Feistel Cipher Architecture

- A Feistel Cipher (invented by Horst Feistel) is a design model from which many different block ciphers are derived. DES and is the most common example of a Feistel Cipher. In fact, virtually all conventional block encryption algorithms are based on Feistel Cipher Structure.

- A cryptographic system based on Feistel cipher structure uses the same algorithm for both encryption and decryption.

- It is based on the concept of an invertible product cipher, which combines two or more transformations to create a cipher which is more secure than the it's individual components.

- This makes the cryptanalysis of the cipher very difficult. The product cipher combines a sequence of simple transformations such as substitution (S-box), permutation (P-box), and modular arithmetic.

- The concept of product ciphers is attributed to Claude Shannon, who presented the idea in his foundational paper, Communication Theory of Secrecy Systems.

- The algorithm partitions the input block into equal two halves and processes them through multiple rounds which perform a **substitution** (S) on left data half.

- The right half is subjected to a round function together with a key, and then uses **permutation** (P) by swapping the halves. In this way it implements Shannon's S-P net concept.

Claude Shannon and Substitution-Permutation Ciphers

- Claude Shannon's idea of substitution-permutation (S-P) networks forms the basis of all modern block ciphers.

- S-P nets are based on the two primitive cryptographic operations:

    - substitution (S-box)
    - permutation (P-box)

- A secure cipher must completely obscure the statistical properties of the original message; the one-time pad does this.

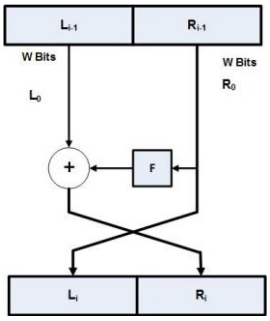
- Shannon proposed that the S-P primitives can be used to introduce **confusion** and **diffusion** into the message and key.

- **Confusion**
    - Makes relationship between ciphertext and key as complex as possible.

- **Diffusion**
    - Dissipates the statistical structure of the plaintext over bulk off ciphertext

Basic Function

- The following diagram illustrates a simple round of a Feistel function:



- The function starts by splitting the block of plain text data into two equal parts, traditionally referred to as termed $\mathbf{L_0}$ and $\mathbf{R_0}$ (the initial round).

- The next round will be $\mathbf{L_1}$ and $\mathbf{R_1}$, and then $\mathbf{L_2}$ and $\mathbf{R_2}$, and so on, for as many rounds as that particular cipher uses.

- After the initial bock of plain text is split into two halves, the round function **F** is applied to one of the two halves. (The round function is a function performed with each iteration, or round, of the Feistel cipher.)

- The precise implementation of the round function **F** will vary with different ciphers DES, Blowfish, etc). They are usually simple functions that allow for increased speed of the algorithm.

- The output of the round function F is then XOR'd with the other half (the half that was not subjected to the round function).

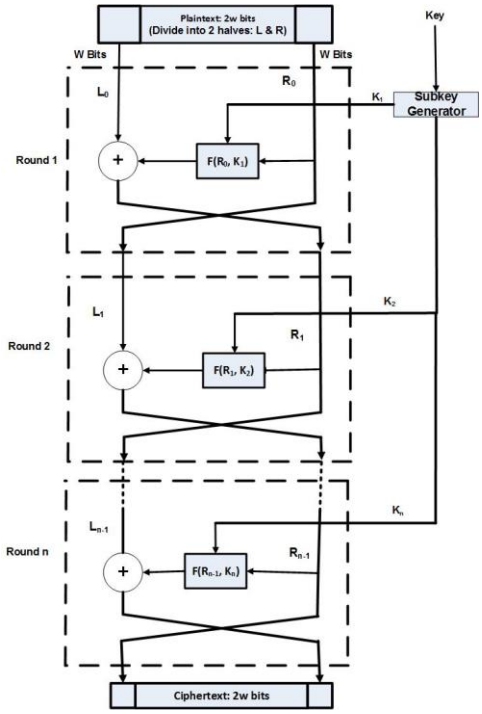
- What this means is that, for example, R_0 is passed through the round function F, and the result is XOR'd with L_0. Then the halves are transposed. So L_0 gets moved to the right and R_0 gets moved to the left.

- This process is repeated a given number of times. The main difference between various Feistel functions is the exact nature of the round function F and the number of iterations.

- All rounds have the same structure with the substitution is performed on the left half of the data. The swap at the very end is the permutation step.

Feistel Networks

- We can now use the basic block described earlier to build a full Feistel network. The next diagram depicts this architecture:



The Encryption Process

- The encryption process uses the Feistel structure consisting multiple rounds of processing of the plaintext, each round consisting of a **substitution** (S) step followed by a **permutation** (P) step.

- The input block to each round is divided into two halves that can be denoted as **L** and **R** for the left half and the right half.

- In each round, the right half of the block, R, goes through unchanged. But the left half, L, undergoes an operation that depends on R and an **encryption key**.

- In actual implementations of the Feistel Cipher, such as DES, instead of using the whole encryption key during each round, a round-dependent key (a **subkey)** is derived from the encryption key. This means that each round uses a different key, although all these subkeys are related to the original key.

- First, an encrypting function **F** is applied to two inputs – the key K and R. This function produces the output ***F(R, K)***.

- Next, the output of the mathematical function is XOR'd with L.

- Finally, the **permutation** step at the end of each round swaps the modified L and unmodified R. Therefore, the L for the next round would be R of the current round. And R for the next round will be the output L of the current round.

- Above substitution and permutation steps form a "**round**". The algorithm design usually dictates the number of rounds that will be implemented in the cipher.

- Once the last round is completed then the two sub blocks, R and L are concatenated in that order to form the output ciphertext block.

The Decryption Process

- The process of decryption in Feistel cipher is very similar. Instead of starting with a block of plaintext, the ciphertext block is fed into the start of the Feistel structure and the subsequent process is exactly the same as described in the encryption part.

- The process is said to be almost similar and not exactly same with one exception. The main difference is that the subkeys used in encryption are now used in the reverse order for the decryption.

- The final swapping of L and R in last step of the Feistel Cipher is essential. If these are not swapped then the resulting ciphertext cannot be decrypted using the same algorithm.

- In other words, to decrypt, we just use the round functions in the reverse order. Thus, the round functions **F** do not have to be invertible, and the Feistel structure lets us turn any one-way function into a block cipher.

- This means that we are less constrained in trying to choose a round function with good diffusion and confusion properties.

- It can more easily satisfy any other design constraints such as code size, table size, software speed, the hardware gate count, and so on.

Mathematical Definitions:

- We can define an iterated block cipher where the output of the **i**th round is determined from the output of the previous round as follows:

  $L_i = R_{i-1}$
  $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

- K_i is the subkey used in the **i**th round and **F** is an arbitrary round function.

- Because XOR is used to combine the left half with the output of the round function, it is necessarily true that:

  $L_{i-1} \oplus F(R_{i-1}, K_i) \oplus F(R_{i-1}, K_i) = L_{i-1}$

- A cipher that uses this construction is guaranteed to be invertible as long as the inputs to F in each round can be reconstructed.

- Thus, it does not matter what F is; F need not be invertible. The function F can be designed to be as complex as need be.

- Also, we do not have to implement two separate algorithms for encryption and another for decryption. The structure of a Feistel network takes care of all this automatically.

Design Features of Feistel Networks

- The most difficult part of designing a Feistel Cipher is the proper selection of round function **F**. In order to be unbreakable scheme, this function needs to have several important properties that are described next:

- **Block Size:** Larger block size means greater security (typically 64 bits).

- **Key Size**:56-128 bits.

- **Sub-key Generation Algorithms**: Greater complexity should lead to a greater difficulty of cryptanalysis.

- **Number of Rounds**:

    - A typical size is 16 rounds.
    - More number of rounds provides a more secure system.
    - However, more rounds results in an inefficient, slow encryption and decryption processes. Number of rounds in the systems thus depends upon efficiency-security tradeoffs.

- **Fast Software encryption/Decryption**: the speed of execution of the algorithm is important.

- **Ease of Analysis**: Important to be able to develop a higher level of assurance as to its strength

- **Decryption**: Ability to use the same algorithm as encryption with reversed keys.

- This basic structure can provide a very robust basis for cryptographic algorithms. The swapping of the two halves guarantees that some transposition occurs, regardless of what happens in the round function.

- The Feistel function is the basis for many algorithms, including DES, CAST-128, Blowfish, Twofish, RC5, FEAL (Fast data Encipherment ALgorithm), MARS, TEA (Tiny Encryption Algorithm), and others. But it was first used in IBM's Lucifer algorithm (the precursor to DES).