**Experiment-9**

**Aim:** Study and implement RSA Encryption and Decryption function.

**Introduction:**

RSA, which stands for Rivest–Shamir–Adleman, is a popular method for keeping our digital messages and information safe when they travel over the internet.

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric means that it works on two different keys i.e., Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

**RSA is like a special lock and key system for your digital messages. Here is how it works:**

1. You use the recipient's public key to lock up your message, making it super secure.
2. The recipient uses their private key to unlock and read the message.

It helps uskeep our messages safe before sending them. And ensure that the message has not been tampered with during its journey.

RSA was the first successful way to use these special keys in the digital world.

**Method to generate public and private keys:**

* We take 2 prime numbers p & q.
* Multiply these numbers to find n = p x q, where n is called the modulus for encryption and decryption.
* Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n),  
  **gcd (e,d(n)) =1**
* If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.

**C = me mod n**  
Here**, m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.

* To determine the private key, we use the following formula to calculate the d such that:

**d\*e mod {(p - 1) x (q - 1)} = 1**  
**Or**  
**d\*e mod φ (n) = 1**

* The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.

**m = cd mod n**

**Program (Source Code):**

#include <bits/stdc++.h>

using namespace std;

int T1 = 0;

int T2 = 1;

int gcd(int a, int b) {

    int temp;

    while (1){

        temp = a % b;

        if (temp == 0){

            return b;

        }

        a = b;

        b = temp;

    }

}

int extendedEuclidean(int a, int b){

    // a > b

    int A = a;

    int B = b;

    int Q = A/B;

    int R = A%B;

    int T = T1 - (T2\*Q);

    // cout<<Q<<" "<<A<<" "<<B<<" "<<R<<" "<<T1<<" "<<T2<<" "<<T<<" "<<endl;

    A = B;

    B = R;

    T1 = T2;

    T2 = T;

    if (B == 0){

        return T1;

    }

    else{

        return extendedEuclidean(A, B);

    }

}

double powerMod(int plaintext, double power, double n){

    if (int(power) == 0) {

        return 1;

    }

    double result = powerMod(plaintext, power / 2, n);

    result = int(result \* result) % int(n);

    if (int(power) % 2 == 1) {

        result = int(result \* plaintext) % int(n);

    }

    return result;

}

int main() {

    double p = 3;

    double q = 7;

    double n = p \* q;

    double phi = (p - 1) \* (q - 1);

    double e = 2;

    // calculating value of e

    while (e < phi){

        // e must be co-prime to phi and

        // smaller than phi.

        if (gcd(e, phi) == 1)

            break;

        else

            e++;

    }

    // calculating value of d

    double d = extendedEuclidean(phi, e);

    while (d<0){

        d += phi;

    }

    // cout<<"e: "<<e<<endl;

    // cout<<"d: "<<d<<endl;

    // cout<<"n: "<<n<<endl;

    int plaintext = 12;

    cout<<"Plaintext message: "<<plaintext<<endl;

    double cipherText = powerMod(plaintext,e,n);

    cout<<"Encrypted Text: "<<cipherText<<endl;

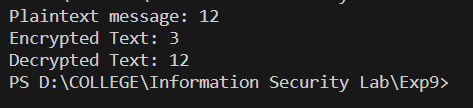
    double decryptedText = powerMod(cipherText,d,n);

    cout<<"Decrypted Text: "<<decryptedText<<endl;

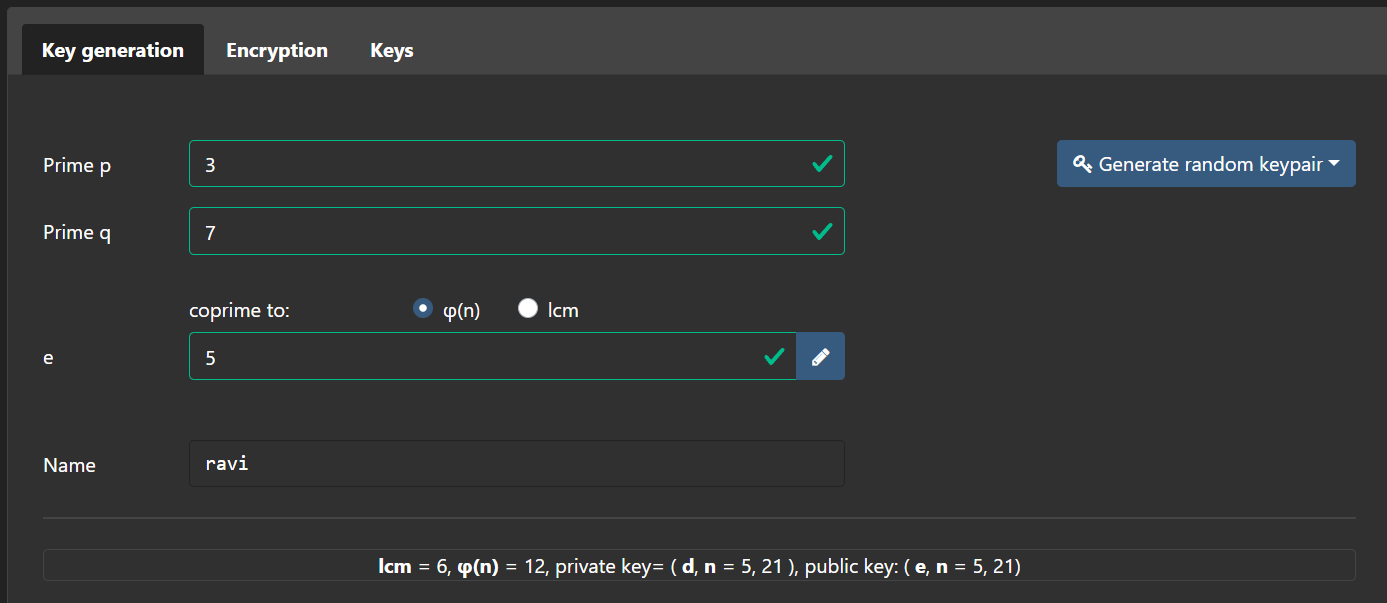
    return 0;

}

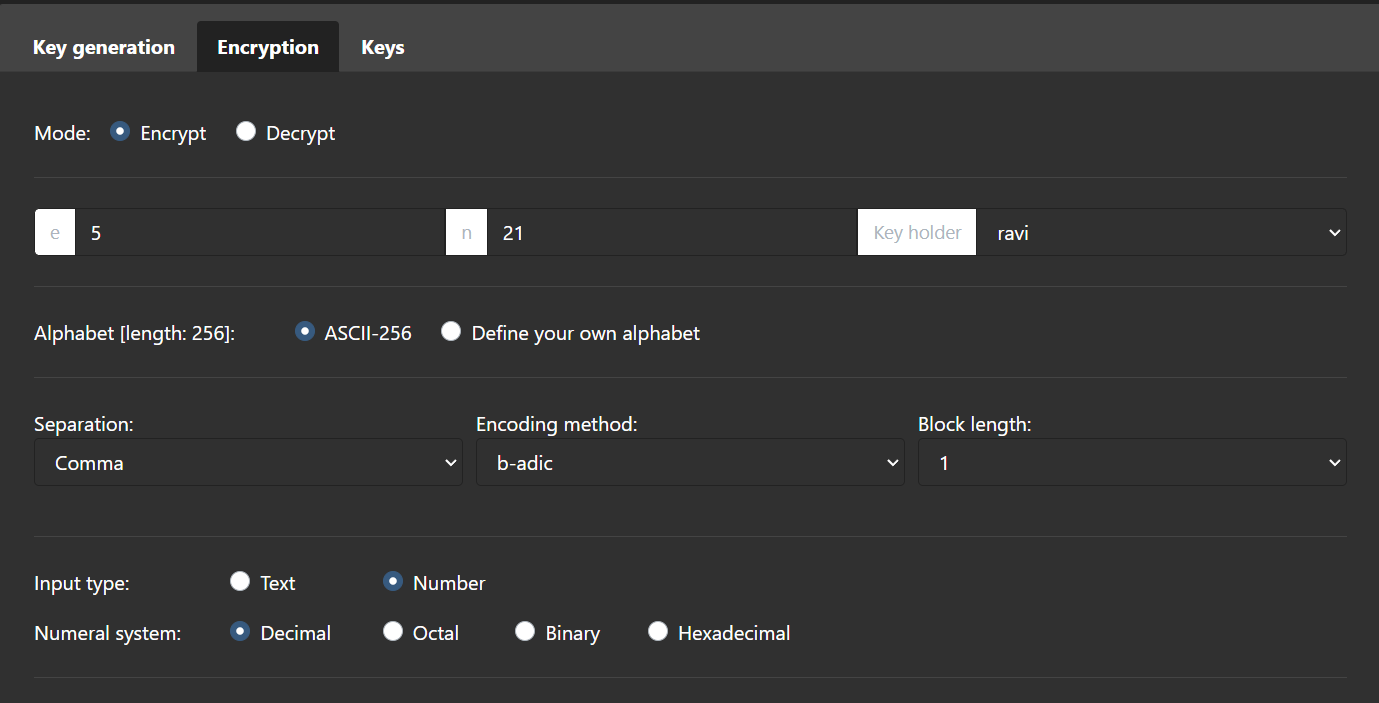
**Output (Program):**

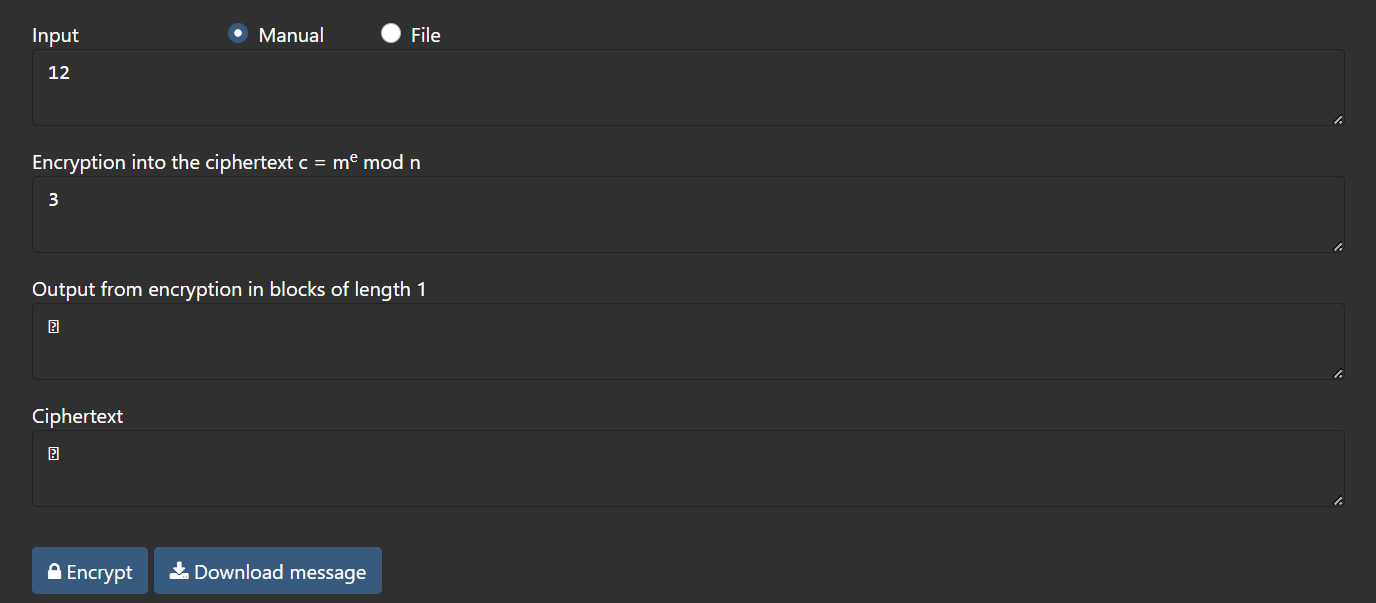
****

**Output (Cryptool):**

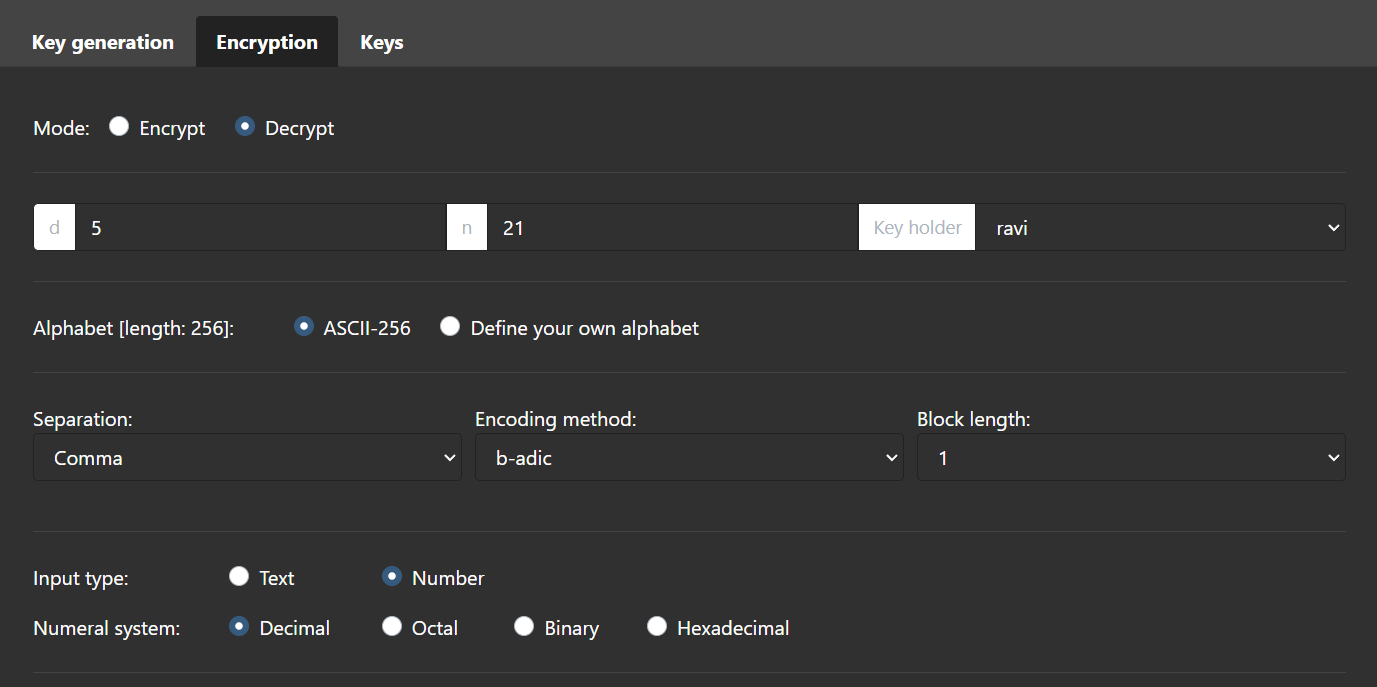
****

Encryption:

****

****

Decryption:





**Cryptanalysis:**

**Timing Attacks**: Think of this like spying on a clock. Attackers watch how long it takes to open a locked message. It is hard to do, and they need super-precise measurements to learn anything.

**Quantum Computing**: Future quantum computers, if sufficiently powerful, could potentially factor large numbers significantly faster than classical computers using Shor's algorithm. This represents a potential threat to RSA encryption.

**Common Modulus Attack**: In some cases where multiple parties use the same modulus (n) with different public exponents (e), a common modulus attack can be mounted to recover the shared modulus and, subsequently, the private keys.

**Applications:**

RSA (Rivest-Shamir-Adleman) encryption is widely used for:

**1. Secure Communication:** Protects data during transmission, like online banking and e-commerce transactions.

**2. SSL/TLS Encryption:** Fundamental for secure web browsing.

**3. Digital Signatures:** Verifies document authenticity and integrity.

**4. Secure Messaging:** Encrypts messages in various apps.

**5. Secure Email:** Used in PGP and S/MIME for email encryption.

**6. Authentication:** Enables secure login, including 2FA and SSH.

**7. VPN Security:** Establishes secure internet tunnels.

**8. IoT Security:** Used in securing IoT device communication.

**9. Smart Cards and Tokens:** Secures private keys for secure access.

**10. Digital Certificates:** Validates website authenticity.

**11. Secure File Storage:** Encrypts data on disks and cloud storage.

**12. Secure Backup:** Ensures backup data is protected.

RSA remains vital for safeguarding digital communication and data.

**References:**

1. GeeksforGeeks
2. www.cryptool.org/en/cto/hill