Cryptography and Information Security Lab

Course code: CSEL – 4110

Submitted By

Farhana Akter Suci

ID: B190305001

Submitted To

DR. MOHAMMED NASIR UDDIN

Professor(Grade-II), Department of CSE

1.Caesar Cipher or Additive Cipher or Shift Cipher

The encryption formula is En(x) = (x + n) mod 26 and the Decryption formula is Dn(x) = (x – n) mod 26.

```
#Encryption Function
def encryption(plaintext, key):
    text = plaintext.lower()
    #Range of lowercase letter is 97 to 122
    ciphertext = ""
    for char in text:
        uniCode = ord(char)
        if uniCode >= 97 and uniCode <= 122:
            value = uniCode - 97
            valueAfterShifting = (value + key) % 26
            uniCode = valueAfterShifting + 97
            new_char = chr(uniCode).upper()
            ciphertext = ciphertext + new_char
        else:
            ciphertext = ciphertext + char
    return ciphertext
```

```
#Decryption Function
def decryption(ciphertext, key):
    text = ciphertext
    #Range of uppercase letter is 65 to 90
    plaintext = ""
    for char in text:
        uniCode = ord(char)
        if uniCode >= 65 and uniCode <= 90:
            value = uniCode - 65
            valueAfterShifting = (value - key) % 26
            uniCode = valueAfterShifting + 65+32
            new_char = chr(uniCode)
            plaintext = plaintext + new_char
        else:
            plaintext = plaintext + char
    return plaintext
```

```
#Input Section
plaintext = input("Enter the plaintext: ")
key = int(input("Enter the key: "))

#Function Calling
ciphertext = encryption(plaintext, key)
decrypted_text = decryption(ciphertext, key)

#Output Section
print("Given plaintext: ", plaintext)
print("Entered key : ", key)
print("Ciphertext: ", ciphertext)
print("Decrypted plaintext: ", decrypted_text)
```

```
Enter the plaintext:  jnucse
Enter the key:  5
Given plaintext:  jnucse
Entered key :  5
Ciphertext:  OSZHXJ
Decrypted plaintext:  jnucse
```

2.Vigenere Cipher

The main difference between the Caesar cipher and the Vigenere cipher is the way they handle the shift. In the Caesar cipher, the shift is fixed and applies to every letter in the plaintext. In the Vigenere cipher, the shift varies for each letter, depending on the corresponding letter in the keyword. This makes the Vigenere cipher more secure than the Caesar cipher.

```
#Key Generation
def key_generation(key):
```

```
    key_len = len(key)
    key_stream = [0]*key_len
    key = key.lower()
    for i in range(key_len):
        uniCode=ord(key[i])
        value = uniCode - 97
        key_stream[i] = value
    return key_stream
```

```
#Encryption Function
def encryption(plaintext, key_stream):
    text = plaintext.lower()
    key_size = len(key_stream)
    ciphertext = ""
    j = 0
    for char in text:
        unicode = ord(char)
        if unicode>=97 and unicode<=122:
            #Storing the key for current plaintext character
            key = key_stream[j]
            if j==(key_size-1):
                j = 0
            else:
                j = j+1
            #Calculating the ciphertext charater
            value = unicode - 97
            valueAfterShifting = (value + key) % 26
            unicode = valueAfterShifting + 97
            new_char = chr(unicode)
            ciphertext = ciphertext + new_char
            ciphertext=ciphertext.upper()
        else:
            ciphertext = ciphertext + char
    return ciphertext
```

```
#Decryption Function
def decryption(ciphertext, key_stream):
    text = ciphertext
    key_size = len(key_stream)
    plaintext = ""
    j = 0
    for char in text:
        unicode = ord(char)
        if unicode>=65 and unicode<=90:
            #Storing the key for current ciphertext character
            key = key_stream[j]
            if j==(key_size-1):
                j = 0
            else:
                j = j+1
            #Calculating the plaintext charater
            value = unicode - 65
            valueAfterShifting = (value - key) % 26
            unicode = valueAfterShifting + 65+32
            new_char = chr(unicode)
            plaintext = plaintext + new_char
        else:
            plaintext = plaintext + char
    return plaintext
```

```
#Input Section
plaintext = input("Enter the plaintext: ")
key = input("Enter the key: ")

#Function Calling
key_stream = key_generation(key)
ciphertext = encryption(plaintext, key_stream)
decrypted_text = decryption(ciphertext, key_stream)

#Output Section
print("Given Plaintext: ", plaintext)
print("Entered key: ", key)
print("Key Stream : ", key_stream)
```

```
print("Ciphertext: ", ciphertext)
print("Decrypted text: ", decrypted_text)
```

```
Enter the plaintext:  csejnu
Enter the key:  cse
Given Plaintext:  csejnu
Entered key:  cse
Key Stream :  [2, 18, 4]
Ciphertext:  EKILFY
Decrypted text:  csejnu
```

3.Multiplicative Cipher

C = (P * K) Mod 26 Here, C = Cipher text, P = Plain text, K = Key, Mod = Modulus

Decryption= (C * Multiplication inverse of the key) Mod 26. Here, c = ciphertext, Mod = Modulo

```
#Encryption Function
def encryption(plaintext, key):
    text = plaintext.lower()

    ciphertext = ""
    for char in text:
        unicode = ord(char)
        #Range of lowercase letter is 97 to 122
        if unicode>= 97 and unicode<=122:
            value = unicode - 97
            valueAfterOperation = (value * key) % 26
            unicode = valueAfterOperation + 97
            new_char = chr(unicode)
            ciphertext = ciphertext + new_char
            ciphertext=ciphertext.upper()
        else:
            ciphertext = ciphertext + char
    return ciphertext
```

```
#Decryption Function
def decryption(ciphertext, key):
    text = ciphertext
    #finding multiplicative inverse of the key
    key_inv = pow(key, -1, 26)
    plaintext = ""
    for char in text:
        unicode = ord(char)
        #Range of uppercase letter is 65 to 90
        if  unicode>= 65 and  unicode<=90:
            value =  unicode - 65
            valueAfterOperation = (value * key_inv) % 26
            unicode = valueAfterOperation + 65+32
            new_char = chr(unicode)
            plaintext = plaintext + new_char
        else:
            plaintext = plaintext + char
    return plaintext
```

```
#Input Section
plaintext = input("Enter the plaintext: ")
key = int(input("Enter the key: "))

#Function Calling
ciphertext = encryption(plaintext, key)
decrypted_text = decryption(ciphertext, key)

#Output Section
print("Entered plaintext : ", plaintext)
print("Entered key : ",key)
print("Cipher text: ", ciphertext)
print("Decrypted plaintext: ", decrypted_text)
```

```
Enter the plaintext:  csejnu
Enter the key:  3
Entered plaintext :  csejnu
```

```
Entered key :  3
Cipher text:  GCMBNI
Decrypted plaintext:  csejnu
```

4.Affine Cipher

Encryption of Affine Cipher:

E(x) = (Ax + B) mod M

Decryption of Affine Cipher:

D(x)= C(x − B) mod M

```
#Encryption Function
def encryption(plaintext, key1, key2):
    text = plaintext.lower()
    ciphertext = ""
    for char in text:
        unicode = ord(char)
        #Range for lowercase letter is 97 to 122
        if unicode>=97 and unicode<=122:
            value = unicode - 97
            valueAfterOperation = ((value * key1) + key2) % 26
            unicode = valueAfterOperation + 97
            new_char = chr(unicode)
            ciphertext = ciphertext + new_char
            ciphertext=ciphertext.upper()
        else :
            ciphertext = ciphertext + char
    return ciphertext
```

```
#Decrption Function
def decryption(ciphertext, key1, key2):
    text = ciphertext
    #finding the inverse of key1 mod 26
    key1_inv = pow(key1, -1, 26)
    plaintext = ""
    for char in text:
        unicode = ord(char)
        #Range for uppercase letter is 65 to 90
        if unicode>=65 and unicode<=90:
            value = unicode - 65
            valueAfterOperation = ((value - key2) * key1_inv) % 26
            unicode = valueAfterOperation + 65+32
            new_char = chr(unicode)
            plaintext = plaintext + new_char
        else:
            plaintext = plaintext + char
    return plaintext
```

```
#Input section
plaintext = input("Enter the plaintext: ")
key1 = int(input("Enter the first key: "))
key2 = int(input("Enter the second key : "))

#Function calling
ciphertext = encryption(plaintext, key1, key2)
decrypted_text = decryption(ciphertext, key1, key2)

#Output Section
print("Entered plaintext: ", plaintext)
print("Entered keys are: \nkey1 = ", key1, "\nkey2 = ", key2)
print("Ciphertext : ", ciphertext)
print("Decrypted text : ", decrypted_text)
```

```
Enter the plaintext:  csejnu
Enter the first key:  3
```

```
Enter the second key :  5
Entered plaintext:  csejnu
Entered keys are:
key1 =  3
key2 =  5
Ciphertext :  LHRGSN
Decrypted text :  csejnu
```

5.DES Cipher

```
#pip install pycryptodome
```

```
import base64
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes

#Input plaintext
plaintext = input("Enter the plaintext: ")
#Padding the plaintext
while len(plaintext) % 8 != 0:
    plaintext = plaintext + " "
#Create a random key
key = get_random_bytes(8)

#Create model of the cipher
des = DES.new(key, DES.MODE_ECB)

#Encryption Part
ciphertext = des.encrypt(plaintext.encode('utf-8'))
print("Ciphertext: ", base64.b64encode(ciphertext))
#Decryptiom Part
decryptedtext = des.decrypt(ciphertext)
print("Decrypted text : ", decryptedtext.decode())
```

```
Enter the plaintext:  This is a secret message
Ciphertext:  b'Mnh6wHNlchjzO588g2HEAXEd7s7rCyPN'
Decrypted text :  This is a secret message
```

6.AES Cipher

```
import base64
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

plaintext = b"This is a secret message"
key = get_random_bytes(16)

cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(plaintext)
print("Ciphertext : ", base64.b64encode(ciphertext))
print("Tag : ", tag)

decrypt_cipher = AES.new(key, AES.MODE_EAX, nonce=cipher.nonce)
decrypted_text =decrypt_cipher.decrypt_and_verify(ciphertext, tag)
print("Decrypted text: ", decrypted_text.decode())
```

```
Ciphertext :  b'SxLccaupW8yBZcOS5slJ8lDrqY6H3nTD'
Tag :  b'e\xd4<\xfc\xd7\xd2\xaf\xc6\xbf\x1cl\xb7\x17\xe17U'
Decrypted text:  This is a secret message
```

7.RSA

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
```

```
#Function for generating public and private key
def generate_key_pair():
    key = RSA.generate(2048)
    public_key = key.publickey().export_key()
    private_key = key.export_key()
    return public_key, private_key
```

```
#Encryption Function
def encrypt(message, public_key):
    cipher = PKCS1_OAEP.new(RSA.import_key(public_key))
    encrypted_message = cipher.encrypt(message)
    return encrypted_message
```

```
#Decryption Function
def decrypt(encrypted_message, private_key):
    cipher = PKCS1_OAEP.new(RSA.import_key(private_key))
    decrypted_message = cipher.decrypt(encrypted_message)
    return decrypted_message
```

```
# Example usage
plaintext = b"This is a secret message from TAJ"
print("Plaintext:", plaintext)
print("----output----")
# Generate key pair
public_key, private_key = generate_key_pair()
# Encrypt the message
encrypted_message = encrypt(plaintext, public_key)
print("Encrypted message:", encrypted_message.hex())
# Decrypt the message
decrypted_message = decrypt(encrypted_message, private_key)
print("Decrypted message:", decrypted_message.decode())
```

```
Plaintext: b'This is a secret message from TAJ'
----output----
Encrypted message: 389031ffec25e0b9e50edaf31b60aa9f3a1eb2e517674e2e5a950d17090ac747f907331ca050f197cef2e407515138efcb50b556fd94c25d33107
Decrypted message: This is a secret message from TAJ
```

8.Keyless Transposition Cipher

```
import numpy as np

#Input section
plaintext = input("Enter the plantext: ")
plaintext = plaintext.replace(" ","")
plaintext = plaintext.upper()
column = int(input("Enter the number of columns: "))

#Add extra characters if necessary
if (len(plaintext) % column) != 0:
    extra = column - (len(plaintext) % column)
    for i in range(extra):
        plaintext = plaintext + "X"

#Encryption Section
row = len(plaintext)//column
table = np.array([["Z"]*column]*row)
#array_2d = np.zeros((3, 4))
index = 0
for i in range(row):
    for j in range(column):
        table[i][j] = plaintext[index]
        index = index + 1

ciphertext = ""
transpose_table = np.transpose(table)
for i in range(column):
    for j in range(row):
        ciphertext = ciphertext + transpose_table[i][j]

print("Ciphertext: ", ciphertext)
```

```
#Decryption Section
cipher_table = np.array([["Z"]*row]*column)
index = 0
for i in range(column):
    for j in range(row):
        cipher_table[i][j] = ciphertext[index]
        index = index + 1

decrypted_text = ""
transpose_cipher_table = np.transpose(cipher_table)
for i in range (row):
    for j in range(column):
        decrypted_text = decrypted_text + transpose_cipher_table[i][j]
decrypted_text=decrypted_text.lower()

print(decrypted_text)
```

```
Enter the plantext:  farhana akter suci
Enter the number of columns:  4
Ciphertext:  FAKSANTURAECHARI
farhanaaktersuci
```

9.ElGamal Cipher

primitive_root: This function finds a primitive root modulo a prime number. A primitive root g of a prime number p is an integer such that the powers of g generate all the integers from 1 to p-1 when taken modulo p.

```
# Sympy is a Python library for symbolic mathematics.
from sympy import primitive_root,randprime
import random
```

```
# The number for which we want to find the primitive root
prime = randprime(124,10**3)
root = primitive_root(prime)
```

```
d=random.randint(1,(prime-2)) # It is private key.
```

```
e=(pow(root,d)%prime)  # It is public key.
r=random.randint(1,10)  # Select a random integer.
```

```
#Define the plaintext.
plaintext = "This is a secret message"
```

```
# Encryption Algorithm.
ciphertext=[]
for char in plaintext:
  ciphertext1=(pow(root,r)%prime)
  ciphertext2=((ord(char)*pow(e,r))%prime)
  ciphertext.append((ciphertext1,ciphertext2))
print(ciphertext)
```

```
[(337, 85), (337, 322), (337, 533), (337, 367), (337, 493), (337, 533), (337, 367), (337, 493), (337, 552), (337, 493), (337, 367), (337
```

```
#Decryption Algorithm
plaintext=""
for pair in ciphertext:
  ciphertext1,ciphertext2=pair
  value=pow(ciphertext1,d)
  multinv = pow(value,-1,prime)
  decrypt_char = (ciphertext2*multinv) % prime
  plaintext += chr(decrypt_char)
print(plaintext)
```

```
This is a secret message
```

10.Hill Cipher

```
import numpy as np
#Define the key matrix
key_matrix = np.array(
    [
        [9, 7, 11, 13],
        [4, 7, 5, 6],
        [2, 21, 14, 9],
        [3, 23, 21, 8]
    ]
)
#Finding inverse of the key matrix
det = int(np.round(np.linalg.det(key_matrix)))
det_inv = pow(det, -1, 26)
inv_key_matrix = det_inv * (np.round(det*np.linalg.inv(key_matrix)).astype(int)) %26
```

```
#Input section
plaintext = input("Enter the plaintext: ")
text = plaintext.replace(" ","")
text_len = len(text)
key_len = len(key_matrix)
text = text.upper()

#Add extra character if required
if (text_len % key_len) !=0:
    extra = key_len - (text_len % key_len)
    for i in range (extra):
        text = text + "X"
print(text)

#Create plaintext array
text_len = len(text)
text_array = np.zeros(text_len, dtype=int)

i = 0
for char in text:
    value = ord(char) - 65
    text_array[i] = value
    i = i + 1

column = len(key_matrix)
row =  len(text_array)//column
#Reshaping the array
text_array = text_array.reshape(row, column)
print("Plaintext array of numerical values: \n",text_array)
```

```
Enter the plaintext:  csejnu
CSEJNUXX
Plaintext array of numerical values:
 [[ 2 18  4  9]
 [13 20 23 23]]
```

```
#Encryption Section
cipher_array = np.dot(text_array, key_matrix)%26
ciphertext = ""
for i in range(row):
    for j in range(column):
        char = chr(cipher_array[i][j]+97)
        ciphertext = ciphertext + char
print("Ciphertext: ",ciphertext)

#Decryption Section
decrypted_array = np.dot(cipher_array, inv_key_matrix)%26
decrypted_text = ""
for i in range(row):
    for j in range(column):
        char = chr(decrypted_array[i][j]+97)
        decrypted_text = decrypted_text + char
print("Decrypted plaintext: ",decrypted_text)
```

```
Ciphertext:  vptiavie
Decrypted plaintext:  csejnuxx
```

+ Code + Text

Start coding or generate with AI.