AIM: Write a program to implement the RSA algorithm logic.

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using RSA algorithm.
RSA encryption algorithm is a type of public-key encryption algorithm. The Public key is used for encryption, and the Private Key is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and it is known only to the user who owns the key.

ALGORITHM:
Step-1: Start
Step-2: Select two large prime numbers, p and q.
Step-3: Multiply these numbers to find n = p x q, where n is called the modulus for encryption and decryption.
Step-4: Choose a number e less than n, such that n is relatively prime to (p - 1) x (q -1). It means that e and (p - 1) x (q - 1) have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), gcd (e,d(n)) =1
Step-5: If n = p x q, then the public key is <e, n>. A plaintext message m is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
$C = m^e \bmod n$
Here, m must be less than n. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
Step-6: To determine the private key, we use the following formula to calculate the d such that: $D_e \bmod \{(p - 1) \times (q - 1)\} = 1$
Or $D_e \bmod \varphi(n) = 1$
Step-7: The private key is <d, n>. A ciphertext message c is decrypted using private key <d, n>. To calculate plain text m from the ciphertext c following formula is used to get plain text m.
$m = c^d \bmod n$
Step-8: End

PROGRAM:
```
import math
def gcd(a, h):
```

Signature of the Faculty……...

```
        temp = 0
        while(1):
                temp = a % h
                if (temp == 0):
                        return h
                a = h
                h = temp
p = int(input("Enter a prime number: "))
q = int(input("Enter another prime number: "))
n = p*q
e = 2
phi = (p-1)*(q-1)
while (e < phi):
        if(gcd(e, phi) == 1):
                break
        else:
                e = e+1
d=2
while (d < phi):
        if (((d*e)%phi)==1):
                break
        else:
                d=d+1
print("Public Key: (", e, ", ", n,")")
print("Private Key: (", d, ", ", n,")")
msg = int(input("Enter the message to be encrypted: "))
print("Message data =", msg)
c = pow(msg, e)
c = c%n
c=int(c)
print("Encrypted data = ", c)
m = pow(c, d)
m = m%n
print("Original Message Sent = ", m)
```

OUTPUT:

```
= RESTART: D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Progr
ams\rsa.py
Enter a prime number: 17
Enter another prime number: 11
Public Key: ( 3 ,  187 )
Private Key: ( 107 ,  187 )
Enter the message to be encrypted: 88
Message data =  88
Encrypted data =  44
Original Message Sent =  88
```

Signature of the Faculty……...

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using RSA algorithm.

Signature of the Faculty……...

AIM: Write a program to implement the Diffie Hellman key exchange algorithm logic.

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Diffie Hellman key exchange algorithm.
Diffie Hellman key-exchange algorithm is a type of public-key encryption algorithm. The Diffie–Hellman (DH) Algorithm is a key-exchange protocol that enables two parties communicating over public channel to establish a mutual secret without it being transmitted over the Internet. DH enables the two to use a public key to encrypt and decrypt their conversation or data using symmetric cryptography.

ALGORITHM:
Step-1: Start
Step-2: Alice and Bob agree on two large prime numbers, p and g, and a public key exchange algorithm.
Step-3: Alice chooses a secret integer, a, and computes A = g^a mod p. She sends A to Bob.
Step-4: Bob chooses a secret integer, b, and computes B = g^b mod p. He sends B to Alice.
Step-5: Alice computes s = B^a mod p. Bob computes s = A^b mod p.
Step-6: Alice and Bob now both have the shared secret key s, which they can use to establish a secure communication channel.
Step-7: End

PROGRAM:

```
def prime_checker(p):
    if p < 1:
        return -1
    elif p > 1:
        if p == 2:
            return 1
        for i in range(2, p):
            if p % i == 0:
                return -1
            return 1
def primitive_check(g, p, L):
    for i in range(1, p):
```

SANYA BHARATHI INSTITUTE OF TECHNOLOGY
స్వయం తేజస్విన్ భవ
1979

```
            L.append(pow(g, i) % p)
        for i in range(1, p):
            if L.count(i) > 1:
                L.clear()
                return -1
            return 1
l = []
while 1:
        P = int(input("Enter P : "))
        if prime_checker(P) == -1:
            print("Number Is Not Prime, Please Enter Again!")
            continue
        break
while 1:
        G = int(input(f"Enter The Primitive Root Of {P} : "))
        if primitive_check(G, P, l) == -1:
            print(f"Number Is Not A Primitive Root Of {P}, Please
Try Again!")
            continue
        break
x1, x2 = int(input("Enter The Private Key Of User 1 : ")), int(
        input("Enter The Private Key Of User 2 : "))
while 1:
        if x1 >= P or x2 >= P:
            print(f"Private Key Of Both The Users Should Be Less
Than {P}!")
            continue
        break
y1, y2 = pow(G, x1) % P, pow(G, x2) % P
print("Private and Public keys of User 1 = ", x1, " ", y1)
print("Private and Public keys of User 2 = ", x2, " ", y2)
k1, k2 = pow(y2, x1) % P, pow(y1, x2) % P
print(f"\nSecret Key For User 1 Is {k1}\nSecret Key For User 2 Is
{k2}\n")
if k1 == k2:
        print("Keys Have Been Exchanged Successfully")
else:
        print("Keys Have Not Been Exchanged Successfully")
```

OUTPUT:

```
= RESTART: D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Progr
ams\dha.py
Enter P : 17
Enter The Primitive Root Of 17 : 3
Enter The Private Key Of User 1 : 15
Enter The Private Key Of User 2 : 13
Private and Public keys of User 1 =  15   6
Private and Public keys of User 2 =  13   12

Secret Key For User 1 Is 10
Secret Key For User 2 Is 10

Keys Have Been Exchanged Successfully
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Diffie Hellman Key Exchange algorithm.

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
స్వయం తేజస్విన్ భవ
1979

AIM: Write a program to implement the Secure Hash algorithm (SHA-1) logic.

DESCRIPTION: In this program, we try to calculate the message digest of the text using SHA-1 algorithm.
SHA-1 or Secure Hash Algorithm 1 is a cryptographic algorithm which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long.

ALGORITHM:
Step-1: Start
Step-2: Append padding bits − The original message is padded and its duration is congruent to 448 modulo 512. Padding is continually inserted although the message already has the desired length. Padding includes a single 1 followed by the essential number of 0 bits.
Step-3: Append length − A 64-bit block considered as an unsigned 64-bit integer (most essential byte first), and defining the length of the original message (before padding in step 1), is added to the message. The complete message's length is a multiple of 512.
Step-4: Initialize the buffer − The buffer includes five (5) registers of 32 bits each indicated by A, B, C, D, and E. This 160-bit buffer can be used to influence temporary and final outcomes of the compression function. These five registers are initialized to the following 32-bit integers (in hexadecimal notation).
A = 67 45 23 01
B = ef cd ab 89
C = 98 ba dc fe
D = 10 32 54 76
E = c3 d2 e1 f0
The registers A, B, C, and D are actually the same as the four registers used in MD5 algorithm. But in SHA-1, these values are saved in big-endian format, which define that the most essential byte of the word is located in the low-address byte position. Therefore, the initialization values (in hexadecimal notation) occurs as follows −
word A = 67 45 23 01
word B = ef cd ab 89
word C = 98 ba dc fe

word D = 10 32 54 76
word E = c3 d2 e1 f0
Step-5: Process message in 512-bit blocks − The compression function is divided into 20 sequential steps includes four rounds of processing where each round is made up of 20 steps.
The four rounds are structurally same as one another with the only difference that each round need a different Boolean function, which it can define as f1, f2, f3, f4 and one of four multiple additive constants Kt (0 ≤t ≤79) which is based on the step under consideration.
Step-6: Output − After processing the final 512-bit message block t (considering that the message is divided into t 512-bit blocks), and it can obtain a 160-bit message digest.
Step-7: End

PROGRAM:

```
def sha1(data):
    bytes = ""
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0
    for n in range(len(data)):
        bytes+='{0:08b}'.format(ord(data[n]))
    bits = bytes+"1"
    pBits = bits
    while len(pBits)%512 != 448:
        pBits+="0"
    pBits+='{0:064b}'.format(len(bits)-1)
    def chunks(l, n):
        return [l[i:i+n] for i in range(0, len(l), n)]
    def rol(n, b):
        return ((n << b) | (n >> (32 - b))) & 0xffffffff
    for c in chunks(pBits, 512):
        words = chunks(c, 32)
        w = [0]*80
        for n in range(0, 16):
            w[n] = int(words[n], 2)
        for i in range(16, 80):
            w[i] = rol((w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16]), 1)
        a = h0
        b = h1
```

```
        c = h2
        d = h3
        e = h4
        for i in range(0, 80):
            if 0 <= i <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= i <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= i <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            elif 60 <= i <= 79:
                f = b ^ c ^ d
                k = 0xCA62C1D6
            temp = rol(a, 5) + f + e + k + w[i] & 0xffffffff
            e = d
            d = c
            c = rol(b, 30)
            b = a
            a = temp
        h0 = h0 + a & 0xffffffff
        h1 = h1 + b & 0xffffffff
        h2 = h2 + c & 0xffffffff
        h3 = h3 + d & 0xffffffff
        h4 = h4 + e & 0xffffffff
    return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)
str=input("Enter the string: ")
print (sha1(str))
```

BHARATHI INSTITUTE OF TECHNOLOGY
TM
స్వయం తేజస్విన్ భవ
1979

OUTPUT:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python sha.py
Enter the string: helloworld
6adfb183a4a2c94a2f92dab5ade762a47889a5a1
```

CONCLUSION:

By executing the above, we have successfully calculated the message digest of the string using Secure Hash Algorithm (SHA-1).

TM
CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
స్వయం తేజస్విన్ భవ
1979

AIM: Write a program to implement the MD5 logic.

DESCRIPTION: In this program, we try to calculate the message digest of the text using MD5 algorithm.
MD5 is a cryptographic hash function algorithm that takes the message as input of any length and changes it into a fixed-length message of 16 bytes. MD5 algorithm stands for the message-digest algorithm. MD5 was developed as an improvement of MD4, with advanced security purposes. The output of MD5 (Digest size) is always 128 bits.

ALGORITHM:
Step-1: Start
Step-2: Append Padding Bits: In the first step, we add padding bits in the original message in such a way that the total length of the message is 64 bits less than the exact multiple of 512.
Length(original message + padding bits) = 512 * i – 64 where i = 1,2,3 . . .
Step-3: Append Length Bits: In this step, we add the length bit in the output of the first step in such a way that the total number of the bits is the perfect multiple of 512. Simply, here we add the 64-bit as a length bit in the output of the first step. i.e. output of first step = 512 * n – 64 length bits = 64.
After adding both we will get 512 * n i.e., the exact multiple of 512.
Step-4: Initialize MD buffer: Here, we use the 4 buffers i.e., J, K, L, and M. The size of each buffer is 32 bits.
- J = 0x67425301
- K = 0xEDFCBA45
- L = 0x98CBADFE
- M = 0x13DCE476

Step-5: Process Each 512-bit Block: This is the most important step of the MD5 algorithm. Here, a total of 64 operations are performed in 4 rounds. In the 1st round, 16 operations will be performed, 2nd round 16 operations will be performed, 3rd round 16 operations will be performed, and in the 4th round, 16 operations will be performed. We apply a different function on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function.
We perform OR, AND, XOR, and NOT (basically these are logic

gates) for calculating functions. We use 3 buffers for each function i.e. K, L, M.

- F(K,L,M) = (K AND L) OR (NOT K AND M)
- G(K,L,M) = (K AND L) OR (L AND NOT M)
- H(K,L,M) = K XOR L XOR M
- I(K,L,M) = L XOR (K OR NOT M)

After applying the function now, we perform an operation on each block. For performing operations, we need

- add modulo 2^{32}
- M[i] – 32-bit message.
- K[i] – 32-bit constant.
- <<<n – Left shift by n bits.

Now take input as initialize MD buffer i.e. J, K, L, M. Output of K will be fed in L, L will be fed into M, and M will be fed into J. After doing this now we perform some operations to find the output for J.

- In the first step, Outputs of K, L, and M are taken and then the function F is applied to them. We will add modulo 2^{32} bits for the output of this with J.
- In the second step, we add the M[i] bit message with the output of the first step.
- Then add 32 bits constant i.e. K[i] to the output of the second step.
- At last, we do left shift operation by n (can be any value of n) and addition modulo by 2^{32}.

After all steps, the result of J will be fed into K. Now same steps will be used for all functions G, H, and I. After performing all 64 operations we will get our message digest.

Step-6: After all, rounds have been performed, the buffer J, K, L, and M contains the MD5 output starting with the lower bit J and ending with Higher bits M.

Step-7: End

PROGRAM:

```
import struct
def left_rotate(x, c):
    return (x << c) | (x >> (32 - c))
def F(x, y, z):
    return (x & y) | ((~x) & z)
def G(x, y, z):
    return (x & z) | (y & (~z))
def H(x, y, z):
    return x ^ y ^ z
def I(x, y, z):
    return y ^ (x | (~z))
def md5(data):
    A = 0x67452301
    B = 0xEFCDAB89
    C = 0x98BADCFE
    D = 0x10325476
    K = [0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE,
         0xF57C0FAF, 0x4787C62A, 0xA8304613, 0xFD469501,
         0x698098D8, 0x8B44F7AF, 0xFFFF5BB1, 0x895CD7BE,
         0x6B901122, 0xFD987193, 0xA679438E, 0x49B40821,
         0xF61E2562, 0xC040B340, 0x265E5A51, 0xE9B6C7AA,
         0xD62F105D, 0x02441453, 0xD8A1E681, 0xE7D3FBC8,
         0x21E1CDE6, 0xC33707D6, 0xF4D50D87, 0x455A14ED,
         0xA9E3E905, 0xFCEFA3F8, 0x676F02D9, 0x8D2A4C8A,
         0xFFFA3942, 0x8771F681, 0x6D9D6122, 0xFDE5380C,
         0xA4BEEA44, 0x4BDECFA9, 0xF6BB4B60, 0xBEBFBC70,
         0x289B7EC6, 0xEAA127FA, 0xD4EF3085, 0x04881D05,
         0xD9D4D039, 0xE6DB99E5, 0x1FA27CF8, 0xC4AC5665,
         0xF4292244, 0x432AFF97, 0xAB9423A7, 0xFC93A039,
         0x655B59C3, 0x8F0CCC92, 0xFFEFF47D, 0x85845DD1,
         0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1,
         0xF7537E82, 0xBD3AF235, 0x2AD7D2BB, 0xEB86D391]
    bit_len = len(data) * 8
    data += b'\x80'
    while (len(data) + 8) % 64 != 0:
        data += b'\x00'
    data += struct.pack('<Q', bit_len)
    for i in range(0, len(data), 64):
        chunk = data[i:i + 64]
        M = struct.unpack('<16I', chunk)
        AA, BB, CC, DD = A, B, C, D
        for j in range(16):
```

```
            A = B + left_rotate((A + F(B, C, D) + M[j] + K[j]), 7)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (1 + 5 * j) % 16
            A = B + left_rotate((A + G(B, C, D) + M[k] + K[j + 16]), 12)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (5 + 3 * j) % 16
            A = B + left_rotate((A + H(B, C, D) + M[k] + K[j + 32]), 17)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (7 * j) % 16
            A = B + left_rotate((A + I(B, C, D) + M[k] + K[j + 48]), 22)
            A, B, C, D = D, A, B, C

        A, B, C, D = (A + AA) & 0xFFFFFFFF, (B + BB) &
0xFFFFFFFF, (C + CC) & 0xFFFFFFFF, (D + DD) & 0xFFFFFFFF
    result = ''.join(f'{x:08x}' for x in (A, B, C, D))
    return result
message = input("Enter the string: ")
hashed = md5(message.encode('utf-8'))
print("MD5 Hash:", hashed)
```

OUTPUT:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python md5.py
Enter the string: helloworld
MD5 Hash: ed9a2c9647a7f145b7afdf49925c3cb1
```

CONCLUSION:

By executing the above, we have successfully calculated the message digest of the string using MD5 algorithm.

AIM: Write a program to implement Simple Columnar Transposition Technique

DESCRIPTION: In this program, we try to encrypt and decrypt the text given by the user using Simple Columnar Technique.
Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

ALGORITHM:
Step-1: Start
Step-2: Write the plain text message row-by-row in grids of rectangle of a pre-defined size.
Step-3: Read the message column by column. However, it need to be in the order of column 1,2,3,4,5,6 etc.
Step-4: After this operation message obtain is cipher text message.
Step-5: Simple columnar transposition technique simple arranges the plain text as a sequence of row of rectangle that is read in column randomly.
Step-6: End

PROGRAM:

```
import math
def encrypt(key, message):
    ciphertext = [''] * key
    for column in range(key):
        index = column
        while index < len(message):
            ciphertext[column] += message[index]
            index += key
    return ''.join(ciphertext)
def decrypt(key, message):
    nrows = key
    ncols = math.ceil(len(message) / key)
    empty_positions = nrows * ncols - len(message)
    plaintext = [''] * ncols
    column = 0
    row = 0
    for symbol in message:
        plaintext[column] += symbol
```

Signature of the Faculty……...

```
        column += 1
        if column == ncols or (column == ncols - 1 and row >=
nrows - empty_positions):
            column = 0
            row += 1
    return ''.join(plaintext)
message = input("Enter the message: ")
key = int(input("Enter the key: "))
ciphertext = encrypt(key, message)
print(f'Ciphertext: {ciphertext}<end>')
plaintext = decrypt(key, ciphertext)
print(f'Plaintext: {plaintext}')
```

OUTPUT:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python SC.py
Enter the message: HelloWorldCNSLAB
Enter the key: 4
Ciphertext: HolSeWdLloCAlrNB<end>
Plaintext: HelloWorldCNSLAB
```

CONCLUSION:

By executing the above, we have successfully encrypted and decrypted the text given by the user using Simple Columnar Transposition technique.

TM
INSTITUTE OF TECHNOLOGY
స్వయం తేజస్విన్ భవ
1979

AIM: Write a program to implement Advanced Columnar Transposition Technique

DESCRIPTION: In this program, we try to encrypt and decrypt the text given by the user using Advanced Columnar Technique. Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one. In Advanced Columnar Transposition technique, the process is performed more than once making the technique more complex

ALGORITHM:

Step-1: Start
Step-2: Write the plain text message row-by-row in grids of rectangle of a pre-defined size.
Step-3: Read the message column by column. However, it needs to be in the order of column 1,2,3,4,5,6 etc.
Step-4: After this operation message obtain is cipher text message.
Step-5: Advanced columnar transposition technique performs the above steps more than once i.e., the number of times given by the user.
Step-6: End

PROGRAM:

```
import math
def encrypt(key, message):
    ciphertext = [''] * key
    for column in range(key):
        index = column
        while index < len(message):
            ciphertext[column] += message[index]
            index += key
    return ''.join(ciphertext)
def decrypt(key, message):
    nrows = key
    ncols = math.ceil(len(message) / key)
    empty_positions = nrows * ncols - len(message)
    plaintext = [''] * ncols
    column = 0
    row = 0
```

Signature of the Faculty……...

```
    for symbol in message:
        plaintext[column] += symbol
        column += 1
        if column == ncols or (column == ncols - 1 and row >=
nrows - empty_positions):
            column = 0
            row += 1
    return ''.join(plaintext)
message = input("Enter the message: ")
key = int(input("Enter the key: "))
loop = int(input("Enter the number of the times to be repeated: "))
for i in range(loop):
    ciphertext = encrypt(key, message)
print(f'Ciphertext: {ciphertext}<end>')
for i in range(loop):
    plaintext = decrypt(key, ciphertext)
print(f'Plaintext: {plaintext}')
```

OUTPUT:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python ASC.py
Enter the message: HelloWorldCNSLAB
Enter the key: 5
Enter the number of the times to be repeated: 4
Ciphertext: HWCBeoNlrSllLodA<end>
Plaintext: HelloWorldCNSLAB
```

CONCLUSION:

By executing the above, we have successfully encrypted and decrypted the text given by the user using Advanced Columnar Transposition technique.

స్వయం తేజస్విన్ భవ
1979

AIM: Write a program to implement Euclidean Algorithm and Advanced Euclidean Algorithm

DESCRIPTION: In this program, we try to implement Euclidean Algorithm and Advanced Euclidean Algorithm.
The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them.

ALGORITHM:

Basic Euclidean Algorithm:

Step-1: Start
Step-2: Define a function gcd(a, b) that takes two integers, a and b, as input.
    Step-2.1: Check if a is equal to 0:
        Step-2.1.1: If a is equal to 0, return b as the GCD.
    Step-2.2: If a is not equal to 0:
        Step-2.2.1: Recursively call the gcd function with the arguments b % a and a.
        Step-2.2.2: Repeat steps 2.1 and 2.2 until a becomes 0.
Step-3: Read two integers a and b from the user as input.
Step-4: Call the gcd function with a and b as arguments to calculate the GCD.
Step-5: Print the result, indicating that it's the GCD of the two input numbers.
Step-6: End

Extended Euclidean Algorithm:

Step-1: Start
Step-2: Define a function gcdExtended(a, b) that takes two positive integers, a and b, as input.
    Step-2.1: Check if a is equal to 0:
        Step-2.1.1: If a is equal to 0, return b as the GCD, and the Bézout coefficients as (0, 1).
    Step-2.2: If a is not equal to 0:
        Step-2.2.1: Recursively call the gcdExtended function with the arguments (b % a, a) and store the result in variables gcd, x1, and y1.
    Step-2.3: Calculate the Bézout coefficients x and y as follows:
        Step-2.3.1: x is equal to y1 - (b // a) * x1.

Step-2.3.2: y is equal to x1.
Step-2.4: Return the GCD gcd, x, and y.
Step-3: Read two positive integers a and b from the user as input.
Step-4: Call the gcdExtended function with a and b as arguments to calculate the GCD and Bézout coefficients.
Step-5: Print the GCD and the Bézout coefficients.
Step-6: End

PROGRAM:

Basic Euclidean Algorithm:

```
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)
if __name__ == "__main__":
  a = int(input("Enter a number: "))
  b = int(input("Enter another number: "))
  print("gcd(", a, ",", b, ") = ", gcd(a, b))
```

Advanced Euclidean Algorithm:

```
def gcdExtended(a, b):
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = gcdExtended(b % a, a)
    x = y1 - (b//a) * x1
    y = x1
    return gcd, x, y
a = int(input("Enter a number: "))
b = int(input("Enter another number: "))
g, x, y = gcdExtended(a, b)
print("gcd(", a, ",", b, ") = ", g)
```

TM
BHARATHI
INSTITUTE OF TECHNOLOGY
1979

OUTPUT:

Basic Euclidean Algorithm:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python BEA.py
Enter a number: 255
Enter another number: 135
gcd( 255 , 135 ) =  15
```

Extended Euclidean Algorithm:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python AEA.py
Enter a number: 435
Enter another number: 565
gcd( 435 , 565 ) =  5
```

CONCLUSION:

By executing the above, we have successfully implemented Euclidean Algorithm and Advanced Euclidean Algorithm.

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
స్వయం తేజస్విన్ భవ
1979

AIM: To Familiarize the cryptographic tools.

DESCRIPTION:

Here, we try to explore cryptographic tools OpenCV (Open Source Computer Vision Library) is a popular library for computer vision and image processing. It provides a wide range of functions for tasks such as image acquisition, feature detection and tracking, image segmentation, and machine learning. OpenCV also includes a number of cryptographic tools, which can be used to encrypt and decrypt images and videos. These tools can be used to protect sensitive data from unauthorized access, or to add security to multimedia applications.

1) OpenSSL

DESCRIPTION

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and related cryptography standards required by them.
The OpenSSL program is a command line program for using the various cryptography functions of OpenSSL's crypto library from the shell.
It can be used for
-Creation and management of private keys, public keys and parameters
-Public key cryptographic operations
-Creation of X.509 certificates, CSRs and CRLs
-Calculation of Message Digests and Message Authentication Codes
-Encryption and Decryption with Ciphers
-SSL/TLS Client and Server Tests
-Handling of S/MIME signed or encrypted mail
-Timestamp requests, generation and verification

INSTALLATION:

Step-1: Update the system

sudo apt update

Step-2: Check if OpenSSL is already installed or not

openssl version -a

Step-3: Install OpenSSL via APT

OpenSSL is available in the default repository of all distributions, and can be installed on Debian-based systems with:

sudo apt install openssl

All of the requirements will be installed automatically, so you just need to confirm the installation and wait a few minutes.

Once done, you can try again the previous command to make sure the installation is complete:

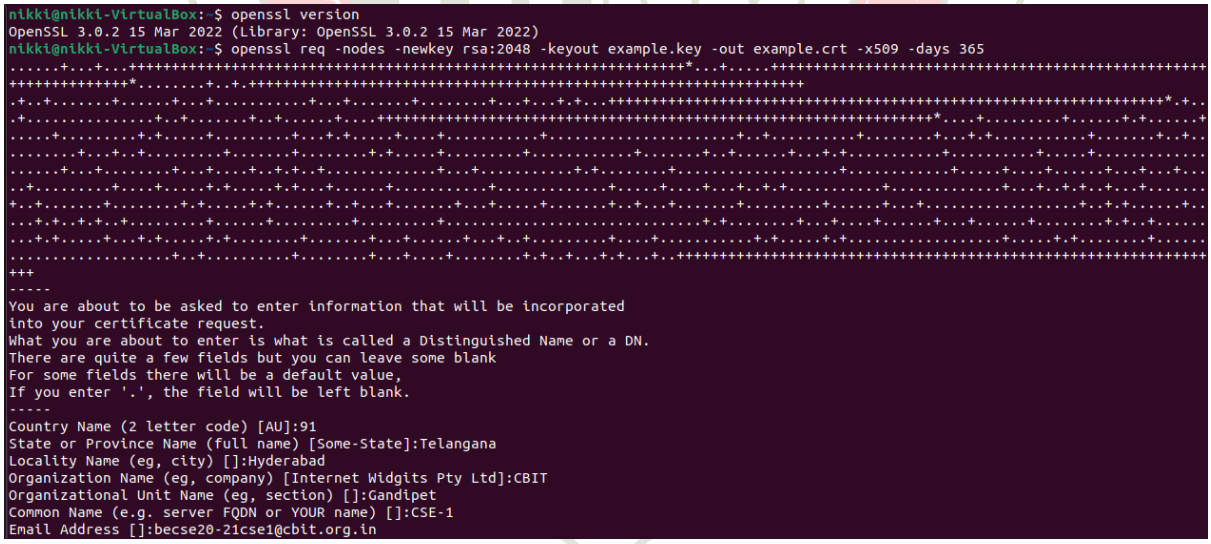
openssl version -a

USECASE

Creating a digital certificate and verifying using OpenSSL

Creating

openssl req -nodes -newkey rsa:2048 -keyout example.key -out example.crt -x509 -days 365



Display Digital Certificate

openssl x509 -in example.crt -text -noout

```
nikki@nikki-VirtualBox:~$ openssl x509 -in example.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            42:b8:e5:cb:f2:81:59:fc:e8:44:d9:b1:46:aa:32:b6:56:77:b2:d8
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = 91, ST = Telangana, L = Hyderabad, O = CBIT, OU = Gandipet, CN = CSE-1, emailAddress = becse20-21cse1@cbit.org.in
        Validity
            Not Before: Nov  5 06:27:29 2023 GMT
            Not After : Nov  4 06:27:29 2024 GMT
        Subject: C = 91, ST = Telangana, L = Hyderabad, O = CBIT, OU = Gandipet, CN = CSE-1, emailAddress = becse20-21cse1@cbit.org.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ad:e8:b9:07:3d:53:24:99:39:7a:7c:c5:78:eb:
                    0e:b2:df:a1:e7:52:d8:a3:77:40:19:68:5e:ce:80:
                    03:63:b2:98:32:c9:a1:b2:eb:cf:60:91:c6:24:41:
                    08:52:32:a7:1a:91:d7:a7:9d:4a:d2:c2:d8:66:27:
                    59:8a:57:d9:79:28:cb:3e:53:64:88:56:08:af:9c:
                    81:11:15:63:9b:25:e2:b4:7d:f8:06:48:31:4a:14:
                    84:2c:f0:4b:8d:3d:77:e1:79:29:87:21:e8:2c:00:
                    20:1b:48:45:26:0c:97:1d:30:28:ba:1c:61:0c:85:
                    33:da:f8:a9:87:e3:05:17:54:a6:6c:a0:b6:65:d1:
                    8e:47:92:1a:df:ef:f7:a9:82:df:9c:4d:88:ab:3d:
                    48:cc:36:e1:10:db:04:10:bf:59:96:32:2f:1a:b2:
                    53:ff:8b:f0:f4:d1:35:80:3d:a6:6b:7b:37:7c:b6:
                    97:03:c8:53:59:5b:43:4e:c8:bd:46:e5:75:98:95:
                    04:84:0e:fd:84:dd:6f:f1:2e:d8:a7:ca:8e:29:99:
                    fb:a8:84:29:53:fe:57:98:5b:13:6b:de:9b:fb:e9:
                    1d:5c:6d:6e:ba:52:31:2e:91:dc:df:f9:01:00:96:
                    ca:9c:ac:4a:bb:de:22:aa:c1:f9:cb:f3:85:a4:b8:
                    fd:af
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                CD:B4:0F:7E:3A:F6:23:75:59:D0:E6:0A:86:69:71:49:9D:51:18:04
            X509v3 Authority Key Identifier:
                CD:B4:0F:7E:3A:F6:23:75:59:D0:E6:0A:86:69:71:49:9D:51:18:04
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        0b:a0:02:e1:c8:5f:f6:36:37:48:39:fd:93:f0:b6:14:64:28:
        bc:fa:9a:be:55:2a:01:ae:67:bd:ca:98:76:03:13:df:9e:9e:
        05:f1:a0:2f:7e:28:88:0c:73:f9:1b:cc:8e:c5:a4:05:3c:09:
        47:1b:fb:8f:05:03:99:4e:44:61:cc:7d:d0:1d:1a:9d:5f:54:
        15:49:b3:fd:e0:37:aa:c0:76:28:30:f1:90:84:53:59:cf:88:
        ba:38:38:bd:c8:8f:1e:2e:ec:cc:d1:8e:17:7c:e4:7a:8c:5b:
        9a:48:4c:1e:3a:52:a5:b7:6d:b0:46:63:f1:ba:c7:9b:f1:b9:
        05:5d:06:02:bf:61:6c:eb:e0:b7:1e:d2:84:91:fa:29:76:9d:
        94:12:6b:ee:e8:cd:f2:69:1f:92:e5:9b:e5:ef:59:3b:1f:2e:
        21:d8:7a:32:9d:e7:9f:c8:a7:8b:f5:27:60:6a:a6:11:69:89:
        72:f4:0b:fc:34:44:3b:f8:6c:2e:3f:eb:e9:71:c9:1d:ba:af:
        b2:19:ec:18:80:ed:9f:c5:3a:6d:ff:dd:c6:c9:d4:f6:e0:1a:
        35:9a:f0:ae:48:a2:65:d3:b1:dc:7f:58:35:d3:86:8e:08:4c:
        cc:ad:9f:38:77:0c:c2:ce:54:52:b9:d4:58:30:e8:61:48:74:
        ea:c4:92:9e
```

Verify

openssl rsa -noout -modulus -in example.key | openssl sha256

```
nikki@nikki-VirtualBox:~$ openssl rsa -noout -modulus -in example.key | openssl sha256
SHA256(stdin)= ae3b0a52c7fd9fd53066ae62a2118d6d713a18dc727bed8b4d5d9b5df253c6ba
```

INSTITUTE OF TECHNOLOGY
1979

Signature of the Faculty……...

2) GNUPG

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories.
GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available. GnuPG also provides support for S/MIME and Secure Shell (ssh).

INSTALLATION:

Step-1: Install GPG.
To do this, run terminal and type in "*sudo apt-get install gnupg*"
Then we can start creating the digital certificates

USECASE

Creating a digital certificate and verifying using GnuPG

Generate key

gpg --gen-key

```
akshay@akshay-VirtualBox:~$ gpg --gen-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Chintala Sai Akshitha
Email address: akshithatanvi12@gmail.com
You selected this USER-ID:
    "Chintala Sai Akshitha <akshithatanvi12@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key FB59AB47D981494A marked as ultimately trusted
gpg: directory '/home/akshay/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/akshay/.gnupg/openpgp-revocs.d/C432B4E8CBC988964CCC8AD4FB59AB47D981494A.rev'
public and secret key created and signed.

pub   rsa3072 2023-11-02 [SC] [expires: 2025-11-01]
      C432B4E8CBC988964CCC8AD4FB59AB47D981494A
uid                      Chintala Sai Akshitha <akshithatanvi12@gmail.com>
sub   rsa3072 2023-11-02 [E] [expires: 2025-11-01]
```

Signing

gpg --sign my_file.txt

```
nikki@nikki-VirtualBox:~/.gnupg$ gpg --sign my_file.txt
```

Verifying

gpg --verify gpg.docx.gpg

```
nikki@nikki-VirtualBox:~/.gnupg$ gpg --verify my_file.txt.gpg
gpg: Signature made Sunday 05 November 2023 12:11:51 PM IST
gpg:                using RSA key 2A643B376A88BB40A387B67FB79014BCD50B9145
gpg: Good signature from "CSE-1 <becse20-21cse1@cbit.org.in>" [ultimate]
```

3) Hashcat

Hashcat is a powerful password recovery tool that can be used to crack and recover passwords from a variety of different types of hash values. It is considered one of the fastest and most advanced password recovery tools available.

hashcat uses a variety of different algorithms to crack passwords, including traditional dictionary attacks, brute-force attacks, and rule-based attacks.

It can work with a variety of different hash types, including MD5, SHA1, SHA256, and others. One of the key features of hashcat is its ability to use the power of a computer's GPU to perform cracking operations. This allows it to perform much faster than traditional CPU-based cracking tools, making it possible to crack even very complex passwords in a relatively short amount of time.

INSTALLATION:

Usually, Hashcat tool comes pre-installed with Kali Linux but if we need to install it write down the given command in the terminal.

sudo apt-get install hashcat

4) Cryptlib

Cryptlib is a powerful security toolkit which allows even inexperienced crypto programmers to easily add encryption and authentication security services to their software. The high-level interface provides anyone with the ability to add strong encryption and authentication capabilities to an application in as little as half an hour, without needing to know any of the low-level details which make the encryption or authentication work. Because of this, cryptlib dramatically reduces the cost involved in adding security to new or existing applications.

INSTALLATION:

Install cryptlib.i686 package

Please follow the steps below to install cryptlib.i686 package:

sudo dnf makecache

sudo dnf install cryptlib.i686

5) Crypto++

Crypto++ is a free and open-source C++ cryptographic library that provides a wide range of cryptographic algorithms and tools. It is widely used for secure communication, data storage, and various cryptographic operations.

It supports a comprehensive range of cryptographic algorithms, including symmetric and asymmetric encryption algorithms, hash functions, message authentication codes (MACs), digital signatures, key derivation functions, and more.

INSTALLATION:

Install the binary version of crypto++ (libcryptopp) on Ubuntu

$sudo apt-get install libcrypto++-dev libcrypto++-doc libcrypto++-utils

This will install crypto++ library and its development support under Linux file system.

CONCLUSION:

Here, we successfully familiarised with some cryptographic tools such as OpenSSL, GNUPG, Hashcat, Cryptlib and Crypto++

CHANAKYA VIDYAPEETHI
INSTITUTE OF TECHNOLOGY
స్వయం తేజస్వీన్ భవ
1979