1. AIM: to implement the Caesar cipher using the substitution technique

Algorithm:

- **Input**: Read `text`, `shift`.
- **Initialize**: Set `result` to an empty string.
- **For each character** `char` in `text`:
    - If `char` is alphabetic:
        - Set `shift_base` to 65 (if uppercase) or 97 (if lowercase).
        - Calculate `new_char` as:
          new_char=chr((ord(char)−shift_base+shift)mod 26+shift_base
        - Append `new_char` to `result`.
    - Else, append `char` to `result`.
- **Output**: Return result

Code:

cc.py - C:\Users\raned\OneDrive\Desktop\TE IT\SEM 5\Security Lab\Practical exam practice\cc.py (3.12.4)

File Edit Format Run Options Window Help

```
def ceaser_cipher(text,shift):
        result=""
        for char in text:
                if char.isalpha():
                        shift_base=65 if char.isupper() else 97
                        result += chr((ord(char)-shift_base+shift)%26 +shift_base)
                else:
                        result+=char
        return result
text= "Hello, World"
shift= 3
cipher_text = ceaser_cipher(text,shift)
print("Cipher text", cipher_text)
```

Output:

```
============================
Cipher text Khoor, Zruog
```

2. AIM: to implement the Playfair cipher using Substitution technique

Algorithm for Playfair Cipher

1. **Input**:
   - Read plain_text from the user.
   - Read keyword from the user.
2. **Create 5x5 Matrix**:
   - Replace 'J' with 'I' in the keyword.
   - Remove duplicates from the keyword.
   - Arrange the keyword in a 5x5 matrix.
   - Fill remaining cells with the missed letters in alphabetical order (A-Z, treating 'I' and 'J' as the same).
3. **Prepare Plain Text**:
   - Replace 'J' with 'I' in plain_text.
   - Remove spaces and special characters.
   - If the length of plain_text is odd, append 'X' to make it even.
   - Group the plain_text into pairs of characters.
4. **Encrypt Plain Text**:
   - For each pair of characters:
     - Find their positions in the 5x5 matrix.
     - If both characters are in the same row, replace them with the letters to their immediate right (wrap around if needed).
     - If both characters are in the same column, replace them with the letters immediately below (wrap around if needed).
     - If they form a rectangle, replace them with the letters in their respective rows and the columns of the other character.
5. **Output**:
   - Display the resulting cipher_text.

Code:

pf.py - C:/Users/raned/OneDrive/Desktop/TE IT/SEM 5/Security Lab/Practical exam practice/pf.py (3.12.4)

File Edit Format Run Options Window Help

```python
def create_matrix(key):
    key = key.upper().replace('J', 'I')  # Replace 'J' with 'I'
    key = ''.join(dict.fromkeys(key))    # Remove duplicates while preserving order
    alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"  # Alphabet excluding 'J'

    # Create a 5x5 matrix
    matrix = [char for char in key if char in alphabet] + [char for char in alphabet if char not in key]
    return [matrix[i:i + 5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    for r, row in enumerate(matrix):
        if char in row:
            return r, row.index(char)

def playfair_cipher(text, key):
    matrix = create_matrix(key)  # Create the Playfair matrix
    text = text.upper().replace("J", "I").replace(" ", "")  # Prepare text

    # Add 'X' if the text length is odd
    if len(text) % 2 != 0:
        text += 'X'

    cipher = ''
    for i in range(0, len(text), 2):
        a, b = text[i], text[i + 1]  # Get pairs of characters
        row_a, col_a = find_position(matrix, a)
        row_b, col_b = find_position(matrix, b)

        # Apply Playfair rules
        if row_a == row_b:  # Same row
            cipher += matrix[row_a][(col_a + 1) % 5] + matrix[row_b][(col_b + 1) % 5]
        elif col_a == col_b:  # Same column
            cipher += matrix[(row_a + 1) % 5][col_a] + matrix[(row_b + 1) % 5][col_b]
        else:  # Rectangle case
            cipher += matrix[row_a][col_b] + matrix[row_b][col_a]

    return cipher

# Example usage
key = "playfair example"
text = "hide the gold"
print("Ciphered Text:", playfair_cipher(text, key))
```

Output:

```
===============================
Ciphered Text: BMODZBXDNAGE
```

3. AIM: to implement the RSA Algorithm

Algorithm for RSA Encryption and Decryption

Step 1: **Input two prime numbers p and q.**

- Let p = 61 and q = 53.

Step 2: **Calculate n.**

- n = p * q.

Step 3: **Calculate Euler's Totient function phi.**

- phi = (p - 1) * (q - 1).

Step 4: **Choose a public key exponent e.**

- Choose e such that 1 < e < phi and gcd(e, phi) = 1.
- In the given code, e = 17.

Step 5: **Calculate the private key d.**

- Compute d such that (e * d) % phi = 1 (modular inverse of e mod phi).

Step 6: **Public and private key generation.**

- Public key: (e, n).
- Private key: (d, n).

Step 7: **Encrypt the message.**

- Convert each character of the message to its ASCII value.
- For each character, calculate the ciphertext as:
    - cipher = (ASCII_value ^ e) % n.

Step 8: **Decrypt the ciphertext.**

- For each encrypted character, calculate the original value as:
    - original_value = (cipher ^ d) % n.
- Convert the decrypted ASCII value back to the original character.

Step 9: **Display the encrypted and decrypted messages.**

Code:

rsa.py - C:/Users/raned/OneDrive/Desktop/TE IT/SEM 5/Security Lab/Practical exam practice/

File Edit Format Run Options Window Help

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
def generate_keys():
    p, q = 61, 53
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 17
    d = mod_inverse(e, phi)
    return (e, n), (d, n)

def encrypt(message, public_key):
    e, n = public_key
    return [(ord(char) ** e) % n for char in message]

def decrypt(cipher, private_key):
    d, n = private_key
    return ''.join([chr((char ** d) % n) for char in cipher])

# Example usage
public_key, private_key = generate_keys()
message = "HELLO"
encrypted = encrypt(message, public_key)
decrypted = decrypt(encrypted, private_key)

print("Original:", message)
print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
========================================
Original: HELLO
Encrypted: [3000, 28, 2726, 2726, 1307]
Decrypted: HELLO
```

4. AIM: to implement the Diffie-Hellman Key Exchange algorithm

Algorithm for Diffie-Hellman Key Exchange

1. **Choose a prime number p and a base g:**
   - Let p = 29 (prime number).
   - Let g = 5 (base or generator).
2. **Generate private keys for both users:**
   - For **User A**, generate a random private key private_key_a such that 1 <= private_key_a < p.
   - For **User B**, generate a random private key private_key_b such that 1 <= private_key_b < p.
3. **Compute public keys for both users:**
   - User A computes their public key as public_key_a = (g^private_key_a) % p.
   - User B computes their public key as public_key_b = (g^private_key_b) % p.
4. **Exchange public keys between User A and User B:**
   - Both users share their public keys over the communication channel.
5. **Compute the shared secret:**
   - **User A** computes the shared secret as shared_secret_a = (public_key_b^private_key_a) % p.
   - **User B** computes the shared secret as shared_secret_b = (public_key_a^private_key_b) % p.
6. **Both users have the same shared secret:**
   - shared_secret_a and shared_secret_b should be equal, as they represent the same shared key.

Code:

Deffi.py - C:/Users/raned/OneDrive/Desktop/TE IT/SEM 5/Security Lab/Practical exam practice/Deffi.py (3.12.4)

File Edit Format Run Options Window Help

```python
import random
def generate_private_key(prime):
    return random.randint(1, prime - 1)

def compute_public_key(base, private_key, prime):
    return (base ** private_key) % prime

def compute_shared_secret(public_key, private_key, prime):
    return (public_key ** private_key) % prime

prime = 29
base = 5
private_key_a = generate_private_key(prime)
public_key_a = compute_public_key(base, private_key_a, prime)

private_key_b = generate_private_key(prime)
public_key_b = compute_public_key(base, private_key_b, prime)

shared_secret_a = compute_shared_secret(public_key_b, private_key_a, prime)
shared_secret_b = compute_shared_secret(public_key_a, private_key_b, prime)

print("User A's private key:", private_key_a)
print("User A's public key:", public_key_a)
print("User B's private key:", private_key_b)
print("User B's public key:", public_key_b)
print("Shared secret (A):", shared_secret_a)
print("Shared secret (B):", shared_secret_b)
```

Output:

```
============================
User A's private key: 12
User A's public key: 7
User B's private key: 21
User B's public key: 28
Shared secret (A): 1
Shared secret (B): 1
```

5. AIM: to implement the md5 Algorithm

Algorithm for MD5 Hashing

1. **Start**.
2. **Input**: Take the string `string = "Darshan"`.
3. **Encode** the string into bytes using `encode()` method: `encoded = string.encode()`.
4. **Apply MD5 Hashing**:
   - Use `hashlib.md5()` to compute the MD5 hash of the encoded string.
   - Store the result in `result`.
5. **Convert to Hexadecimal**:
   - Call `result.hexdigest()` to get the hexadecimal equivalent of the hash.
6. **Output**:
   - Display the hash object `result`.
   - Display the **hexadecimal equivalent** of the hash.
7. **End**.

Code:

md5.py - C:\Users\raned\OneDrive\Desktop\TE IT\SEM 5\Security Lab\Practi

File Edit Format Run Options Window Help

```python
import hashlib

string = "Darshan"
encoded=string.encode()
result=hashlib.md5(encoded)
print("Hash value", result)
print("Hexadecimal equvalent:",result.hexdigest())
```

Output:

```
============================================ RESTART: C:\Use
Hash value <md5 _hashlib.HASH object @ 0x000001CD529724B0>
Hexadecimal equvalent: 5e79b66b78498aa1aec126b0b4ca1a1e
```

6. AIM: to implement the SHA-384 and SHA-512 Hash algorithm

Algorithm for SHA-384 and SHA-512 Hashing

1. **Start**.
2. **Input**: Take the string message `str = "Hello, World!"`.
3. **Convert** the input string to bytes using `str.encode()`.
4. **Hash using SHA-384**:
   - Apply the `sha384()` function from the `hashlib` library on the encoded message.
   - Store the resulting hash object in `result`.
   - Use the `hexdigest()` method to convert the hash object to a readable hexadecimal format.
   - **Output** the SHA-384 hash value.
5. **Hash using SHA-512**:
   - Apply the `sha512()` function from the `hashlib` library on the encoded message.
   - Store the resulting hash object in `result`.
   - Use the `hexdigest()` method to convert the hash object to a readable hexadecimal format.
   - **Output** the SHA-512 hash value.
6. **End**.

Code:

sha.py - C:\Users\raned\OneDrive\Desktop\TE IT\SEM 5\Security Lab\Prac

File Edit Format Run Options Window Help

```
import hashlib

str="Hello, World!"

result = hashlib.sha384(str.encode())
print("The Hexadecimal equivalent of SHA348 is:")
print(result.hexdigest())
print("\r")

result = hashlib.sha512(str.encode())
print("The Hexadecimal equivalent of SHA512 is:")
print(result.hexdigest())
```

Output:

```
=========================================== RESTART: C:\Users\raned\OneDrive\Desktop\TE IT\SEM 5\Security Lab\Practical exam pract
The Hexadecimal equivalent of SHA348 is:
5485cc9b3365b4305dfb4e8337e0a598a574f8242bf17289e0dd6c20a3cd44a089de16ab4ab308f63e44b1170eb5f515

The Hexadecimal equivalent of SHA512 is:
374d794a95cdcfd8b35993185fef9ba368f160d8daf432d08ba9f1ed1e5abe6cc69291e0fa2fe0006a52570ef18c19def4e617c33ce52ef0a6e5fbe318cb0387
```

7. AIM: to implement the DES Algorithm

Algorithm for DES encryption and decryption:

1. **Step 1: Import the required library**
   - Import the `DES` module from the `Crypto.Cipher` package.
2. **Step 2: Define the encryption key**
   - Create an 8-byte key (`key = b'8bytekey'`).
   - DES requires a key of exactly 8 bytes, so ensure the key length is correct.
3. **Step 3: Initialize the DES cipher**
   - Use `DES.new(key, DES.MODE_ECB)` to create a DES cipher object.
   - Here, the ECB (Electronic Codebook) mode is chosen for encryption.
4. **Step 4: Pad the plaintext**
   - DES operates on 8-byte blocks, so the plaintext must be padded to ensure its length is a multiple of 8 bytes.
   - Padding is done by adding spaces (`' '`) to the end of the plaintext until its length is divisible by 8.
5. **Step 5: Encrypt the plaintext**
   - Convert the padded plaintext to bytes using `padded_text.encode()` and pass it to the `encrypt()` method.
   - The `encrypt()` method returns the encrypted bytes.
6. **Step 6: Decrypt the ciphertext**
   - To reverse the encryption process, pass the encrypted data to the `decrypt()` method.
   - Decode the decrypted bytes back to a string using `.decode()` and remove any padding (spaces) using `.strip()`.
7. **Step 7: Output the encrypted and decrypted text**
   - Print the encrypted ciphertext in bytes format and the decrypted plaintext.

To run the code

Run: pip install pycryptodome

Code :

DES.py - C:/Users/raned/OneDrive/Desktop/TE IT/SEM 5/Security Lab/Practical exam practice/DES.py

File Edit Format Run Options Window Help

```
from Crypto.Cipher import DES

key = b'8bytekey'  # 8-byte key
cipher = DES.new(key, DES.MODE_ECB)

plaintext = "Hello DES"  # Text to encrypt
padded_text = plaintext + ' ' * (8 - len(plaintext) % 8)  # Padding

encrypted = cipher.encrypt(padded_text.encode())
print("Encrypted:", encrypted)

decrypted = cipher.decrypt(encrypted).decode().strip()
print("Decrypted:", decrypted)
```

Output:

```
=========================================== RESTART: C:/Users/ra
Encrypted: b'\x83\x03\xe1\xc8\x85V\xde\xbb\xe6\x91LA<>\xd5\xf6'
Decrypted: Hello DES
```

Codes to copy and run

1. Ceaser Cipher

```
def ceaser_cipher(text,shift):
        result=""
        for char in text:
                if char.isalpha():
                        shift_base=65 if char.isupper() else 97
                        result += chr((ord(char)-shift_base+shift)%26
+shift_base)
                else:
                        result+=char
        return result
text= "Hello, World"
shift= 3
cipher_text = ceaser_cipher(text,shift)
print("Cipher text", cipher_text)
```

2. Playfare

```
def create_matrix(key):
   key = key.upper().replace('J', 'I')  # Replace 'J' with 'I'
   key = ''.join(dict.fromkeys(key))    # Remove duplicates while preserving
order
   alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"  # Alphabet excluding 'J'

   # Create a 5x5 matrix
   matrix = [char for char in key if char in alphabet] + [char for char in
alphabet if char not in key]
   return [matrix[i:i + 5] for i in range(0, 25, 5)]
```

```
def find_position(matrix, char):
    for r, row in enumerate(matrix):
        if char in row:
            return r, row.index(char)

def playfair_cipher(text, key):
    matrix = create_matrix(key)  # Create the Playfair matrix
    text = text.upper().replace("J", "I").replace(" ", "")  # Prepare text

    # Add 'X' if the text length is odd
    if len(text) % 2 != 0:
        text += 'X'

    cipher = ''
    for i in range(0, len(text), 2):
        a, b = text[i], text[i + 1]  # Get pairs of characters
        row_a, col_a = find_position(matrix, a)
        row_b, col_b = find_position(matrix, b)

        # Apply Playfair rules
        if row_a == row_b:  # Same row
            cipher += matrix[row_a][(col_a + 1) % 5] + matrix[row_b][(col_b + 1) %
5]
        elif col_a == col_b:  # Same column
            cipher += matrix[(row_a + 1) % 5][col_a] + matrix[(row_b + 1) %
5][col_b]
```

```
        else:  # Rectangle case
            cipher += matrix[row_a][col_b] + matrix[row_b][col_a]

    return cipher

# Example usage
key = "playfair example"
text = "hide the gold"
print("Ciphered Text:", playfair_cipher(text, key))
```

3. RSA

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
def generate_keys():
    p, q = 61, 53
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 17
    d = mod_inverse(e, phi)
```

```
    return (e, n), (d, n)

def encrypt(message, public_key):
    e, n = public_key
    return [(ord(char) ** e) % n for char in message]

def decrypt(cipher, private_key):
    d, n = private_key
    return ''.join([chr((char ** d) % n) for char in cipher])

# Example usage
public_key, private_key = generate_keys()
message = "HELLO"
encrypted = encrypt(message, public_key)
decrypted = decrypt(encrypted, private_key)

print("Original:", message)
print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

4. Diffie-helman

```
import random
def generate_private_key(prime):
    return random.randint(1, prime - 1)
```

```
def compute_public_key(base, private_key, prime):
    return (base ** private_key) % prime

def compute_shared_secret(public_key, private_key, prime):
    return (public_key ** private_key) % prime

prime = 29
base = 5
private_key_a = generate_private_key(prime)
public_key_a = compute_public_key(base, private_key_a, prime)

private_key_b = generate_private_key(prime)
public_key_b = compute_public_key(base, private_key_b, prime)

shared_secret_a = compute_shared_secret(public_key_b, private_key_a, prime)
shared_secret_b = compute_shared_secret(public_key_a, private_key_b, prime)

print("User A's private key:", private_key_a)
print("User A's public key:", public_key_a)
print("User B's private key:", private_key_b)
print("User B's public key:", public_key_b)
print("Shared secret (A):", shared_secret_a)
print("Shared secret (B):", shared_secret_b)
```

5.Md5

```
import hashlib

string = "Darshan"
encoded=string.encode()
result=hashlib.md5(encoded)
print("Hash value", result)
print("Hexadecimal equvalent:",result.hexdigest())
```

6.sha

```
import hashlib

str="Hello, World!"

result = hashlib.sha384(str.encode())
print("The Hexadecimal equivalent of SHA348 is:")
print(result.hexdigest())
print("\r")

result = hashlib.sha512(str.encode())
print("The Hexadecimal equivalent of SHA512 is:")
print(result.hexdigest())
```

7.DES

```
from Crypto.Cipher import DES

key = b'8bytekey'  # 8-byte key
cipher = DES.new(key, DES.MODE_ECB)

plaintext = "Hello DES"  # Text to encrypt
padded_text = plaintext + ' ' * (8 - len(plaintext) % 8)  # Padding

encrypted = cipher.encrypt(padded_text.encode())
print("Encrypted:", encrypted)

decrypted = cipher.decrypt(encrypted).decode().strip()
print("Decrypted:", decrypted)
```