[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Archit Agrawal (202051213)

Winter 2022-2023
Lecture 11 and 12(Week 7)

1 Quick Recap

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

2 Round Function of AES-128

The Subbyte function was already discussed in the previous week. We will begin the discussion with the Shift Row function.

2.1 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.

$$\text{Shift Rows: } \{0,1\}^{128} \to \{0,1\}^{128}$$

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

2.2 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:

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

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.

$$S'_{10} = (S_{00} + x * S_{10} + (x+1) * S_{20} + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{10} = (S_{00} + (x^7 + x^6 + x^3 + x) + (x^8 + x^2 + x + 1) + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{10} = (x^8 + x^7 + x^5 + x^3 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{10} = (x^7 + x^5 + x^4) = 10110000 = (b0)_{16}$$

$$S'_{20} = (S_{00} + S_{10} + x * S_{20} + (x+1) * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{20} = (S_{00} + S_{10} + (x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x) + (x^8 + x^4 + x^2 + 1)) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{20} = (x^4 + x^3 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{20} = (x^4 + x^3 + x^2 + x + 1) = 00011111 = (1f)_{16}$$

$$S'_{30} = ((x+1) * S_{00} + S_{10} + S_{20} + x * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{30} = ((x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) + S_{10} + S_{20} + (x^8 + x^7 + x^6 + x^5 + x^2 + x)) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{30} = (x^7 + x^6 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$$
$$S'_{30} = (x^7 + x^6 + 1) = 11000001 = (c1)_{16}$$

Therefore, we have,

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

3 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) which are denoted by 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:

$$\text{SUBWORD}(B_0B_1B_2B_3) = B'_0B'_1B'_2B'_3$$
$$\text{where } B'_i = Subbytes(B_i) \ \forall \ i \in \{0, 1, 2, 3\}$$

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

$$RCON[1] = 01000000$$
$$RCON[2] = 02000000$$
$$RCON[3] = 04000000$$
$$RCON[4] = 08000000$$
$$RCON[5] = 10000000$$
$$RCON[6] = 20000000$$
$$RCON[7] = 40000000$$
$$RCON[8] = 80000000$$
$$RCON[9] = 1b000000$$
$$RCON[10] = 36000000$$

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**
  $w_i \leftarrow key[4i]||key[4i+1]||key[4i+2]||key[4i+3]$;
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-4] $\oplus$ temp
end

Now, the round keys will be given as:

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

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.

4 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 10^{th} round function, the inverse is defined as:

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

4.1 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,

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

4.2 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 $i \in \{0, 1, 2, 3\}$.

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

4.3 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,

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

5 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.

5.1 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 $C_i = C_j$.

5.2 Cipher Block Chaining Mode

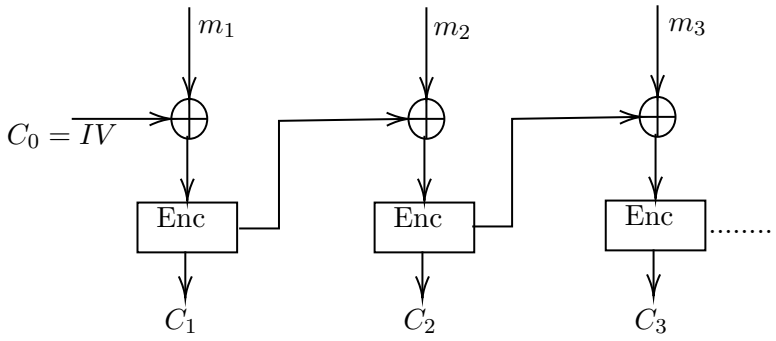
It is the most used mode of operation. It is used in applications like WhatsApp. There is a block of l-bit which is public. It is known as Initialization Vector (IV).

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:

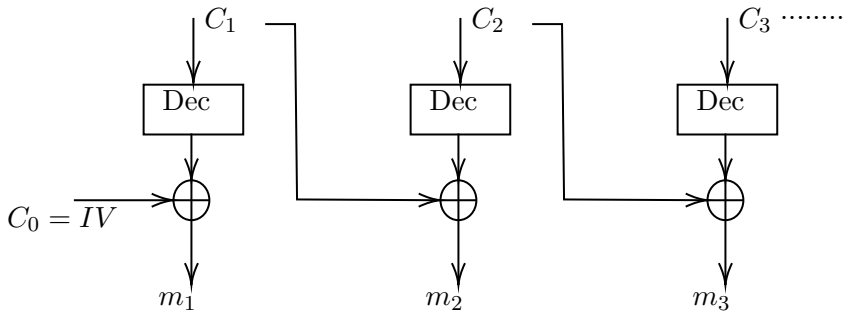
$$C_0 = IV$$
$$C_i = Enc(C_{i-1} \oplus m_i, K) \ \forall \ i \in \{1, 2, 3...., t\}$$
$$C = C_0||C_1||.....||C_t$$



Encryption in CBC Mode

Decryption:

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



Decryption in CBC Mode

6 Hash Function

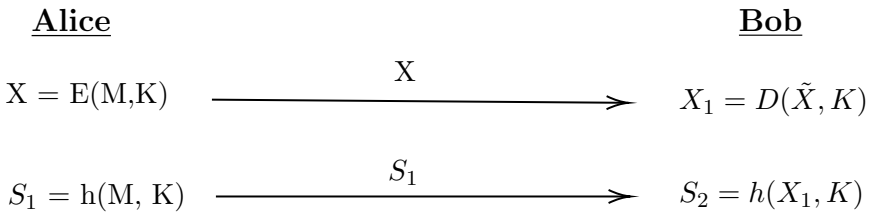
A hash function is nothing but a mapping from one set to another set with certain properties.

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

Every hash function must have the 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 such that $h(X) = Y$.
3. Given X and $Y = h(X)$, it is practically infeasible to find X' such that $h(X) = h(X')$.

Let us consider the scenario wherein Alice is encrypting a message using some symmetric key encryption algorithm and sending it to Bob.



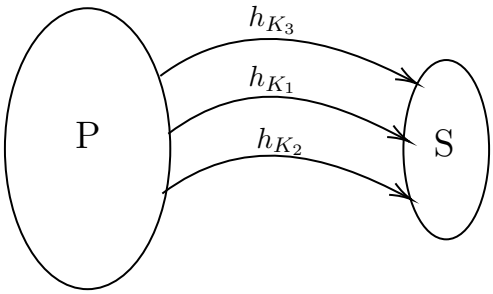
Now, we want to verify that the message is coming from Alice and it has not been altered. Suppose if Bob receives the altered cipher text $\tilde{X}$. On decrypting the message using the key K, Bob get X_1. Now, if $\tilde{X}$ is altered somehow, then on decryption, $X_1 \neq M$. But Bob has no mechanism to ensure if X_1 was the actual message that Alice sent, or the message received by Bob is actually from Alice and not from an attacker. Now, suppose Alice has also sent $S_1 = h(M, K)$ along with X. Bob decrypts $\tilde{X}$ and gets the output as X_1. Now, Bob can verify if X_1 is the message sent by Alice by computing $h(X_1, K)$. If Bob will receive the same message sent from Alice, X_1 will be equal to M and hence, $S_2 = h(X_1, K)$ will be equal to $S_1 = h(M, k)$. Hence, Bob can verify if he has received the same message as was sent from Alice. However, there is a possibility that Bob has received the same message but has received different S_1. In this case also, $h(X_1, K) \neq S_2$. In any case when $h(X_1, K) \neq S_2$, Bob will discard the message. This is known as the Message Authentication Code.

6.1 Definition of a Hash Function

Let us define a hash function more formally. 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 $|P| \geq |S|$
more interestingly $|P| \geq 2 \times |S|$



where H is the set of all hash functions and h_{k_i} is a hash function belonging to H.

6.2 Types of Hash Functions

- **Keyed Hash function:** If key is involved in the computation of hash value, then that hash function is known as Keyed Hash Function.
- **Unkeyed Hash function:** If key is not involved in the computation of hash value, then that hash function is known as Unkeyed Hash Function.

6.3 Problems

1. **Pre-Image Finding Problem:** Given a hash function $h : P \to S$ and $y \in S$, find $x \in P$ such that $h(x) = y$.
   For an hash function h, if finding pre-image in a feasible time is not possible, then h is known as **pre-image resistant hash function**.
2. **Second Pre-Image Finding Problem:** Given a hash function $h : P \to S$, $x \in P$ and $h(x)$, find $x' \in P$ such that $x' \neq x$ and $h(x') = h(x)$.
   For an hash function h, if finding second pre-image in a feasible time is not possible, then h is known as **second pre-image resistant hash function**.
3. **Collision Finding Problem:** Given a hash function h, find $x, x' \in P$ such that $x \neq x'$ and $h(x) = h(x')$.
   For an hash function h, if finding collision in a feasible time is not possible, then h is known as **collision resistant hash function**.

6.4 Ideal and Non-Ideal Hash Functions

Let $h : P \to S$ be a hash function. h will be called an **ideal hash function** 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). If there is a way to compute $h(x)$ other than mentioned above, then h is a **non-ideal hash function**.

6.5 Algorithms for Pre-Image, Second Pre-Image and Collision Finding Problem

6.5.1 Pre-Image Finding Algorithm

The most naive solution to solve this problem will be to compute $h(x)$ for each $x \in X$ for a hash function $h : X \to Y$. This has a complexity of $O(|X|)$. However, we can do better for finding the pre-image. Infact, we can find the pre-image in $O(M)$ where $M = |Y|$. Let us discuss how we can do this.

Let us choose $X_0 \subseteq X$ such that $|X_0| = Q$. Now, we can compute $h(x)$ for each $x \in X_0$ and return x iff $h(x) = y$ (y is given). Therefore, the chance of getting the pre-image depends on the selection of X_0.
Let us find the probability of finding the pre-image in X_0.

$$X_0 = \{x_1, x_2, \ldots, x_Q\}$$
$$E_i : \text{Event } h(x_i) = y; 1 \leq i \leq Q$$

Since, $h(x)$ can have M different values, out of which, only one (y) is required for success of event E_i. Therefore,

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

Therefore, probability that we will find the pre-image in X_0 is:

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

Since, the hash value of $x_i \in X_0$ does not depend on the computation of hash value of some other $x_j \in X_0$, we can say that E_i and E_j are independent events for each $1 \leq i, j \leq Q$. Therefore,

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

We can expand this using binomial expansion,

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

The terms with M^2, M^3... can be ignored as value of M is generally very large. Therefore,

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

Therefore,

$$\text{Pr[pre image finding]} \approx \frac{Q}{M}$$

This means, if we choose M different $x \in X$, then there is a very low probability that we will not find the pre-image of y. Hence, the complexity of finding the pre-image is reduced to $O(M)$.