[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Raghav Agiwal (202151124)

Winter 2023-2024
Lecture (Week 6)

1 One Time Padding

One Time Pad (OTP) in cryptography is a method of encrypting plaintext into ciphertext and decrypting ciphertext back into plaintext. It's based on the principle of exclusive OR (XOR) operation between the plaintext and a randomly generated secret key. The key must be as long as the plaintext and should never be reused for encrypting multiple messages As we have discussed earlier, so we know encryption and decryption in OTP are given as:

1.1 Encryption and Decryption

Encryption: Ciphertext (C) is obtained by XORing the plaintext (M) with the secret key (K): $C = M \oplus K$
Decryption: Plaintext (M) is obtained by XORing the ciphertext (C) with the same secret key (K): $M = C \oplus K$
Encryption: $C = M \oplus K$
Decryption: $M = C \oplus K$

1.2 Conditions for Perfect Secrecy

The key (K) must be truly random, never reused, and at least as long as the plaintext.
The length of the key must be greater than or equal to the length of the message.
The key (K) must be uniformly selected from the entire key space.

1.3 OTP's Limitations

- Not practical for most real-world scenarios due to the impracticality of securely sharing and managing keys as long as the messages.
- Vulnerable to key distribution problem and key management issues.

However, OTP is not practical because if we have a method to share the secret key (length equal to the message), then we can share the message using that mechanism. Here, we want to design an algorithm as efficient as OTP but which can be used for practical purposes. Also, it may not provide perfect secrecy, but it is not possible to prove or disprove that it provides perfect secrecy. Let us define a function F as: $F(K, IV) = Z_i$ where $Z_i \in \{0, 1\}$. K is the secret key, IV is the initialization vector (a public parameter). Function F produces n bits $Z_0, Z_1, \ldots, Z_{n-1}$. The function F is efficiently computable.

Plaintext: $m_0, m_1, \ldots, m_{n-1}$
Z: $Z_0, Z_1, \ldots, Z_{n-1}$
Encryption: $C_0 = m_0 \oplus Z_0, C_1 = m_1 \oplus Z_1, \ldots, C_{n-1} = m_{n-1} \oplus Z_{n-1}$

The function F, known as Pseudo Random Bit Generation Function, has certain properties. These properties are listed below:

1. The sequence of outputs, denoted as $Z_0, Z_1, \ldots, Z_{n-1}$, appears random. To illustrate the concept of a random-looking string, consider flipping an unbiased coin n times, assigning 0 for heads and 1 for tails. Let $x_0, x_1, \ldots, x_{n-1}$ represent the resulting string from the coin tosses. Given the string $Z_0, Z_1, \ldots, Z_{n-1}$, it is impossible to distinguish whether it originated from a coin-tossing experiment or from a pseudo-random bit generator with a key and initialization vector (IV) as input. However, reproducing the string $Z_0, Z_1, \ldots, Z_{n-1}$ is feasible by passing the same inputs to the function F. Such a string, which appears random but can be reproduced by feeding the same inputs to F, is termed a random-looking string.

2. When provided with the same input (K, IV) at different instances, the function F will generate the same Z_i.

3. If the key K is randomly chosen and kept confidential, the outputs $Z_0, Z_1, \ldots, Z_{n-1}$ will be indistinguishable from a bit string generated by a truly random bit generator (such as coin tossing).

4. The length of the output bits $Z_0, Z_1, \ldots, Z_{n-1}$, denoted as n, is significantly greater than the length of K. Efficient functions exist that can produce output of length $2^{80} - 1$ for a key of length 80 bits. Leveraging this property, a short key can efficiently encrypt extensive messages. However, if there is repetition in the output bits, differences between corresponding message bits can be calculated, akin to the one-time pad (OTP). Block ciphers like AES and DES would require extensive time to encrypt a message of length $2^{80} - 1$. DES, for instance, would perform approximately 2^{64} encryptions to encrypt the entire message, rendering this technique significantly more efficient.

5. Altering at least one bit of K or IV leads to an unpredictable change in the output Z_i. For instance, $F(K, IV_1) = Z_i^{(1)}$ and $F(K, IV_2) = Z_i^{(2)}$, where $0 \leq i \leq n-1$, and $Z_i^{(1)}$ and $Z_i^{(2)}$ are uncorrelated. This property enables the encryption of two distinct messages using the same key by adjusting the IV.

Hence, the advantages of this technique are that very long messages can be encrypted using a small key and the same key can be used to encrypt different messages.

Advantages:

- Allows for efficient encryption of very long messages using relatively small keys.
- The same key can be reused to encrypt different messages by altering the IV, providing practicality and versatility.

2 Stream Cipher

A stream cipher is a type of encryption algorithm that operates on individual bits or bytes of plaintext using a stream of key bits. It consists of two main components: a pseudo-random bit generator (PRBG) and encryption/decryption techniques.

2.1 Stream Cipher Operations

PRBG: $F(K, ...) = Z_i$ (keystream bits)
Encryption: $C_i = m_i \oplus Z_i$
Decryption: $m_i = C_i \oplus Z_i$

Here, XOR is commonly used for encryption, but other efficient computation methods can also be employed.

2.2 Synchronous Stream Cipher

In a synchronous stream cipher, the keystream is generated independently of plaintext and ciphertext bits. It consists of the following functions:

- **State Update function:** $S_{i+1} = f(S_i, K)$
- **Keystream Generator function:** $Z_i = g(S_i, K)$
- **Ciphertext Generation Function:** $C_i = h(Z_i, m_i)$

Here, S_0 is the initial state, which may be derived from K and an initialization vector (IV). Synchronous stream ciphers offer high-speed encryption and decryption with minimal computational overhead.

2.3 Asynchronous or Self-Synchronizing Stream Ciphers

An asynchronous stream cipher generates keystream bits based on a function of the key and a fixed number of previous ciphertext bits. It involves the following functions:

- **State Update Function:** $\sigma_{i+1} = f(\sigma, K, IV)$ where $\sigma = (C_{i-t}, C_{i-t+1}, ..., C_{i-1})$
- **Keystream Generation Function:** $Z_i = g(\sigma_i, K)$
- **Ciphertext Generation Function:** $C_i = h(Z_i, m_i)$

Here, $\sigma_0 = (C_{-t}, C_{-t+1}, ..., C_{-1})$ represents the non-secret initial state. Asynchronous stream ciphers provide synchronization between the encryption and decryption processes without requiring an explicit synchronization mechanism.

2.4 Linear Feedback Shift Register

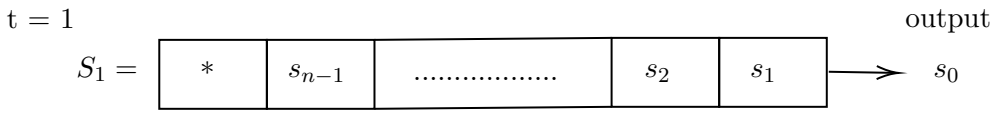
This is one of the most used stream cipher and has been used for every communication (voice calls) till 4G. It contains a n-bit register. The states are denoted by S, and the bits are denoted by s. There is a clock and at each clocking number the state of the register updates and the, an output (keystream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

$S_0 =$

s_{n-1}	s_{n-2}	.....	s_1	s_0

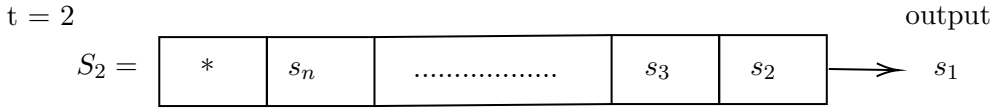
At each clocking number, a shift by one bit takes place (either right or left, depends on the design). Here, we will take right shift for understanding.



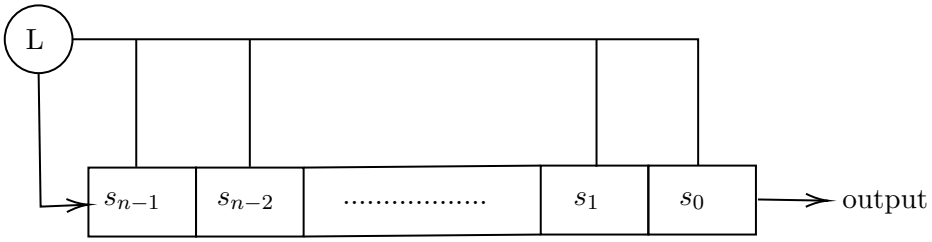
The rightmost bit s_0 moves out and is the output (the keystream bit). However, the leftmost bit s_n becomes empty. s_n is known as feedback bit and is calculated as:

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

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.

$$L = s_0 \oplus s_2$$

	s_2	s_1	s_0	output
t = 0, S_0 =	1	0	1	
t = 1, S_1 =	0	1	0	1
t = 2, S_2 =	0	0	1	0
t = 3, S_3 =	1	0	0	1
t = 4, S_4 =	1	1	0	0
t = 5, S_5 =	1	1	1	0
t = 6, S_6 =	0	1	1	1
t = 7, S_7 =	1	0	1	1

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	
t = 0 $S_0 =$	1	0	1	
				output
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.

2.4.1 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,

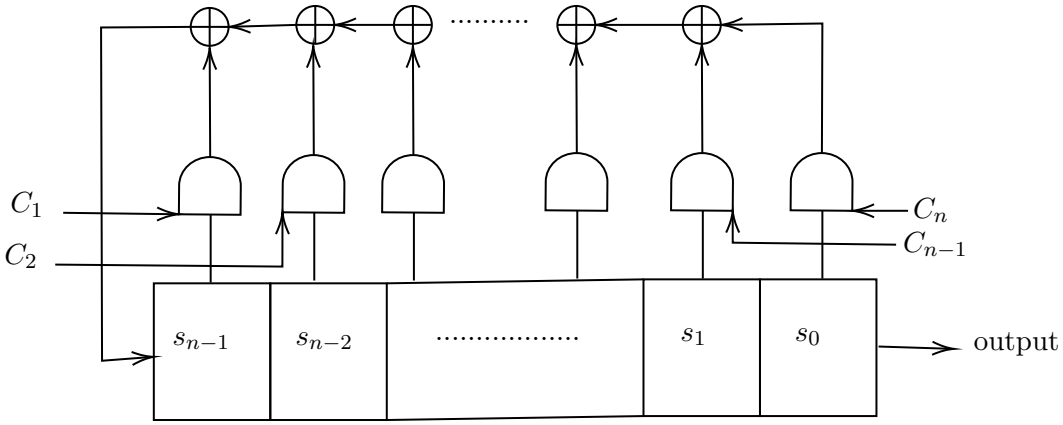
t = 0

$S_0 =$ | 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,

n-bit LFSR $\iff$ Linear Feedback Function $\iff$ one polynomial in $F_2[x]$ 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 states will have different cycle length (different period).

Security of Linear Feedback Shift Register (LFSR)

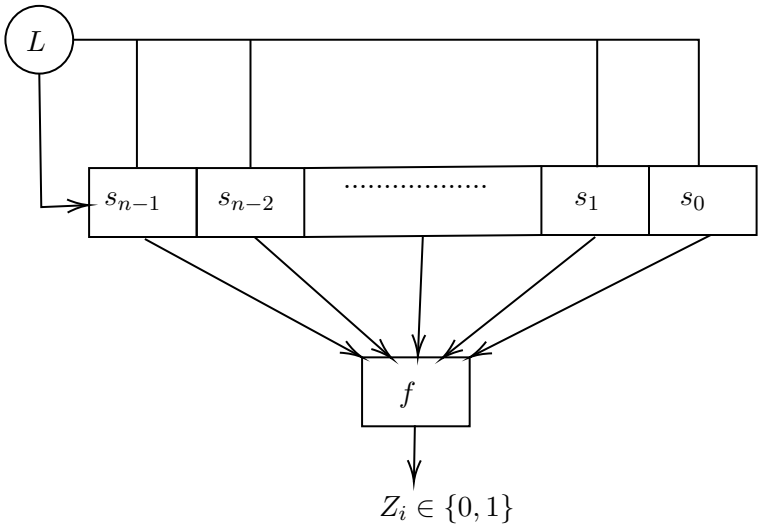
In the context of Linear Feedback Shift Registers (LFSRs), the confidential key is commonly stored within the register's memory. The register comprises n bits, denoted as K, where $K = K_0K_1 \dots K_{n-1}$. Typically, the secret key or other essential parameters are stored in this register, and from it, output bits x_i are generated, serving as keystream bits Z_i. These keystream bits are pivotal in encryption processes, where the relationship $m_i \oplus Z_i = C_i$ denotes the ciphertext bits. Each clock cycle of the LFSR yields one output bit and involves a feedback mechanism. In scenarios like the Known Plaintext Attack Model, where specific plaintext and corresponding ciphertext bits are disclosed, the goal is to ascertain the secret key K. Given m_i and c_i, we can calculate Z_i as:

$$Z_i = m_i \oplus C_i$$

Thus, we have Z_i corresponding to the known m_i and C_i. Since the first output bit according to the LFSR is K_0, the second output bit is K_1, and so on, if we know the first n bits of the message and corresponding ciphertext, we can obtain the entire secret key.

2.4.2 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 mechanism of the LFSR remains unchanged - it still involves a linear feedback function (L) and shifting operations as before. Thus, if we designate l fixed positions out of the n positions in the register, the values at these positions will be updated at each clocking. Assuming the function f is sufficiently robust, the resulting output will continue to exhibit randomness. Furthermore, if f is a good function and the connection polynomial corresponding to L is primitive, this setup can achieve full periodicity.

The benefit of this approach is that even in scenarios where we have knowledge of m_i and their corresponding c_i, obtained through the Known Plaintext Attack model, and subsequently determine Z_i, the resulting Z_i becomes a nonlinear function of the LFSR's state bits.

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.

$$S_t = (s_{n-1}^t, s_{n-2}^t, ..., s_0^t)$$

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

$$S_{t+1} = (s_{n-1}^{t+1}, s_{n-2}^{t+1}, ..., s_0^{t+1})$$

Suppose the shifting to be right shift, therefore,

$$s_0^{t+1} = s_1^t, s_1^{t+1} = s_2^t, ..., s_{n-2}^{t+1} = s_{n-1}^t, s_{n-1}^{t+1} = L(s_{n-1}^t, s_{n-2}^t, ..., s_0^t)$$

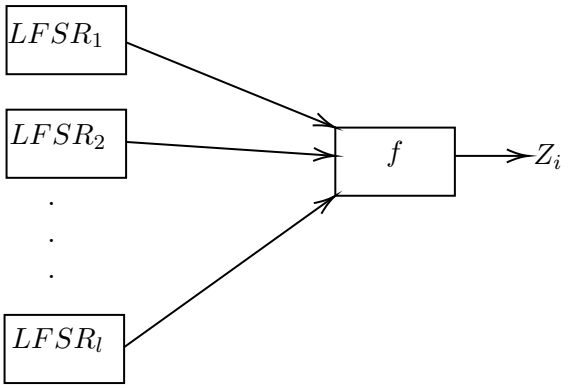
The state update can be represented as a matrix multiplication in the following way,

$$S^{t+1} = \begin{bmatrix} s_0^{t+1} \\ s_1^{t+1} \\ \vdots \\ S_{n-1}^{t+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ c_n & c_{n-1} & c_{n-2} & c_{n-3} & \dots & c_1 \end{bmatrix} \begin{bmatrix} s_0^t \\ s_1^t \\ \vdots \\ S_{n-1}^t \end{bmatrix}$$

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

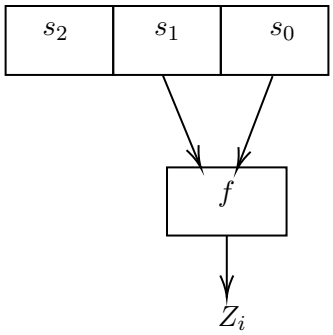
2.4.3 LFSR with Combiner Function

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.



$$C_i = m_i \oplus Z_i$$

Example: Consider the following 3-LFSR with non-liner filter function f.



$$f : \{0,1\}^2 \to \{0,1\}$$
$$f(x_0, x_1) = x_0 \cdot x_1$$

Here, if we draw the truth table of f, it will look like,

x_0	x_1	f
0	0	0
0	1	0
1	0	0
1	1	1

We can see that $Pr[Z = 0] = \frac{3}{4}$. Here, f is not a good function because $Pr[Z = 0] > Pr[Z = 1]$, that is, it is highly biased. We need to design f such that the output is unpredictable. Here, we can predict that output will be zero with a higher probability. If we have a stream cipher with this model and we are able to find out the f function. We can eventually break the stream cipher. Therefore, f must be selected carefully.

2.5 Non-Linear Feedback Shift Register

After a period of time, it became evident that while Linear Feedback Shift Registers (LFSRs) were effective ciphers, many encryption methods could still be compromised through various cryptanalysis techniques. To address this vulnerability, a new concept emerged in stream ciphers known as Non-Linear Feedback Shift Registers (NFSRs). As the name implies, NFSRs incorporate a non-linear feedback function, distinguishing them from LFSRs. In contemporary cryptography, NFSRs serve as the foundation for most stream ciphers.

Unlike LFSRs, where the feedback function is linear, NFSRs introduce non-linearity into the feedback mechanism. However, one challenge with NFSRs is the absence of a systematic method for determining their period. In LFSRs, if the connection polynomial is primitive, the register achieves full periodicity. Conversely, in NFSRs, no such proof or methodology exists to determine their period.

3 Hash Function

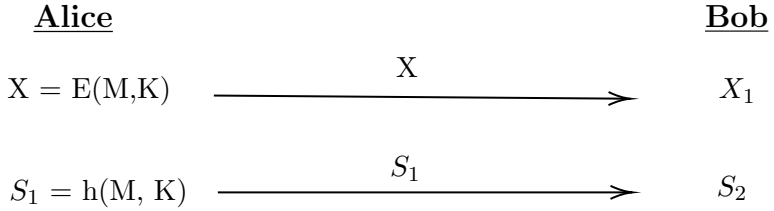
It is a mapping from one set to another set.

$$h : A \to B$$
$$h(X) = Y$$

Every hash function has following properties:

1. If X is altered to X', then h(X') will be completely different from h(X).
2. Given Y, it is practically infeasible to find X s.t. h(X) = Y.
3. Given X and Y = h(X), it is practically infeasible to find X' s.t. h(X) = h(X').

Let us consider this scenario where Alice is encrypting some message using symmetric key and sending it to Bob.



Now, we want to verify that the message is coming from Alice and it has not been altered. Suppose if Bob receives the altered cipher text $\tilde{X}$. Then on decryption using key K, Bob will not get the message M. So we need to ensure that the cipher text is coming from Alice and has not been altered. For that we use hash function.

If h(X_1, K) = S_2, then Bob accepts X_1.

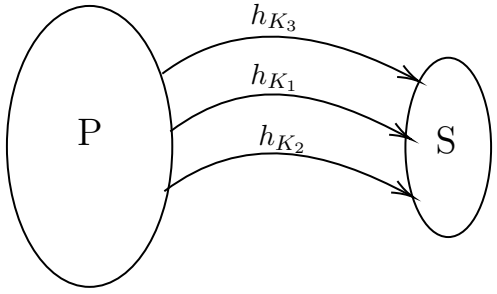
Since for hash function, if the input is changed even slightly, there is huge change in output. So if X_1 would have been altered even on a single bit, the output will not satisfy. Also, since we know input X_1 and the output of h(M, K) = S_2, still because of the properties of hash function, we still cannot determine message m and it is secure. This is known as the Message Authentication Code. We are able to verify :

- Whether X is altered during communication
- Whether S_1 is altered during communication

Definition

A hash family is defined as a four-tuple (P, S, K, H) where the following conditions hold:

1. P represents the set of all possible messages.
2. S denotes the set of all possible message digests or authentication tags (all output).
3. K signifies the key space.
4. For each $K_1 \in K$, there exists a hash function h_{K_1} such that $h_{K_1} : P \to S$, where $|P| \geq |S|$, and notably, $|P| \geq 2 \times |S|$.



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

3.1 Types of Hash Function

- **Keyed Hash function :** In these hash functions, key is involved in the computation of hash value.
- **Unkeyed Hash function :** In these hash functions, key is not required to compute the hashed value.

3.2 Problems

Problem 1:

$$\text{h} : P \to S$$

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

This problem is known as the 'preimage finding problem'. For a hash function, h if you cannot find preimage in feasible time, then h is known as preimage resistant hash function. It is computationally difficult to find preimage for such functions.The preimage finding problem involves finding a specific input x in the domain P such that its hash value $h(x)$ equals a given value y in the codomain S. Formally, given $y \in S$, the objective is to find $x \in P$ such that $h(x) = y$. This challenge is significant in hash function analysis, as it determines the resistance of the hash function against preimage attacks. A hash function is considered preimage resistant if it is computationally infeasible to find a preimage x for a given hash value y within a reasonable amount of time.

Problem 2:

$$\text{h} : P \to S$$

Given $x \in p$ and h(x), find $x' \in P$ such that $x' \neq x$ and $h(x') = h(x)$.

This problem is known as 'Second preimage finding problem'. If finding second preimage is computationally hard for h, then it is known as second preimage resistant hash function.The second preimage finding problem entails finding a distinct input x' in the domain P such that its hash value $h(x')$ matches the hash value $h(x)$ of a given input x. Specifically, for a hash function $h : P \to S$, given an input x and its hash value $h(x)$, the goal is to find another input x' such that $x' \neq x$ and $h(x') = h(x)$. If it is computationally challenging to find such a second preimage for a given input, then the hash function is considered second preimage resistant.

Problem 3:

$$\text{h} : P \to S$$

Find x, $x' \in P$ such that $x \neq x'$ and $h(x) = h(x')$

Additionally, in hash function analysis, another critical problem arises, known as the collision finding problem. This issue pertains to the identification of two distinct inputs x and x' in the domain P that produce the same hash value $h(x) = h(x')$. A hash function h is considered collision-resistant if finding such collisions is computationally challenging.

3.3 Ideal Hash Function

An ideal hash function, denoted as $h : P \to S$, is characterized by a unique property. It ensures that given an input $x \in P$, obtaining its hash value $h(x)$ can only be achieved through two methods: directly applying the hash function to x or consulting the hash table where $h(x)$ is stored. This exclusive characteristic defines the ideal nature of the hash function.

3.4 Pre image finding algorithm

$$\text{h} : X \to Y$$

$|Y| = M$
Choose any $X_o \subseteq X$ such that $|X_o| = Q$

for each $x \in X_o$

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

The chance of getting pre-image depends on the selection of X_o.

Let us find the probability of finding pre-image using X_o

$$X_o = \{x_1, x_2, \ldots, x_Q\}$$
$$E_i : even th(x_i) = y; 1 \leq i \leq Q$$

h(x) can have M values, out of which only one will give success. So,

$$Pr[E_i] = \tfrac{1}{M}$$
$$Pr[E_i{}'] = 1 - \tfrac{1}{M}$$

Now we accumulate the probabilities of $E_1, E_2 \ldots E_Q$

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] = 1 - Pr[E_1{}' \cap E_2{}' \cap E_3{}' \cap \cdots \cap E_Q{}']$$

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] = 1 - \textstyle\prod_{i=1}^{Q} Pr[E_i{}']$$

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] = 1 - (1 - \tfrac{1}{M})^Q$$

Let us expand it now.

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] = 1 - [1 - (\tbinom{Q}{1}\tfrac{1}{M} + \tbinom{Q}{2}\tfrac{1}{M^2} \ldots]$$

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] \approx 1 - [1 - (\tbinom{Q}{1}\tfrac{1}{M}]$$

$$Pr[E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_Q] \approx \tfrac{Q}{M}$$

Therefore,

$$\text{Pr[pre image finding]} = \tfrac{Q}{M}$$

$$\text{Complexity(pre image finding)} = \text{O(M)}$$