Assignment No 5

Title:

Write c/c++/java/python program to implement RSA algorithm.

Theory:

RSA Algorithm

The RSA algorithm is a widely used public-key encryption algorithm named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. It is based on the mathematical concepts of prime factorization and modular arithmetic.

The algorithm for RSA is as follows:

1. Select 2 prime numbers, preferably large, p and q.
2. Calculate n = p*q.
3. Calculate phi(n) = (p-1)*(q-1)
4. Choose a value of e such that 1<e<phi(n) and gcd(phi(n), e) = 1.
5. Calculate d such that d = (e^-1) mod phi(n).

Here the public key is {e, n} and private key is {d, n}. If M is the plain text then the cipher text C = (M^e) mod n. This is how data is encrypted in RSA algorithm. Similarly, for decryption, the plain text M = (C^d) mod n.

- **Example:** Let p=3 and q=11 (both are prime numbers)
- Now, n = p*q = 3*11 = 33
- phi(n) = (p-1)*(q-1) = (3-1)*(11-1) = 2*10 = 20
- Value of e can be 7 since 1<7<20 and gcd(20, 7) = 1.
- Calculating d = 7^-1 mod 20 = 3.
- Therefore, public key = {7, 33} and private key = {3, 33}.

Suppose our message is M=31. You can encrypt and decrypt it using the RSA algorithm as follows:

Encryption: C = (M^e) mod n = 31^7 mod 33 = 4

Decryption: M = (C^d) mod n = 4^3 mod 33 = 31

Since we got the original message that is plain text back after decryption, we can say that the algorithm worked correctly.

Implementation RSA algorithm using python

This program generates a pair of RSA keys (a public key and a private key), uses the public key to encrypt a message, and then uses the private key to decrypt the ciphertext back into the original message.

Here's a step-by-step explanation of how the program works:

1. The is_prime(n) function checks whether a given number n is prime. It does this by checking if n is divisible by any number from 2 to the square root of n.
2. The generate_primes() function generates two random prime numbers p and q between 100 and 1000. It does this by generating random numbers in that range and checking whether they are prime using the is_prime() function.
3. The gcd(a, b) function computes the greatest common divisor of two numbers a and b using the Euclidean algorithm.

4. The lcm(a, b) function computes the least common multiple of two numbers a and b using the formula a * b / gcd(a, b).
5. The generate_keys() function generates a pair of RSA keys. It does this by:
    - Generating two random prime numbers p and q using the generate_primes() function.
    - Computing the modulus n as p * q.
    - Computing the totient of n as lcm(p - 1, q - 1).
    - Generating a random number e between 2 and phi_n - 1
6. Encrypt() function encrypts the given message using public key.
7. Decrypt() function decrypts the encrypted message to give original message using private key.

Conclusion:

It is concluded that, while establishing RSA key pairs, usage keys and general-purpose keys are integrated. In usage RSA keys, two key pairs are used for encryption and signatures. In General-purpose key, one single pair is used for both encryption and signature.

Sure, here's a Python program to implement the RSA algorithm:

```
import random

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

def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi

    while e > 0:
        temp1 = temp_phi // e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2
```

```
        x = x2 - temp1 * x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y

    if temp_phi == 1:
        return d + phi

def generate_keypair(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 = multiplicative_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(pk, plaintext):
    key, n = pk
    cipher = [(ord(char) ** key) % n for char in plaintext]
    return cipher

def decrypt(pk, ciphertext):
    key, n = pk
    plain = [chr((char ** key) % n) for char in ciphertext]
    return ''.join(plain)
```

```
if __name__ == '__main__':
    p = int(input("Enter a prime number (p): "))
    q = int(input("Enter another prime number (q): "))

    public, private = generate_keypair(p, q)

    print("Public key: ", public)
    print("Private key: ", private)

    message = input("Enter a message to encrypt: ")
    encrypted_message = encrypt(public, message)
    print("Encrypted message: ", ''.join(map(lambda x: str(x), encrypted_message)))

    decrypted_message = decrypt(private, encrypted_message)
    print("Decrypted message: ", decrypted_message)
```

Now, let's go through the steps of the RSA algorithm:

1. Choose two distinct prime numbers, p and q.
2. Calculate n = p * q.
3. Calculate phi = (p-1) * (q-1).
4. Choose an integer e such that 1 < e < phi and gcd(e, phi) = 1. This is the public key exponent.
5. Calculate d, the modular multiplicative inverse of e modulo phi. This is the private key exponent.
6. The public key is (e, n).
7. The private key is (d, n).
8. To encrypt a message, convert each letter to a number, m, such that 0 <= m <= n-1. Then, calculate c = m^e mod n for each letter. The ciphertext is the sequence of c values.
9. To decrypt the ciphertext, calculate m = c^d mod n for each c value. Then, convert each m value back to a letter.

In the Python program, the `generate_keypair` function implements steps 2-5. The `encrypt` function implements step 8. The `decrypt` function implements step 9. The `gcd` and `multiplicative_inverse` functions are helper functions used in step 5.

When you run the program, it prompts you to enter two prime numbers, p and q. It then generates the public and private keys using those numbers. You can then enter a message to encrypt, and the program will encrypt and decrypt the message using the RSA algorithm.