Experiment 8

AIM: Study and Implement RSA Digital Signature.

Experiment 8

Shashwat Shah
60004220126
TYBtech Comps B

Aim: To implement RSA digital signature

Theory: Algorithm.

Steps.

1) Sender A uses hashing algorithm to calculate the message digest (MD1) over the original message M.
2) Sender A now encrypts the message digest with its private key. Output of this process is called Digital Signature of A.
3) Now A sends the digital signature along the original message M.
4) When B receives the original message M and digital signature, it uses the same message digest algorithm as was used by A. and calculates its own message digest (MD2) for M.
5) Now B uses A's public key to decrypt the digital signature because it was encrypted by A's public key. Result of this process is the original MD1 calculated by A.
6) If MD1 == MD2, B accepts the original message and ensures that message has come from A, not someone posing as A.

Conclusion: Thus, we have successfully implemented RSA digital signature.

Code:

```
import hashlib
import random

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def generate_key_pair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randrange(1, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)
    d = mod_inverse(e, phi)
    return ((e, n), (d, n))

def rsa_encrypt(message, public_key):
    e, n = public_key
    encrypted_message = [pow(char, e, n) for char in message]
    return encrypted_message

def rsa_decrypt(encrypted_message, private_key):
    d, n = private_key
    decrypted_message = [chr(pow(char, d, n)) for char in
encrypted_message]
    return ''.join(decrypted_message)

def md5_hash(message):
    hash_object = hashlib.md5(message.encode())
    return int.from_bytes(hash_object.digest(), byteorder='big')

def sign_message(message, private_key):
    hashed_message = md5_hash(message)
```

```
    signature = pow(hashed_message, private_key[1])
    return signature % (p * q)

def verify_signature(message, public_key):
    hashed_message = md5_hash(message)
    decrypted_signature = pow(hashed_message, public_key[1])
    return decrypted_signature % (p * q)

# Test the functions
p = 61
q = 53
public_key, private_key = generate_key_pair(p, q)

# print(public_key, private_key)

message = "Aksh"
signature = sign_message(message, private_key)
print("Signature:", signature)
verified = verify_signature(message, public_key)
print("Verified:", verified)
```

Output:

```
Signature: 97
Verified: 97
```

Conclusion: Thus we have studied and implemented RSA with Digital Signature.