Reg.no: 210701092

Experiment no: 3 Date: 5/3/24	DSA Algorithm (Digital Signature Algorithm)

AIM:

To write the python program to perform DSA Algorithm

ALGORITHM:

Step 1: STAR

Step 2: get the Value of two prime Value p and q from the User

Step 3: Ge the Values of g, x, hm, and k

Step 4: Set up the ~~Secr~~ Secret key And public key And perform operations to find v

Step 5: Calculate $S = [K^{-1}(H(M) + xr] \bmod q$ to find S

Step 6: To Verify the Message, find $w = (s)^{-1} \bmod q$

Step 7: Calculate u_1 and u_2 Values

Step 8: Finally Calculate $(g^{u1} y^{u2} \bmod p$ to find v

Step 9: Compare r and v If Two Values are Same, then it is Verified

Reg no: 210701092 Name: Jeffrey Jesudasan R

Exp 3: Digital Signature Algorithm

Code:

```
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()
```

<u>Output:</u>

Reg no: 210701092   Name: Jeffrey Jesudasan R

```
Enter a prime number (17, 19, 23, etc): 19
Enter another prime number (Not one you entered above): 23
Generating your public/private keypairs now . . .
Your public key is  (23, 437)  and your private key is  (155, 437)
Enter a message to encrypt with your private key: Jeff is Amazing

Encrypting message with private key  (155, 437)  . . .
Number representation before encryption:  [56, 100, 57, 56, 56, 100, 48, 57, 98, 101, 51, 49, 99, 101, 48, 56, 53, 57, 101, 101, 51, 51, 51, 56, 99, 97, 100, 101, 99, 54
, 49, 52, 56, 101, 97, 98, 101, 101, 51, 101, 48, 99, 100, 54, 101, 50, 101, 99, 57, 101, 98, 53, 99, 48, 49, 100, 51, 57, 53, 54, 55, 54, 101, 55]
Your encrypted hashed message is:
562155756562157157295597261685571564215755597979756168281215551681232637456551281295559755711682151235527551685755294211687126215975742112330812355308

Decrypting message with public key  (23, 437)  . . .
Decrypted number representation is:  [56, 100, 57, 56, 56, 100, 48, 57, 98, 101, 51, 49, 99, 101, 48, 56, 53, 57, 101, 101, 51, 51, 51, 56, 99, 97, 100, 101, 99, 54, 49,
 52, 56, 101, 97, 98, 101, 101, 51, 101, 48, 99, 100, 54, 101, 50, 101, 99, 57, 101, 98, 53, 99, 48, 49, 100, 51, 57, 53, 54, 55, 54, 101, 55]
Your decrypted message is:
8d988d09be31ce0859ee3338cadec6148eabee3e0cd6e2ec9eb5c01d395676e7

Verification process . . .
Verification successful:
8d988d09be31ce0859ee3338cadec6148eabee3e0cd6e2ec9eb5c01d395676e7  =  8d988d09be31ce0859ee3338cadec6148eabee3e0cd6e2ec9eb5c01d395676e7
```