[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 6)

1 Pseudo Random Bit Generator

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

Where K is the secret key and IV is the initialization vector. Function F produces n bits $Z_0, Z_1, ..., Z_{n-1}$.

Plaintext: $m_0, m_1, ..., m_{n-1}$

Z: $Z_0, Z_1, ..., Z_{n-1}$

Encryption:

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

This function has the following properties:

1. The output $Z_0, Z_1, ..., Z_{n-1}$ is a random-looking string. A "random-looking string" behaves like a sequence of random outcomes, such as coin tosses. Given such a string, it's impossible to discern whether it was produced by genuine randomness or by a pseudo-random generator with specific inputs. However, this string can be replicated precisely by applying the same inputs to a function, indicating its pseudo-random nature.

2. If we provide the same input (K, IV) to F at different times, it will produce the same Z_i.

$$\begin{tikzcd}
\underline{\text{A}} & \underline{\text{B}} \\
\text{K} & \text{K} \\
F(K, IV) = Z_i & F(K, IV) = Z_i \\
C_i = m_i \oplus Z_i \arrow[r, "{C_i, IV}"] & C_i \oplus Z_i = m_i
\end{tikzcd}$$

3. If a randomly selected secret key K is used, then the outputs $Z_0, Z_1, ..., Z_{n-1}$ will be indistinguishable from a bit string generated by a random bit generator (coin tossing).

4. $F(K, IV) = Z_i$, $0 \leq i \leq n$, the length of the output will be much greater than the length of K. There exist efficient functions capable of producing an output of length $2^{80} - 1$ for a key of length 80 bits. Leveraging this property, a relatively small key can encrypt exceedingly large messages.

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

$$F(K, IV_1) = Z_i^{(1)}, \quad 0 \le i \le n-1$$

$$F(K, IV_2) = Z_i^{(2)}, \quad 0 \le i \le n-1$$

$Z_i^{(1)}$ and $Z_i^{(2)}$ are uncorrelated.

$$\begin{tikzcd}
\underline{\text{A}} & & \underline{\text{B}} \\
\text{K} & & \text{K} \\
F(K, IV_1) = Z_i^{(1)} & & F(K, IV_1) = Z_i^{(1)} \\
C_i^{(1)} = m_i^{(1)} \oplus Z_i^{(1)} \arrow[rr, "{C_i^{(1)}, IV_1}"] & & C_i^{(1)} \oplus Z_i^{(1)} = m_i^{(1)} \\
F(K, IV_2) = Z_i^{(2)} & & F(K, IV_2) = Z_i^{(2)} \\
C_i^{(2)} = m_i^{(2)} \oplus Z_i^{(2)} \arrow[rr, "{C_i^{(2)}, IV_2}"] & & C_i^{(2)} \oplus Z_i^{(2)} = m_i^{(2)}
\end{tikzcd}$$

This property allows for the encryption of two distinct messages using the same key. By varying the initialization vector (IV), different Z_i values can be generated while utilizing the same key .

2 Stream Ciphers

$$\begin{tikzcd}
& \boxed{\text{Stream Cipher}} \arrow[dl] \arrow[dr] & \\
\boxed{\text{Synchronous Stream Cipher}} & & \boxed{\text{Self-Synchronizing Stream Cipher}}
\end{tikzcd}$$

2.1 Synchronous Stream Cipher

A synchronous stream cipher is one in which the key stream is generated independently of the plaintext bits and the ciphertext bits.

It has the following functions:

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

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

2.2 Self-Synchronizing Stream Cipher

A self-synchronizing stream cipher is one in which the keystream bits are generated as a function of the key and a fixed number of previous ciphertext bits.

It has the following functions:

- $\sigma_i = (C_{i-t}, C_{i-t+1}, ..., C_{i-1})$
- 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: $C_i = h(Z_i, m_i)$

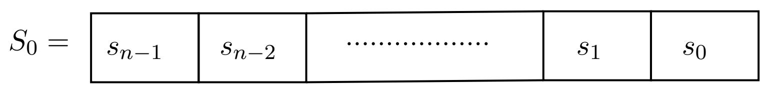
Here, $\sigma_0 = (C_{-t}, C_{-t+1}, ..., C_{-1})$ is the non-secret initial state.

3 Linear Feedback Shift Register (LFSR)

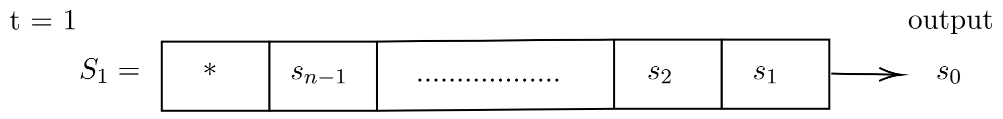
This setup comprises an n-bit register, with states represented by S and individual bits by s. With each clock cycle, the register undergoes an update, generating an output (keystream bit) that can be used for encrypting messages.

A register of length n implies it is an n-bit Linear Feedback Shift Register (LFSR), indicating it has a state of length n. At clock cycle $t = 0$, the register's state is labeled as S_0, where each bit s_i for $0 \leq i \leq n-1$ can take values from the set $\{0, 1\}$.

t = 0, t $\rightarrow$ clocking number

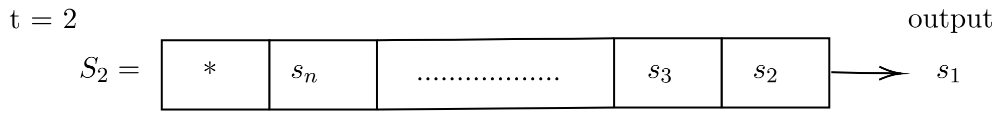


At each clocking number, a right shift by one bit takes place:

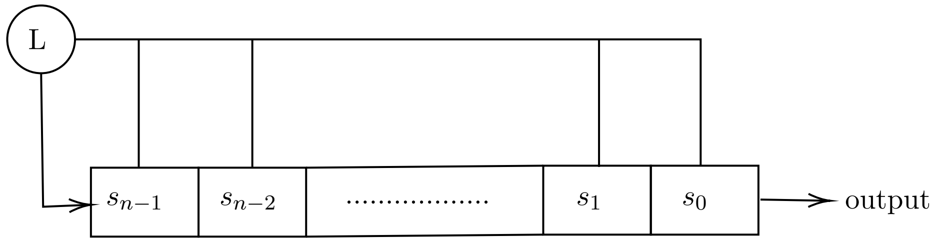


The rightmost bit s_0 serves as the output (the keystream bit), while the leftmost bit s_n becomes empty. The leftmost bit, s_n, is referred to as the feedback bit, and its value is determined by a function L applied to the entire state S_0, calculated as:

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



The feedback bit $s_{n+1} = L(S_1)$.



LFSR with Right Shift

The function L is a linear function on the bits of the previous state. $L : \{0,1\}^n \to \{0,1\}$, $L(s_0, s_1, \ldots, s_{n-1}) = s_n$.

A linear function can be represented as:

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

Suppose, an arbitrary function L be defined as:

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

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

Function's Linearity

A function's linearity can be demonstrated by the property:

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

which implies that $L(X) \oplus L(Y) \oplus L(X \oplus Y)$ equals 0.

Example 01:

Check if the functions are linear or not. Solve it considering 2-bit inputs.

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

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.

$L_2(x) \oplus L_2(y) \oplus L_2(x \oplus y) = (1 \oplus x_1 \oplus x_2) \oplus (1 \oplus y_1 \oplus y_2) \oplus (1 \oplus (x_1 \oplus y_1) \oplus (x_2 \oplus y_2))$
$L_2(x) \oplus L_2(y) \oplus L_2(x \oplus y) = 1$
Therefore, L_2 is not a linear function.

Example 02:

Consider a 3 bit LFSR. $L = s_0 \oplus s_2$

$$L = s_0 \oplus s_2$$

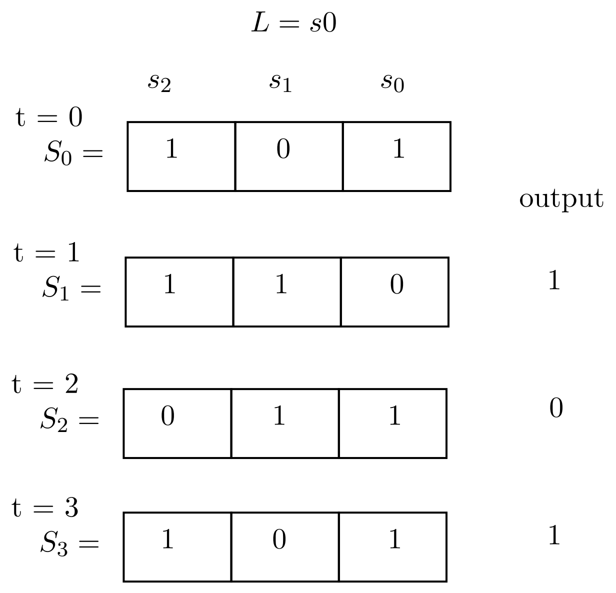
		s_2	s_1	s_0	output
t = 0	$S_0 =$	1	0	1	
t = 1	$S_1 =$	0	1	0	1
t = 2	$S_2 =$	0	0	1	0
t = 3	$S_3 =$	1	0	0	1
t = 4	$S_4 =$	1	1	0	0
t = 5	$S_5 =$	1	1	1	0
t = 6	$S_6 =$	0	1	1	1
t = 7	$S_7 =$	1	0	1	1

In the given example, we observe that from the initial state at time $t = 0$ to the state at $t = 7$, all non-zero states are generated sequentially. Subsequently, after $t = 7$, the output bits begin to repeat as the initial state is reached again. Therefore, the maximum achievable output length in this Linear Feedback Shift Register (LFSR) without repetition is 7. As a result, when employing an LFSR, the maximum number of non-zero states that can be generated is 2^{n-1}, where n represents the number of bits in the register.

Additionally, if the initial state is the all-zero state, signifying that all bits in the register are 0, it will persist in this zero state indefinitely. Consequently, in any LFSR, if the input state is 0, it will remain as zero.

Example 03:

Consider a 3 bit LFSR with $L = s_0$



In this example, we can see that the initial state is reached again at $t = 3$.

3.1 Period of an LFSR

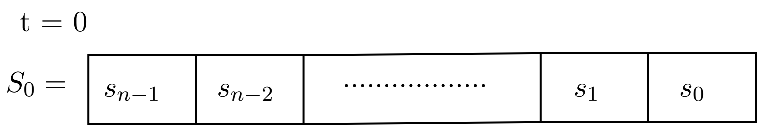
If S_0 represents a non-zero state and S_0 recurs after m clock cycles of the LFSR, then m defines the period of the LFSR. An LFSR comprising n bits can achieve a maximum period of $2^n - 1$.

Consider an LFSR where different non-zero states recur after varying numbers of clock cycles, labeled as x_1 states repeating after P_1 clock cycles, x_2 states repeating after P_2 clock cycles, and so on until x_n states repeating after P_n clock cycles. Here, each non-zero state is uniquely represented by one of the x_i's. Consequently, any non-zero state will reappear after a certain number of clock cycles, belonging to the set $\{P_1, P_2, \ldots, P_n\}$.

Hence, the period of the LFSR, defining the time it takes for the sequence to repeat, can be calculated by determining the least common multiple (LCM) of $P_1, P_2, \ldots, P_n$. Thus, the period of the LFSR can be expressed as:

$$\textbf{Period of LFSR} = \textbf{LCM}(P_1, P_2, \ldots, P_n)$$

Consider a n-bit LFSR:

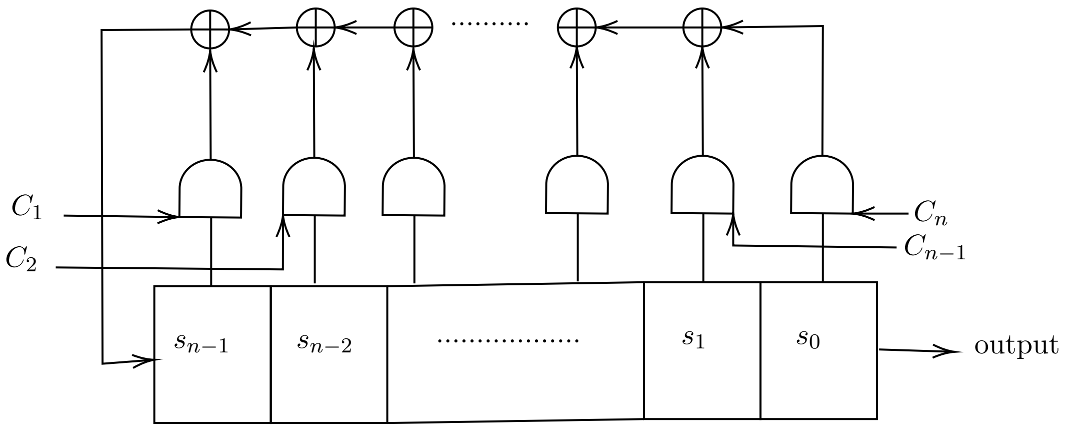


After 1 clocking, s_n will be:

$$s_n = L(s_0, s_1, \ldots, s_{n-1})$$

$$s_n = c_1 \cdot s_{n-1} \oplus c_2 \cdot s_{n-2} \oplus \cdots \oplus c_n \cdot s_0 \quad \text{where } c_i \in \{0, 1\}$$

To implement LFSR in hardware, we only need AND and XOR gates as shown 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 (LFF). Corresponding to LFF, we can construct a polynomial $f(x)$.

$$L = c_1 \cdot s_{n-1} \oplus c_2 \cdot s_{n-2} \oplus \cdots \oplus c_n \cdot s_0$$

$$f(x) = 1 + c_1 \cdot x + c_2 \cdot x^2 + \cdots + c_n \cdot x^n$$

The polynomial $f(x)$ is known as the connection polynomial of LFSR.

If any one of the linear feedback function or the connection polynomial is known, the other can be easily constructed. Since $c_i \in \{0, 1\}$ for $1 \leq i \leq n$, therefore, $f(x) \in \mathbb{F}_2[x]$. Therefore,

n-bit LFSR $\iff$ Linear Feedback Function $\iff$ one polynomial in $\mathbb{F}_2[x]$ of degree $\leq n$

FULL PERIOD LFSR:

If S_0 repeats after 2^{n-1} clock cycles, then it constitutes a full period LFSR. Now, let's consider a connection polynomial of degree n in $\mathbb{F}_2[x]$.

1. When the connection polynomial is primitive, the LFSR achieves its maximum period. In the context of AES, we learned that if $G(x)$ represents a primitive polynomial, then $(\mathbb{F}_2[x]/\langle G(x)\rangle, +, \times)$ forms a field containing all polynomials with degrees lower than that of $G(x)$. Similarly, in our scenario with an n-degree connection polynomial, if it is primitive, we can generate all polynomials with degrees less than n, resulting in the ability to construct $2^n - 1$ polynomials. Consequently, all possible non-zero states of the LFSR can be generated.

2. If connecting polynomial is irreducible (and not primitive), $\mathbb{F}_2[x]/\langle G(x)\rangle$, 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).

3.2 Known Plaintext Attack on n-bit LFSR

$K = (K_0, K_1, \ldots, K_{N-1})$

Output bits $x_i \to$ key stream bits Z_i

$M_i \oplus Z_i = C_i \to$ Ciphertext bits

$Z_i = m_i \oplus C_i$, $0 \leq i \leq n-1$

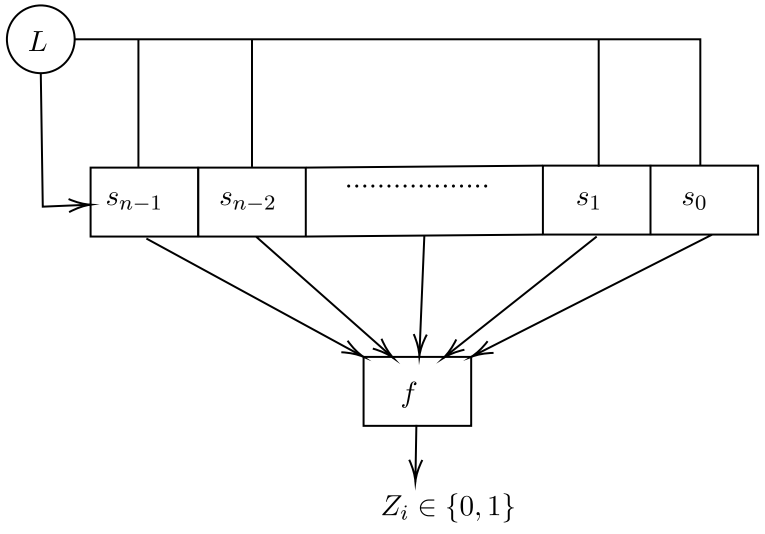
$Z_0, Z_1, \ldots, Z_{n-1}$

$Z_0 = K_0, Z_1 = K_1, \ldots, Z_{n-1} = K_{n-1}$

If I give you keystream bits then you will be able to prepare a system of linear equations. By solving the system of linear equations, you will get back the state.

3.3 LFSR with Nonlinear Filter Function

Here, we are analyzing an l-bit boolean function, represented as f, which takes l bits as input and produces a single bit as output. From the n bits composing the state of the LFSR, we will choose l bits to act as input for f, using the resulting output of f as $Z_i \in \{0, 1\}$. It's crucial to emphasize that the function f in this scenario is nonlinear.



$Z_i \in \{0, 1\}$

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

3.3.1 State Update Function:

The state update function of the LFSR remains unchanged. It still involves a linear feedback function (L) and shifting, as previously described.

The advantage here is that even if we have m_i and the corresponding C_i from the Known Plaintext

Attack model, and thereby know Z_i, Z_i becomes a non-linear function of the LFSR's state bits. Solving a non-linear system of equations can present substantial computational hurdles.

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,

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

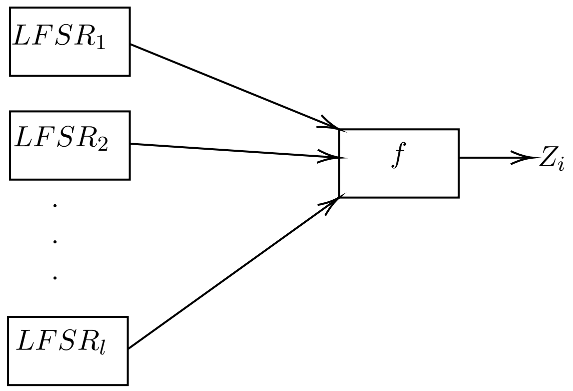
The state update can be represented as a matrix multiplication in the following way,

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

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

3.4 LFSR With Combiner Function

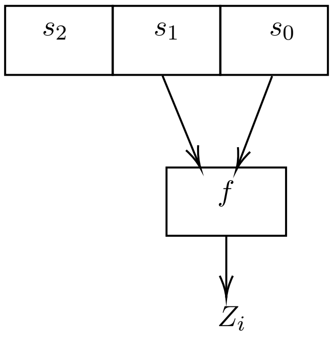
We utilize a function f resembling the one described earlier. However, in this scenario, we incorporate l Linear Feedback Shift Registers (LFSRs). The outputs of these l LFSRs, forming l bits, are inputted into the combiner function f, resulting in an output denoted as Z_i. It's important to emphasize that function f remains nonlinear in this context.



$$C_i = m_i \oplus Z_i$$

Example:

Consider the following 3-LFSR with nonlinear filter function f.



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

Solution:

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

Noting that $Pr[Z = 0] = \frac{3}{4}$, we conclude that f does not exhibit ideal behavior since $Pr[Z = 0]$ exceeds $Pr[Z = 1]$, indicating a notable bias. It's crucial to devise f in a manner that guarantees unpredictability in the output. Currently, predicting an output of zero is more likely. If such a model were employed in a stream cipher and we managed to ascertain the function f, it would expose the stream cipher to vulnerabilities. Therefore, the selection of f must be approached cautiously to mitigate such risks.

3.5 Non-Linear Feedback Shift Register (NFSR)

In NFSR, the mechanism is similar to LFSR, but the feedback is non-linear.

$$f : \{0,1\}^l \to \{0,1\}$$

4 Hash Function

A hash function is a mapping from one set to another set with certain properties.

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

4.1 Properties of Hash Function

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 such that $h(X) = Y$.
3. Given X and $Y = h(X)$, it is practically infeasible to find X' such that $h(X) = h(X')$.

Consider a scenario

$$\begin{tikzcd}
\underline{\textbf{Alice}} & & \underline{\textbf{Bob}} \\
\text{X = E(M,K)} \arrow[rr, "X"] & & X_1 \\
S_1 = \text{h(M, K)} \arrow[rr, "S_1"] & & S_2
\end{tikzcd}$$

If $h(X_1, K) = S_2$, then Bob will accept X_1. We are able to check whether X Is altered during communication.

4.2 Formal definition of Hash Function

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

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

4.3 Types of Hash Function

$$\begin{tikzcd}
 & \textbf{TYPES OF HASH FUNCTION} \arrow[dl] \arrow[dr] & \\
\textbf{Keyed Hash function} \arrow[d] & & \textbf{Unkeyed Hash function} \arrow[d] \\
\text{If a key is involved in the computation of the hash value} & & \text{If a key is not involved in the computation of the hash value}
\end{tikzcd}$$

4.4 Essential Problems

1. **Pre-Image Finding Problem:** Given a hash function $h : P \rightarrow S$ and $y \in S$, find $x \in P$ such that $h(x) = y$.

2. **Second Pre-Image Finding Problem:** Given a hash function $h : P \rightarrow S$, $x \in P$, and $h(x)$, find $x' \in P$ such that $x' \neq x$ and $h(x') = h(x)$.

3. **Collision Finding Problem:** Given a hash function h, find $x, x' \in P$ such that $x \neq x'$ and $h(x) = h(x')$.

Ideal Hash Function:

An ideal hash function $h : P \rightarrow S$ will be called ideal if given $x \in P$, to find $h(x)$, either we have to apply h on x or we have to look into the table corresponding to h (hash table).

4.5 Algorithms for the formulated problems

4.5.1 Pre-Image Finding Algorithm

Given: $y \in Y$

Find: $x \in X$ such that $h(x) = y$

```
**Result:** Pre-image $x$
$h : X \rightarrow Y$
Choose any $X_0 \subseteq X$ such that $|X_0| = Q$ for each $x \in X_0$
Compute $y_x = h(x)$
**if** $y_x = y$ **then**
    Return $x$
**else**
    Continue
**end**
```

Pr [the above algorithm returns correct pre-image] - gives you the complexity.

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

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

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

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

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

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

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

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

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,

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

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