Cryptography and Network Security Lab

Assignment 6

Student Details

Name : Krunal Rank
Adm No. : U18CO081

1

```
# sample text : once upon a time there was a little girl named goldilocks she went
for a walk in the forest pretty soon she came upon a house she knocked and when no
one answered she walked right in at the table in the kitchen there were three bowls
of porridge goldilocks was hungry she tasted the porridge from the first bowl
import math
MOD = 27


class ERRORS:
    """
    Error Messages
    """
    INVALID_CHOICE = "Please enter a valid Integer choice."
    INVALID_KEY_DIM = "Please enter a valid Key Dimension. A valid Key Dimension must
be an integer between 1 and 10."
    INVALID_KEY = "Please enter a valid Key. A valid Key must be a square matrix which
is inversible. The Key determinant must have a modular inverse."
    INVALID_TEXT = "Please enter a valid Text. Text must only contain Lowercase
Alphabets."
    INVALID_MATRIX = "Given Matrices are invalid for Multiplication."
    INVALID_MATRIX_MUL_COMBO = "Matrices cannot be multiplied. Number of columns of
first matrix must be equal to number of rows of second matrix."


class UtilityHelper:
    """
    Contains Utility Methods
    """

    @staticmethod
    def mod_inverse(a):
        """
        Returns Modular Inverse of a
```

```
        """
        for i in range(MOD):
            if (a*i) % MOD == 1:
                return i
        return -1

    @staticmethod
    def mod_add(a, b):
        """
        Returns Modular Addition of a and b
        """
        return (a+b) % MOD

    @staticmethod
    def mod_expo(a, b):
        """
        Returns Modular Exponentiation of a^b
        """
        result = 0
        while b > 0:
            if b % 2 == 1:
                result = UtilityHelper.mod_add(result, a)
            a = UtilityHelper.mod_mul(a, a)
            b //= 2
        return result

    @staticmethod
    def mod_sub(a, b):
        """
        Returns Modular Subtraction of a and b
        """
        return (a-b+MOD) % MOD

    @staticmethod
    def mod_mul(a, b):
        """
        Returns Modular Multiplication of a and b
        """
        return (a % MOD*b % MOD) % MOD

    @staticmethod
    def get_determinant(key):
        """
        Calculates Determinant of 2 Dimensional Square Key Matrix
        """
        if len(key) < 0:
```

```
            raise Exception(ERRORS.INVALID_KEY)
        if len(key) == 1:
            return key[0][0]
        determinant = 0
        for j in range(len(key)):
            new_key = []
            for k in range(len(key)):
                if k == 0:
                    continue
                new_key_row = []
                for l in range(len(key)):
                    if l == j:
                        continue
                    new_key_row.append(key[k][l])
                new_key.append(new_key_row)
            determinant += ((-1)**j) * key[0][j] * \
                UtilityHelper.get_determinant(new_key)
        return determinant

    @staticmethod
    def get_mod_determinant(key):
        """
        Gets Modular Determinant of Key
        """
        return UtilityHelper.get_determinant(key) % MOD

    @staticmethod
    def get_matrix_inverse(key):
        """
        Calculates Matrix inverse of the 2D Matrix given as Key
        """
        determinant = UtilityHelper.get_mod_determinant(key)
        if determinant == 0:
            raise Exception(ERRORS.INVALID_KEY)
        determinant_inverse = UtilityHelper.mod_inverse(determinant)
        if determinant_inverse == -1:
            raise Exception(ERRORS.INVALID_KEY)
        inverse = [[0 for _ in range(len(key))] for _ in range(len(key))]
        for i in range(len(key)):
            for j in range(len(key)):
                minor = []
                for k in range(len(key)):
                    if k == i:
                        continue
                    minor_row = []
                    for l in range(len(key)):
```

```
                    if l == j:
                        continue
                    minor_row.append(key[k][l])
                minor.append(minor_row)
            minor_cofactor = (-1)**(i+j) * \
                UtilityHelper.get_determinant(minor)
            inverse[j][i] = UtilityHelper.mod_mul(
                determinant_inverse, minor_cofactor)
    return inverse

@staticmethod
def get_matrix_mul(a, b):
    """
    Returns a*b Matrix Multiplication
    """
    if len(a) == 0 or len(b) == 0:
        raise Exception(ERRORS.INVALID_MATRIX)
    if len(a[0]) != len(b):
        raise Exception(ERRORS.INVALID_MATRIX_MUL_COMBO)
    result = [[0 for _ in range(len(b[0]))] for _ in range(len(a))]
    for i in range(len(a)):
        for j in range(len(b[0])):
            for k in range(len(b)):
                result[i][j] = UtilityHelper.mod_add(
                    result[i][j], UtilityHelper.mod_mul(a[i][k], b[k][j]))
    return result


def hill_cipher_encrypt(text: str, key) -> str:
    """
    Hill Cipher Encryption using given Key Matrix
    Returns:
        Encrypted Text
    """
    if len(key) == 0 or len(key) != len(key[0]):
        raise Exception(ERRORS.INVALID_KEY)

    determinant = UtilityHelper.get_determinant(key)
    if determinant == 0 or UtilityHelper.mod_inverse(determinant) == -1:
        raise Exception(ERRORS.INVALID_KEY)

    key_dim = len(key)
    text = text.lower()
    text = text.replace(' ', '{')
    text_matrix = [
        [0 for _ in range(int(math.ceil(len(text)/key_dim)))] for _ in range(key_dim)]
```

```
    for i in range(len(text)):
        text_matrix[i % key_dim][i//key_dim] = ord(text[i]) - ord('a')

    cipher_matrix = UtilityHelper.get_matrix_mul(key, text_matrix)

    encrypted_chars = []

    for j in range(len(cipher_matrix[0])):
        for i in range(len(cipher_matrix)):
            encrypted_chars.append(chr(cipher_matrix[i][j] + ord('a')))

    return "".join(encrypted_chars).replace('{', ' ')


def hill_cipher_decrypt(text: str, key) -> str:
    """
    Hill Cipher Decryption using given Key Matrix
    Returns:
        Decrypted Text
    """
    if len(key) == 0 or len(key) != len(key[0]):
        raise Exception(ERRORS.INVALID_KEY)

    if UtilityHelper.get_determinant(key) == 0:
        raise Exception(ERRORS.INVALID_KEY)

    key = UtilityHelper.get_matrix_inverse(key)
    key_dim = len(key)

    text = text.lower()
    text = text.replace(' ', '{')
    text_matrix = [
        [0 for _ in range(int(math.ceil(len(text)/key_dim)))] for _ in range(key_dim)]
    for i in range(len(text)):
        text_matrix[i % key_dim][i//key_dim] = ord(text[i]) - ord('a')

    cipher_matrix = UtilityHelper.get_matrix_mul(key, text_matrix)

    decrypted_chars = []

    for j in range(len(cipher_matrix[0])):
        for i in range(len(cipher_matrix)):
            decrypted_chars.append(chr(cipher_matrix[i][j] + ord('a')))

    return "".join(decrypted_chars).replace('{', ' ')
```

```
def hill_cipher_encrypt_dialog():
    """
    Runs Hill Cipher Encryption Dialog
    """
    text = input("Enter text to be encrypted: ")
    key_dim = int(input("Enter Key Matrix Dimension : "))
    if key_dim < 1 or key_dim > 10:
        raise Exception(ERRORS.INVALID_KEY_DIM)
    key = [[0 for _ in range(key_dim)] for _ in range(key_dim)]
    for i in range(key_dim):
        for j in range(key_dim):
            key[i][j] = int(input(f"Enter Key Matrix Element ({i},{j}): "))

    for i in range(key_dim):
        for j in range(key_dim):
            key[i][j] %= MOD

    encrypted_text = hill_cipher_encrypt(text, key)
    print(f"Encrypted Text: {encrypted_text}")


def hill_cipher_decrypt_dialog():
    """
    Runs Hill Cipher Decryption Dialog
    """
    text = input("Enter text to be decrypted: ")
    key_dim = int(input("Enter Key Matrix Dimension : "))
    if key_dim < 1 or key_dim > 10:
        raise Exception(ERRORS.INVALID_KEY_DIM)
    key = [[0 for _ in range(key_dim)] for _ in range(key_dim)]
    for i in range(key_dim):
        for j in range(key_dim):
            key[i][j] = int(input(f"Enter Key Matrix Element ({i},{j}): "))

    for i in range(key_dim):
        for j in range(key_dim):
            key[i][j] %= MOD

    decrypted_text = hill_cipher_decrypt(text, key)
    print(f"Decrypted Text: {decrypted_text}")


def main_dialog():
    """
    Runs main dialog sequence
```

```
    """
    try:
        choice = int(input(
            "Hill Cipher Program\n1. Encrypt\n2. Decrypt\nPlease enter your choice:
"))
    except Exception as e:
        raise Exception(ERRORS.INVALID_CHOICE)

    if choice == 1:
        hill_cipher_encrypt_dialog()
    elif choice == 2:
        hill_cipher_decrypt_dialog()
    else:
        raise Exception(ERRORS.INVALID_CHOICE)



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

```
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_6$ python3 1.py
Hill Cipher Program
1. Encrypt
2. Decrypt
Please enter your choice: 1
Enter text to be encrypted: once upon a time there was a little girl named goldilocks she went for a walk in the forest pretty
soon she came upon a house she knocked and when no one answered she walked right in at the table in the kitchen there were thre
e bowls of porridge goldilocks was hungry she tasted the porridge from the first bowl
Enter Key Matrix Dimension : 2
Enter Key Matrix Element (0,0): 1
Enter Key Matrix Element (0,1): 2
Enter Key Matrix Element (1,0): 3
Enter Key Matrix Element (1,1): 4
Encrypted Text: nnkwmxqulizxiiuzktpkznqejs yvottouciwxmozwyvkylvjftomiwtqxfbcidbyhjrvc yqewri hwktpkjrvcndr wfphnsipprlifbcicgu
zmxqulizxixcycifbcijbsxstbf zbfjndkzwmlnnci zihl kyippkqewrstbfgcutr hw yr gecitdxucihwktpktkttqhdkktpkznqel cigezncicfrcqxyicd
vcgcpgcihursdosxtvqejsnztenfwofbcitdcwkyktpkcdvcgcpgcimcljktpkjrplcwbbewlg
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_6$ ^C
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_6$ python3 1.py
Hill Cipher Program
1. Encrypt
2. Decrypt
Please enter your choice: 2
Enter text to be decrypted: nnkwmxqulizxiiuzktpkznqejs yvottouciwxmozwyvkylvjftomiwtqxfbcidbyhjrvc yqewri hwktpkjrvcndr wfphnsi
pprlifbcicguzmxqulizxixcycifbcijbsxstbf zbfjndkzwmlnnci zihl kyippkqewrstbfgcutr hw yr gecitdxucihwktpktkttqhdkktpkznqel cigezn
cicfrcqxyicdvcgcpgcihursdosxtvqejsnztenfwofbcitdcwkyktpkcdvcgcpgcimcljktpkjrplcwbbewlg
Enter Key Matrix Dimension : 2
Enter Key Matrix Element (0,0): 1
Enter Key Matrix Element (0,1): 2
Enter Key Matrix Element (1,0): 3
Enter Key Matrix Element (1,1): 4
Decrypted Text: once upon a time there was a little girl named goldilocks she went for a walk in the forest pretty soon she cam
e upon a house she knocked and when no one answered she walked right in at the table in the kitchen there were three bowls of p
orridge goldilocks was hungry she tasted the porridge from the first bowla
```