degree

$$(x_1 \cdot x_2) \oplus (x_3 \cdot x_4)$$

- ▶ Has non-linear order 2

$$(x_1 \cdot x_2 \cdot x_4) \oplus (x_3 \cdot x_4)$$

- ▶ Has non-linear order 3

Non-Linear Functions

A high non-linear order is a good property

- ▶ Ensures high linear complexity of the resulting stream cipher

If L_i are all distinct and greater than two then if we use

- ▶ $z_i = F(x_1, \dots, x_n)$ as our keystream

then the linear complexity is given by

- ▶ $f(L_1, \dots, L_n)$

where we replace \oplus/\cdot by standard addition/multiplication

Size of the linear complexity is then (essentially) a function of the
non-linear order

Geffe Generator

Early example is the Geffe Generator (1973)

Three maximal period LFSRs of distinct sizes L_1, L_2, L_3

$$\blacktriangleright z_i = f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_2 \cdot x_3 \oplus x_3$$

This has nice properties

- ▶ Linear Complexity is $L_1 \cdot L_2 + L_2 \cdot L_3 + L_3$
- ▶ Period is $(2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1)$

But it turns out to be weak cryptographically...

Geffe Generator

Suppose

- ▶ Output bits of the three constituent LFSR's are x_1 , x_2 and x_3
- ▶ Output bit of Geffe generator is z

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

Note, $\Pr[z = x_1] = 3/4$ and $\Pr[z = x_3] = 3/4$

- ▶ But if Geffe was a good generator should have probability $1/2$

Geffe Generator

The previous observation leads to a **correlation attack**

- ▶ The output of the Geffe generator is correlated with the output of two of its constituent LFSRs

Suppose we know the lengths L_i of the generators

- ▶ but not the connection polynomials

For each possible (primitive) connection polynomial for the first LFSR

- ▶ For each possible starting state for the first LFSR
 - ▶ Compute the output of the first LFSR (say $2L_1$ bits)
 - ▶ Compute how many are equal to the Geffe output
 - ▶ A large value signals that this is the correct choice of generator and starting state

Geffe Generator

Key size is given by

- ▶ $S = \phi(2^{L_1} - 1) \cdot \phi(2^{L_2} - 1) \cdot \phi(2^{L_3} - 1) / (L_1 \cdot L_2 \cdot L_3)$ possible connection polynomials
- ▶ $T = (2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1) \approx 2^{L_1+L_2+L_3}$ initial states

i.e. Key Size is about

- ▶ $S \cdot (2^{L_1+L_2+L_3})$

But using the above correlation attack we reduce this to

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

as we can deal with each of the three LFSRs in turn

Correlation Immunity vs Linear Complexity

The notion of immunity to such attacks can be characterised precisely

- ▶ See HAC for more details

Problem

- ▶ One needs to trade off immunity against correlation attacks against the non-linear order

Need different way of producing the output key stream from a set of LFSRs

- ▶ Many other ideas been proposed (some examples follow)
- ▶ Most have some problems
- ▶ **Hard** to design a good stream cipher

Filter Generator

Basic Idea

Take a single primitive LFSR with state s_0, \dots, s_{L-1}

Make output bit of the stream cipher be a non-linear function F of the whole state

$$\blacktriangleright z = F(s_0, \dots, s_{L-1})$$

Suppose F has non-linear order m

Linear complexity of sequence is then

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

Clock Controlled Generators

Basic Idea

Use one LFSR to control the **clocking** (i.e. stepping) of a second LFSR

To make output appear at same rate use **Alternating Step Generator**

- ▶ Have three LFSRs with output x_1 , x_2 and x_3
- ▶ If $x_1 = 1$ then clock LFSR 2 and repeat the previous bit of LFSR 3
- ▶ If $x_1 = 0$ then clock LFSR 3 and repeat the previous bit of LFSR 2
- ▶ Output $z = x_2 \oplus x_3$

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

Linear complexity is about $(L_2 + L_3) \cdot 2^{L_1}$

- ▶ Needs L_1 , L_2 and L_3 to be pairwise coprime and roughly the same size

Shrinking Generator

Basic Idea

Take two LFSRs with output x_1 and x_2 .

- ▶ Works by throwing away some of the x_2 stream (shrinks it)

Both LFSRs are clocked at the same time

- ▶ If x_1 is one then output x_2
- ▶ Otherwise discard x_2

Requires $\gcd(L_1, L_2) = 1$

This has a period of $(2^{L_2} - 1) \cdot 2^{L_1 - 1}$

Linear complexity about $L_2 \cdot 2^{L_1}$

A5 Stream Cipher

Used for encryption in GSM

- ▶ Strong and weak versions A5/1 and A5/2

Has three fixed primitive LFSRs of lengths 19, 22 and 23

Key is the initial input of the registers

- ▶ Each LFSR is clocked if certain equations hold between certain bits of the current states
 - ▶ Very strange clocking behaviour
- ▶ Output is xor of all three outputs

A5 Stream Cipher

Attacked by Biryukov, Shamir and Wagner in 2000

Attack requires: Either

- ▶ Two minutes of conversation
- ▶ Then computes key in one second

Or

- ▶ Two seconds of conversation
- ▶ Then computes key in several minutes