Department of Computer Science and Engineering

Jagannath University, Dhaka-1100

Course Title: Cryptography and Information Security Lab

Course Code: CSEL-4110

Submitted By:

Name: Shamim

ID: B190305006

Submitted To:

Name: DR. MOHAMMED NASIR UDDIN

Professor

Department of CSE Jagannath University

Table of Contents

Additive Cipher 3

Vigenere Cipher 5

Multiplicative Cipher 7

Affine Cipher 8

Playfair Cipher 9

Data Encryption Standard (DES) 13

Advanced Encryption Standard (AES) 14

RSA Encryption 15

ElGamal Encryption 16

Rabin Cryptosystem 17

Additive Cipher

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

#Decryption Function
def decryption(ciphertext, key):
   text = ciphertext.upper()
   #Range of uppercase letter is 65 to 90
   plaintext = ""
   for char in text:
      order = ord(char)
      if order >= 65 and order <= 90:
         order = order - 65
         order = (order - key) % 26
         order = order + 65
         new_char = chr(order)
         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:  hello
Enter the key:  7
Given plaintext:  hello
Entered key :  7
Ciphertext:  olssv
Decrypted plaintext:  HELLO
```

Vigenere 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):
    order = ord(key[i]) - 97
    key_stream[i] = order
  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:
    order = ord(char)
    if order>=97 and order<=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
      order = order - 97
      order = (order + key) % 26
      order = order + 97
      new_char = chr(order)
      ciphertext = ciphertext + new_char
    else:
      ciphertext = ciphertext + char
  return ciphertext

#Decryption Function
def decryption(ciphertext, key_stream):
  text = ciphertext.upper()
  key_size = len(key_stream)
  plaintext = ""
  j = 0
  for char in text:
    order = ord(char)
    if order>=65 and order<=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
            order = order - 65
            order = (order - key) % 26
            order = order + 65
            new_char = chr(order)
            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:  she is listening
Enter the key:  PASCAL
Given Plaintext:  she is listening
Entered key:  PASCAL
Key Stream :  [15, 0, 18, 2, 0, 11]
Ciphertext:  hhw ks wxslgntcg
Decrypted text:  SHE IS LISTENING
```

Multiplicative Cipher

```
#Encryption Function
def encryption(plaintext, key):
   text = plaintext.lower()
   ciphertext = ""
   for char in text:
      order = ord(char)
      #Range of lowercase letter is 97 to 122
      if order>= 97 and order<=122:
         order = order - 97
         order = (order * key) % 26
         order = order + 97
         new_char = chr(order)
         ciphertext = ciphertext + new_char
      else:
         ciphertext = ciphertext + char
   return ciphertext
#Decryption Function
def decryption(ciphertext, key):
   text = ciphertext.upper()
   #finding multiplicative inverse of the key
   key_inv = pow(key, -1, 26)
   plaintext = ""
   for char in text:
      order = ord(char)
      #Range of uppercase letter is 65 to 90
      if order>= 65 and order<=90:
         order = order - 65
         order = (order * key_inv) % 26
         order = order + 65
         new_char = chr(order)
         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 : ")
print("Cipher text: ", ciphertext)
print("Decrypted plaintext: ", decrypted_text)
```

```
Enter the plaintext:  hello
Enter the key:  7
Entered plaintext :  hello
Entered key :
Cipher text:  xczzu
Decrypted plaintext:  HELLO
```

Affine Cipher

```
#Encryption Function
def encryption(plaintext, key1, key2):
    text = plaintext.lower()
    ciphertext = ""
    for char in text:
        order = ord(char)
        #Range for lowercase letter is 97 to 122
        if order>=97 and order<=122:
            order = order - 97
            order = ((order * key1) + key2) % 26
            order = order + 97
            new_char = chr(order)
            ciphertext = ciphertext + new_char
        else :
            ciphertext = ciphertext + char
    return ciphertext
#Decrption Function
def decryption(ciphertext, key1, key2):
    text = ciphertext.upper()
    #finding the inverse of key1 mod 26
    key1_inv = pow(key1, -1, 26)
    plaintext = ""
    for char in text:
        order = ord(char)
        #Range for uppercase letter is 65 to 90
        if order>=65 and order<=90:
            order = order - 65
            order = ((order - key2) * key1_inv) % 26
            order = order + 65
            new_char = chr(order)
            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:  hello
Enter the first key:  7
Enter the second key :  2
Entered plaintext:  hello
Entered keys are:
key1 =  7
key2 =  2
Ciphertext :  zebbw
Decrypted text :  HELLO
```

Playfair Cipher

```
import numpy as np
#Declaring the 5*5 key matrix
key_matrix = np.array(
  [
    ['L', 'G', 'D', 'B', 'A'],
    ['Q', 'M', 'H', 'E', 'C'],
    ['U', 'R', 'N', 'J', 'F'],
    ['X', 'V', 'S', 'O', 'K'],
    ['Z', 'Y', 'W', 'T', 'P']
  ]
)
#Creating transpose matrix of the key matrix
transpose_key_matrix = np.transpose(key_matrix)
#Input Section
plaintext = input("Enter the plaintext: ")
print("Given Plaintext: ", plaintext)
#Removing all the whitespaces from the plaintext
text = plaintext.replace(" ","")
text_len = len(text)
text = text.upper()

#Replace all "I" in the plaintext to "j"
text = text.replace("I", "J")
```

```
#Make pair of two(different) characters from plaintext
plaintextpair = []
i = 0
while i < text_len:
   char1 = text[i]
   char2 = ""
   #If the letter is the last character of the plaintext add a vogus character "X"
   if (i+1) == len(text):
      char2 = "X"
   #Else add the next character
   else:
      char2 = text[i+1]
   #If the two characters are different insert them in the pair
   if char1 != char2:
      plaintextpair.append(char1+char2)
      i = i+2
   #else add "X" as the second character
   else:
      plaintextpair.append(char1+"X")
      i = i + 1
print("Pairs of plaintext :", plaintextpair)
#Encryption Function
ciphertext = ""
ciphertextpair = []
for pair in plaintextpair:
   apply_rule = True
   #Rule 1: If the two characters are in the same row replace them with their right charater
   if apply_rule :
      for row in range(5):
         if pair[0] in key_matrix[row] and pair[1] in key_matrix[row]:
            for i in range(5):
               if key_matrix[row][i]==pair[0]:
                  char1 = key_matrix[row][(i+1)%5]
               elif key_matrix[row][i]==pair[1]:
                  char2 = key_matrix[row][(i+1)%5]
            apply_rule = False
            ciphertextpair.append(char1+char2)
            ciphertext = ciphertext + char1 + char2

   #Rule 2: If the two characters are in the same column replace them with their below character
   #for this we will use transpose matrix
   if apply_rule :
      for column in range(5):
         if pair[0] in transpose_key_matrix[column] and pair[1] in transpose_key_matrix[column]:
            for i in range(5):
               if transpose_key_matrix[column][i]==pair[0]:
                  char1 = transpose_key_matrix[column][(i+1)%5]
```

```
                elif transpose_key_matrix[column][i]==pair[1]:
                    char2 = transpose_key_matrix[column][(i+1)%5]
            apply_rule = False
            ciphertextpair.append(char1+char2)
            ciphertext = ciphertext + char1 + char2

    #Rule 3: If the two letters are not in the same row or column,replace them with letter
    # that is in its own row but in the same column as the other letter.
    if apply_rule :
        for row in range (5):
            for column in range (5):
                if key_matrix[row][column] == pair[0]:
                    x0 = row
                    y0 = column
                elif key_matrix[row][column] == pair[1]:
                    x1 = row
                    y1 = column
        char1 = key_matrix[x0][y1]
        char2 = key_matrix[x1][y0]
        ciphertextpair.append(char1+char2)
        ciphertext = ciphertext + char1 + char2
print("Ciphertext: ", ciphertext)
#Decryption Function
decryptedtext = ""
for pair in ciphertextpair:
    apply_rule = True
    #Rule 1: If the two characters are in the same row replace them with their left charater
    if apply_rule :
        for row in range(5):
            if pair[0] in key_matrix[row] and pair[1] in key_matrix[row]:
                for i in range(5):
                    if key_matrix[row][i]==pair[0]:
                        char1 = key_matrix[row][(i-1)%5]
                    elif key_matrix[row][i]==pair[1]:
                        char2 = key_matrix[row][(i-1)%5]
                apply_rule = False
                decryptedtext = decryptedtext + char1 + char2

    #Rule 2: If the two characters are in the same column replace them with their upper character
    #for this we will use transpose matrix
    if apply_rule :
        for column in range(5):
            if pair[0] in transpose_key_matrix[column] and pair[1] in transpose_key_matrix[column]:
                for i in range(5):
                    if transpose_key_matrix[column][i]==pair[0]:
                        char1 = transpose_key_matrix[column][(i-1)%5]
                    elif transpose_key_matrix[column][i]==pair[1]:
                        char2 = transpose_key_matrix[column][(i-1)%5]
```

```
            apply_rule = False
            decryptedtext = decryptedtext + char1 + char2

   #Rule 3: If the two letters are not in the same row or column,replace them with letter
   # that is in its own row but in the same column as the other letter.
   if apply_rule :
      for row in range (5):
         for column in range (5):
            if key_matrix[row][column] == pair[0]:
               x0 = row
               y0 = column
            elif key_matrix[row][column] == pair[1]:
               x1 = row
               y1 = column
      char1 = key_matrix[x0][y1]
      char2 = key_matrix[x1][y0]
      decryptedtext = decryptedtext + char1 + char2
print("Decrypted text: ", decryptedtext.lower())
```

```
Enter the plaintext:  csejnu
Given Plaintext:  csejnu
```

```
Pairs of plaintext : ['CS', 'EJ', 'NU']
```

```
Ciphertext:  HKJOJR
```

```
Decrypted text:  csejnu
```

Data Encryption Standard (DES)

```
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: csejnu
Generated Key:  b'VMvQLwAGKMk='
Ciphertext:  b'xAhfmJaC6DY='
Decrypted text:  csejnu
```

Advanced Encryption Standard (AES)

```
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'xvIpA7YJArtzsmGdd+AWPt+gQSPY3mGv'
Tag :  b'\xa0\x84\xe8y\xd0\xa3\x17\x11\x99\xa8\xd6r\xc6\xc0\x1e]'
Decrypted text:  This is a secret message
```

RSA Encryption

```python
#A Python Code for Encryption Using RSA Algorithm

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: 7b949749b675dc04184334686aa5795706d9b98b8346cf75766a13113310e9dbf2df72d0756c847e19349e39a158009c046c69
3b9a05a6480edf7d0dd38df05c1253c6962304a29386e1e790cca88f55e26a6c0f85d58b2c4695214953cad4d45e7d719e543fbd6f521ab88767de012
67afa63b20b2fac0568680d3c5f15249349f5c5af586a1dfd6a1ffb3ef9372521db2e07a6fee0c0ef8c0f5641ff08e9d149af009ecca5863f9124b15c
9af4449d9d0aa0f40f4b83b3f50de6753ce8daa35a03eb26af72cb72697ddc04cd4ce337fb6627c7a51df7a85048d53a21c459561b978d742c1f672c1
bf92ea5d284e721ada28c66596838e087f5cacbe535a340
Decrypted message: This is a secret message from TAJ
```

ElGamal Encryption

```
# Sympy is a Python library for symbolic mathematics.
from sympy import primitive_root,randprime
import random
# The number for which you 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)
#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
```

```
[(278, 25), (278, 102), (278, 311), (278, 163), (278, 347), (278, 311), (278, 163), (278, 347), (278, 131), (278, 347), (278,
163), (278, 221), (278, 176), (278, 327), (278, 221), (278, 372), (278, 347), (278, 28), (278, 221), (278, 163), (278, 163),
(278, 131), (278, 266), (278, 221)]
```

```
This is a secret message
```

Rabin Cryptosystem

```
# Helper function: Extended Euclidean Algorithm to find the modular inverse
def extended_gcd(a, b):
    if b == 0:
        return a, 1, 0
    gcd, x1, y1 = extended_gcd(b, a % b)
    x = y1
    y = x1 - (a // b) * y1
    return gcd, x, y

def mod_inverse(a, m):
    gcd, x, _ = extended_gcd(a, m)
    if gcd != 1:
        raise ValueError("Modular inverse does not exist")
    else:
        return x % m

# Helper function: Modular Exponentiation
def mod_exp(base, exp, mod):
    result = 1
    base = base % mod
```

```
    while exp > 0:
        if (exp % 2) == 1:  # If exp is odd, multiply base with result
            result = (result * base) % mod
        exp = exp >> 1  # Divide the exponent by 2
        base = (base * base) % mod  # Square the base
    return result

# Step 1: Key Generation
def generate_keys():
    # Prime numbers p and q (small values for simplicity, use larger primes in real-world use)
    p = 7
    q = 11
    n = p * q
    return p, q, n

# Step 2: Encryption (Public key is n, plaintext is m)
def encrypt(m, n):
    return mod_exp(m, 2, n)

# Step 3: Decryption (Private keys are p, q)
def decrypt(c, p, q, n):
    # Compute square roots modulo p and q
    mp = mod_exp(c, (p + 1) // 4, p)
    mq = mod_exp(c, (q + 1) // 4, q)

    # Use Chinese Remainder Theorem to get four possible plaintexts
    inv_q = mod_inverse(q, p)
    inv_p = mod_inverse(p, q)

    x1 = (mp * q * inv_q + mq * p * inv_p) % n
    x2 = (mp * q * inv_q - mq * p * inv_p) % n
```

```python
    # Return the four possible solutions
    return x1, n - x1, x2, n - x2

# Example Usage
if __name__ == "__main__":
    # Generate keys
    p, q, n = generate_keys()
    print(f"Public key (n): {n}, Private keys (p, q): ({p}, {q})")

    # Sample message (must be smaller than n)
    m = 5
    print(f"Original message: {m}")

    # Encrypt the message
    c = encrypt(m, n)
    print(f"Encrypted message (ciphertext): {c}")

    # Decrypt the ciphertext
    possible_messages = decrypt(c, p, q, n)
    print(f"Decrypted possible messages: {possible_messages}")
```

```
Public key (n): 77, Private keys (p, q): (7, 11)
Original message: 5
Encrypted message (ciphertext): 25
Decrypted possible messages: (16, 61, 72, 5)
```