MIT World Peace University

Information and Cyber Security

Assignment 2

Naman Soni Roll No. 10

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 *Explain Symmetric Key Cryptography*

Symmetric key cryptography is a type of encryption scheme in which the similar key is used both to encrypt and decrypt messages. Such an approach of encoding data has been largely used in the previous decades to facilitate secret communication between governments and militaries.
Symmetric-key cryptography is called a shared-key, secret-key, single-key, one-key and eventually private-key cryptography. With this form of cryptography, it is clear that the key should be known to both the sender and the receiver that the shared. The complexity with this approach is the distribution of the key.
Symmetric key cryptography schemes are usually categorized such as stream ciphers or block ciphers. Stream ciphers work on a single bit (byte or computer word) at a time and execute some form of feedback structure so that the key is repeatedly changing.
A block cipher is so-called because the scheme encrypts one block of information at a time utilizing the same key on each block. In general, the same plaintext block will continually encrypt to the same ciphertext when using the similar key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

Block cipher can operate in one of several modes which are as follows:

1. Electronic Codebook (ECB) mode is the simplest application and the shared key can be used to encrypt the plaintext block to form a ciphertext block. There are two identical plaintext blocks will always create the same ciphertext block. Although this is the most common mode of block ciphers, it is affected to multiple brute-force attacks.

2. Cipher Block Chaining (CBC) mode insert a feedback structure to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the prior ciphertext block prior to encryption. In this mode, there are two identical blocks of plaintext not encrypt to the similar ciphertext.

3. Cipher Feedback (CFB) mode is a block cipher implementation as a selfsynchronizing stream cipher. CFB mode enable data to be encrypted in units lower than the block size, which can be beneficial in some applications including encrypting interactive terminal input. If it is using 1-byte CFB mode.

4. Output Feedback (OFB) mode is a block cipher implementation conceptually same to a synchronous stream cipher. OFB avoids the similar plaintext block from making the same ciphertext block by using an internal feedback structure that is independent of both the plaintext and ciphertext bitstreams.

3.2 *Explain Feistal Cipher*

A Feistel Cipher is a block ciper that uses a symmetric structure. In this structure, the ciphertext is produced from the plaintext input through a sequence of round functions. Each round function takes two inputs, a 32-bit half-block and a subkey, and produces one output. These rounds are followed by a post-processing step which permutes the bits in the ciphertext block to produce the final output.
Although the number of rounds can vary, a typical Feistel cipher has sixteen rounds. The subkeys used in the round functions are often derived by applying a so-called key schedule algorithm to a user-supplied key.

The security of a Feistel cipher lies largely in the design of the round functions and key scheduling algorithm.As long as they remain secret and are designed properly, the cipher will be secure.

Feistal cipher algorithm

- Create a list of all the Plain Text characters.
- Convert the Plain Text to Ascii and then 8-bit binary format.
- Divide the binary Plain Text string into two halves: left half (L1)and right half (R1)
- Generate a random binary keys (K1 and K2) of length equal to the half the length of the Plain Text for the two rounds.

4 Programming Language Used

Python

5 Code

```
# creating the fiester cipher.
# Assignment 2
# Naman Soni
# Roll No. PA-10
# 1032210715 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


    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):
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 plain 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)

    # # Decrypting the cipher text.
    # deciphered_plain_text = decrypt_fiestal_cipher(cipher_text, key_1, key_2)
    # print("The deciphered plain text is : ", deciphered_plain_text)

    # DECRYPTING


main()
```

Listing 1: Input Code

```
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]
```

Listing 2: Output Result

6 Conclusion

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

7 FAQ's

1. Differentiate between stream and block ciphers.

   Ans. Stream ciphers and block ciphers are both types of symmetric encryption algorithms. The main difference between them is in their input and output. Stream Ciphers: Stream ciphers take a continuous stream of data (known as a keystream) as input and encrypts plaintext into ciphertext based on that key stream. It only processes one bit (or byte) at a time. This makes it very fast and its strong security depends on the security of the keystream. Examples of stream ciphers are RC4 and Salsa20.

   Block Ciphers: Block ciphers take a fixed-length block of data as input and encrypts it into a fixed-length block of ciphertext. They generally use a combination of two transformations: substitution and permutation. Common examples of block ciphers are AES and Triple DES. They are slower than stream ciphers and can be used for much longer blocks of data.

2. Write advantages and disadvantages of DES Algorithm.

   Ans. Advantages of DES Algorithm
   It is a secure algorithm with good encryption and decryption capabilities. The algorithm has been tested and verified extensively, thus making it more reliable. It is efficient, especially when used in hardware implemen- tation. DES is quite flexible and suitable for many different applications.

   Disadvantages of DES Algorithm
   The 56-bit key size is deemed to be too small, which significantly impacts the security of DES. It is vulnerable to brute force attacks where an attacker can use various methods to crack the key. Due to its complexity, DES may require additional power for hardware implementation.

3. Explain block cipher modes of operations.

   Ans. Block cipher modes of operation are algorithms used to encrypt data that is larger than the block size. These modes are designed to offer a higher level of security while also addressing potential issues such as:

   - Message repetition
   - Message length
   - Message integrity
   - Ciphertext malleability

   The four most common modes of operations for block ciphers are:

(a) **CBC (Cipher Block Chaining)** In CBC mode, each plaintext block is XORed with the previous ci- phertext block before it's encrypted. This prevents identical plaintext blocks from producing identical ciphertext, thus reducing malicious manipulation.

(b) **ECB (Electronic Code Book) ECB** mode uses the same encryption key for every plaintext block, allowing for improved performance. However, this can lead to message repetition and weakened system leverage against certain types of attacks.

(c) **CTR (Counter Mode)** CTR mode generates a unique key stream for each block of plaintext, which allows for parallel processing. The counter value is encrypted and then XORed with the plaintext block to produce the ciphertext.

(d) **OFB(OutputFeedback)** OFBmodecreatesauniquekeystreamforeachplaintextblockbygenerating random bits from the cipher. After the key stream is generated, it is XORed with the plaintext block to produce the ciphertext.