[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Chitranshi Srivastava (202051055) — Lecture 11 and 12 (Week 7)

In previous week, we have studied the AES-128 algorithm. We discussed about the round function of AES - SubBytes, Shift Rows and Mix Column. We learnt about the Subbyte function in detail. In this week, we will study the rest of AES. After completing, AES, we will discuss Hash Functions in detail.

1 Advanced Encryption Standard

1.1 Round Function of AES-128

The Subbyte function was already discussed in the previous week.

1.1.1 Shift Rows

Shift Row function is a mapping from 128-bit to 128-bit. It takes a 4×4 matrix as input (the output of Subbyte function). It performs left circular shift on the elements of i^{th} row by i positions, where row index begins from 0.

$$\text{Shift Rows: } \{0,1\}^{128} \to \{0,1\}^{128}$$
$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \xrightarrow{ShiftRows} \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{11} & S_{12} & S_{13} & S_{10} \\ S_{22} & S_{23} & S_{20} & S_{21} \\ S_{33} & S_{30} & S_{31} & S_{32} \end{bmatrix}$$

1.1.2 Mix Columns

Mix columns, again, is a mapping from 128-bit to 128-bit. It also takes a 4×4 matrix as input (the output of Shift Rows function).

$$\text{Mix Columns: } \{0,1\}^{128} \to \{0,1\}^{128}$$
$$(S_{ij})_{4\times 4} \xrightarrow{MixColumns} (S'_{ij})_{4\times 4}$$

Consider the column $c \in 0,1,2,3$ of matrix S.

$$\text{column} = \begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$$

The Mix Columns function is defined as follows. For i = 0 to i = 3, let t_i be the polynomial constructed from S_{ic}. Define four polynomials as:

$$u_0 = [(x * t_0) + (x+1) * t_1 + t_2 + t_3] \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$u_1 = [t_0 + (x * t_1) + (x+1) * t_2 + t_3] \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$u_2 = [t_0 + t_1 + (x * t_2) + (x+1) * t_3] \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$u_3 = [(x+1) * t_0 + t_1 + t_2 + (x * t_3)] \bmod (x^8 + x^4 + x^3 + x + 1)$$

Now, $S^{'}_{ij}$ is the binary 8-bits constructed using u_i. Therefore,

$$\begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix} \xrightarrow{MixColumns} \begin{bmatrix} S^{'}_{0c} \\ S^{'}_{1c} \\ S^{'}_{2c} \\ S^{'}_{3c} \end{bmatrix}$$

Applying Mix Columns to each columns, will give us the entire $(S^{'}_{ij})_{4\times4}$ matrix. Therefore, Mix Column can be defined as a matrix multiplication as:

$$(S^{'}_{ij})_{4\times4} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \times \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \bmod (x^8+x^4+x^3+x+1)$$

In terms of hexadecimal, the above multiplication can be represented as:

$$(S^{'}_{ij})_{4\times4} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \bmod (x^8+x^4+x^3+x+1)$$

The polynomial $(x^8+x^4+x^3+x+1)$ is a primitive polynomial, hence, it is possible to construct the inverse of the Mix Columns function.

Example: Find $\begin{bmatrix} S^{'}_{00} \\ S^{'}_{10} \\ S^{'}_{20} \\ S^{'}_{30} \end{bmatrix}$ after doing the Mix Column operation on $\begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$ where $S_{00} = 95, S_{10} = 65, S_{20} = fd, S_{30} = f3$.

Solution:

$$\begin{gathered} S_{00} = 95 = 10010101 = x^7+x^4+x^2+1 \\ S_{10} = 65 = 01100101 = x^6+x^5+x^2+1 \\ S_{20} = fd = 11111101 = x^7+x^6+x^5+x^4+x^3+x^2+1 \\ S_{30} = f3 = 11110011 = x^7+x^6+x^5+x^4+x+1 \end{gathered}$$

Now, let's calculate $S^{'}_{00}$.

$S^{'}_{00} = (x * S_{00} + (x+1) * S_{10} + S_{20} + S_{30}) \bmod (x^8+x^4+x^3+x+1)$

$S^{'}_{00} = ((x^8+x^5+x^3+x) + (x^7+x^5+x^3+x^2+x+1) + S_{20} + S_{30}) \bmod (x^8+x^4+x^3+x+1)$

$S^{'}_{00} = (x^8+x^7+x^3+x+1) \bmod (x^8+x^4+x^3+x+1)$

$S^{'}_{00} = (x^7+x^4) = 10010000 = (90)_{16}$

Similarly we can compute $S^{'}_{10}, S^{'}_{20}$ and $S^{'}_{30}$. Let's calculate each one of them.

$S^{'}_{10} = (S_{00} + x * S_{10} + (x+1) * S_{20} + S_{30}) \bmod (x^8+x^4+x^3+x+1)$

$S_{10}^{'} = (S_{00} + (x^7 + x^6 + x^3 + x) + (x^8 + x^2 + x + 1) + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{10}^{'} = (x^8 + x^7 + x^5 + x^3 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{10}^{'} = (x^7 + x^5 + x^4) = 10110000 = (b0)_{16}$

Similarly,

$S_{20}^{'} = (S_{00} + S_{10} + x * S_{20} + (x + 1) * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{20}^{'} = (S_{00} + S_{10} + (x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x) + (x^8 + x^4 + x^2 + 1)) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{20}^{'} = (x^4 + x^3 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{20}^{'} = (x^4 + x^3 + x^2 + x + 1) = 00011111 = (1f)_{16}$

Similarly,

$S_{30}^{'} = ((x + 1) * S_{00} + S_{10} + S_{20} + x * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = ((x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) + S_{10} + S_{20} + (x^8 + x^7 + x^6 + x^5 + x^2 + x)) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = (x^7 + x^6 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = (x^7 + x^6 + 1) = 11000001 = (c1)_{16}$

Therefore, we have,

$$\begin{bmatrix} S_{00}^{'} \\ S_{10}^{'} \\ S_{20}^{'} \\ S_{30}^{'} \end{bmatrix} = \begin{bmatrix} 90 \\ b0 \\ 1f \\ c1 \end{bmatrix}$$

1.2 Key Scheduling Algorithm of AES 128

The key scheduling algorithm of AES-128 takes 128-bit key as input and generate 11 round keys of 128-bits each.

$$\text{key} = (key[0], key[1], ..., key[15]) \text{ where each } key[i] \text{ is 1 byte long}$$

We will generate 44 words (32-bit) - w[0], w[1],...., w[43]. We can see that (32 * 44)/128 = 11. Therefore, we can generate 11 round keys from these 44 words.

There are two functions involved in key scheduling algorithm. These are given below:

1. ROTWORD($B_0B_1B_2B_3$): It takes a word as input and performs left circular shift of its bytes once (or left circular shift by 8 bits). Here, B_0, B_1, B_2, B_3 are bytes of the input word. The output of the ROTWORD function is:

$$\text{ROTWORD}(B_0B_1B_2B_3) = B_1B_2B_3B_0$$

2. SUBWORD($B_0B_1B_2B_3$): It takes a word as input and performs the Subbyte function (defined in round function of AES) of its bytes B_0, B_1, B_2, B_3. The output of SUBWORD function is:

$$\text{SUBWORD}(B_0B_1B_2B_3) = B_0^{'}B_1^{'}B_2^{'}B_3^{'}$$
$$\text{where } B_i^{'} = Subbytes(B_i)\ \forall\ i \in \{0,1,2,3\}$$

Also, there are ten round constants (which are words, i.e, 32-bit) defined as:

$$\begin{aligned} RCON[1] &= 01000000 \\ RCON[2] &= 02000000 \\ RCON[3] &= 04000000 \\ RCON[4] &= 08000000 \\ RCON[5] &= 10000000 \\ RCON[6] &= 20000000 \\ RCON[7] &= 40000000 \\ RCON[8] &= 80000000 \\ RCON[9] &= 1b000000 \\ RCON[10] &= 36000000 \end{aligned}$$

Note that the values of constants written above are in hexadecimal. The 44 words are generated using the below algorithm.

```
for $i \leftarrow 0$ to 3 do
    $w_i \leftarrow key[4i]||key[4i+1]||key[4i+1]||key[4i+3]$;
end
for $i \leftarrow 4$ to 43 do
    temp = w[i-1];
    if $i\%4 == 0$ then
        temp = SUBWORD(ROTWORD(temp)) $\oplus$ RCON[i/4];
    end
    w[i] = w[i-1] $\oplus$ temp
end
```

Now, the round keys will be given as:

$$\begin{aligned} K_1 &= w[0]||w[1]||w[2]||w[3] \\ K_2 &= w[4]||w[5]||w[6]||w[7] \\ &. \\ &. \\ K_{11} &= w[40]||w[41]||w[42]||w[43] \end{aligned}$$

During decryption, we don't need the inverse of key scheduling algorithm as round keys are xored only. Hence, the same round keys will work during decryption.

1.3 Inverse of Round Function

As AES is based on Substitution Permutation Network (SPN) and for SPN we need inverse of round functions during decryption. Hence, we need inverse of round function for AES. The inverse of round function for round 1 to 9 is defined as:

$$f_i^{-1} = Subbyte^{-1}(ShiftRows^{-1}(MixColumns^{-1}(S))) \;\forall\; i \in \{1, 2, .., 9\}$$

For the 10th round function, the inverse is defined as:

$$f_{10}^{-1} = Subbyte^{-1}(ShiftRows^{-1}(S))$$

1.3.1 Inverse Subbyte Function

During computation of Subbyte of a byte, say A, we took the 4 MSB bits as row number and 4 LSB bits as column number. Then, we looked up the value in the subbyte table. That is,

$$A = X||Y$$
$$Subbyte(A) = B = Subbyte\_Table[X][Y]$$

Now, to compute inverse subbyte of B, we will find out the location of B in the table. Suppose, it is in row number X' and column number Y'. Then,

$$InverseSubbyte(B) = X'||Y'$$

1.3.2 Inverse Shift Row Function

Inverse Shift Row function is defined as right circular shifting the elements of the i_{th} row of the input matrix by i positions. Note that $i \in \{0, 1, 2, 3\}$.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \xrightarrow{InverseShiftRows} \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{13} & S_{10} & S_{11} & S_{12} \\ S_{22} & S_{23} & S_{20} & S_{21} \\ S_{31} & S_{32} & S_{33} & S_{30} \end{bmatrix}$$

1.3.3 Inverse Mix Column Function

If we consider the Mix Column function as matrix multiplication (stated earlier), it can be written as:

$$MixColumns(S) = S' = M * S$$

It has been proven that,

$$MixColumns(MixColumns(MixColumns(MixColumns(S)))) = I$$
$$M^4 * S = I$$

where I is identity matrix. Therefore, inverse of M is M^3. Hence,

$$InverseMixColumns(S') = MixColumns(MixColumns(MixColumns(S'))) = S$$

1.4 Mode of Operation

We know that we can encrypt 128 bit blocks using AES at once. Suppose we want to encrypt more data, let's say 256 bit data. We need some mechanism to encrypt this data. There are certain mode of operations which we will be discussing in encrypting more data than the block length. Few examples of mode of operation are mentioned below.

1. Electronic CodeBook Mode (EBC)
2. Cipher FeedBack Mode (CFB)
3. Cipher Block Chaining Mode (CBC)
4. Output FeedBack Mode (OFB)
5. Counter Mode
6. Count with Cipher Block Chaining Mode (CCM)

We will be discussing the EBC and CBC mode in detail.

Let's say we have 256-bit of data that is to be encrypted using AES. We can first encrypt the first 128-bit block and then we can encrypt the second 128-bit block. Then, we can concatenate both the encrypted texts to get the ciphertext for the 256-bit data. Now, suppose there is an image of a human face (say of 200kB). We can follow the same approach that we discussed above to encrypt the image. However, there is a problem with this approach. There will be certain exactly identical parts in the image such as the right eye and the left eye. Let us consider the right eye is a 128-bit block and left eye is a 128-bit block. Since, these blocks are exactly similar, their corresponding ciphertext will also be exactly similar. Therefore, we can say that these two components of ciphertext, and hence plaintext, are same. That means, there is leakage of information. This mode of operation is known as Electronic CodeBook Mode (ECB). Therefore, in this mode we can get some patterns about plaintext from the ciphertext.

1.4.1 Electronic CodeBook Mode

As discussed above, in this mode, the plaintext is divided into continous blocks of l-bit size and each block is encrypted separately. The corresponding ciphertexts are concatenated in the same order to get the final ciphertext.

$$M = m_0||m_1||.....||m_t \text{ (plaintext)}$$
$$len(m_i) = l - bit \text{ (each } m_i \text{ is a block, for AES, l = 128)}$$

Encryption:

$$C = C_0||C_1||.....||C_t$$
$$C_i = Enc(m_i, K) \ \forall \ i \in \{0, 1, ..., t\}$$

Decryption:

$$M = m_0||m_1||.....||m_t$$
$$m_i = Dec(C_i, K) \ \forall \ i \in \{0, 1, ..., t\}$$

The advantages of using ECB mode is that multiple blocks can be encrypted in parallel. However, it will reveal information if few blocks are same, i.e., if $m_i = m_j$ then $C_i = C_j$.

1.4.2 Cipher Block Chaining Mode

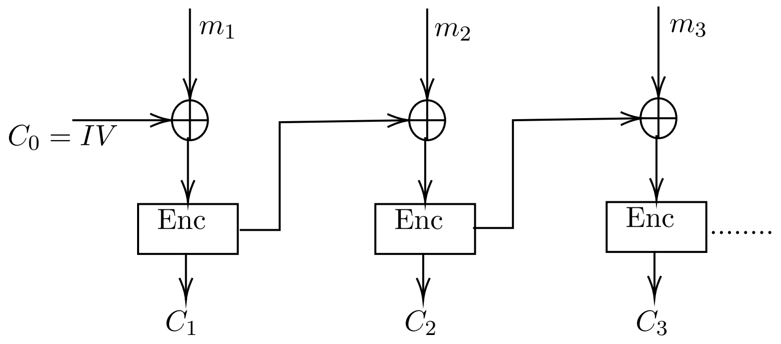
It is the most used mode of operation. It is used in applications like WhatsApp. There is a block of l-bit which is public. It is known as Initialization Vector (IV).

Encryption:

$$M = m_1||m_2||.....||m_t \text{ (plaintext)}$$
$$len(m_i) = l - bit \text{ (each } m_i \text{ is a block, for AES, l = 128)}$$

The encrypted text in CBC mode contains (n+1) blocks for a plaintext of n blocks. The ciphertext is computed as:

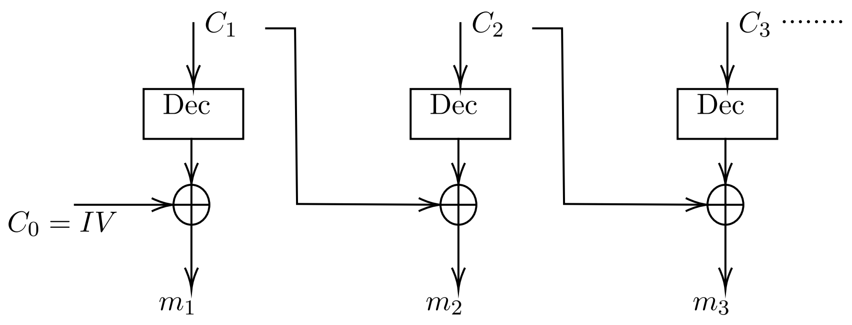
$$C_0 = IV$$
$$C_i = Enc(C_{i-1} \oplus m_i, K) \ \forall \ i \in \{1, 2, 3...., t\}$$
$$C = C_0||C_1||.....||C_t$$



Encryption in CBC Mode

Decryption:

$$m_i = Dec(C_i, K) \oplus C_{i-1} \ \forall \ i \in \{1, 2..., t\}$$
$$\text{where } C_0 = IV$$
$$M = m_1||m_2||.....||m_t$$



Decryption in CBC Mode

2 Hash Function

It is a mapping from one set to another set.

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

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.

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

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 :

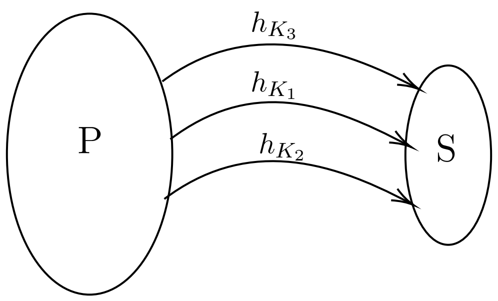
- Whether X is altered during communication
- Whether S_1 is altered during communication

2.1 Definition

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 $K_1 \in k$, there is a hash function h_{K_1} such that

$$h_{K_1} : P \rightarrow S$$
$$\text{where } |P| \geq |S|$$
$$\text{more interestingly } |P| \geq 2 \times |S|$$



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

2.2 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.

2.3 Problems

Problem 1:

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

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.

Problem 2:

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

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.

Problem 3:

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

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

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

2.4 Ideal Hash Function

Let $h : P \to S$ be a hash function.
h will be called ideal hash function if given $x \in P$ to find h(x) or either you have to apply h on x or you have to look into the table corresponding to h(hash table). If there are only these two ways, then it is ideal hash function.

2.5 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

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

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 $E_1, E_2 \dots E_Q$

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

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

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

Let us expand it now.

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

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

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

Therefore,

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

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