VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 1 : WRITE A CODE OF THE FOLLOWING IN YOUR SUITABLE PROGRAMMING LANGUAGE(PYTHON/C/C++/JAVA)

i. CAESER CIPHER
ii. PLAYFAIR CIPHER

NAME : OM SUBRATO DEY

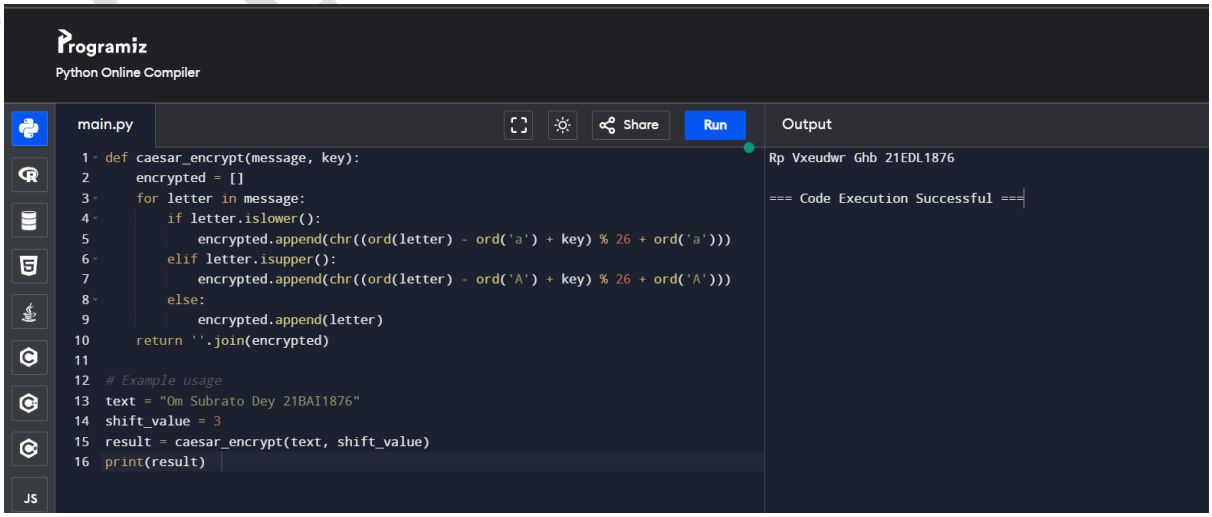
REGISTER NUMBER : 21BAI1876

i. CAESER CIPHER:

CODE:

```
def caesar_encrypt(message, key):
    encrypted = []
    for letter in message:
        if letter.islower():
            encrypted.append(chr((ord(letter) - ord('a') + key) % 26 + ord('a')))
        elif letter.isupper():
            encrypted.append(chr((ord(letter) - ord('A') + key) % 26 + ord('A')))
        else:
            encrypted.append(letter)
    return ''.join(encrypted)

text = "Om Subrato Dey 21BAI1876"
shift_value = 3
result = caesar_encrypt(text, shift_value)
print(result)
```

OUTPUT SCREENSHOT:



ii. PLAYFAIR CIPHER:

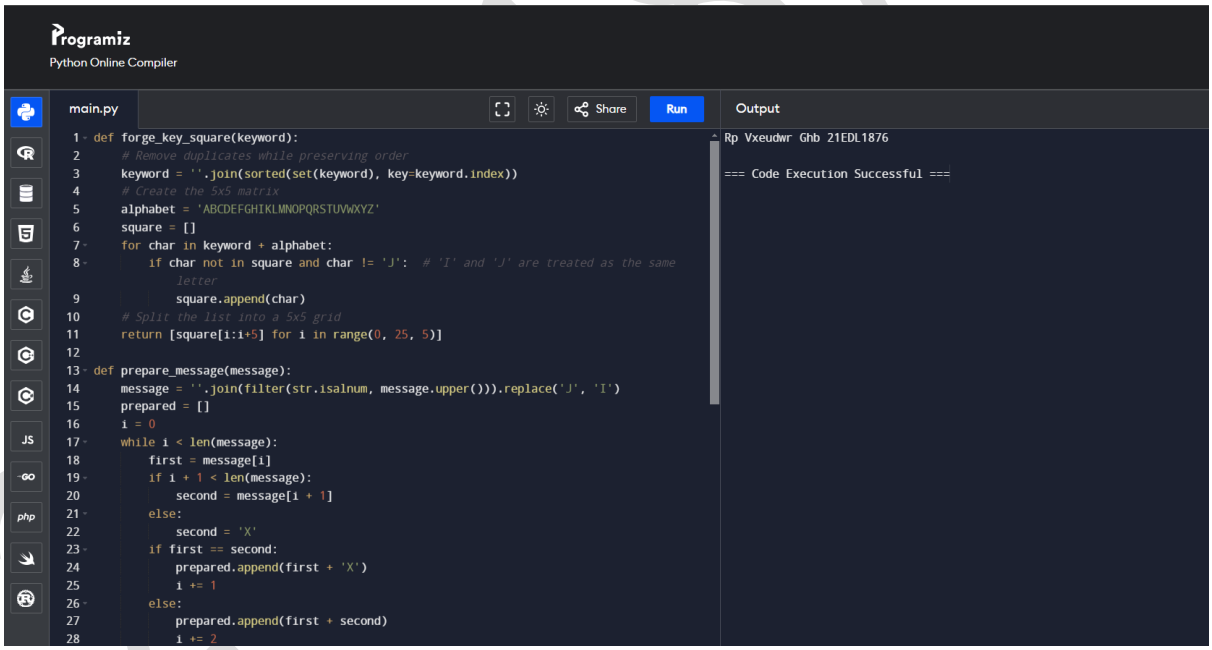
CODE:

```
def forge_key_square(keyword):
    # Remove duplicates while preserving order
    keyword = ''.join(sorted(set(keyword), 
key=keyword.index))
    # Create the 5x5 matrix
    alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    square = []
    for char in keyword + alphabet:
        if char not in square and char != 'J':  # 'I' and 
'J' are treated as the same letter
            square.append(char)
    # Split the list into a 5x5 grid
    return [square[i:i+5] for i in range(0, 25, 5)]


def prepare_message(message):
    message = ''.join(filter(str.isalnum, 
message.upper())).replace('J', 'I')
    prepared = []
    i = 0
    while i < len(message):
        first = message[i]
        if i + 1 < len(message):
            second = message[i + 1]
        else:
            second = 'X'
        if first == second:
            prepared.append(first + 'X')
            i += 1
```

```
            else:
                prepared.append(first + second)
                i += 2
        if len(prepared[-1]) == 1:
            prepared[-1] += 'X'
        return prepared

    def locate_position(square, char):
        for row in range(5):
            for col in range(5):
                if square[row][col] == char:
                    return row, col
        return None

    def encrypt_digraph(square, digraph):
        row1, col1 = locate_position(square, digraph[0])
        row2, col2 = locate_position(square, digraph[1])

        if row1 == row2:
            return square[row1][(col1 + 1) % 5] +
square[row2][(col2 + 1) % 5]
        elif col1 == col2:
            return square[(row1 + 1) % 5][col1] + square[(row2 +
1) % 5][col2]
        else:
            return square[row1][col2] + square[row2][col1]

    def playfair_encryption(message, keyword):
        key_square = forge_key_square(keyword)
        prepared_msg = prepare_message(message)
        encrypted_msg = ''
```

```
    for digraph in prepared_msg:
        encrypted_msg += encrypt_digraph(key_square,
digraph)
    return encrypted_msg

keyword = "KEYWORD"
message = "Om Subrato Dey 21BAI1876"
ciphertext = playfair_encryption(message, keyword)
print(ciphertext)
```

OUTPUT SCREENSHOT:



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 2 : WRITE A PROGRAM IN SUITABLE PROGRAMMING LANGUAGE FOR THE FOLLOWING:

- 1. VIGENERE CIPHER
- 2. RAIL FENCE CIPHER
- 3. EUCLIDEAN ALGORITHM

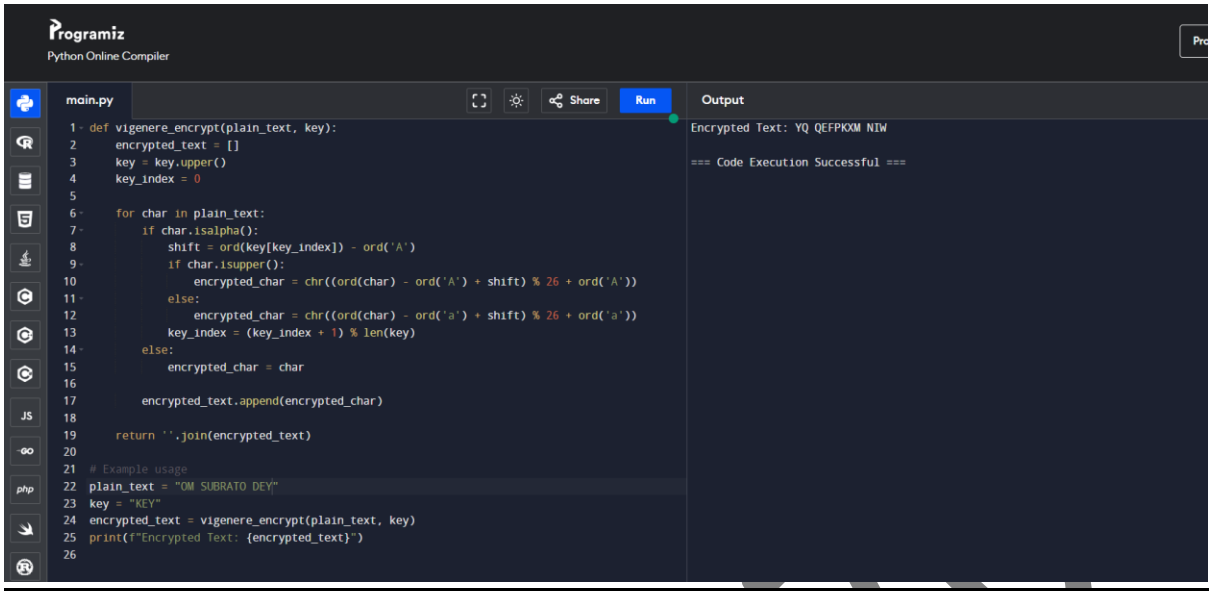
NAME : OM SUBRATO DEY

REGISTER NUMBER : 21BAI1876

1. VIGENERE CIPHER:

ENCRYPTION CODE AND OUTPUT:

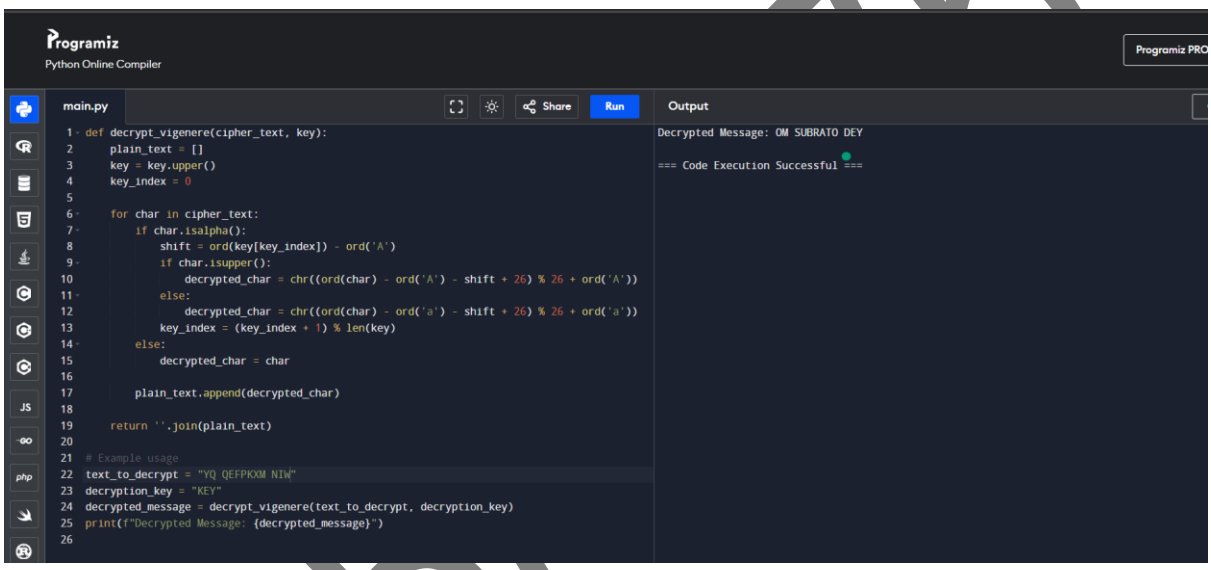
```
def vigenere_encrypt(plain_text, key):
    encrypted_text = []
    key = key.upper()
    key_index = 0

    for char in plain_text:
        if char.isalpha():
            shift = ord(key[key_index]) - ord('A')
            if char.isupper():
                encrypted_char = chr((ord(char) - ord('A') + shift)
% 26 + ord('A'))
            else:
                encrypted_char = chr((ord(char) - ord('a') + shift)
% 26 + ord('a'))
            key_index = (key_index + 1) % len(key)
        else:
            encrypted_char = char

        encrypted_text.append(encrypted_char)

    return ''.join(encrypted_text)

# Example usage
plain_text = "OM SUBRATO DEY"
key = "KEY"
encrypted_text = vigenere_encrypt(plain_text, key)
print(f"Encrypted Text: {encrypted_text}")
```



DECRYPTION CODE AND OUTPUT:

```
def decrypt_vigenere(cipher_text, key):
    plain_text = []
    key = key.upper()
    key_index = 0

    for char in cipher_text:
        if char.isalpha():
            shift = ord(key[key_index]) - ord('A')
            if char.isupper():
                decrypted_char = chr((ord(char) - ord('A') - shift + 
26) % 26 + ord('A'))
            else:
                decrypted_char = chr((ord(char) - ord('a') - shift + 
26) % 26 + ord('a'))
            key_index = (key_index + 1) % len(key)
        else:
            decrypted_char = char
```

```
        plain_text.append(decrypted_char)
    return ''.join(plain_text)


# Example usage
text_to_decrypt = "YQ QEFPKXM NIW"
decryption_key = "KEY"
decrypted_message = decrypt_vigenere(text_to_decrypt,
decryption_key)
print(f"Decrypted Message: {decrypted_message}")
```

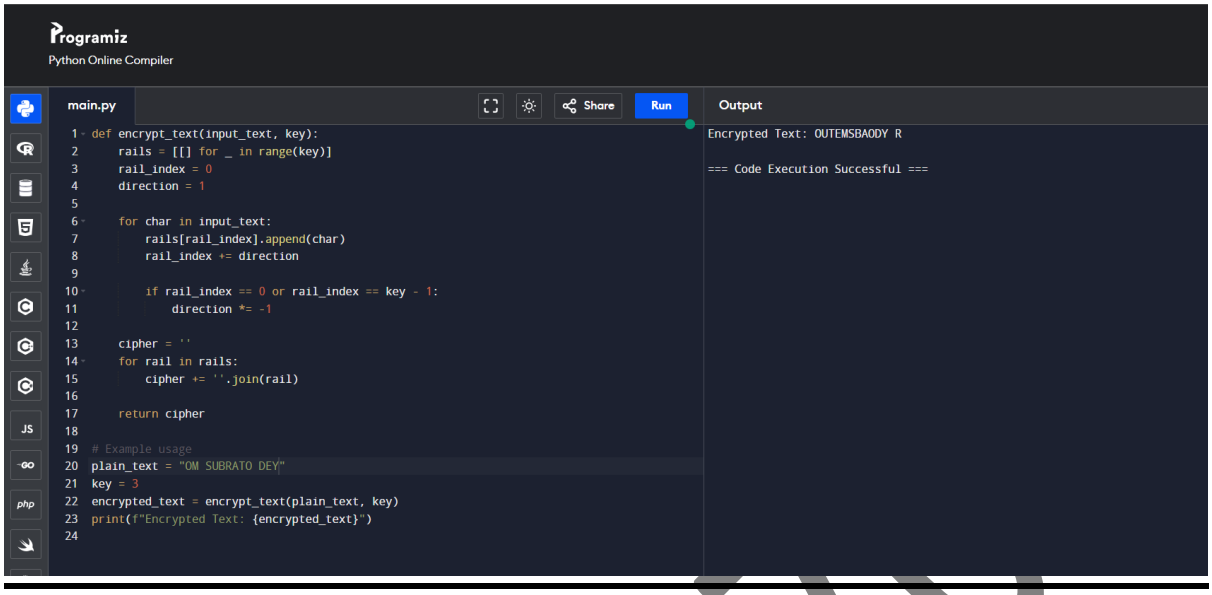


2. RAIL FENCE CIPHER:

CODE:

```
def encrypt_text(input_text, key):
    rails = [[] for _ in range(key)]
    rail_index = 0
    direction = 1

    for char in input_text:
        rails[rail_index].append(char)
        rail_index += direction

        if rail_index == 0 or rail_index == key - 1:
            direction *= -1

    cipher = ''
    for rail in rails:
        cipher += ''.join(rail)

    return cipher

# Example usage
plain_text = "OM SUBRATO DEY"
key = 3
encrypted_text = encrypt_text(plain_text, key)
print(f"Encrypted Text: {encrypted_text}")
```


OUTPUT:

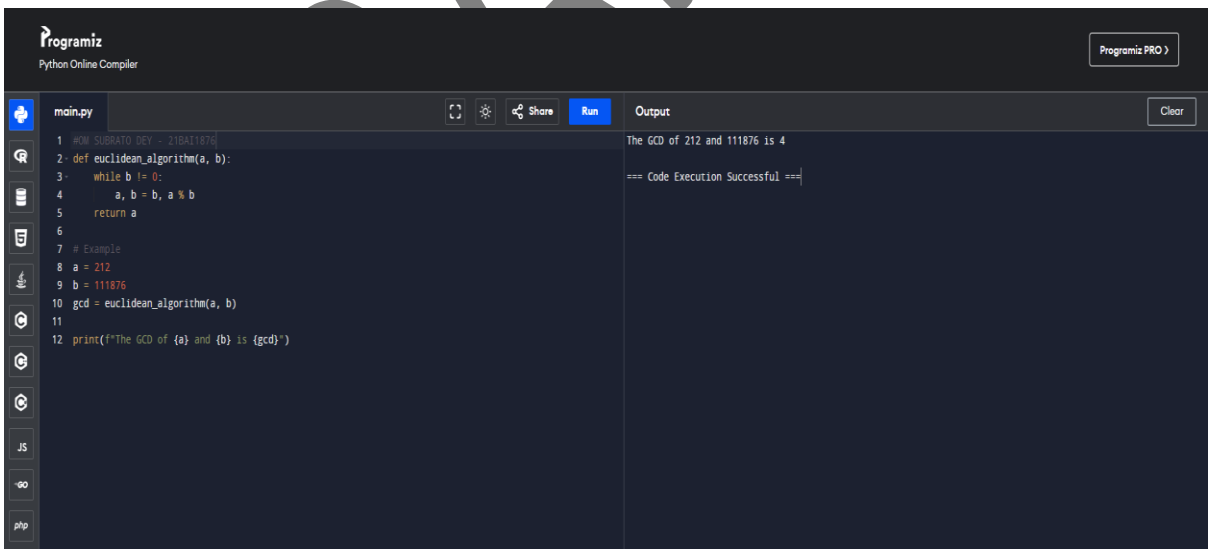


3. EUCLIDEAN TECHNIQUE:

CODE:

```
def euclidean_algorithm(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Example
a = 212
b = 111876
gcd = euclidean_algorithm(a, b)

print(f"The GCD of {a} and {b} is {gcd}")
```


OUTPUT:



CRYPTOGRAPHY AND NETWORK SECURITY

LAB 3 : WRITE THE CODE AND EXECUTE FOR THE FOLLOWING ALGORITHMS:

- 1. CHINESE REMAINDER THEOREM
- 2. EXTENDED EUCLIDEAN ALGORITHM

NAME : OM SUBRATO DEY

REGISTER NO.: 21BAI1876

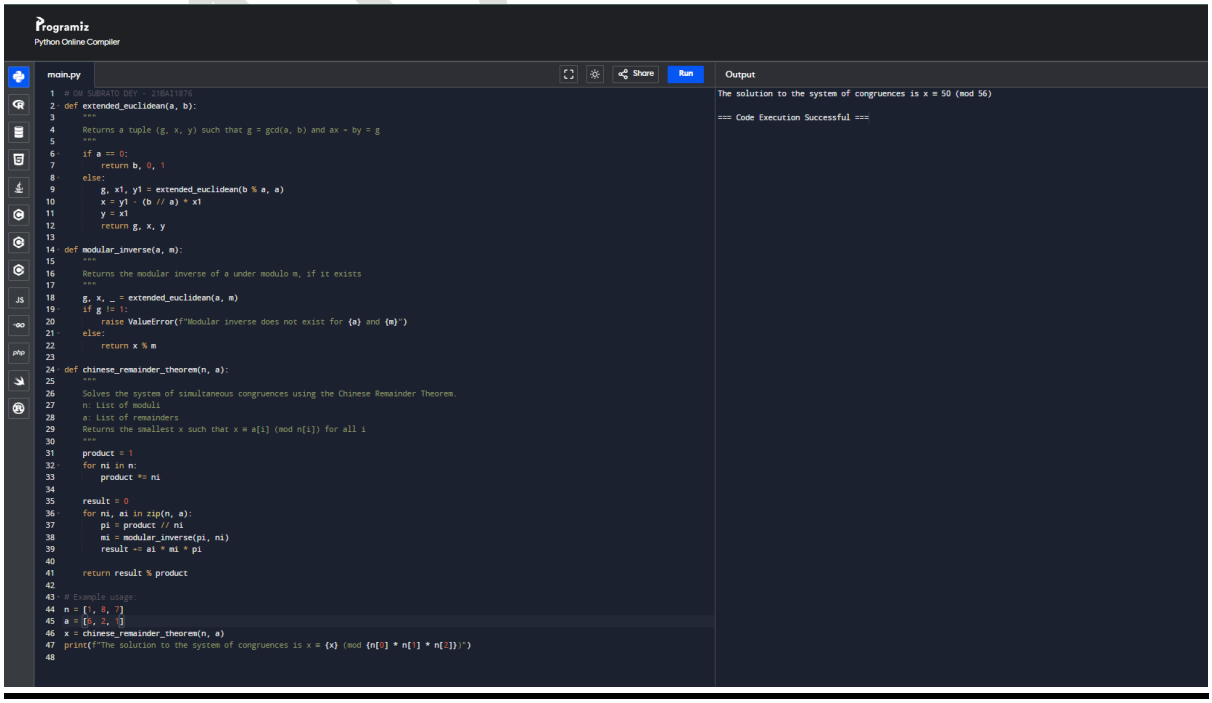
1. CHINESE REMAINDER THEOREM:

CODE:

```
def extended_euclidean(a, b):
    """
    Returns a tuple (g, x, y) such that g = gcd(a, b) and ax + by = g
    """
    if a == 0:
        return b, 0, 1
    else:
        g, x1, y1 = extended_euclidean(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return g, x, y


def modular_inverse(a, m):
    """
    Returns the modular inverse of a under modulo m, if it exists
    """
    g, x, _ = extended_euclidean(a, m)
    if g != 1:
        raise ValueError(f"Modular inverse does not exist for {a} and
{m}")
    else:
        return x % m


def chinese_remainder_theorem(n, a):
    """
    Solves the system of simultaneous congruences using the Chinese
Remainder Theorem.
    n: List of moduli
    a: List of remainders
```

```
    Returns the smallest x such that x ≡ a[i] (mod n[i]) for all i
    """
    product = 1
    for ni in n:
        product *= ni
    result = 0
    for ni, ai in zip(n, a):
        pi = product // ni
        mi = modular_inverse(pi, ni)
        result += ai * mi * pi

    return result % product
# Example usage:
n = [1, 8, 7]
a = [6, 2, 1]
x = chinese_remainder_theorem(n, a)
print(f"The solution to the system of congruences is x ≡ {x} (mod
{n[0] * n[1] * n[2]})")
```

OUTPUT:

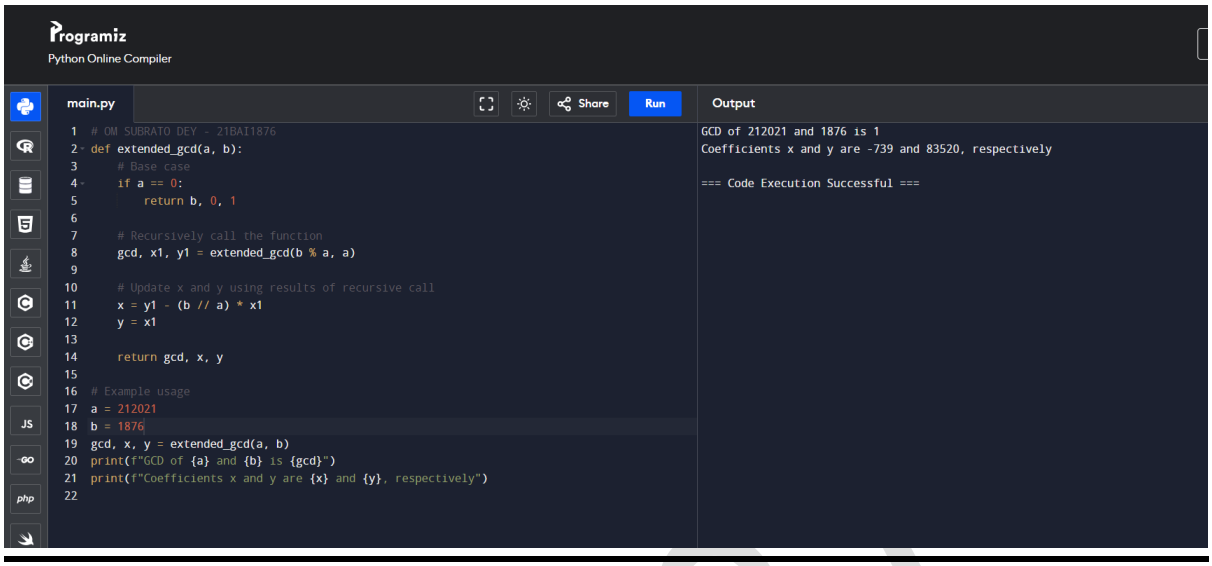


2. EXTENDED EUCLIDEAN ALGORITHM:

CODE:

```
# OM SUBRATO DEY - 21BAI1876
def extended_gcd(a, b):
    # Base case
    if a == 0:
        return b, 0, 1

    # Recursively call the function
    gcd, x1, y1 = extended_gcd(b % a, a)

    # Update x and y using results of recursive call
    x = y1 - (b // a) * x1
    y = x1

    return gcd, x, y

# Example usage
a = 212021
b = 1876
gcd, x, y = extended_gcd(a, b)
print(f"GCD of {a} and {b} is {gcd}")
print(f"Coefficients x and y are {x} and {y}, respectively")
```

OUTPUT:



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 4: WRITE A CODE FOR IMPLEMENTING SDES IN SUITABLE PROGRAMMING LANGUAGE.

NAME : OM SUBRATO DEY

REG NO.: 21BAI1876

CODE:

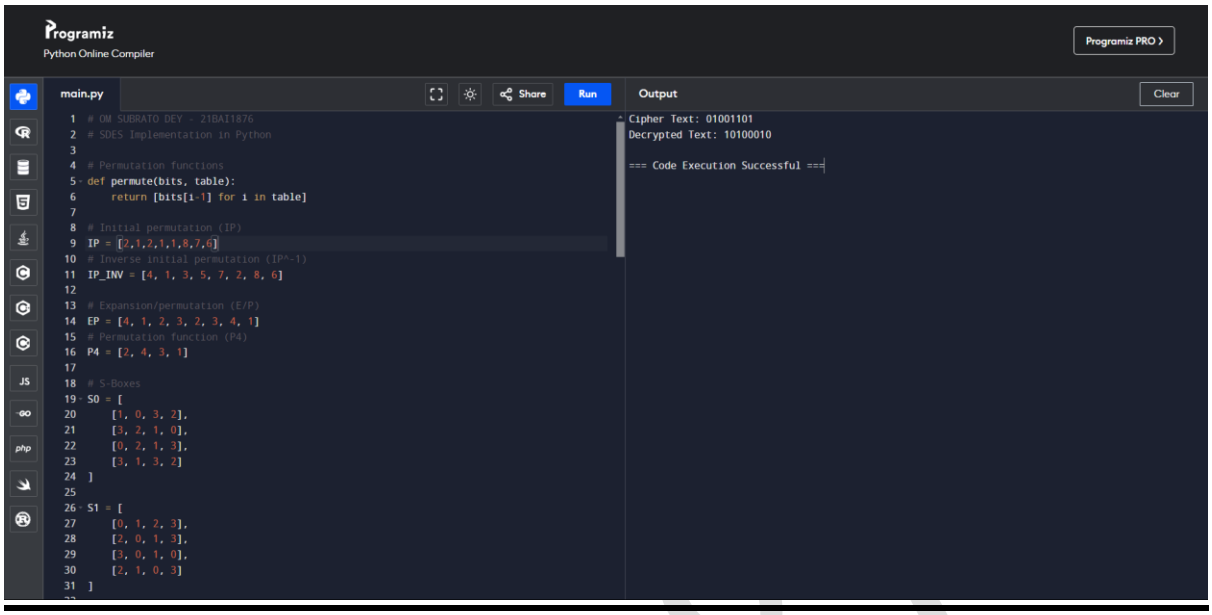
```
# OM SUBRATO DEY - 21BAI1876
# SDES Implementation in Python

# Permutation functions
def permute(bits, table):
    return [bits[i-1] for i in table]

# Initial permutation (IP)
IP = [2,1,2,1,1,8,7,6]
# Inverse initial permutation (IP^-1)
IP_INV = [4, 1, 3, 5, 7, 2, 8, 6]

# Expansion/permutation (E/P)
EP = [4, 1, 2, 3, 2, 3, 4, 1]
# Permutation function (P4)
P4 = [2, 4, 3, 1]

# S-Boxes
S0 = [
    [1, 0, 3, 2],
    [3, 2, 1, 0],
    [0, 2, 1, 3],
    [3, 1, 3, 2]
]

S1 = [
    [0, 1, 2, 3],
    [2, 0, 1, 3],
    [3, 0, 1, 0],
    [2, 1, 0, 3]
]
```

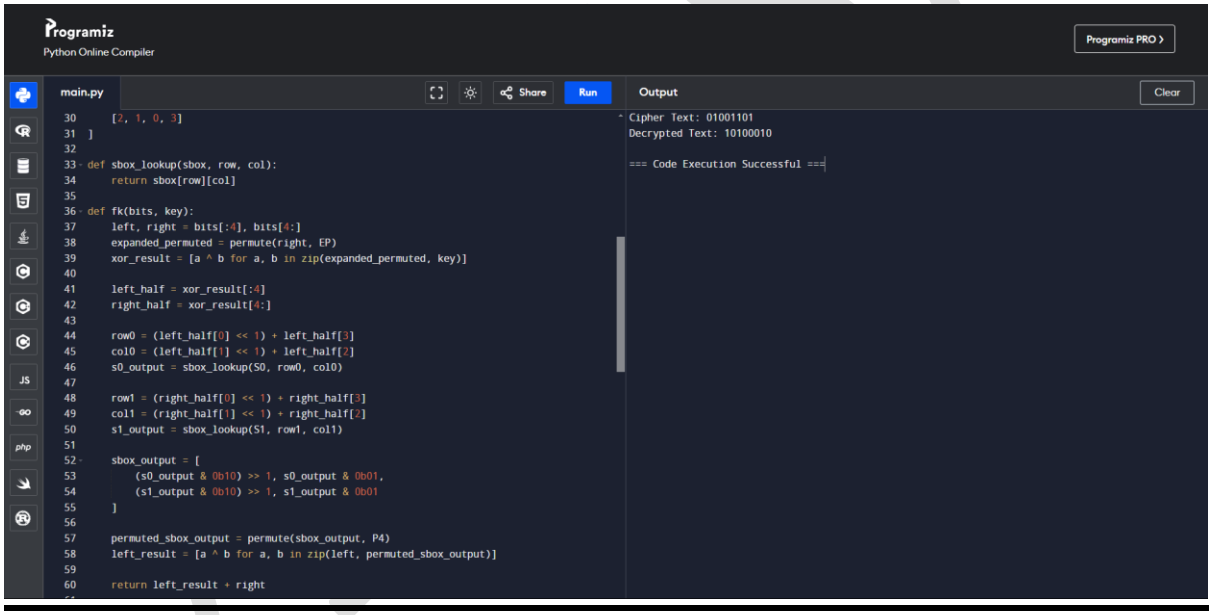
```
def sbox_lookup(sbox, row, col):
    return sbox[row][col]

def fk(bits, key):
    left, right = bits[:4], bits[4:]
    expanded_permuted = permute(right, EP)
    xor_result = [a ^ b for a, b in zip(expanded_permuted, key)]

    left_half = xor_result[:4]
    right_half = xor_result[4:]

    row0 = (left_half[0] << 1) + left_half[3]
    col0 = (left_half[1] << 1) + left_half[2]
    s0_output = sbox_lookup(S0, row0, col0)

    row1 = (right_half[0] << 1) + right_half[3]
    col1 = (right_half[1] << 1) + right_half[2]
    s1_output = sbox_lookup(S1, row1, col1)

    sbox_output = [
        (s0_output & 0b10) >> 1, s0_output & 0b01,
        (s1_output & 0b10) >> 1, s1_output & 0b01
    ]

    permuted_sbox_output = permute(sbox_output, P4)
    left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]

    return left_result + right

def switch(bits):
    return bits[4:] + bits[:4]

# Key generation
def generate_keys(key):
```

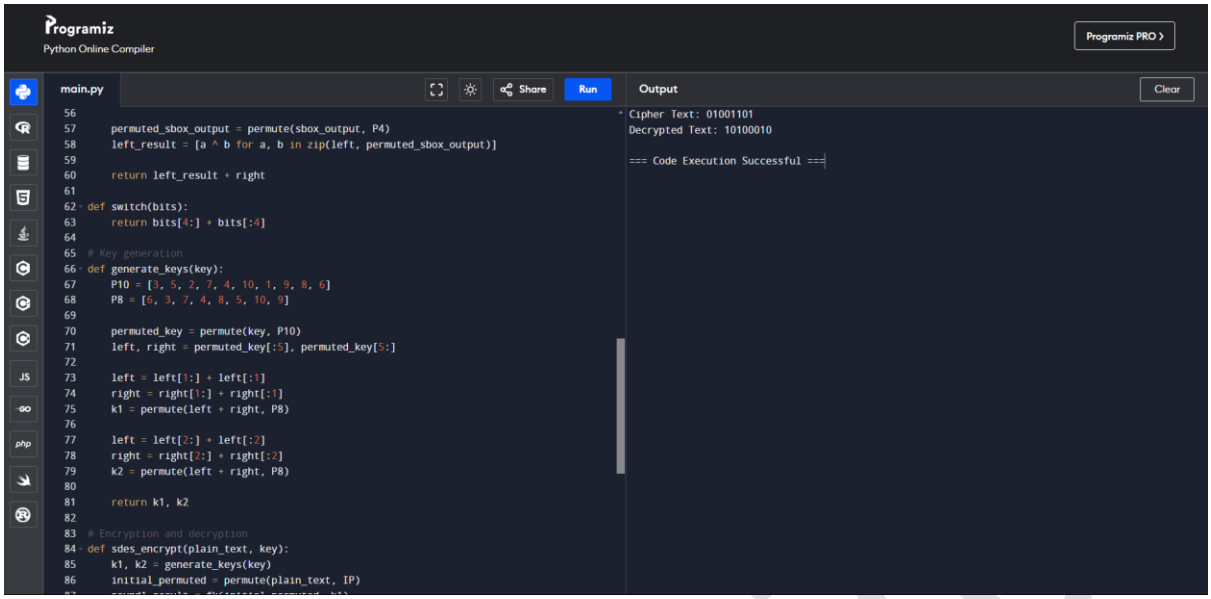
```
    P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    P8 = [6, 3, 7, 4, 8, 5, 10, 9]

    permuted_key = permute(key, P10)
    left, right = permuted_key[:5], permuted_key[5:]

    left = left[1:] + left[:1]
    right = right[1:] + right[:1]
    k1 = permute(left + right, P8)

    left = left[2:] + left[:2]
    right = right[2:] + right[:2]
    k2 = permute(left + right, P8)

    return k1, k2

# Encryption and decryption
def sdes_encrypt(plain_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(plain_text, IP)
    round1_result = fk(initial_permuted, k1)
    switched = switch(round1_result)
    round2_result = fk(switched, k2)
    cipher_text = permute(round2_result, IP_INV)
    return cipher_text

def sdes_decrypt(cipher_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(cipher_text, IP)
    round1_result = fk(initial_permuted, k2)
    switched = switch(round1_result)
    round2_result = fk(switched, k1)
    plain_text = permute(round2_result, IP_INV)
    return plain_text
```

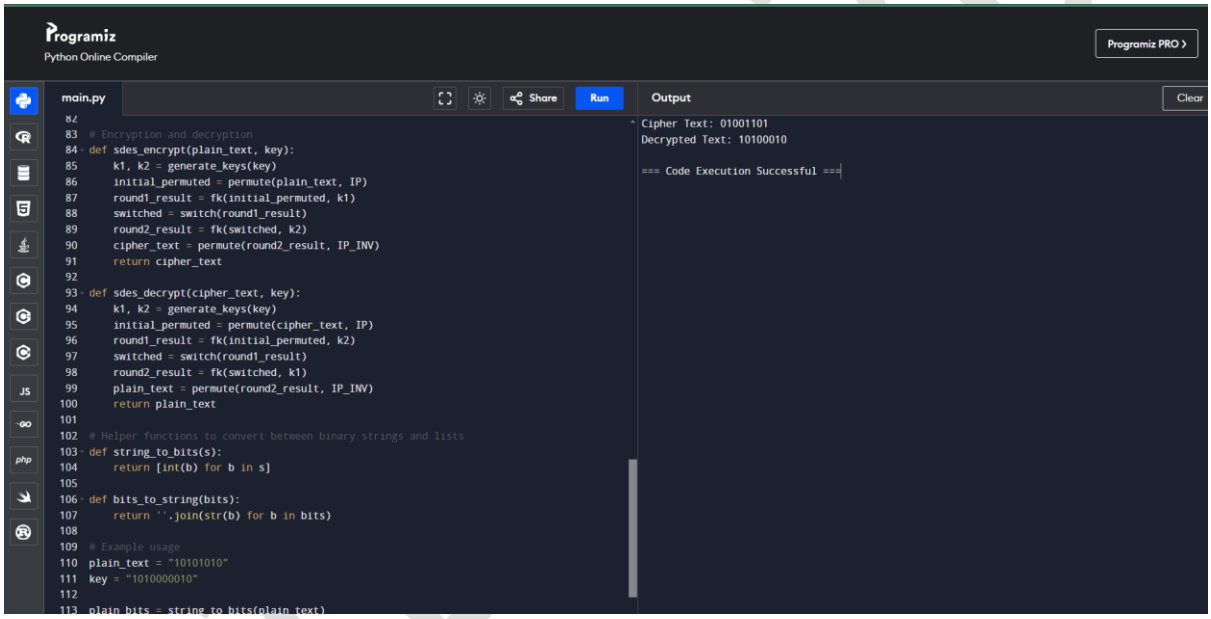
```
# Helper functions to convert between binary strings and lists
def string_to_bits(s):
    return [int(b) for b in s]

def bits_to_string(bits):
    return ''.join(str(b) for b in bits)

# Example usage
plain_text = "10101010"
key = "1010000010"

plain_bits = string_to_bits(plain_text)
key_bits = string_to_bits(key)

cipher_bits = sdes_encrypt(plain_bits, key_bits)
cipher_text = bits_to_string(cipher_bits)

print(f"Cipher Text: {cipher_text}")

decrypted_bits = sdes_decrypt(cipher_bits, key_bits)
decrypted_text = bits_to_string(decrypted_bits)

print(f"Decrypted Text: {decrypted_text}")
```

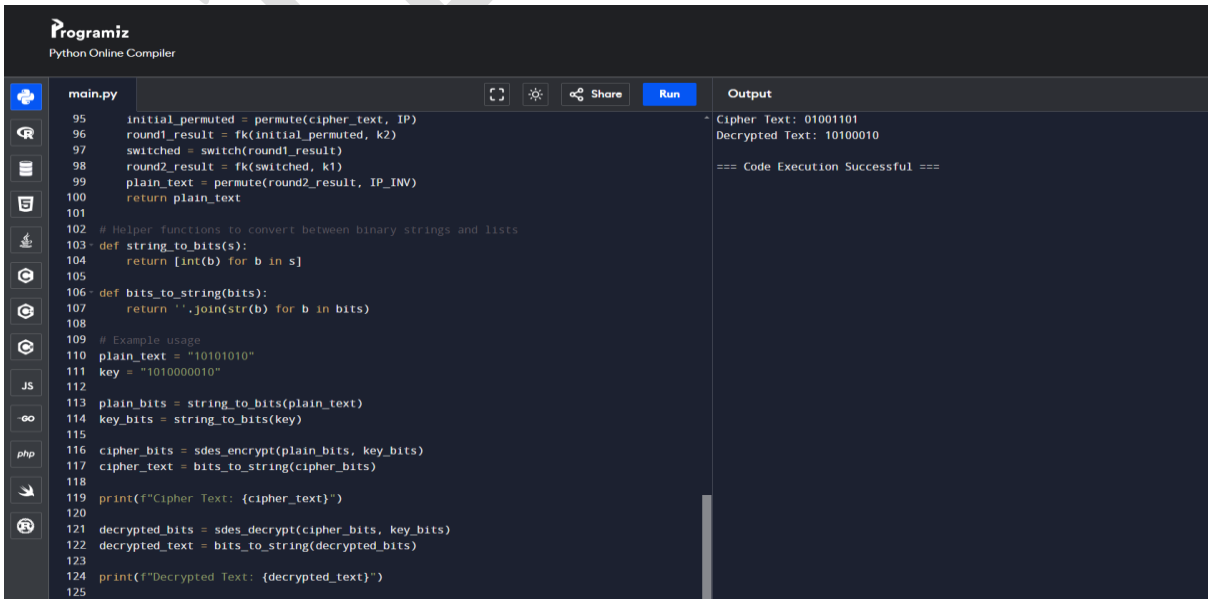
OUTPUT:


Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
# OM SUBRATO DEY - 21BAI1876
# SDES Implementation in Python
# Permutation functions
def permute(bits, table):
return [bits[i-1] for i in table]
# Initial permutation (IP)
IP = [2,1,2,1,1,8,7,6]
# Inverse initial permutation (IP^-1)
IP_INV = [4, 1, 3, 5, 7, 2, 8, 6]
# Expansion/permutation (E/P)
EP = [4, 1, 2, 3, 2, 3, 4, 1]
# Permutation function (P4)
P4 = [2, 4, 3, 1]
# S-Boxes
S0 = [
[1, 0, 3, 2],
[3, 2, 1, 0],
[0, 2, 1, 3],
[3, 1, 3, 2]
]
S1 = [
[0, 1, 2, 3],
[2, 0, 1, 3],
[3, 0, 1, 0],
[2, 1, 0, 3]
]
Cipher Text: 01001101
Decrypted Text: 10100010
=== Code Execution Successful ===


Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
[2, 1, 0, 3]
]
def sbox_lookup(sbox, row, col):
return sbox[row][col]
def fk(bits, key):
left, right = bits[:4], bits[4:]
expanded_permuted = permute(right, EP)
xor_result = [a ^ b for a, b in zip(expanded_permuted, key)]
left_half = xor_result[:4]
right_half = xor_result[4:]
row0 = (left_half[0] << 1) + left_half[3]
col0 = (left_half[1] << 1) + left_half[2]
s0_output = sbox_lookup(S0, row0, col0)
row1 = (right_half[0] << 1) + right_half[3]
col1 = (right_half[1] << 1) + right_half[2]
s1_output = sbox_lookup(S1, row1, col1)
sbox_output = [
(s0_output & 0b10) >> 1, s0_output & 0b01,
(s1_output & 0b10) >> 1, s1_output & 0b01
]
permuted_sbox_output = permute(sbox_output, P4)
left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]
return left_result + right
Cipher Text: 01001101
Decrypted Text: 10100010
=== Code Execution Successful ===


Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
permuted_sbox_output = permute(sbox_output, P4)
left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]
return left_result + right
def switch(bits):
return bits[4:] + bits[:4]
# Key generation
def generate_keys(key):
P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
P8 = [6, 3, 7, 4, 8, 5, 10, 9]
permuted_key = permute(key, P10)
left, right = permuted_key[:5], permuted_key[5:]
left = left[1:] + left[:1]
right = right[1:] + right[:1]
k1 = permute(left + right, P8)
left = left[2:] + left[:2]
right = right[2:] + right[:2]
k2 = permute(left + right, P8)
return k1, k2
# Encryption and decryption
def sdes_encrypt(plain_text, key):
k1, k2 = generate_keys(key)
initial_permuted = permute(plain_text, IP)
Cipher Text: 01001101
Decrypted Text: 10100010
=== Code Execution Successful ===


Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
# Encryption and decryption
def sdes_encrypt(plain_text, key):
k1, k2 = generate_keys(key)
initial_permuted = permute(plain_text, IP)
round1_result = fk(initial_permuted, k1)
switched = switch(round1_result)
round2_result = fk(switched, k2)
cipher_text = permute(round2_result, IP_INV)
return cipher_text
def sdes_decrypt(cipher_text, key):
k1, k2 = generate_keys(key)
initial_permuted = permute(cipher_text, IP)
round1_result = fk(initial_permuted, k2)
switched = switch(round1_result)
round2_result = fk(switched, k1)
plain_text = permute(round2_result, IP_INV)
return plain_text
# Helper functions to convert between binary strings and lists
def string_to_bits(s):
return [int(b) for b in s]
def bits_to_string(bits):
return ''.join(str(b) for b in bits)
# Example usage
plain_text = "10101010"
key = "1010000010"
plain_bits = string_to_bits(plain_text)
Cipher Text: 01001101
Decrypted Text: 10100010
=== Code Execution Successful ===


Programiz
Python Online Compiler
main.py
Share
Run
Output
initial_permuted = permute(cipher_text, IP)
round1_result = fk(initial_permuted, k2)
switched = switch(round1_result)
round2_result = fk(switched, k1)
plain_text = permute(round2_result, IP_INV)
return plain_text
# Helper functions to convert between binary strings and lists
def string_to_bits(s):
return [int(b) for b in s]
def bits_to_string(bits):
return ''.join(str(b) for b in bits)
# Example usage
plain_text = "10101010"
key = "1010000010"
plain_bits = string_to_bits(plain_text)
key_bits = string_to_bits(key)
cipher_bits = sdes_encrypt(plain_bits, key_bits)
cipher_text = bits_to_string(cipher_bits)
print(f"Cipher Text: {cipher_text}")
decrypted_bits = sdes_decrypt(cipher_bits, key_bits)
decrypted_text = bits_to_string(decrypted_bits)
print(f"Decrypted Text: {decrypted_text}")
Cipher Text: 01001101
Decrypted Text: 10100010
=== Code Execution Successful ===

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 5: IMPLEMENTATION OF DES ENCRYPTION USING SUITABLE PROGRAMMING LANGUAGE

NAME : OM SUBRATO DEY

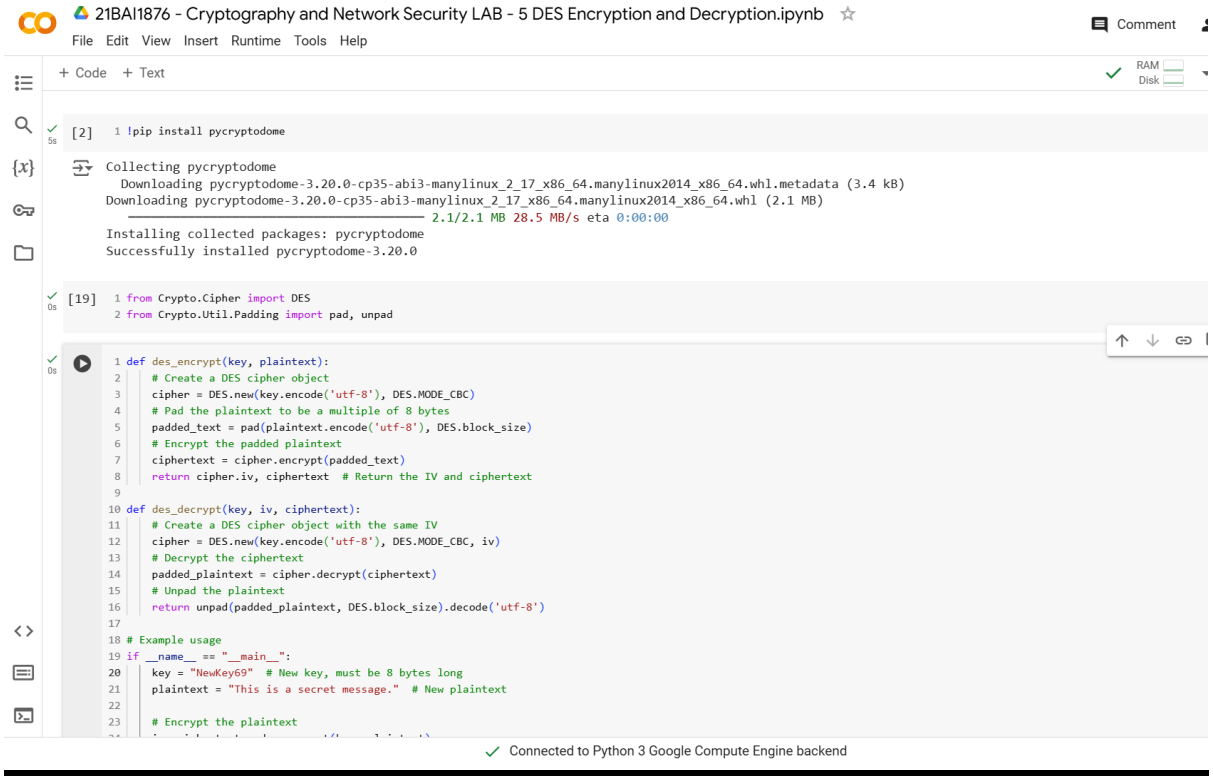
REGISTER NO.: 21BAI1876

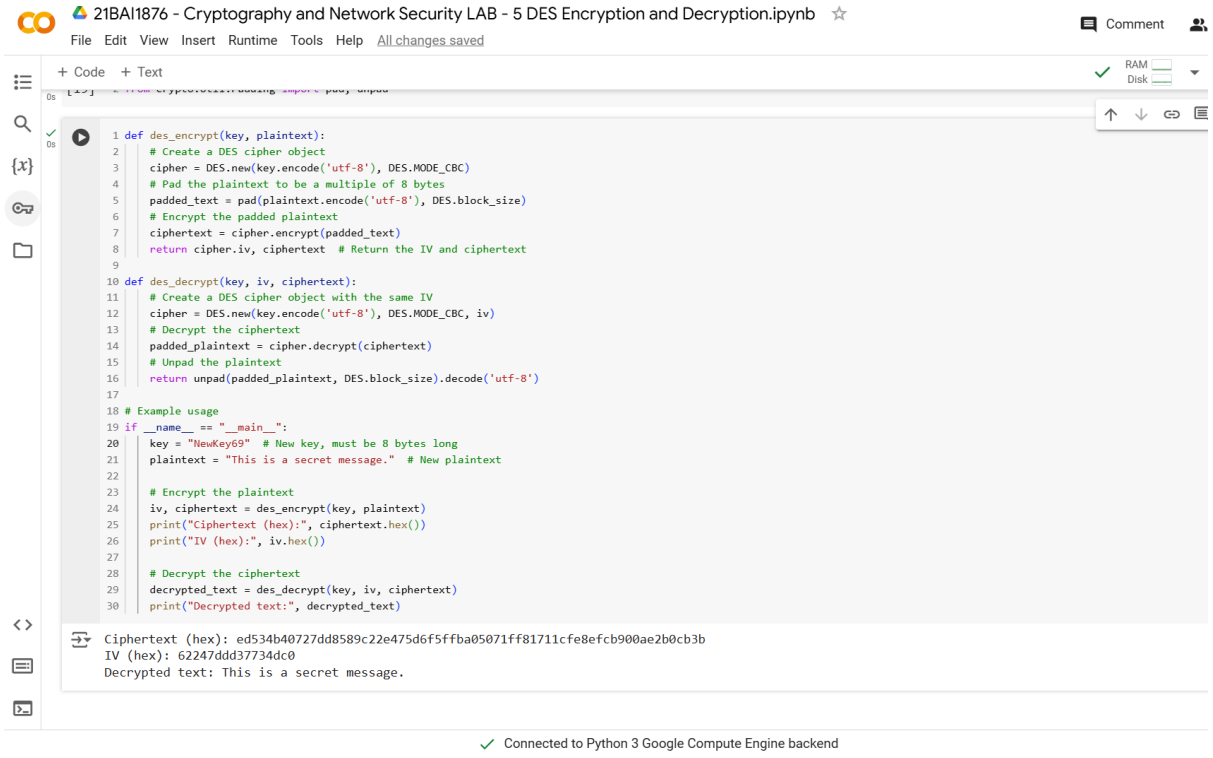
CODE:

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
def des_encrypt(key, plaintext):
    # Create a DES cipher object
    cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
    # Pad the plaintext to be a multiple of 8 bytes
    padded_text = pad(plaintext.encode('utf-8'),
DES.block_size)
    # Encrypt the padded plaintext
    ciphertext = cipher.encrypt(padded_text)
    return cipher.iv, ciphertext  # Return the IV and
ciphertext


def des_decrypt(key, iv, ciphertext):
    # Create a DES cipher object with the same IV
    cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC,
iv)
    # Decrypt the ciphertext
    padded_plaintext = cipher.decrypt(ciphertext)
    # Unpad the plaintext
    return unpad(padded_plaintext,
DES.block_size).decode('utf-8')


# Example usage
if __name__ == "__main__":
    key = "NewKey69"  # New key, must be 8 bytes long
```

```
    plaintext = "This is a secret message."  # New
plaintext


    # Encrypt the plaintext
    iv, ciphertext = des_encrypt(key, plaintext)
    print("Ciphertext (hex):", ciphertext.hex())
    print("IV (hex):", iv.hex())

    # Decrypt the ciphertext
    decrypted_text = des_decrypt(key, iv, ciphertext)
    print("Decrypted text:", decrypted_text)
```

OUTPUT:


21BAI1876 - Cryptography and Network Security LAB - 5 DES Encryption and Decryption.ipynb
File Edit View Insert Runtime Tools Help
Comment
+ Code + Text
RAM
Disk
[2] 1 !pip install pycryptodome
Collecting pycryptodome
Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
2.1/2.1 MB 28.5 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.20.0
[19] 1 from Crypto.Cipher import DES
2 from Crypto.Util.Padding import pad, unpad
1 def des_encrypt(key, plaintext):
2 # Create a DES cipher object
3 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
4 # Pad the plaintext to be a multiple of 8 bytes
5 padded_text = pad(plaintext.encode('utf-8'), DES.block_size)
6 # Encrypt the padded plaintext
7 ciphertext = cipher.encrypt(padded_text)
8 return cipher.iv, ciphertext # Return the IV and ciphertext
9
10 def des_decrypt(key, iv, ciphertext):
11 # Create a DES cipher object with the same IV
12 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC, iv)
13 # Decrypt the ciphertext
14 padded_plaintext = cipher.decrypt(ciphertext)
15 # Unpad the plaintext
16 return unpad(padded_plaintext, DES.block_size).decode('utf-8')
17
18 # Example usage
19 if __name__ == "__main__":
20 key = "NewKey69" # New key, must be 8 bytes long
21 plaintext = "This is a secret message." # New plaintext
22
23 # Encrypt the plaintext
Connected to Python 3 Google Compute Engine backend


21BAI1876 - Cryptography and Network Security LAB - 5 DES Encryption and Decryption.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
+ Code + Text
RAM
Disk
1 def des_encrypt(key, plaintext):
2 # Create a DES cipher object
3 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
4 # Pad the plaintext to be a multiple of 8 bytes
5 padded_text = pad(plaintext.encode('utf-8'), DES.block_size)
6 # Encrypt the padded plaintext
7 ciphertext = cipher.encrypt(padded_text)
8 return cipher.iv, ciphertext # Return the IV and ciphertext
9
10 def des_decrypt(key, iv, ciphertext):
11 # Create a DES cipher object with the same IV
12 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC, iv)
13 # Decrypt the ciphertext
14 padded_plaintext = cipher.decrypt(ciphertext)
15 # Unpad the plaintext
16 return unpad(padded_plaintext, DES.block_size).decode('utf-8')
17
18 # Example usage
19 if __name__ == "__main__":
20 key = "NewKey69" # New key, must be 8 bytes long
21 plaintext = "This is a secret message." # New plaintext
22
23 # Encrypt the plaintext
24 iv, ciphertext = des_encrypt(key, plaintext)
25 print("Ciphertext (hex):", ciphertext.hex())
26 print("IV (hex):", iv.hex())
27
28 # Decrypt the ciphertext
29 decrypted_text = des_decrypt(key, iv, ciphertext)
30 print("Decrypted text:", decrypted_text)
Ciphertext (hex): ed534b40727dd8589c22e475d6f5ffba05071ff81711cfe8efcb900ae2b0cb3b
IV (hex): 62247ddd37734dc0
Decrypted text: This is a secret message.
Connected to Python 3 Google Compute Engine backend

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 6: IMPLEMENTATION OF DES ENCRYPTION USING SUITABLE PROGRAMMING LANGUAGE

NAME : OM SUBRATO DEY

REGISTER NO.: 21BAI1876

CODE:

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
def des_encrypt(key, plaintext):
    # Create a DES cipher object
    cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
    # Pad the plaintext to be a multiple of 8 bytes
    padded_text = pad(plaintext.encode('utf-8'),
DES.block_size)
    # Encrypt the padded plaintext
    ciphertext = cipher.encrypt(padded_text)
    return cipher.iv, ciphertext  # Return the IV and
ciphertext


def des_decrypt(key, iv, ciphertext):
    # Create a DES cipher object with the same IV
    cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC,
iv)
    # Decrypt the ciphertext
    padded_plaintext = cipher.decrypt(ciphertext)
    # Unpad the plaintext
    return unpad(padded_plaintext,
DES.block_size).decode('utf-8')


# Example usage
if __name__ == "__main__":
    key = "NewKey69"  # New key, must be 8 bytes long
```

```
    plaintext = "This is a secret message."  # New
plaintext


    # Encrypt the plaintext
    iv, ciphertext = des_encrypt(key, plaintext)
    print("Ciphertext (hex):", ciphertext.hex())
    print("IV (hex):", iv.hex())

    # Decrypt the ciphertext
    decrypted_text = des_decrypt(key, iv, ciphertext)
    print("Decrypted text:", decrypted_text)
```

OUTPUT:

21BAI1876 - Cryptography and Network Security LAB - 5 DES Encryption and Decryption.ipynb

File Edit View Insert Runtime Tools Help

Comment

+ Code + Text

RAM Disk

```
[2]  1 !pip install pycryptodome
```

```
Collecting pycryptodome
  Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 28.5 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.20.0
```

```
[19]  1 from Crypto.Cipher import DES
      2 from Crypto.Util.Padding import pad, unpad
```

```
 1 def des_encrypt(key, plaintext):
 2     # Create a DES cipher object
 3     cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
 4     # Pad the plaintext to be a multiple of 8 bytes
 5     padded_text = pad(plaintext.encode('utf-8'), DES.block_size)
 6     # Encrypt the padded plaintext
 7     ciphertext = cipher.encrypt(padded_text)
 8     return cipher.iv, ciphertext  # Return the IV and ciphertext
 9
10 def des_decrypt(key, iv, ciphertext):
11     # Create a DES cipher object with the same IV
12     cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC, iv)
13     # Decrypt the ciphertext
14     padded_plaintext = cipher.decrypt(ciphertext)
15     # Unpad the plaintext
16     return unpad(padded_plaintext, DES.block_size).decode('utf-8')
17
18 # Example usage
19 if __name__ == "__main__":
20     key = "NewKey69"  # New key, must be 8 bytes long
21     plaintext = "This is a secret message."  # New plaintext
22
23     # Encrypt the plaintext
```

Connected to Python 3 Google Compute Engine backend

21BAI1876 - Cryptography and Network Security LAB - 5 DES Encryption and Decryption.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment

+ Code + Text

RAM Disk

```
 1 def des_encrypt(key, plaintext):
 2     # Create a DES cipher object
 3     cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
 4     # Pad the plaintext to be a multiple of 8 bytes
 5     padded_text = pad(plaintext.encode('utf-8'), DES.block_size)
 6     # Encrypt the padded plaintext
 7     ciphertext = cipher.encrypt(padded_text)
 8     return cipher.iv, ciphertext  # Return the IV and ciphertext
 9
10 def des_decrypt(key, iv, ciphertext):
11     # Create a DES cipher object with the same IV
12     cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC, iv)
13     # Decrypt the ciphertext
14     padded_plaintext = cipher.decrypt(ciphertext)
15     # Unpad the plaintext
16     return unpad(padded_plaintext, DES.block_size).decode('utf-8')
17
18 # Example usage
19 if __name__ == "__main__":
20     key = "NewKey69"  # New key, must be 8 bytes long
21     plaintext = "This is a secret message."  # New plaintext
22
23     # Encrypt the plaintext
24     iv, ciphertext = des_encrypt(key, plaintext)
25     print("Ciphertext (hex):", ciphertext.hex())
26     print("IV (hex):", iv.hex())
27
28     # Decrypt the ciphertext
29     decrypted_text = des_decrypt(key, iv, ciphertext)
30     print("Decrypted text:", decrypted_text)
```

```
Ciphertext (hex): ed534b40727dd8589c22e475d6f5ffba05071ff81711cfe8efcb900ae2b0cb3b
IV (hex): 62247ddd37734dc0
Decrypted text: This is a secret message.
```

Connected to Python 3 Google Compute Engine backend

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 7: DIFFIE HELMAN KEY EXCHANGE ALGORITHM IMPLEMENTATION

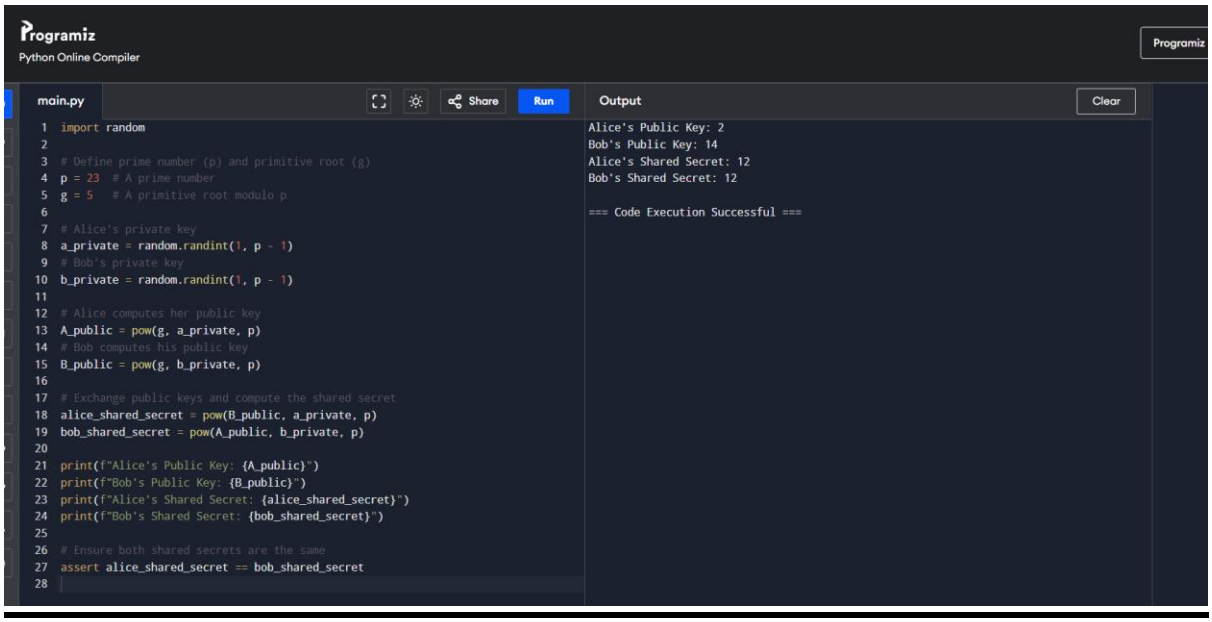
NAME : OM SUBRATO DEY

REGISTER NUMBER : 21BAI1876

CODE:

```
import random
# Define prime number (p) and primitive root (g)
p = 23  # A prime number
g = 5   # A primitive root modulo p

# Alice's private key
a_private = random.randint(1, p - 1)
# Bob's private key
b_private = random.randint(1, p - 1)

# Alice computes her public key
A_public = pow(g, a_private, p)
# Bob computes his public key
B_public = pow(g, b_private, p)

# Exchange public keys and compute the shared secret
alice_shared_secret = pow(B_public, a_private, p)
bob_shared_secret = pow(A_public, b_private, p)

print(f"Alice's Public Key: {A_public}")
print(f"Bob's Public Key: {B_public}")
print(f"Alice's Shared Secret: {alice_shared_secret}")
print(f"Bob's Shared Secret: {bob_shared_secret}")

# Ensure both shared secrets are the same
assert alice_shared_secret == bob_shared_secret
```

OUTPUT:


Programiz
Python Online Compiler
Programiz
main.py
Share
Run
Output
Clear
1 import random
2
3 # Define prime number (p) and primitive root (g)
4 p = 23 # A prime number
5 g = 5   # A primitive root modulo p
6
7 # Alice's private key
8 a_private = random.randint(1, p - 1)
9 # Bob's private key
10 b_private = random.randint(1, p - 1)
11
12 # Alice computes her public key
13 A_public = pow(g, a_private, p)
14 # Bob computes his public key
15 B_public = pow(g, b_private, p)
16
17 # Exchange public keys and compute the shared secret
18 alice_shared_secret = pow(B_public, a_private, p)
19 bob_shared_secret = pow(A_public, b_private, p)
20
21 print(f"Alice's Public Key: {A_public}")
22 print(f"Bob's Public Key: {B_public}")
23 print(f"Alice's Shared Secret: {alice_shared_secret}")
24 print(f"Bob's Shared Secret: {bob_shared_secret}")
25
26 # Ensure both shared secrets are the same
27 assert alice_shared_secret == bob_shared_secret
28
Alice's Public Key: 2
Bob's Public Key: 14
Alice's Shared Secret: 12
Bob's Shared Secret: 12
=== Code Execution Successful ===

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 8 : MAN IN THE MIDDLE ATTACK

NAME : OM SUBRATO DEY

REGISTER NO. : 21BAI1876

THEORY:

To explain an idea of the Man-in-the-Middle (MITM) attack in python we can write a basic example of how an attacker will interrupt communication between two clients. But it is important to note that practical MITM attacks involve a lot of network tweaking and normally involve scapy or mitmproxy.

For this demonstration, we'll simulate the following:For this demonstration, we'll simulate the following:

Alice: The agent which this means that Bob wants to receive a message from the sender.

Bob: The receiver who anticipate a message be sent by Alice.

Mallory: The receiver who intent to alter the Information content of the message that has been sent by the sender.

We shall develop another Python script to mimic the message exchange and present how Mallory can modify the actual message.

MITM Simulation code in python

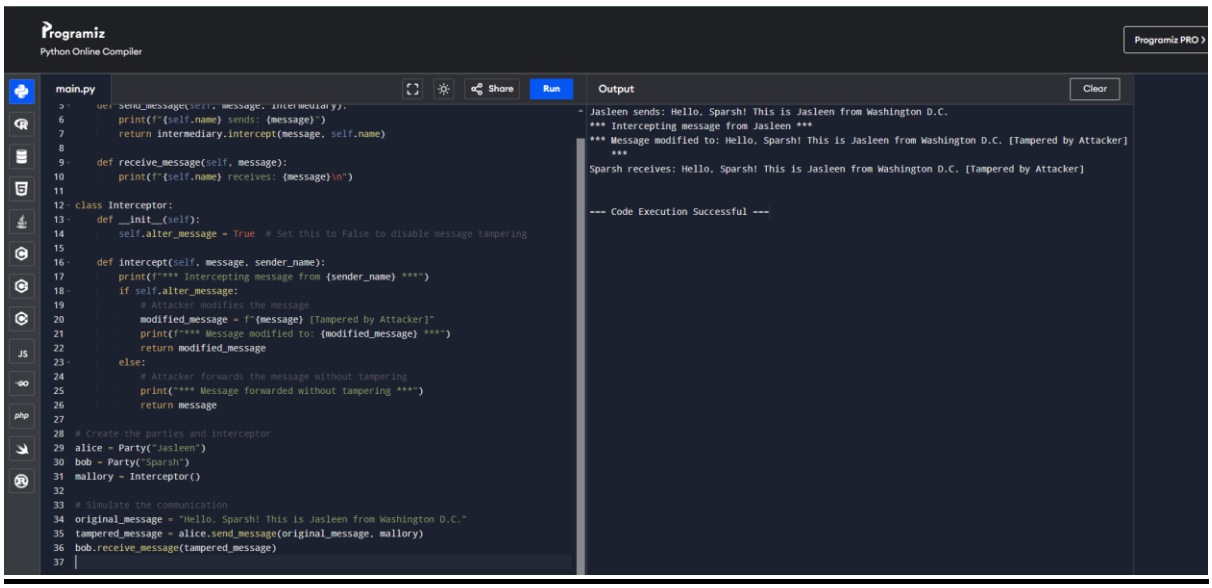
Here is a Python script that demonstrates a simple MITM attack:Here is a Python script that demonstrates a simple MITM attack:

CODE:

```
class Party:
    def __init__(self, name):
        self.name = name

    def send_message(self, message, intermediary):
        print(f"{self.name} sends: {message}")
        return intermediary.intercept(message, self.name)

    def receive_message(self, message):
        print(f"{self.name} receives: {message}\n")

class Interceptor:
    def __init__(self):
        self.alter_message = True  # Set this to False to
disable message tampering

    def intercept(self, message, sender_name):
        print(f"*** Intercepting message from {sender_name}
***")
        if self.alter_message:
            # Attacker modifies the message
            modified_message = f"{message} [Tampered by
Attacker]"
            print(f"*** Message modified to: {modified_message}
***")
```

```
            return modified_message
        else:
            # Attacker forwards the message without tampering
            print("*** Message forwarded without tampering ***")
            return message

# Create the parties and interceptor
alice = Party("Alice")
bob = Party("Bob")
mallory = Interceptor()

# Simulate the communication
original_message = "Hello, Bob! This is Alice."
tampered_message = alice.send_message(original_message, mallory)
bob.receive_message(tampered_message)
```

OUTPUT:



CRYPTOGRAPHY AND NETWORK SECURITY

LAB 9 : ELGAMMAL ENCRYPTION AND DECRYPTION

NAME : OM SUBRATO DEY

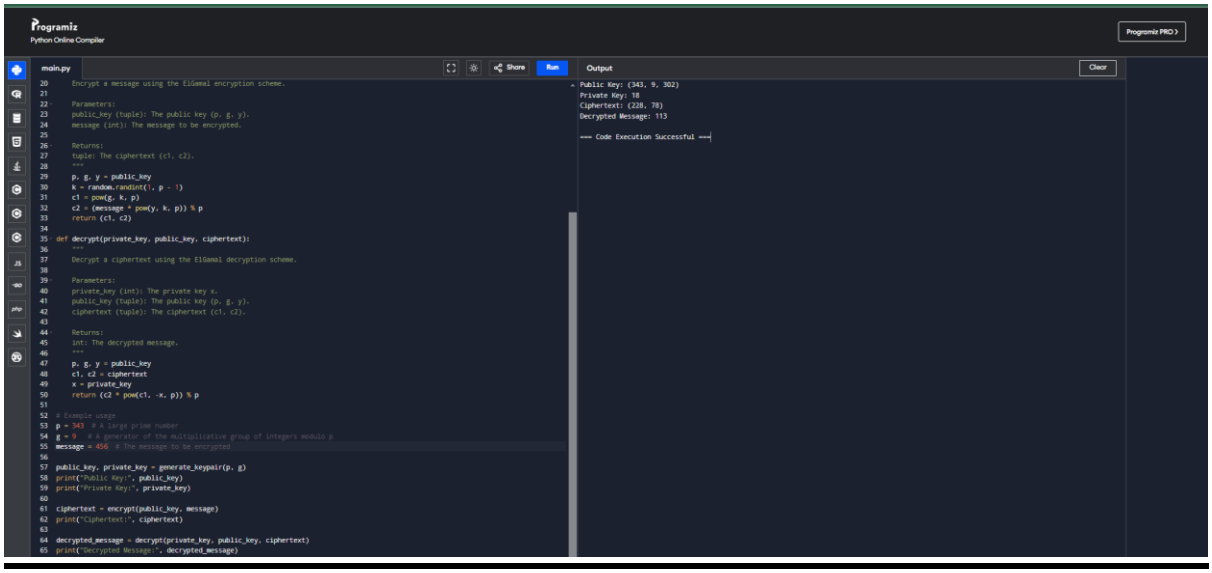
REGISTER NO.: 21BAI1876

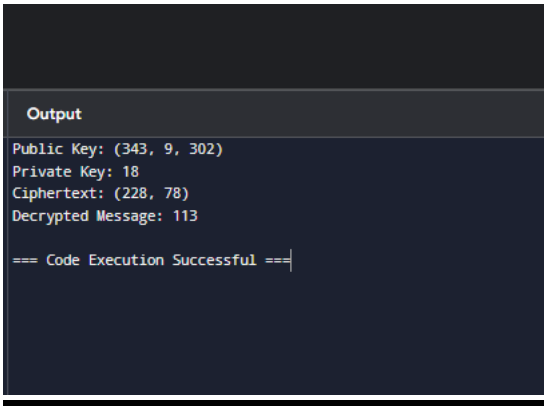
CODE:

```
import random

def generate_keypair(p, g):
    """
    Generate a public and private key pair.

    Parameters:
    p (int): A large prime number.
    g (int): A generator of the multiplicative group of integers modulo p.

    Returns:
    tuple: A tuple containing the public key (p, g, y) and the private key x.
    """
    x = random.randint(1, p - 1)
    y = pow(g, x, p)
    return ((p, g, y), x)

def encrypt(public_key, message):
    """
    Encrypt a message using the ElGamal encryption scheme.

    Parameters:
    public_key (tuple): The public key (p, g, y).
    message (int): The message to be encrypted.

    Returns:
    tuple: The ciphertext (c1, c2).
    """
    p, g, y = public_key
    k = random.randint(1, p - 1)
    c1 = pow(g, k, p)
    c2 = (message * pow(y, k, p)) % p
```

```
    return (c1, c2)

def decrypt(private_key, public_key, ciphertext):
    """
    Decrypt a ciphertext using the ElGamal decryption scheme.

    Parameters:
    private_key (int): The private key x.
    public_key (tuple): The public key (p, g, y).
    ciphertext (tuple): The ciphertext (c1, c2).

    Returns:
    int: The decrypted message.
    """
    p, g, y = public_key
    c1, c2 = ciphertext
    x = private_key
    return (c2 * pow(c1, -x, p)) % p

# Example usage
p = 343  # A large prime number
g = 9   # A generator of the multiplicative group of integers modulo p
message = 456  # The message to be encrypted

public_key, private_key = generate_keypair(p, g)
print("Public Key:", public_key)
print("Private Key:", private_key)

ciphertext = encrypt(public_key, message)
print("Ciphertext:", ciphertext)

decrypted_message = decrypt(private_key, public_key, ciphertext)
print("Decrypted Message:", decrypted_message)
```

OUTPUT:


Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
Public Key: (343, 9, 302)
Private Key: 18
Ciphertext: (228, 78)
Decrypted Message: 113
=== Code Execution Successful ===


Output
Public Key: (343, 9, 302)
Private Key: 18
Ciphertext: (228, 78)
Decrypted Message: 113
=== Code Execution Successful ===

CRYPTOGRAPHY AND NETWORK SECURITY
LAB 10 : MESSAGE DIGEST 5 ALGORITHM

NAME : OM SUBRATO DEY

REGISTER NO.: 21BAI1876

Belonging to the family of hash functions, MD5 generates a 128 bit hash value. Here are some of its main advantages and disadvantages:

Advantages:

The hash value is produced by MD5 almost right away.

- It can be applied in a program effortlessly.

Small fingerprint - Another advantage of the algorithm, a 128-bit MD5 hash is easy to work with.

In order to produce the hash using MD5, no secret encryption key is necessary.

Disadvantages:

Although MD5 is a common choice, certainty has emerged about its collision vulnerability, demonstrating that it is possible for researchers to intentionally produce two different messages that yield the same hash.

That means the attacker can get inputs that hash to a given MD5 hash through a preimage attack … They are prone to preimage attacks since the construction of the hash is only 128 bits.

It is not capable enough - MD5 is no longer considered cryptographically sound hashing algorithm. MD5 ought to be discarded; on the other hand, hash functions derived from the SHA-256 family are favored.

- At risk of birthday attacks - The preceding hash size is insufficient when compared to others and, therefore, is susceptible to birthday attacks to discover collisions

Briefly, it is safe to say that, despite MD5 being computationally quick and simple to implement, it doesn't provide sufficient collision resistance and is unsatisfactory for present applications. It is primarily used today for data check summing but not for encryption or authentication..

CODE:

```
import hashlib
import os


# Function to generate an MD5 hash of a string
def md5_encrypt(data: str) -> str:
    """Generate MD5 hash of the given string."""
    md5_hasher = hashlib.md5()
    md5_hasher.update(data.encode('utf-8'))
    return md5_hasher.hexdigest()


# Simulate user registration (storing password as MD5 hash)
def register_user(username: str, password: str, users_db: dict) ->
None:
    """Registers a user by storing their username and hashed
password."""
    if username in users_db:
        print(f"User '{username}' already exists!")
        return


    # Hash the password
    hashed_password = md5_encrypt(password)


    # Simulate storing in the database
    users_db[username] = hashed_password
    print(f"User '{username}' registered successfully with hashed
password: {hashed_password}")


# Simulate user login (verifying password)
def login_user(username: str, password: str, users_db: dict) -> None:
    """Verifies a user's login by comparing the hashed password."""
    if username not in users_db:
```

```
        print(f"User '{username}' does not exist!")
        return

    # Hash the entered password to compare
    entered_password_hash = md5_encrypt(password)

    # Check if the hash matches the stored hash
    if entered_password_hash == users_db[username]:
        print(f"Login successful for user '{username}'!")
    else:
        print("Incorrect password.")

# Simulate the "database" to store user data
users_db = {}

# Registering users
print("Registering users...")
register_user("alice", "password123", users_db)
register_user("bob", "mySecretPass", users_db)
print()

# Attempting logins
print("Attempting logins...")
login_user("alice", "password123", users_db)  # Correct password
login_user("bob", "wrongPass", users_db)      # Incorrect password
login_user("charlie", "randomPass", users_db) # User doesn't exist

print("\nCurrent user database (hashed passwords):")
print(users_db)
```

OUTPUT:

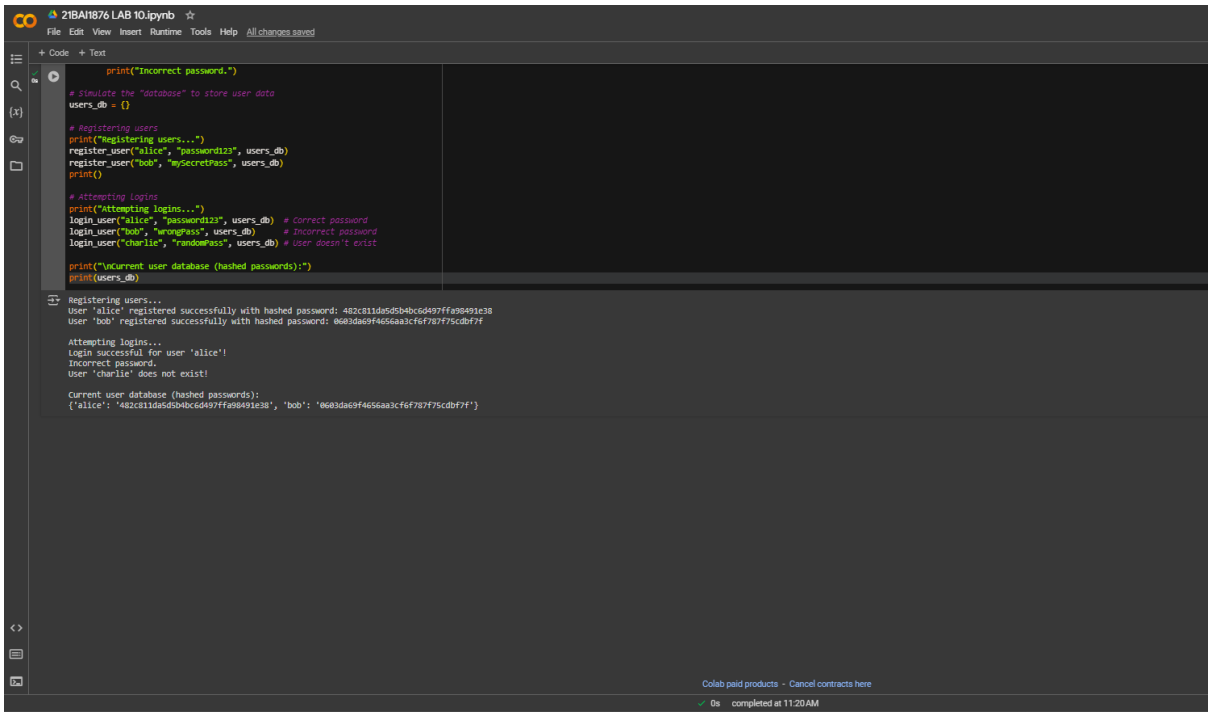
21BAI1876 LAB 10.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
import hashlib
import os

# Function to generate an MD5 hash of a string
def md5_encrypt(data: str) -> str:
    """Generate MD5 hash of the given string."""
    md5_hasher = hashlib.md5()
    md5_hasher.update(data.encode('utf-8'))
    return md5_hasher.hexdigest()

# Simulate user registration (storing password as MD5 hash)
def register_user(username: str, password: str, users_db: dict) -> None:
    """Registers a user by storing their username and hashed password."""
    if username in users_db:
        print(f"User '{username}' already exists!")
        return

    # Hash the password
    hashed_password = md5_encrypt(password)

    # Simulate storing in the database
    users_db[username] = hashed_password
    print(f"User '{username}' registered successfully with hashed password: {hashed_password}")

# Simulate user login (verifying password)
def login_user(username: str, password: str, users_db: dict) -> None:
    """Verifies a user's login by comparing the hashed password."""
    if username not in users_db:
        print(f"User '{username}' does not exist!")
        return

    # Hash the entered password to compare
    entered_password_hash = md5_encrypt(password)

    # Check if the hash matches the stored hash
    if entered_password_hash == users_db[username]:
        print(f"Login successful for user '{username}'!")
    else:
        print("Incorrect password.")

# Simulate the "database" to store user data
users_db = {}

# Registering users
print("Registering users...")
register_user("alice", "password123", users_db)
register_user("bob", "mySecretPass", users_db)
print()

# Attempting logins
print("Attempting logins...")
login_user("alice", "password123", users_db)  # Correct password
login_user("bob", "wrongPass", users_db)      # Incorrect password
```

0s completed at 11:20 AM



CRYPTOGRAPHY AND NETWORK SECURITY

LAB 11 : SHA – 512 ALGORITHM

NAME : OM SUBRATO DEY

REGISTER NO. : 21BAI1876

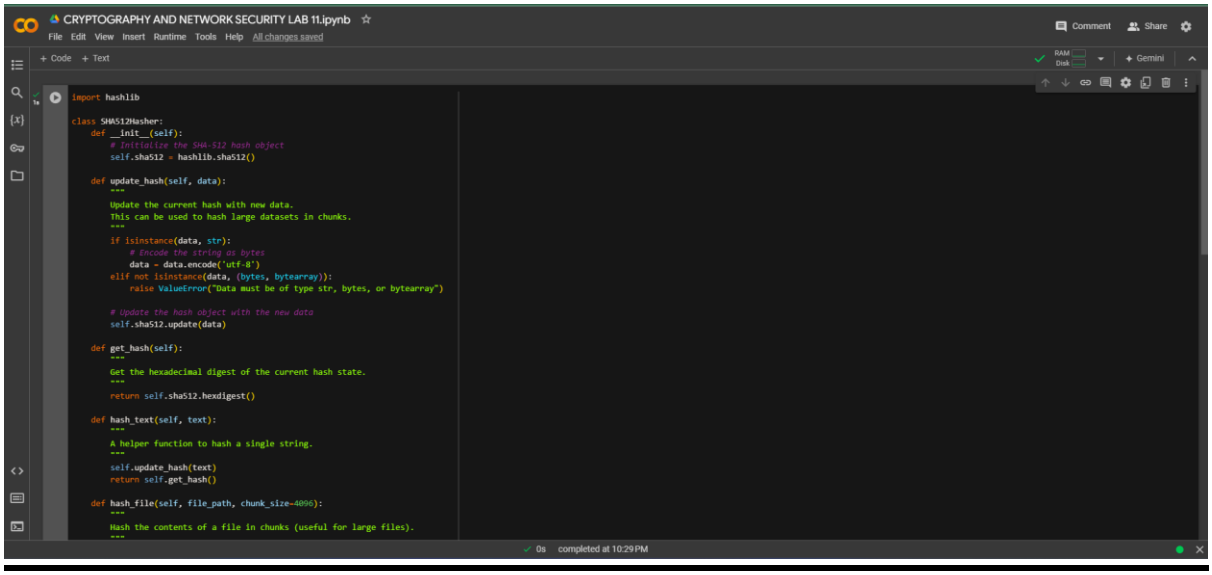
CODE:

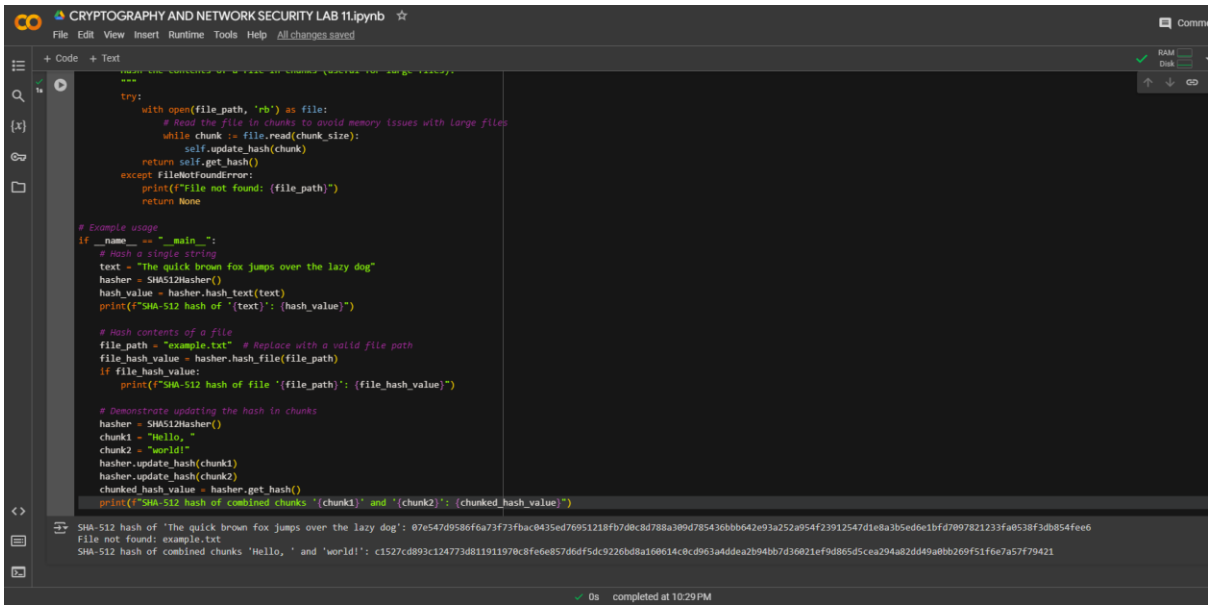
```
import hashlib
class SHA512Hasher:
    def __init__(self):
        # Initialize the SHA-512 hash object
        self.sha512 = hashlib.sha512()

    def update_hash(self, data):
        """
        Update the current hash with new data.
        This can be used to hash large datasets in
chunks.
        """
        if isinstance(data, str):
            # Encode the string as bytes
            data = data.encode('utf-8')
        elif not isinstance(data, (bytes, bytearray)):
            raise ValueError("Data must be of type str,
bytes, or bytearray")

        # Update the hash object with the new data
        self.sha512.update(data)

    def get_hash(self):
        """
        Get the hexadecimal digest of the current hash
state.
```

```
        """
        return self.sha512.hexdigest()

    def hash_text(self, text):
        """
        A helper function to hash a single string.
        """
        self.update_hash(text)
        return self.get_hash()

    def hash_file(self, file_path, chunk_size=4096):
        """
        Hash the contents of a file in chunks (useful for large files).
        """
        try:
            with open(file_path, 'rb') as file:
                # Read the file in chunks to avoid memory issues with large files
                while chunk := file.read(chunk_size):
                    self.update_hash(chunk)
            return self.get_hash()
        except FileNotFoundError:
            print(f"File not found: {file_path}")
            return None

# Example usage
if __name__ == "__main__":
```

```
    # Hash a single string
    text = "The quick brown fox jumps over the lazy
dog"
    hasher = SHA512Hasher()
    hash_value = hasher.hash_text(text)
    print(f"SHA-512 hash of '{text}': {hash_value}")


    # Hash contents of a file
    file_path = "example.txt"  # Replace with a valid
file path
    file_hash_value = hasher.hash_file(file_path)
    if file_hash_value:
        print(f"SHA-512 hash of file '{file_path}':
{file_hash_value}")


    # Demonstrate updating the hash in chunks
    hasher = SHA512Hasher()
    chunk1 = "Hello, "
    chunk2 = "world!"
    hasher.update_hash(chunk1)
    hasher.update_hash(chunk2)
    chunked_hash_value = hasher.get_hash()
    print(f"SHA-512 hash of combined chunks '{chunk1}'
and '{chunk2}': {chunked_hash_value}")
```

OUTPUT:


CRYPTOGRAPHY AND NETWORK SECURITY LAB 11.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Gemini
import hashlib
class SHA512Hasher:
def __init__(self):
# Initialize the SHA-512 hash object
self.sha512 = hashlib.sha512()
def update_hash(self, data):
"""
Update the current hash with new data.
This can be used to hash large datasets in chunks.
"""
if isinstance(data, str):
# Encode the string as bytes
data = data.encode('utf-8')
elif not isinstance(data, (bytes, bytearray)):
raise ValueError("Data must be of type str, bytes, or bytearray")
# Update the hash object with the new data
self.sha512.update(data)
def get_hash(self):
"""
Get the hexadecimal digest of the current hash state.
"""
return self.sha512.hexdigest()
def hash_text(self, text):
"""
A helper function to hash a single string.
"""
self.update_hash(text)
return self.get_hash()
def hash_file(self, file_path, chunk_size=4096):
"""
Hash the contents of a file in chunks (useful for large files).
"""
0s completed at 10:29 PM


CRYPTOGRAPHY AND NETWORK SECURITY LAB 11.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
try:
with open(file_path, 'rb') as file:
# Read the file in chunks to avoid memory issues with large files
while chunk := file.read(chunk_size):
self.update_hash(chunk)
return self.get_hash()
except FileNotFoundError:
print(f"File not found: {file_path}")
return None
# Example usage
if __name__ == "__main__":
# Hash a single string
text = "The quick brown fox jumps over the lazy dog"
hasher = SHA512Hasher()
hash_value = hasher.hash_text(text)
print(f"SHA-512 hash of '{text}': {hash_value}")
# Hash contents of a file
file_path = "example.txt" # Replace with a valid file path
file_hash_value = hasher.hash_file(file_path)
if file_hash_value:
print(f"SHA-512 hash of file '{file_path}': {file_hash_value}")
# Demonstrate updating the hash in chunks
hasher = SHA512Hasher()
chunk1 = "Hello, "
chunk2 = "world!"
hasher.update_hash(chunk1)
hasher.update_hash(chunk2)
chunked_hash_value = hasher.get_hash()
print(f"SHA-512 hash of combined chunks '{chunk1}' and '{chunk2}': {chunked_hash_value}")
SHA-512 hash of 'The quick brown fox jumps over the lazy dog': 07e547d9586f6a73f73fbac0435ed76951218fb7d0c8d788a309d785436bbb642e93a252a954f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6
File not found: example.txt
SHA-512 hash of combined chunks 'Hello, ' and 'world!': c1527cd893c124773d811911970c8fe6e857d6df5dc9226bd8a160614c0cd963a4ddea2b94bb7d36021ef9d865d5cea294a82dd49a0bb269f51f6e7a57f79421
0s completed at 10:29 PM

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 12 : DIGITAL SIGNATURES [DSS AND ELGAMMAL]

NAME : OM SUBRATO DEY

REGISTER NO. : 21BAI1876

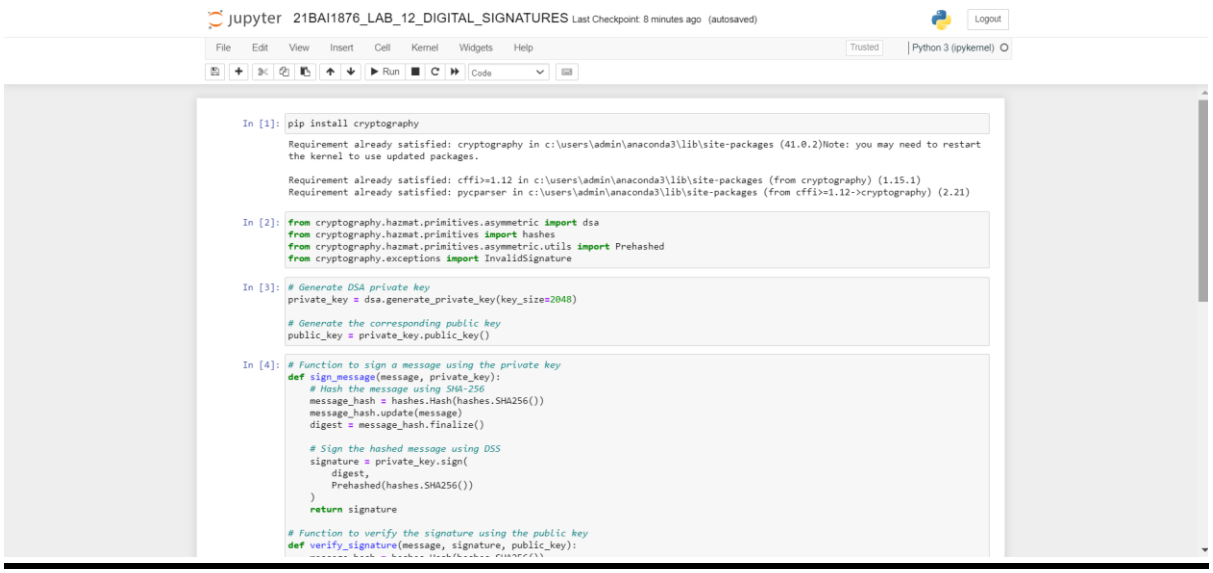
1. DSS DIGITAL SIGNATURE

CODE:

```
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.utils import Prehashed
from cryptography.exceptions import InvalidSignature

# Generate DSA private key
private_key = dsa.generate_private_key(key_size=2048)

# Generate the corresponding public key
public_key = private_key.public_key()

# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature
```

```
# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False

# Example usage
message = b"This is a secure message."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)

if is_valid:
    print("Signature is valid!")
else:
    print("Signature is invalid.")
```

OUTPUT:


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

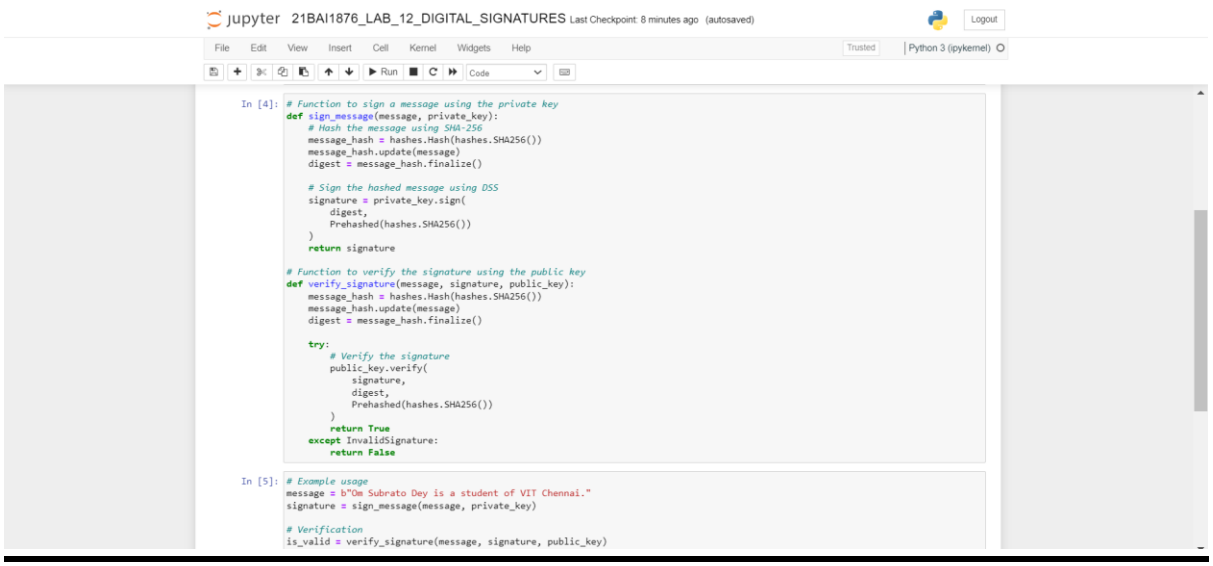
In [1]:
```
pip install cryptography
```

```
Requirement already satisfied: cryptography in c:\users\admin\anaconda3\lib\site-packages (41.0.2)Note: you may need to restart
the kernel to use updated packages.

Requirement already satisfied: cffi>=1.12 in c:\users\admin\anaconda3\lib\site-packages (from cryptography) (1.15.1)
Requirement already satisfied: pycparser in c:\users\admin\anaconda3\lib\site-packages (from cffi>=1.12->cryptography) (2.21)
```

In [2]:
```
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.utils import Prehashed
from cryptography.exceptions import InvalidSignature
```

In [3]:
```
# Generate DSA private key
private_key = dsa.generate_private_key(key_size=2048)

# Generate the corresponding public key
public_key = private_key.public_key()
```

In [4]:
```
# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature

# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
```


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

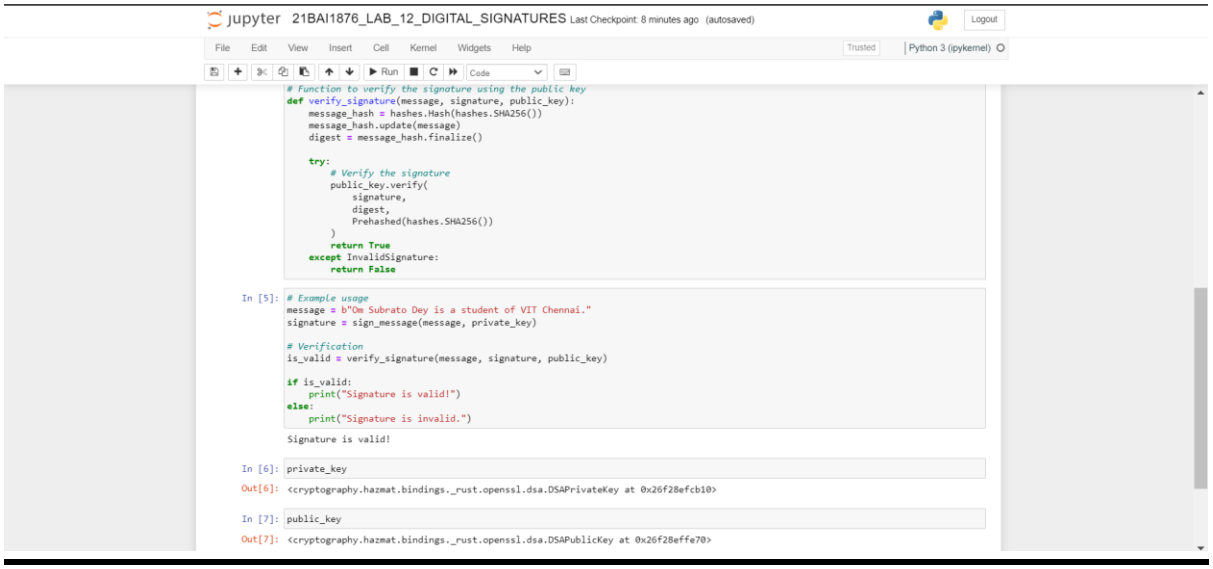
File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [4]:
```
# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature

# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False
```

In [5]:
```
# Example usage
message = b"Om Subrato Dey is a student of VIT Chennai."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)
```


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False
```

In [5]:
```
# Example usage
message = b"Om Subrato Dey is a student of VIT Chennai."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)

if is_valid:
    print("Signature is valid!")
else:
    print("Signature is invalid.")
```

```
Signature is valid!
```

In [6]:
```
private_key
```

Out[6]:
```
<cryptography.hazmat.bindings._rust.openssl.dsa.DSAPrivateKey at 0x26f28efcb10>
```

In [7]:
```
public_key
```

Out[7]:
```
<cryptography.hazmat.bindings._rust.openssl.dsa.DSAPublicKey at 0x26f28effe70>
```

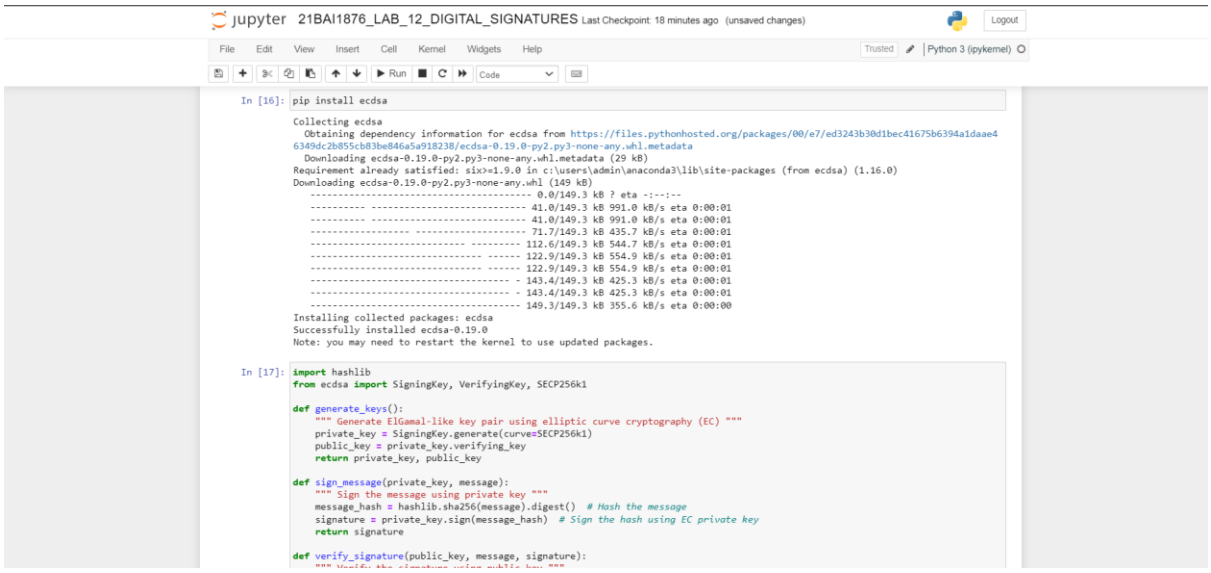
2. ELGAMMAL DIGITAL SIGNATURE:

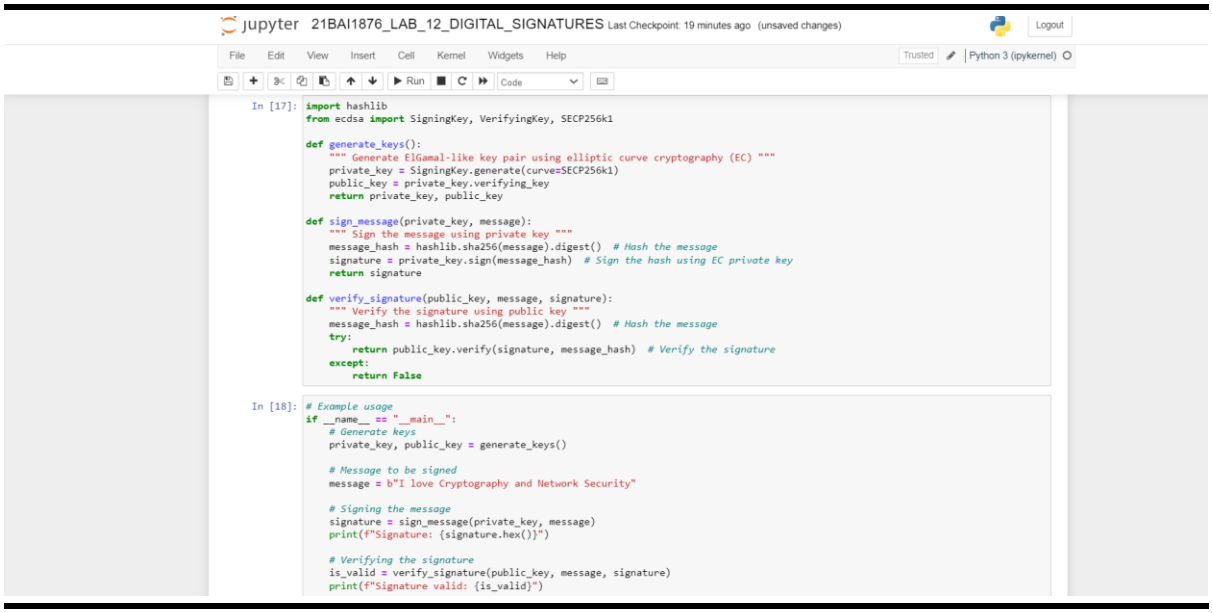
CODE:

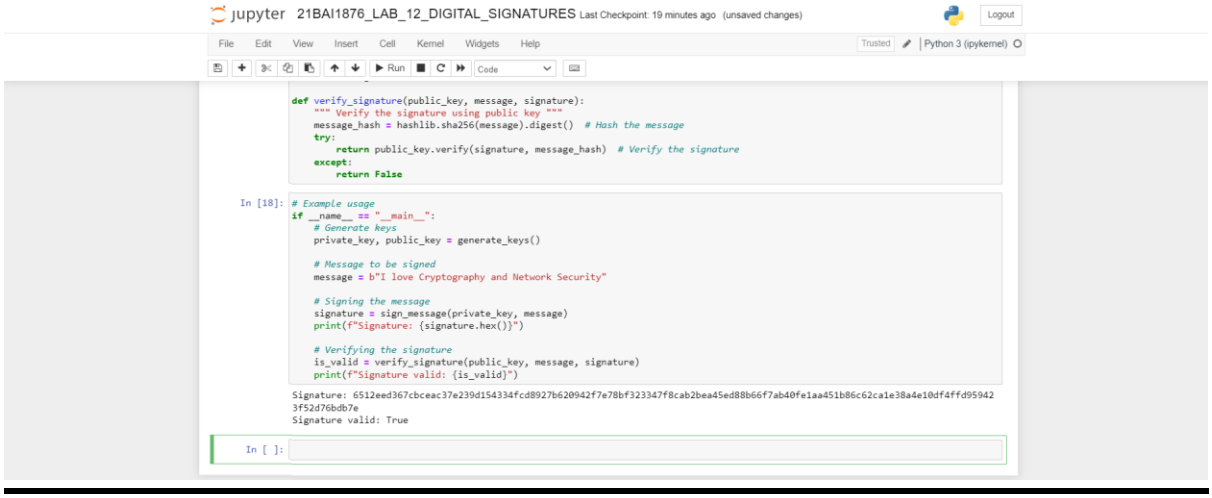
```
import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1


def generate_keys():
    """ Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
    private_key = SigningKey.generate(curve=SECP256k1)
    public_key = private_key.verifying_key
    return private_key, public_key


def sign_message(private_key, message):
    """ Sign the message using private key """
    message_hash = hashlib.sha256(message).digest()  # Hash the message
    signature = private_key.sign(message_hash)  # Sign the hash using EC private key
    return signature


def verify_signature(public_key, message, signature):
    """ Verify the signature using public key """
    message_hash = hashlib.sha256(message).digest()  # Hash the message
    try:
        return public_key.verify(signature, message_hash)  # Verify the signature
    except:
        return False
```

```
# Example usage
if __name__ == "__main__":
    # Generate keys
    private_key, public_key = generate_keys()

    # Message to be signed
    message = b"Elliptic Curve ElGamal Digital Signature
Example"

    # Signing the message
    signature = sign_message(private_key, message)
    print(f"Signature: {signature.hex()}")

    # Verifying the signature
    is_valid = verify_signature(public_key, message,
signature)
    print(f"Signature valid: {is_valid}")
```

OUTPUT:


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 18 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [16]: pip install ecdsa
Collecting ecdsa
Obtaining dependency information for ecdsa from https://files.pythonhosted.org/packages/00/e7/ed3243b30d1bec41675b6394a1daae46349dc2b855cb83be846a5a91823 8/ecdsa-0.19.0-py2.py3-none-any.whl.metadata
Downloading ecdsa-0.19.0-py2.py3-none-any.whl.metadata (29 kB)
Requirement already satisfied: six>=1.9.0 in c:\users\admin\anaconda3\lib\site-packages (from ecdsa) (1.16.0)
Downloading ecdsa-0.19.0-py2.py3-none-any.whl (149 kB)
0.0/149.3 kB ? eta -:--:--
41.0/149.3 kB 991.0 kB/s eta 0:00:01
41.0/149.3 kB 991.0 kB/s eta 0:00:01
71.7/149.3 kB 435.7 kB/s eta 0:00:01
112.6/149.3 kB 544.7 kB/s eta 0:00:01
122.9/149.3 kB 554.9 kB/s eta 0:00:01
122.9/149.3 kB 554.9 kB/s eta 0:00:01
143.4/149.3 kB 425.3 kB/s eta 0:00:01
143.4/149.3 kB 425.3 kB/s eta 0:00:01
149.3/149.3 kB 355.6 kB/s eta 0:00:00
Installing collected packages: ecdsa
Successfully installed ecdsa-0.19.0
Note: you may need to restart the kernel to use updated packages.
In [17]: import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1
def generate_keys():
""" Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
private_key = SigningKey.generate(curve=SECP256k1)
public_key = private_key.verifying_key
return private_key, public_key
def sign_message(private_key, message):
""" Sign the message using private key """
message_hash = hashlib.sha256(message).digest() # Hash the message
signature = private_key.sign(message_hash) # Sign the hash using EC private key
return signature
def verify_signature(public_key, message, signature):


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [17]: import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1
def generate_keys():
""" Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
private_key = SigningKey.generate(curve=SECP256k1)
public_key = private_key.verifying_key
return private_key, public_key
def sign_message(private_key, message):
""" Sign the message using private key """
message_hash = hashlib.sha256(message).digest() # Hash the message
signature = private_key.sign(message_hash) # Sign the hash using EC private key
return signature
def verify_signature(public_key, message, signature):
""" Verify the signature using public key """
message_hash = hashlib.sha256(message).digest() # Hash the message
try:
return public_key.verify(signature, message_hash) # Verify the signature
except:
return False
In [18]: # Example usage
if __name__ == "__main__":
# Generate keys
private_key, public_key = generate_keys()
# Message to be signed
message = b"I love Cryptography and Network Security"
# Signing the message
signature = sign_message(private_key, message)
print(f"Signature: {signature.hex()}")
# Verifying the signature
is_valid = verify_signature(public_key, message, signature)
print(f"Signature valid: {is_valid}")


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
def verify_signature(public_key, message, signature):
""" Verify the signature using public key """
message_hash = hashlib.sha256(message).digest() # Hash the message
try:
return public_key.verify(signature, message_hash) # Verify the signature
except:
return False
In [18]: # Example usage
if __name__ == "__main__":
# Generate keys
private_key, public_key = generate_keys()
# Message to be signed
message = b"I love Cryptography and Network Security"
# Signing the message
signature = sign_message(private_key, message)
print(f"Signature: {signature.hex()}")
# Verifying the signature
is_valid = verify_signature(public_key, message, signature)
print(f"Signature valid: {is_valid}")
Signature: 6512eed367cbceac37e239d154334fcd8927b620942f7e78bf323347f8cab2bea45ed88b66f7ab40fe1aa451b86c62ca1e38a4e10df4ffd959423f52d76bdb7e
Signature valid: True
In [ ]:

LAB 13

CRYPTOGRAPHY AND NETWORK SECURITY

ANALYSE THE DATA PACKET WITH WIRESHARK PACKET TRACER

NAME: OM SUBRATO DEY REG NO.: 21BAI1876

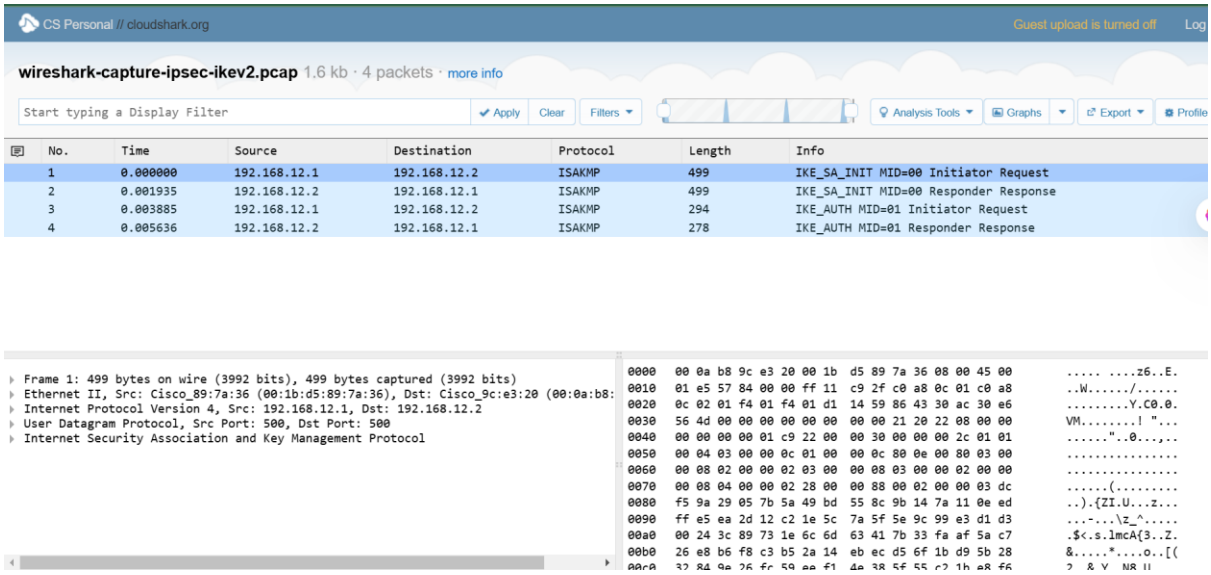
COLAB LINK:

https://colab.research.google.com/drive/1ONarxyrFl19EDAnCQ7bN6gNofQ4Aa-iZ?usp=sharing

We will directly be using a downloaded pcap format file from the link accessible:

https://www.cloudshark.org/captures/767a93d720ad

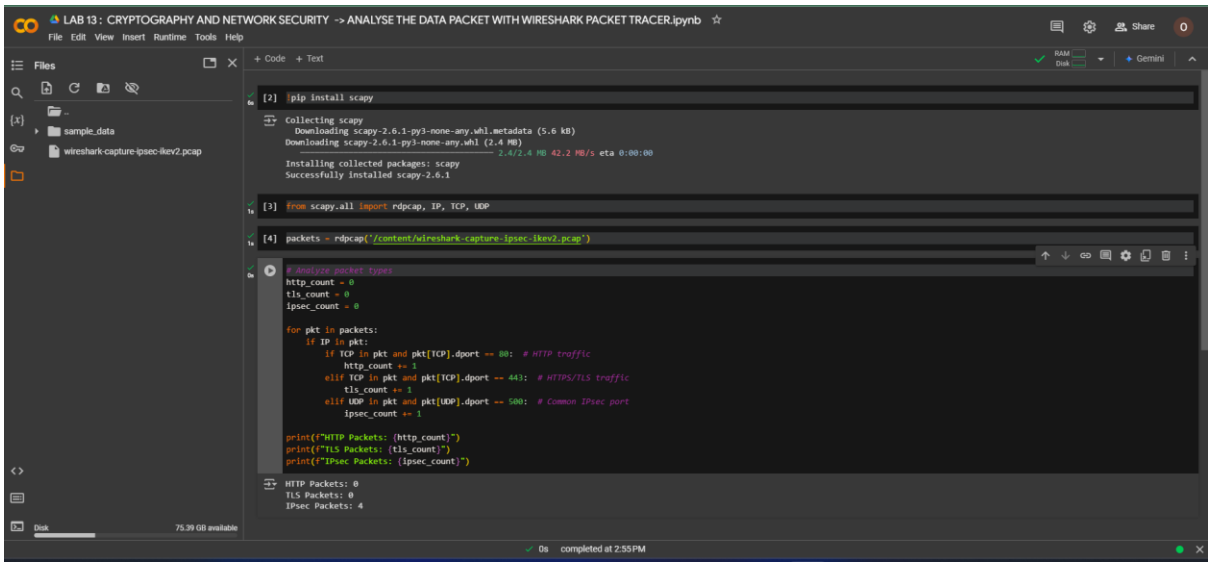
So, above link consists of the wireshark file.



CODE:

```
!pip install scapy

from scapy.all import rdpcap, IP, TCP, UDP

packets = rdpcap('/content/wireshark-capture-ipsec-ikev2.pcap')


# Analyze packet types

http_count = 0

tls_count = 0

ipsec_count = 0


for pkt in packets:

    if IP in pkt:

        if TCP in pkt and pkt[TCP].dport == 80:  # HTTP traffic

            http_count += 1

        elif TCP in pkt and pkt[TCP].dport == 443:  # HTTPS/TLS traffic

            tls_count += 1

        elif UDP in pkt and pkt[UDP].dport == 500:  # Common IPsec port

            ipsec_count += 1


print(f"HTTP Packets: {http_count}")

print(f"TLS Packets: {tls_count}")

print(f"IPsec Packets: {ipsec_count}")
```

OUTPUT:


LAB 13 : CRYPTOGRAPHY AND NETWORK SECURITY -> ANALYSE THE DATA PACKET WITH WIRESHARK PACKET TRACER.ipynb
File Edit View Insert Runtime Tools Help
Share
Files
sample_data
wireshark-capture-ipsec-ikev2.pcap
+ Code + Text
Gemini
[2] pip install scapy
Collecting scapy
Downloading scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
Downloading scapy-2.6.1-py3-none-any.whl (2.4 MB)
2.4/2.4 MB 42.2 MB/s eta 0:00:00
Installing collected packages: scapy
Successfully installed scapy-2.6.1
[3] from scapy.all import rdpcap, IP, TCP, UDP
[4] packets = rdpcap('/content/wireshark-capture-ipsec-ikev2.pcap')
# Analyze packet types
http_count = 0
tls_count = 0
ipsec_count = 0
for pkt in packets:
if IP in pkt:
if TCP in pkt and pkt[TCP].dport == 80: # HTTP traffic
http_count += 1
elif TCP in pkt and pkt[TCP].dport == 443: # HTTPS/TLS traffic
tls_count += 1
elif UDP in pkt and pkt[UDP].dport == 500: # Common IPsec port
ipsec_count += 1
print(f"HTTP Packets: {http_count}")
print(f"TLS Packets: {tls_count}")
print(f"IPsec Packets: {ipsec_count}")
HTTP Packets: 0
TLS Packets: 0
IPsec Packets: 4
Disk 75.39 GB available
0s completed at 2:55PM