[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Sanidhya Kumar (202151138)

Winter 2023-2024
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**

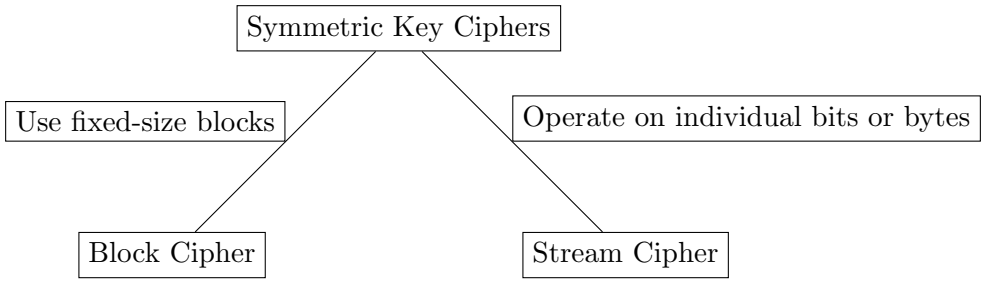

Figure 1: Types of Symmetric Key Ciphers

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,

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

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

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

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

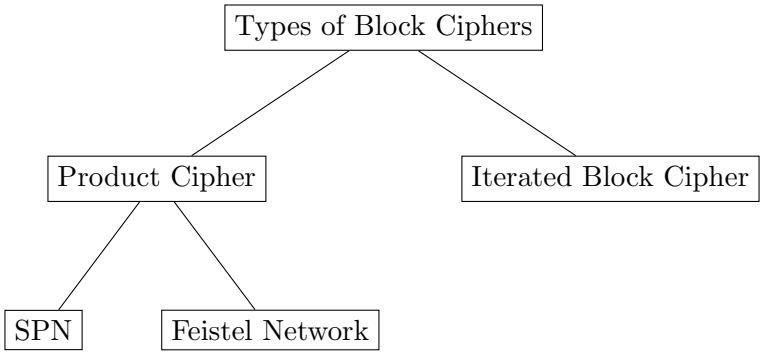

Figure 2: Tree Diagram of Block Ciphers

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.

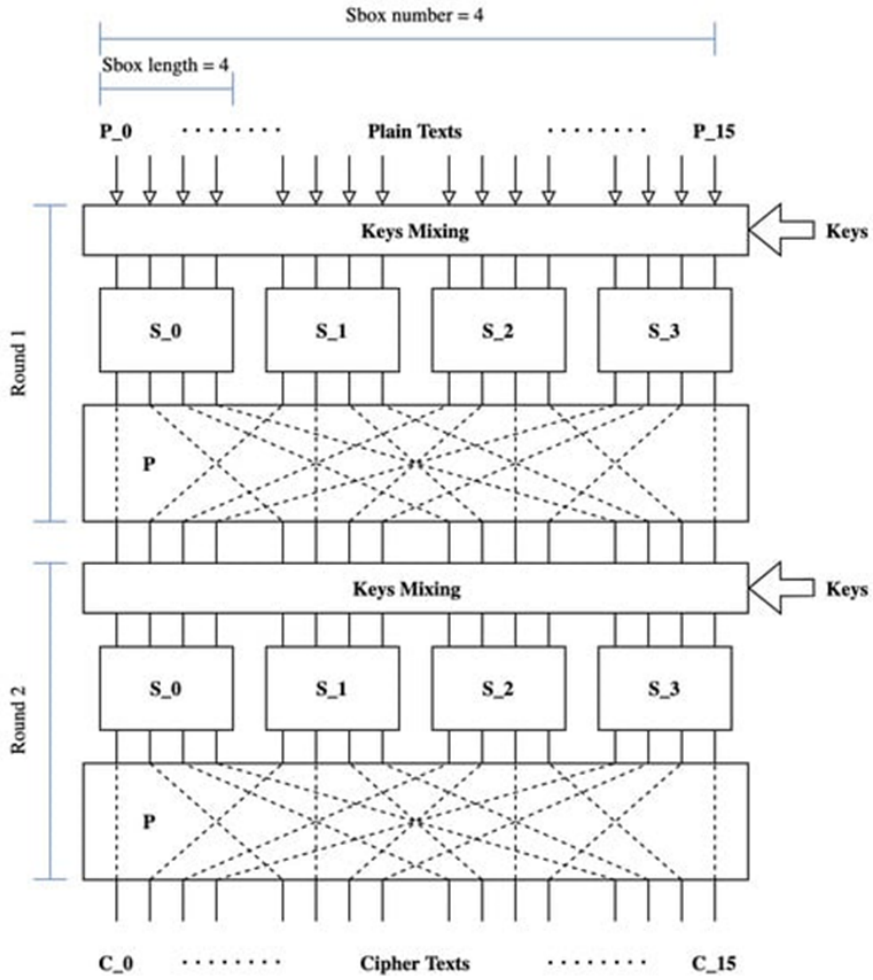

Figure 3: Substitution-Permutation Network (SPN)

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.

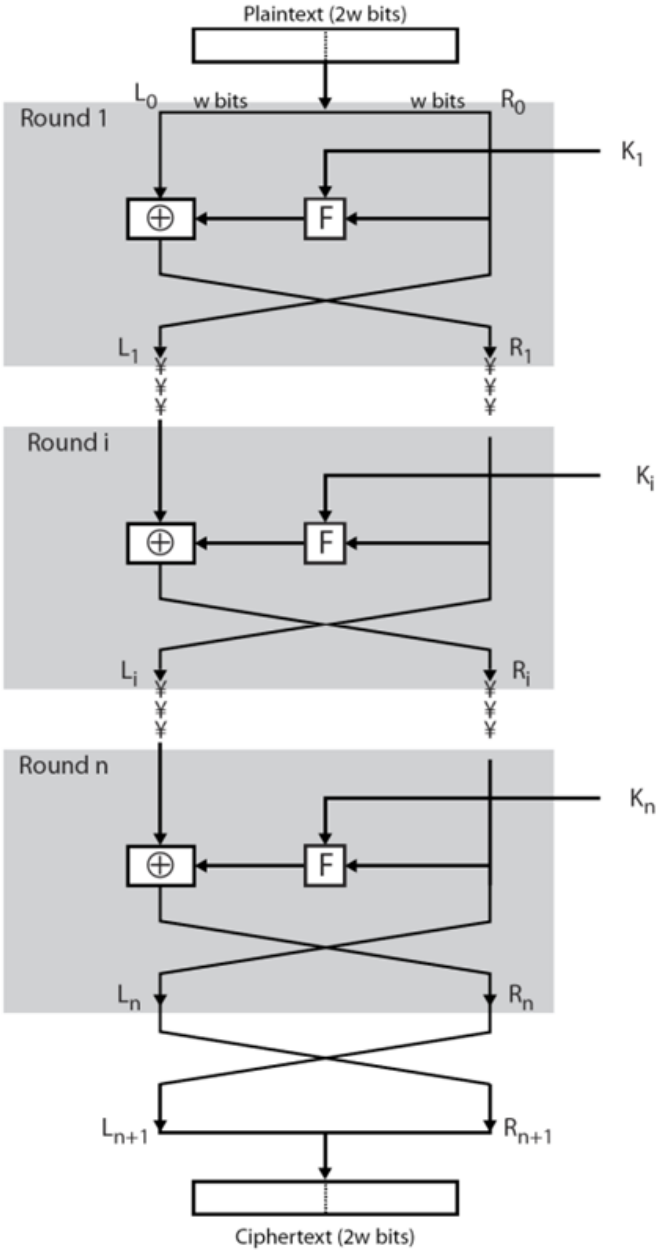

Figure 4: Feistel Network

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: $F(k_1, P) \rightarrow C_1 \rightarrow F(k_2, C_1) \rightarrow C_2 \rightarrow$ 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 –

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

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

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:

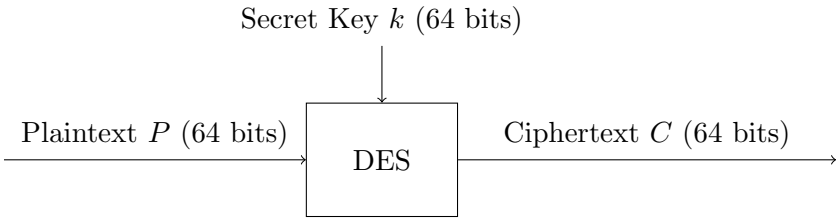

Figure 5: DES encryption block diagram

Decryption:

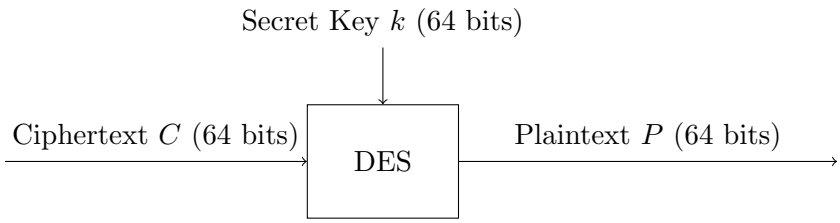

Figure 6: DES decryption block diagram

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:

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

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

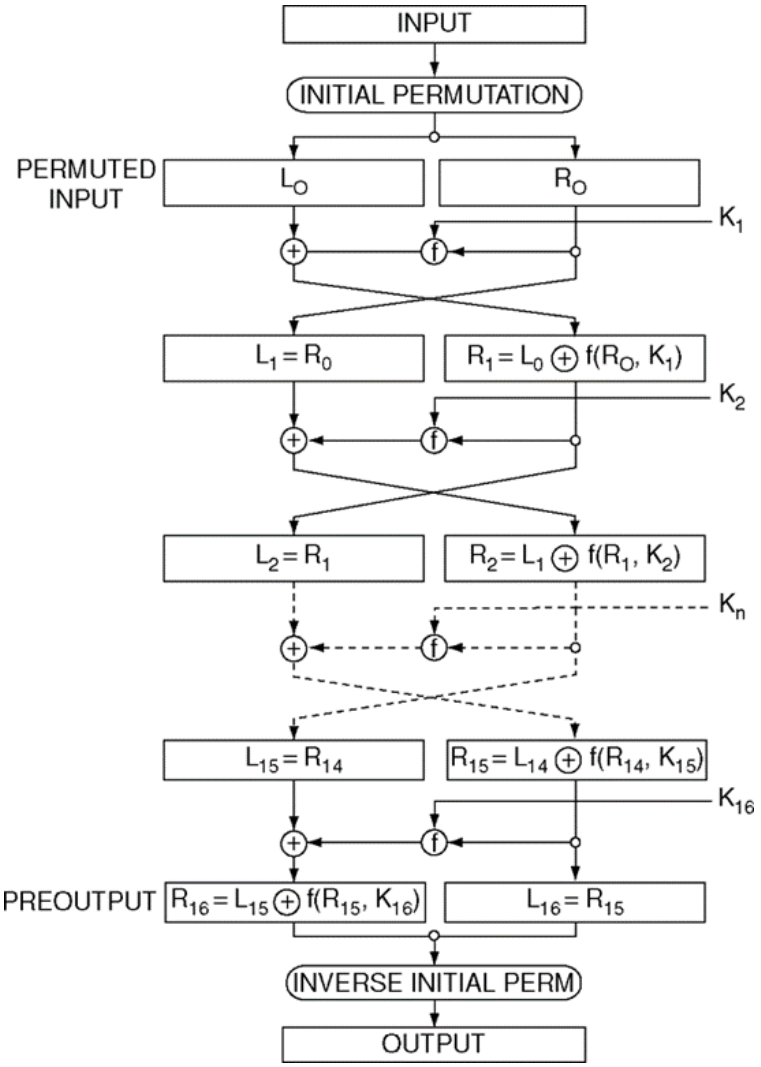

Figure 7: DEC Encryption

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`