Cryptography and Network Security Lab

Assignment 9

Student Details

Name : Krunal Rank
Adm No. : U18CO081

1

```
import os
import random
from typing import final


class CONSTANTS:
   SUBKEY_SIZE = 48
   KEY_SIZE = 64
   MAX_ROUNDS = 16
   ROUNDS_SHIFT = [1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28]

   PC_1_TABLE = [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, 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]
   PC_1_KEY_SIZE = 56

   PC_2_TABLE = [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]
   PC_2_KEY_SIZE = 48

   E_FUNCTION = [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]
   E_FUNCTION_SIZE = 48

   P_FUNCTION = [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]
   P_FUNCTION_SIZE = 32
```

```
    S_BOX = [[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
             [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
             [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
             [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]]

    INITIAL_PERMUTATION = [58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12,
4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32,
                           24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,
27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7]
    INITIAL_PERMUTATION_SIZE = 64

    FINAL_PERMUTATION = [40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,
38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53,
                         21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51,
19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25]
    FINAL_PERMUTATION_SIZE = 64

class ERRORS:
    INVALID_CHOICE = "Invalid choice. Please select a valid choice."
    FILE_NOT_FOUND = "File not found."


def get_bit(key, bit_index, size=CONSTANTS.KEY_SIZE):
    """
    Returns bit at bit_index from key
        key contains size bits
        bit_index ranges from 1 to size
    """
    return (key >> (size - bit_index)) & 1


def initial_permutation(plain_text):
    """
    Performs Initial Permutation for DES Encryption
        plain_text contains 64 bit int
    Returns permuted plain_text as 64 bit int
    """
    initial_permuted_text = 0
    for i in range(CONSTANTS.INITIAL_PERMUTATION_SIZE):
        initial_permuted_text |= get_bit(plain_text, CONSTANTS.INITIAL_PERMUTATION[i],
CONSTANTS.INITIAL_PERMUTATION_SIZE) << (
            CONSTANTS.INITIAL_PERMUTATION_SIZE - i - 1)
    return initial_permuted_text


def final_permutation(plain_text):
```

```
    """
    Performs Final Permutation for DES Encryption
        plain_text contains 64 bit int
    Returns permuted plain_text as 64 bit int
    """
    final_permuted_text = 0
    for i in range(CONSTANTS.FINAL_PERMUTATION_SIZE):
        final_permuted_text |= get_bit(plain_text, CONSTANTS.FINAL_PERMUTATION[i],
CONSTANTS.FINAL_PERMUTATION_SIZE) << (
            CONSTANTS.FINAL_PERMUTATION_SIZE - i - 1)
    return final_permuted_text


def pc_1(key):
    """
    Returns Permutation Choice 1 of key
    Returns key as 56 bit int
    """
    pc_1_key = 0
    for i in range(CONSTANTS.PC_1_KEY_SIZE):
        pc_1_key |= get_bit(key, CONSTANTS.PC_1_TABLE[i]) << (
            CONSTANTS.PC_1_KEY_SIZE - i - 1)
    return pc_1_key


def pc_2(key):
    """
    Returns Permutation Choice 2 of key
    Returns key as 48 bit int
    """
    pc_2_key = 0
    for i in range(CONSTANTS.PC_2_KEY_SIZE):
        pc_2_key |= get_bit(key, CONSTANTS.PC_2_TABLE[i], CONSTANTS.PC_1_KEY_SIZE) <<
(
            CONSTANTS.PC_2_KEY_SIZE - i - 1)
    return pc_2_key


def left_rotate(n, d, bits=CONSTANTS.KEY_SIZE):
    """
    Left Rotates a number n by d bits
    """
    return (n << d) | (n >> (bits - d))


def key_schedule(key, round):
```

```
    """
    Generates SUBKEY_SIZE bit subkey out of KEY_SIZE bit key for given round
    """
    pc_1_key = pc_1(key)

    left_key = pc_1_key >> CONSTANTS.PC_1_KEY_SIZE//2
    right_key = pc_1_key & (2**CONSTANTS.PC_1_KEY_SIZE//2 - 1)

    left_key = left_rotate(
        left_key, CONSTANTS.ROUNDS_SHIFT[round], CONSTANTS.PC_1_KEY_SIZE//2)
    right_key = left_rotate(
        right_key, CONSTANTS.ROUNDS_SHIFT[round], CONSTANTS.PC_1_KEY_SIZE//2)

    combined_key = left_key << CONSTANTS.PC_1_KEY_SIZE//2 | right_key

    return pc_2(combined_key)


def e_function(plain_text):
    """
    Performs E Function on plain_text
        plain_text contains 32 bit int
    Returns permuted plain_text as 48 bit int
    """
    e_plain_text = 0
    for i in range(CONSTANTS.E_FUNCTION_SIZE):
        e_plain_text |= get_bit(plain_text, CONSTANTS.E_FUNCTION[i], 32) << (
            CONSTANTS.E_FUNCTION_SIZE - i - 1)
    return e_plain_text


def p_function(plain_text):
    """
    Performs P function on plain_text
        plain_text contains 32 bit int
    Returns permuted plain text as 32 bit int
    """
    p_plain_text = 0
    for i in range(CONSTANTS.P_FUNCTION_SIZE):
        p_plain_text |= get_bit(plain_text, CONSTANTS.P_FUNCTION[i], 32) << (
            CONSTANTS.P_FUNCTION_SIZE - i - 1)
    return p_plain_text


def s_function(plain_text):
    """
```

```
    Performs S Box Substitution on plain_text
        plain_text contains 48 bit int
    Returns substituted plain_text as 32 bit int
    """
    subsituted_plain_text = 0
    for i in range(8):
        row = (get_bit(plain_text, 6*i + 1, CONSTANTS.PC_2_KEY_SIZE) <<
               1) | get_bit(plain_text, 6*i + 6, CONSTANTS.PC_2_KEY_SIZE)
        column = (get_bit(plain_text, 6*i + 2, CONSTANTS.PC_2_KEY_SIZE) << 3) |
(get_bit(plain_text, 6*i + 3, CONSTANTS.PC_2_KEY_SIZE) << 2) | (
            get_bit(plain_text, 6*i + 4, CONSTANTS.PC_2_KEY_SIZE) << 1) |
get_bit(plain_text, 6*i + 5, CONSTANTS.PC_2_KEY_SIZE)
        subsituted_plain_text |= (CONSTANTS.S_BOX[row][column] << 4*(7-i))

    return subsituted_plain_text


def des_round(plain_text, key, round):
    """
    Performs one round of DES on plain_text and key for given round
        plain_text contains 64 bit int
        key contains 64 bit int
        round is a value between 0 and CONSTANTS.MAX_ROUNDS - 1
    Returns cipher text as 64 bit int
    """
    subkey = key_schedule(key, round)

    left_plain_text = plain_text >> 32
    right_plain_text = plain_text & (2**32 - 1)
    final_left_plain_text = left_plain_text ^ p_function(
        s_function(subkey ^ e_function(right_plain_text)))
    final_right_plain_text = right_plain_text

    return final_left_plain_text << 32 | final_right_plain_text


def segment_des_encrypt(plain_text, key):
    """
    Performs 16 rounds of DES on plain_text and key
        plain_text contains 64 bit int
        key contains 64 bit int
    Returns cipher text as 64 bit int
    """

    cipher_text = initial_permutation(plain_text)
    for i in range(CONSTANTS.MAX_ROUNDS):
```

```
        cipher_text = des_round(cipher_text, key, i)

    return final_permutation(cipher_text)


def segment_des_decrypt(cipher_text, key):
    """
    Performs 16 rounds of DES on cipher_text and key
        cipher_text contains 64 bit int
        key contains 64 bit int
    Returns plain text as 64 bit int
    """

    plain_text = initial_permutation(cipher_text)
    for i in range(CONSTANTS.MAX_ROUNDS):
        plain_text = des_round(plain_text, key, CONSTANTS.MAX_ROUNDS - (i+1))

    return final_permutation(plain_text)


def des_encrypt(plain_text_file_path, encrypted_text_file_path):
    """
    Encrypts data in plain_text_file_path and saves encrypted data in provided file
path
    Returns generated key used for encryption
    """
    key = random.getrandbits(64)

    if not os.path.exists(plain_text_file_path):
        raise Exception(ERRORS.FILE_NOT_FOUND)

    plain_text = open(plain_text_file_path, "rb").read()
    plain_text_ints = [int(x, 16) for x in plain_text.hex(
        ":", bytes_per_sep=8).split(":")]

    cipher_text_ints = []

    for plain_text_int in plain_text_ints:
        cipher_text_ints.append(segment_des_encrypt(plain_text_int, key))

    final_cipher_int = 0
    for i in range(len(cipher_text_ints)):
        final_cipher_int |= cipher_text_ints[i] << 64 * \
            (len(cipher_text_ints) - (i+1))
    hex_string = hex(final_cipher_int)[2:]
    hex_string = hex_string if len(hex_string) % 2 == 0 else "0" + hex_string
```

```
    encrypted_text = open(encrypted_text_file_path, "wb")
    encrypted_text.write(bytes.fromhex(hex_string))
    encrypted_text.close()

    return key


def des_decrypt(encrypted_text_file_path, plain_text_file_path, key):
    """
    Decrypts data in encrypted_text_file_path and saves encrypted data in provided
plain_text_file_path
    """

    if not os.path.exists(encrypted_text_file_path):
        raise Exception(ERRORS.FILE_NOT_FOUND)

    encrypted_text = open(encrypted_text_file_path, "rb").read()
    encrypted_text_ints = [int(x, 16) for x in encrypted_text.hex(
        ":", bytes_per_sep=8).split(":")]

    plain_text_ints = []
    for i in range(len(encrypted_text_ints)):
        plain_text_ints.append(
            segment_des_decrypt(encrypted_text_ints[i], key))

    final_plain_text_int = 0
    for i in range(len(plain_text_ints)):
        final_plain_text_int |= plain_text_ints[i] << 64 * \
            (len(plain_text_ints) - (i+1))

    hex_string = hex(final_plain_text_int)[2:]
    hex_string = hex_string if len(hex_string) % 2 == 0 else "0" + hex_string

    plain_text = open(plain_text_file_path, "wb")
    plain_text.write(bytes.fromhex(hex_string))
    plain_text.close()


def des_encrypt_dialog():
    """
    Runs DES Encryption Dialog
    """
    plain_text_file_path = input("Enter Plain Text File Path: ")
    encrypted_text_file_path = input("Enter Encrypted Text File Path: ")
    key = des_encrypt(plain_text_file_path, encrypted_text_file_path)
```

```
    print(
        f"DES encryption complete. Please check {encrypted_text_file_path} for the
cipher text.\nUse {key} as decryption key.")


def des_decrypt_dialog():
    """
    Runs DES Decryption Dialog
    """
    encrypted_text_file_path = input("Enter Encrypted Text File Path: ")
    plain_text_file_path = input("Enter Plain Text File Path: ")
    key = int(input("Enter Key: "))

    des_decrypt(encrypted_text_file_path, plain_text_file_path, key)

    print(
        f"DES decryption complete. Please check {plain_text_file_path} for the
decrypted text.")


def main_dialog():
    """
    Runs main dialog
    """
    choice = int(
        input(("DES Program\n1. Encrypt\n2. Decrypt\nEnter your choice: ")))

    if choice == 1:
        des_encrypt_dialog()
    elif choice == 2:
        des_decrypt_dialog()
    else:
        raise Exception(ERRORS.INVALID_CHOICE)


if __name__ == "__main__":
    try:
        main_dialog()
    except Exception as e:
        print(e)
```

Plain Text:
Hi, this is some random text right here

Encrypted Text:
QH=|00m|#`9c 2k8u`2qn1k8141(t0v<v8 0(1w0

Key:
8002777519580698261