Amrita Vishwa Vidyapeetham, Amritapuri

Department of Computer Science and Engineering

22AIE314 Computer Security

Lab Sheet 4

Public Key Cryptography and Secure Authentication

1. **Implement RSA Algorithm**

   Given:

   Primes: p=13,q=17, Public exponent: e=5, Plain Text: "AMRITA"

   a) Compute n, totient function $\phi(n)$ and private key d.
   b) Encrypt the message using the public key (e,n).
   c) Decrypt the cipher using the private key (d,n)
   d)Verify that decrypted message matches the original.

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

def modinv(e, phi):
    t, newt = 0, 1
    r, newr = phi, e
    while newr != 0:
        quotient = r // newr
        t, newt = newt, t - quotient * newt
        r, newr = newr, r - quotient * newr
    if r > 1:
        raise Exception("No inverse exists")
    if t < 0:
        t += phi
    return t

def encrypt_char(m, e, n):
    return pow(m, e, n)

def decrypt_char(c, d, n):
    return pow(c, d, n)



p = 11
q = 19
e = 7
text = "Girish"



n = p * q
phi = (p - 1) * (q - 1)
d = modinv(e, phi)
print("\na) Compute n, φ(n), and d")
print(f"p = {p}, q = {q}")
print(f"n = p * q = {n}")
print(f"φ(n) = (p-1)*(q-1) = {phi}")
print(f"Public exponent e = {e}")
print(f"Private key d = {d}")
```

```

a) Compute n, φ(n), and d
p = 11, q = 19
n = p * q = 209
φ(n) = (p-1)*(q-1) = 180
Public exponent e = 7
Private key d = 103
```

```
ascii_vals = [ord(c) for c in text]
ciphertext = [encrypt_char(m, e, n) for m in ascii_vals]
```

```
print("\nb) Encryption using public key (e, n)")
print(f"Plaintext = {text}")
print(f"ASCII values = {ascii_vals}")
print(f"Ciphertext = {ciphertext}")
```

```
b) Encryption using public key (e, n)
Plaintext = Girish
ASCII values = [71, 105, 114, 105, 115, 104]
Ciphertext = [3, 129, 38, 129, 58, 80]
```

```
decrypted_vals = [decrypt_char(c, d, n) for c in ciphertext]
decrypted_text = ''.join(chr(m) for m in decrypted_vals)
print("\nc) Decryption using private key (d, n)")
print(f"Decrypted ASCII values = {decrypted_vals}")
print(f"Decrypted Text = {decrypted_text}")
```

```
c) Decryption using private key (d, n)
Decrypted ASCII values = [71, 105, 114, 105, 115, 104]
Decrypted Text = Girish
```

```
print("\nd) Verification")
if decrypted_text == text:
    print("Decrypted message matches original.")
else:
    print("Decrypted message does NOT match original.")
```

```
d) Verification
Decrypted message matches original.
```

2. **Implement Diffie Hellman Key Exchange**

Given:

- Prime q=5, Primitive root α=2
- Private keys: A = 3, B = 7

a) Compute public values of A and B
b) Find the shared secret key at both ends.

```
def diffie_hellman(q, alpha, private_a, private_b):
    A_pub = pow(alpha, private_a, q)
    B_pub = pow(alpha, private_b, q)
    shared_key_a = pow(B_pub, private_a, q)
    shared_key_b = pow(A_pub, private_b, q)

    return A_pub, B_pub, shared_key_a, shared_key_b


q = 5
alpha = 2
private_a = 3
private_b = 7


# a)
A_pub, B_pub, key_a, key_b = diffie_hellman(q, alpha, private_a, private_b)
print("\na) Compute Public Values")
print(f"Prime q = {q}, Primitive root α = {alpha}")
print(f"Private Key A = {private_a}, B = {private_b}")
print(f"Public Key A_pub = α^A mod q = {A_pub}")
print(f"Public Key B_pub = α^B mod q = {B_pub}")
```

```
a) Compute Public Values
Prime q = 5, Primitive root α = 2
Private Key A = 3, B = 7
Public Key A_pub = α^A mod q = 3
Public Key B_pub = α^B mod q = 3
```

```
# b)
print("\nb) Shared Secret Key")
print(f"Shared Key at A's end: (B_pub)^A mod q = {key_a}")
print(f"Shared Key at B's end: (A_pub)^B mod q = {key_b}")
```

```
if key_a == key_b:
    print(f"Shared key matched: {key_a}")
else:
    print("Shared keys do not match.")
```

```
b) Shared Secret Key
Shared Key at A's end: (B_pub)^A mod q = 2
Shared Key at B's end: (A_pub)^B mod q = 2
Shared key matched: 2
```