In The Name of God

Exercise No.2 Cryptography Course

Mohammadhossein Arsalan 98243005

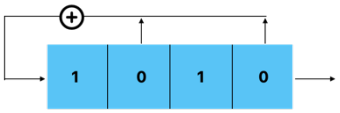
1. Avalanche Effect

In order to answer this question, we need to describe Avalanche effect first; it means that in a block cipher cryptography a little change in the plaintext cause a meaningful difference in the output which is our ciphertext, something like 50% of the ciphertext before the change. In this problem Item1 Y_1 before and after the change (1 bit in plaintext) changed only in 2 bits, $Y_1[0] \neq Y_2[0]$ and $Y_1[4] \neq Y_2[4]$. But in the second item, which is Item2, after the change in plaintext, our output differs in 4 bits $Y_2[2]$, $Y_2[4]$, $Y_2[5]$ and $Y_2[6]$; so, we find out the second item (Item2) satisfies the Avalanche effect.

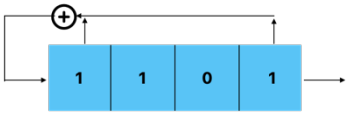
2. LFSR, Alternate Steps

First, we design the LFSR's with the given F(x) for each one of them plus the initialized values stored in them; I explain the way of design only for the LFSR1 and so on.

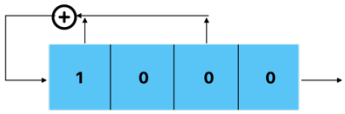
LFSR1 => $F_1(x) = x^4 + x^2 + 1$ => we have a 4-block LFSR because $\mathbf{x^n = x^4 \Rightarrow n = 4}$, it means that first block [from right side] (because of term 1) and 3rd block (because of term x^2) are connected to the XOR stream. Also, at each clock last block's value is result of XOR operation on the block 3 and 1, other blocks will contain the propagated value from the previous clock in the left-side block (values shift right each clock). So, the design will be:



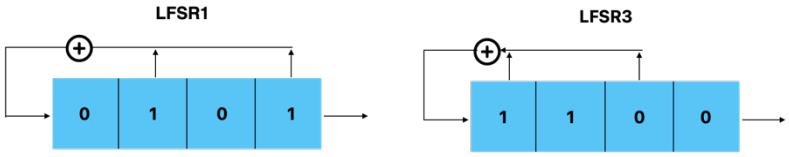
LFSR2 => $F_2(x) = x^4 + x^3 + 1$ => first and last block are connected:



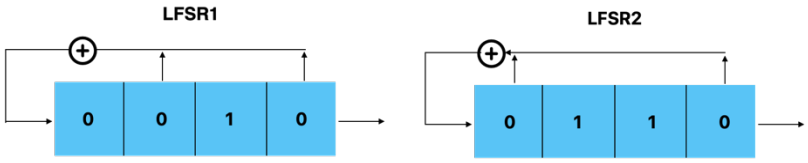
LFSR3 => $F_3(x) = x^4 + x^3 + x$ => we can divide this F(x) to x, so it means that this LFSR will stuck in a loop after a while, also 2nd and last blocks are connected.



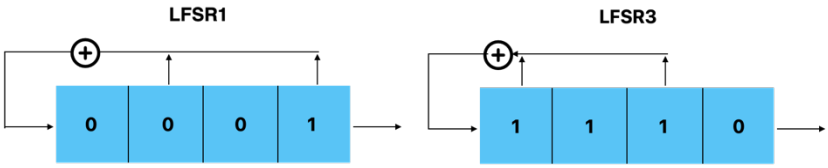
In each clock we check the LFSR1's output (x) and if it's 1 the LFSR2 will run (calculate the XOR and shift executes) instead of LFSR3, and if it's 0 the reverse action considered. Also the main output is result of XOR operation on both LFSR2 and LFSR3's outputs (at least one of these outputs is repeated in each clock because only one of them (LFSR2/LFSR3) performs in each clock. So, at first main output is result of XOR operation on the LFSR2 and LFSR3 outputs (based on the initialized values), which is $(out_0 = (1 \text{ XOR } 0) = 1)$, during this output calculation our LFSR3 performs the shift because LFSR1 output at first is 0, at the next clock (which is first) our LFSR1 and LFSR3 changed like this:



Output in first clock will be result of (1 XOR 0) because LFSR3 output is 0 again, $out_1 = 1$. During this. Clock (first) LFSR1 output is 1 so this time LFSR2 will perform and in 2nd clock our LFSR1 and LFSR2 look like:



Same as previous levels output is 0 because both of outputs in LFSR2 and LFSR3 are 0 now, $out_3 = 0$. During this clock, LFSR1's output is 0 and again LFSR3 performs. So, in the last clock our LFSR1 and LFSR3 looks like:



So, again our output is 0 because LFSR3's output again became 0, $out_3 = 0$.

3. Linear Feedback Shift Register

In order to find a recognition pattern for each function (g_1 and g_2) based on output values, we should create the truth tables for each.

g_1's truth table:

$$g_1 = x_0.x_1.x_2 \oplus x_0 \oplus x_1.x_2 \oplus x_1$$

x_0	x_1	x_2	g_1
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**0**

g_2's truth table:

$$g_2 = x_0 \oplus x_1.x_2 \oplus x_1$$

x_0	x_1	x_2	g_2
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

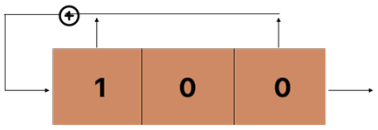
As we see, the only difference between provided functions g_1 and g_2 is the term ($x_0.x_1.x_2$) which is available in g_1 while it doesn't appear in g_2, so we expect difference in output occurs whenever we have all inputs (x_0, x_1 and x_2) set to **1**; so, based on truth table our output based on last input combination is different in g_1 and g_2, so, at last cycle (if we consider our inputs change like the pattern we provided in truth tables and we have 8 cycles of output) we can find out which algorithm is used based on the 8^{th} (which is last) output that we see.

4. Linear Feedback Shift Register

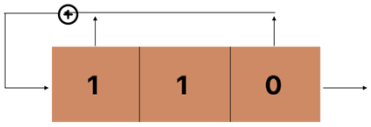
First, we design the LFSR based on the explanations in the Part 2, we create the F(x) for the given LFSR and then design it. Each C_i is the coefficient of x^i in the F(x) equation. Also, we have 3 c_i's which means that our LFSR is based on 3 blocks.

$$F(x) = x^3 + x^2 + 1$$

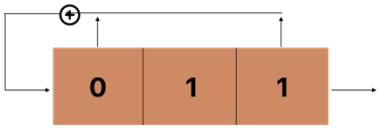
So, based on the c_i's given, we find out that first block (from right side) and last blocks are connected to the XOR stream. Our LFSR looks like:



As you see I entered the initial value given in the first bullet of this question. With this initialized vector we'll have XOR of 1 and 0 in last block and shifted value of 10 to the next blocks. Something like this:



The next bullet changed the initialized vector so we'll have this LFSR arrangement at first:



Same as previous action, we generate the XOR result and shift other values, result is:

The first scenario will arrive at a state of 2nd scenario after 3 clocks (110, 111 and **011**). Also 2nd scenario arrives at the state of 1st scenario after 4 clocks (101, 010, 001 and **100**) so there is a loop in the given LFSR, means that these two initialized states are related to each other and they are in a same sequence.

5. Golomb Sequence

Golomb triple criterions are:

1- Difference of 1's and 0's count in the sequence should have the minimum value (for even bits sequences should be zero, and for odd bits sequences, 1.

Here we have **1010011010011101101** which contains **10** ones and **7** zeroes, difference is 3, So this criterion does not match for this sequence.

2- Possibility of i-length runs to occur should be $\left(\frac{1}{2}\right)^i$, or number of this runs should be equal to (total runs × $\left(\frac{1}{2}\right)^i$), so we count number of occurred runs with each possible length. (Run is a repeated series of 1's or 0's):

1-length: 10100110100111101 => 7 => (7+3+1) × $\left(\frac{1}{2}\right)^1 = 5.5 \neq 7$

2-length: 10100110100111101 => 3 => 11 × $\left(\frac{1}{2}\right)^2$ = 2.75 ≅ 3

4-length: 10100110100111101 => 1 => 11 × $\left(\frac{1}{2}\right)^4$ = ~1 = 1

This criterion does not match too because 1-length runs doesn't obey the rules.

3- Autocorrelation function must have only 2 values for this sequence, in order to know what is condition of seeing these values, we need to describe autocorrelation function formula:

If we rotate and shift right the sequence **k** times, Autocorrelation function of **C(k)** would be:

$$C(k) = \frac{same\ bit\ values - not\ same\ bit\ values}{period\ length}$$

Hence for k=0 both sequences (shifted and original) has no difference, so the head of fraction is equal to number of same bit values, since period length for this example is length of sequence, this C(k) or C(0) is equal to 1. And the other value for C(k) with other k's must be a fixed value. Also, we know that C(k) = C(n-k) which n is length of sequence and it means that we need only n/2 shifts (k values) and C(k) checking to be assured that last criterion is matched or not. So, we check the C(k) for each k in [1,8]:

k = 1 => $\begin{matrix}1010011010011101\\1101001101001110\end{matrix}$ => $C(1) = \frac{7-10}{17} = -\frac{3}{17}$

k = 2 => $\begin{matrix}1010011010011101\\0110100110100111\end{matrix}$ => $C(2) = \frac{7-10}{17} = -\frac{3}{17}$

k = 3 => $\begin{matrix}1010011010011101\\1011010011010011\end{matrix}$ => $C(3) = \frac{11-6}{17} = \frac{5}{17}$ (oops **x**)

So, this criterion does not match too.