**Ex. No.:3**

**Date: 09/03/2024**

**DSA Algorithm**

**Aim:**

To write a Python program to perform DSA Algorithm.

**Algorithm:**

1. Start.
2. Get the values of two prime numbers p and q from the user.
3. Get the value of y, x, hm and k.
4. Set up the secret text key and public key and perform operations to find v.
5. Calculate s = [k^-1(H(M) -xy) ] mod q to find s.
6. To verify the message, find w = (s)^-1 mod q.
7. Calculate v1 and v2 values.
8. Finally calculate y^v1y^v2 mod p to find v.
9. Compute r and v, if two values are same, then it is verified.

**Code:**

#Digital Signature Algorithm

import random

from hashlib import sha256

def coprime(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def extended\_gcd(aa, bb):

    lastremainder, remainder = abs(aa), abs(bb)

    x, lastx, y, lasty = 0, 1, 1, 0

    while remainder:

        lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)

        x, lastx = lastx - quotient\*x, x

        y, lasty = lasty - quotient\*y, y

    return lastremainder, lastx \* (-1 if aa < 0 else 1), lasty \* (-1 if bb < 0 else 1)

def modinv(a, m):

    g, x, y = extended\_gcd(a, m)

    if g != 1:

        raise Exception('Modular inverse does not exist')

    return x % m

def is\_prime(num):

    if num == 2:

        return True

    if num < 2 or num % 2 == 0:

        return False

    for n in range(3, int(num\*\*0.5)+2, 2):

        if num % n == 0:

            return False

    return True

def generate\_keypair(p, q):

    if not (is\_prime(p) and is\_prime(q)):

        raise ValueError('Both numbers must be prime.')

    elif p == q:

        raise ValueError('p and q cannot be equal')

    n = p \* q

    phi = (p-1) \* (q-1)

    e = random.randrange(1, phi)

    g = coprime(e, phi)

    while g != 1:

        e = random.randrange(1, phi)

        g = coprime(e, phi)

    d = modinv(e, phi)

    return ((e, n), (d, n))

def encrypt(privatek, plaintext):

    key, n = privatek

    numberRepr = [ord(char) for char in plaintext]

    print("Number representation before encryption: ", numberRepr)

    cipher = [pow(ord(char),key,n) for char in plaintext]

    return cipher

def decrypt(publick, ciphertext):

    key, n = publick

    numberRepr = [pow(char, key, n) for char in ciphertext]

    plain = [chr(pow(char, key, n)) for char in ciphertext]

    print("Decrypted number representation is: ", numberRepr)

    return ''.join(plain)

def hashFunction(message):

    hashed = sha256(message.encode("UTF-8")).hexdigest()

    return hashed

def verify(receivedHashed, message):

    ourHashed = hashFunction(message)

    if receivedHashed == ourHashed:

        print("Verification successful: ", )

        print(receivedHashed, " = ", ourHashed)

    else:

        print("Verification failed")

        print(receivedHashed, " != ", ourHashed)

def main():

    p = int(input("Enter a prime number (17, 19, 23, etc): "))

    q = int(input("Enter another prime number (Not one you entered above): "))

    print("Generating your public/private keypairs now . . .")

    public, private = generate\_keypair(p, q)

    print("Your public key is ", public ," and your private key is ", private)

    message = input("Enter a message to encrypt with your private key: ")

    print("")

    hashed = hashFunction(message)

    print("Encrypting message with private key ", private ," . . .")

    encrypted\_msg = encrypt(private, hashed)

    print("Your encrypted hashed message is: ")

    print(''.join(map(lambda x: str(x), encrypted\_msg)))

    print("")

    print("Decrypting message with public key ", public ," . . .")

    decrypted\_msg = decrypt(public, encrypted\_msg)

    print("Your decrypted message is:")

    print(decrypted\_msg)

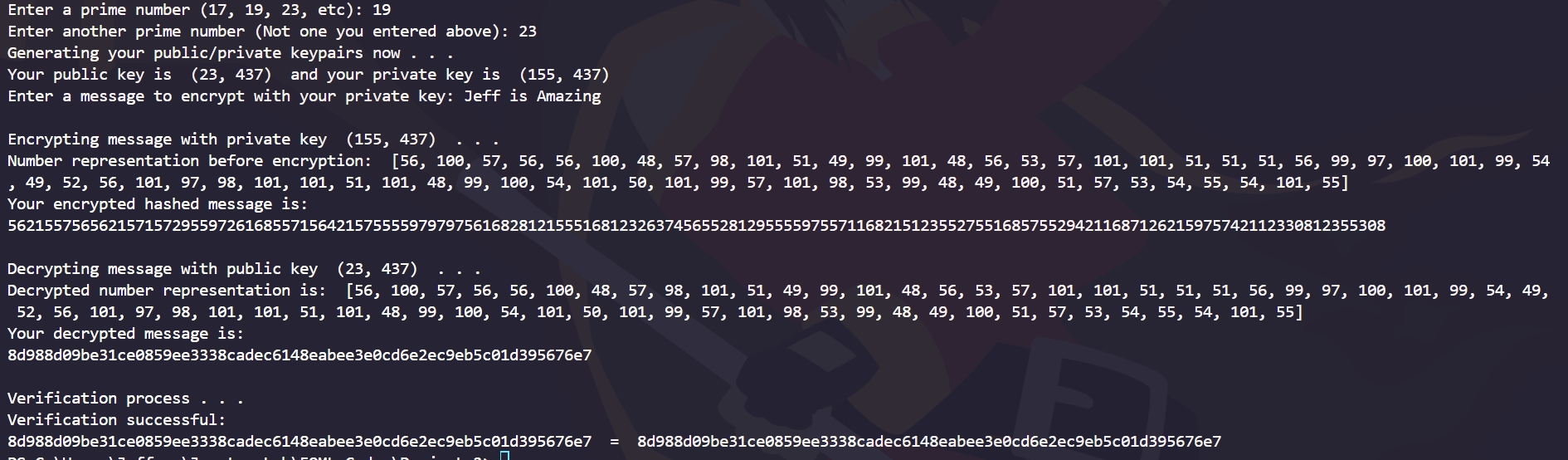
    print("")

    print("Verification process . . .")

    verify(decrypted\_msg, message)

main()

**Output:**



**Result:**

Hence, Digital Signature Algorithm has been implemented successfully.