[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024

Scribed by: Manas Jitendrakumar Ingle (202151086) — Lecture 1,2 (Week 6)

Stream Cipher

OTP [One Time Padding]

$$C = M \oplus K$$

1. $\boxed{C_1 = M_1 \oplus \text{K}}$ → this is wrong
   $\boxed{C_2 = M_2 \oplus \text{K}}$ → this is wrong

2. len(K) ≥ len(M)

$$F(K, IV) = Z_i \in \{0, 1\}$$

K - Secret key, IV - Initialization Vector

$$Z_0, ..., Z_{n-1} \oplus m_0, ..., m_{n-1} = C_0, ..., C_{n-1}$$

- Z will be a random-looking variable.
-

$$A - - - - - - - - - B$$
$$K - - - - - - - - - K$$
$$F(K, IV) = Z_i - - - - - - - - F(K, IV) = Z_i$$
$$C_i = m_i \oplus Z_i \rightarrow (C_i, IV) \rightarrow C_i \oplus Z_i = m_i$$

- if K is selected randomly and is kept secret then the outputs $\text{Z}_0, ..., \text{Z}_{(n-1)}$ will be indistinguishable from bits string generated by using a random bit generator.
- '$F(K, IV) = \text{Z}_i (0 \leq i \leq n-1)$→Pseudorandom bit generator
- Length of the output bits ≫ the length of K.

$$K \rightarrow Secret, IV \rightarrow Public \tag{1}$$

- If we modify at least one bit of the IV or K then there will be an unpredictable change in the output Z_i

$$F(K, IV_1) = Z_i^1 (0 \leq i \leq n-1)$$
$$F(K, IV_2) = Z_i^2 (0 \leq i \leq n-1)$$

Stream Cipher

Synchronous Stream Cipher

Self-Synchronizing or Asynchronous Stream Cipher

Synchronous Stream Cipher

A Synchronous Stream Cipher is one in which the Keystream is generated independently of the plaintext bits and the ciphertext bits.

State Update Function : $S_{i+1} = \mathbf{f}(S_i, K)$

Keystream Generation Function: $\mathbb{Z}_i = \mathbf{g}(S_i, K)$

Ciphertext Generation Function: $C = \mathbf{h}(\mathbb{Z}_i, M_i)$

Here S_0 is the initial state and may be determined from the secret key K and IV.

Self Synchronizing Stream Cipher

A Self Synchronizing Stream Cipher or Asynchronous Stream Cipher is one in which the Keystream is generated as a function of the key and a fixed number of previous ciphertext bits.

State : $\sigma_i = (C_{i-t}, C_{i-t+1}, ..., C_{i-1})$

Keystream Generation Function: $\mathbb{Z}_i = \mathbf{g}(\sigma_i, K)$

Ciphertext Generation Function: $C = \mathbf{h}(\mathbb{Z}_i, M_i)$

Here $\sigma_0 = (C_{i-t}, C_{i-t+1}, ..., C_{i-1})$ is the non secret initial state.

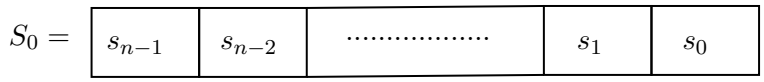
0.1 Linear Feedback Shift Register

This is one of the most used stream cipher and has been used for every communication (voice calls) till 4G. It contains a n-bit register. The states are denoted by S, and the bits are denoted by s. There is a clock and at each clocking number the state of the register updates and the, an output (keystream bit) is generated using which encryption of message can be done.

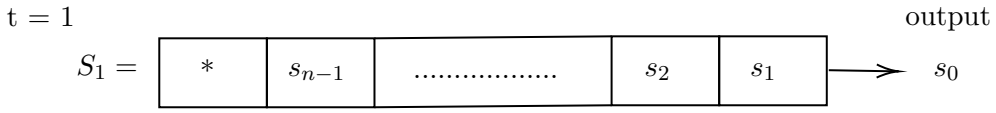
A register of length n means that it is a n-bit LFSR, or equivalently it has a state of length n.

Let us say at clocking number t = 0, the state of the register is S_0. For each $0 \leq i \leq n-1$, $s_i \in \{0, 1\}$.

t = 0, t $\rightarrow$ clocking number



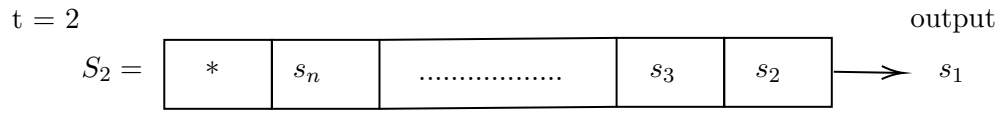
At each clocking number, a shift by one bit takes place (either right or left, depends on the design). Here, we will take right shift for understanding.



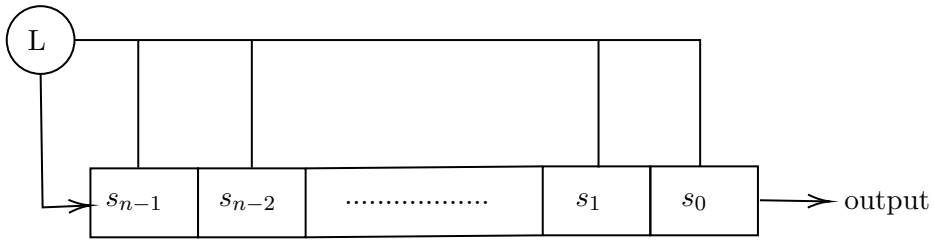
The rightmost bit s_0 moves out and is the output (the keystream bit). However, the leftmost bit s_n becomes empty. s_n is known as feedback bit and is calculated as:

$$s_n = L(s_0, s_1, ..., s_{n-1}) = L(S_0)$$

After one more clocking, the state can be represented as:



Again, the feedback bit $S_{n+1} = L(S_1)$. We will look at the function L in some time. The LFSR with right shift operation can be represented with the following circuit.



LFSR with Right Shift

The function L is a linear function on the bits of the previous state.

$$L : \{0,1\}^n \to \{0,1\}$$
$$L(s_0, s_1, ..., s_{n-1}) = s_n$$

A linear function can be represented as:

$$L_a = a_0 \cdot s_0 \oplus a_1 \cdot s_1 \oplus ..... \oplus a_{n-1} \cdot s_{n-1} \text{ where } a_i \in \{0,1\}$$

Suppose, an arbitrary function L be defined as:

$$L = a_0 \cdot s_0 \oplus a_1 \cdot s_1 \oplus ..... \oplus a_{n-1} \cdot s_{n-1} \oplus a_n \text{ where } a_i \in \{0,1\}$$

In this function, if $a_n = 0$ then $L = L_a$, a linear function. Otherwise, if $a_n = 1$ the $L \neq L_a$. In fact, such a function is known as Affine function.

A function can be proved to be linear using the following property.

$$L(X) \oplus L(Y) = L(X \oplus Y)$$
$$\implies L(X) \oplus L(Y) \oplus L(X \oplus Y) = 0$$

Example: Find if the following functions are linear or not. Solve it considering 2-bit inputs.

1. $L_1(x, y) = x \oplus y$
2. $L_2(x, y) = 1 \oplus x \oplus y$

Solution:

1. Let's compute $L_1(x) \oplus L_1(y) \oplus L_1(x \oplus y)$

$$L_1(x) \oplus L_1(y) \oplus L_1(x \oplus y) = (x_1 \oplus x_2) \oplus (y_1 \oplus y_2) \oplus ((x_1 \oplus y_1) \oplus (x_2 \oplus y_2))$$
$$L_1(x) \oplus L_1(y) \oplus L_1(x \oplus y) = 0$$

   Therefore, L_1 is a linear function.

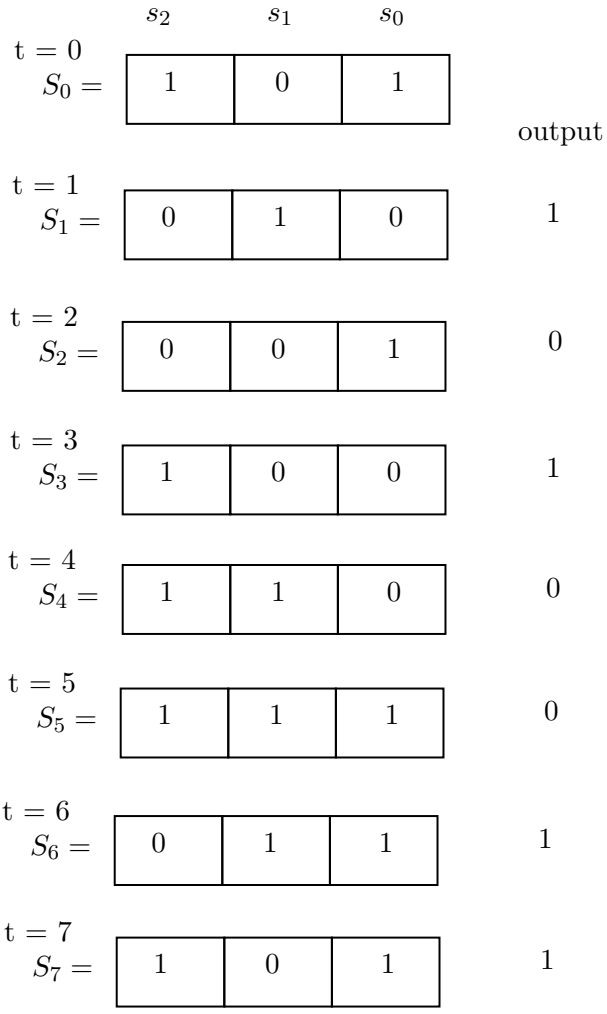
2. $L_2(x) \oplus L_2(y) \oplus L_2(x \oplus y) = (1 \oplus x_1 \oplus x_2) \oplus (1 \oplus y_1 \oplus y_2) \oplus (1 \oplus (x_1 \oplus y_1) \oplus (x_2 \oplus y_2))$

$$L_2(x) \oplus L_2(y) \oplus L_2(x \oplus y) = 1$$

Therefore, L_2 is not a linear function.

LFSR has a linear function, whose output depends on previous state, therefore it provides feedback. There is shift operation on the bits stored in register. Hence, the name is Linear Feedback Shift Register.
Now, let us see an example of a 3-bit LFSR.

$$L = s_0 \oplus s_2$$



The L function for this LFSR is $L = s_0 \oplus s_2$.
In any LFSR, if you have reached the initial state again, the the output bits will be repeated. In the above example it can be seen that from start state t = 0 to the state t = 7, all the non-zero states are obtained.
The output bits in the above example are repeated after $t = 2^3 - 1$ states because at t = 7, the initial state is reached. Hence, maximum output length that can be achieved in this LFSR without repetition is 7. Hence, using LFSR, only $2^n - 1$ maximum number of non-zero states can be generated.
Also, if we take the 0 state i.e. all bits in the register are 0, it will remain in the zero state forever.

Hence, in any LFSR, if input state is 0, then it will remain zero. Hence, LFSR has a fixed point (0-state).

Let us look at another example of a 3-bit LFSR with a different L function. This time the L function is $L = s_0$.

$L = s0$

	s_2	s_1	s_0	output
t = 0, $S_0 =$	1	0	1	
t = 1, $S_1 =$	1	1	0	1
t = 2, $S_2 =$	0	1	1	0
t = 3, $S_3 =$	1	0	1	1

Here, we can see the initial state is reached again only at t = 3. For LFSR, the linear functionis very important in deciding the period after which the bits will repeat.

Period of an LFSR

Consider S_0 to be a non-zero state. S_0 is repeated after m clocking of LFSR, then m will be the period of LFSR. An LFSR with n-bits can have a maximum period of $2^n - 1$.

If there is an LFSR where different non-zero states are repeating at certain different number of clocking. Let us say x_1 number of states repeat after P_1 clocking, x_2 number of states repeat after P_2 clocking and so on x_n number of states repeat after P_N clocking. Here, every non-zero state is present in at least and only one x_i. Therefore, if we take any non-zero state, then it will repeat after certain number of clocking which will belong to the set $\{P_1, P_2, ..., P_N\}$. Therefore, the period of LFSR will be:

$$\text{Period of LFSR} = LCM(P_1, P_2, ..., P_N)$$

Consider an n-bit LFSR,

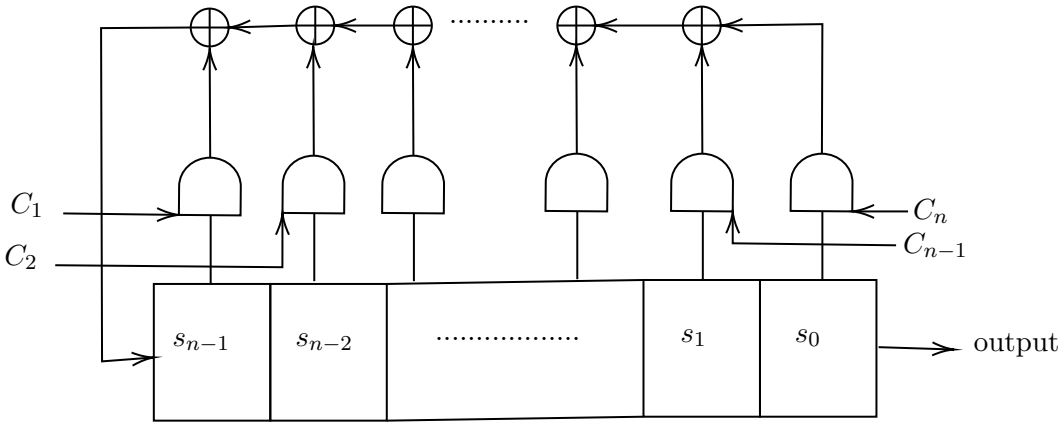
t = 0

$S_0 =$	s_{n-1}	s_{n-2}	.....	s_1	s_0

We know that after clocking, s_n will be,

$$s_n = L(s_0, s_1, ..., s_{n-1})$$
$$s_n = c_1 \cdot s_{n-1} \oplus c_2 \cdot s_{n-2} \oplus ..... \oplus c_n \cdot s_0 \text{ where } c_i \in \{0, 1\}$$

This is known as Algebraic Normal Form of writing s_n. Therefore, we can see that to implement LFSR in hardware, we only need AND and XOR gates. The circuit for LFSR is given below. The value of s_i will be xored or not depends on the value of c_{n-i}.



Corresponding to every LFSR, we have a Linear Feedback Function (L). Corresponding to LFF, we can construct a polynomial $f(x)$.

$$L = c_1 \cdot s_{n-1} \oplus c_2 \cdot s_{n-2} \oplus ..... \oplus c_n \cdot s_0$$
$$f(x) = 1 + c_1 \cdot x + c_2 \cdot x^2 + ... + c_n \cdot x^n$$

The polynomial $f(x)$ is known as connection polynomial of LFSR. If anyone of the linear feedback function or the connection poylnomial is known, the other can be easily constructed. Since, $c_i \in \{0, 1\}$ for $1 \leq i \leq n$, therefore, $f(x) \in F_2[x]$. Therefore,

$$\text{n-bit LFSR} \iff \text{Linear Feedback Function} \iff \text{one polynomial in } F_2[x] \text{ of degree} \leq n$$

We know that is S_0 repeats after $2^n - 1$ clocking, then it is a full period LFSR. Now, consider a connection polynomial of degree n in $F_2[x]$.

1. If the connection polynomial is primitive polynomial, then the LFSR will have full period.

   If we recall, during AES we studied that if $G(x)$ is a primitive polynomial, then $(F_2[x]/\langle G(x)\rangle, +, *)$ is a field where $F_2[x]/\langle G(x)\rangle$ contains all polynomials with degree less than degree of G(x). Similarly, here we have a n degree connection polynomial. If it is primitive, then we can construct all polynomials of degree less than n. That is, we can construct $2^n - 1$ polynomials. Therefore, we can generate all the possible non-zero states of LFSR.

2. If connecting polynomial is irreducible (and not primitive), $F_2[x]/\langle G(x)\rangle$, then the period of LFSR will divide $2^n - 1$.

3. If connecting polynomial is reducible, then different state will have different cycle length (different period).

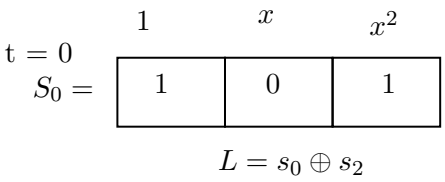
Example: Consider the example of the 3-bit LFSR taken earlier where $L = s_0 \oplus s_2$.
Solution: The connecting polynomial for the LFSR is:

$$f(x) = 1 + x + x^3$$

Now, checking if $f(x)$ is primitive or not. Let's try to generate all possible non-zero polynomial from $f(x)$.

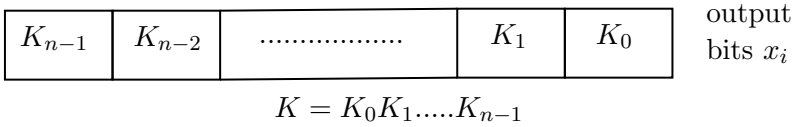
$$\{1, x, x^2, x^3 = x + 1, x^2 + x, x^3 + x^2 = 1 + x + x^2, x^3 + x^2 + x = x^2 + 1, x^3 + x = 1\}$$

Since, all polynomials of degree less than 3 are generated by using $\langle x \rangle$. Hence, $x^3 + x + 1$ is a primitive polynomial. Hence, the given 3-bit LFSR is full periodic.
Another but not so good way to find if LFSR is fully periodic or not is to construct a polynomial g(x) from the initial state and then keep multiplying it by x, again and again until we get g(x) again. g(x) is constructed by taking those powers of x for which the bit is 1 in the initial state. Use the connecting polynomial in case degree of g(x) gets greater than degree of connecting polynomial.



Therefore, the polynomial $g(x) = 1 + x^2$ and connecting polynomial $f(x) = x^3 + x + 1$.
$1 \times g(x) = 1 + x^2$
$x \times g(x) = x + x^3 = 1$
$x^2 \times g(x) = x$
$x^3 \times g(x) = x^2$
$x^4 \times g(x) = x^3 = x + 1$
$x^5 \times g(x) = x^2 + x$
$x^6 \times g(x) = x^3 + x^2 = x^2 + x + 1$
$x^7 \times g(x) = x^3 + x^2 + x = x + 1 + x^2 + x = 1 + x^2 = g(x)$
Since, after multiplication from x^7, g(x) is repeated again and $7 = 2^3 - 1$. Therefore, the given LFSR is fully periodic LFSR.

Let us now talk about the security of LFSR. Usually, we keep the secret key in the memory of LFSR, that is, in the register.



We will keep secret key or some other public parameter in the register and we will generate some bits.

$$\text{output bits } x_i \rightarrow \text{keystream bits } Z_i$$
$$m_i \oplus Z_i = C_i \rightarrow \text{ciphertext bits}$$

Every time we will be clocking the LFSR, we will get one bit as output and will have feedback also. Now, if we consider Known Plaintext Attack Model, that is, we know certain plaintext bits and corresponding ciphertext bits and our aim is to find the secret key. Therefore, we know m_i and c_i. Clearly,

$$Z_i = m_i \oplus C_i$$

Therefore, we know Z_i corresponding to the know m_i and C_i. Now, first output bit according to LFSR id K_0, the second output bit is K_1 and so on. That is, if we know the first n bits of the message and corresponding ciphertext, we can get entire secret key.

Even if we don't know the first n keystream bits, we can form a linear system of equations, because whatever bit we will get at leftmost position during clocking will be a linear function on the initial state only. For example, in the first clocking,

$$K_n = L(K_0, K_1, ..., K_{n-1})$$

and in the second clocking,

$$K_{n+1} = L(K_1, K_2, ..., K_n)$$

but K_n in again a linear combination of initial state only. Eventually, every bit coming to leftmost position is linear function on initial state only. Therfore, even if we know any key stream bits, that is any message bit and corresponding ciphertext bit, then,

$$Z_i = m_i \oplus C_i$$

and Z_i will always be a linear function of $K_0, K_1, ..., K_{n-1}$. If we know first n bits, Z_i is a linear function with only one term (K_0 or K_1 or K_{n-1}). So, if we know Z_i from known plaintext attack for any message bits. It is not required to know consecutive first n-bits. If we know any mesage bits and corresponding ciphertext bits, we can form a system of linear equations and solve that to get the secret key. Hence, simple LFSR does not provide good security.

Let us say you know one bit of message and corresponding ciphertext, that means you have only one linear equation. Let us say some i^{th} bit (that is, not necessarily the first bit). Is is possible to reduce the search complexity of the secret key using that equation. The exhaustive search complexity of recovering the secret key is 2^n because this many keys are possible.

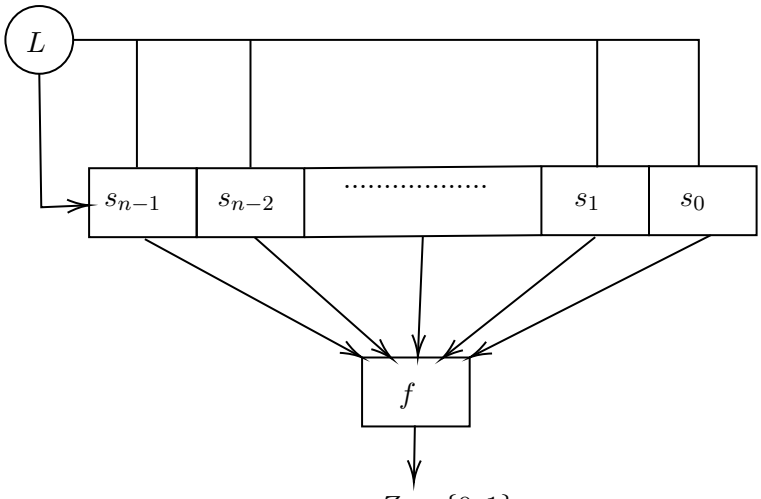
Let us say there is one $Z_i = 0$ and the linear equation formed is,

$$Z_i = L(K_0, K_1, ..., K_{n-1})$$
$$0 = Z_i = K_2 \oplus K_6 \oplus K_{n-2}$$

Using the linear equation, is it possible to reduce the complexity of searching the key? If we have correct guess for say K_6 and K_{n-2}, then K_2 is always determined. The search complexity, thus, is reduced to half.
Therefore, depending upon how many equations you are getting and the property of the system of equations, the search complexity will be reduced definitely. Therefore, knowing only 1 bit creates huge impact on the search complexity.
Say we have one system of linear equations with infinite solutions, so we will get all possible solutions here. So, if we have n variables but we have only few equations (say 5 or 6), still these equations will reduce the search complexity by a good margin. Therefore, using Known Plaintext Attack it is easy to break LFSR.

LFSR with Non-Linear Filter Function

Here, we will consider an l-bit boolean function f which takes l-bits as input and produces one bit as output. We will take l-bits out of the n-bits of the state of LFSR as input to f and use the output of f as $Z_i \in \{0, 1\}$. The function f here is a non-linear function.



$$f : \{0,1\}^l \to \{0,1\}$$
$$n \geq l$$
$$C_i = m_i \oplus Z_i$$

The state update function of LFSR will be same - there will be linear feedback function (L) and shifting as earlier. Therefore, if we select l fixed position from the n positions in the register, at each clocking the value at these position will update. If the function f is good enough, the output will still be random looking. In fact, this will also have full period if f is a good function and the connection polynomial corresponding to L is primitive.

The advantage here is that even if we know m_i and corresponding c_i from Known Plaintext Attack model, and consequently we know Z_i. The Z_i is now a non-linear function of state bits of LFSR. Solving a non-linear system of equation might be computationally difficult.

The state update function of LFSR is, say α. Therefore,

$$S_{t+1} = \alpha(S_t)$$
$$Z_{t+1} = f(S_{t+1})$$

Let us look at the LFSR state at clocking time t.

$$S_t = (s^t_{n-1}, s^t_{n-2}, ..., s^t_0)$$

The state of LFSR at clocking time (t+1) will be,

$$S_{t+1} = (s^{t+1}_{n-1}, s^{t+1}_{n-2}, ..., s^{t+1}_0)$$

Suppose the shifting to be right shift, therefore,

$$s^{t+1}_0 = s^t_1, s^{t+1}_1 = s^t_2, ..., s^{t+1}_{n-2} = s^t_{n-1}, s^{t+1}_{n-1} = L(s^t_{n-1}, s^t_{n-2}, ..., s^t_0)$$

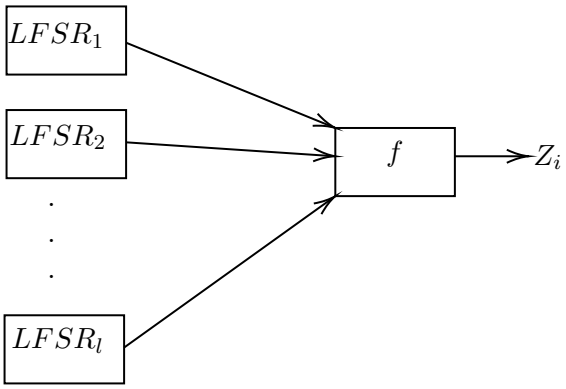
The state update can be represented as a matrix multiplication in the following way,

$$S^{t+1} = \begin{bmatrix} s^{t+1}_0 \\ s^{t+1}_1 \\ \vdots \\ S^{t+1}_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ c_n & c_{n-1} & c_{n-2} & c_{n-3} & \dots & c_1 \end{bmatrix} \begin{bmatrix} s^t_0 \\ s^t_1 \\ \vdots \\ S^t_{n-1} \end{bmatrix}$$

$$L = c_n \cdot s_0 \oplus c_{n-1} \cdot s_1 \oplus \ldots \oplus c_1 \cdot s_{n-1}$$

LFSR with Combiner Function

We have a similar function f as we discussed above. However, here we have l number of LFSR's. The output of these l LFSR's, that is, l-bits becomes the input for the combiner function f whose output is treated as Z_i. The function f is non-linear.



$$C_i = m_i \oplus Z_i$$

Non-Linear Feedback Shift Register

After a few years, it was observed that LFSRs are good ciphers which are still secure but most of the ciphers can be broken using various adverse cryptanalysis techniques.

To prevent all this, there is a concept of stream ciphers known as Non-Linear Feedback Shift Register (NFSR's). From the name, it can be anticipated that the feedback function will be non-linear in NFSR. In fact, in modern scenarios, all stream ciphers are based on NFSR.

In NFSR, the mechanism is similar to LFSR, but the feedback is non-linear.
Note: The problem in NFSR is that if we consider a Non-Linear Feedback Function, there is no methodical proof which can determine the period of NFSR. For LFSR, if the connection polynomial is primitive, then the LFSR is fully periodic. In case of NFSR, we don't have any such proof or method.

Hash Function:

$$h : A \to B$$

$h(x) = Y$

1. If x is altered to $\acute{x}$ then $h(\acute{x})$ will be completely different from $h(x)$
2. Given Y it is practically infeasible to find x such that $h(x) = Y$
3. Given x and $Y = h(x)$ practically infeasible to find $\acute{x}$ such that $h(x) = h(\acute{x})$

- Alice $\longleftrightarrow$ Bob

$$x = E(M, k) \longrightarrow x_1 = Dec(\tilde{x}, k)$$

$$s_1 = h(M, k) \longrightarrow s_2$$

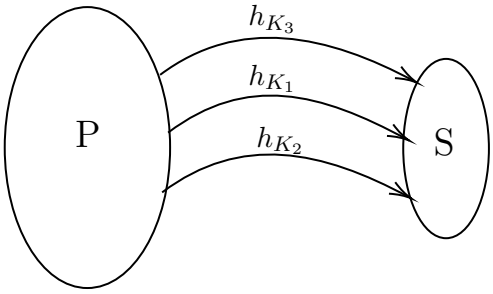
If $h(x, k) = s_2$, then Bob would accept x_1
We will be able to verify:

1. Whether x is altered during communication.
2. Whether s_1 is altered during communication.

- **Definition:**
  A hash family is a four tuple (P, S, K, H) satisfying:

  1. P is the set of all possible messages.
  2. S is the set of all possible message digests or authentication tags.
  3. K is the key space.
  4. For each $K_1 \in K$ there is a hash function $h_{k_1} \in H$ such that $h_{k_1} : P \to S$

  Here $|P| \geq |S|$
  more interestingly $|P| \geq 2x|S|$



Where H : set of all hash function h_{k_i} : hash function

where H is the set of all hash functions and h_{k_i} is the hash function

If the key is involved in the computation of hashed value, then that hash function is known as **keyed hash function**.

If the key is not required to compute the hashed value then that hash function is known as **unkeyed hash function**.

Problems:

- **Problem 1: Pre-Image Finding**
  $h : P \to S$

  - Given $y \in S$, Find $x \in P$ such that $h(x) = y$.

This problem is known as pre-image finding problem.

For an hash function h, if you cannot find pre-image in a feasible time then h is known as pre-image resistant hash function.

- Finding pre-image is computationally hard for pre-image resistant hash function.

- **Problem 2: Second Pre-Image Finding**
  $h : P \to S$

  - Given $x \in P$ and $h(x)$ find $\acute{x} \in P$ such that $\acute{x} \neq x$ and $h(\acute{x}) = h(x)$

  This problem is known as second pre-image finding problem.

  If finding second pre-image is computationally hard for h then, h is known as second pre-image resistant hash function.

- **Problem 3: Collision Finding Problem**
  $h : P \to S$

  - Find $x, \acute{x} \in P$ such that $\acute{x} \neq x$ and $h(x) = h(\acute{x})$.

  This problem is known as collision finding problem.
  For an hash function h if finding collision is computationally hard then h is known as collision resistant hash function.

Ideal Hash Function:

Consider a hash function $h : P \to S$.

h will be called ideal hash function if given $x \in P$ to find $h(x)$ either you have to apply h on x or you have to look into the total corresponding to h(hash table).

Pre-Image Finding Algorithm:

Question: Given $y \in Y$, find $x \in X$ such that $h(x) = y$ $h : X \to Y$

Choose any $x_0 \subseteq x$ such that $|x_0| = Q$ for each $x \in x_0$

Compute $y_x = h(x)$
if $y_x = y$
return x

$x_0 = \{x_1, x_2, \ldots, x_Q\}$
E_i : event $h(x_i) = y; 1 \leq i \leq Q$

$P_r[E_i] = \frac{1}{M}$

$P_r[{E_i}^c] = 1 - \frac{1}{M}$

$P_r[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q]$

$= 1 - P_r[{E_1}^c \cup {E_2}^c \cup {E_3}^c \cup \cdots \cup {E_Q}^c]$

$= 1 - \prod_{i=1}^{Q} P_r[{E_i}^c]$

$= 1 - (1 - \frac{1}{M})^Q$

$= 1 - [1 - \binom{Q}{1}\frac{1}{M} + \binom{Q}{2}\frac{1}{M^2} - \ldots]$

$\simeq 1 - [1 - \binom{Q}{1}\frac{1}{M}]$

$= \frac{Q}{M}$

$P_r[\text{Pre-image finding}] \simeq \frac{Q}{M}$

Complexity of finding pre-image $= O(M)$.