TABLE OF CONTENTS

EX.NO:1	**Implementation of Caesar Cipher**

Aim:

To implements a Caesar cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:

1) Define Function:

    Define a function encrypt_text(plaintext, n) which takes plaintext (the text to be encrypted) and n (the shift pattern) as input.

2) Initialize Answer:

    Initialize an empty string ans to store the encrypted text.

3) Iterate Over Plaintext:

    Loop through each character ch in plaintext using its index i.

4) Check Character Type:

    If ch is a space, append a space to ans.
    Else if ch is an uppercase letter:
    Convert ch to its corresponding encrypted character using the formula:
    encrypted_char ==((ord(ch)+n−65)%26)+65
    Append the encrypted character to ans.
    Else if ch is a lowercase letter:
    Convert ch to its corresponding encrypted character using the formula:
    encrypted_char =((ord(ch)+n−97)%26)+97
    Append the encrypted character to ans.

5) Return Encrypted Text:

    Return the encrypted text stored in ans.

6) Print Results:

    Define plaintext and n.
    Print the original plaintext and shift pattern.
    Call encrypt_text(plaintext, n) and print the resulting encrypted text.

Example

Given the plaintext = "HELLO EVERYONE" and n = 1:
Define plaintext and n.
Initialize ans as an empty string.
Loop through each character in "HELLO EVERYONE":
'H' is uppercase, so encrypt it to 'I'.
'E' is uppercase, so encrypt it to 'F'.
'L' is uppercase, so encrypt it to 'M'.
'L' is uppercase, so encrypt it to 'M'.

'O' is uppercase, so encrypt it to 'P'.
' ' is a space, so append ' '.
'E' is uppercase, so encrypt it to 'F'.
'V' is uppercase, so encrypt it to 'W'.
'E' is uppercase, so encrypt it to 'F'.
'R' is uppercase, so encrypt it to 'S'.
'Y' is uppercase, so encrypt it to 'Z'.
'O' is uppercase, so encrypt it to 'P'.
'N' is uppercase, so encrypt it to 'O'.
'E' is uppercase, so encrypt it to 'F'.
Return the encrypted text: "IFMMP FWFSZPOF".
Print the plaintext, shift pattern, and encrypted text.

Program:

```
def encrypt_text(plaintext,n):
    ans = ""
    # iterate over the given text
    for i in range(len(plaintext)):
        ch = plaintext[i]
         # check if space is there then simply add space
        if ch==" ":
            ans+=" "
        # check if a character is uppercase then encrypt it accordingly
        elif (ch.isupper()):
            ans += chr((ord(ch) + n-65) % 26 + 65)
        # check if a character is lowercase then encrypt it accordingly
        else:
            ans += chr((ord(ch) + n-97) % 26 + 97)
        return ans
plaintext = "HELLO EVERYONE"
n = 1
print("Plain Text is : " + plaintext)
print("Shift pattern is : " + str(n))
print("Cipher Text is : " + encrypt_text(plaintext,n))
```

Output:

```
Plain Text is: HELLO EVERYONE
Shift pattern is: 1
Cipher Text is: IFMMP FWFSZPOF
```

Result:

EX.NO:2	
	Basic Monoalphabetic Cipher

Aim:

To implements a basic monoalphabetic cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:

1) Define generate_cipher_key Function:
    - Input: shift (integer)
    - Initialize alphabet as a string containing 'abcdefghijklmnopqrstuvwxyz'.
    - Create shifted_alphabet by shifting alphabet by shift positions.
    - Create a dictionary key by mapping each character in alphabet to the
    - corresponding character in shifted_alphabet.
    - Return key.
2) Define encrypt Function:
    - Input: message (string), key (dictionary)
    - Initialize an empty string encrypted_message.
    - Loop through each character char in message:
    - If char is alphabetic:
    - If char is lowercase, append key[char] to encrypted_message.
    - If char is uppercase, append key[char.lower()].upper() to encrypted_message.
    - Else, append char to encrypted_message.
    - Return encrypted_message.
3) Define decrypt Function:
    - Input: ciphertext (string), key (dictionary)
    - Create reverse_key by reversing the key dictionary.
    - Initialize an empty string decrypted_message.
    - Loop through each character char in ciphertext:
    - If char is alphabetic:
    - If char is lowercase, append reverse_key[char] to decrypted_message.
    - If char is uppercase, append reverse_key[char.lower()].upper() to decrypted_message.
    - Else, append char to decrypted_message.
    - Return decrypted_message.
4) Define main Function:
    - Prompt user to input shift value.
    - Generate key using generate_cipher_key(shift).
    - Prompt user to choose between encryption and decryption (e or d).

- If the user chooses 'e':
- Prompt for the plaintext message.
- Encrypt the plaintext using encrypt(plaintext, key).
- Print the encrypted message.
- If the user chooses 'd':
- Prompt for the ciphertext message.
- Decrypt the ciphertext using decrypt(ciphertext, key).
- Print the decrypted message.
- If the user inputs an invalid choice, print an error message.

5) Execute main Function:

If this script is run as the main module, call the main function.

Program:

```
def generate_cipher_key(shift):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    shifted_alphabet = alphabet[shift:] + alphabet[:shift]
    key = dict(zip(alphabet, shifted_alphabet))
    return key
def encrypt(message, key):
    encrypted_message = ''
    for char in message:
        if char.isalpha():
            if char.islower():
                encrypted_message += key[char]
            else:
                encrypted_message += key[char.lower()].upper()
        else:
            encrypted_message += char
    return encrypted_message
def decrypt(ciphertext, key):
    reverse_key = {v: k for k, v in key.items()}
    decrypted_message = ''
    for char in ciphertext:
        if char.isalpha():
            if char.islower():
                decrypted_message += reverse_key[char]
            else:
                decrypted_message += reverse_key[char.lower()].upper()
        else:
            decrypted_message += char
    return decrypted_message
def main():
    shift = int(input("Enter the shift value for the cipher: "))
    key = generate_cipher_key(shift)
```

```
    choice = input("Encrypt or decrypt? (e/d): ").lower()
    if choice == 'e':
        plaintext = input("Enter the message to encrypt: ")
        encrypted = encrypt(plaintext, key)
        print("Encrypted message:", encrypted)
    elif choice == 'd':
        ciphertext = input("Enter the message to decrypt: ")
        decrypted = decrypt(ciphertext, key)
        print("Decrypted message:", decrypted)
    else:
        print("Invalid choice. Please enter 'e' for encrypt or 'd' for decrypt.")
if __name__ == "__main__":
    main()
```

Output:

```
Enter the shift value for the cipher: 3
Encrypt or decrypt? (e/d): e
Enter the message to encrypt: hello world
Encrypted message: khoor zruog

Enter the shift value for the cipher: 3
Encrypt or decrypt? (d): d
Enter the message to decrypt: khoor zruog
Decrypted message: hello world
```

Result:

EX.NO:3	
	Message Authentication Code

Aim:

To calculate the messages digest of a text using the SHA-1 algorithm and thereby verifying data integrity

Algorithm:

1) Import Hashlib Module:

   Import the hashlib module to use various SHA hash functions.
2) Compute SHA256 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode() to convert it to bytes.
   - Pass the encoded string to hashlib.sha256() to compute the SHA256 hash.
   - Get the hexadecimal representation of the hash using result.hexdigest().
   - Print the message "The hexadecimal equivalent of SHA256 is :" followed by the hexadecimal value of the SHA256 hash.
3) Compute SHA384 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode().
   - Pass the encoded string to hashlib.sha384() to compute the SHA384 hash.
   - Get the hexadecimal representation of the hash using result.hexdigest().
   - Print the message "The hexadecimal equivalent of SHA384 is :" followed by the hexadecimal value of the SHA384 hash.
4) Compute SHA224 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode().
   - Pass the encoded string to hashlib.sha224() to compute the SHA224 hash.
   - Get the hexadecimal representation of the hash using `result.hexdigest()
5) Encode and hash using SHA512:
   - Reinitialize the string: str = "GeeksforGeeks"
   - Encode the string: encoded_str = str.encode()
   - Hash the encoded string using SHA512: result = hashlib.sha512(encoded_str)
   - Print the hexadecimal equivalent
6) Encode and hash using SHA1:
   - Reinitialize the string: str = "GeeksforGeeks"
   - Encode the string: encoded_str = str.encode()
   - Hash the encoded string using SHA1: result = hashlib.sha1(encoded_str)

- Print the hexadecimal equivalent.

Program:

```
import hashlib
  # initializing string
str = "GeeksforGeeks"
  # encoding GeeksforGeeks using encode()
# then sending to SHA256()
result = hashlib.sha256(str.encode())
  # printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA256 is : ")
print(result.hexdigest())
  print ("\r")
  # initializing string
str = "GeeksforGeeks"
  # encoding GeeksforGeeks using encode()
# then sending to SHA384()
result = hashlib.sha384(str.encode())
  # printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())
  print ("\r")
  # initializing string
str = "GeeksforGeeks"
  # encoding GeeksforGeeks using encode()
# then sending to SHA224()
result = hashlib.sha224(str.encode())
 # printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())
  print ("\r")
  # initializing string
str = "GeeksforGeeks"
  # encoding GeeksforGeeks using encode()
# then sending to SHA512()
result = hashlib.sha512(str.encode())
  # printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())
 print ("\r")
  # initializing string
str = "GeeksforGeeks"
  # encoding GeeksforGeeks using encode()
# then sending to SHA1()
result = hashlib.sha1(str.encode())
 # printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ")
```

print(result.hexdigest())

Output:
The hexadecimal equivalent of SHA256 is :
f6071725e7ddeb434fb6b32b8ec4a2b14dd7db0d785347b2fb48f9975126178f
The hexadecimal equivalent of SHA384 is :
d1e67b8819b009ec7929933b6fc1928dd64b5df31bcde6381b9d3f90488d253240490460c0a5a1a873da8236c12ef9b3
The hexadecimal equivalent of SHA224 is :
173994f309f727ca939bb185086cd7b36e66141c9e52ba0bdcfd145d
The hexadecimal equivalent of SHA512 is :
0d8fb9370a5bf7b892be4865cdf8b658a82209624e33ed71cae353b0df254a75db63d1baa35ad99f26f1b399c31f3c666a7fc67ecef3bdcdb7d60e8ada90b722
The hexadecimal equivalent of SHA1 is :
4175a37afd561152fb60c305d4fa6026b7e79856

Result:

EX.NO:4	Data Encryption Standard

Aim:

To implement a symmetric-key block cipher algorithm known as Data Encryption Standard (DES).

Algorithm:

1) **Hexadecimal to Binary Conversion (`hex2bin`)**:

- Initialize a dictionary that maps each hexadecimal digit to its 4-bit binary equivalent.
- Initialize an empty string for the binary result.
- For each character in the input hexadecimal string:
  - Append the corresponding binary string from the dictionary to the result.
- Return the binary result.

2) **Binary to Hexadecimal Conversion (`bin2hex`)**:

- Initialize a dictionary that maps each 4-bit binary string to its hexadecimal equivalent.
- Initialize an empty string for the hexadecimal result.
- For each group of 4 bits in the input binary string:
  - Append the corresponding hexadecimal character from the dictionary to the result.
- Return the hexadecimal result.

3) **Binary to Decimal Conversion (`bin2dec`)**:

- Initialize the decimal result to 0.
- For each bit in the input binary string, from least significant to most significant:
  - Multiply the bit by 2i2^i2i (where iii is the bit's position) and add to the decimal result.
- Return the decimal result.

4) **Decimal to Binary Conversion (`dec2bin`)**:

- Convert the decimal number to its binary representation using Python's `bin` function and remove the "0b" prefix.

- If the length of the binary result is not a multiple of 4, pad with leading zeros to make it a multiple of 4.

- Return the padded binary result.

5) **Permute Function (`permute`)**:

- Initialize an empty string for the permutation result.
- For each position in the permutation array:
  - Append the bit from the input string at the given position to the result.
- Return the permutation result.

6) **Left Shift Function (`shift_left`)**:

- For the specified number of shifts:
  - Perform a left circular shift on the input string.
- Return the shifted string.

7) **XOR Function (`xor`)**:

- Initialize an empty string for the XOR result.
- For each bit in the input strings:
  - Append the result of the XOR operation on the corresponding bits to the result.
- Return the XOR result.

8) **Encryption Function (`encrypt`)**:

- Convert the plaintext from hexadecimal to binary using `hex2bin`.
- Perform an initial permutation using the `initial_perm` table.
- Split the permuted text into left and right halves.
- For each of the 16 rounds:
  - Expand the right half from 32 to 48 bits using the `exp_d` table.
  - XOR the expanded right half with the round key.
  - Substitute the result using the S-boxes.
  - Perform a permutation using the `per` table.
  - XOR the result with the left half.
  - Swap the left and right halves, except in the final round.
- Combine the final left and right halves.
- Perform a final permutation using the `final_perm` table.
- Return the result as binary.

9) **Key Generation**:

- Convert the key from hexadecimal to binary using `hex2bin`.
- Perform a parity bit drop using the `keyp` table to get a 56-bit key.
- Split the key into left and right halves.
- For each of the 16 rounds:
  - Perform left shifts on both halves according to the `shift_table`.
  - Combine the left and right halves.
  - Compress the key from 56 to 48 bits using the `key_comp` table.

- Append the round key in both binary and hexadecimal form to the round key lists.

10) **Main Process**:

- Define the plaintext and key in hexadecimal format.
- Generate the round keys.
- Perform encryption using the generated round keys.
- Reverse the round keys for decryption.
- Perform decryption using the reversed round keys.

Program:

```
# Hexadecimal to binary conversion
def hex2bin(s):
    mp = {'0': "0000",
          '1': "0001",
          '2': "0010",
          '3': "0011",
          '4': "0100",
          '5': "0101",
          '6': "0110",
          '7': "0111",
          '8': "1000",
          '9': "1001",
          'A': "1010",
          'B': "1011",
          'C': "1100",
          'D': "1101",
          'E': "1110",
          'F': "1111"}
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin
# Binary to hexadecimal conversion
def bin2hex(s):
    mp = {"0000": '0',
          "0001": '1',
          "0010": '2',
          "0011": '3',
          "0100": '4',
          "0101": '5',
          "0110": '6',
          "0111": '7',
          "1000": '8',
          "1001": '9',
```

```
        "1010": 'A',
        "1011": 'B',
        "1100": 'C',
        "1101": 'D',
        "1110": 'E',
        "1111": 'F'}
  hex = ""
  for i in range(0, len(s), 4):
      ch = ""
      ch = ch + s[i]
      ch = ch + s[i + 1]
      ch = ch + s[i + 2]
      ch = ch + s[i + 3]
      hex = hex + mp[ch]
   return hex
# Binary to decimal conversion
def bin2dec(binary):
   binary1 = binary
  decimal, i, n = 0, 0, 0
  while(binary != 0):
      dec = binary % 10
      decimal = decimal + dec * pow(2, i)
      binary = binary//10
      i += 1
  return decimal
# Decimal to binary conversion
def dec2bin(num):
  res = bin(num).replace("0b", "")
  if(len(res) % 4 != 0):
      div = len(res) / 4
      div = int(div)
      counter = (4 * (div + 1)) - len(res)
      for i in range(0, counter):
          res = '0' + res
  return res
# Permute function to rearrange the bits
def permute(k, arr, n):
  permutation = ""
  for i in range(0, n):
      permutation = permutation + k[arr[i] - 1]
  return permutation
# shifting the bits towards left by nth shifts
def shift_left(k, nth_shifts):
  s = ""
  for i in range(nth_shifts):
      for j in range(1, len(k)):
```

```
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k
 # calculating xow of two strings of binary number a and b
 def xor(a, b):
    ans = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans
# Table of Position of 64 bits at initial level: Initial Permutation Table
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,
                62, 54, 46, 38, 30, 22, 14, 6,
                64, 56, 48, 40, 32, 24, 16, 8,
                57, 49, 41, 33, 25, 17, 9, 1,
                59, 51, 43, 35, 27, 19, 11, 3,
                61, 53, 45, 37, 29, 21, 13, 5,
               63, 55, 47, 39, 31, 23, 15, 7]
 # Expansion D-box Table
exp_d = [32, 1, 2, 3, 4, 5, 4, 5,
         6, 7, 8, 9, 8, 9, 10, 11,
         12, 13, 12, 13, 14, 15, 16, 17,
         16, 17, 18, 19, 20, 21, 20, 21,
         22, 23, 24, 25, 24, 25, 26, 27,
         28, 29, 28, 29, 30, 31, 32, 1]
 # Straight Permutation Table
per = [16,  7, 20, 21,
       29, 12, 28, 17,
       1, 15, 23, 26,
       5, 18, 31, 10,
       2,  8, 24, 14,
       32, 27,  3,  9,
       19, 13, 30,  6,
       22, 11,  4, 25]
 # S-box Table
sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
         [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
         [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
         [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],
         [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
         [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
```

```
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],
        [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]],
        [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]],
        [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]],
        [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]],
        [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]],
        [[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]]
# Final Permutation Table
final_perm = [40, 8, 48, 16, 56, 24, 64, 32,
              39, 7, 47, 15, 55, 23, 63, 31,
              38, 6, 46, 14, 54, 22, 62, 30,
              37, 5, 45, 13, 53, 21, 61, 29,
              36, 4, 44, 12, 52, 20, 60, 28,
              35, 3, 43, 11, 51, 19, 59, 27,
              34, 2, 42, 10, 50, 18, 58, 26,
              33, 1, 41, 9, 49, 17, 57, 25]
def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)
    # Initial Permutation
    pt = permute(pt, initial_perm, 64)
    print("After initial permutation", bin2hex(pt))
    # Splitting
    left = pt[0:32]
    right = pt[32:64]
    for i in range(0, 16):
        #  Expansion D-box: Expanding the 32 bits data into 48 bits
        right_expanded = permute(right, exp_d, 48)
```

```
        # XOR RoundKey[i] and right_expanded
        xor_x = xor(right_expanded, rkb[i])
        # S-boxex: substituting the value from s-box table by calculating row and column
        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
            col = bin2dec(
                int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6 + 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)
        # Straight D-box: After substituting rearranging the bits
        sbox_str = permute(sbox_str, per, 32)
        # XOR left and sbox_str
        result = xor(left, sbox_str)
        left = result
        # Swapper
        if(i != 15):
            left, right = right, left
        print("Round ", i + 1, " ", bin2hex(left),
              " ", bin2hex(right), " ", rk[i])

    # Combination
    combine = left + right
    # Final permutation: final rearranging of bits to get cipher text
    cipher_text = permute(combine, final_perm, 64)
    return cipher_text
pt = "123456ABCD132536"
key = "AABB09182736CCDD"
# Key generation
# --hex to binary
key = hex2bin(key)
# --parity bit drop table
keyp = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4]
# getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56)
# Number of bit shifts
shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
               1, 2, 2, 2,
```

```
            2, 2, 2, 1]
 # Key- Compression Table : Compression of key from 56 bits to 48 bits
key_comp = [14, 17, 11, 24, 1, 5,
        3, 28, 15, 6, 21, 10,
        23, 19, 12, 4, 26, 8,
        16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32]
 # Splitting
left = key[0:28]    # rkb for RoundKeys in binary
right = key[28:56]  # rk for RoundKeys in hexadecimal
 rkb = []
rk = []
for i in range(0, 16):
    # Shifting the bits by nth shifts by checking from shift table
    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])
     # Combination of left and right string
    combine_str = left + right
     # Compression of key from 56 to 48 bits
    round_key = permute(combine_str, key_comp, 48)
     rkb.append(round_key)
    rk.append(bin2hex(round_key))

print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ", cipher_text)
print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ", text)
```

Output 1:

```
...60AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 19BA9212 CF26B472 181C5D75C66D
Cipher Text: C0B7A8D05F3A829C
Decryption
After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472
```

Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C
Plain Text: 123456ABCD132536

Output 2:
Encryption:
After initial permutation: 14A7D67818CA18AD
After splitting: L0=14A7D678 R0=18CA18AD
Round 1 18CA18AD 5A78E394 194CD072DE8C
Round 2 5A78E394 4A1210F6 4568581ABCCE
Round 3 4A1210F6 B8089591 06EDA4ACF5B5
Round 4 B8089591 236779C2 DA2D032B6EE3
Round 5 236779C2 A15A4B87 69A629FEC913
Round 6 A15A4B87 2E8F9C65 C1948E87475E
Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0
Round 8 A9FC20A3 308BEE97 34F822F0C66D
Round 9 308BEE97 10AF9D37 84BB4473DCCC
Round 10 10AF9D37 6CA6CB20 02765708B5BF
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 19BA9212 CF26B472 181C5D75C66D
Cipher Text: C0B7A8D05F3A829C
Decryption
After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472
Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F

Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C
Plain Text: 123456ABCD132536

Result:

EX.NO:5	**Advanced Encryption Standard**

Aim:

To understand the need of highly secured symmetric encryption algorithm known as Advanced Encryption Standard (AES)

Algorithm:

1. **Import Libraries:**
    - `AES` for AES encryption/decryption.
    - `get_random_bytes` to generate a random key.
    - `pad` and `unpad` to ensure data is of a valid block size.
2. **Encrypt Function:**
    - Creates a new AES cipher object in CBC mode.
    - Pads the data to be a multiple of the block size.
    - Encrypts the padded data.
    - Returns the initialization vector (IV) and the ciphertext.
3. **Decrypt Function:**
    - Creates a new AES cipher object with the same IV.
    - Decrypts the ciphertext.
    - Unpads and decodes the decrypted data.
4. **Example Usage:**
    - Generates a random 16-byte key.
    - Encrypts a sample message.
    - Prints the ciphertext in hexadecimal format.
    - Decrypts the ciphertext.
    - Prints the decrypted message.

Input

The input for the code consists of the plaintext message that you want to encrypt. In the example provided, the input message is hardcoded as "This is a secret message."

Output

The output of the code will consist of the ciphertext (in hexadecimal format) and the decrypted message.

Example

Let's break down the expected output when running the code:

1. **Encrypted Ciphertext:** The encrypted version of the plaintext message, displayed in hexadecimal format.
2. **Decrypted Data:** The original message after decrypting the ciphertext

Program:

The `pycryptodome` library in Python provides a simple way to implement AES.

pip install pycryptodome

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Function to encrypt data
def encrypt(data, key):
    # Create a cipher object
    cipher = AES.new(key, AES.MODE_CBC)
    # Pad the data to make it a multiple of AES block size
    padded_data = pad(data.encode(), AES.block_size)
    # Encrypt the data
    ciphertext = cipher.encrypt(padded_data)
    # Return the ciphertext and the IV (Initialization Vector)
    return cipher.iv, ciphertext

# Function to decrypt data
def decrypt(iv, ciphertext, key):
    # Create a cipher object with the same IV
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    # Decrypt the data
    decrypted_data = cipher.decrypt(ciphertext)
    # Unpad the decrypted data and decode it
    return unpad(decrypted_data, AES.block_size).decode()

# Example usage
if __name__ == "__main__":
    # 16-byte key (128 bits)
    key = get_random_bytes(16)
    data = "This is a secret message."

    # Encrypt the data
    iv, ciphertext = encrypt(data, key)
    print(f"Ciphertext: {ciphertext.hex()}")

    # Decrypt the data
    decrypted_data = decrypt(iv, ciphertext, key)
    print(f"Decrypted data: {decrypted_data}")
```

Output:

Ciphertext: 2f8f7e80b87f8a60e617f5075a2b7c0d1b3d8f4f5b6a0e8d1e6f8a2d3b7a4c5d

Decrypted data: This is a secret message.

Result:

EX.NO:6	**Asymmetric Key Encryption**

Aim:

To implement the popular asymmetric key algorithm Rivest,Shamir ,Adleman (RSA)

Algorithm:

1) Input:
   - Two prime numbers p and q.
   - A plaintext message .
   - Calculate n:

     n = p * q
     For p=53 and q=59, n = 53 * 59 = 3127

2) Calculate the totient t:
   - t = (p - 1) * (q - 1)
   - For p=53 and q=59, t = (53 - 1) * (59 - 1) = 52 * 58 = 3016
   - Select the public key e:
   - Iterate from 2 to t to find the smallest integer e such that gcd(e, t) == 1.

3) Select the public key **e:**
   - Find the smallest integer e such that gcd(e, t) == 1
   - In this case, e = 3 (assuming the smallest integer that satisfies the condition)

4) Select the private key d:
   - Initialize j = 0.
   - Increment j in a loop until (j * e) % t == 1.
   - Set d = j.
   - Find d such that (d * e) % t == 1
   - Through iteration, if e = 3, then d = 2011 (assuming this is found through the while loop)

5) Encrypt the message:

   Calculate the ciphertext ct = (message ** e) % n.
   - ct = (message ** e) % n
   - For message=89, ct = (89 ** 3) % 3127 = 1394

6) Decrypt the message:
   - Calculate the decrypted message mes = (ct ** d) % n.
   - Print the encrypted message ct.

- Print the decrypted message mes
- mes = (ct ** d) % n
- For ct=1394, mes = (1394 ** 2011) % 3127 = 89

Program:

```
        from math import gcd
# defining a function to perform RSA approch
def RSA(p: int, q: int, message: int):
    # calculating n
    n = p * q
    # calculating totient, t
    t = (p - 1) * (q - 1)
    # selecting public key, e
    for i in range(2, t):
        if gcd(i, t) == 1:
            e = i
            break
        # selecting private key, d
    j = 0
    while True:
        if (j * e) % t == 1:
            d = j
            break
        j += 1

    # performing encryption
    ct = (message ** e) % n
    print(f"Encrypted message is {ct}")
    # performing decryption
    mes = (ct ** d) % n
    print(f"Decrypted message is {mes}")
# Testcase - 1
RSA(p=53, q=59, message=89)
# Testcase - 2
RSA(p=3, q=7, message=12)
```

Output:

Output for Test Cases

Test Case 1

- Input: `p=53, q=59, message=89`
- Output:

  Encrypted message is 1394
  Decrypted message is 89

Test Case 2

- Input: `p=3, q=7, message=12`
- Output:

  Encrypted message is 3
  Decrypted message is 12

Result:

EX.NO:7	
	Secure Key exchange

Aim:

To securely exchange the crypto graphic keys over Internet to implement Diffie- Hellman key exchange mechanism

Algorithm:

1. **Input:**
   - p: A prime number.
   - g: A primitive root of p.
   - The user is prompted to enter a prime number p and a number g (which is a primitive root of p).

2. **Initialize Classes:**
   - **Class `A`:** Represents Alice and Bob.
     - `__init__`: Generate a random private number `n` for Alice/Bob.
     - `publish`: Calculate and return the public value `g^n % p`.
     - `compute_secret`: Compute the shared secret `(gb^n) % p` using another party's public value `gb`.
     - Represents Alice and Bob.
     - Generates a random private number n.
     - Computes and returns the public value using publish.
     - Computes the shared secret using compute_secret.
   - **Class `B`:** Represents Eve.
     - `__init__`: Generate two random private numbers `a` and `b` for Eve.
     - `publish`: Calculate and return the public value `g^arr[i] % p` for Eve's private numbers.
     - `compute_secret`: Compute the shared secret `(ga^arr[i]) % p` using another party's public value `ga`.
     - Represents Eve.
     - Generates two random private numbers a and b.
     - Computes and returns the public value using publish.
     - Computes the shared secret using compute_secret

3. **Create Instances:**
   - Create an instance of A for Alice.
   - Create an instance of A for Bob.
   - Create an instance of B for Eve.
   - Instances of A are created for Alice and Bob.

- An instance of B is created for Eve.
- Private numbers selected by Alice, Bob, and Eve are printed.
- Public values are generated and printed.
- Shared secrets are computed and printed.

4. **Print Private Numbers:**

   Print the private numbers selected by Alice, Bob, and Eve.

5. **Generate Public Values:**
   - Calculate Alice's public value ga = g^alice.n % p.
   - Calculate Bob's public value gb = g^bob.n % p.
   - Calculate Eve's public values gea = g^eve.a % p and geb = g^eve.b % p.
6. **Print Public Values:**

   Print the public values generated by Alice, Bob, and Eve.

7. **Compute Shared Secrets:**
   - Calculate Alice's shared secret with Eve sa = gea^alice.n % p.
   - Calculate Eve's shared secret with Alice sea = ga^eve.a % p.
   - Calculate Bob's shared secret with Eve sb = geb^bob.n % p.
   - Calculate Eve's shared secret with Bob seb = gb^eve.b % p.

8. **Print Shared Secrets:**

   Print the shared secrets computed by Alice, Bob, and Eve.

Program:

```
import random
 # public keys are taken
# p is a prime number
# g is a primitive root of p
p = int(input('Enter a prime number : '))
g = int(input('Enter a number : '))
 class A:
    def __init__(self):
        # Generating a random private number selected by alice
        self.n = random.randint(1, p)
     def publish(self):
        # generating public values
        return (g**self.n)%p
     def compute_secret(self, gb):
        # computing secret key
```

```
        return (gb**self.n)%p
class B:
    def __init__(self):
        # Generating a random private number selected for alice
        self.a = random.randint(1, p)
        # Generating a random private number selected for bob
        self.b = random.randint(1, p)
        self.arr = [self.a,self.b]
    def publish(self, i):
        # generating public values
        return (g**self.arr[i])%p
    def compute_secret(self, ga, i):
        # computing secret key
        return (ga**self.arr[i])%p
alice = A()
bob = A()
eve = B()
# Printing out the private selected number by Alice and Bob
print(f'Alice selected (a) : {alice.n}')
print(f'Bob selected (b) : {bob.n}')
print(f'Eve selected private number for Alice (c) : {eve.a}')
print(f'Eve selected private number for Bob (d) : {eve.b}')
# Generating public values
ga = alice.publish()
gb = bob.publish()
gea = eve.publish(0)
geb = eve.publish(1)
print(f'Alice published (ga): {ga}')
print(f'Bob published (gb): {gb}')
print(f'Eve published value for Alice (gc): {gea}')
print(f'Eve published value for Bob (gd): {geb}')
# Computing the secret key
sa = alice.compute_secret(gea)
sea = eve.compute_secret(ga,0)
sb = bob.compute_secret(geb)
seb = eve.compute_secret(gb,1)
print(f'Alice computed (S1) : {sa}')
print(f'Eve computed key for Alice (S1) : {sea}')
print(f'Bob computed (S2) : {sb}')
print(f'Eve computed key for Bob (S2) : {seb}')
```

Output:

Enter a prime number (p) : 227
Enter a number (g) : 14
Alice selected (a) : 227
Bob selected (b) : 170

Eve selected private number for Alice (c) : 65
Eve selected private number for Bob (d) : 175
Alice published (ga): 14
Bob published (gb): 101

Eve published value for Alice (gc): 41
Eve published value for Bob (gd): 32
Alice computed (S1) : 41
Eve computed key for Alice (S1) : 41
Bob computed (S2) : 167
Eve computed key for Bob (S2) : 167

Result:

EX.NO:8	
	Digital Signature Generation

Aim:

To authenticate a message sent over the Internet using digital signature mechanism

Algorithm:

1) Generate RSA Keys:
   - The RSA key pair (private and public keys) is generated with a keysize of 2048 bits.
   - The keys are saved to files private.pem and public.pem.

2) Sign Message Function:
   - Imports the private key.
   - Creates a SHA-256 hash of the message.
   - Signs the hash using pkcs1_15 with the private key.
   - Returns the signature.

3) Verify Signature Function:
   - Imports the public key.
   - Creates a SHA-256 hash of the message.
   - Verifies the signature using pkcs1_15 with the public key.
   - Returns True if the signature is valid, otherwise returns False.

Example Usage:

The message "This is a secret message." is signed with the private key.
The signature is printed in hexadecimal format.
The signature is verified with the public key, and the result is printed.

Program:

Installation

First, install the pycryptodome library if you haven't already:

pip install pycryptodome

```
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.Random import get_random_bytes
# Generate RSA keys
key = RSA.generate(2048)
private_key = key.export_key()
public_key = key.publickey().export_key()
# Save the private and public keys to files
with open('private.pem', 'wb') as f:
    f.write(private_key)
```

```
with open('public.pem', 'wb') as f:
    f.write(public_key)
# Function to sign data
def sign_message(message, private_key):
    # Import the private key
    key = RSA.import_key(private_key)
    # Create a hash of the message
    h = SHA256.new(message.encode())
    # Sign the hash
    signature = pkcs1_15.new(key).sign(h)
    return signature
# Function to verify signature
def verify_signature(message, signature, public_key):
    # Import the public key
    key = RSA.import_key(public_key)
    # Create a hash of the message
    h = SHA256.new(message.encode())
    try:
        # Verify the signature
        pkcs1_15.new(key).verify(h, signature)
        return True
    except (ValueError, TypeError):
        return False
# Example usage
if __name__ == "__main__":
    message = "This is a secret message."
    # Sign the message
    signature = sign_message(message, private_key)
    print(f"Signature: {signature.hex()}")
    # Verify the signature
    is_valid = verify_signature(message, signature, public_key)
    print(f"Signature valid: {is_valid}")
```

Output:

When you run the code, you should see an output similar to the following:

```
Signature: <hexadecimal representation of the signature>
Signature valid: True
```

Result:

EX.NO:9	
	Implementation of Mobile Security

Aim:

To implements basic mobile security functionalities such as scanning for known malicious apps, encrypting and decrypting sensitive data, monitoring network traffic, and authenticating users.

Algorithm:

1) Import Required Libraries

    Import hashlib, os, socket, ssl, base64, and Fernet from cryptography.fernet.
    Import getpass for password input.

2) Define Known Malicious App Hashes

    Initialize known_malicious_apps with hashes of known malicious apps.

3) Function Definitions:

    Initialize an empty list malicious_apps.
    Iterate through each app in app_list.
    Compute the MD5 hash of app.
    If the computed hash exists in known_malicious_apps, add app to malicious_apps.
    Return malicious_apps.

4) generate_key()

    Use Fernet.generate_key() to generate a symmetric encryption key.
    Return key.

5) encrypt_data(data, key)

    Input: data (plaintext data to encrypt), key (encryption key)
    Output: encrypted_data (encrypted data)
    Initialize a Fernet object with key.
    Encrypt data using Fernet.encrypt() method.
    Return encrypted_data.

6) decrypt_data(encrypted_data, key)

    Input: encrypted_data (data to decrypt), key (encryption key)
    Output: decrypted_data (decrypted plaintext)
    Initialize a Fernet object with key.
    Decrypt encrypted_data using Fernet.decrypt() method.
    Decode decrypted_data and return.

7) monitor_network_traffic()

    Print "Monitoring network traffic..." (simulated functionality).

8) secure_connection(host, port)

Input: host (hostname or IP address), port (port number)
Create a default SSL context with ssl.create_default_context().
Create a TCP connection to host and port using socket.create_connection().
Wrap the socket with SSL/TLS using context.wrap_socket() and server_hostname=host.
Print the negotiated SSL/TLS version (ssock.version()).

9) authenticate_user(username, password, stored_hash)

Input: username (entered username), password (entered password), stored_hash (hashed password from storage)
Output: True (authentication successful) or False (authentication failed)
Compute the SHA-256 hash of password.
Compare the computed hash with stored_hash.
Return True if they match, otherwise False.
Main Program Execution (if __name__ == "__main__":)

Part 1 - Scan for Malicious Apps:

Demonstrates scanning installed apps (["app1", "malicious_app"] in this case) against a predefined list of known malicious app hashes (known_malicious_apps). It identifies 'malicious_app' as malicious.

Part 2 - Secure Data Storage:

Generates a key for encryption using generate_key().
Encrypts the sensitive data "This is a sensitive information" and prints both encrypted and decrypted versions.

Part 3 - Monitor Network Traffic:

Simulates monitoring network traffic.

Part 4 - Establish Secure Connection:

Establishes a secure TLS connection to www.example.com on port 443.

Part 5 - User Authentication:

Simulates user authentication where the user enters a username and password (user123 and secure_password respectively in this example). It verifies the entered password against a stored hash (stored_hash).

Program:

```
import hashlib
import os
import socket
import ssl
import base64
from cryptography.fernet import Fernet
from getpass import getpass
# Sample list of known malicious app hashes
known_malicious_apps = [
    "5d41402abc4b2a76b9719d911017c592",  # example hash of a malicious app
]
# Function to scan installed apps for malicious software
def scan_for_malicious_apps(app_list):
    malicious_apps = []
    for app in app_list:
```

```
        app_hash = hashlib.md5(app.encode()).hexdigest()
        if app_hash in known_malicious_apps:
            malicious_apps.append(app)
    return malicious_apps
# Function to generate a key for encryption
def generate_key():
    return Fernet.generate_key()
# Function to encrypt data
def encrypt_data(data, key):
    fernet = Fernet(key)
    encrypted_data = fernet.encrypt(data.encode())
    return encrypted_data
# Function to decrypt data
def decrypt_data(encrypted_data, key):
    fernet = Fernet(key)
    decrypted_data = fernet.decrypt(encrypted_data).decode()
    return decrypted_data
# Function to monitor network traffic
def monitor_network_traffic():
    # Simulating network traffic monitoring (For actual implementation, use libraries
like scapy)
    print("Monitoring network traffic...")
# Function to establish a secure connection
def secure_connection(host, port):
    context = ssl.create_default_context()
    with socket.create_connection((host, port)) as sock:
        with context.wrap_socket(sock, server_hostname=host) as ssock:
            print(ssock.version())
# Function to implement user authentication
def authenticate_user(username, password, stored_hash):
    password_hash = hashlib.sha256(password.encode()).hexdigest()
    return password_hash == stored_hash
# Sample usage
if __name__ == "__main__":
    # Part 1: Scan for malicious apps
    print("=== Part 1: Scan for Malicious Apps ===")
    installed_apps = ["app1", "malicious_app"]
    malicious_apps_found = scan_for_malicious_apps(installed_apps)
    if malicious_apps_found:
        print("Malicious apps found:", malicious_apps_found)
    else:
        print("No malicious apps found.")
    # Part 2: Secure data storage
    print("\n=== Part 2: Secure Data Storage ===")
    key = generate_key()
    sensitive_data = "This is a sensitive information"
```

```
encrypted_data = encrypt_data(sensitive_data, key)
print("Sensitive data:", sensitive_data)
print("Encrypted data:", encrypted_data)
decrypted_data = decrypt_data(encrypted_data, key)
print("Decrypted data:", decrypted_data)
# Part 3: Monitor network traffic
print("\n=== Part 3: Monitor Network Traffic ===")
monitor_network_traffic()
# Part 4: Establish a secure connection
print("\n=== Part 4: Establish Secure Connection ===")
secure_connection("www.example.com", 443)
# Part 5: User authentication
print("\n=== Part 5: User Authentication ===")
username = input("Enter username: ")
password = getpass("Enter password: ")
stored_hash = hashlib.sha256("secure_password".encode()).hexdigest()
if authenticate_user(username, password, stored_hash):
    print("Authentication successful.")
else:
    print("Authentication failed.")
```

Output :

```
=== Part 1: Scan for Malicious Apps ===
Malicious apps found: ['malicious_app']
=== Part 2: Secure Data Storage ===
Sensitive data: This is a sensitive information
Encrypted data: b'...'
Decrypted data: This is a sensitive information
=== Part 3: Monitor Network Traffic ===
Monitoring network traffic...
=== Part 4: Establish Secure Connection ===
TLSv1.3
=== Part 5: User Authentication ===
Enter username: user123
Enter password:
Authentication successful.
```

Result:

EX.NO:10	**Intrusion Detection/Prevention System with Snort**

Aim:

To Configure Snort to monitor network traffic, detect intrusion attempts, log them, and report when an intrusion attempt is detected.

Algorithm /Program:

1. Install Snort

1.1 Update your system:

```
bash
sudo apt-get update
```

1.2 Install necessary dependencies:

```
bash
sudo apt-get install -y build-essential libpcap-dev libpcre3-dev
libdumbnet-dev bison flex
```

1.3 Download and install Snort:

```
bash
wget https://www.snort.org/downloads/snort/snort-2.9.17.tar.gz
tar -xzvf snort-2.9.17.tar.gz
cd snort-2.9.17
./configure
make
sudo make install
```

1.4 Verify Snort installation:

```
bash
snort -V
```

2. Configure Snort

Create necessary directories:

```
bash
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /var/log/snort
sudo mkdir /usr/local/lib/snort_dynamicrules
```

3. Copy configuration files:

```
bash
sudo cp etc/* /etc/snort
sudo cp src/dynamic-preprocessors/build/usr/local/
lib/snort_dynamicpreprocessor/*
/usr/local/lib/snort_dynamicpreprocessor/
```

4.Download and update the rule set:

```
bash
wget https://www.snort.org/rules/snortrules-
snapshot-29170.tar.gz -O /tmp/snortrules-
```

snapshot.tar.gz
tar -xzvf /tmp/snortrules-snapshot.tar.gz -C /etc/snort/rules

5.Edit the Snort configuration file:

Open /etc/snort/snort.conf in a text editor:

bash
sudo nano /etc/snort/snort.conf

1. Set the following variables:

plaintext
ipvar HOME_NET any
ipvar EXTERNAL_NET any
var RULE_PATH /etc/snort/rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
var WHITE_LIST_PATH /etc/snort/rules
var BLACK_LIST_PATH /etc/snort/rules
output unified2: filename snort.u2, limit 128

2.Include rules:

plaintext
include $RULE_PATH/local.rules

3. Create a Local Rule File

Create local.rules file:
bash
sudo nano /etc/snort/rules/local.rules

4.Add a sample rule:

plaintext
alert icmp any any -> $HOME_NET any (msg:"ICMP Packet Detected"; sid:1000001; rev:001;)

5.Run Snort

Test Snort configuration:
bash
sudo snort -T -c /etc/snort/snort.conf

6.Run Snort in intrusion detection mode:

bash
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
Replace eth0 with the appropriate network interface.

7.Generate and Test Intrusion Detection

Generate network traffic:
Use ping or other network utilities to generate traffic.

For example:
bash
ping -c 4 8.8.8.8
Check Snort alerts:
Snort should display alerts on the console for detected ICMP packets as specified in local.rules.

8. Log and Report Intrusion Attempts

Configure logging in snort.conf:

Ensure the following line is present for unified2 logging:
plaintext
Copy code
output unified2: filename snort.u2, limit 128
Analyze logs:
Install Barnyard2 to process Snort logs:
bash
sudo apt-get install -y barnyard2
Configure Barnyard2 to read Snort's unified2 logs and output to a database or other formats.
Start Barnyard2:
bash
sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w
/var/log/snort/barnyard2.waldo
9. Automate Snort and Barnyard2 Startup
Create a systemd service for Snort:
bash
sudo nano /etc/systemd/system/snort.service
Add the following content:
plaintext
[Unit]
Description=Snort NIDS Daemon
After=network.target
[Service]
ExecStart=/usr/local/bin/snort -c /etc/snort/snort.conf -i eth0
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
[Install]
WantedBy=multi-user.target
Enable and start the service:
bash
sudo systemctl enable snort
sudo systemctl start snort
Create a systemd service for Barnyard2:
bash
sudo nano /etc/systemd/system/barnyard2.service
Add the following content:
plaintext
[Unit]
Description=Barnyard2 Daemon
After=network.target
[Service]
ExecStart=/usr/bin/barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w
/var/log/snort/barnyard2.waldo
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process

```
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

Enable and start the service:

```bash
sudo systemctl enable barnyard2
sudo systemctl start barnyard2
```

Result:

EX.NO:11	DEFEATING MALWARE - BUILDING TROJANS

Aim:

To build a Trojan and know the harmness of the Trojan malwares in a computer system.

Algorithm:

1. Create a simple Trojan by using Windows Batch File (***.bat***)
2. Type these below code in notepad and save it as **Trojan.bat**
3. Double click on ***Trojan.bat*** file.
4. When the Trojan code executes, it will open MS-Paint, Notepad, Command Prompt,Explorer, etc., infinitely.
5. Restart the computer to stop the execution of this Trojan.

TROJAN:

- In computing, a Trojan horse,or Trojan, is any malware which misleads users of itstrue intent.
- Trojans are generally spread by some form of social engineering, for example where a user is duped into executing an email attachment disguised to appear not suspicious, (e.g., a routine form to be filled in), or by clicking on some fake advertisement on social media or anywhere else.
- Although their payload can be anything, many modern forms act as a backdoor, contacting a controller which can then have unauthorized access to the affected computer.
- Trojans may allow an attacker to access users' personal information such as bankinginformation, passwords, or personal identity.
- ***Example:*** *Ransomware* attacks are often carried out using a *trojan*.

Setting Up a Safe Environment

1. Create a Virtual Machine

1. **Download and Install VirtualBox or VMware**: These are popular VM managers.
2. **Create a New Virtual Machine**: Install a Windows operating system on it.

2. Prepare the Virtual Environment

1. **Isolate the VM**: Ensure the VM network settings are set to "Host-only" or disconnected to prevent any potential spread.
2. **Take a Snapshot**: Before starting, take a snapshot of your VM. This allows you to revert to a clean state if needed.

Creating and Running the Batch Script

1. **Open Notepad in the VM**:
   - Press Win + R, type notepad, and press Enter.
2. **Create the Batch Script**:
   - Copy and paste the following code into Notepad:

     ```
     batch
     @echo off
     :x
     start mspaint
     start notepad
     start cmd
     start explorer
     start control
     start calc
     goto x
     ```

3. **Save the Script**:
   - Save the file with a .bat extension, for example, Trojan.bat.
4. **Execute the Script**:
   - Double-click the Trojan.bat file to execute it.

Observing the Behavior

- **Open Task Manager**:
  - Press Ctrl + Shift + Esc to open Task Manager and observe the running processes.
- **Monitor System Performance**:
  - Check CPU and memory usage to see the impact of the script.

Stopping the Script

1. **Open Task Manager**:
   - Press Ctrl + Shift + Esc to open Task Manager.
2. **End the Batch Script Process**:
   - Find the running cmd.exe processes related to the batch script and end them.
3. **Close the Applications**:
   - Manually close any opened applications (MS Paint, Notepad, CMD, Explorer, Control Panel, Calculator).

Clean Up and Restore

1. **Delete the Batch Script**:
   - Delete the Trojan.bat file to prevent accidental re-execution.
2. **Revert the VM**:
   - Revert to the snapshot taken before running the script to ensure the VM is clean.

Program:

Trojan.bat

```
@echo off
:x
start mspaint
start notepad
start cmd
start explorer
start control
start calc
goto x
```

Output:

(MS-Paint, Notepad, Command Prompt, Explorer will open infinitely)

Result:

EX.NO:12	
	DEFEATING MALWARE - ROOTKIT HUNTER

Aim:

To install a rootkit hunter and find the malwares in a computer.

Algorithm:

ROOTKIT HUNTER:

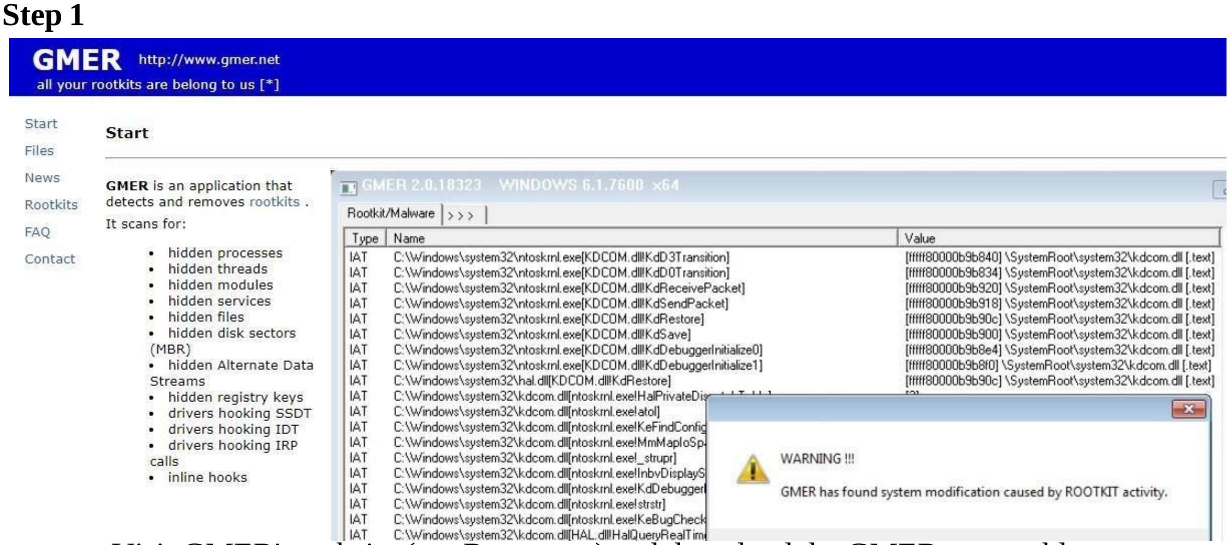
- rkhunter (Rootkit Hunter) is a Unix-based tool that scans for rootkits, backdoors andpossible local exploits.
- It does this by comparing SHA-1 hashes of important files with known good ones in online databases, searching for default directories (of rootkits), wrong permissions, hidden files, suspicious strings in kernel modules, and special tests for Linux and FreeBSD.
- rkhunter is notable due to its inclusion in popular operating systems (Fedora, Debian,etc.)
- The tool has been written in Bourne shell, to allow for portability. It can run on almostall UNIX-derived systems.

GMER ROOTKIT TOOL:

- GMER is a software tool written by a Polish researcher Przemysław Gmerek, fordetecting and removing rootkits.
- It runs on Microsoft Windows and has support for Windows NT, 2000, XP, Vista, 7, 8

and 10. With version 2.0.18327 full support for Windows x64 is added

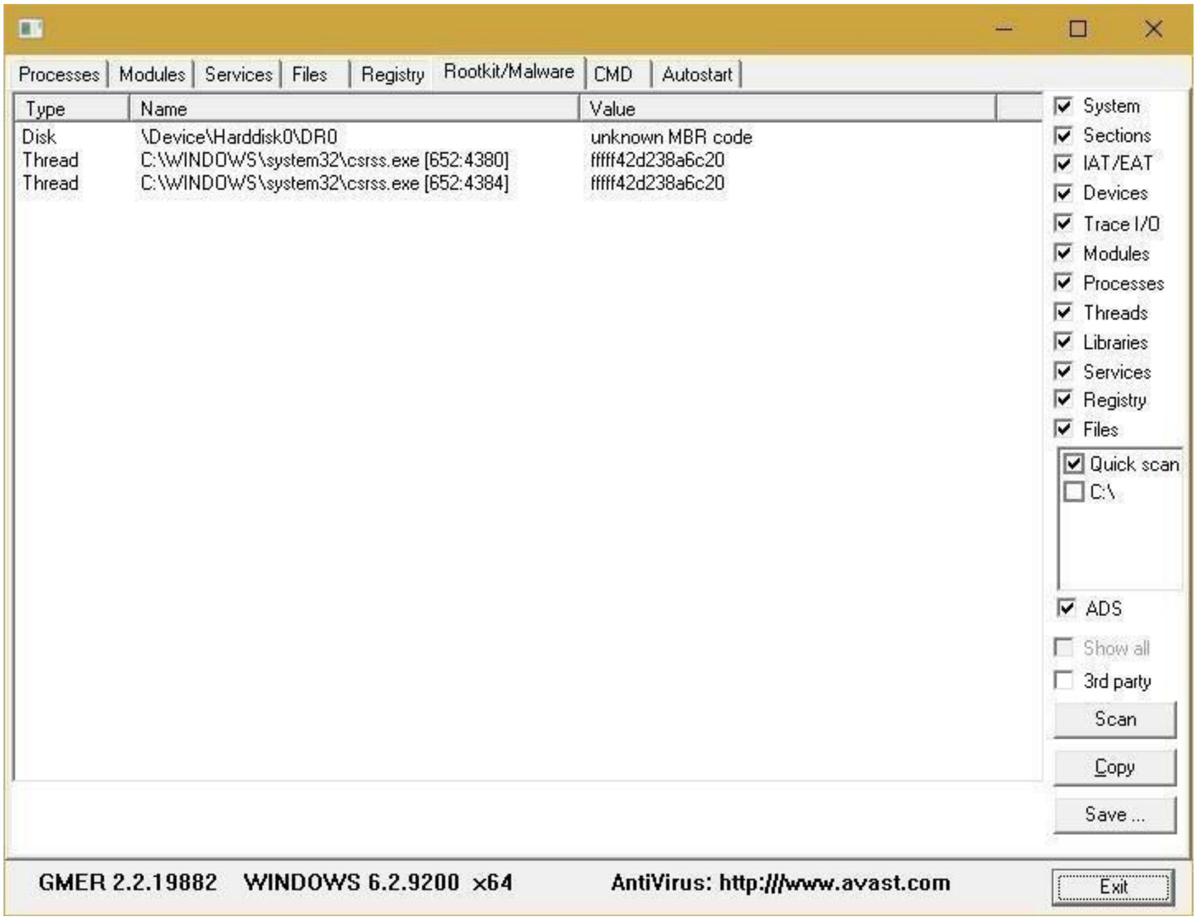
Step 1



Visit GMER's website (see Resources) and download the GMER executable.

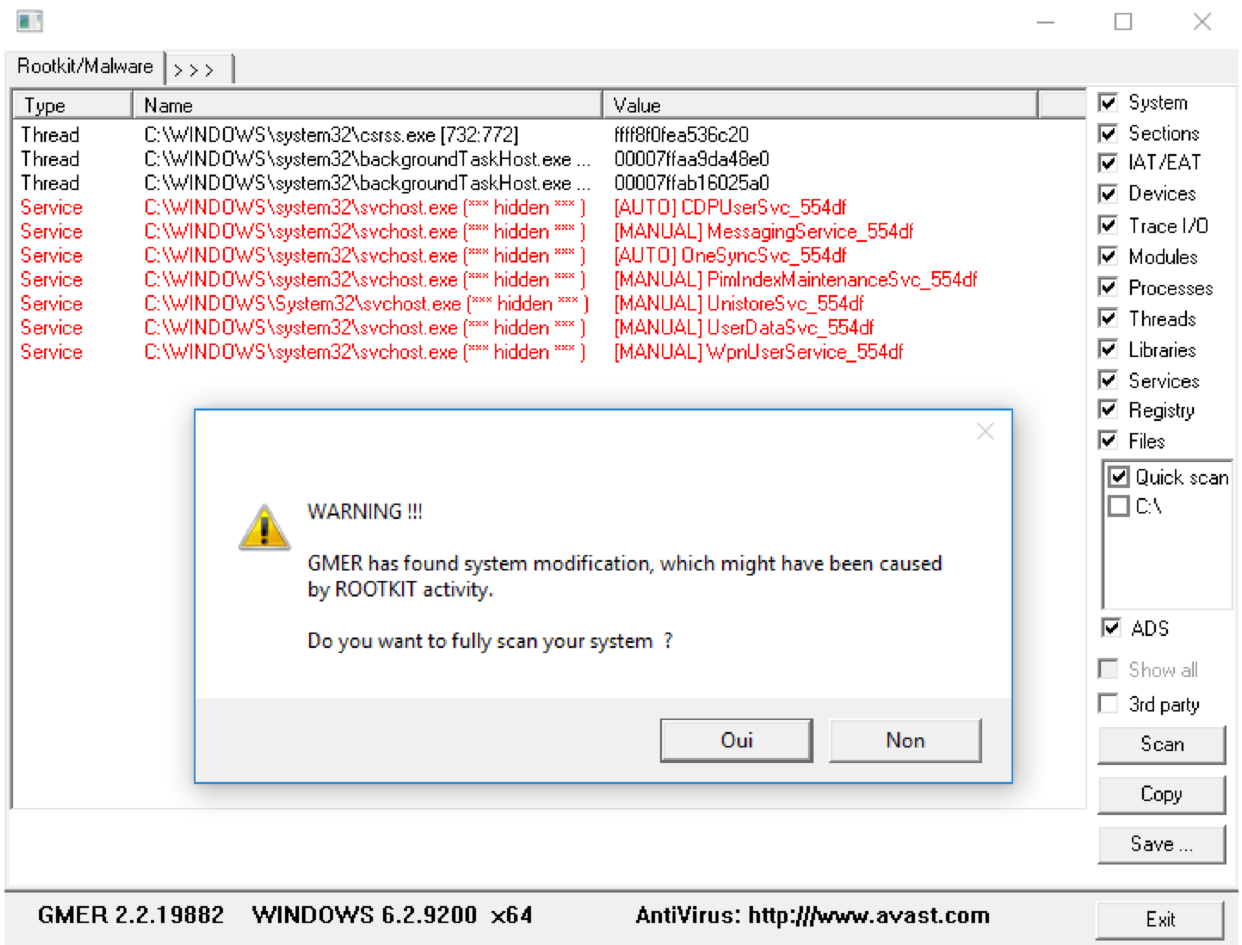
Click the "Download EXE" button to download the program with a random file name, assome rootkits will close “gmer.exe” before you can open it.

Step 2



Double-click the icon for the program.
Click the "Scan" button in the lower-right corner ofthe dialog box. Allow the program toscan your entire hard drive.

Step 3



When the program completes its scan, select any program or file listed in red. Right-click it and select "Delete."

If the red item is a service, it may be protected. Right-click the service and select "Disable." Reboot your computer and run the scan again, this time selecting "Delete" when that serviceis detected.

When your computer is free of Rootkits, close the program and restart your PC.

RESULT:

A rootkit hunter software tool *gmer* has been installed and the rootkits have beendetected.

EX.NO:13	Implement Database Security

Aim:

To implementing database security mechanisms, specifically access control and authentication, on a Windows OS

Algorithm / Program:

1. **Install Microsoft SQL Server**: Ensure that Microsoft SQL Server is installed on your Windows machine. You can download it from the Microsoft SQL Server website.
2. **Install SQL Server Management Studio (SSMS)**: This is a graphical tool for managing SQL Server instances. You can download it from the SSMS download page.

Step-by-Step Guide :

This environment involves several steps. Here, we'll use Microsoft SQL Server as an example database management system to demonstrate how to set up and secure a database on Windows.

1. Understanding Database Security Threats

Common security threats to databases include:

- Unauthorized access
- SQL injection attacks
- Privilege escalation
- Data breaches due to weak authentication
- Insider threats

2. Implementing Access Control

Access control involves defining who can access the database and what actions they can perform. This is typically done using roles and permissions.

1. **Create Database and User Roles**:
    - Open SQL Server Management Studio (SSMS).
    - Connect to your SQL Server instance.
    - Create a new database:

      sql

      ```
      CREATE DATABASE SecureDB;
      GO
      ```

2. **Create Users and Roles**:
    - Create a user with administrative privileges:

      sql
      ```
      USE SecureDB;
      CREATE LOGIN admin_user WITH PASSWORD = 'StrongPassword';
      CREATE USER admin_user FOR LOGIN admin_user;
      EXEC sp_addrolemember 'db_owner', 'admin_user';
      GO
      ```

    - Create a read-only user:

      sql
      ```
      USE SecureDB;
      CREATE LOGIN read_only_user WITH PASSWORD = 'ReadOnlyPassword';
      CREATE USER read_only_user FOR LOGIN read_only_user;
      EXEC sp_addrolemember 'db_datareader', 'read_only_user';
      GO
      ```

3. **Verify Access Control**:
    - Log in as read_only_user and try to perform write operations to confirm that they are restricted.

*3. **Implementing Authentication***

Authentication ensures that only authorized users can access the database.

1. **Enforce Strong Password Policies**:
    - Open SQL Server Management Studio (SSMS).
    - Connect to your SQL Server instance.
    - Set up strong password policies:

      sql
      Copy code
      ```
      ALTER LOGIN admin_user WITH CHECK_POLICY = ON,
      CHECK_EXPIRATION = ON;
      ALTER LOGIN read_only_user WITH CHECK_POLICY = ON,
      CHECK_EXPIRATION = ON;
      ```

2. **Enable Windows Authentication**:
    - SQL Server supports both SQL Server authentication and Windows authentication. Windows authentication is generally more secure as it integrates with Windows Active Directory.
    - In SSMS, navigate to **Security > Logins**.
    - Right-click and select **New Login**.
    - Choose **Windows authentication** and specify the Windows user or group.

3. **Configure SQL Server for Mixed Mode Authentication (if needed)**:
    - Open SQL Server Configuration Manager.
    - Navigate to **SQL Server Services**.
    - Right-click on the SQL Server instance and select **Properties**.
    - In the **Security** tab, choose **SQL Server and Windows Authentication mode**.
    - Restart the SQL Server service for the changes to take effect.

4. Testing and Verification

1. **Test Access Control**:
    - Ensure users have the correct permissions and cannot access or modify data beyond their privileges.
2. **Test Authentication Mechanisms**:
    - Attempt to log in with weak passwords and verify that access is denied.
    - Ensure that both SQL Server and Windows Authentication modes work as expected.
3. **Audit and Logging**:
    - Enable SQL Server auditing to monitor access and detect any unauthorized attempts.

```
sql
CREATE SERVER AUDIT AuditTest
TO FILE (FILEPATH = 'C:\Audit\');
ALTER SERVER AUDIT AuditTest WITH (STATE = ON);
GO
CREATE DATABASE AUDIT SPECIFICATION AuditSpec
FOR SERVER AUDIT AuditTest
ADD (SELECT ON DATABASE::SecureDB BY read_only_user),
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::SecureDB BY
admin_user);
ALTER DATABASE AUDIT SPECIFICATION AuditSpec WITH (STATE =
ON);
GO
```

Output :

```
SELECT name FROM sys.databases;
name
----
master
tempdb
model
msdb
SecureDB
```

```
DatabaseRoleName  DatabaseUserName
---------------- ----------------
db_owner          admin_user
```

Result:

EX.NO:14	**Implement Encryption and Integrity Control-Database Security**

Aim:

To implementing encryption and integrity controls for databases is crucial to protect sensitive data and ensure that it remains unaltered.

Algorithm /Program:

1. **Microsoft SQL Server**: Ensure that SQL Server is installed on your Windows system.
2. **SQL Server Management Studio (SSMS)**: Ensure SSMS is installed for managing the SQL Server instance.

Steps to Implement Encryption and Integrity Controls

1. Transparent Data Encryption (TDE)

Transparent Data Encryption (TDE) helps protect data at rest by encrypting the database files. This ensures that the database files are not readable if accessed directly from the disk.

1. **Create a Master Key**

   The master key is required to encrypt the database encryption key.

   ```
   sql
   USE master;
   GO
   CREATE MASTER KEY ENCRYPTION BY PASSWORD =
   'StrongPasswordForMasterKey';
   GO
   ```

2. **Create a Certificate**

   The certificate is used to protect the database encryption key.

   ```
   sql
   USE master;
   GO
   CREATE CERTIFICATE TDE_Cert WITH SUBJECT = 'TDE Certificate';
   GO
   ```

3. **Create a Database Encryption Key**

The database encryption key is used to encrypt the database.

```sql
USE SecureDB;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE_Cert;
GO
```

4. **Enable TDE on the Database**

```sql
ALTER DATABASE SecureDB
SET ENCRYPTION ON;
GO
```

5. **Verify Encryption**

```sql
USE SecureDB;
GO
SELECT name, is_encrypted
FROM sys.databases
WHERE name = 'SecureDB';
GO
```

Expected Output:

  - is_encrypted should be 1 for the SecureDB database.

2. Column-Level Encryption

Column-level encryption provides fine-grained control over the encryption of specific data within a table.

1. **Create a Symmetric Key**

```sql
USE SecureDB;
GO
CREATE SYMMETRIC KEY SymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE TDE_Cert;
GO
```

2. **Encrypt Data in a Table**
    - Create a table and insert some data:

      ```sql
      CREATE TABLE SensitiveData (
          ID INT PRIMARY KEY,
          SensitiveInfo VARBINARY(MAX)
      );
      GO

      OPEN SYMMETRIC KEY SymmetricKey
      DECRYPTION BY CERTIFICATE TDE_Cert;

      INSERT INTO SensitiveData (ID, SensitiveInfo)
      VALUES (1, ENCRYPTBYKEY(KEY_GUID('SymmetricKey'), 'Sensitive
      Information'));
      GO
      CLOSE SYMMETRIC KEY SymmetricKey;
      ```

3. **Decrypt Data for Viewing**

   ```sql
   OPEN SYMMETRIC KEY SymmetricKey
   DECRYPTION BY CERTIFICATE TDE_Cert;

   SELECT ID, CONVERT(VARCHAR(MAX), DECRYPTBYKEY(SensitiveInfo)) AS
   SensitiveInfo
   FROM SensitiveData;
   GO

   CLOSE SYMMETRIC KEY SymmetricKey;
   ```

   Expected Output:

    - The SensitiveInfo column should display the decrypted data.

3. Data Integrity Controls

Implementing data integrity controls ensures that the data is not tampered with and maintains its accuracy and consistency.

1. **Using Hashes for Data Integrity**
    - Create a table to store hashed data:

      ```sql
      CREATE TABLE DataIntegrity (
      ```

```
    ID INT PRIMARY KEY,
    OriginalData NVARCHAR(255),
    DataHash VARBINARY(64)
);
GO
```

- Insert data with a hash:

```sql
INSERT INTO DataIntegrity (ID, OriginalData, DataHash)
VALUES (1, 'Important Data', HASHBYTES('SHA2_256', 'Important Data'));
GO
```

- Verify data integrity:

```sql
DECLARE @OriginalData NVARCHAR(255);
DECLARE @Hash VARBINARY(64);

SELECT @OriginalData = OriginalData, @Hash = DataHash
FROM DataIntegrity
WHERE ID = 1;

IF @Hash = HASHBYTES('SHA2_256', @OriginalData)
    PRINT 'Data integrity verified.';
ELSE
    PRINT 'Data has been tampered with.';
GO
```

2. **Expected Output:**
    - Data integrity verified. Should be printed if the data has not been altered.

Output:

COLUMN_NAME	DATA_TYPE
------------	----------
ID	int
OriginalData	nvarchar
DataHash	varbinary

ID	OriginalData	DataHash
--	---------------	----------------------------------
1	Important Data	0A6D3A6E1C5F4A12F8A4F5F6E7D8A9B7C6D7E8E7A9B7D6E8A9

Result: