[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 1)

1 Cryptology

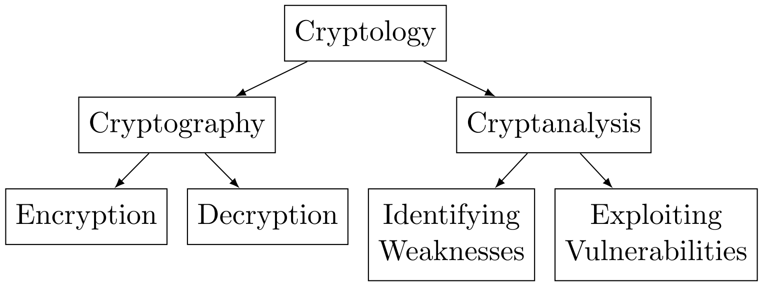


Figure 1: Cryptology Tree Structure

Cryptology is the overarching field that includes both Cryptography and Cryptanalysis. Hence, cryptology is defined as the study of cryptography as well as cryptanalysis. Now, Cryptography is essentially the process of hiding or encoding information so that only the intended recipient can read it. Cryptanalysis, on the other hand, is the process of finding weaknesses or vulnerabilities in cryptographic algorithms and exploiting them to decipher the encoded message.

2 NIST

NIST stands for the National Institute of Standards and Technology. NIST standardizes cryptographic algorithms. It also holds open competitions wherein participants can propose their cryptographic algorithms. These proposals undergo rigorous testing, and if they meet the standards, they are published to the real world.

3 An Example of Encryption

Suppose I have five ATM PINs that I want to encrypt for privacy. One approach could involve encoding each PIN using various rules, but this could complicate the decoding process due to the need to remember the specific encoding methods for each PIN. A straightforward encoding method involves adding a secret key X to all the ATM PINs. This secret key X is kept confidential. During decoding, I simply subtract X from all the encoded PINs to easily recover the original ATM PINs. This straightforward method simplifies the process and provides privacy to our ATM PINs.

$$\text{ATM1} \rightarrow \text{PIN1} + X = Y1$$
$$\text{ATM2} \rightarrow \text{PIN2} + X = Y2$$

Here,

- PIN1 is the original PIN.
- X is the secret key.
- $Y1$ is the ciphertext.
- The function PIN1 $+ X$ is known as the encryption function.

Formally, we can write the Encryption and Decryption functions as follows:

$$\begin{aligned} \textbf{Encryption:} \quad & E(P, K) = C \\ \textbf{Decryption:} \quad & D(C, K) = P \end{aligned}$$

In this mathematical formulation:

- E is the encryption function.
- D is the decryption function.
- P is the plaintext.
- C is the ciphertext/encoded text.
- K is the secret key.

Kindly note that we will be using the above notations extensively throughout this scribe.

4 Types of Cryptography

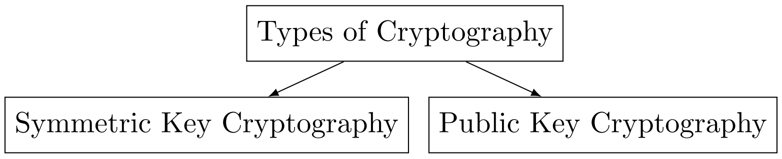


Figure 2: Types of Cryptography Tree Structure

4.1 Symmetric Key Cryptography

In this type, we use only one secret key for both encryption and decryption. Formally,

$$E(P, k) = C$$

$$D(C, k) = P$$

4.2 Public Key Cryptography

In this type, we use two keys: one is known as the public key (e_k), and the other is known as the secret key (d_k). Generally, the public key is used for encryption, and the secret key is used for decryption. Formally,

$$E(P, e_k) = C$$

$$D(C, d_k) = P$$

Note: The notations have their usual meanings as defined in the earlier sections.

5 Functionalities of Cryptography

A good cryptography method is expected to follow the following functionalities:

1. **Confidentiality** – Secrecy must be maintained throughout. This ensures that unauthorized parties cannot access the contents of the message.
2. **Integrity** – The message should be delivered from the sender to the receiver as a whole and should not be tampered with in between. This guarantees that the message remains unchanged during transmission.
3. **Authentication** – Verification must be in place to ensure that the user and the message are authentic and are coming from the user who sent them. This prevents unauthorized entities from posing as legitimate users.
4. **Non-repudiation** – This process ensures that you cannot deny that a message is not coming from you if you have really sent it. It provides evidence that the sender indeed sent the message and cannot later deny their involvement.

Note regarding mapping from Plaintext to Ciphertext and vice-versa

Encryption: $P \times \text{Encryption} \to C$
Decryption: $C \times \text{Decryption} \to P$
$P \times$ Encryption and $C \times$ Decryption are respective domains. Encryption transforms plaintext (P) into ciphertext (C), while decryption reverses this process.

6 Types of Functions

A function $f : A \to B$ is a relation between sets A and B if $a, b \in A$ and if $a = b$, then $f(a) = f(b)$. Here, the relation is a subset of $A \times B$. Let us understand a few types of functions:

1. **One-to-One functions:** Simply, $f(a) = f(b)$ if and only if $a = b$. These functions are injective, ensuring that each element in A maps to a distinct element in B.
2. **Onto functions:** If $f : A \to B$, then for all $b \in B$, there exists $a \in A$ such that $f(a) = b$. Onto functions are surjective, covering the entire set B with their mappings.
3. **Bijective functions:** Functions that are both one-to-one and onto. These functions are injective and surjective, meaning each element in A maps to a distinct element in B, and the entire set B is covered.
4. **Permutation:** Let π be a permutation on set S, then $\pi : S \to S$ is a bijection from S to S. Permutations represent rearrangements of elements in a set.

$$\pi : \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \to \begin{bmatrix} 2 & 3 & 1 & 4 \end{bmatrix}$$

5. **One-way functions:** These functions have an inverse, but the computation for finding the inverse is computationally heavy, taking decades to calculate. For example, consider a function that takes $p_1, p_2, p_3, \ldots, p_n$ and multiplies all of them together to form N. Computing this is relatively easy. However, given N, finding $p_1, p_2, \ldots, p_n$ such that their product equals N is extremely challenging, especially for large n of the order of some thousands. Therefore, the described function is an example of a one-way function.

Substitution Box/S-Box

A Substitution Box (S-Box) is defined as $S : A \to B$ with $|B| \leq |A|$, meaning one mapping will always repeat.

$$S : \{1, 2, 3, 4\} \to \{1, 2, 3\}$$

Here, let's say 1 maps to 1, 2 maps to 2, 3 maps to 3, and 4 maps again to 1. Hence, it cannot be one-to-one. S-Boxes are often used in cryptographic algorithms.n.

7 Various Kinds of Cipher (Symmetric Key Ciphers)

7.1 Caesar Cipher

Caesar cipher involves shifting every alphabet by three characters. We can shift it by any other number as well but in Caesar cipher this agreed number is 3 and is fixed. All the alphabets are mapped sequentially from 0 to 25 as follows:

$$A \to 0, B \to 1, \ldots, Z \to 25$$

Mathematically,

$$E(X, 3) = (X + 3) \mod 26$$
$$D(C, 3) = (C + 26 - 3) \mod 26 = (C + 23) \mod 26$$

It is interesting to note that in modulus 26 system, 23 is the additive inverse of 3.
Example:

SANIDHYA

$$S \to 18, A \to 0, N \to 13, I \to 8, D \to 3,$$
$$H \to 7, Y \to 24, A \to 0$$

Encrypted message: $21, 3, 16, 11, 6, 10, 27, 3$ (VDQLGKBD)
To decrypt, perform $(C + 23) \mod 26$, which gives $18, 0, 13, 8, 3, 7, 24, 0$, which is SANIDHYA.

7.2 Transposition Cipher

The message M is written into components $m_1 m_2 m_3 \ldots m_t$. We define a permutation C which is a permutation on t elements. For encryption, choose a permutation which can be given by:

$$\text{Encryption: } C = m_{e(1)} m_{e(2)} \ldots m_{e(t)}$$

For decryption, choose a permutation such that it reverses the effect of encryption permutation:

$$\text{Decryption: } M = C_{e^{-1}(1)} C_{e^{-1}(2)} \ldots C_{e^{-1}(t)}$$

Example: Let M be $CAESAR$ and secret key e be a mapping $[1, 2, 3, 4, 5, 6] \to [6, 4, 1, 3, 5, 2]$. Using this secret key, we encode as follows:

M: CAESAR $\to$ C: RSCEAA

To decode, use the inverse e which is a mapping $[1, 2, 3, 4, 5, 6] \to [3, 6, 4, 2, 5, 1]$.Using this secret key, we decode:

C: RSCEAA $\to$ M: CAESAR

7.3 Substitution Cipher

A substitution cipher replaces each letter in the plaintext with another letter based on a predetermined mapping, providing a simple form of encryption. This means that in this cipher, we simply substitute one alphabet with another alphabet:

$$A' = \{A, B, C, \ldots, Z\}$$

$$e : \text{substitution from } A' \text{ to } A'$$

$$C = e(m_1)e(m_2)\ldots e(m_t)$$

Example: Let $e(A) = Z, e(B) = D, e(C) = A$. Hence, using this cipher, the message ABC can be encoded as ZDA. To decrypt, perform the back mapping to get the plaintext.

7.4 Affine Cipher

Affine cipher uses the below mapping from alphabets to numbers and then performs some mathematical function to encode those numbers. The alphabets A to Z are mapped from 0 to 25 as,

$$A \to 0, B \to 1, \ldots, Z \to 25$$

We define the following,

$$A' = \text{Set of all alphabets}$$

$$\mathbb{Z}_{26} = \{0, 1, \ldots, 25\}$$

$$A' \to \mathbb{Z}_{26} \text{ with the mapping as defined above}$$

$$k = \text{secret key which is a tuple } (a, b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26}$$

$$x = \text{plaintext and } x \in \mathbb{Z}_{26}$$

For encryption we use the following function –

$$\text{Encryption: } e(x, k) = (ax + b) \mod 26 = c$$

For decryption:

$$\text{Decryption: } d(c, k) = ((c–b)a^{-1}) \mod 26$$

Here a^{-1} is such that $a \cdot a^{-1} \equiv 1 \mod 26$. We will be able to use affine cipher if and only if we find some a whose a^{-1} exists given the condition that $\gcd(a, 26) = 1$.

7.4.1 Multiplicative Inverse

Let y be the multiplicative inverse of x modulo m. Hence,

$$x \cdot y \equiv 1 \mod m$$

Extended Euclidean Algorithm is used to find the GCD of two integers. Let us first compute GCD of $x = 3$ and $y = 17$ using Euclid's Division Algorithm:

$$17 = 3 \cdot 5 + 2$$

$$3 = 2 \cdot 1 + 1$$

$$2 = 1 \cdot 2 + 0$$

Hence, $\text{GCD}(3, 17) = 1$. Now, going in reverse direction of this will lead us to the values of a and b.

$$1 = 1 \cdot 3 - 1 \cdot 2$$

$$1 = 1 \cdot 3 - 1 \cdot (1 \cdot 17 - 5 \cdot 3)$$

$$1 = 6 \cdot 3 - 1 \cdot 17$$

Hence, $a = 6$ and $b = -1$.

7.4.2 Euler Totient Function ($\phi(n)$)

This function is used to find the number of positive integers which are relatively prime to n and also smaller than n. Mathematically,

$$\phi(n) = \begin{cases} (p-1)(q-1) & \text{if } n \text{ is non-prime where } p \text{ and } q \text{ are co-prime factors of } n \\ (p-1) & \text{if } p \text{ is prime} \end{cases}$$

In our case where $n = 26$ (since 26 alphabets are in the English alphabet), we get $\phi(n) = (2-1)(13-1) = 12$. Therefore, the possible values for a in the key are 12 out of 26, and there are 26 possible values for b. Consequently, there is a total of $12 \cdot 26 = 312$ keys possible for the Affine Cipher. This limitation arises from the requirement for a to have a multiplicative inverse modulo 26.

7.5 Playfair Cipher

This cipher consists of a 5×5 matrix constructed using the secret key chosen and the alphabets of the English alphabet. Since there are 26 alphabets in the English language and this cipher involves the use of 25 (a 5×5 matrix), it is assumed in this cipher that $i = j$. It would be easier to understand this cipher using an example.

7.5.1 Matrix Construction

Let's consider our secret key as "PLAYFAIREXAMPLE." We will construct a 5×5 matrix by entering alphabets in a row-wise fashion ((0,0), (0,1), (0,2), (0,3), (0,4), (1,0), (1,1),...,(4,4)). We will start by entering all the unique alphabets from our secret key into the matrix without repetition and then fill the remaining English alphabets in the matrix. Note that $i = j$ during this process.

For the secret key "PLAYFAIREXAMPLE," the matrix would be as follows:

$$\begin{bmatrix} P & L & A & Y & F \\ I & R & E & X & M \\ B & C & D & G & H \\ K & N & O & Q & S \\ T & U & V & W & Z \end{bmatrix}$$

7.5.2 Encryption and Decryption Process

Now, let's understand how encryption and decryption are done using the Playfair cipher. We will break our plaintext into blocks of two and then process each block.

Example If the message is HIDE, we will break it into two blocks: HI and DE. Now consider the first block HI, where H and I form a rectangle in the matrix. We will replace each letter with the opposite corner letter of the same row. So, H becomes B, and I becomes M. Similarly, consider the second block DE, where D and E form a column in the matrix. We will replace each letter with the letter just below it, wrapping to the top if needed. So, D becomes O, and E becomes D. Hence, we successfully encoded HIDE as BMOD.

$$\begin{array}{|c|c|}\hline \text{H} & \text{I} \\ \hline \text{D} & \text{E} \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|}\hline \text{B} & \text{M} \\ \hline \text{O} & \text{D} \\ \hline \end{array}$$

To decode, we will do the opposite of what we did while encoding. Using the ciphertext BMOD, we break it into two blocks: BM and OD. For the first block BM, the letters form a rectangle in the matrix. We replace each letter with the opposite corner letter of the same row. So, B becomes H, and M becomes I. Similarly, for the second block OD, the letters form a column. We replace each letter with the letter just above it, wrapping to the bottom if needed. Therefore, O becomes D, and D becomes E. Hence, we successfully decoded BMOD back to HIDE.

$$\begin{array}{|c|c|}\hline \text{B} & \text{M} \\ \hline \text{O} & \text{D} \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|}\hline \text{H} & \text{I} \\ \hline \text{D} & \text{E} \\ \hline \end{array}$$

7.5.3 Rules for Playfair Cipher

The following conditions are to be followed in the Playfair cipher:

To Encode

1. If letters of a block form a rectangle: Replace the letters with letters in the opposite corner of the same row.
2. If letters of a block form a row: Replace the letters with the letters just right to it and wrap it to the left if needed.
3. If letters of a block form a column: Replace the letters with the letters just below it and wrap it to the top if needed.

To Decode

1. To decode (1): follow (1).
2. To decode (2): Replace the letters with the letters just left to it and wrap it to the right if needed.
3. To decode (3): Replace the letters with the letters just above it and wrap it to the bottom if needed.

Handling Odd-Length and Repetition

To ensure a smooth encryption process and accommodate odd-length and repeated characters in the Playfair cipher, we follow these guidelines:

1. For even-length texts, divide them into blocks of 2 as usual.

2. For odd-length texts, add an X to the end of the plaintext to make it even length before dividing it into blocks.
3. Address repetition by adding an extra X after every repeated alphabet in the plaintext.

For example, with the plaintext "BALL," we add an X after the first L, resulting in the blocks BA, LX, LX. The encoding process is then performed as defined above.

7.5.4 Issues with Playfair Cipher

There are various issues with the Playfair cipher. For example, HIDE is encoded as BMOD using the secret key PLAYFAIREXAMPLE, and while decoding, we will get two options: HIDE and HJDE because we assumed $i = j$. So, we will have to manually check which of the possible combinations is correct.

[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 2)

Important: Total number of bijective mappings from $\mathbb{Z}_{26}$ to $\mathbb{Z}_{26}$ is 26!.

1 Hill Cipher

In the Hill Cipher, the secret key is represented by an $n \times n$ invertible matrix A. The message M is then broken down into components $m_1, m_2, m_3, \ldots, m_n$.

- Secret Key: $A = (a_{ij})_{n \times n}$, where A is an invertible matrix.
- Plaintext: $M = m_1 m_2 m_3 \ldots m_n$
- For encryption: $C(Ciphertext) = A \cdot M$, where each C_i is calculated using $C_i = \sum_{j=1}^{n} a_{ij} \cdot m_j$.
- For decryption: $M = A^{-1} \cdot C$, here the inverse of matrix A is used to decode the ciphertext back into plaintext.

2 Symmetric Key Ciphers

Symmetric Key ciphers are those in which the same key is used for both encryption and decryption. If you know the key for encryption, then you can calculate the key for decryption. There is no concept of a private key and public key in symmetric key ciphers. There are two types of ciphers:

1. **Block Cipher**
2. **Stream Cipher**

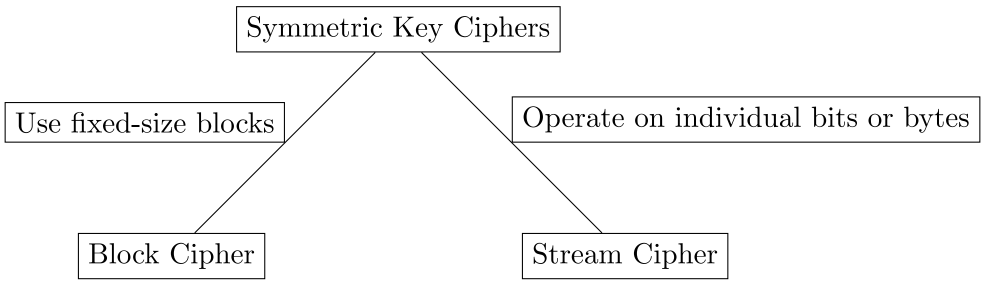


Figure 1: Types of Symmetric Key Ciphers

2.1 Block Ciphers

In block cipher, the plaintext M is broken down into r number of blocks, each of fixed block size n.

$$\text{len}(M) = n \cdot r$$

$$\text{len}(m_i) = n$$

It is important to note here that Block Size n is very important in terms of block ciphers. So,

$$\text{Plaintext } M = m_0||m_1||m_2||m_3||\ldots||m_r$$

For example, we can divide a 32-bit message into 4 blocks of size 8 each. Here, $n = 8$ and $r = 4$.

Now, we apply the encryption function $E(m_i, k)$ on each of these blocks separately to calculate our ciphertext C, and similarly, we apply the decryption function $D(C_i, k)$ on each block separately to get back our plaintext P.

Encryption: $C = E(m_0, k)||E(m_1, k)||\ldots||E(m_r, k)$ Hence, $C = c_0||c_1||\ldots||c_r$

Decryption: $P = D(c_0, k)||D(c_1, k)||\ldots||D(c_r, k)$

Now there can be two cases which can arise while breaking our plaintext into blocks-

Case 1 − Length of plaintext P is exactly divisible by block size n: Here, there would not be an issue, and the process remains the same as described in the above section.

Case 2 - Length of plaintext P is not divisible by block size n: Here, we would not be able to divide the message into blocks of equal size because the last block will have characters which are less than the fixed block size. Let's consider the below example where M is divided into m_0 (size = n), m_1 (size = n), m_2 (size = l, which is less than n). Total length of the message = $2n + l$ where $l < n$.

$$M = m_0||m_1||m_2$$

Now we would pad our m_2 and add extra 0's to the last so that we attain block size n for the last block. Let's say we add r 0's to the last block,

$$l + r = n$$

Next, we would follow the encryption process as usual as described above –

$$C = c_0||c_1||c_2$$

$$c_0 = E(m_0, k)$$

$$c_1 = E(m_1, k)$$

$$c_2 = E(m_2||0^{n-l}, k)$$

$m_2||0^{n-l}$ since we did padding with 0's.

Similarly for decryption, we would simply perform decryption block by block and then remove the last r characters from our decoded plaintext to get actual plaintext.

ECB (Electronic Control Board) mode of operation – The method we studied above is referred to as ECB mode of operation. There are other modes of operation as well. This ECB mode is not secure due to the following reasons –

1. If two blocks are the same, then their corresponding encrypted blocks would also be the same. This reveals the information that we have 2 blocks of plaintext that are the same, causing information leakage.
2. As we saw that we need to add extra 0s if the length of plaintext M is not exactly divisible by block size n, we would also need to share how many 0s we padded in the last block to subtract that many numbers of 0s while decryption. This sharing of information regarding the number of 0s padded is also a kind of information leakage.

Types of Block Ciphers

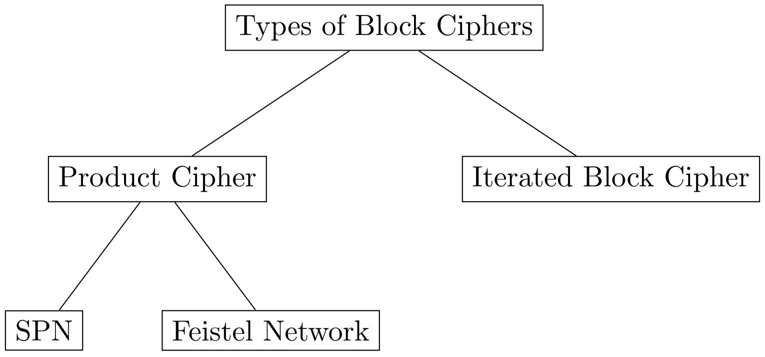


Figure 2: Tree Diagram of Block Ciphers

2.1.1 Product Cipher

Product cipher combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components. Basically, it incorporates multiple cipher techniques to come up with a new, more secured one!

Now, any product cipher falls into any one of the following two types –

a. Substitution-Permutation Network (SPN)

Here the advantage of both – Substitution cipher (S-box) and Permutation Cipher is considered. First, the one or more S-boxes operate on each block of plaintext, and then one or more permutation ciphers permute these substituted blocks to come up with the encrypted text.

Encryption:

$$S : \{0,1\}^n \to \{0,1\}^m$$
$$P : \{0,1\}^{mr} \to \{0,1\}^{mr}$$

Here, (consider Fig. 3) we have r blocks of size n each; the output of each box is a block of size

m ($m < n$ according to S-box definition). Then these r blocks of size m are combined into mr length and then permuted using a permutation cipher. This marks the completion of 1 round of encryption. In practice, there can be many such rounds. In round 2, again, this mr length is broken down into blocks of size n, and the process is repeated r_1 number of times. Repetition r_1 number of times denotes that this SPN is of r_1 rounds.

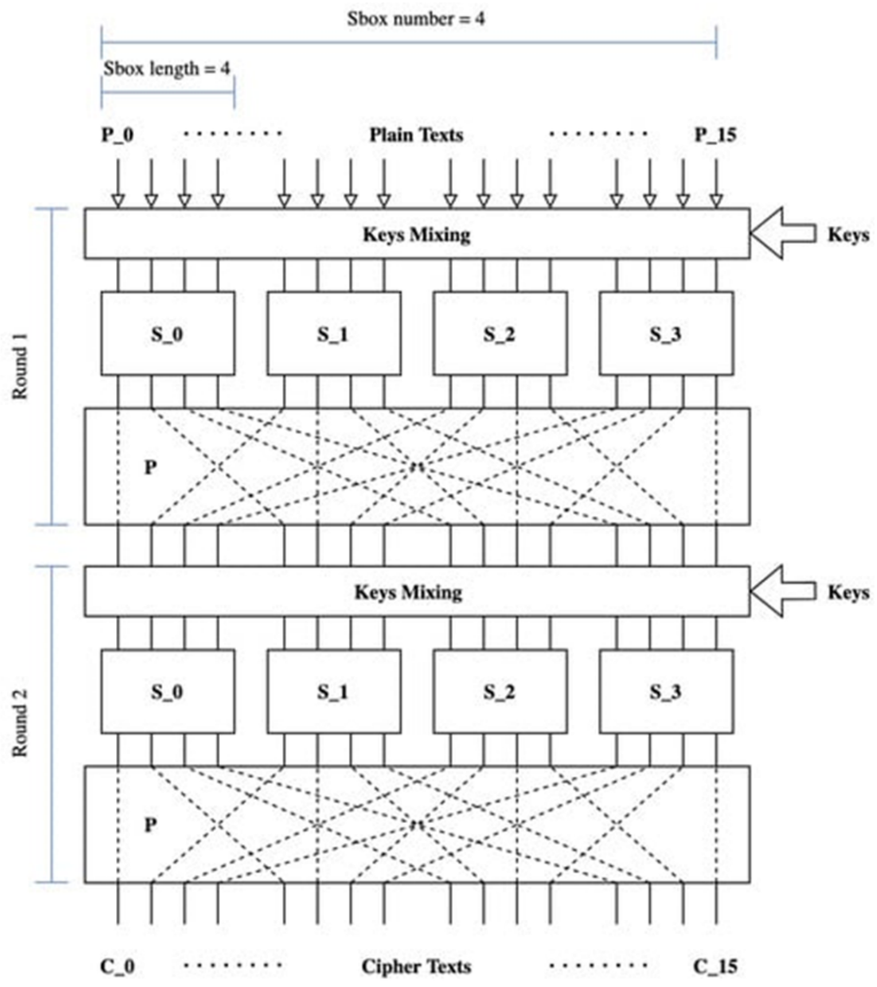


Figure 3: Substitution-Permutation Network (SPN)

Decryption:

For decryption, we would simply backtrack and apply the reverse permutation and apply S^{-1} S-box (usually S-boxes are invertible in nature) on each of the block to get back our plaintext M.

b. Feistel Network

This is the second type of product cipher, and here the plaintext M is broken down into 2 blocks, each of size n. If the plaintext M is of odd length, then we pad an extra 0 at the end to make it divisible by 2, and hence we can split it equally into two halves. In both cases, the length of the final plaintext is $2n$. Now, these two blocks are referred to as L_0 (left block) and R_0 (right block). We perform some basic operation using secret key k and an encryption function f on these blocks to come up with encrypted blocks L_1 and R_1, which are again of size $2n$.

Encryption:

$$F : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$$

$$L_1 : R_0$$

$$R_1 : L_0 \oplus f(R_0, k)$$

Here $\oplus$ is the XOR operation.

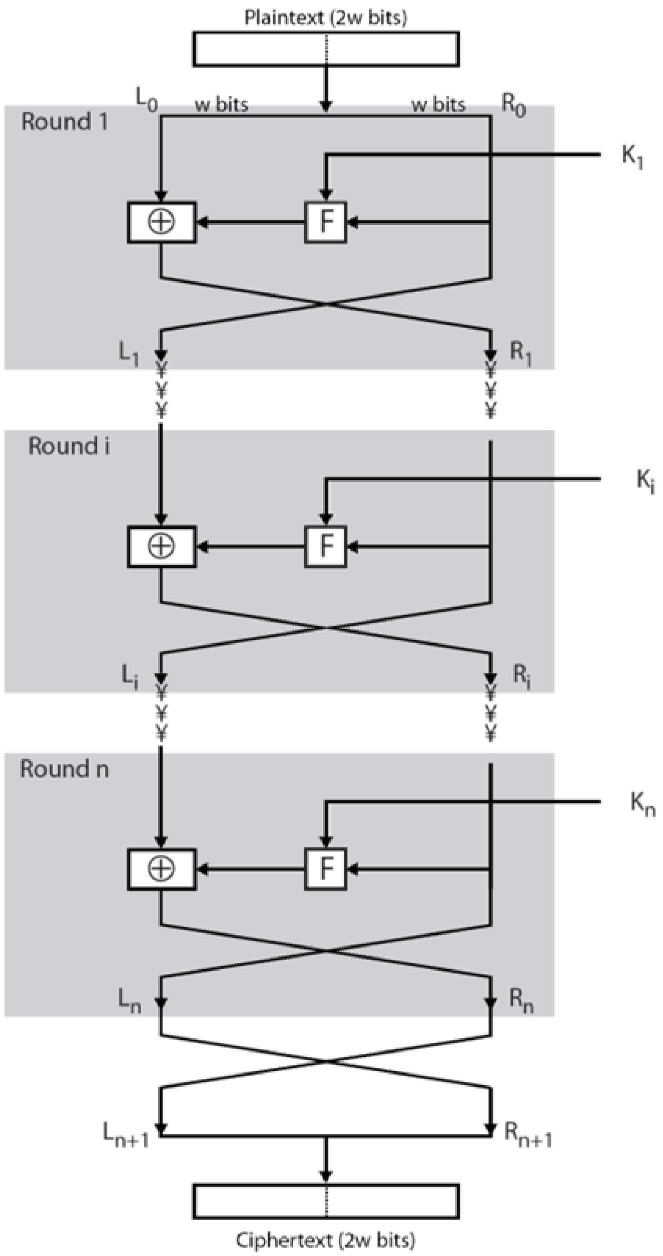


Figure 4: Feistel Network

Here, we have our plaintext of size $2w$, which is broken down into two blocks of size w each. Now we have L_0 and R_0 with us. To get L_1, we simply copy-paste R_0, and to get R_1, we will perform $L_0 \oplus f(R_0, k_1)$, where f is some encryption function and k_1 is the secret key for round 1. Like SPN, we can have multiple rounds of encryption with k_i representing the secret key used in each round. Performing this r_1 number of times, we call it to be a r_1-round Feistel network (here n is used instead of r_1 to denote the number of rounds). At the end, we obtain a ciphertext C of the same length as that of the plaintext M.

Decryption:

For decryption, we simply backtrack, and we can obtain L_0 and R_0 from L_1 and R_1 as follows:

$$L_0 = R_1 \oplus f(L_1, k)$$

$$R_0 = L_1$$

A similar approach can be applied to r_1-round Feistel network.

The main advantage which is to be noted here is that even if the function f is not invertible, we are able to decrypt our ciphertext C because in decryption also the same function f is used which was used for encryption and not its inverse, unlike other techniques.

2.1.2 Iterated Block Cipher

An iterated block cipher is a block cipher involving the sequential repetition of an internal function called the round function. The parameters include the number of rounds r_1, the block size n, and the bit size k of the input key from which r subkeys k_i (round keys) are derived. Let's consider the following example to understand it better –

Encryption: $F(k_1, P) \to C_1 \to F(k_2, C_1) \to C_2 \to$ final ciphertext

This encryption illustrated above is a 2-round Iterated block cipher. Here, the secret keys of each round (k_i) can be generated by a Key Scheduling Algorithm as follows –

$$K \to G(K) \to k_1, k_2$$

Here, K is the master Key, G is the key scheduling algorithm, and k_1 and k_2 are the round keys derived from the above algorithm.

Decryption: $C \to F^{-1}(C, k_2) \to C_1 \to F^{-1}(C_1, k_1) \to P$

Do note that here F^{-1} means a function which reverses the effect of F and not the actual mathematical inverse of function F.

3 DES (Data Encryption Standard)

This method of encryption was designed by IBM. The design is based on the Feistel network, which we studied above. Here, we use a 64-bit secret key, and the block size of each plaintext is also 64 bits. The output of each round of encryption is also a 64-bit ciphertext. Similar properties apply for decryption. The block diagrams of encryption and decryption are as follows –

Encryption:

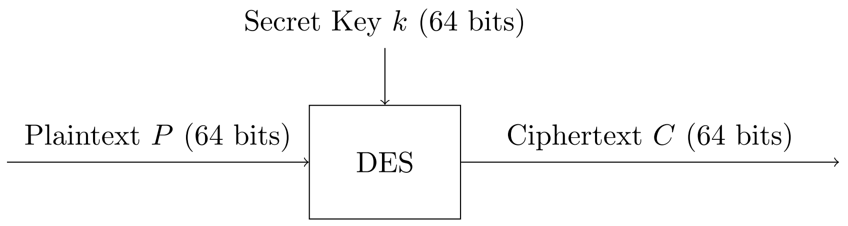


Figure 5: DES encryption block diagram

Decryption:

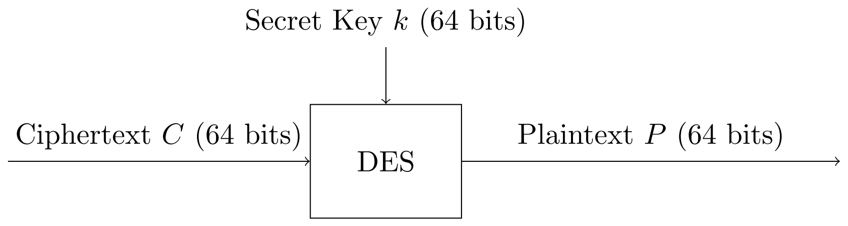


Figure 6: DES decryption block diagram

Secret Key: The secret key k is of 64 bits, of which 8 are the parity bits and are used to verify the other 56 bits against transmission.

Let us understand how this 64-bit key is made by considering the below example –

$$\underbrace{0111011\mathbf{1}}_{\text{Chunk 1}}\underbrace{1100101\mathbf{0}}_{\text{Chunk 2}}\underbrace{1010101\mathbf{0}}_{\text{Chunk 3}}\ldots$$

The secret key is divided into 8 chunks of 8 bits each. The last bit in each chunk is the parity bit of that chunk of 7 bits. Consider chunk 1 – 0111011; we have five 1s, and hence the parity bit (8th bit of chunk 1) would be 1. Hence, we obtain 0111011**1**.

Therefore, if you know 56 bits, then you can easily derive the rest 8 bits, hence obtaining the complete 64-bit secret key k.

There are a few important properties of DES –

a. DES is based on the Feistel Network

b. It is an Iterated Block Cipher

c. Number of rounds $(r_1) = 16$

d. Key scheduling algorithm is used which generates 16 keys $(k_1, k_2, \ldots, k_{16})$ for each of the 16 rounds in the network. Key scheduling algorithm takes a master key K of 64 bits and generates $k_1, k_2, k_3, \ldots, k_{16}$.

e. Each k_is are of 48 bits: K (64 bits) $\rightarrow$ [G] $\rightarrow k_1, k_2, \ldots, k_{16}$ (48 bits each)

Encryption network:

$$IP(InitialPermutation) : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$$

$$f : (\{0,1\}^{32}) \times \{0,1\}^{48} \rightarrow \{0,1\}^{32}$$

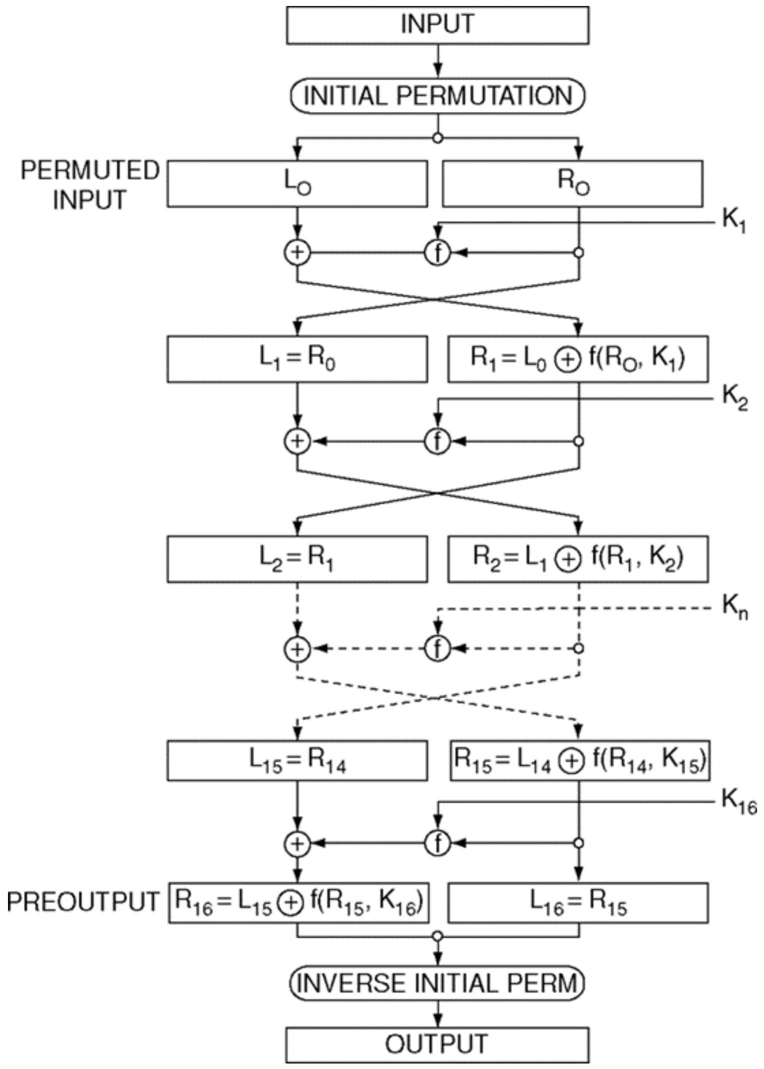


Figure 7: DEC Encryption

We start with the input plaintext M of 64 bits and apply an Initial Permutation over it. The result is then split into two equal halves – L_0 and R_0, each of 32 bits. We perform operations similar to a Feistel network for 15 rounds:

$$L_1 = R_0$$
$$R_1 = L_0 \oplus f(R_0, k_1)$$

These k_i keys are generated by the key scheduling algorithm. The Feistel network is repeated for 15 rounds, and the 16th round is slightly different:

$$L_{16} = R_{15}$$
$$R_{16} = L_{15} \oplus f(R_{15}, k_{16})$$

Note that this time, L_{16} is the right-side block, and R_{16} is the left-side block. This was just a notation strategy which was followed by the creators. Also a small note that the keys are different is each round. After these 16 rounds, both blocks are concatenated, and an Inverse Initial Permutation is applied to reverse the effect of the Initial Permutation applied at the start. The final ciphertext C is obtained, which is again of 64 bits.

Decryption Process in Feistel Network:

For decryption, we backtrack and reverse each of the effects by applying appropriate reversal strategies, as discussed in the Feistel network section.

4 References

- **Figure 3:** `https://www.mdpi.com/2410-387X/6/4/60?type=check_update&version=2`
- **Figure 4:** `https://iasbs.ac.ir/~farajian/slides/network%20security/ns_session3.pdf`
- **Figure 7:** `https://link.springer.com/referenceworkentry/10.1007/0-387-23483-7_94`

[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 3)

1 Decryption in DES

Ciphertext is first processed using IP (since at the end of DES encryption network we had an IP^{-1}), then further the inversion is done as follows –

$$R_{15} = L_{16}$$
$$L_{15} = R_{16} \oplus f(L_{16}, k_{16})$$

This process continues until we reach L_1 and R_1. After getting L_1 and R_1, we perform the following to obtain R_0 and L_0 –

$$R_0 = L_1$$
$$L_0 = R_1 \oplus f(L_1, k_1)$$

At last, we perform an IP^{-1} (since at the start of DES encryption network we had an IP). After the successful completion of the above-mentioned process, we finally get the plaintext!

2 Elements used in DES

2.1 Initial Permutation (IP)

The Initial Permutation (IP) design involves rearranging the bit positions in the data using the table presented below. The bits are permuted according to the specified table, resulting in two halves: L_0 and R_0, with a 32-bit split. In mathematical terms,

$$IP : \{0,1\}^{64} \to \{0,1\}^{64}$$

can be expressed as

$$IP(m_1, m_2, m_3, \ldots, m_{64}) = m_{58}m_{50}m_{42}\ldots m_7.$$

The IP matrix illustrates the bit positions that replace the actual bit positions during this permutation process. Calculating the inverse of IP is straightforward; it involves populating the matrix with position values that correspond to the actual bit positions in the plaintext message matrix.

For instance, the 1st bit in the plaintext occupies the 40th position, the 2nd bit is in the 8th position, and the 3rd bit is in the 48th position of the IP matrix. Consequently, the first three entries of the IP^{-1} matrix will be 40, 8, and 48, respectively.

2.2 Round function (f)

The function $F(R_i, K_i)$ is defined as X_{i+1}, where R_i and X_{i+1} are both 32 bits in length, and K_i is 48 bits.

Mathematically, $F : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$ is expressed as:

$$F(R_i, K_i) = X_{i+1}, \text{ where } R_i \text{ and } X_{i+1} \text{ are of 32 bits and } K_i \text{ is of 48 bits.}$$

Further elaborating on the expression, we have:

$$F(R_{i-1}, K_i) = P(S(E(R_{i-1} \oplus K_i)))$$

Here, E maps a 32-bit input to a 48-bit output, and P involves permuting the positions of 32 bits. The S-Box takes a 48-bit input and produces a 32-bit output.

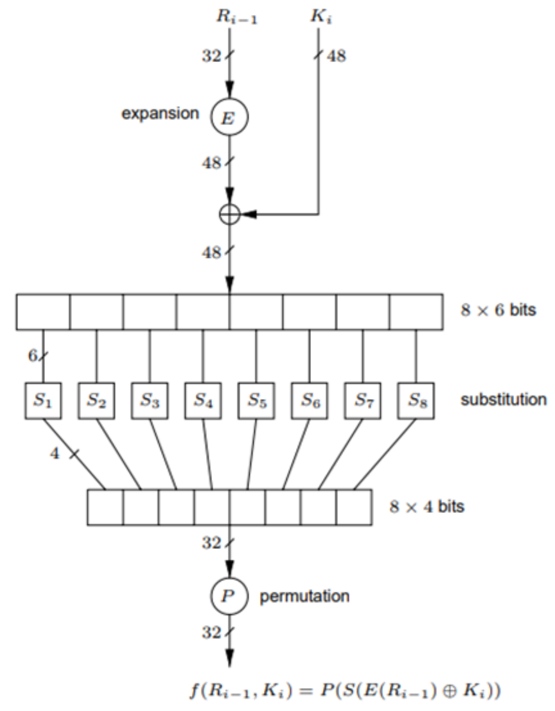
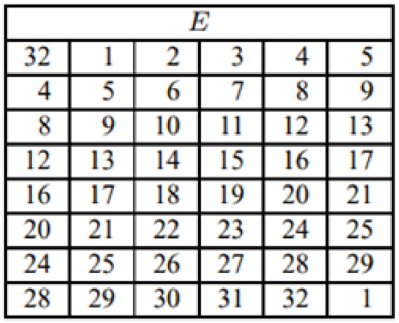


Figure 1: Round function f in DES

2.2.1 E - Mapping

E is a mapping shown by the following diagram:

$$\begin{array}{ccccccc}
 & \vdots & & \vdots & & \vdots & \vdots \\
x_1 & x_2 & x_3 & \ldots & x_{32} & & \\
\hline
 & \vdots & & \vdots & & \vdots & \vdots \\
y_1 & y_2 & y_3 & \cdots & y_8 & &
\end{array}$$



E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Figure 2: E inside round function f

It's observed that the last two column values are repeated. For instance, in the second row, the values 4 and 5 from the last two columns of the first row are duplicated. This repetition pattern is illustrated as:

$$E : \{0,1\}^{32} \to \{0,1\}^{48}$$
$$E(x_1x_2x_3 \dots x_{32}) = y_1y_2y_3 \dots y_8$$
$$E(x_1x_2x_3 \dots x_{32}) = (x_{32}x_1x_2x_3x_4 \dots x_{32}x_1)$$

2.2.2 S-Box

The S-Box in function f is responsible for mapping 48 bits of data to 32 bits, denoted as:

$$S : \{0,1\}^{48} \to \{0,1\}^{32}$$

For a given 48-bit input $X = B_1B_2B_3B_4B_5B_6B_7B_8$, where each B_i represents a block of length 6 bits, the S-Box function is defined as $S(X) = Y$, where Y is a 32-bit output.

Each B_i is further broken down into $b_1b_2b_3b_4b_5b_6$, where b_i belongs to $\{0,1\}$. The mapping for each S_i from 6 bits to 4 bits is expressed as:

$$S_i : \{0,1\}^6 \to \{0,1\}^4 \text{ for } i = 1,2,3,\dots,8$$

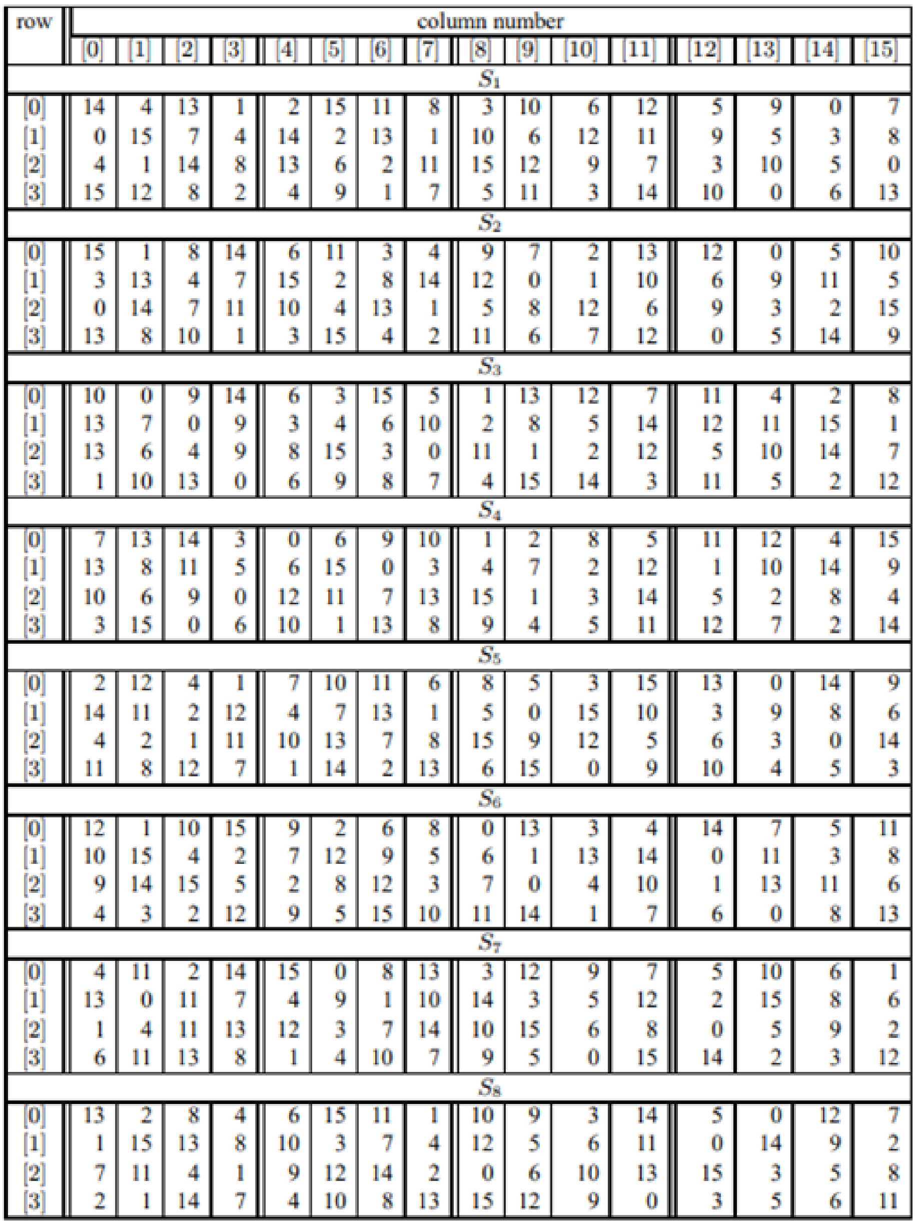
The mapping function $S_i(B_i) = C_i$, where C_i is a 4-bit output. The overall S-Box operation for the 48-bit input X is given by:

$$S(X) = (S_1(B_1), S_2(B_2), \dots, S_8(B_8))$$

In the figure below, the S-Boxes in DES are fixed. Each S_i is represented as a matrix with 4 rows (0 to 3) and 16 columns (0 to 15). The values a_{ij} in the matrix range from 0 to 15, and the table in Figure 10 associates these values with x in the range from 0 to 15. The calculations for r and c are specified as:

$$r = (2 \cdot b_1 + b_6) \quad \text{where } 0 \leq r \leq 3$$

$$c = \text{integer representation of } (b_2 b_3 b_4 b_5) \quad \text{where } 0 \leq c \leq 15$$



row	column number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure 3: S inside round function f

2.2.3 Permutation (P)

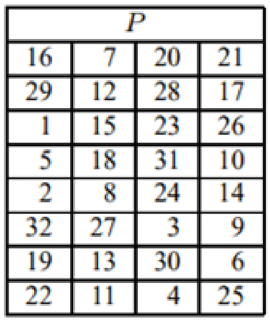
The Permutation function in DES performs the permutation of 32 bits of data and outputs 32 bits. The specific permutation applied in DES is illustrated in Figure 11.

$$P : \{0,1\}^{32} \to \{0,1\}^{32}$$

For a given input $x_1x_2x_3\ldots x_{32}$, the permutation function is defined as:

$$P(x_1x_2x_3\ldots x_{32}) = (x_{16}x_7x_{20}x_{21}\ldots x_4x_{25})$$

In this permutation, the output rearranges the input bits based on the specified order outlined below.



P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Figure 4: P inside round function f

2.3 Key Scheduling Algorithm

Key K is a 64-bit sequence, denoted as $k_1\ldots k_{64}$, inclusive of 8 odd-parity bits. The desired outcome involves generating 16 round keys, K_i $(1 \leq i \leq 16)$, each with a length of 48 bits.

Algorithm Breakdown:

1. Define v_i, where $1 \leq i \leq 16$, as follows: v_i is set to 1 for i belonging to $\{1, 2, 9, 16\}$; otherwise, v_i is set to 2. These values represent the left-shift amounts for 28-bit circular rotations.

2. Discard the 8 parity check bits from K, resulting in $\tilde{k}$.

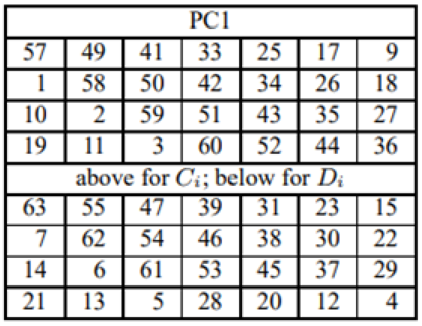
3. Represent T as 28-bit halves (C_0, D_0) after applying the permutation choice $PC1$ to $\tilde{k}$. $PC1$ maps $\{0,1\}^{56}$ to $\{0,1\}^{56}$.

4. Utilize $PC1$ (refer to the table in Figure 12) to select bits from K, obtaining $C_0 = k_{57}k_{49}\ldots k_{36}$ and $D_0 = k_{63}k_{55}\ldots k_4$. Both C_0 and D_0 are 28 bits in length.

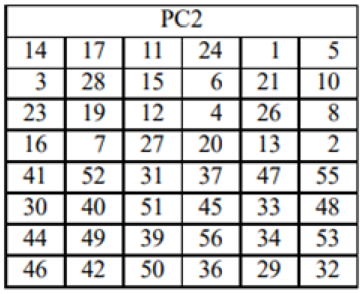
5. For each i from 1 to 16, compute K_i as follows:

   - C_i undergoes left circular shift by v_i, denoted as $C_i \leftarrow (C_{i-1} \hookleftarrow v_i)$.
   - D_i undergoes left circular shift by v_i, denoted as $D_i \leftarrow (D_{i-1} \hookleftarrow v_i)$.
   - Perform the left circular shifts using $\hookleftarrow$. C_i and D_i represent the left and right components, respectively.
   - $PC2$, detailed in the table from Figure 12, is used to select 48 bits from the concatenation $b_1b_2\ldots b_{56}$ of C_i and D_i, resulting in $K_i = b_{14}b_{17}\ldots b_{32}$.

- Lastly, compute K_i using $PC2(C_i, D_i)$, where K_i is the round key for the ith round. $PC2$ maps as follows: $PC2 : \{0,1\}^{56} \rightarrow \{0,1\}^{48}$.



PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4



PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figure 5: PC1 and PC2 inside Key scheduling algorithm

Few Interesting Properties

The application of $PC1$ to the key $K_1K_2K_3 ... K_{63}K_{64}$ involves a permutation of positions, excluding the parity bits. The resulting sequence is $K_{57}K_{49}K_{41}K_{33} \dots K_4$.

Notably, in $PC1$, the operation is solely focused on rearranging positions. If the keys are complemented, the output will also be complemented.

Expressed in a general form, $PC1(K_{1,K_{2,\dots,K_{64})}}$ yields $K_{57K_{49\dots K_9}}$, equivalent to $PC1(K)$.

Regarding $PC2$, according to the information in Figure 12, the input bit at the 14th position is transferred to the 1st position in the output during the permutation.

3 Complement Properties

The DES encryption operation, denoted as $DES(M, K) = C$, and its complemented counterpart, $DES(\bar{M}, \bar{K}) = \bar{C}$, are linked.

Key scheduling, denoted as $KS(K)$, produces round keys $K_1, K_2, \dots, K_{16}$, while the complemented key scheduling, $KS(\bar{K})$, yields round keys $\bar{K}_1, \bar{K}_2, \dots, \bar{K}_{16}$. This complementation is consistent across $PC1$, $PC2$, and left shifts.

The rationale behind the result in equation (1) is explained as follows:

Consider the initial setup:

$$\begin{array}{ccc} M & & \bar{M} \\ L_0 & R_0 & \bar{L}_0\bar{R}_0 \\ L_1 & R_1 & \bar{L}_1\bar{R}_1 \end{array}$$

In the first round:

$$\begin{aligned} M: \quad R_1 &= L_0 \oplus F(R_0, K_1) \\ \bar{M}: \quad R_1 &= \bar{L}_0 \oplus F(\bar{R}_0, \bar{K}_1) \end{aligned}$$

If we compare the uncomplemented case (L_1, R_1) to (L_0, R_0), the complementation of both plaintext and key results in complemented inputs for the XOR before the S-boxes. This double complementation cancels out, yielding S-box inputs and the overall result $f(R_0, K_1)$. The result is then XORed with $\bar{L}_0$ (previously L_0), resulting in $\bar{L}_1$.

This effect continues in subsequent rounds. For instance, in the next step:

$$\begin{gathered} \bar{M} : E(\bar{R}_0) = (E(R_0))^{\bar{}} \oplus \bar{K}_1 = E(R_0) \oplus K_1 \\ R_1 = \bar{L}_0 \oplus f(E_0, K) \\ R_1 = \bar{R}_1 \end{gathered}$$

Thus, R_1 will also be complemented. The next step involves the initial permutation (IP), which is also complemented. Consequently, the ciphertext will be the complement!

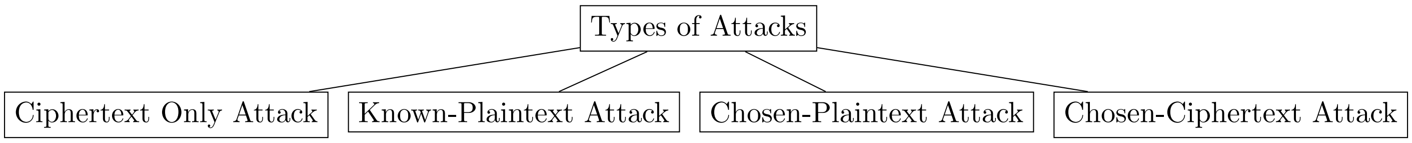
Exhaustive Search or Brute-force Attack in DES:

- The key size in DES is 56 bits, leading to a total of possible keys (S) denoted as $K_1, K_2, \ldots, K_{2^{56}}$.
- The time complexity of a brute-force attack on DES is 2^{56}, exploring all potential keys.

Chosen-Plaintext Attack in DES:

- In this attack, the attacker selects specific plaintexts for which the corresponding ciphertexts are to be provided.
- The total possible keys (S) are $K_1, K_2, \ldots, K_{2^{56}}$.
- Assuming the attacker chooses two plaintexts, M and $\bar{M}$, and uses the secret key K to generate the corresponding ciphertexts:
  - $DES(M, K) = C_1$
  - $DES(\bar{M}, K) = C_2$
- Leveraging the complement property in DES, we have $DES(\bar{M}, \bar{K}) = \bar{C}_2$, where $\bar{M} = M$.
- The attacker employs a test key, K_i, to attempt decryption of M using K_i to obtain a ciphertext C. According to the complement property, decrypting the complement of M with the complement of K_i yields the complement of C.
- If the equality holds true $(C \neq C_1)$, discard K_i from S as K_i is not equal to K.
- If the complemented equality holds true $(\bar{C} \neq \bar{C}_2)$, discard $\bar{K_i}$ from S as $\bar{K_i}$ is not equal to K.
- This approach enables finding the key in 2^{55} attempts, one less than the total keys, as two keys are discarded during the process.

4 Types of Attacks

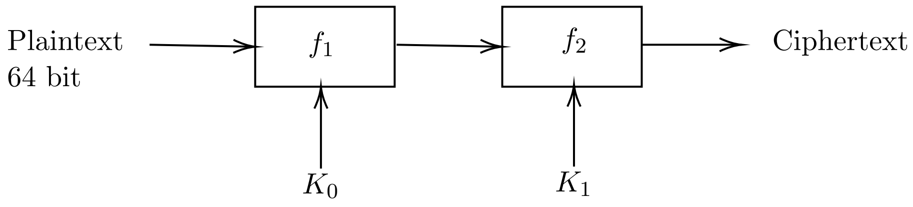


- **Ciphertext Only Attack:**
  - The attacker possesses only the ciphertext and aims to recover either the plaintext or the secret key.
- **Known-Plaintext Attack:**
  - The attacker has knowledge of certain plaintexts and their corresponding ciphertexts.
  - The objective is to either find a plaintext corresponding to a different ciphertext or uncover the secret key.
- **Chosen-Plaintext Attack:**
  - In this attack, the assailant selects specific plaintexts and is allowed to obtain the corresponding ciphertexts.
  - The goal is to generate a new plaintext/ciphertext pair or discover the secret key.
- **Chosen-Ciphertext Attack:**
  - The attacker chooses certain ciphertexts and is provided with their corresponding plaintexts.
  - The aim is to create a different valid plaintext/ciphertext pair or reveal the secret key.

It is worth noting that in public-key cryptography, the Chosen-Ciphertext Attack is considered the strongest, given the pivotal role of the decryption secret key. In symmetric key cryptography, both Chosen-Plaintext and Chosen-Ciphertext attacks are nearly equally potent since the same key is employed for both encryption and decryption.

5 Variations of DES

5.1 Double DES



To enhance the security of DES, a common practice is to perform double encryption. However, the assumption that this would provide $2 \times 56 = 112$ bits of security is proven incorrect, and a Meet-in-the-Middle attack illustrates this.

In double encryption, a key K is represented as 128 bits (K_0, K_1), with 16 parity check bits among them. The encryption process involves two stages using different keys. The diagram depicts plaintext (P) of 64 bits undergoing encryption with DES.

Exhaustive search on double encryption would imply a time complexity of 2^{112}, but this is refuted by the Meet-in-the-Middle attack:

Meet-in-the-Middle Attack:

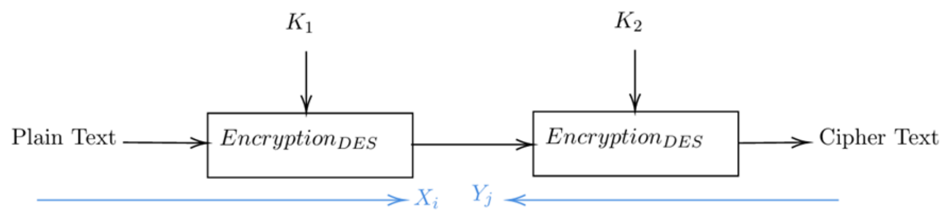
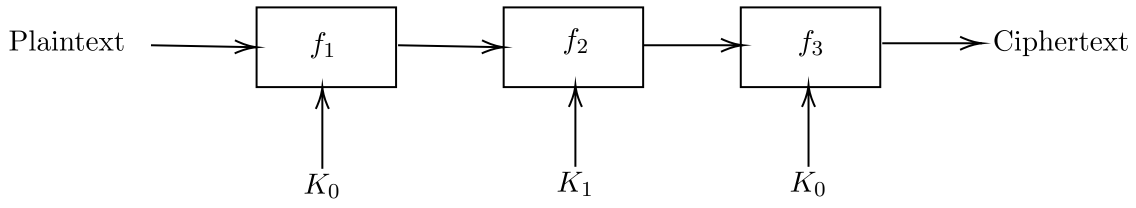


Figure 6: Meet in the middle attack (just a representation - not specific to DES)

With one valid plaintext and ciphertext pair in a known-plaintext model, the attacker performs encryption in reverse order on key K_1 (right to left) and encryption on key K_0 (left to right). If the results match, the keys are found. Let $Enc\_DES(P, K_i) = X_i$, and store (X_i, K_i) in Table 1. Similarly, let $Enc\_DES(C, K_j) = Y_j$, and store (Y_j, K_j) in Table 2. If X_i equals Y_j, then (K_i, K_j) is the secret key. The time complexity is approximately 2^{57} ($\sim 2^{56}$), as Table 1 and Table 2 are independent, and the attacker needs to search 2^{56} possibilities for both.

Key Length Increase: Increasing the length of the secret key from m bits to $2m$ bits, even in an encryption algorithm providing n bits of security, does not proportionally increase security to $2n$. Therefore, using two secret keys in double encryption does not result in a complexity of 2^{2n} but rather maintains a complexity of almost 2^n.

5.2 Triple DES



In the context of triple DES, where two secret keys K_0 and K_1 are employed, and three encryptions are performed, a diagram illustrates the process:

In a Meet-in-the-Middle attack on triple DES, decryption on K_1 is conducted for every possible value of K_0, as depicted in the figure. This approach multiplies the time complexity, resulting in 2^{2n} bit security, where 'n' is the size of the key.

6 Few Mathematical Things

Let's say we have an algorithm which claims n-bit security using exhaustive search (2^n). But if we have Quantum or supercomputers, time complexity will reduce to $2^{n/2}$.

To achieve n-bit security in this setup, we will use a $2n$-length key or a triple encryption setup.

6.1 Binary Operator

$$R \subseteq X \times Y$$

A binary operator $*$ on a set S is a mapping from $S \times S$ to S. In other words, $*$ is a rule that assigns to each ordered pair of elements from S an element of S.

$$* : S \times S \to S$$
$$*(a, b) = c, \text{ where } a, b, c \in S$$
$$*(b, a) = d, \text{ where } d \in S$$

It is not necessary that $d = c$.

6.2 Groups

A group $(G, *)$ consists of a set G with a binary operation $*$ on G satisfying the following three axioms.

(i) **Associativity:** $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.

(ii) **Identity Element:** There is an element $1 \in G$, called the identity element, such that $a * 1 = 1 * a = a$ for all $a \in G$.

(iii) **Inverse:** For each $a \in G$, there exists an element $a^{-1} \in G$, called the inverse of a, such that $a * a^{-1} = a^{-1} * a = 1$.

NOTE: A group G is abelian (or commutative) if, furthermore, $a * b = b * a$ for all $a, b \in G$.

Examples:

1. Matrix multiplication over square matrices of order $n \times n$:
   - It is not commutative: $A \cdot B$ is not equal to $B \cdot A$.
   - $(G, *) = \{\text{set of all invertible matrices}\}$
   - It satisfies all three group axioms but is not abelian.
     (a) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
     (b) $A \cdot I_n = I_n = I_n \cdot A$
     (c) $A \cdot A^{-1} = I_n = A^{-1} \cdot A$

     So, it is a group with a multiplication operator but is not abelian (commutative).

2. $(\mathbb{Z}, +)$: $\mathbb{Z}$ set of integers with the operation of addition

- It is commutative.
- Identity element: 0
- Inverse of $a \in \mathbb{Z}$: $-a$
- It forms a group under addition.
  (a) $a + (b + c) = (a + b) + c$ for all $a, b, c \in \mathbb{Z}$
  (b) 0: identity element, $a + 0 = a = 0 + a$, for all $a \in \mathbb{Z}$
  (c) For every $a \in \mathbb{Z}$, there exists $-a \in \mathbb{Z}$ such that: $a + (-a) = 0 = (-a) + a$

3. $(\mathbb{Z}, *)$: $\mathbb{Z}$ set of integers with the operation of multiplication
   - It is not a group as there is no inverse for every element.
     (a) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in \mathbb{Z}$
     (b) $a \cdot 1 = 1 \cdot a$
     (c) For $a \in \mathbb{Z}$, there does not exist $1/a\,(a^{-1}) \in \mathbb{Z}$

     So, set of integers $\mathbb{Z}$ with the operation of multiplication is not a group.

4. $(\mathbb{Z}, -)$: $\mathbb{Z}$ set of integers with the operation of subtraction
   - It is not a group due to a lack of associativity.
     (a) $(a - (b - c)) \neq (a - b) - c$

     So, it is not a group.

5. $(\mathbb{Q}, *)$: $\mathbb{Q}$ set of all rational numbers with multiplication operation
   - Additional information or examples are needed to analyze its group properties.

6. $(\mathbb{Q} - \{0\}, *)$: $\mathbb{Q}$ set of all rational numbers except 0 with multiplication operation
   - Yes, it is a group!

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 4)

1 Groups

We already saw what groups are. Let us quickly revise. A group $(G, *)$ consists of a set G with a binary operation $*$ on G satisfying the following three axioms:

1. **Associativity**: $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.
2. **Identity Element**: There is an element $1 \in G$, called the identity element, such that $a * 1 = 1 * a = a$ for all $a \in G$.
3. **Inverse**: For each $a \in G$, there exists an element $a^{-1} \in G$, called the inverse of a, such that $a * a^{-1} = a^{-1} * a = 1$.

Note: A group G is abelian (or commutative) if, furthermore, $a * b = b * a$ for all $a, b \in G$.

1.1 Examples

1. $(G, *)$ where G is the set of all invertible matrices.
2. $(\mathbb{Z}, +)$
3. $(\mathbb{Z}, *)$
4. $(\mathbb{Z}, -)$
5. $(\mathbb{Q}, *)$
6. $(\mathbb{Q} - \{0\}, *)$
7. $(\mathbb{Z}_n, +_n)$

   Let's check if $(\mathbb{Z}_n, +_n)$ is a group or not where the set $\mathbb{Z}_n$ contains integers from 0 to $n-1$ (inclusive), and the operation $+_n$ is defined as $x +_n y = (x + y) \mod n$.

   - Checking for associativity:

   $$\begin{aligned}(x +_n y) +_n z &= (((x + y) \mod n) + z) \mod n \\ &= (x + (y + z) \mod n) \mod n \\ &= x +_n (y +_n z)\end{aligned}$$

     Hence, $(\mathbb{Z}_n, +_n)$ is associative.
   - Checking for identity: 0 is the identity element as $x +_n 0 = x = 0 +_n x$.

- Checking for inverse: For any x in $\mathbb{Z}_n$, the inverse of x is $n - x$, since:

$$\begin{aligned} x +_n (n-x) &= (x + (n-x)) \mod n \\ &= n \mod n \\ &= 0 \end{aligned}$$

  Thus, every element in $\mathbb{Z}_n$ has an inverse.

Therefore, $(\mathbb{Z}_n, +_n)$ is a group, and it's also an abelian group.

8. Let's check if $(\mathbb{Z}_n - \{0\}, *_n)$ is a group or not where the operation $*_n$ is defined as $(a, b) *_n (c, d) = (a *_1 c) \mod n \times (b *_2 d) \mod n$.
   - This operation is associative on the given set.
   - Identity Element: The identity element is $(1, 1)$.
   - Inverse Element: The inverse of an element (x, y), denoted as (x^{-1}, y^{-1}), exists only if $\gcd(x, n) = 1$ and $\gcd(y, n) = 1$. Hence, not every element in $(\mathbb{Z}_n - \{0\}) \times (\mathbb{Z}_n - \{0\})$ has an inverse under this operation.

     Hence, $(\mathbb{Z}_n - \{0\}, *_n)$ is a group if $\gcd(x, n) = 1$.

1.2 Subgroups

A non-empty subset H of a group $(G, *)$ is a subgroup of G if H is itself a group with respect to the operation $*$ of G. If H is a proper subset and a group with respect to $*$ of G and $H \neq G$, then H is called a proper subgroup of $(G, *)$. H will have the following properties:

1. $H \subseteq G$
2. H is itself a group with $*$

Do note that $(G, *)$ is a group because $a \in G$, $a * a \in G$, $a * a * a \in G$,

Here $*$ is just a notation for the operation which would be performed $a^i = a * a * a \cdots a \in G$. A group G is cyclic if there is an element $\alpha \in G$ such that for every $b \in G$, there is an integer i with $b = \alpha^i$. This α is called the generator of $(G, *)$. Order of an element $a \in G$, $O(a)$, is the least positive integer m such that $a^m = e$ (where e is the identity element of G).

1.2.1 An Example

Given – $O(a) = 5$, $a^5 = e$. So, $S = \{e, a, a^2, a^3, a^4\}$ will be a subgroup of G as:

- $S \subseteq G$ and
- $(S, *)$ is a group since it is associative, commutative, and has an inverse ($a^{-1} = a^4$ and so on).
- All elements in S are generated by a only. So, a is a cyclic subgroup of G.

Note: Every subset of G is not necessarily a subgroup.

1.2.2 Cyclic Subgroups

If G is a group and $a \in G$, then the set of all powers of a will form a cyclic subgroup generated by a and denoted by $\langle a \rangle$. Let G be a group and $a \in G$ be an element of finite order t, then $|\langle a \rangle|$ denotes the size of the subgroup generated by a and equals t.

1.2.3 Lagrange's Subgroups

If G is a finite group and H is a subgroup of G, then $|H|$ divides $|G|$. Since the order of the element generating the subgroup is equal to the cardinality of the subgroup, therefore, the order of the element also divides $|G|$.

Let $a \in G$ and $O(a)$ be the order of element a. Therefore,

$$S = \{a^0, a^1, a^2, \ldots, a^{O(a)-1}\} = \langle a \rangle$$

$(S, *)$ is a subgroup of $(G, *)$.

From Lagrange's Theorem,

$$|H| \text{ divides } |G| \Rightarrow O(a) \text{ divides } |G|$$

1.3 Important Result

If the order of $a \in G$ is t, then the order of a^k is $t/\gcd(t, k)$.

$$\langle a \rangle = \{e, a, a^2, \ldots, a^{O(a)-1}\}$$

$$\langle at \rangle = \{e, ar, a^2t, \ldots, (at)^{O(at)-1}\}$$

$$B = at$$

$$\langle at \rangle = \langle b \rangle = \{e, b, b^2, \ldots, b^{O(b)-1}\}$$

If gcd(t, O(a)) = 1, then O(at) = O(a)

2 Ring

A ring $(R, +_R, \times_R)$ consists of one set R with two binary operations arbitrarily denoted by $+_R$ (addition) and $\times_R$ (multiplication) on R satisfying the following properties:

1. $(R, +_R)$ is an abelian group with the identity element 0_R.
2. The operation $\times_R$ is associative, that is,
$$a \times_R (b \times_R c) = (a \times_R b) \times_R c \text{ for all } a, b, c \in R$$
3. There is a multiplicative identity denoted by 1_R with $1_R \neq 0_R$ such that $1_R \times_R a = a \times_R 1_R = a$ for all $a \in R$.

4. The operation $\times_R$ is distributive over $+_R$, that is,

$$(b +_R c) \times_R a = (b \times_R a) +_R (c \times_R a)$$

$$a \times_R (b +_R c) = (a \times_R b) +_R (a \times_R c)$$

Note: We do not worry about the inverse of $\times_R$.

2.1 Examples

1. $(\mathbb{Z}, +, \cdot)$:
   - $(\mathbb{Z}, +)$: abelian group
     (a) Associativity: $a + (b + c) = (a + b) + c$
     (b) Identity Element: $a + 0 = a = 0 + a$ (0 is the identity element)
     (c) Inverse: $a + (-a) = 0 = (-a) + a$
     (d) Abelian Property: $a + b = b + a$ for all $a, b \in \mathbb{Z}$
   - $(\mathbb{Z}, \cdot)$:
   - Distributive property: $a \cdot 1 = 1 \cdot a = a$, where 1 is the identity element.
   - Distributive property: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   - Distributive property: $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$

2. $(\mathbb{R}, +_R, \times_R)$: $a \times_R b = b \times_R a$ for all $a, b \in \mathbb{R}$, hence it is a commutative ring.

3. $(\mathbb{Z}, +, \cdot)$: Commutative ring.

2.2 Units

An element a of a ring R is called a unit or an invertible element if there exists an element $b \in R$ such that $a \times_R b = 1_R$. (1 is the unit element in $(\mathbb{Z}, +, \cdot)$). The set of units in a ring R forms a group under the multiplication operation. This is known as the group of units of R. (Since, the inverse was missing and we added that as well).

3 Field

A field is a non-empty set F together with two binary operations, addition $(+)$ and multiplication $(\cdot)$, for which the following properties are satisfied:

1. $(F, +)$ is an abelian group.

2. If 0_F denotes the additive identity element of $(F, +)$, then $(F - \{0_F\}, \cdot)$ is an abelian group.

3. For all $a, b, c \in F$, we have $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

3.1 Examples

1. $(\mathbb{Z}, +, \cdot)$:
   - $(\mathbb{Z}, +)$ is an abelian group with identity element 0.
   - But for the set $\mathbb{Z} - \{0\}$, multiplicative inverse does not exist. Hence, $(\mathbb{Z} - \{0\}, \cdot)$ is not an abelian group.
   - Hence, $(\mathbb{Z}, +, \cdot)$ is not a field.
2. $(\mathbb{Q}, +, \cdot)$:
   - $(\mathbb{Z}, +)$ is an abelian group with identity element 0.
   - For the set $\mathbb{Q} - \{0\}$, multiplicative inverse exists for every rational number.
   - Multiplication is distributive over addition on rational numbers. Hence, $(\mathbb{Q}, +, \cdot)$ is a field.
3. $(\mathbb{F}_p, +_p, \cdot_p)$, where $\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$ and p is any prime number:
   - $+_p$: $(x + y) \mod p$ - Trivially, this is an abelian group.
   - $\cdot_p$: $(x \cdot y) \mod p$ - Remove 0 from $\mathbb{F}_p$ then $\gcd(x, p) = 1$ since p is prime. Also, it is trivial that $\cdot_p$ is distributive over $+_p$.
   - Hence $(\mathbb{F}_p, +_p, \cdot_p)$ is a field.

3.2 Field Extension

Suppose K_2 is a field with addition $(+)$ and multiplication $(\cdot)$. Suppose $K_1 \subseteq K_2$ is closed under both these operations such that K_1 itself is a field with the restriction of $+$ and $\cdot$ to the set K_1. Then K_1 is called a subfield of K_2 and K_2 is called a field extension of K_1.

4 Polynomial Ring

Let $(F, +, *)$ be a field. The set of polynomials of any degree $F[x]$ is defined as:

$$F[x] = \{a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots | a_i \in F\}$$

The polynomial ring, denoted as $F[x]$, consists of all polynomials in the variable x whose coefficients are elements of the field F. This ring is formed by combining the set of polynomials with the field's binary operations, thereby establishing a structure where polynomial addition and multiplication satisfy the ring properties.

$$(F[x], +, *) \rightarrow \text{Polynomial Ring}$$

Let $P_1(x) \in F[x] = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$

$P_2(x) \in F[x] = b_0 + b_1 \cdot x + \ldots + b_n \cdot x^n$

If we want to add the two polynomials:

$$P_1(x) + P_2(x) = (a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n) + (b_0 + b_1 \cdot x + \ldots + b_n \cdot x^n)$$

$$P_1(x) + P_2(x) = (a_0 + b_0) + (a_1 + b_1) \cdot x + \ldots + (a_n + b_n) \cdot x^n$$

Multiplication operation of the two polynomials:

$$P_1(x) * P_2(x) = (a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n) * (b_0 + b_1 \cdot x + \ldots + b_n \cdot x^n)$$

$$P_1(x) * P_2(x) = (a_0 * b_0) + (a_0 * b_1 + b_0 * a_1) \cdot x + \ldots + (a_n * b_n) \cdot x^n$$

Additive inverse of $P(x)$:

$$P(x) = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$$

$$P(-x) = -a_0 + (-a_1) \cdot x + \ldots + (-a_n) \cdot x^n$$

Clearly, $P(-x)$ is the additive inverse of $P(x)$. Here, the negative sign does not mean the standard negation.

A polynomial ring is formally defined as the set of all polynomials $F[x]$ along with the operations of addition $(+)$ and multiplication $(*)$, and it is referred to as a ring if it satisfies the following conditions:

1. $(F[x], +)$ is an abelian group.
2. $*$ is associative over $F[x]$.
3. An identity element over multiplication exists.
4. $*$ is distributive over $+$.

4.1 Example

Consider the set $F = \{0, 1\}$ and the field $(F, +_2, *_2)$. Therefore, the polynomial set $F_2[x]$ is:

$$F_2[x] = \{a_0 + a_1 \cdot x + \ldots | a_i \in F\}$$

Let us take two example polynomials:

$$p(x) = x + 1$$

$$q(x) = x^2 + x + 1$$

$$p(x) +_2 q(x) = (x + 1) +_2 (x^2 + x + 1) = x^2 + (1 +_2 1) \cdot x + (1 +_2 1) = x^2$$

$$p(x) *_2 q(x) = (x + 1) *_2 (x^2 + x + 1)$$

$$p(x) *_2 q(x) = (x^3 + x^2 + x) + (x^2 + x + 1) = x^3 + (1 +_2 1) \cdot x^2 + (1 +_2 1) \cdot x + 1$$

$$p(x) *_2 q(x) = x^3 + 1$$

Here, to get the coefficient of x^i, we will perform addition (of terms forming power i) or multiplication (of terms forming power i) modulo 2 operation.

5 Irreducible Polynomials

A polynomial $P(x) \in F[x]$ of degree $n \geq 1$ is called irreducible if it cannot be written in the form of $P_1(x) * P_2(x)$ with $P_1(x), P_2(x) \in F[x]$ and the degree of $P_1(x), P_2(x)$ must be greater than or equal to 1.

5.1 Important Property

$x^2 + 1$ belongs to $F_2[x]$.

$$(x+1) * (x+1) = x^2 + (1+1) * x + 1 = x^2 + 1.$$

Therefore, $x^2 + 1 = (x+1) * (x+1)$ in $F_2[x]$. Hence, $x^2 + 1$ is reducible in $F_2[x]$. Now, consider a set denoted by I, containing polynomials defined as:

$$I = \langle P(x) \rangle = \{q(x) * P(x) | q(x) \in F[x]\}$$

Also, consider the set denoted by $F[x]/\langle P(x) \rangle$, where each element is formed by dividing an element from $F[x]$ by $P(x)$.

For any $q(x) \in F[x]$, there exist polynomials $d(x)$ and $r(x)$ such that:

$$q(x) = d(x) * P(x) + r(x)$$

where $r(x) \in F[x]/\langle P(x) \rangle$.

Also, if $P(x)$ is an irreducible polynomial, then $(F[x]/\langle P(x) \rangle, +, *)$ forms a field. In this context, addition and multiplication operations are performed modulo $P(x)$. Notably, the degree of $r(x)$ is always less than the degree of $P(x)$.

5.2 Examples

1. $x^2 + 1$ in $\mathbb{R}[x]$:
   - It is not possible to factor $x^2 + 1$ in $\mathbb{R}[x]$, where $\mathbb{R}$ is the set of real numbers.
   - Let $P(x) = q_1(x) \cdot q_2(x)$.
   - $\text{Deg}(q_1) \geq 1$
   - $\text{Deg}(q_2) \geq 1$
   - $x^2 + 1 = 0$
   - $x^2 = -1$
   - $x = \pm i$
   - $(x + \alpha)$ and $(x - \alpha)$
   - It is not a reducible polynomial because to reduce, it would result in $(x + i)$ and $(x - i)$ which are complex numbers, but it is in $\mathbb{R}$ (Real numbers). So, it is irreducible.
2. $x^2 + x + 1$ in $\mathbb{F}_2[x]$, where $\mathbb{F}_2 = \{0, 1\}$:
   - The polynomial $P(x) = x^2 + x + 1$ is irreducible.
   - We will put $x = 0$ and $x = 1$:
   - $P(0) = 1$
   - $P(1) = 1$
   - So, $(x + 0)$ and $(x + 1)$ are not factors of $P(x)$. There are no degree 1 factors of this $P(x)$. Hence, it is irreducible.

Consider the set $\mathbb{F}_2[x]/\langle x^2+x+1\rangle$: For any polynomial $q(x)$, we can express it as:

$$q(x) = d(x) \cdot P(x) + r(x)$$

where $\deg(d(x)) < 2$ and $\deg(r(x)) < 2$. The possible remainders $r(x)$ can be $\{0, 1, x, x+1\}$. If $P(x)$ is an n-degree polynomial under modulo 2, then there will be 2^{2n} polynomials in $r(x)$, meaning $\mathbb{F}_2[x]/\langle x^2+x+1\rangle$ will have $2^2 = 4^2 = 16$ polynomials.

6 Primitive Polynomials

Consider the set $\mathbb{F}_2[x]/\langle x^2+x+1\rangle$. We've established that if $P(x)$ is irreducible, then $(\mathbb{F}_2[x]/\langle x\rangle, +, *)$ forms a field. Now, suppose α is a root of $x^2+x+1=0$, i.e., $\alpha^2+\alpha+1=0$. This implies $\alpha^2 = -\alpha-1 = \alpha+1$. If α can generate all possible polynomials in $\mathbb{F}_2[x]/\langle x^2+x+1\rangle$, then x^2+x+1 is termed a primitive polynomial.

Let's demonstrate this:

$$\langle\alpha\rangle = \{0, 1 = \alpha^0, \alpha, \alpha+1 = \alpha^2\}$$

The order of α, denoted as $O(\alpha)$, is 2. Hence, x^2+x+1 is a primitive polynomial.

6.1 Example

Consider $\mathbb{F}_2[x]/\langle x^3+x+1\rangle$. The maximum number of polynomials that can be generated: $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$.

Let's check if the root of $x^3+x+1=0$ is a generator:

$$\alpha^3+\alpha+1 = 0 \Rightarrow \alpha^3 = \alpha+1$$

$$\langle\alpha\rangle = \{0, 1 = \alpha^0, \alpha, \alpha^2, \alpha+1 = \alpha^3, \alpha^2+\alpha = \alpha^4, \alpha^2+\alpha+1 = \alpha^5, \alpha^2+1 = \alpha^6\}$$

Since we can generate all the polynomials, x^3+x+1 is a primitive polynomial.

Note that there may exist a polynomial that is not primitive but still forms a field. That implies we can find a multiplicative inverse. Consider the polynomial αx. Instead of $1/\alpha x$, we have the polynomial α^2+1/x^2+1, which results in 1 on multiplication.

$$x \cdot (x^2+1) = x^3+x = x+1+x = 1$$

Similarly, for x^2, the multiplicative inverse is x^2+x+1.

$$x^2 \cdot (x^2+x+1) = x^4+x^3+x^2 = x\cdot(x+1)+(x+1)+x^2$$

$$x^2 \cdot (x^2+x+1) = x^2+x+x+1+x^2 = 1$$

7 AES (Advanced Encryption Standard)

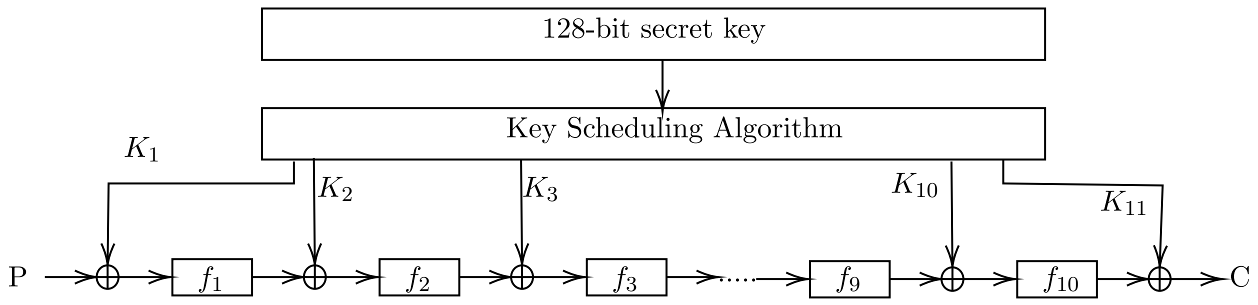
After DES was found to be vulnerable once it was released to the public, a new competition was held named AES to find a better cipher. In the competition, Rindel was the winning cipher and hence according to the rules of the competition, it was renamed to AES. AES is a NIST Standardized iterated block cipher and a substitution permutation network (SPN) as well.

Let us see the variations of AES:

1. **AES-128**:
    - Block size – 128 bit
    - Number of rounds – 10
    - Secret Key size – 128
2. **AES-192**:
    - Block size – 128 bit
    - Number of rounds – 12
    - Secret Key size – 192
3. **AES-256**:
    - Block size – 128 bit
    - Number of rounds – 14
    - Secret Key size – 256

Note: In all these three, only the number of rounds and the secret key size change!

7.1 Structure of AES



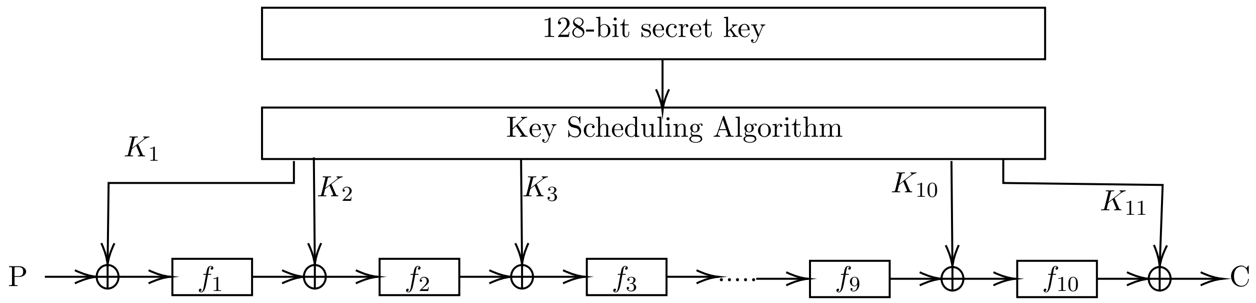
Important Observations -

- 10 Rounds
- 11 Keys Generated
- Ciphertext also of 128 bits

[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 5)



AES Structure

1 Components of AES

1.1 Round Function of AES

The Advanced Encryption Standard (AES) employs a round function to iteratively transform plaintext blocks into ciphertext blocks. This round function, denoted as f, comprises various operations applied sequentially, including substitution, permutation, and mixing. Each round function f_i, where i ranges from 1 to 10, maps from a 128-bit input to a 128-bit output:

$$f_i : \{0,1\}^{128} \to \{0,1\}^{128} \quad \text{for} \quad 1 \leq i \leq 10$$

The Subbytes operation is a bijective mapping from 128 bits to 128 bits:

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

1.1.1 Substitution (SubBytes)

The SubBytes step of AES replaces each byte in the state matrix with another byte using a substitution table called the S-box. This table performs a non-linear byte substitution, which helps enhance the confusion property of AES.

For a byte at row i and column j of the state matrix S, denoted as $S_{i,j}$, the substitution is calculated as:

$$\text{SubBytes}(S)_{i,j} = \text{S-box}(S_{i,j})$$

The input S to the SubBytes function is a 128-bit binary input. A 4×4 matrix can be constructed from this input by arranging the bits in a specific manner.

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

where S_{ij} is a byte (8-bits). Consider the 128-bit plaintext of 128-bit. The plaintext again can be written as a 4×4 matrix in the following way. Keep in mind, the ordering of the plaintext bytes.

$$P = P_0P_1P_2....P_{15}, \text{ length of each } P_i \text{ is 8-bit}$$

$$P \rightarrow \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix}$$

Similarly, the first round key K_1 can also be written as a matrix.

$$K_1 = K_0K_1K_2....K_{15}, \text{ length of each } K_i \text{ is 8-bit}$$

$$K_1 \rightarrow \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix}$$

For AES-128, we first xor the plaintext with the first round key K_1. The output is then passed to first round function, wherein, it is first passed to subbytes function.

$$S = (S_{ij})_{4\times4} = P \oplus K_1$$

The output after the subbyte function is performed on S is S'.

$$S' = Subbytes(S)$$

We will see an overview of the subbyte function. Below are the steps involved:

1. A constant C = $C_7C_6C_5C_4C_3C_2C_1C_0 = (01100011) = (63)_{16}$ is declared.
2. Substitution box $\mathbb{S}$ maps every element of the S matrix from 8-bit to 8-bit. This $\mathbb{S}$ box is applied to every element of S matrix, i.e., S_{ij}. Therefore, it is an 8-bit to 8-bit mapping. Also, $\mathbb{S}(0) = 0$ is taken as a rule (fixed for AES).
3. Suppose $\mathbb{S}(S_{ij}) = m_7m_6m_5m_4m_3m_2m_1m_0$.
4. For $i = 0$ to $i = 7$, compute b_i as:

$$b_i = (m_i + m_{(i+4)\%8} + m_{(i+5)\%8} + m_{(i+6)\%8} + m_{(i+7)\%8} + C_i)\%2$$

5. Therefore, the output is $b_7b_6b_5b_4b_3b_2b_1b_0$.
6. $S'_{ij} = (b_7b_6b_5b_4b_3b_2b_1b_0)$

Hence, S'_{ij} is computed for each S_{ij}, and the output matrix is the output of the subbyte function.

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

Now, we will discuss the substitution box $\mathbb{S}$ that takes 8-bit input and produces 8-bit output.

$$\mathbb{S} : \{0,1\}^8 \to \{0,1\}^8 \text{ and } \mathbb{S}(0) = 0$$

Let's say input X is given to this $\mathbb{S}$ box and $X \neq 0$. We need to find $Y = \mathbb{S}(X)$.

$$X = a_7a_6a_5a_4a_3a_2a_1a_0, \text{ where } a_i \in \{0,1\}$$

We can construct a polynomial $P(x)$ using the bits of X as coefficients.

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_7 \cdot x^7$$

Clearly, degree of $P(x) \leq 7$. Also, $P(x) \in F_2[x]$. Moreover, since $X \neq 0 \implies P(x) \neq 0$. Another polynomial $g(x) = x^8 + x^4 + x^3 + x + 1$ is fixed for AES. $g(x)$ is a primitive polynomial.

Given that $g(x)$ is a primitive polynomial, it defines a field denoted by $(\mathbb{F}_2[x]/\langle g(x)\rangle, +, *)$, where $+$ and $*$ represent addition and multiplication modulo $g(x)$, respectively. All polynomials within this field have a maximum degree of 7. Consequently, $P(x)$ is a member of this field. Moreover, since $\langle g(x)\rangle$ is primitive, we can ascertain the multiplicative inverse of any polynomial in $(\mathbb{F}_2[x]/\langle g(x)\rangle)$. Hence, we aim to determine the multiplicative inverse of $P(x)$ modulo $g(x)$, denoted as $q(x)$.

$$\begin{gathered} P(x) \cdot q(x) \equiv 1 mod\ g(x) \\ P(x) \cdot q(x) \equiv 1 mod(x^8 + x^4 + x^3 + x + 1) \\ P(x) \cdot q(x) - 1 = h(x) \cdot (x^8 + x^4 + x^3 + x + 1) \\ 1 = P(x) \cdot q(x) + h(x) \cdot (x^8 + x^4 + x^3 + x + 1) \end{gathered}$$

Therefore, we can find the $q(x)$ with the help of Extended Euclidean Algorithm. Also, $q(x)$ will be a polynomial of degree at most 7.

$$q(x) = r_0 + r_1 \cdot x + r_2 \cdot x^2 + ... + r_7 \cdot x^7, \text{ where } r_i \in \{0,1\}$$

From the coefficients, we can build a 8-bit binary string as $r_7r_6r_5r_4r_3r_2r_1r_0$. This string is the output of the $\mathbb{S}$ box. Therefore,

$$\mathbb{S}(X) = Y = (r_7r_6r_5r_4r_3r_2r_1r_0)$$

Example: Find Subbytes(01010011).

Solution: X = 01010011, therefore $P(x) = x^6 + x^4 + x + 1$. Also, $g(x) = x^8 + x^4 + x^3 + x + 1$. Now, let's perform the Extended Euclidean Algorithm.

$$\begin{array}{l}
x^6+x^4+x+1 \,\big)\; x^8+x^4+x^3+x+1 \;\big(\, x^2+1 \\
\qquad\qquad\qquad\;\; x^8+x^6+x^3+x^2 \\
\hline
\qquad\qquad\qquad\;\; x^6+x^4+x^2+x+1 \\
\qquad\qquad\qquad\;\; x^6+x^4+x+1 \\
\hline
\qquad\qquad\qquad\qquad\qquad x^2 \,\big)\; x^6+x^4+x+1 \;\big(\, x^4+x^2 \\
\qquad\qquad\qquad\qquad\qquad\quad\; x^6 \\
\hline
\qquad\qquad\qquad\qquad\qquad\qquad x^4+x+1 \\
\qquad\qquad\qquad\qquad\qquad\qquad x^4 \\
\hline
\qquad\qquad\qquad\qquad\qquad\qquad\quad x+1 \,\big)\; x^2 \;\big(\, x+1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad x^2+x \\
\hline
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x+1 \\
\hline
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\;\; 1 \\
\hline
\end{array}$$

Now, let's work upside down to find the multiplicative inverse. Therefore, $1 = q(x) \cdot P(x) = h(x) \cdot g(x)$

$$\begin{aligned}
&1 = 1 \cdot x^2 + (x+1) \cdot (x+1) \\
&1 = x^2 + (x+1) \cdot [(x^6+x^4+x+1) + x^2 \cdot (x^4+x^2)] \\
&1 = (x+1) \cdot (x^6+x^4+x+1) + x^2 \cdot [1 + (x+1) \cdot (x^4+x^2)] \\
&1 = (x+1) \cdot (x^6+x^4+x+1) + x^2 \cdot (x^5+x^4+x^3+x^2+1) \\
&1 = (x+1) \cdot (x^6+x^4+x+1) + (x^5+x^4+x^3+x^2+1) \cdot [(x^8+x^4+x^3+x+1) + (x^6+x^4+x+1) \cdot (x^2+1)] \\
&1 = [(x+1) + (x^2+1) \cdot (x^5+x^4+x^3+x^2+1)] \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x) \\
&1 = (x+1+x^7+x^6+x^5+x^4+x^2+x^5+x^4+x^3+x^2+1) \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x) \\
&1 = (x^7+x^6+x^3+x) \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x)
\end{aligned}$$

Therefore, Multiplicative Inverse of P(x) is $q(x) = x^7 + x^6 + x^3 + x$. Therefore,

$$\mathbb{S}(01010011) = (11001010) = m_7m_6m_5m_4m_3m_2m_1m_0$$

Now, let's compute $b_7b_6b_5b_4b_3b_2b_1b_0$. The constant C = $c_7c_6c_5c_4c_3c_2c_1c_0 = 01100011$.

	0	1	2	3	4	5	6	7
c	1	1	0	0	0	1	1	0
m	0	1	0	1	0	0	1	1

$$\begin{gathered}
b_0 = (0+0+0+1+1+1)\%2 = 1 \\
b_1 = (1+0+1+1+0+1)\%2 = 0 \\
b_2 = 1, b_3 = 1, b_4 = 0 \\
b_5 = 1, b_6 = 1, b_7 = 1
\end{gathered}$$

$$\therefore\ subbytes(01010011) = b_7b_6b_5b_4b_3b_2b_1b_0 = (11101101)$$
$$subbytes(53) = (ED)$$

The value returned by the *subbyte* function is a hexadecimal number. The first four digits give the row number, the next four digits give the column numbers, and together they highlight the cell in a 16x16 table, which contains the required encrypted characters.

$$\text{Input} = \text{XY}$$

$$\text{Subbyte(Input)} \rightarrow \text{element present in row number X and column number Y}$$

The lookup table is given in the diagram below -

Input	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
01	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
02	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
03	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
04	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
05	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
06	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
07	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
08	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
09	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
0A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
0B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
0C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
0D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
0E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
0F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

1.1.2 Permutation (ShiftRows)

In the ShiftRows step, bytes in each row of the state matrix are cyclically shifted to the left. This permutation operation provides diffusion, spreading the influence of each byte throughout the state matrix.

$$\text{ShiftRows}(S) = \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{pmatrix}$$

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.

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

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

1.1.3 Mixing (MixColumns)

MixColumns is a linear transformation that operates on each column of the state matrix independently. It involves multiplying each column by a fixed matrix and then applying modular polynomial arithmetic. This mixing operation increases the diffusion further.

$$\text{MixColumns}(S) = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix}$$

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).

$$\text{Mix Columns: } \{0,1\}^{128} \to \{0,1\}^{128}$$

$$(S_{ij})_{4\times4} \xrightarrow{MixColumns} (S'_{ij})_{4\times4}$$

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:

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

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:

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

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.

1.1.4 Combining everything together!

The round function F is a composition of these three operations:

$$f(S) = \text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(S)))$$

For the first 9 round functions:

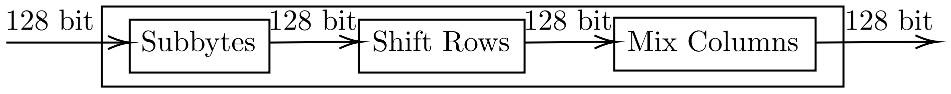
$$f_i(X) = MixColumns(ShiftRows(Subbytes(X))) \ \forall\ 1 \leq i \leq 9$$

However, the last round function f_{10} is based on subbytes and shift rows only. Therefore,

$$f_{10}(X) = ShiftRows(Subbytes(X))$$

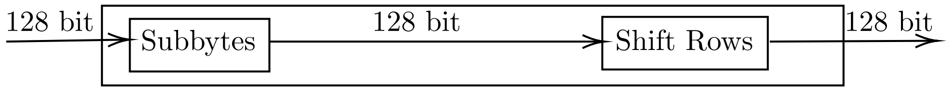
Each Subbyte, Shift Rows and Mix Columns is a bijection from 128-bit to 128-bit. This means each of them has existing inverse. Therefore, given $y = f_i(X)$, to get X, apply inverse of mix columns, then apply inverse of Shift Rows and then apply inverse of Subbytes.

For the first 9 rounds $f_1, f_2, ..., f_9$:



$f_i, 1 \leq i \leq 9$

For the last round function f_{10}:



f_{10}

1.2 Key Scheduling Algorithm of AES-128

The AES-128 key scheduling algorithm operates on a 128-bit encryption key, producing 11 distinct round keys, each consisting of 128 bits. The key is represented as $(key[0], key[1], ..., key[15])$, where each $key[i]$ is a single byte. This process involves generating 44 words, each 32 bits long, denoted as $w[0], w[1], ..., w[43]$. Notably, $\frac{32\times 44}{128} = 11$, providing the necessary material for creating 11 round keys from these 44 words.

The AES-128 key scheduling algorithm comprises two essential functions:

1. **ROTWORD($B_0B_1B_2B_3$):** This function takes a word as input and performs a left circular shift of its constituent bytes, equivalent to a rotation by 8 bits. The input word, represented by B_0, B_1, B_2, B_3, undergoes transformation, resulting in the output: ROTWORD$(B_0B_1B_2B_3) = B_1B_2B_3B_0$.
2. **SUBWORD($B_0B_1B_2B_3$):** This function operates on a word, initiating the Subbyte operation, which is crucial to the round function of AES, on each of its constituent bytes B_0, B_1, B_2, B_3. Consequently, the output of the SUBWORD function takes the form: SUBWORD$(B_0B_1B_2B_3) = B_0'B_1'B_2'B_3'$, where $B_i' = \text{Subbytes}(B_i)$ for all $i \in \{0, 1, 2, 3\}$.

In the AES-128 key scheduling algorithm, ten round constants are employed, each represented as a 32-bit word in hexadecimal format.

In the AES-128 key scheduling algorithm, ten round constants are utilized, each represented as a 32-bit word in hexadecimal format:

$$\begin{gathered}
RCON[1] = 0x01000000 \\
RCON[2] = 0x02000000 \\
RCON[3] = 0x04000000 \\
RCON[4] = 0x08000000 \\
RCON[5] = 0x10000000 \\
RCON[6] = 0x20000000 \\
RCON[7] = 0x40000000 \\
RCON[8] = 0x80000000 \\
RCON[9] = 0x1b000000 \\
RCON[10] = 0x36000000
\end{gathered}$$

These round constants play a crucial role in the key expansion process, contributing to the generation of 44 words, each 32 bits in length. The remaining portion of the algorithm is as follows:

```
for $i \leftarrow 0$ to 3 do
end
$w_i \leftarrow key[4i]||key[4i+1]||key[4i+2]||key[4i+3]$;
for $i \leftarrow 4$ to 43 do
end
temp = w[i-1]; if $i\%4 == 0$ then
end
temp = SUBWORD(ROTWORD(temp)) $\oplus$ RCON[i/4]; $w[i] = w[i-4] \oplus$ temp;
```

The resulting 44 words are then utilized to derive the round keys:

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

During decryption, the inverse of the key scheduling algorithm is unnecessary, as the round keys are XORed only, ensuring the same round keys can be utilized for encryption and decryption.

1.3 Decryption of AES

In AES decryption, the inverse of the round function is crucial. For rounds 1 to 9, the inverse function is defined as follows:

$$f_i^{-1} = \text{Subbyte}^{-1}(\text{ShiftRows}^{-1}(\text{MixColumns}^{-1}(S))) \text{ for } i \in \{1, 2, \dots, 9\}$$

For the 10th round function, the inverse is slightly different:

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

Inverse Subbyte Function:

The Subbyte function maps an 8-bit input to another 8-bit output using a lookup matrix. To invert this process, the input is divided into two halves to obtain the original value from the lookup matrix.

$$\text{Subbyte}^{-1}(B) = \text{InverseSubbyte}(B) = \text{InverseSubbyte}(X'||Y') = X||Y = A$$

Inverse Shift Row Function:

To reverse the Shift Row operation, the rows are right-circular shifted by the same number of positions they were left-shifted during encryption.

Inverse Mix Column Function:

The Mix Column function is essentially a matrix multiplication, and its inverse can be obtained by multiplying the result with the inverse of the mixing matrix.

$$\text{MixColumns}^{-1}(S) = S' \cdot M^{-1} = S$$

These inverse functions are essential for decrypting data encrypted with AES.

2 Mode of Operation

When employing AES for encryption, data exceeding the 128-bit block size necessitates specific modes of operation. These modes extend the capability to encrypt larger datasets efficiently. Notable modes include:

1. Electronic CodeBook Mode (ECB)
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)

In this discussion, focus will be placed on Electronic CodeBook (ECB) and Cipher Block Chaining (CBC) modes.

Consider a scenario where AES encryption is applied to 256-bit data. Initially, the data is divided into 128-bit blocks, with subsequent encryption applied to each block individually. Concatenating the resulting ciphertext yields the encrypted form of the entire 256-bit data. However, when applied to larger data sets, such as a 200kB image, ECB mode exhibits a notable drawback. Identical components within the data, like corresponding features in an image, lead to identical ciphertext segments. Consequently, patterns in the plaintext become discernible from the ciphertext, compromising security.

2.1 Electronic CodeBook Mode

In Electronic CodeBook (ECB) mode, the plaintext undergoes segmentation into continuous blocks of l-bit size, with each block encrypted independently. Concatenating the resulting ciphertext blocks in the same order yields the final ciphertext.

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

Encryption:

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

Decryption:

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

While ECB mode facilitates parallel encryption of multiple blocks, its susceptibility to information leakage arises when identical plaintext blocks result in identical ciphertext blocks, i.e., $m_i = m_j$ implies $C_i = C_j$.

2.2 Cipher Block Chaining Mode

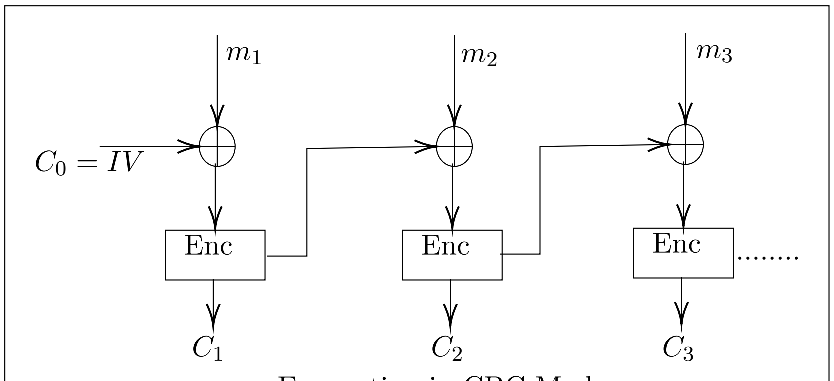
Cipher Block Chaining (CBC) mode is a widely used method in cryptography. It relies on an l-bit public block called the Initialization Vector (IV), which is crucial for the encryption process.

Encryption:

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

In CBC mode, the ciphertext comprises (n+1) blocks for a plaintext of n blocks. The encryption process unfolds as follows:

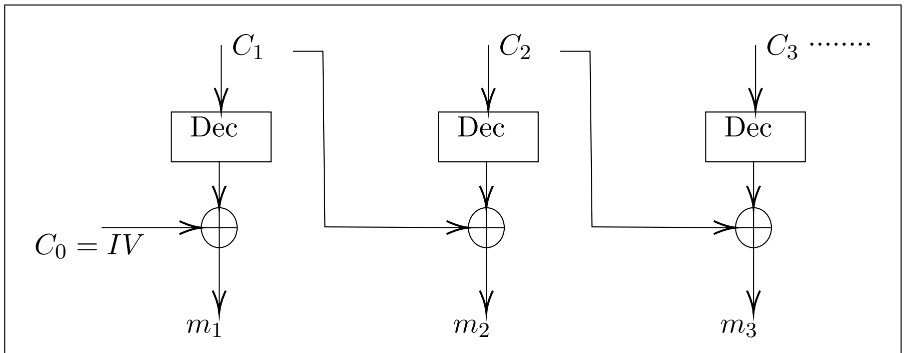
$$C_0 = IV$$
$$C_i = \text{Enc}(C_{i-1} \oplus m_i, K) \quad \forall \quad i \in \{1, 2, 3...., t\}$$
$$C = C_0||C_1||.....||C_t$$



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

3 Stream Cipher

Stream ciphers operate on the level of individual bits, encrypting data bit by bit. Let $M = m_0m_1...m_l$, where $m_i \in \{0, 1\}$ represents each bit of the plaintext message. The encryption process involves combining each plaintext bit m_i with a corresponding keystream bit k_i generated by the stream cipher algorithm, resulting in the ciphertext C:

$$C = M \oplus K = (m_0 \oplus k_0)(m_1 \oplus k_1)...(m_l \oplus k_l)$$

Where $\oplus$ denotes the bitwise XOR operation.

Encryption: $C_i = m_i \oplus K_i$

Decryption: $m_i = C_i \oplus K_i$

3.1 Shannon's Notion of Perfect Secrecy

Shannon's Theorem of Perfect Secrecy stands as a fundamental principle within cryptography, rigorously defining an algorithm's perfect security as one where the ciphertext divulges no additional information about the plaintext, irrespective of an adversary's computational capabilities.

Mathematically, Shannon's theorem can be expressed as follows:

The probability of a specific plaintext message m_1 occurring, denoted as $Pr[M = m_1]$, remains unchanged even when conditioned on a particular ciphertext CH_1, denoted as $Pr[M = m_1|C = CH_1]$. This equality signifies that the ciphertext does not offer any extra insight into the likelihood of m_1.

For illustration, let's consider binary variables m and k, where m (belonging to the set $\{0, 1\}$) represents the plaintext bit and k (also from the set $\{0, 1\}$) denotes the corresponding key bit. Assuming the probabilities of m and k being 0 are P and $\frac{1}{2}$ respectively, and similarly for 1, we can compute the probabilities of the resulting ciphertext C as follows:

$$
\begin{aligned}
Pr[C = 0] &= Pr[M = 0 \wedge K = 0] + Pr[M = 1 \wedge K = 1] \\
&= P \times \frac{1}{2} + (1 - P) \times \frac{1}{2} \\
&= \frac{1}{2} \\
Pr[C = 1] &= Pr[M = 0 \wedge K = 1] + Pr[M = 1 \wedge K = 0] \\
&= P \times \frac{1}{2} + (1 - P) \times \frac{1}{2} \\
&= \frac{1}{2}
\end{aligned}
$$

This demonstrates that the resulting ciphertext C is uniformly distributed. Thus, even if the plaintext displays bias, the ciphertext remains effectively randomized, thereby preserving perfect secrecy.

3.2 Conditions for Perfect Secrecy

In order for a stream cipher to achieve perfect secrecy, it must fulfill the following criteria:

1. **Unique Keys:** Each message must be encrypted using a different key. Reusing the same key for multiple messages can lead to potential information leakage, as patterns in the ciphertext may reveal aspects of the plaintext.

2. **Key Length vs. Message Length:** The length of the key ($|K|$) should be equal to or greater than the length of the message ($|M|$). If the key length is shorter than the message length ($|K| < |M|$), padding the key with repeated bits to match the message length can introduce vulnerabilities, potentially exposing information about the plaintext.

Adhering to these conditions ensures that the stream cipher maintains perfect secrecy, preventing any inference about the plaintext from the ciphertext, even in the presence of adversaries with significant computational resources.

For example, the Vernam cipher, also known as the **One-Time Pad (OTP)**, achieves perfect secrecy by using each bit of the key only once. However, OTP is impractical due to the challenges of securely sharing keys that are as long as the messages themselves.

[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.

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

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.

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

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: $C_i = h(Z_i, m_i)$

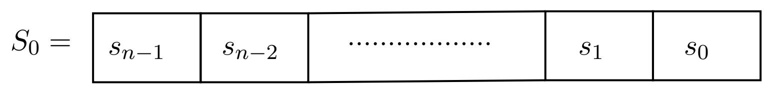
Here, $\sigma_0 = (C_{-t}, C_{-t+1}, ..., C_{-1})$ is the non-secret initial state.

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.

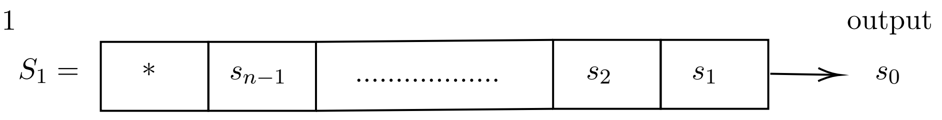
A register of length n implies it is an n-bit Linear Feedback Shift Register (LFSR), indicating it has a state of length n. At clock cycle $t = 0$, the register's state is labeled as S_0, where each bit s_i for $0 \leq i \leq n-1$ can take values from the set $\{0, 1\}$.

t = 0, t → clocking number



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

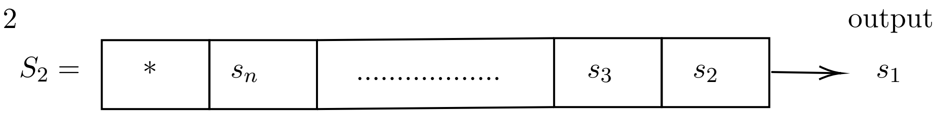
t = 1



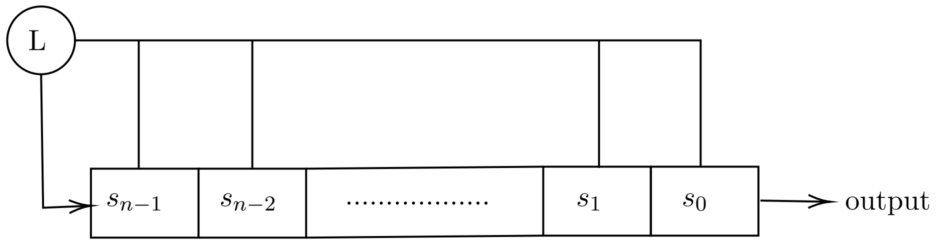
The rightmost bit s_0 serves as the output (the keystream bit), while the leftmost bit s_n becomes empty. The leftmost bit, s_n, is referred to as the feedback bit, and its value is determined by a function L applied to the entire state S_0, calculated as:

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

t = 2



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. $L = s_0 \oplus s_2$

$$L = s_0 \oplus s_2$$

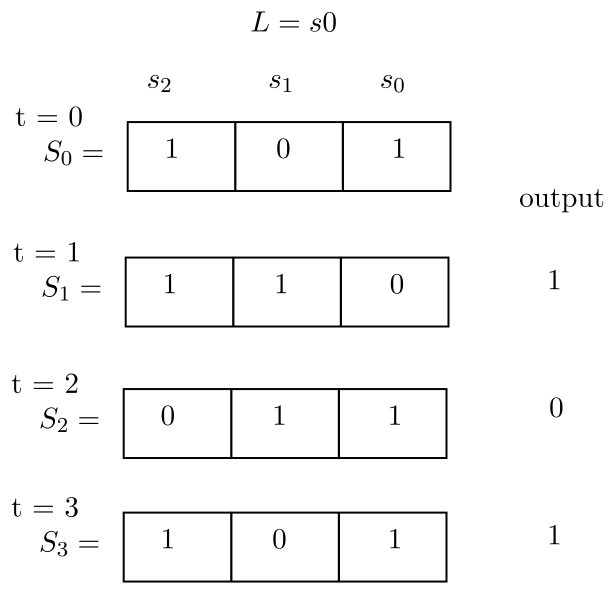
		s_2	s_1	s_0	output
t = 0	$S_0 =$	1	0	1	
t = 1	$S_1 =$	0	1	0	1
t = 2	$S_2 =$	0	0	1	0
t = 3	$S_3 =$	1	0	0	1
t = 4	$S_4 =$	1	1	0	0
t = 5	$S_5 =$	1	1	1	0
t = 6	$S_6 =$	0	1	1	1
t = 7	$S_7 =$	1	0	1	1

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

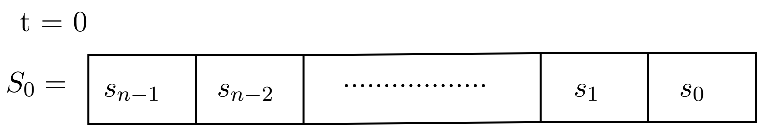
If S_0 represents a non-zero state and S_0 recurs after m clock cycles of the LFSR, then m defines the period of the LFSR. An LFSR comprising n bits can achieve a maximum period of $2^n - 1$.

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:

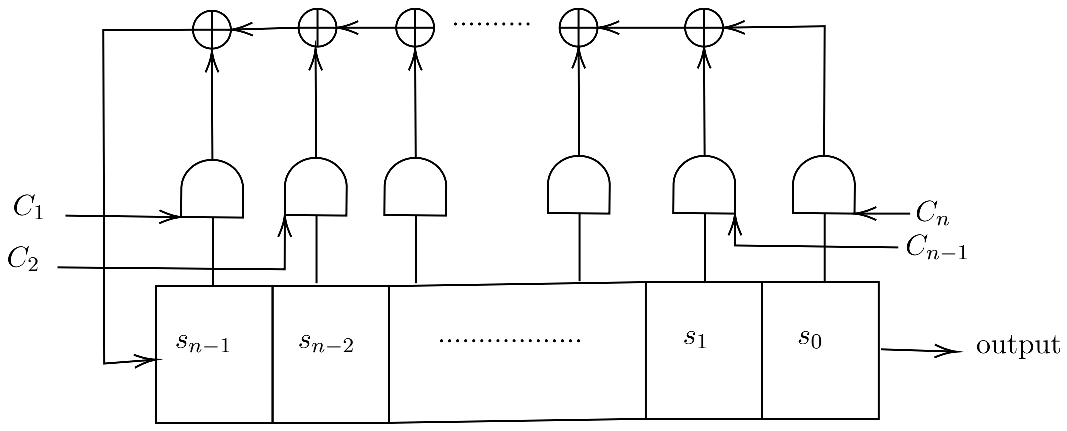


After 1 clocking, s_n will be:

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

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

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 $1 \le i \le n$, 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 $\le n$

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 $x_i \rightarrow$ key stream bits Z_i

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

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

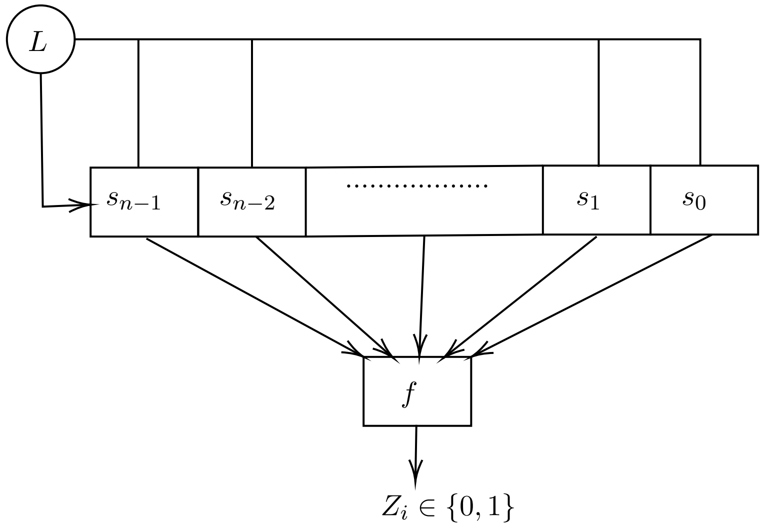
$Z_0, Z_1, \ldots, Z_{n-1}$

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

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.



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

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.

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

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

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

Suppose the shifting to be right shift, therefore,

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

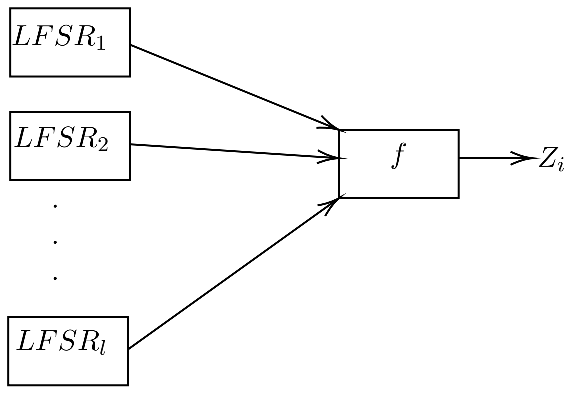
The state update can be represented as a matrix multiplication in the following way,

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

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

3.4 LFSR With Combiner Function

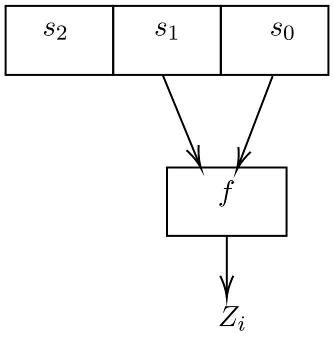
We utilize a function f resembling the one described earlier. However, in this scenario, we incorporate l Linear Feedback Shift Registers (LFSRs). The outputs of these l LFSRs, forming l bits, are inputted into the combiner function f, resulting in an output denoted as Z_i. It's important to emphasize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

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

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 $h : P \to S$ and $y \in S$, find $x \in P$ such that $h(x) = y$.
2. **Second Pre-Image Finding Problem:** Given a hash function $h : P \to S$, $x \in P$, 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 $h : P \to S$ 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 $x$
$h : X \to Y$
Choose any $X_0 \subseteq X$ such that $|X_0| = Q$ for each $x \in X_0$
Compute $y_x = h(x)$
if $y_x = y$ 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

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

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

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

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,

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

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

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 7 & 8)

1 Hash Function

In the field of cryptography, a hash function is a mathematical process that accepts an input, or "message", and outputs a fixed-length string of bytes, usually in the form of a hash value or hash code. Often called a digest, the output is a distinct representation of the input data. Fast and effective hash functions offer a safe and dependable means of confirming the integrity of data, authenticating communications, and creating digital signatures.

A hash family is a four-tuple (X, Y, K, H), where:

1. X is a set of possible messages.
2. Y is a finite set of possible message digests or authentication tags (or just tags).
3. K, the keyspace, is a finite set of possible keys.
4. For each $k \in K$, there is a hash function $h_k \in H$, where $h_k : X \to Y$.

While Y is always a finite set in the definition above, it may not always be a finite set. The function is sometimes referred to as a compression function if X is a finite set and $|X| > |Y|$. In this case, we'll assume the more favorable circumstance. $|X| > 2^{|Y|}$.

1.1 Types of Hash Functions

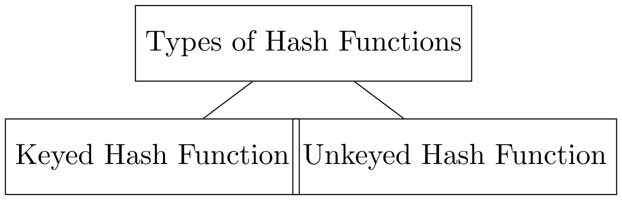


Figure 1: Types of Hash Functions

1.1.1 Unkeyed Hash Function

A function $h : X \to Y$, where X and Y are the same, is an unkeyed hash function. An unkeyed hash function can be conceptualized as a hash family where $|K| = 1$, or one with a single potential key. The output of an unkeyed hash function is commonly referred to as a "message digest."

1.1.2 Keyed Hash Function

If the key is involved in the computation of hashed value then that hash function is known as keyed hash function. The output of a keyed hash function is referred to as a "tag."

1.2 Legitimacy Under Hash Function

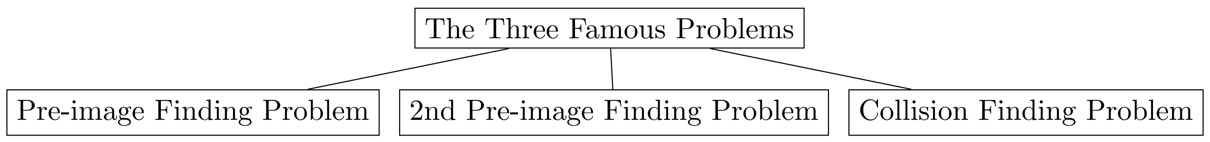
If $h(x) = y$, then a pair $(x, y) \in X \times Y$ is considered legitimate under a hash function h. In this case, h may be an unkeyed or keyed hash function. In this chapter, we mainly cover techniques to stop an opponent from creating specific kinds of valid pairs.

Let $F_{X,Y}$ denote the set of all functions from X to Y. Suppose that $|X| = N$ and $|Y| = M$. Then it is clear that $|F_{X,Y}| = M^N$. Any hash family F consisting of functions with domain X and range Y can be considered to be a subset of $F_{X,Y}$, i.e., $F \subseteq F_{X,Y}$. Such a hash family is termed an (N, M)-hash family.

1.3 Ideal Hash Function

Let $h : P \to S$. h is ideal if, given $x \in P$, to find $h(x)$, you either have to apply h on x or you have to look into the table corresponding to h.

1.4 The Three Famous Problems


The Three Famous Problems
Pre-image Finding Problem
2nd Pre-image Finding Problem
Collision Finding Problem

1.4.1 Solution for Pre-image Finding Problem

Given $y \in Y$, $h : X \to Y$, $|Y| = M$, find $x \in X$.

Algorithm 1 Finding x such that $h(x) = y$

```
Input: $X_0 \subseteq X$ such that $|X_0| = Q$
Output: An element $x \in X_0$ such that $h(x) = y$
foreach $x \in X_0$ do
    Compute $y' = h(x)$  if $y' = y$ then
        return $x$
    end
end
```

This is also known as the exhaustive search method for finding the pre-image x given y. The probability of the above algorithm returning the correct pre-image is inversely proportional to the time complexity.

The provided equations analyze the probability and time complexity of finding pre-images in set X0. They demonstrate that as the size of X0 increases, both the probability and time complexity decrease, resulting in a more efficient pre-image search.

For example, $X_0 = \{x_1, x_2, \ldots, x_q\}$:

$$\begin{aligned}
P(\text{event } E_i) &= P(h(x_i) = y) = 1 - \frac{1}{M} \quad \text{(as 1 out of the length of } y \text{ will be the outcome)} \\
P(\text{event } E_i') &= 1 - P(E_i) = 1 - \frac{1}{M} \\
P(E_1 \cup E_2 \cup \ldots \cup E_q) &= 1 - P(E_1' \cap E_2' \cap \ldots \cap E_q') \\
&= 1 - P(E_1') \cdot P(E_2') \cdot \ldots \cdot P(E_q') \\
&= 1 - \left(1 - \frac{1}{M}\right)^q \\
&= \frac{Q}{M}
\end{aligned}$$

Therefore, complexity $= O(M/Q) = O(M)$, i.e., the bigger set of X_0, the greater the probability and the lesser the time complexity to find the pre-image.

1.4.2 Solution for 2nd Pre-image Finding Problem

Given x, $h(x)$, find x' such that $h(x) = h(x')$, where X_0 is a subset of X without x, and $|X_0| = Q$. Therefore, we can use the same algorithm as above and perform an exhaustive search. Time complexity $= O(M)$.

1.4.3 Solution for Collision Finding Problem

We have $h : X \to Y$, where $|Y| = M$. We need to find x' and x such that $x' \neq x$ and $h(x) = h(x')$.

Algorithm 2 Finding a pair of elements $\{x, x'\}$ with equal hash values

```
Input: $X_0 \subseteq X - \{x\}$ with $|X_0| = Q$
Output: A pair of elements $\{x, x'\}$ such that $h(x) = h(x')$ and $x \neq x'$
foreach $x' \in X_0$ do
    Compute $y_x = h(x)$ and $y_{x'} = h(x')$  if $y_x = y_{x'}$ then
        return $\{x, x'\}$
    end
end
```

Let E_i be the event that $h(x_i)$ is not equal to any of $\{h(x_1), h(x_2), \ldots, h(x_{i-1})\}$.

$$\begin{aligned}
\text{For } i = 1, &\quad Pr[E_1] = 1 \quad \text{(since } h(x_1) \text{ should not belong to the empty set).} \\
\text{For } i = 2, &\quad \text{given } E_1, Pr[E_2|E_1] = \frac{M-1}{M} \quad \text{(mapping to all elements except } h(x_1)\text{).} \\
\text{Similarly, for } i = 3, &\quad \text{given } E_1 \cap E_2, Pr[E_3|E_1 \cap E_2] = \frac{M-2}{M}. \\
\text{In general, for } i = k, &\quad \text{given } E_1 \cap E_2 \cap \ldots \cap E_{k-1}, Pr[E_k|E_1 \cap E_2 \cap \ldots \cap E_{k-1}] = \frac{M-(k-1)}{M}.
\end{aligned}$$

Therefore, the collision probability ϵ is given by:

$$\epsilon = 1 - Pr[E_1 \cap E_2 \cap \ldots \cap E_Q]' = 1 - \left(\frac{M-1}{M} \cdot \frac{M-2}{M} \cdot \ldots \cdot \frac{M-(Q-1)}{M}\right).$$

Solving for Q:

$$Q = \sqrt{2 \ln \left(\frac{1}{1-\epsilon} \right) \sqrt{M}}.$$

Hence, the value of Q is approximately the square root of M.

2 Compression Function

$h : \{0,1\}^{m+t} \to \{0,1\}^m$ is a hash function that takes inputs of length $m+t$ and produces outputs of length m. The goal is to construct a function $H : \{0,1\}^* \to \{0,1\}^m$ from h, where H takes inputs of any length and produces outputs of length m.

$h : \{0,1\}^{m+t} \to \{0,1\}^m$
$Secondpreimage, preimage \to O(2^m)$
$Collision \to O(2^{m/2})$

Algorithm 3 Compress

Suppose that Compress: $\{0{,}1\}^{m+t} \to \{0,1\}^m$ is a compression function.
Input:

- x : An input string of length greater than m + t + 1.

Output:

- $h(x)$: The hash value of the input string x.

Process

- Pad x with 0s to get a string y with a length divisible by t.
- Let $y = y_1||y_2||...||y_r$ where each y_i has length t (except possibly the last one).
- Initialize $z_0 \leftarrow IV$.
  For $i = 1$ to r do:
  $z_i \leftarrow compress(z_{i-1}||y_i)$

3 Merkle-Damgard Construction

The Merkle-Damgard construction has the property that the resulting hash function satisfies desirable security properties, such as collision resistance, provided that the compression function does. It helps in constructing a hash function from a compression function.

Suppose **Compress:** $\{0,1\}^{(m+t)} \to \{0,1\}^m$ is a collision-resistant compression function, where $t \geq 1$. So **compress** takes $m+t$ input bits and produces m output bits. We will use **compress** to construct a collision-resistant hash function $h : X \to \{0,1\}^m$; the hash function h takes any finite bitstring of length at least $m+t+1$ and creates a message digest that is a bitstring of length m.

Merkle-Damgård Hash Construction

Input: x: Input message

Output: $h(x)$: Hash value of x

Step 1: Calculate the length of the input message:

$$n = |x| \quad \text{(Length of input message)}$$

Step 2: Determine the number of blocks:

$$K = \left\lfloor \frac{n}{t-1} \right\rfloor \quad \text{(Number of blocks)}$$

Step 3: Calculate the padding size:

$$d = K(t-1) - n \quad \text{(Padding size)}$$

Step 4: For $i = 1$ to $K-1$, set $y_i = x_i$.

Step 5: Pad the last block:

$$y_K = x_K || 0^d \quad \text{(Pad last block)}$$

Step 6: Convert the padding size to binary representation:

$$y_{K+1} = \text{binary}(d) \quad \text{(Binary representation of padding size)}$$

Step 7: Initialize the state:

$$Z_1 = 0^{m+1} || y_1 \quad \text{(Initialize state)}$$

Step 8: Compress the initial state:

$$g_1 = \text{compress}(Z_1) \quad \text{(Compress initial state)}$$

Step 9: For $i = 1$ to K, update the state and compress it:

$$Z_{i+1} = g_i || 1 || y_{i+1} \quad \text{(Update state)}$$
$$g_{i+1} = \text{compress}(Z_{i+1}) \quad \text{(Compress state)}$$

Step 10: Final hash value:

$$h(x) = g_{K+1} \quad \text{(Final hash value)}$$

4 Secure Hash Functions (SHA)

The Secure Hash Algorithm (SHA) was devised by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. A revised edition, known as SHA-1, was released as FIPS 180-1 in 1995.

There exist three variants of SHA: SHA-160, SHA-256, and SHA-512, generally denoted as $SHA : \{0,1\}^* \to \{0,1\}^n$.

4.1 Types of SHA

Secure Hash Functions (SHA)

SHA-1	**SHA-256**	**SHA-512**
Message Size: $< 2^{64}$ bits	Message Size: $< 2^{64}$ bits	Message Size: $< 2^{128}$ bits
Block Size: 512 bits	Block Size: 512 bits	Block Size: 1024 bits
Word Size: 32 bits	Word Size: 32 bits	Word Size: 64 bits
Message Digest Size: 160 bits	Message Digest Size: 256 bits	Message Digest Size: 512 bits

4.2 SHA I in Detail

For SHA I, the message size is limited to 2^{64} bits. If $|x| \leq 2^{64}$, padding is applied such that y becomes a multiple of 512 bits by appending a single '1' followed by necessary '0's.

The SHA I algorithm involves four distinct functions and five constants, along with four keys and five initial hash values.

Algorithm 4 SHA I Algorithm

Input: x: Input message

Output: $h(x)$: Hash value of x

```
$n \leftarrow |x|$ ; $K \leftarrow \left\lfloor \frac{n}{t-1} \right\rfloor$ ; $d \leftarrow K(t-1) - n$ ; for $i = 1$ to $K-1$ do
    $y_i \leftarrow x_i$ ;
end
$y_K \leftarrow x_K || 0^d$ ; $y_{K+1} \leftarrow \text{binary}(d)$ ; $Z_1 \leftarrow 0^{m+1} || y_1$ ; $g_1 \leftarrow \text{compress}(Z_1)$ ; for $i = 1$ to $K$ do
    $Z_{i+1} \leftarrow g_i || 1 || y_{i+1}$ ; $g_{i+1} \leftarrow \text{compress}(Z_{i+1})$ ;
end
$h(x) \leftarrow g_{K+1}$ ; return $h(x)$ ;
```

5 Message Authentication Code (MAC)

A Message Authentication Code (MAC) serves as a cryptographic tool ensuring both the integrity and authenticity of a message or data transmission. Its primary function is to validate that a message hasn't been tampered with during transmission and originates from a trusted source.

5.1 HMAC (Hash-based Message Authentication Code)

HMAC stands for Hash-based Message Authentication Code and is widely adopted for message authentication and integrity verification purposes.

Working Principle:

- HMAC takes the message and a secret key as inputs.
- It utilizes a cryptographic hash function (e.g., SHA-256 or SHA-512) on the message, with the secret key as the hash function's "key".

- The resulting hash output undergoes further processing to generate the MAC.
- The MAC produced ensures both the integrity and authenticity of the message.

5.2 CBC-MAC (Cipher Block Chaining Message Authentication Code)

CBC-MAC, or Cipher Block Chaining Message Authentication Code, is employed for generating a MAC using a block cipher in Cipher Block Chaining (CBC) mode.

Operation:

- CBC-MAC processes fixed-size data blocks. If the message surpasses the block size, it is partitioned into blocks.
- Each block of the message undergoes processing using a symmetric encryption algorithm (e.g., AES) in CBC mode with a secret key.
- The resultant output of CBC-MAC constitutes the MAC for the entire message, which can be appended to or transmitted with the message for verification purposes.
- To thwart certain attacks like length extension attacks, CBC-MAC mandates a distinct initialization vector (IV) for each message.

6 Other SHAs

They are defined in the NIST Standard FIPS (Federal Information Processing Standards) 180-4. Rest of the things we were asked to refer from this PDF: **https://csrc.nist.gov/files/pubs/fips/180-2/final/docs/fips180-2.pdf**. Below is a small overview of the same.

6.1 SHA 256

Properties:

- **Message Size:** $< 2^{64}$ bits
- **Block Size:** 512 bits
- **Word Size:** 32 bits
- **Message digest size:** 256 bits

6.2 SHA-512

Properties:

- **Message Size:** Up to 2^{128} bits
- **Block Size:** 1024 bits
- **Word Size:** 64 bits
- **Message Digest Size:** 512 bits

[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 9)

1 Diffie-Hellman Key Exchange

The Diffie-Hellman algorithm introduces the concept of public key cryptography, facilitating secure key exchange over insecure channels, ensuring data transmission security and integrity. Consider two users, Alice and Bob, who agree to communicate using a cyclic group $G = \langle g \rangle$, where $|G| = P$ (P is a large prime number). Alice chooses a secret key a, $0 \leq a \leq p-1$, and computes $K_a = g^a$. Similarly, Bob chooses b, $0 \leq b \leq p-1$, and computes $K_b = g^b$. They exchange K_a and K_b publicly.

$$\begin{array}{ll} \text{Alice} & \text{Bob} \\ K_a = g^a & K_b = g^b \end{array}$$

Then, Alice computes $(K_b)^a = (g^b)^a = g^{ba}$, and Bob computes $(K_a)^b = (g^a)^b = g^{ab}$. And, the shared secret key is $g^{ab} = g^{ba}$. The difficulty of compuing x from g^x is known as the discrete logarithm problem. This algorithm was published in IEEE Transactions on Information Theory. Let us understand this in mathematical format:

1. If $a, b \in G$, then $a * b \in G$.
2. There exists an element $e \in G$ such that $a * e = a = e * a$, where e is the identity element of G.
3. $a * (b * c) = (a * b) * c$ (associative property).
4. For every element $a \in G$, there exists an inverse.

$$|G| = n$$

$$0 \leq a \leq n-1 \quad K_a = g^a \quad 0 \leq b \leq n-1$$

$$\begin{array}{cccc} & K_a = g^a & K_a = g^b & \\ a & & b & \\ & K_b = g^b & K_b = g^a & \end{array}$$

Compute:

$$\begin{array}{ll} (K_b)^a = (g^b)^a & (K_a)^b = (g^a)^b \\ \Rightarrow g^{ba} & \Rightarrow g^{ab} \end{array}$$

Shared Secret Key: g^{ab}

Alice	**Bob**
Secret Key $= a$	Secret Key $= b$
Public Key $= g^a$	Public Key $= g^b$

Without knowing a and b (secret keys), we cannot compute g^{ab} (shared secret key). Given g^x (public) and x (secret), finding x from g^x is computationally difficult due to the properties of the group $< G >=< g >$.

$$\boxed{\textbf{Note: } \text{This hard problem is known as the discrete logarithm problem.}}$$

$$y = g^x$$

$$\log_g y = x$$

1. $|G|$ is very large.
2. The $*$ operation must be sufficiently secure.

$$(Z_p, +_p) =< g >$$

$$g_i = (g +_p g +_p \cdots +_p g) = ig$$

$$g^i = (g +_p g +_p \dots) = ig = y$$

$$i = gy^{-1} \bmod (p)$$

1.1 Man-in-the-Middle Attack

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. In this attack, an opponent, Oscar, intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Oscar substitutes it with her own and sends it to Alice. Thus, Oscar and Alice agree on one shared key, and Oscar and Bob agree on another shared key.

For example:

Alice has a and computes g^a, sending it towards Bob. Oscar intercepts this and has c, calculates and shares g^c with Bob. Bob, unaware that g^c is sent by Oscar, assumes it's from Alice. Similarly, Bob has b, calculates g^b, and shares it with Alice, intercepted by Oscar, who shares g^c with Alice as well.

Hence, Alice encrypts a message m as $C_1 = \text{Enc}(m, g^{ac})$. When intercepted by Oscar, who also has g^{ac}, Oscar can alter the message or read it. Oscar then encrypts again with $C_2 = \text{Enc}(m, g^{bc})$ and sends it to Bob, who decrypts with g^{bc}, thinking nothing is wrong.

$$\begin{array}{ccc} Alice & |G| = n - large & Bob \\ 0 < a < n & Z_p{}^*, \dot{m}od(p) & 0 < b < n \\ g^a & & g^b \end{array}$$

$$=< g >\rightarrow Cyclic - Group$$

P: Prime Number
$P = 2^{255} - 19$

How to continue $g^a(\underbrace{g * g * g * g}_{a-Times})$

$t = 1$
for $(i = 2; i < a; i++)$
$t = (t * g)\%p$

2 Square and Multiply Algorithm

We wish to compute x^c

Convert c to binary: $C \rightarrow (C_{l-1} \ldots C_0)$

```
$z \leftarrow 1$
for $i \leftarrow l-1$ downto 0
  do $z \leftarrow z^2 \mod n$
     if $c_i = 1$
        then $z \leftarrow (z \times x) \mod n$
return $(z)$
```

Complexity: $log(c)$

$$c = \sum_{i=0}^{l} c_i 2^i$$
$$x^c = x^{\sum_{i=0}^{l} c_i 2^i}$$
$$= \prod_{i=0}^{l} x^{c_i z^i}$$
$$= x^{c_0 z^0} . x^{c_1 z^1} \ldots .$$

3 RSA (Rivest, Shamir, Adleman)

$$\phi(m) = \text{number of positive integers } < m \text{ which are co-prime to } m.$$
$$g(x, m) = 1$$
$$\phi(8) = 4; \{1, 3, 5, 7\}$$
$$\gcd(a, m) = 1$$
$$S = \{x \mod (m)\}$$
$$= \{r_1, r_2, \ldots, r_m\}$$
$$S_1 = \{ar_1, ar_2, \ldots, ar_m\}$$
$$ar_i = ar_j \quad r_i \neq r_j$$
$$ar_i \equiv ar_j \mod (n) \quad r_i \neq r_j \mod (m)$$
$$\gcd(a, m) = 1$$
$$1 = ab + ms$$
$$\exists b \text{ such that } ab \equiv 1 \mod (m)$$
$$ar_i \equiv ar_j \mod (m)$$
$$bar_i = bar_j \mod (m)$$
$$\therefore ar_i \not\equiv ar_j \mod (m)$$

3.1 Euler's Theorem

If $\gcd(a, m) = 1$, then $a^{\phi(m)} \equiv 1 \pmod{m}$. Let us assume that we have a set S, such that:

$$S = \{x \mid \gcd(x, m) = 1\}$$

$$S = \{s_1, s_2, s_3, s_4, \ldots, s_{\phi(m)}\}$$

Let us consider $\gcd(a, m) = 1$ and create another set S_1 such that

$$S_1 = \{as_1, as_2, as_3, \ldots, as_{\phi(m)}\}$$

We know from the discussion above that, if $as_i \equiv as_j \pmod{m}$, then $s_i \equiv s_j \pmod{m}$. Given that $\gcd(a, m) = 1$ and $b \cdot a \equiv 1 \pmod{m}$,

$$|S| = \phi(m)$$

$$|S_1| = \phi(m)$$

Since a is coprime with m and s_i is also coprime with m, there must be some correspondence between elements of S and S_1.

$$s_i \equiv as_j \pmod{m}$$

Let us now take product on both sides and simplify:

$$a^{\phi(m)} \equiv 1 \pmod{m}$$

3.2 Fermat's Theorem

If p is a prime number and p does not divide a (meaning that p is coprime to a), then

$$a^{p-1} \equiv 1 \pmod{p}$$

Using Fermat's theorem, $\Rightarrow a^p \equiv a \pmod{p}$.

Note: Fermat's theorem will not hold when p does not divide a.

3.3 RSA Cryptosystem

Facts:

- If $\gcd(a, m) = 1$, then $a^{\phi(m)} \equiv 1 \pmod{m}$.
- $a^{p-1} \equiv 1 \pmod{p}$ if p is prime and p does not divide a.

Now, let's understand the components of RSA:

1. $n = pq$, where p and q are primes.
2. Plaintext space: $\mathbb{Z}_n$
   Ciphertext space: $\mathbb{Z}_n$
3. Key space: $\{\ K = (n, p, q, e, d) \mid ed \equiv 1 \pmod{\phi(n)}\ \}$
4. Encryption:

$$E(x, K) = c$$

$$c = E(x, K) = x^e \pmod{n}$$

5. Decryption:

$$\mathrm{Dec}(c, K) = x$$

$$c = \mathrm{Dec}(c, K) = c^d \pmod{n}$$

We know that e and d are related as:

$$ed \equiv 1 \pmod{\phi(n)}$$

$$\Rightarrow ed - 1 = t \cdot \phi(n)$$

$$\Rightarrow 1 = ed + t_1 \cdot \phi(n)$$

$$1 = \gcd(e, \phi(n)) = ed + t_1 \cdot \phi(n)$$

Encryption:

$$c = x^e \pmod{n}$$

Decryption:

$$x = c^d \pmod{n}$$

$$c^d = (x^e)^d \pmod{n}$$

$$c^d = x^{ed} \pmod{n}$$

Now using $ed = 1 + t \cdot \phi(n)$ from above:

$$c^d = x^{1+t\cdot\phi(n)} \pmod{n}$$

$$c^d = x \cdot x^{t\cdot\phi(n)} \pmod{n}$$

Since p and q are primes and $n = pq$, then $\phi(n) = (p-1)(q-1)$:

$$c^d = x \cdot x^{t[(p-1)(q-1)]} \pmod{n}$$

Finally,

$$c^d = x \cdot x^{t[(p-1)(q-1)]} \pmod{pq}$$

Now, let us simplify the part $x^{t[(p-1)(q-1)]} \pmod{pq}$, where $x \in \mathbb{Z}$:

We check $x^{t[(p-1)(q-1)]} \pmod{p}$:

$$\equiv (x^{p-1})^{t(q-1)} \pmod{p}$$

$$\equiv 1 \pmod{p}$$

(As $x^{p-1} \equiv 1 \pmod{p}$)

Now we check $x^{t[(p-1)(q-1)]} \pmod{q}$:

$$\equiv (x^{q-1})^{t(p-1)} \pmod{q}$$

$$\equiv 1 \pmod{q}$$

(As $x^{q-1} \equiv 1 \pmod{q}$) We finally have:

$$x^{t[(p-1)(q-1)]} \equiv 1 \pmod{p}$$

$$x^{t[(p-1)(q-1)]} \equiv 1 \pmod{q}$$
$$\Rightarrow x^{t[(p-1)(q-1)]} \equiv 1 \pmod{pq}$$

Substituting the above result:

$$c^d = x \cdot x^{t[(p-1)(q-1)]} \pmod{pq}$$

$$c^d = x \cdot 1 \pmod{pq}$$

$$c^d = x \pmod{pq}$$

Hence, our decryption is successful.

Now let us consider a scenario where Alice is trying to communicate with Bob. Here, two keys play the main role: one is the public key and the other is the secret key. Bob encrypts the message and sends it to Alice, who has both keys. The public key is known to Bob, but the secret key is not known to Bob.

Alice and Bob communication using RSA	
Alice	**Bob**
$n = pq$ and p and q are large prime numbers.	Now Bob selects a message x from $\mathbb{Z}_n$.
$ed \equiv 1 \pmod{\phi(n)}$	x
She chooses e.	He knows n and e for Alice, so he can encrypt.
Public key of Alice: (n, e)	$y = x^e \pmod{n}$
She can generate d using the extended Euclidean algorithm.	
Secret key of Alice: (p, q, d)	

Now the message y is sent to Alice, who can decrypt it with her secret key as:

$$x = y^d \pmod{n}$$

Note: If we are able to compute p and q from n, then we will be able to compute $\phi(n)$. And then we already have e, so we will be able to find d using the extended Euclidean algorithm and the security of RSA will be broken. So, RSA is based on the fact that the factorization problem is hard.

3.4 RSA Vulnerability

The RSA encryption scheme faces a critical vulnerability: if given the public key (n, e) and a ciphertext c, allowing the retrieval of the original plaintext x ($c = x^e$), the security of RSA is compromised.

While breaking RSA implies the capability to factorize n, the converse is not necessarily true. Therefore, RSA security relies on two fundamental assumptions: first, the difficulty of factorization, and second, the complexity of decryption. It has the following two problems.

3.4.1 The RSA Problem

When presented with the public key (n, e) and a ciphertext c generated by encrypting a plaintext x ($c = x^e$), the RSA problem involves deducing the corresponding private key d directly from these elements.

3.4.2 The Factorization Problem

On the other hand, the factorization problem revolves around determining the prime factors p and q of n. Once these factors are determined, computing d becomes feasible.

Note: Solving the factorization problem allows for the resolution of the RSA problem. However, the reverse is not necessarily true. RSA security relies on the assumption that both the RSA problem and the factorization problem pose significant computational challenges.

4 Digital Signature Algorithm (DSA)

The Digital Signature Algorithm (DSA) is a cryptographic algorithm utilized to generate digital signatures, authenticate message senders, and prevent message tampering. It is represented as $(P, S, K, \text{Sign}, V)$, where:

- P is the plaintext space,
- S is the signature space,
- K is the K-tuple,
- Sign is the signature algorithm, and
- V is the validation algorithm.

It operates with **two keys K:** a private key held by the sender and a public key distributed to recipients. The sign algorithm in DSA is represented as:

$$S = m^d \pmod{n}$$

Here, m is the message, d is the private key, and n is the modulus.

4.1 Differences between RSA and DSA

In RSA encryption:

$$c_1 = m_1^e \mod n$$
$$c_2 = m_2^e \mod n$$
$$c_1 \times c_2 = (m_1 \times m_2)^e \mod n$$

However, in DSA, hash functions are used:

$$s_1 = (h(m_1))^d \mod n$$
$$s_2 = (h(m_2))^d \mod n$$
$$s_1 \times s_2 \neq (h(m_1 \times m_2))^d \mod n$$

4.2 Signature Generation

1. Choose two distinct prime numbers p and q.
2. Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$.
3. Select an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
4. Compute the integer d such that $1 < d < \varphi(n)$ and $ed \equiv 1 \pmod{\varphi(n)}$.
5. The public key is (n, e) and the private key is (n, d).

4.3 Signature Validation

To encrypt a message m using the public key (n, e), compute the ciphertext S as:

$$S = m^d \mod n$$

To decrypt the ciphertext S using the private key (n, d), compute the message V as:

$$V = S^e \mod n$$

In DSA, the security verification involves checking the equality:

$$S_1 \times S_2 \neq (h(m_1 \times m_2))^d \mod n$$

where $h()$ represents a hash function. This algorithm ensures message integrity and authenticity in digital communications.

[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 10)

1 Modular Equations

We'll explore solving systems of linear equations to find x in the form:

$$a \cdot x \equiv b \mod m \text{ (Eq.1)}$$

To begin, let's express Eq.1 as:

$$a \cdot x - m \cdot y = b \text{ (Eq.2)}$$

where y is an integer. Utilizing Bezout's Identity:

$$a \cdot x_0 + m \cdot y_0 = \gcd(a, m) \text{ (Eq.3)}$$

where x_0 and y_0 can be determined using the Extended Euclidean Algorithm.

Eq.2 is solvable if and only if $\gcd(a, m)$ divides b. Assuming $\gcd(a, m)$ divides b, we have:

$$t \cdot \gcd(a, m) = b$$

By multiplying Eq.3 by t, we get:

$$a \cdot (t \cdot x_0) + m \cdot (t \cdot y_0) = t \cdot \gcd(a, m) \implies a \cdot X_0 + m \cdot Y_0 = b$$

Hence, given an equation to solve, we first verify if $\gcd(a, m)$ divides b. If so, a solution exists. Then, we find x_0 and y_0 using the Extended Euclidean Algorithm and multiply them by $t = \frac{b}{\gcd(a,m)}$ to obtain X_0 and Y_0.

Once X_0 and Y_0 are identified as solutions of Eq.2, we can substitute x and y as follows:

$$x = X_0 + \frac{m}{\gcd(a,m)} \cdot n$$
$$y = Y_0 + \frac{a}{\gcd(a,m)} \cdot n$$

where n is an integer. For any n, the derived x and y satisfy Eq.2, establishing them as the general solution.

Now, let's consider a system of two modular equations:

$$x \equiv a_1 \mod m_1 \text{ (Eq.1)}$$
$$x \equiv a_2 \mod m_2 \text{ (Eq.2)}$$

where m_1 and m_2 are coprime. We aim to find x satisfying both equations. If x is a solution of Eq.1, then:

$$x = a_1 + m_1 \cdot y \text{ (Eq.3)}$$

If x is also a solution to Eq.2, then:

$$x \equiv a_2 \mod m_2$$

Substituting the value of x from Eq.3 into Eq.2, we obtain:

$$m_1 \cdot y \equiv (a_2 - a_1) \mod m_2 \text{ (Eq.4)}$$

Since $\gcd(m_1, m_2) = 1$, we find the solution to Eq.4 as:

$$y = y_0 + m_2 \cdot n$$

From Eq.3, we deduce:

$$x = (a_1 + m_1 \cdot y_0) + n \cdot m_1 \cdot m_2$$

If y_0 is known, let $x_0 = a_1 + m_1 \cdot y_0$, then:

$$x = x_0 + m_1 \cdot m_2 \cdot n$$
$$x \equiv x_0 \mod m_1 \cdot m_2$$

x_0 is congruent modulo to any x under modulo $m_1 \cdot m_2$. Since x is the general solution of the two equations, every solution of the given system of equations will always be congruent to x_0 under modulo $m_1 \cdot m_2$.

Chinese Remainder Theorem (CRT)

The Chinese Remainder Theorem (CRT) is a fundamental concept in number theory used to solve systems of simultaneous congruences.

Consider a system of congruences:

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

where $m_1, m_2, \ldots, m_n$ are pairwise coprime positive integers.

The CRT states that this system of congruences has a unique solution modulo $M = m_1 \cdot m_2 \cdots m_n$. Furthermore, the solution x can be expressed as:

$$x \equiv a_1 \cdot M_1 \cdot y_1 + a_2 \cdot M_2 \cdot y_2 + \cdots + a_n \cdot M_n \cdot y_n \pmod{M}$$

where $M_i = M/m_i$ and y_i is the modular multiplicative inverse of M_i modulo m_i, i.e., $M_i \cdot y_i \equiv 1 \pmod{m_i}$.

Uniqueness

Let's assume x' is another solution of the above system. Then we have $x' \equiv x \pmod{m_1, m_2, \ldots, m_r}$.

$$\begin{aligned} x' &\equiv x \pmod{m_1} \\ x' &\equiv x \pmod{m_2} \\ x' &\equiv x \pmod{m_3} \\ &\vdots \\ x' &\equiv x \pmod{m_r} \\ x' &\equiv x \pmod{m_1, m_2, \ldots, m_r} \end{aligned}$$

This implies that x' and x are congruent modulo each individual modulus m_i, and thus they are congruent modulo the product of all moduli.

2 Exploring Elliptic Curve Cryptography (ECC)

- **Introduction**: While RSA offers a straightforward approach to cryptography with the Square and Multiply Algorithm, Elliptic Curve Cryptography (ECC) introduces a novel concept.
- **Computations on Curves**: ECC operates on elliptic curves rather than integers, leading to the development of modern cryptographic techniques such as the Diffie-Hellman Key Exchange Algorithm and the Signature Algorithm.
- **Key Exchange**: ECC employs Elliptic Curve Diffie-Hellman (ECDH) for secure key exchange, offering enhanced security with smaller prime numbers compared to RSA.
- **Digital Signatures**: Signatures in ECC are generated using Elliptic Curve Digital Signature Algorithm (ECDSA), providing robust security while minimizing computational complexity.
- **Security Benefits**: ECC's utilization of elliptic curves enables better security using smaller prime numbers, making it a preferred choice over RSA in many applications.
- **Transition to Discrete Structures**: ECC's foundation lies in discrete systems, highlighting the importance of understanding real numbers as a precursor to exploring its discrete aspects.

Let's define two real numbers a and b such that:

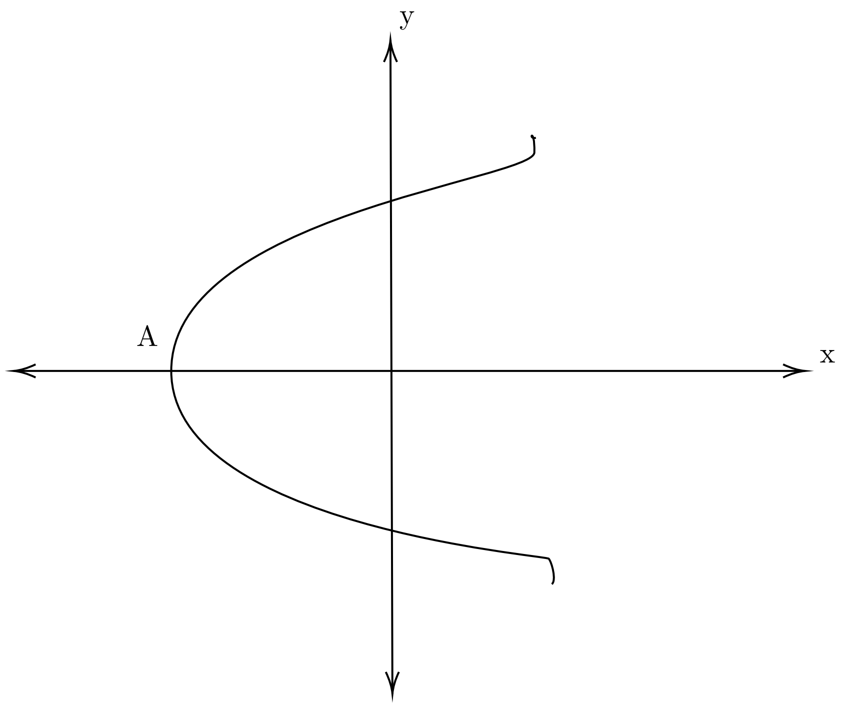
$$a, b \in \mathbb{R} \text{ and } 4a^3 + 27b^2 \neq 0$$

Consider the curve:

$$y^2 = x^3 + ax + b$$

where $(x, y) \in \mathbb{R}_2$. This curve is known as an Elliptic Curve.

When plotted, the curve exhibits two structures, one of which is depicted below:

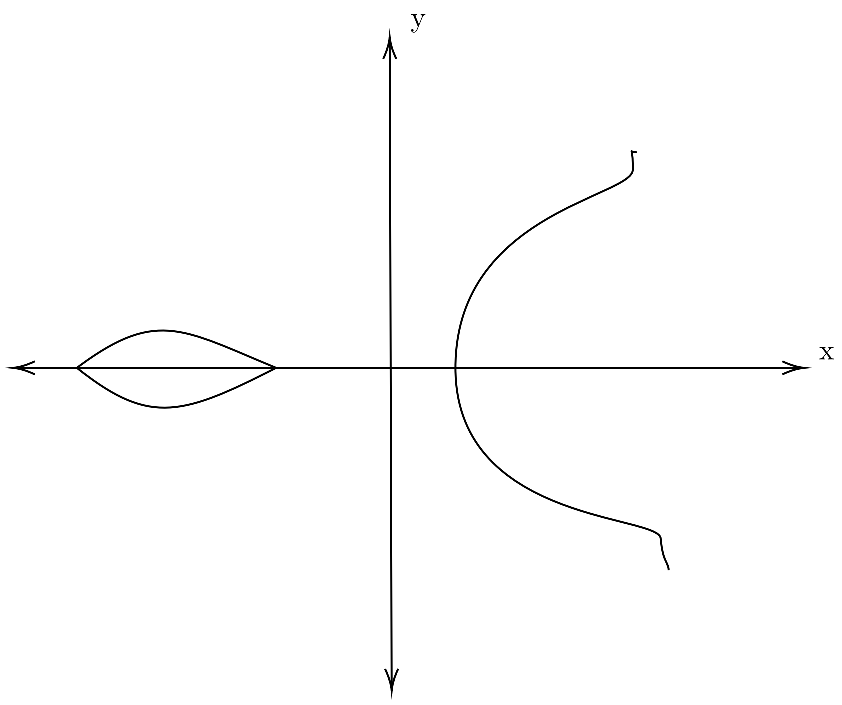


At point A, $y = 0$, implying $x^3 + ax + b = 0$ (Eq.1). This equation has three roots, which can either be:

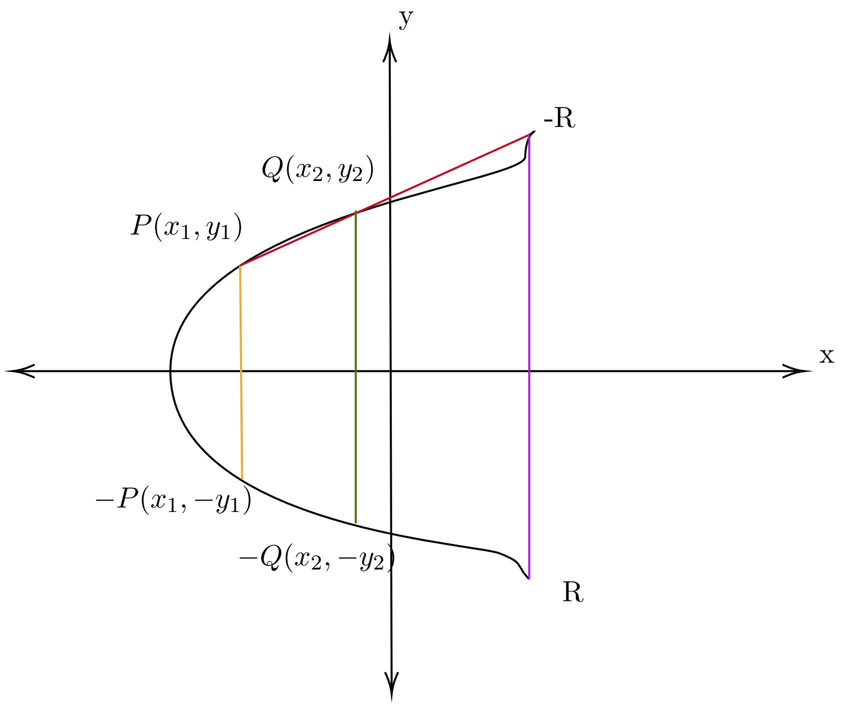
- Three real roots
- One real root and two complex roots

Eq.1 has three distinct roots if and only if $4a^3 + 27b^2 \neq 0$ (which can be real or complex). For the depicted curve, substituting $y = 0$ yields one real root and two complex roots.

If Eq.1 has three real roots, the resulting curve appears as shown below:



Let's introduce some properties of the previously defined curve:



Consider two points P and Q on the curve. When joined with a straight line, they intersect the curve again at a point, denoted as -R. The point -X is the mirror image of X with the x-axis as the mirror. Alternatively, we can say that the perpendicular from point X to the x-axis intersects the curve again at point -X.

1. $P \boxed{+} Q = R$. The $\boxed{+}$ operation is a binary operator defined as follows: take two points, join them with a straight line. The line intersects the curve again, and the image of this point on the x-axis is the output point.
2. Θ, known as the point at infinity, is introduced. Joining P and -P results in a straight line parallel to the y-axis, intersecting the curve at one point, assumed to be the point of infinity.
3. $P \boxed{+} - P = \Theta$
4. $P \boxed{+} \Theta = P$
5. $(P \boxed{+} Q) \boxed{+} R = P \boxed{+} (Q \boxed{+} R)$
6. $P \boxed{+} Q = Q \boxed{+} P$

The associativity and commutativity of the $\boxed{+}$ operator can be proved graphically. Treating Θ as an identity element and $-P$ as the inverse of P, the curve with the $\boxed{+}$ operator forms a commutative group.

Suppose we need to find $P \boxed{+} P$. We draw the tangent to the curve at point P, and wherever this tangent intersects the curve again, its image is the result. So, $P \boxed{+} P = R$ implies $2P = R$.

Elliptic Curve Mathematics:

$$y^2 = x^3 + ax + b$$
$$4a^3 + 27b^2 \neq 0$$

Let us consider two points $P(x_1, y_1)$ and $Q(x_2, y_2)$. We have three cases:

1. $x_1 \neq x_2$, $y_1 \neq y_2$
2. $x_1 = x_2$, $y_1 = -y_2$
3. $x_1 = x_2$, $y_1 = y_2$

Case-1:

$$y = mx + c \text{ ...Eqn(a)}$$
$$m = \tfrac{y_2 - y_1}{x_2 - x_1}$$
$$c = y_1 - mx_1,\ c = y_2 - mx_2$$

All the points on this line will satisfy this equation of the straight line. Equation of the straight line (Eqn(a)) will cut the curve at a point, so we substitute the value of y in the curve equation:

$$y_2 = x_3 + ax + b$$
$$m^2x^2 + 2mxc + c^2 = x_3 + ax + b$$
$$x^3 - m^2x^2 + (a - 2mc)x + (b - c^2) = 0$$

We already know that (x_1, y_1) and (x_2, y_2) will satisfy this equation. If x_3 is another solution of the above system, then:

$$x_1 + x_2 + x_3 = m^2$$
$$\implies x_3 = m^2 - x_1 - x_2$$
$$\text{We already know that } m = \tfrac{y_2 - y_1}{x_2 - x_1} = \tfrac{y_3 - y_1}{x_3 - x_1}$$
$$\implies y_3 = y_1 + m(x_3 - x_1)$$

So, we see that we obtained the coordinate of $R(x_3, y_3)$:

$$P \boxed{+} Q = R$$

Case-2: P = (x_1, y_1), Q = (x_2, y_2) where $x_1 = x_2$, $y_1 = -y_2$. In this case:

$$P \boxed{+} Q = \theta$$

Case-3: P = (x_1, y_1), Q = (x_2, y_2) where $x_1 = x_2$, $y_1 = y_2$:

$$y = mx + c$$
$$y_2 = x_3 + ax + b$$
$$\implies 2y\tfrac{dy}{dx} = 3x^2 + a$$
$$\implies \tfrac{dy}{dx} = \tfrac{3x^2 + a}{2y}$$
$$\left(\tfrac{dy}{dx}\right)_{(x_1, y_1)} = \tfrac{3x_1^2 + a}{2y_1} = m$$
$$c = y_1 - mx_1$$

Let us substitute in the curve:

$$y_2 = x_3 + ax + b$$
$$\implies (mx + c)^2 = x_3 + ax + b$$
$$x_1 + x_2 + x_3 = m^2$$
$$\implies x_3 = m^2 - x_1 - x_2$$
$$m = \tfrac{y_3 - y_1}{x_3 - x_1}$$
$$\implies y_3 = y_1 + m(x_3 - x_1)$$
$$\text{R} \rightarrow (x_3, -y_3)$$

Now, we will be considering the same curve in $\mathbb{Z}_{\mathbb{P}} \times \mathbb{Z}_{\mathbb{P}}$, where P is a prime number:

$$y^2 = x^3 + ax + b, \text{ where } (x, y) \in \mathbb{Z}_{\mathbb{P}} \times \mathbb{Z}_{\mathbb{P}} \text{ and } a, b \in \mathbb{Z}_{\mathbb{P}}$$
$$4a^3 + 27b^2 \neq 0 \mod P$$

Since we are now working on discrete values, we will not obtain this curve. We will obtain points.

Case-1:

$$x^3 = m^2 - x_1 - x_2$$
$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Now, here we don't divide, we take the inverse under mod P. Since x_2, x_1 are different values, $x_2 - x_1$ will be non-zero, and we will be able to find its inverse under mod P since P is prime, so its gcd with $(x_2 - x_1)$ will be 1. $m = (y_2 - y_1) \times (x_2 - x_1)^{-1} \mod P$

$$\implies y_3 = y_1 + m(x_3 - x_1) \in \mathbb{Z}_{\mathbb{P}}$$

2.0.1 Elliptic Curve Diffie-Hellman (ECDH)

Let us consider a scenario where Alice and Bob want to exchange messages. They have a curve E and a point P, and (E, P) is public.

$$(E, P)$$

$$\begin{tikzcd}
\underline{\text{Alice}} & & \underline{\text{Bob}} \\
\text{secret: } a & & \text{secret: } b \\
\text{public: } aP \arrow[rr, "aP"] & & \text{public: } bP \\
a(bP) & & b(aP) \arrow[ll, "bP"'] \\
abP & & baP
\end{tikzcd}$$

$$abP = baP$$

$$\text{Shared secret key} = abP$$

In the above scenario, E and P were public while a and b were secret. Since a, b, P are all discrete, we can find aP (a times P), bP (b times P), and so on. Since they are discrete, abP and baP are the same. Since both Alice and Bob finally reached the same point on the curve, they have successfully exchanged messages.

Note: Security of ECDH depends on the fact that finding xP from P is computationally difficult. This hard problem is known as the **Discrete Log Problem on EC**.

RSA Signature

RSA Signature Encryption/Decryption:

$$\text{Encryption: } c = x^e \mod n, \quad \text{Decryption: } x = c^d \mod n$$

$$\text{Signature: } s = x^d \mod n, \quad \text{Verification: } x = s^e \mod n$$

Elliptic Curve Digital Signature Algorithm (ECDSA)

In ECDSA, a public key P corresponds to a secret key a, where the public key is represented as aP.

Properties of ECDSA

- ECDSA relies on a large prime number n, ensuring that $nG = 0$ on the elliptic curve, where G is the base point, and $(n-1)G \oplus G = nG$.

Signature Generation Process

1. Compute the hash e of the message m.
2. Select the leftmost L_n bits of e, where L_n is the bit length of n.
3. Choose a random integer k from the range $[1, n-1]$.
4. Generate a key pair (x_1, y_1).
5. Calculate $r = x_1 \mod n$. If $r = 0$, repeat step 1.
6. Calculate $s = k^{-1}(z + r \cdot d_A) \mod n$, where d_A is the secret key. If $s = 0$, repeat step 1.
7. The signature on message m is (r, s).

Verification Process by Bob

1. Ensure that the public key Q_A is not equal to 0.
2. Verify if Q_A lies on the elliptic curve.
3. Verify if $n \cdot Q_A = n \cdot d_A \cdot a$, where $Q_A = d_A \cdot G$.

Additional Verification Steps

1. Check if r and s are within the range $[1, n-1]$.
2. Compute e by hashing the message m.
3. Select the leftmost bits of e.
4. Compute $u_1 = z \cdot s^{-1} \mod n$ and $u_2 = r \cdot s^{-1} \mod n$.
5. Calculate the point $(x_2, y_2) = u_1 \cdot a + u_2 \cdot Q_A$. If $(x_2, y_2) = 0$, the signature is invalid.
6. Verify if $r \equiv x_2 \mod n$. If this condition holds, the signature is valid; otherwise, it is invalid.
7. Recompute e using $u_1 \cdot a + u_2 \cdot Q_A$. If the result matches the original hash e, the verification is successful.

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 11)

1 The Discrete Logarithm Problem (DLP)

The challenge of the Discrete Logarithm Problem lies in finding an integer x within the range $0 \leq x \leq (n-1)$ such that $\alpha^x = \beta$, given a cyclic group G of order n, its generator α, and an element $\beta \in G$.

To compute a when given g and g^a, an exhaustive search method can be employed. By iterating a loop from $i = 1$ to $i = n$, the size of the group (n), we calculate g^i. If $g^i = g^a$, we have found the desired result. The complexity of this approach corresponds to the size of the group. Alternatively, the Baby-Step Giant-Step Algorithm offers a solution to the Discrete Logarithm Problem with a complexity of approximately $\sqrt{n}$.

1.1 Baby-Step Giant-Step Algorithm

Initially, we calculate $m = \sqrt{n}$. Since n represents the order of the group and α serves as the generator of the group, it follows that $\alpha^n = 1$. Now, assuming $\beta = \alpha^x$, we can express x using the Division Algorithm as follows:

$$x = i \cdot m + j, \text{ where } 0 \leq i < j$$
$$\text{Thus, } \alpha^x = \alpha^{i \cdot m} \cdot \alpha^j$$
$$\text{Implying } \beta = \alpha^{i \cdot m} \cdot \alpha^j$$

Shifting $\alpha^{i \cdot m}$ to the right side, we obtain:

$$\alpha^j = \beta(\alpha^{im})^{-1}$$
$$\alpha^j = \beta(\alpha^{-m})^i$$

Now, instead of determining x, the focus shifts to finding i and j, both of which range within $m = \sqrt{n}$. We aim to accomplish this without significantly escalating complexity.

Formalizing the algorithm, the input comprises the generator α of cyclic group G, the order of group G (n), and $\beta \in G$. The output is the discrete logarithm $x = \log_\alpha \beta$. The procedure is outlined below:

1. Assign $m \leftarrow \lceil \sqrt{n} \rceil$
2. Create a table T with entries j, α^j $0 \leq j < m$. Sort T based on α^j values.
3. Compute α^{-m} and set $\gamma \leftarrow \beta$
4. For $i = 0$ to $i = (m-1)$, perform:
   - Check if γ is the second component of any entry in T.

- If $\gamma = \alpha^j$, then compute $x = i \cdot m + j$
- Update γ as $\gamma = \gamma \cdot \alpha^{-m}$

The table can be generated offline, necessitating $O(\sqrt{n})$ space. The runtime of the algorithm involves $O(\sqrt{n})$ multiplications. Sorting the table requires $O(\sqrt{n} \cdot \log(\sqrt{n})) \implies O(\sqrt{n} \cdot \log(n))$ time.

Example

Let us consider an example to understand this algorithm more clearly.

Given -

$$G = \mathbb{Z}_{113} \quad (p = 113)$$
$$\alpha = 3$$
$$|G| = 112 \quad \text{(where } |G| \text{ is the order of the group)}$$
$$\beta = 57$$

To find x such that $3^x = 57$ and $0 \leq x \leq \sqrt{112}$:

Solution -

Step 1: Find m which is $\lceil \sqrt{n} \rceil = \lceil \sqrt{112} \rceil \approx 11$

Step 2: Find $(j, \alpha^j \mod p)$ such that $0 \leq j \leq 11$

j	0	1	2	3	4	5	6	7	8	9	10
$(3^j \mod 113)$	1	3	9	27	81	17	51	40	7	21	63

Step 3: Sort the table based on $3^j \mod 113$ value

j	0	1	8	2	5	9	3	7	6	10
4										
$3^j \mod 113$	1	3	7	9	17	21	27	40	51	63
81										

Step 4: Find α inverse: $3^{-11} = (3^{-1})^{11} = (38)^{11} \mod 113 = 58$ Note: Inverse is under modulo 113 and is calculated using the extended Euclidean algorithm.

Step 5: Calculate $r = \beta \cdot \alpha^{-mi} \mod 113$ for $i = 0, 1, 2, \ldots, 10$

i	0	1	2	3	4	5	6	7	8	9
10										
$r = 57 \cdot (58)^i$	57	29	100	37	112	55	26	39	2	3
no-need										

$$\beta \cdot \alpha^{-9m} = 3 = \alpha^1$$
$$\beta \cdot \alpha^{-9\cdot 11} = \alpha^1$$
$$\text{Hence, } i = 9 \text{ and } j = 1$$
$$\text{Now, } x = im + j$$
$$x = 9 \cdot 11 + 1 = 100$$

Hence, $x = 100$**, which is the solution.**

2 ElGamal Public Key Cryptosystem

ElGamal Public Key Cryptosystem operates similarly to RSA but relies on the Discrete Log Problem for security. The following steps outline the encryption and decryption processes in detail:

1. Choose a prime number p.
2. Define the group $(Z_p^*, *_p)$.

   $$Z_p^* = \{1, 2, 3, \ldots, (p-1)\}$$
   $$x *_p y = x * y \pmod p$$

   If $x \in Z_p^*$, then $\gcd(x, p) = 1$, implying the existence of the multiplicative inverse of x modulo p.
3. Choose a primitive element $\alpha \in Z_p^*$, also referred to as the generator of Z_p^*.

   $$Z_p^* \text{ forms a cyclic group}$$
   $$Z_p^* = \langle \alpha \rangle$$
4. The plaintext space is Z_p^*, and the key space is $\{(p, \alpha, a, \beta), \beta = \alpha^a \pmod p)\}$.
5. The public key consists of $\{P, \alpha, \beta\}$, and the secret key is $\{a\}$.
6. Select a random number $x \in Z_{p-1}$, keeping it secret.
7. **Encryption:** The encryption algorithm yields a ciphertext tuple.

   $$e_K(m, x) = (y_1, y_2)$$
   $$y_1 = \alpha^x \pmod p$$
   $$y_2 = m \cdot \beta^x \pmod p$$
8. **Decryption:** The decryption process is as follows,

   $$d_K(y_1, y_2) = y_2 \cdot (y_1^a)^{-1} \pmod p = m$$
   $$y_1^a = (\alpha^x)^a \pmod p = \beta^x \pmod p$$
   $$y_2 \cdot (y_1^a)^{-1} = m \cdot \beta^x \cdot \beta^{x-1} \pmod p = m \pmod p$$

   The presence of randomness in the ciphertext stems from the randomly chosen x.

The public key is $\{\beta, \alpha, p\}$. The challenge lies in finding a from β and α (the Discrete Log Problem). However, if we possess the ciphertext and

$$y_1 = \alpha^x,$$

computing α^{ax} from $\beta = \alpha^a$ and α^x would enable us to derive the message m without needing to determine a or x. This would break the ElGamal Encryption, yet it wouldn't guarantee solving the Discrete Log Problem. This scenario mirrors the Diffie-Hellman Problem. If one can compute g^{ab} from g^a and g^b, the Diffie-Hellman Key Exchange Algorithm is compromised, but the Discrete Log Problem remains unsolved.

Example

Let us consider an example of ElGamal to understand this algorithm more clearly.

Given common information **(Public Information)**:

$$P = 7, \quad \alpha = 4, \quad \beta = 4^3 \equiv 1 \pmod 7$$

Alice's Side:

$$\begin{aligned} \text{Message } M &= 5 \\ x &= 2 \\ y_1 &= \alpha^x = 4^2 \equiv 2 \pmod 7 \\ y_2 &= M \cdot \beta^x = 5 \pmod 7 \end{aligned}$$

Now, Alice sends (y_1, y_2) to Bob.

Bob's Side:

$$\begin{aligned} \text{Bob's secret key } a = 3 \\ \text{Bob receives } (y_1, y_2) \text{ from Alice} \\ \text{Bob calculates:} \\ y_2 \cdot (y_1^a)^{-1} &= 5 \cdot (2^3)^{-1} \\ &= 5 \cdot (8)^{-1} \\ &= 5 \cdot (1)^{-1} \text{ (calculate inverse modulo } P \text{ using extended Euclidean algorithm)} \\ &= 5 \\ &= M \text{ (original message by Alice)} \end{aligned}$$

Note: Everything is under modulo P (7 here).

3 Kerberos (Version 4)

Kerberos is a network security protocol designed to authenticate service requests among trusted hosts over potentially untrusted networks, such as the internet. It relies on secret-key cryptography and a trusted third-party system to validate client-server interactions and authenticate user identities. It is important to note that this protocol uses Symmetric Key Cryptography at its core.

The Kerberos protocol involves three key entities:

- Ticket-Granting Server (TGS)
- Authentication Server (AS)
- Verifier (V)

The authentication process involves communication between a client C and various servers, namely the Authentication Server (AS), the Ticket Granting Server (TGS), and a Verifier (V). Below is a step-by-step description of the communication flow:

1. The client initiates communication by logging into the server and sending the following details to the Authentication Server:

$$\begin{aligned} C \rightarrow AS &: ID_c || ID_{TGS} || TS_1 \\ ID_c &\rightarrow \text{Identity of Client} \\ ID_{TGS} &\rightarrow \text{Identity of TGS} \\ TS_1 &\rightarrow \text{Timestamp} \end{aligned}$$

2. The AS will receive the information and send an encrypted message back to the client with the following information. AS performs symmetric key encryption using the key SK_c, which is shared between the AS and the client.

$$AS \rightarrow C : E(SK_c, [SK_{c,TGS} \mid\mid ID_{TGS} \mid\mid TS_2 \mid\mid Lifetime_2 \mid\mid Ticket_{TGS}])$$
$$SK_{c,TGS} \rightarrow \text{key used for communication between Client and TGS}$$
$$Lifetime_2 \rightarrow \text{for how long this ticket/data will be valid}$$

   ID_{TGS} and TS_2 are ID of TGS and timestamp as earlier. The ticket is generated by AS and contains the following information.

$$Ticket_{TGS} = E(SK_{TGS}, [SK_{c,TGS} \mid\mid ID_c \mid\mid AD_c \mid\mid ID_{TGS} \mid\mid Lifetime_2])$$
$$AD_c \rightarrow \text{Address of client}$$

   As it can be seen the $Ticket_{TGS}$ is encrypted using the key SK_{TGS}. It is the key used for communication between AS and TGS and it is not known to any other party.
   The client will receive the response from the AS and will be able to decrypt it as it was encrypted using SK_c, which is shared between client and AS. The client will recieve the following information:

$$[SK_{c,TGS},\ ID_{TGS},\ TS_2,\ Lifetime_2,\ Ticket_{TGS}]$$

   The client will receive the ticket but will not be able to decrypt it as client doesn't have SK_{TGS}. The client will also receive the key $SK_{c,TGS}$ which is the shared secret key between client and TGS. This secret key is also called as session key. When you establish a session in server, you get a session key for encrypted communication.

3. With the session key $SK_{c,TGS}$, the client initiates communication with the TGS:

$$C \rightarrow TGS : ID_v || Ticket_{TGS} || Authenticator_c$$
$$Authenticator_c = E(SK_{c,TGS}, [ID_c || AD_c || TS_3])$$

4. The TGS decrypts the ticket and authenticator to verify the client's identity and sends back encrypted information:

$$TGS \rightarrow C : E(SK_{c,TGS}, [SK_{c,v} || ID_v || TS_4 || Ticket_v])$$

   The ticket $Ticket_v$ includes information encrypted using the key SK_v, known only to TGS and the verifier.

5. The client sends the ticket and a fresh authenticator to the verifier:

$$C \rightarrow V : Ticket_v || Authenticator_c$$

6. The verifier decrypts the received data to authenticate the client:

$$V \rightarrow C : E(SK_{c,v}, [TS_5 + 1])$$

Note - Refer Cryptography and Network Security - Principles and Practice by William Stallings for further reading.

[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 12)

1 Introduction to the Signal Protocol

The Signal Protocol stands as a cryptographic framework ensuring end-to-end encryption for confidential communication. It serves as the foundational structure for various widely-used messaging applications such as Signal, WhatsApp, and Facebook Messenger. Below is an elucidation of its operational mechanism:

1. **Key Establishment:** The Signal Protocol initiates with a key establishment phase to facilitate secure communication between two entities, typically denoted as Alice and Bob. It employs a variant of the Diffie-Hellman key exchange known as the Double Ratchet algorithm.

2. **Double Ratchet Algorithm:** This algorithm generates a sequence of ephemeral keys to ensure both forward secrecy and future secrecy. Forward secrecy guarantees that compromising one key does not jeopardize the security of prior or subsequent keys, while future secrecy ensures the continued security of past messages even if long-term keys are compromised.

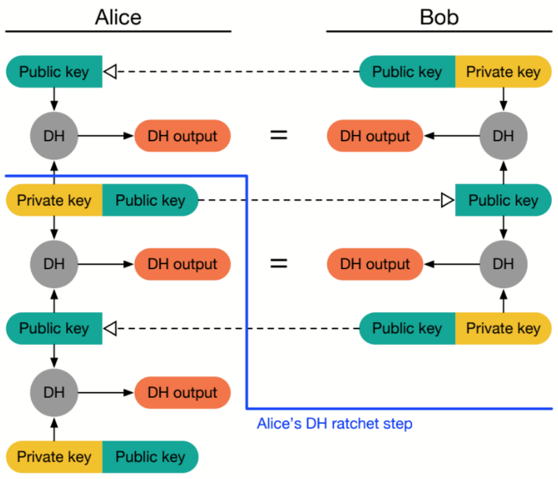


Figure 1: Double Ratchet Algorithm - Alice Side

3. **Session Initialization:** Alice and Bob exchange their long-term identity keys along with one-time prekeys to establish a secure session.

4. **Chain Key:** Each participant maintains a chain key, responsible for deriving fresh keys for every message. This key is updated following the transmission or reception of each message.

5. **Root Key:** The root key, a durable secret shared between Alice and Bob, serves to derive new chain keys and authenticate messages.

6. **Message Encryption:** When Alice intends to transmit a message to Bob, she generates a new message key for encrypting the message. Furthermore, she generates a fresh chain key based on her existing one.

7. **Message Authentication:** A Message Authentication Code (MAC) is utilized to authenticate the message, ensuring its integrity during transit.

8. **Forward and Future Secrecy:** Due to the generation of new keys for each message and the maintenance of a limited key set, compromise of a single key does not compromise the security of past or future messages.

9. **Message Reception:** Upon receiving the encrypted message, Bob decrypts it using his session keys and updates his chain key accordingly.

10. **Asynchronous Communication:** The Signal Protocol accommodates asynchronous communication, allowing message transmission and reception even if one party is offline. Upon reconnection, missed messages can be securely retrieved.

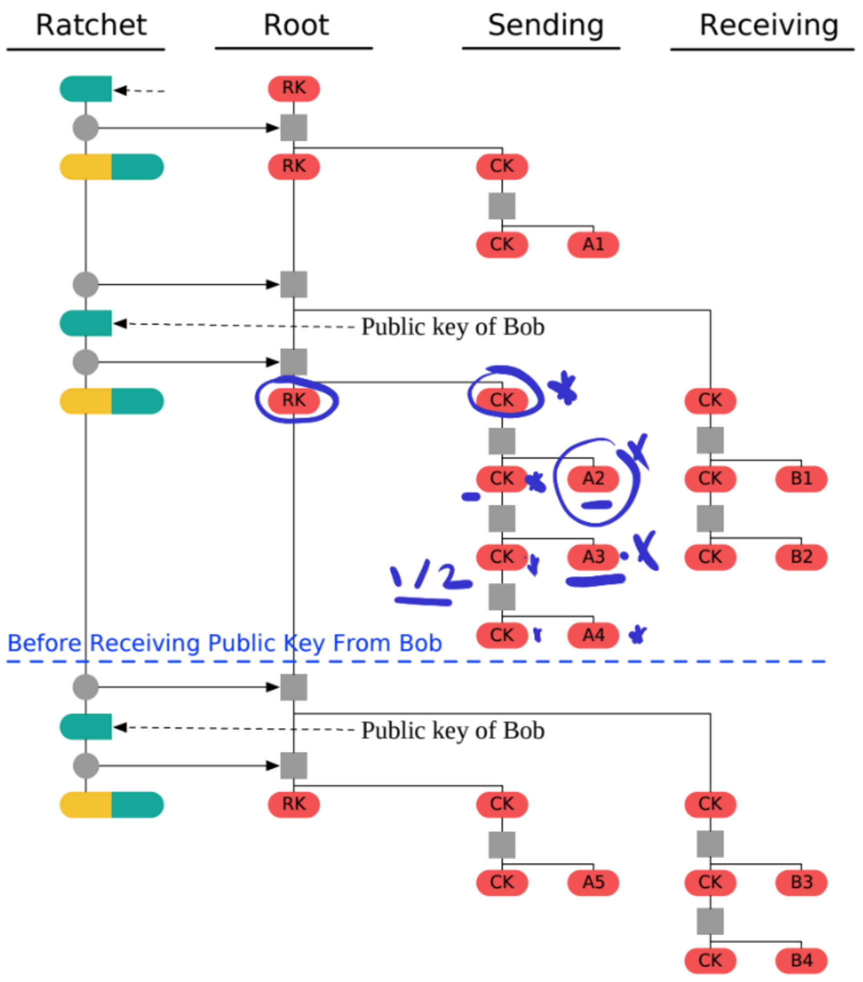


Figure 2: Communication between Alice and Bob

2 Zero Knowledge Proof using Discrete Log Problem

In a Zero Knowledge Proof (ZKP), a cryptographic technique is utilized wherein one party, the prover, can demonstrate to another party, the verifier, the validity of a statement without revealing any additional information except its truthfulness. When employing the Discrete Logarithm Problem, a ZKP allows a party to confirm their familiarity with a discrete logarithm without explicitly disclosing the logarithm itself.

The Discrete Logarithm Problem (DLP) is a fundamental issue in mathematics and cryptography. Given a group G with a generator g and an element h in the group, finding the integer x such that $g^x = h$ is computationally challenging, especially when the group is large and properly selected.

2.1 Zero Knowledge Proof (ZKP)

A ZKP for the DLP involves the following steps:

- **Setup:** The prover and verifier agree on a prime number p, a generator g of cyclic group G of order q (where q is prime), and an element h in G.
- **Commitment:** The prover selects a random integer r and computes $y = g^r \mod p$. Prover sends y to the verifier.
- **Challenge:** The verifier selects a random challenge c (bit).
- **Response:** If $c = 0$, the prover sends r to the verifier. Otherwise, the prover sends $r + c \times x$ to the verifier.
- **Verification:** The verifier checks whether $g^{r+x\times c} = h \times y^c$ holds. If yes, it indicates that the prover knows the discrete logarithm x of h with respect to g, thus validating the ZKP.

2.2 Explanation

- **Soundness:** The ZKP is sound because if the prover does not know the discrete logarithm x, it is computationally infeasible for them to generate a convincing response to the verifier.
- **Completeness:** If the prover indeed knows the discrete logarithm x, they can generate a convincing response that satisfies the verifier with overwhelming probability.
- **Zero-Knowledge Property:** The ZKP does not disclose any information about the discrete logarithm x to the verifier other than the fact that the prover possesses it. This property ensures that the protocol reveals no additional information about the prover's secret.

2.3 Application

- ZKPs find various applications in cryptography and beyond, including authentication protocols, secure multi-party computation, and anonymous cryptocurrencies like Zcash.
- By employing ZKPs for the DLP, parties can prove possession of secret keys or knowledge of specific values without revealing those secrets, thereby enhancing privacy and security in numerous cryptographic protocols.

3 RC4 Stream Cipher

RC4, devised by Ron Rivest in 1987, stands as a symmetric stream cipher renowned for its straightforwardness and rapidity, rendering it prevalent across diverse applications. The acronym ”RC” denotes ”Rivest Cipher,” while it is alternatively recognized as ”ARC4” or ”ARCFOUR” (reportedly owing to its purported utilization in the initial Netscape SSL protocol, though this claim lacks official substantiation).

3.1 Algorithm

The RC4 algorithm functions through the following steps:

1. **Key Initialization**:
    - RC4 employs a variable-length key, typically ranging from 40 to 2048 bits, to initialize its internal state.
    - Utilizing a key-scheduling algorithm (KSA), the key is expanded into an initial permutation of the internal state array (S-box).
2. **Pseudo-Random Bit Generation**:
    - After completing the key setup, RC4 generates a stream of pseudo-random bits or bytes.
    - This generation process involves iteratively swapping elements of the state array based on the current state and outputting a byte from the array.
3. **XOR with Plaintext/Ciphertext**:
    - The generated pseudo-random stream is XORed with the plaintext to produce ciphertext (or with ciphertext to retrieve plaintext).
    - This XOR operation ensures reversibility of the encryption and decryption processes with the same key.

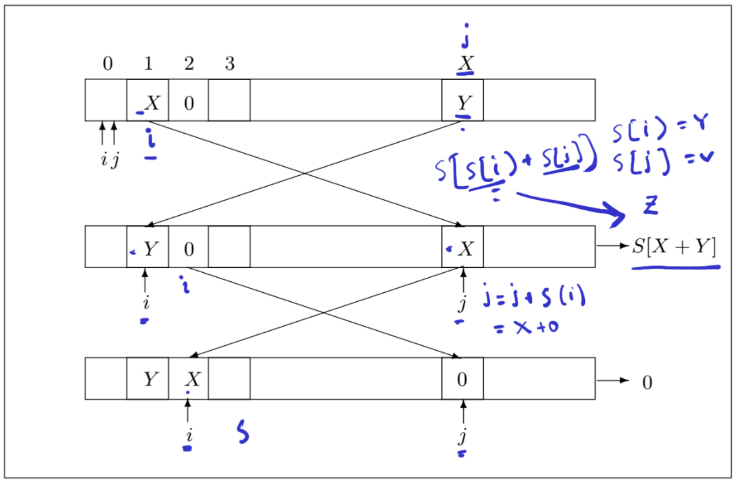


Figure 3: The Biased Second Output of RC4

3.2 Drawbacks

While RC4 enjoyed widespread adoption previously, it has since fallen out of favor for most cryptographic applications due to numerous vulnerabilities uncovered over time. These vulnerabilities comprise biases in the initial stream bytes and weaknesses within the key scheduling algorithm.

Despite its security shortcomings, RC4 persists in application within certain legacy systems and protocols. However, contemporary cryptographic standards, such as AES (Advanced Encryption Standard), are strongly advocated for new implementations owing to their superior security assurances.

4 Secured Sockets Layer (SSL)

Many cryptographic protocols rely on three fundamental cryptographic primitives:

- Symmetric Key Encryption Algorithms: These are favored because computations in Public Key Cryptography are typically more computationally intensive than those in symmetric key cryptography.
- Key Exchange Protocols: These facilitate the establishment of a common shared key between communicating parties. A signature mechanism is often employed to verify the authenticity of the source and the public keys to mitigate attacks such as Man-in-the-Middle attacks.
- Message Authentication Codes (MACs): These are used to authenticate encrypted data, providing assurance that the message originated from the expected party and has not been tampered with.

The SSL protocol is widely implemented, especially in HTTPS. It ensures secure communication between two parties by incorporating key exchange, symmetric key encryption, signature-based authentication of public keys, and Message Authentication Codes to authenticate messages.

The process begins with a connection. A **connection** refers to a transport mechanism that furnishes a suitable type of service. These connections are transient, with each connection being linked to a single session. A **session**, for instance, is established when accessing a HTTPS website, wherein a temporal duration is defined for exchanging data securely. Throughout this period, all data exchanges remain fully encrypted. Sessions are initiated by the Handshake Protocol and define a collection of cryptographic security parameters that can be utilized across multiple connections. They serve to circumvent the resource-intensive renegotiation of new security parameters for each connection.

The state of a session is determined by the following parameters:

1. **Session identifier:** An arbitrary byte sequence designated by the server to identify an active or resumable session state.
2. **Peer certificate:** An X509.v3 certificate of the peer, which may be null. Typically, it represents a signed public key. Acquiring an SSL certificate entails having your public key authenticated by a certified authority using their secret key, thus enabling verification by others that the authority has endorsed it. X509.v3 is an example of such a certified authority.

3. **Compression method:** The algorithm employed to compress data before encryption, accompanied by a corresponding decompression algorithm.

4. **Cipher spec:** This specification outlines the bulk data encryption algorithm (e.g., null, AES), a hash algorithm (e.g., MD5 or SHA-1) utilized for MAC calculation, and the mechanism for key exchange (e.g., Diffie-Hellman, ECDH). Additionally, it defines cryptographic attributes such as the hash size.

5. **Master secret:** A 48-byte secret mutually shared between the client and server. It serves to generate certain keys, including those used for data encryption and MAC generation.

6. **Is resumable:** A flag denoting whether the session is capable of initiating new connections.

The state of a connection is characterized by the following parameters:

1. **Server and client random:** Byte sequences individually chosen by the server and client for each connection. These numbers are utilized in specific computations to thwart certain attacks.

2. **Server write MAC secret:** The secret key employed in MAC operations on data transmitted by the server, shared between the client and server.

3. **Client write MAC secret:** The symmetric key utilized in MAC operations on data transmitted by the client.

4. **Server write key:** The symmetric encryption key utilized for encrypting data by the server and decrypting it by the client.

5. **Client write key:** The symmetric encryption key employed for encrypting data by the client and decrypting it by the server.

6. **Initialization vectors:** In the case of using a block cipher in CBC mode, an initialization vector (IV) is maintained for each key. This field is initially set during the SSL Handshake Protocol. Subsequently, the final ciphertext block from each record is retained for utilization as the IV with the succeeding record.

7. **Sequence numbers:** Each party manages separate sequence numbers for transmitted and received messages within each connection. Upon sending or receiving a "change cipher spec message," the relevant sequence number is reset to zero. Sequence numbers are restricted to a maximum value of $2^{64} - 1$.

4.1 SSL Record Protocol

Within the SSL framework, the initial protocol in operation is the SSL Record Protocol. This protocol furnishes two vital services for SSL connections:

1. **Confidentiality:** As stipulated by the Handshake Protocol, a shared secret key is established for the conventional encryption of SSL payloads.

2. **Message Integrity:** Additionally, the Handshake Protocol delineates a shared secret key employed to construct a message authentication code (MAC), thereby ensuring message integrity.

Given a packet, it is first broken into multiple blocks of fixed size, with the block length depending on the protocol version. Each block undergoes compression, compressing the fragmented data to a fixed length size. The MAC is then generated using the MAC Secret Key on this compressed data. The MAC is concatenated with the compressed data. Subsequently, this concatenated data is encrypted using an encryption algorithm. Finally, a header containing public information (not to be encrypted) is added before the encrypted data. This constitutes the complete packet to be transferred either from client to server or vice versa.

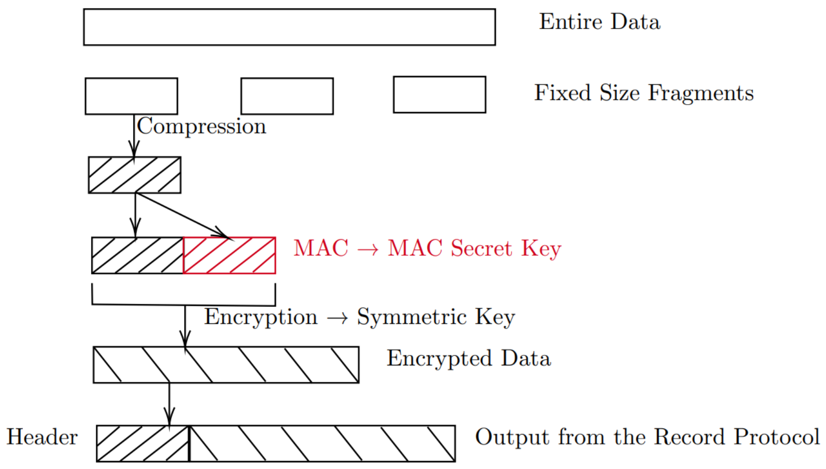


Figure 4: Flowchart for SSL

Suppose we have received data from the server. The length of the header part is known, so it is discarded from the packet, and any necessary information required is extracted from it. Then, decryption is performed, yielding the compressed data concatenated with the MAC. Since the length of the MAC is known, that part is removed, leaving only the compressed data. From this compressed data, a new MAC is generated. If the generated MAC matches the received MAC, the data is authenticated. If the MACs match, decompression is performed to retrieve the original data contained in the packet.

The two keys required are the MAC Secret Key and the encryption key, which are the same at both the client and server sides. The MAC is generated using the formula:

$$\begin{aligned}\text{MAC :}&\text{hash(MAC\_write\_secret}||\text{pad2}||\\ &\text{hash(MAC\_write\_secret}||\text{pad1}||\text{seq\_num}||\\ &\text{SSLCompressed.type}||\text{SSLCompressed.length}||\text{SSLCompressed.fragment))}\end{aligned}$$

The list of encryption algorithms supported by SSL is as follows:

AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128

During negotiation between the client and server, they agree upon one encryption algorithm. The header contains the following information, which is public and not required to be encrypted:

- Content Type (8 bits): The type of data.
- Major Version (8 bits): For example, if using version 3 of SSL, the major version will be 3.
- Minor Version (8 bits): For example, if using version 3 of SSL, the minor version will be 0.
- Compressed Length (16 bits): Length of the compressed data.

4.2 Change Cipher Spec Protocol

Another protocol utilized in SSL is the Change Cipher Spec Protocol. Its primary function is to update the cipher suite to be utilized on the current connection. For instance, suppose RC4 is being used for encrypting the data. After a certain period, the client and server may desire to alter the encryption mechanism and transition to AES. The Change Cipher Spec Protocol serves this purpose in such scenarios.

4.3 Alert Protocol

The Alert Protocol serves the primary function of issuing alerts in case of anomalies or errors. Below is a partial list of alerts, along with descriptions of when they are triggered by the Alert Protocol.

- **unexpected_message:** Indicates the receipt of an inappropriate message.
- **bad_record_mac:** Indicates the reception of an incorrect MAC.
- **decompression_failure:** Indicates a failure in the decompression function, such as the inability to decompress or decompressing beyond the maximum allowable length.
- **handshake_failure:** Indicates the inability of the sender to negotiate an acceptable set of security parameters from the available options.
- **illegal_parameter:** Indicates a field in a handshake message that is out of range or inconsistent with other fields.
- **close_notify:** Notifies the recipient that the sender will cease sending any further messages on this connection. Both parties are required to send a close_notify alert before closing the write side of the connection.
- **bad_certificate:** Indicates the receipt of a corrupt certificate, such as one containing a signature that does not verify.
- **unsupported_certificate:** Indicates that the type of the received certificate is not supported.
- **certificate_revoked:** Indicates that a certificate has been revoked by its signer.
- **certificate_expired:** Indicates that a certificate has expired.
- **certificate_unknown:** Indicates that some unspecified issue arose during the processing of the certificate, rendering it unacceptable.

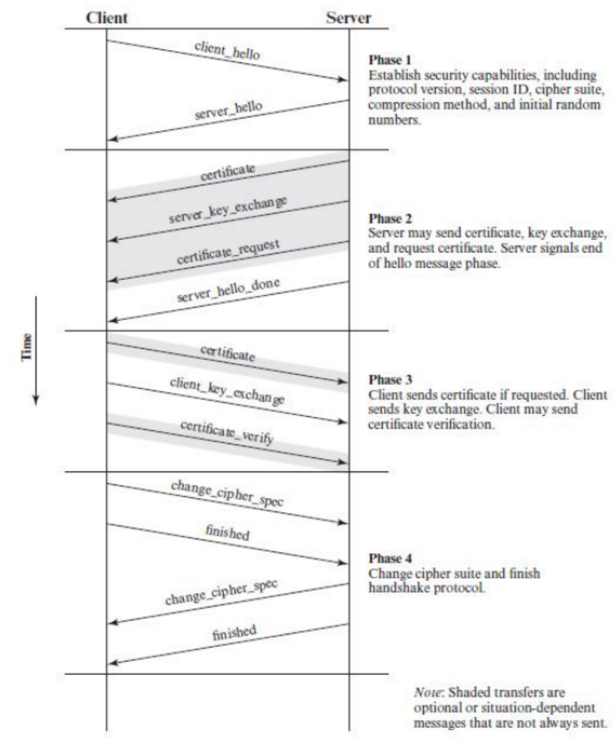

Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Figure 5: Handshake Protocol Flowchart

4.4 Handshake Protocol

Step 1: Initiation and Exchange The Handshake Protocol begins with the client initiating communication by sending a `client_hello` message, followed by the server responding with a `server_hello` message.

Client:

- Sends `client_hello` message containing:
  - Version
  - Random structure
  - Session ID
  - Cipher Suite
  - Compression Method

Server:

- Responds with `server_hello` message mirroring client's parameters.

Step 2: Certificate Exchange and Key Initiation The server sends its certificate (`server_key_exchange`) and may request the client's certificate (`certificate_request`). It then concludes with a hello done message.

Server:

- Sends certificate and initiates key exchange.
- Optionally requests client's certificate.
- Concludes with a hello done message.

Step 3: Client Response and Verification The client responds by sending its certificate, key exchange data (`client_key_exchange`), and, if requested, a certificate verification (`certificate_verify`).

Client:

- Responds with its certificate and key exchange data.
- Optionally sends certificate verification.

Step 4: Cipher Specification and Finish Both parties signal cipher specification compatibility (`change_cipher_spec`) and send finish messages (`finished`).

Client:

- Signals cipher specification compatibility.
- Sends finish message.

Server:

- Signals cipher specification compatibility.
- Sends finish message.

Step 5: Master Secret Key Generation The master secret key is generated using a pre-master secret and random values.

Formula:

$$\text{master\_secret} = \text{MD5}(\text{pre\_master\_secret}||\text{SHA}(A||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$
$$||\text{MD5}(\text{pre\_master\_secret}||\text{SHA}(BB||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$
$$||\text{MD5}(\text{pre\_master\_secret}||\text{SHA}(CCC||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$

Step 6: Key Block Generation The master secret is used to generate a key block for encryption, MAC, and initialization vectors.

Formula:

$$\text{key\_block} = \text{MD5}(\text{master\_secret}||\text{SHA}(A||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$
$$||\text{MD5}(\text{master\_secret}||\text{SHA}(BB||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$
$$||\text{MD5}(\text{master\_secret}||\text{SHA}(CCC||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$

This detailed breakdown outlines the handshake protocol's steps, including key generation and exchange, ensuring secure communication between parties.