



CRYPTOGRAPHY AND NETWORK SECURITY

LAB I : 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')))

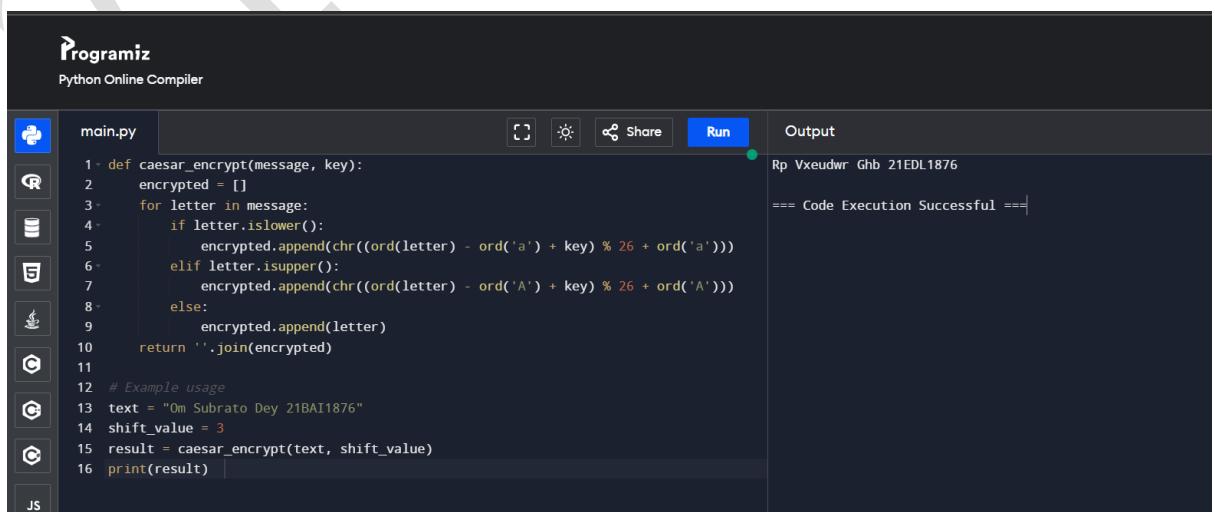
        else:

            encrypted.append(letter)

    return ''.join(encrypted)

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

OUTPUT SCREENSHOT:



The screenshot shows the Python Online Compiler interface on Programiz. The code in main.py is identical to the one above. The output window shows the encrypted text "Rp Vxeudwr Ghb 21EDL1876" and a success message "Code Execution Successful".

```
Programiz
Python Online Compiler

Main.py
Run
Output
Rp Vxeudwr Ghb 21EDL1876
== Code Execution Successful ==

1 - def caesar_encrypt(message, key):
2     encrypted = []
3     for letter in message:
4         if letter.islower():
5             encrypted.append(chr((ord(letter) - ord('a') + key) % 26 + ord('a')))
6         elif letter.isupper():
7             encrypted.append(chr((ord(letter) - ord('A') + key) % 26 + ord('A')))
8         else:
9             encrypted.append(letter)
10    return ''.join(encrypted)
11
12 # Example usage
13 text = "Om Subrato Dey 21BAI1876"
14 shift_value = 3
15 result = caesar_encrypt(text, shift_value)
16 print(result)
```

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 = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'  
    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:

The screenshot shows the Python Online Compiler interface. The code editor contains the provided Python script. The output window shows the execution results, including the generated key square and the encrypted message.

```

Programiz
Python Online Compiler

main.py
1- def forge_key_square(keyword):
2-     # Remove duplicates while preserving order
3-     keyword = ''.join(sorted(set(keyword), key=keyword.index))
4-     # Create the 5x5 matrix
5-     alphabet = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
6-     square = []
7-     for char in keyword + alphabet:
8-         if char not in square and char != 'J': # 'I' and 'J' are treated as the same
letter
9-             square.append(char)
10-    # Split the list into a 5x5 grid
11-    return [square[i:i+5] for i in range(0, 25, 5)]
12-
13- def prepare_message(message):
14-     message = ''.join(filter(str.isalnum, message.upper())).replace('J', 'I')
15-     prepared = []
16-     i = 0
17-     while i < len(message):
18-         first = message[i]
19-         if i + 1 < len(message):
20-             second = message[i + 1]
21-         else:
22-             second = 'X'
23-         if first == second:
24-             prepared.append(first + 'X')
25-             i += 1
26-         else:
27-             prepared.append(first + second)
28-             i += 2

```

Output:

```

Rp Vxeudwr Ghb 21EDL1876
--- Code Execution Successful ---

```



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**

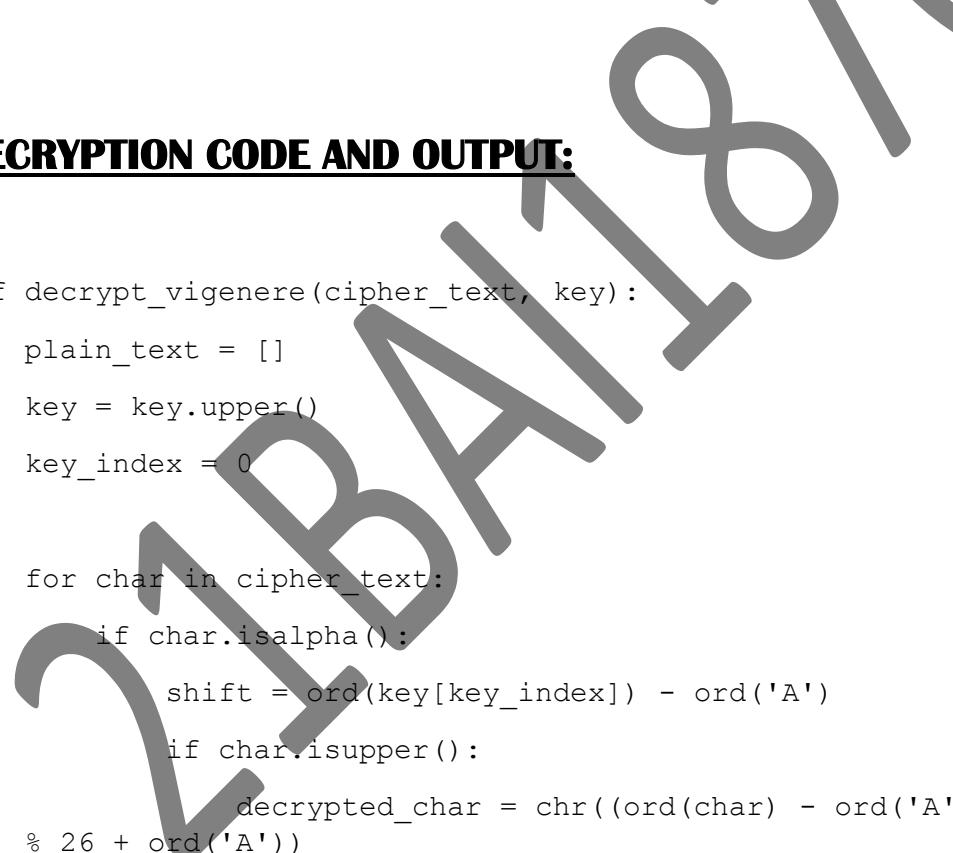
NAME : OM SUBRATO DEY

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}")
```



Programiz

Python Online Compiler

```

main.py | Run | Output
1- def vigenere_encrypt(plain_text, key):
2     encrypted_text = []
3     key = key.upper()
4     key_index = 0
5
6     for char in plain_text:
7         if char.isalpha():
8             shift = ord(key[key_index]) - ord('A')
9             if char.isupper():
10                 encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
11             else:
12                 encrypted_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
13             key_index = (key_index + 1) % len(key)
14         else:
15             encrypted_char = char
16
17         encrypted_text.append(encrypted_char)
18
19     return ''.join(encrypted_text)
20
21 # Example usage
22 plain_text = "OM SUBRATO DEY"
23 key = "KEY"
24 encrypted_text = vigenere_encrypt(plain_text, key)
25 print(f"Encrypted Text: {encrypted_text}")
26

```

Encrypted Text: YQ QEFPKXM NIW
== Code Execution Successful ==

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)

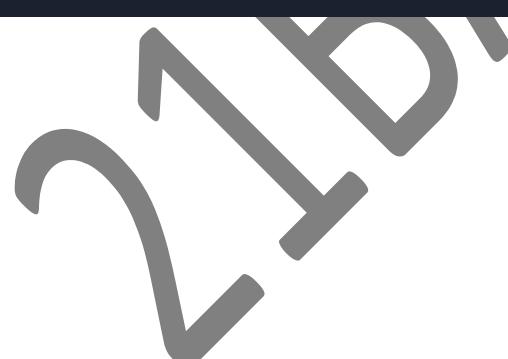
# Example usage

text_to_decrypt = "YQ QEFPKXM NIW"
decryption_key = "KEY"

decrypted_message = decrypt_vigenere(text_to_decrypt,
decryption_key)

print(f"Decrypted Message: {decrypted_message}")

```



Programiz
Python Online Compiler

main.py

```

1 def decrypt_vigenere(cipher_text, key):
2     plain_text = []
3     key = key.upper()
4     key_index = 0
5
6     for char in cipher_text:
7         if char.isalpha():
8             shift = ord(key[key_index]) - ord('A')
9             if char.isupper():
10                 decrypted_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))
11             else:
12                 decrypted_char = chr((ord(char) - ord('a') - shift + 26) % 26 + ord('a'))
13             key_index = (key_index + 1) % len(key)
14         else:
15             decrypted_char = char
16
17         plain_text.append(decrypted_char)
18
19     return ''.join(plain_text)
20
21 # Example usage
22 text_to_decrypt = "YQ QEFPKXM NIW"
23 decryption_key = "KEY"
24 decrypted_message = decrypt_vigenere(text_to_decrypt, decryption_key)
25 print(f"Decrypted Message: {decrypted_message}")
26

```

Output

```

Decrypted Message: OM SUBRATO DEY
*** Code Execution Successful ***

```

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:

Programiz

Python Online Compiler

The screenshot shows a Python code editor with a dark theme. On the left, there's a sidebar with icons for various programming languages: Python, C, C++, Java, JavaScript, Go, PHP, and Swift. The main area has a tab labeled "main.py". The code is as follows:

```
1 def encrypt_text(input_text, key):
2     rails = [[] for _ in range(key)]
3     rail_index = 0
4     direction = 1
5
6     for char in input_text:
7         rails[rail_index].append(char)
8         rail_index += direction
9
10    if rail_index == 0 or rail_index == key - 1:
11        direction *= -1
12
13    cipher = ''
14    for rail in rails:
15        cipher += ''.join(rail)
16
17    return cipher
18
19 # Example usage
20 plain_text = "OM SUBRATO DEY"
21 key = 3
22 encrypted_text = encrypt_text(plain_text, key)
23 print(f"Encrypted Text: {encrypted_text}")
```

The output window on the right shows the result of running the code: "Encrypted Text: OUTEMSBAODY R" and "== Code Execution Successful ==".

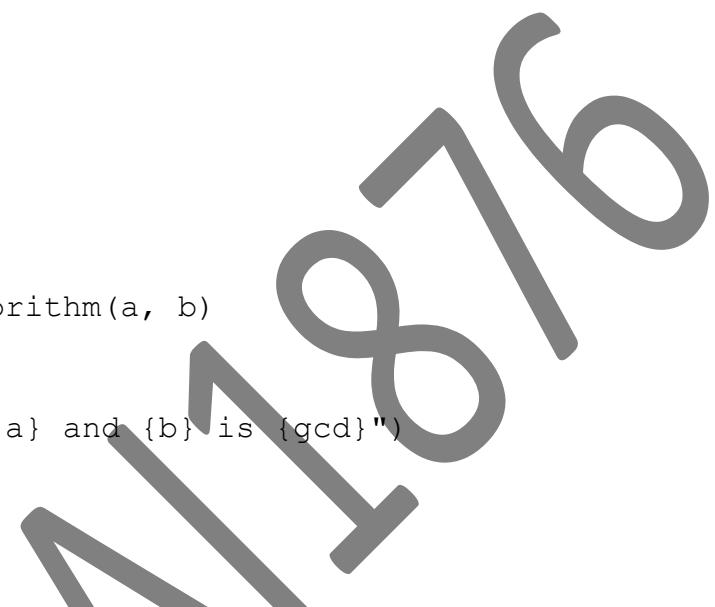
ZIBALI81

3. EUCLEDIAN 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:



The image shows a screenshot of the Programiz Python Online Compiler. The code in the editor is identical to the one above. The output panel shows the execution results: "The GCD of 212 and 111876 is 4" and "==== Code Execution Successful ===". The interface includes tabs for Python, C, C++, JS, and PHP, and buttons for Run and Share.

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  $x \equiv a[i] \pmod{n[i]}$  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  $x \equiv {x} \pmod{{n[0]} * {n[1]} * {n[2]}}$ ")

```

OUTPUT:

```

Programiz
Python Online Compiler

main.py
1 # OM SUBRATO DEY - 21BBA11876
2 def extended_euclidean(a, b):
3     """
4         Returns a tuple (g, x, y) such that g = gcd(a, b) and ax + by = g
5     """
6     if a == 0:
7         return b, 0, 1
8     else:
9         g, x1, y1 = extended_euclidean(b % a, a)
10        x = y1 - (b // a) * x1
11        y = x1
12    return g, x, y
13
14 def modular_inverse(a, m):
15     """
16         Returns the modular inverse of a under modulo m, if it exists
17     """
18     g, x, _ = extended_euclidean(a, m)
19     if g != 1:
20         raise ValueError(f"Modular inverse does not exist for {a} and {m}")
21     else:
22         return x % m
23
24 def chinese_remainder_theorem(n, a):
25     """
26         Solves the system of simultaneous congruences using the Chinese Remainder Theorem.
27         n: List of moduli
28         a: List of remainders
29         Returns the smallest x such that  $x \equiv a[i] \pmod{n[i]}$  for all i
30     """
31     product = 1
32     for ni in n:
33         product *= ni
34
35     result = 0
36     for ni, ai in zip(n, a):
37         pi = product // ni
38         mi = modular_inverse(pi, ni)
39         result += ei * mi * pi
40
41     return result % product
42
43 # Example usage:
44 n = [1, 8, 7]
45 a = [6, 2, 1]
46 x = chinese_remainder_theorem(n, a)
47 print(f"The solution to the system of congruences is  $x \equiv {x} \pmod{{n[0]} * {n[1]} * {n[2]}}$ ")

```

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:

Programiz

Python Online Compiler

main.py

```
1 # OM SUBRATO DEY - 21BATA1876
2 def extended_gcd(a, b):
3     # Base case
4     if a == 0:
5         return b, 0, 1
6
7     # Recursively call the function
8     gcd, x1, y1 = extended_gcd(b % a, a)
9
10    # Update x and y using results of recursive call
11    x = y1 - (b // a) * x1
12    y = x1
13
14    return gcd, x, y
15
16 # Example usage
17 a = 212021
18 b = 1876
19 gcd, x, y = extended_gcd(a, b)
20 print(f"GCD of {a} and {b} is {gcd}")
21 print(f"Coefficients x and y are {x} and {y}, respectively")
22
```

Run

Output

```
GCD of 212021 and 1876 is 1
Coefficients x and y are -739 and 83520, respectively
==== Code Execution Successful ===
```



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/permuation (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

main.py

```
1 # OM SUBRATO DEY - 21BBA1876
2 # SDES Implementation in Python
3
4 # Permutation functions:
5 def permute(bits, table):
6     return [bits[i-1] for i in table]
7
8 # Initial permutation (IP)
9 IP = [2,1,2,1,1,8,7,6]
10 # Inverse initial permutation (IP^-1)
11 IP_INV = [4, 1, 3, 5, 7, 2, 8, 6]
12
13 # Expansion/permutation (E/P)
14 EP = [4, 1, 2, 3, 2, 3, 4, 1]
15 # Permutation function (P4)
16 P4 = [2, 4, 3, 1]
17
18 # S-Boxes
19 S0 = [
20     [1, 0, 3, 2],
21     [3, 2, 1, 0],
22     [0, 2, 1, 3],
23     [3, 1, 3, 2]
24 ]
25
26 S1 = [
27     [0, 1, 2, 3],
28     [2, 0, 1, 3],
29     [3, 0, 1, 0],
30     [2, 1, 0, 3]
31 ]
```

Output

```
Cipher Text: 01001101
Decrypted Text: 10100010
== Code Execution Successful ==
```

Programiz PRO > Clear

Programiz
Python Online Compiler

main.py

```
30     [2, 1, 0, 3]
31 ]
32
33 def sbox_lookup(sbox, row, col):
34     return sbox[row][col]
35
36 def fk(bits, key):
37     left, right = bits[:4], bits[4:]
38     expanded_permuted = permute(right, EP)
39     xor_result = [a ^ b for a, b in zip(expanded_permuted, key)]
40
41     left_half = xor_result[:4]
42     right_half = xor_result[4:]
43
44     row0 = (right_half[0] << 1) + left_half[3]
45     col0 = (left_half[1] << 1) + left_half[2]
46     s0_output = sbox_lookup(S0, row0, col0)
47
48     row1 = (right_half[0] << 1) + right_half[3]
49     col1 = (right_half[1] << 1) + right_half[2]
50     s1_output = sbox_lookup(S1, row1, col1)
51
52     sbox_output = [
53         (s0_output & 0b10) >> 1, s0_output & 0b01,
54         (s1_output & 0b10) >> 1, s1_output & 0b01
55     ]
56
57     permuted_sbox_output = permute(sbox_output, P4)
58     left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]
59
60     return left_result + right
61
```

Output

```
Cipher Text: 01001101
Decrypted Text: 10100010
== Code Execution Successful ==
```

Programiz PRO > Clear

Programiz
Python Online Compiler

main.py

```

56     permuted_sbox_output = permute(sbox_output, P4)
57     left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]
58
59     return left_result + right
60
61 def switch(bits):
62     return bits[4:] + bits[:4]
63
64 # Key generation
65 def generate_keys(key):
66     P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
67     P8 = [6, 3, 7, 4, 8, 5, 10, 9]
68
69     permuted_key = permute(key, P10)
70     left, right = permuted_key[:5], permuted_key[5:]
71
72     JS
73     left = left[:] + left[:1]
74     right = right[:] + right[:1]
75     k1 = permute(left + right, P8)
76
77     PHP
78     left = left[2:] + left[:2]
79     right = right[2:] + right[:2]
80     k2 = permute(left + right, P8)
81
82     return k1, k2
83
84 # Encryption and decryption
85 def sdes_encrypt(plain_text, key):
86     k1, k2 = generate_keys(key)
87     initial_permuted = permute(plain_text, IP)
88     round1_result = fk(initial_permuted, k1)
89     switched = switch(round1_result)
90     round2_result = fk(swapped, k2)
91     cipher_text = permute(round2_result, IP_INV)
92
93     return cipher_text
94
95 def sdes_decrypt(cipher_text, key):
96     k1, k2 = generate_keys(key)
97     initial_permuted = permute(cipher_text, IP)
98     round1_result = fk(initial_permuted, k2)
99     swapped = switch(round1_result)
100    round2_result = fk(swapped, k1)
101    plain_text = permute(round2_result, IP_INV)
102
103    return plain_text
104
105 # Helper functions to convert between binary strings and lists
106 def string_to_bits(s):
107     return [int(b) for b in s]
108
109 # Example usage
110 plain_text = "10101010"
111 key = "1010000010"
112
113 plain_bits = string_to_bits(plain_text)

```

Cipher Text: 01001101
Decrypted Text: 10100010
== Code Execution Successful ==

Programiz
Python Online Compiler

main.py

```

82
83 # Encryption and decryption
84 def sdes_encrypt(plain_text, key):
85     k1, k2 = generate_keys(key)
86     initial_permuted = permute(plain_text, IP)
87     round1_result = fk(initial_permuted, k1)
88     swapped = switch(round1_result)
89     round2_result = fk(swapped, k2)
90     cipher_text = permute(round2_result, IP_INV)
91
92     return cipher_text
93
94 def sdes_decrypt(cipher_text, key):
95     k1, k2 = generate_keys(key)
96     initial_permuted = permute(cipher_text, IP)
97     round1_result = fk(initial_permuted, k2)
98     swapped = switch(round1_result)
99     round2_result = fk(swapped, k1)
100    plain_text = permute(round2_result, IP_INV)
101
102    return plain_text
103
104 # Helper functions to convert between binary strings and lists
105 def string_to_bits(s):
106     return [int(b) for b in s]
107
108 def bits_to_string(bits):
109     return ''.join(str(b) for b in bits)
110
111 # Example usage
112 plain_text = "10101010"
113 key = "1010000010"
114
115 plain_bits = string_to_bits(plain_text)
116 key_bits = string_to_bits(key)
117
118 cipher_bits = sdes_encrypt(plain_bits, key_bits)
119 cipher_text = bits_to_string(cipher_bits)
120
121 print(f"Cipher Text: {cipher_text}")
122
123 decrypted_bits = sdes_decrypt(cipher_bits, key_bits)
124 decrypted_text = bits_to_string(decrypted_bits)
125
126 print(f"Decrypted Text: {decrypted_text}")
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
786
786
787
788
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
821
822
822
823
823
824
825
825
826
826
827
827
828
828
829
829
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
```

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

```
+ Code + Text
```

[2] 1 !pip install pycryptodome

{x} 2 Collecting pycryptodome
3 Downloading pycryptodome-3.20.0-cp35abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
4 Downloading pycryptodome-3.20.0-cp35abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
5 2.1/2.1 MB 28.5 MB/s eta 0:00:00
6 Installing collected packages: pycryptodome
7 Successfully installed pycryptodome-3.20.0

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

3 def des_encrypt(key, plaintext):
4 # Create a DES cipher object
5 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC)
6 # Pad the plaintext to be a multiple of 8 bytes
7 padded_text = pad(plaintext.encode('utf-8'), DES.block_size)
8 # Encrypt the padded plaintext
9 ciphertext = cipher.encrypt(padded_text)
10 return cipher.iv, ciphertext # Return the IV and ciphertext

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

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

```
+ Code + Text
```

File Edit View Insert Runtime Tools Help All changes saved

0s 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 def des_decrypt(key, iv, ciphertext):
10 # Create a DES cipher object with the same IV
11 cipher = DES.new(key.encode('utf-8'), DES.MODE_CBC, iv)
12 # Decrypt the ciphertext
13 padded_plaintext = cipher.decrypt(ciphertext)
14 # Unpad the plaintext
15 return unpad(padded_plaintext, DES.block_size).decode('utf-8')

16
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)

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

RAM Disk

```
[2] 1 !pip install pycryptodome
Collecting pycryptodome
  Downloading pycryptodome-3.20.0-cp35abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
  Downloading pycryptodome-3.20.0-cp35abi3-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

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
```

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

RAM Disk

```
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)

Ciphertext (hex): ed534b40727dd8589c22e475d6f5ffba05071ff81711cf8efcb900ae2b0cb3b
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:

The screenshot shows a Python code editor and its output window. The code, named main.py, implements the Diffie-Hellman key exchange protocol. It defines prime numbers p and g, generates private keys for Alice and Bob, computes public keys, exchanges them, and then calculates and prints the shared secrets. An assert statement ensures the two shared secrets are equal.

```
main.py
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("Alice's Public Key: {A_public}")
22 print("Bob's Public Key: {B_public}")
23 print("Alice's Shared Secret: {alice_shared_secret}")
24 print("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
```

Output

```
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:

```

Programiz
Python Online Compiler
Programiz PRO

main.py
1 Jasleen sends: Hello, Sparsh! This is Jasleen from Washington D.C.
2     *** Intercepting message from Jasleen ***
3     *** Message modified to: Hello, Sparsh! This is Jasleen from Washington D.C. [Tampered by Attacker]
4
5 Sparsh receives: Hello, Sparsh! This is Jasleen from Washington D.C. [Tampered by Attacker]
6
7 --- Code Execution Successful ---
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

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

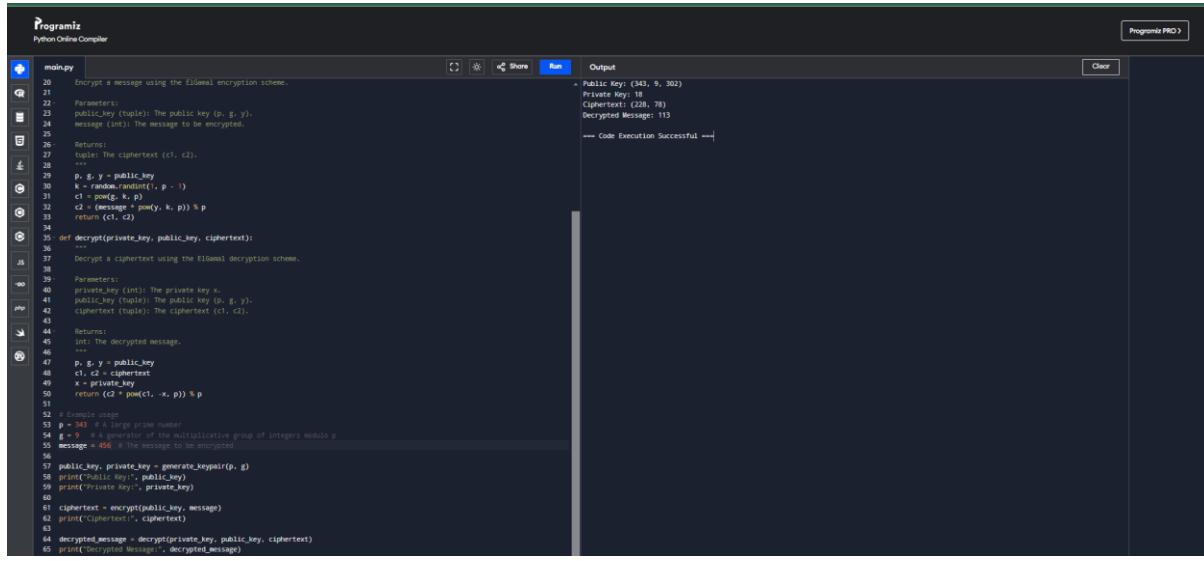
public_key, private_key = generate_keypair(p, g)
print("Public Key:", public_key)
print("Private Key:", private_key)

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

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

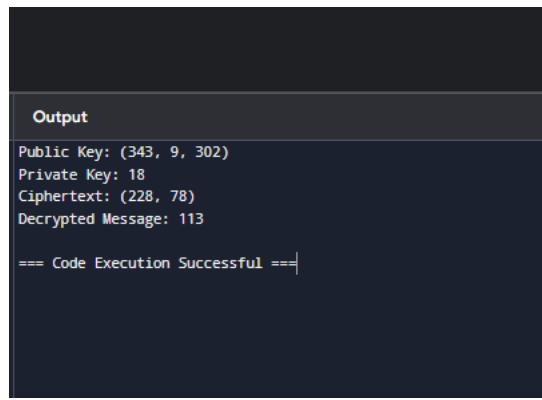
```

OUTPUT:



The screenshot shows a Python code editor with the file name 'main.py'. The code implements the ElGamal encryption scheme. It defines two functions: 'encrypt(private_key, public_key)' and 'decrypt(private_key, public_key, ciphertext)'. The 'encrypt' function takes a private key (p, g, y) and a public key (p, g, y). It generates a random k, calculates c1 = pow(g, k, p) and c2 = (message * pow(y, k, p)) % p, and returns the ciphertext (c1, c2). The 'decrypt' function takes a private key (p, g, y) and a public key (p, g, y), along with a ciphertext (c1, c2). It calculates x = private_key and deciphers the message as (c2 * pow(c1, -x, p)) % p. The output window shows the execution results: Public Key: (343, 9, 302), Private Key: 18, Ciphertext: (228, 78), Decrypted Message: 113, followed by a success message.

```
20 Encrypt a message using the ElGamal encryption scheme.
21
22 Parameters:
23 public_key (tuple): The public key (p, g, y).
24 message (int): The message to be encrypted.
25
26 Returns:
27 tuple: The ciphertext (c1, c2).
28
29 p, g, y = public_key
30 k = random.randint(1, p - 1)
31 c1 = pow(g, k, p)
32 c2 = (message * pow(y, k, p)) % p
33 return (c1, c2)
34
35 def decrypt(private_key, public_key, ciphertext):
36     Decrypt a ciphertext using the ElGamal decryption scheme.
37
38 Parameters:
39 private_key (int): The private key x.
40 public_key (tuple): The public key (p, g, y).
41 ciphertext (tuple): The ciphertext (c1, c2).
42
43 Returns:
44 int: The decrypted message.
45
46 p, g, y = public_key
47 c1, c2 = ciphertext
48 x = private_key
49 return (c2 * pow(c1, -x, p)) % p
50
51 # Example usage
52 p = 343 # A large prime number
53 g = 9 # A generator of the multiplicative group of integers modulo p
54 message = 456 # The message to be encrypted
55
56 public_key, private_key = generate_keys(p, g)
57 print("Public Key:", public_key)
58 print("Private Key:", private_key)
59
60 ciphertext = encrypt(public_key, message)
61 print("Ciphertext:", ciphertext)
62 decrypted_message = decrypt(private_key, public_key, ciphertext)
63 print("Decrypted Message:", decrypted_message)
```



The screenshot shows the terminal output of the 'main.py' script. It displays the generated public key (343, 9, 302), the private key (18), the ciphertext (228, 78), and the decrypted message (113). The output concludes with a success message.

```
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:

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains Python code for simulating user registration and login. It includes functions for generating MD5 hashes, registering users by storing their username and hashed password in a dictionary, and attempting to log in by comparing the entered password's hash against the stored one. The code also handles cases where a user might already exist or enter an incorrect password.

```
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:
    """Attempt 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("Current user database (hashed passwords):")
print(users_db)

# Attempting logins...
User 'alice' registered successfully with hashed password: 482c811da5d5b4bc6d497ffa38a91e38
User 'bob' registered successfully with hashed password: 0e03da69f4a66aa3cf6f787f75cd8f7f

Attempting logins...
Login successful for user 'alice'!
Incorrect password.
User 'charlie' does not exist!

Current user database (hashed passwords):
{'alice': '482c811da5d5b4bc6d497ffa38a91e38', 'bob': '0e03da69f4a66aa3cf6f787f75cd8f7f'}
```

The screenshot shows the execution results of the code cell from the previous screenshot. The output area displays the generated MD5 hashes for the registered users ('alice' and 'bob'), the successful login attempt for 'alice', and the failure for 'bob' and 'charlie'. It also shows the current state of the user database.

```
Colab paid products - Cancel contracts here
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:

The image shows two screenshots of a Jupyter Notebook interface. Both screenshots have a dark theme and are titled "CRYPTOGRAPHY AND NETWORK SECURITY LAB 11.ipynb".

Screenshot 1 (Top): This screenshot displays the implementation of a SHA-512 hasher. The code defines a class `SHA512Hasher` with methods for updating the hash with chunks of data, getting the current hash as a hex digest, hashing a single string, and hashing a file. A note at the bottom indicates that the file path is hashed in chunks for large files.

```
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 to 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=8096):
        """
        Hash the contents of a file in chunks (useful for large files).
        """
        # Hash the contents of a file in chunks (useful for large files).
```

Screenshot 2 (Bottom): This screenshot shows how to use the `SHA512Hasher` class. It demonstrates hashing a single string, reading a file in chunks, and combining the results. The code also shows how to update the hash with multiple chunks.

```
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': 07e547d958bf6a73f73fhac0435ed76951218fb7d0c8d788a309d785436bbb642e93a252a054f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6
File not found: example.txt
SHA-512 hash of combined chunks 'Hello, ' and 'world!': c1527cd893c124773d811911970c8fe6e857d6df5dc9226bd8a166614c0cd963a4ddeab2b94bb7d36021ef9d805d5cea294a82dd49a0bb269f51f6e7a57ff9421
```

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)

File Edit View Insert Cell Kernel Widgets Help

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.

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):

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)

In [5]: # 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)

In [16]: pip install ecdsa

```
Collecting ecdsa
  Obtaining dependency information for ecdsa from https://files.pythonhosted.org/packages/00/e7/ed3243b30d1bec41675b6394a1daae46349dc2b855cb83be846a5918238/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)
Downloaded ecdsa-0.19.0-py2.py3-none-any.whl (149 kB)
----- 0.0/149.3 kB 0.0/s eta: 0:00:01
----- 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 444.7 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):
    """ 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("Signature: " + str(signature.hex()))

 # Verifying the signature
 is_valid = verify_signature(public_key, message, signature)
 print("Signature valid: " + str(is_valid))

```
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("Signature: " + str(signature.hex()))

    # Verifying the signature
    is_valid = verify_signature(public_key, message, signature)
    print("Signature valid: " + str(is_valid))

Signature: 6512eed367cbeac37e239d154334fcd8927b620942f78bf323347f8cab2bea45ed8b866f7ab40fe1aa451b86c62c4e38a4e10df4ff95942
3f52d76db7e
Signature valid: True
```

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.

The screenshot shows a Wireshark capture window titled "wireshark-capture-ipsec-ikev2.pcap" containing 4 packets. The table lists the following information:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.12.1	192.168.12.2	ISAKMP	499	IKE_SA_INIT MID=00 Initiator Request
2	0.001935	192.168.12.2	192.168.12.1	ISAKMP	499	IKE_SA_INIT MID=00 Responder Response
3	0.003885	192.168.12.1	192.168.12.2	ISAKMP	294	IKE_AUTH MID=01 Initiator Request
4	0.005636	192.168.12.2	192.168.12.1	ISAKMP	278	IKE_AUTH MID=01 Responder Response

Below the table, the packet details and bytes panes show the raw hex and ASCII data for each packet. For example, the first packet (IKE_SA_INIT) has a length of 499 bytes and contains the following hex data: 00 0a b8 9c e3 20 00 1b d5 89 7a 36 08 00 45 00 The bytes pane shows the raw binary data for the selected packet.

CODE:

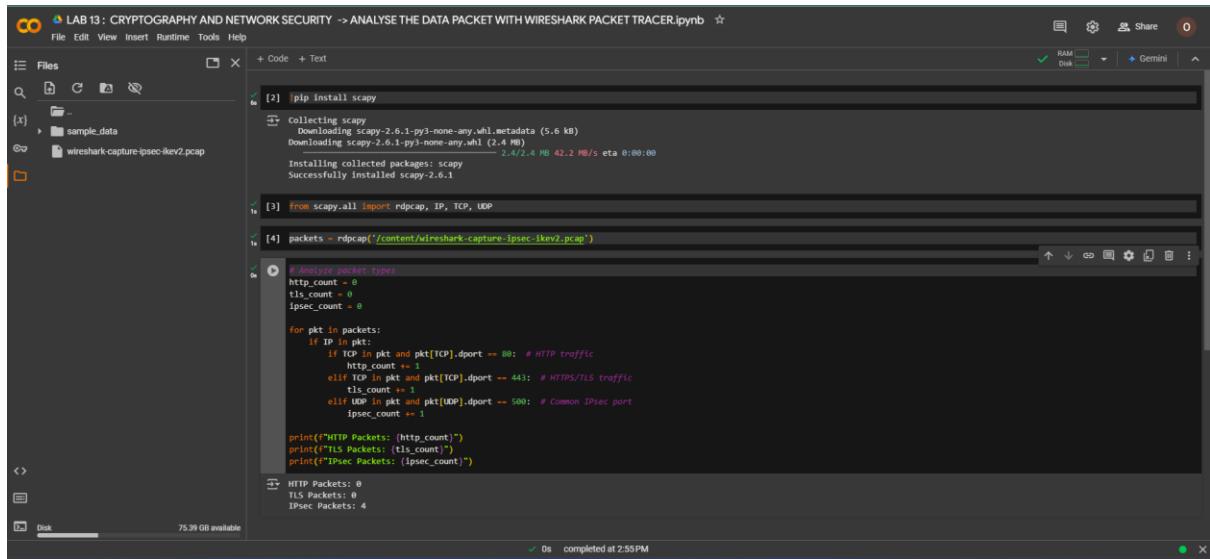
```
!pip install scapy
from scapy.all import rdpcap, IP, TCP, UDP
packets = rdpcap('/content/wireshark-capture-ipsec-
ikev2.pcap')

# 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:



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** LAB 13 : CRYPTOGRAPHY AND NETWORK SECURITY -> ANALYSE THE DATA PACKET WITH WIRESHARK PACKET TRACER.ipynb
- File Explorer:** Shows a directory structure with 'sample_data' and 'wireshark-capture-ipsec-ikev2.pcap'.
- Code Cell 1:** pip install scapy
- Code Cell 2:** 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 kB)
Installing collected packages: scapy
Successfully installed scapy-2.6.1
- Code Cell 3:** from scapy.all import rdpcap, IP, TCP, UDP
- Code Cell 4:** packets = rdpcap('/content/wireshark-capture-ipsec-ikev2.pcap')
- Code Cell 5:** # 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("HTTP Packets: (http_count)")
print("TLS Packets: (tls_count)")
print("IPsec Packets: (ipsec_count)")
- Output:** HTTP Packets: 0
TLS Packets: 0
IPsec Packets: 0
- System Status:** RAM 75.39 GB available
- Bottom Status:** 0s completed at 2:55PM