MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Classical Cryptographic Techniques - *"Fiestal Cipher"*

Lab Assignment 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

February 19, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement Feistal Cipher structure

2 Objectives

To understand the concepts of symmetric key cryptographic system.

3 Theory

3.1 Symmetric Key Cryptography

Symmetric key cryptography is a cryptographic system in which the same key is used for both encryption and decryption. The key is shared between the sender and the receiver. The sender encrypts the message using the key and sends it to the receiver. The receiver decrypts the message using the same key. The key is kept secret and is never sent along with the message.

The most commonly used symmetric key algorithm is the Data Encryption Standard (DES). It uses a 64-bit block size and a 56-bit key. The 64-bit block is divided into two halves of 32-bits each. The key is also divided into two halves of 28-bits each. The first half of the key is used to generate 16 subkeys. Each subkey is 48-bits long. The first 28-bits of the key are shifted left by 1 bit. The first 28-bits of the key are then shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. This process is repeated for the remaining 16 rounds. The 16 subkeys are then used to encrypt the message.

3.2 Feistal Cipher

The Feistal cipher is a symmetric key cryptographic system. It is a block cipher that uses a symmetric key. The key is shared between the sender and the receiver. The sender encrypts the message using the key and sends it to the receiver. The receiver decrypts the message using the same key. The key is kept secret and is never sent along with the message.

The Feistal cipher uses a 64-bit block size and a 56-bit key. The 64-bit block is divided into two halves of 32-bits each. The key is also divided into two halves of 28-bits each. The first half of the key is used to generate 16 subkeys. Each subkey is 48-bits long. The first 28-bits of the key are shifted left by 1 bit. The first 28-bits of the key are then shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. This process is repeated for the remaining 16 rounds. The 16 subkeys are then used to encrypt the message.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
The plain text, key
```

```
[1, 1, 1, 1, 0, 0, 1, 1] [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
The left and right keys are :  [1, 0, 1, 0, 0, 1, 0, 0] [0, 1, 0, 0, 0, 0, 1, 1]
Starting to cipher.
The cipher text is :  [0, 1, 0, 0, 0, 0, 0, 1]
```

6 Code

```python
# creating the fiester cipher.
# Assignment 2
# Krishnaraj Thadesar
# 10322108888 Batch A1

############ Defining Constants ###########

block_size = 8

binary_to_decimal = {(0, 0): 0, (0, 1): 1, (1, 0): 2, (1, 1): 3}

PT_IP_8 = [2, 6, 3, 1, 4, 8, 5, 7]
PT_IP_8_INV = [4, 1, 3, 5, 7, 2, 8, 6]

S0_MATRIX = [[1, 0, 3, 2],
             [3, 2, 1, 0],
             [0, 2, 1, 3],
             [3, 1, 3, 2]]

S1_MATRIX = [
    [0, 1, 2, 3],
    [2, 0, 1, 3],
    [3, 0, 1, 0],
    [2, 1, 0, 3],
]

############ Defining P Boxes ###########

PT_P_10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
PT_P_8 = [6, 3, 7, 4, 8, 5, 10, 9]
PT_P_4 = [2, 4, 3, 1]
PT_EP = [4, 1, 2, 3, 2, 3, 4, 1]


############ Functions ###########


def shift_left(list_to_shift):
    """Function to shift bits by 1 to the left

    Args:
        list_to_shift (list): list of the bunch of binary bits that you wanna
    shift to left.

    Returns:
        list: shifted list.
    """
    shifted_list = [i for i in list_to_shift[1:]]
    shifted_list.append(list_to_shift[0])
    return shifted_list
```

```python


def make_keys(key):
    """Function to Generate 8 bit K1 and 8 bit K2 from given 10 bit key.

    Args:
        key (list): list of 0's and 1's describing the key.

    Returns:
        (K1, K2): touple containing k1 and k2.
    """
    # make key_p10
    key_P10 = [key[i - 1] for i in PT_P_10]

    # Splitting into lshift and rshift
    key_P10_left = key_P10[: int(len(key) / 2)]
    key_P10_right = key_P10[int(len(key) / 2) :]

    # left shifting the key one time
    key_P10_left_shifted = shift_left(key_P10_left)
    key_P10_right_shifted = shift_left(key_P10_right)

    # temporarily combining the 2 shifted lists.
    temp_key = key_P10_left_shifted + key_P10_right_shifted
    # this gives the first key
    key_1 = [temp_key[i - 1] for i in PT_P_8]

    # now shifting the key 2 times for both left and right.
    key_P10_left_shifted = shift_left(key_P10_left_shifted)
    key_P10_left_shifted = shift_left(key_P10_left_shifted)

    key_P10_right_shifted = shift_left(key_P10_right_shifted)
    key_P10_right_shifted = shift_left(key_P10_right_shifted)

    temp_key = []
    temp_key = key_P10_left_shifted + key_P10_right_shifted

    key_2 = [temp_key[i - 1] for i in PT_P_8]
    # key_1, key_2 = 0, 0
    return (key_1, key_2)


def function_k(input_text, key):

    # splitting the plain text after applying initial permutation on it.
    PT_left_after_ip = input_text[: int(len(input_text) / 2)]
    PT_right_after_ip = input_text[int(len(input_text) / 2) :]

    # Applying Explansion Permutation on the right part of plain text after ip
    PT_right_after_EP = [PT_right_after_ip[i - 1] for i in PT_EP]

    # xoring the right part of pt after ep with key 1
    PT_after_XOR_with_key_1 = [x ^ y for x, y in zip(PT_right_after_EP, key)]

    # splitting the xor output of the right part of the plain text after ep.
    PT_after_XOR_with_key_1_left = PT_after_XOR_with_key_1[
        : int(len(PT_after_XOR_with_key_1) / 2)
    ]
    PT_after_XOR_with_key_1_right = PT_after_XOR_with_key_1[
```

```
            int(len(PT_after_XOR_with_key_1) / 2) :
    ]

    # getting the row and column number for S0 matrix.
    row_number_for_S0 = (
        PT_after_XOR_with_key_1_left[0],
        PT_after_XOR_with_key_1_left[-1],
    )

    col_number_for_S0 = (
        PT_after_XOR_with_key_1_left[1],
        PT_after_XOR_with_key_1_left[2],
    )

    # getting the row and column number for the S1 matrix.
    row_number_for_S1 = (
        PT_after_XOR_with_key_1_right[0],
        PT_after_XOR_with_key_1_right[-1],
    )

    col_number_for_S1 = (
        PT_after_XOR_with_key_1_right[1],
        PT_after_XOR_with_key_1_right[2],
    )

    # Getting the value from the S0 matrix.
    S0_value = S0_MATRIX[binary_to_decimal.get(row_number_for_S0)][
        binary_to_decimal.get(col_number_for_S0)
    ]

    # getting the value from the S1 matrix.
    S1_value = S1_MATRIX[binary_to_decimal.get(row_number_for_S1)][
        binary_to_decimal.get(col_number_for_S1)
    ]

    # converting the decimal numbers from s box output into binary.
    S0_value = list(binary_to_decimal.keys())[
        list(binary_to_decimal.values()).index(S0_value)
    ]
    S1_value = list(binary_to_decimal.keys())[
        list(binary_to_decimal.values()).index(S1_value)
    ]

    s_box_output = list(S0_value + S1_value)

    # applying P4 to s box output.
    s_box_output_after_P4 = [s_box_output[i - 1] for i in PT_P_4]

    # xoring the output of sbox after p4 with the left part of the plain text
    after ip.
    fk_xor_output = [x ^ y for x, y in zip(s_box_output_after_P4, PT_left_after_ip
    )]

    fk_concat_output_8_bit = fk_xor_output + PT_right_after_ip

    return fk_concat_output_8_bit


def encrypt_fiestal_cipher(plain_text, key_1, key_2):
```

```python
    print("Starting to cipher. ")

    # Initial permutation for the plain text
    plain_text_after_ip = [plain_text[i - 1] for i in PT_IP_8]

    # getting partial output from running f(k) with key 1
    output_1_function_k = function_k(plain_text_after_ip, key_1)

    # splitting that output.
    output_1_function_k_left = output_1_function_k[:4]
    output_1_function_k_right = output_1_function_k[4:]

    # switching that output.
    temp = output_1_function_k_right + output_1_function_k_left

    # running function again with switched output from running f(k) with key 2
    output_2_function_k = function_k(temp, key_2)

    # running IP Inverse on it.
    cipher_text = [output_2_function_k[i - 1] for i in PT_IP_8_INV]

    return cipher_text


def decrypt_fiestal_cipher(cipher_text, key_1, key_2):
    print("Starting to Decipher. ")

    # Initial permutation for the cipher text
    cipher_text_after_ip = [cipher_text[i - 1] for i in PT_IP_8_INV]

    # getting partial output from running f(k) with key 2
    output_1_function_k = function_k(cipher_text_after_ip, key_2)

    # splitting that output.
    output_1_function_k_left = output_1_function_k[:4]
    output_1_function_k_right = output_1_function_k[4:]

    # switching that output.
    temp = output_1_function_k_right + output_1_function_k_left

    # running function again with switched output from running f(k) with key 1
    output_2_function_k = function_k(temp, key_1)

    # running IP Inverse on it.
    deciphered_plain_text = [output_2_function_k[i - 1] for i in PT_IP_8]

    return deciphered_plain_text


def main():

    # this will make the plaintext a list.
    # plain_text = [int(i) for i in input("Enter the Plain text with spaces: ").
    split()]
    # key = [int(i) for i in input("Enter the Key with spaces: ").split()]
    plain_text = [1, 1, 1, 1, 0, 0, 1, 1]
    key = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
    print("The plain text, key")
    print(plain_text, key)
```

```

    key_1, key_2 = make_keys(key)
    print("The left and right keys are : ", key_1, key_2)

    # Generating the Cipher text.
    cipher_text = encrypt_fiestal_cipher(plain_text, key_1, key_2)
    print("The cipher text is : ", cipher_text)


main()
```

Listing 1: "Fiestal Cipher"

7 Conclusion

Thus, learnt about the different kinds of ciphers, classical cryptographic techniques, and how to implement some of them in python.

8 FAQ

1. **Differentiate between stream and block ciphers.**
   Answer:

   (a) Stream ciphers encrypt the data one bit at a time. Block ciphers encrypt the data in blocks of fixed size.

   (b) Stream ciphers are faster than block ciphers.

   (c) Block ciphers are more secure than stream ciphers.

   (d) Stream ciphers are more suitable for real-time applications.

   (e) Block ciphers are more suitable for bulk data encryption.

   (f) Stream ciphers are more suitable for applications where the data is encrypted and decrypted in a single pass.

   (g) Block ciphers are more suitable for applications where the data is encrypted and decrypted in multiple passes.

2. **Write advantages and disadvantages of DES algorithm.**
   Answer: Advantages:

   (a) It is a fast, simple, efficient, and secure algorithm.

   (b) The algorithm has been in use since 1977. Technically, no weaknesses have been found in the algorithm. Brute force attacks are still the most efficient attacks against the DES algorithm.

   (c) DES is the standard set by the US Government. The government recertifies DES every five years, and has to ask for its replacement if the need arises.

   (d) The American National Standards Institute (ANSI) and International Organization for Standardization (ISO) have declared DES as a standard as well. This means that the algorithm is open to the public—to learn and implement.

   (e) DES was designed for hardware; it is fast in hardware, but only relatively fast in software.

   Disadvantages:

   (a) Probably the biggest disadvantage of the DES algorithm is the key size of 56-bit. There are chips available that can encrypt and decrypt a million DES operations in a second. A DES cracking machine that can search all the keys in about seven hours is available for 1 million.

   (b) DES can be implemented quickly on hardware. But since it was not designed for software, it is relatively slow on it.

   (c) It has become easier to break the encrypted code in DES as the technology is steadily improving. Nowadays, AES is preferred over DES.

   (d) DES uses a single key for encryption as well as decryption as it is a type of symmetric encryption technique. In case that one key is lost, we will not be able to receive decipherable data at all.

3. **Explain block cipher modes of operations.**
   Answer:

(a) Electronic Code Book (ECB) - In this mode, the plaintext is divided into blocks of equal size. Each block is encrypted separately. The same plaintext block will always result in the same ciphertext block. This mode is not secure as it is vulnerable to a known plaintext attack.

(b) Cipher Block Chaining (CBC) - In this mode, the plaintext is divided into blocks of equal size. Each block is XORed with the previous ciphertext block before being encrypted. This mode is more secure than ECB as it is not vulnerable to a known plaintext attack.

(c) Cipher Feedback (CFB)

(d) Output Feedback (OFB)

(e) Counter (CTR)