Project Report
On
Computer Networking: Security
(CSE3752)

Develop a program to demonstrate the RSA cryptosystem.

Submitted by:

Name: Dinanath Dash	**Regd. No.:** 2241004161
Name: Ashutosh Raj	**Regd. No.:** 2241018015
Name: Swarnabha Roy	**Regd. No.:** 2241018192
Name: Sanjay Kumar Satapathy	**Regd. No.:** 2241019339

B. Tech. CSE 6th Semester (Section 2241026)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY),
BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **CSE** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled **“Develop a program to demonstrate the RSA cryptosystem”** submitted to **Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking: Security (CSE 3752)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Name: Dinanath Dash	Regd. No.: 2241004161
Name: Ashutosh Raj	Regd. No.: 2241018015
Name: Swarnabha Roy	Regd. No.: 2241018192
Name: Sanjay Kumar Satapathy	Regd. No.: 2241019339

Date: 30-05-2025
Place: Bhubaneswar

Abstract

This project focuses on the implementation of the RSA cryptosystem, a widely used asymmetric encryption technique in modern cryptography. The objective is to demonstrate the core principles behind public-key infrastructure using RSA, which includes key generation, encryption, and decryption. The implementation ensures proper validation of the selected prime numbers, calculation of modular inverse through the Extended Euclidean Algorithm, and secure conversion of plaintext characters to ciphertext using ASCII representation. The project not only helps in understanding RSA's working mechanism but also emphasizes the importance of mathematical rigor in cryptographic systems. The overall system ensures secure message encryption and decryption through a structured and verified key management process.

Contents

1. Introduction

In today's digital world, securing communication is of utmost importance. Cryptography plays a crucial role in safeguarding sensitive data, and among the various techniques available, RSA (Rivest–Shamir–Adleman) stands out as a prominent asymmetric cryptographic algorithm. Unlike symmetric encryption, RSA uses two different keys — a public key for encryption and a private key for decryption — which enhances security, especially for open networks. This project implements RSA from the ground up, focusing on fundamental concepts such as the generation of prime numbers, computation of public/private key pairs using modular arithmetic, and accurate handling of plaintext and ciphertext. ASCII-based processing allows string messages to be securely encrypted and later retrieved in their original form. The goal is to help understand RSA not just theoretically but also in practical, programmatic terms.

2. Problem Statement

Objective: The aim of this project is to design and implement a functional RSA cryptosystem that demonstrates the core principles of public key encryption. The specific objectives are as follows:

1. To understand and implement the RSA algorithm for secure communication.
2. To validate input parameters, ensuring that the numbers provided for key generation (p and q) are prime.
3. To incorporate the Extended Euclidean Algorithm to compute the modular multiplicative inverse, which is crucial for generating the private key.
4. To enable the system to handle plaintext messages by converting characters to their ASCII values and processing them correctly through encryption.
5. To accurately retrieve the original message after decryption, validating the entire encryption-decryption cycle.

This project also focuses on the internal logic of RSA to make sure the implementation is mathematically sound and secure for message encryption and decryption.

3. Methodology

The implementation of the RSA cryptosystem was approached methodically to ensure both educational value and practical functionality. The following steps outline the methodology adopted:

1. Prime Number Validation

- Objective: Ensure that the inputs p and q are prime numbers, as RSA's security relies on the difficulty of factoring large primes.
- Implementation: A primality test function was developed to check if the provided numbers are prime. This function checks divisibility up to the square root of the number, which is efficient for small to moderately large numbers.

2. Key Generation

- Compute n: Calculate $n = p \times q$. This value is used as the modulus for both the public and private keys.
- Compute Euler's Totient Function $\phi(n)$: Calculate $\phi(n) = (p - 1) \times (q - 1)$. This value is crucial for determining the public and private exponents.
- Choose Public Exponent e: Select an integer e such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$. This ensures that e and $\phi(n)$ are coprime.
- Compute Private Exponent d: Calculate the modular multiplicative inverse of e modulo $\phi(n)$, i.e., find d such that $d \times e \equiv 1 \bmod \phi(n)$. This is achieved using the Extended Euclidean Algorithm.

3. Encryption Process

- Plaintext Conversion: Convert the plaintext message into a list of ASCII values.
- Encryption: For each ASCII value m, compute the ciphertext c using the formula $c = m^e \bmod n$. This process transforms the plaintext into an unreadable format without the private key.

4. Decryption Process

- Decryption: For each ciphertext value c, compute the original ASCII value m using the formula $m = c^d \bmod n$.
- Plaintext Reconstruction: Convert the list of ASCII values back into characters to reconstruct the original message.

Code:

```
import math

# Check if number is prime
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
            return False
        for i in range(5, int(math.sqrt(n)) + 1, 6):
            if n % i == 0 or n % (i + 2) == 0:
                return False
        return True

# Extended Euclidean Algorithm
def extended_gcd(a, b):
        if a == 0:
            return b, 0, 1
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return gcd, x, y

# Modular Inverse
def mod_inverse(e, phi):
        gcd, x, _ = extended_gcd(e, phi)
        if gcd != 1:
            return None
        return x % phi

# Encryption
def encrypt(plaintext, e, n):
        ascii_vals = [ord(char) for char in plaintext]
        cipher = [pow(val, e, n) for val in ascii_vals]
        return cipher

# Decryption
def decrypt(ciphertext, d, n):
        decrypted_vals = [pow(val, d, n) for val in ciphertext]
        plaintext = ''.join(chr(val) for val in decrypted_vals)
        return plaintext

# Main RSA setup and execution
def rsa_demo():
        print("RSA Cryptosystem Demonstration\n")

        # Get primes p and q
        p = int(input("Enter prime number p: "))
        while not is_prime(p):
            p = int(input("Invalid! Enter a valid prime number for p:
                     "))

        q = int(input("Enter prime number q: "))
        while not is_prime(q):
            q = int(input("Invalid! Enter a valid prime number for q:
                     "))

        n = p * q
        phi = (p - 1) * (q - 1)

        print(f"\nCalculated n = {n} and φ(n) = {phi}")

        # Choose e
        e = int(input("Enter encryption key e (1 < e < φ(n)) such that
         gcd(e, φ) = 1: "))
        while math.gcd(e, phi) != 1:
            e = int(input("Invalid! Enter a valid e: "))

        # Compute d
        d = mod_inverse(e, phi)
        if d is None:
            print("Modular inverse could not be found. Try different
                    values.")
            return

        print(f"Public Key (e, n): ({e}, {n})")
        print(f"Private Key (d, n): ({d}, {n})")

        # Message input
        plaintext = input("\nEnter message to encrypt: ")

        # Encrypt and Decrypt
        ciphertext = encrypt(plaintext, e, n)
        print(f"\nEncrypted Message (ciphertext): {ciphertext}")

        decrypted_message = decrypt(ciphertext, d, n)
        print(f"Decrypted Message: {decrypted_message}")

    # Run the demo
    rsa_demo()
```

4. Results & Interpretation

The RSA cryptosystem was successfully implemented in Python, adhering to the methodology outlined above. The implementation includes functions for primality testing, key generation, encryption, and decryption. Below is a summary of the implementation and the results obtained:

Key Generation

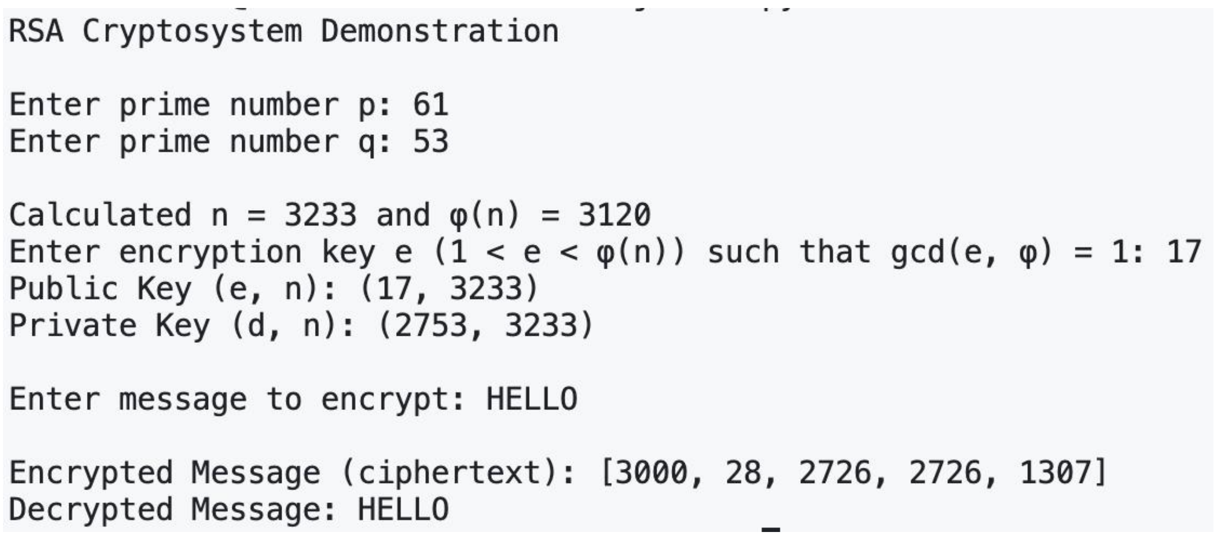
- Input Primes: $p = 61, q = 53$
- Computed Values:
  - $n = p \times q = 3233$
  - $\phi(n) = (p - 1) \times (q - 1) = 3120$
- Public Exponent: $e = 17$ (chosen such that $\gcd(e, \phi(n)) = 1$)
- Private Exponent: d=2753 (computed using the Extended Euclidean Algorithm)

Encryption and Decryption Example

- Plaintext Message: "HELLO"
- ASCII Conversion: [72, 69, 76, 76, 79]
- Encryption:
  - Each ASCII value m is encrypted using $c = m^e \bmod n$
  - Resulting Ciphertext: [3000, 28, 2726, 2726, 1307]
- Decryption:
  - Each ciphertext value c is decrypted using $\mathrm{m} = \mathrm{c}^{\mathrm{d}} \bmod \mathrm{n}$
  - Reconstructed ASCII Values: [72, 69, 76, 76, 79]
  - Reconstructed Message: "HELLO"

The implementation accurately encrypts and decrypts messages, demonstrating the RSA algorithm's effectiveness. The use of ASCII values for character representation ensures compatibility with standard text messages.

Output:



```
RSA Cryptosystem Demonstration

Enter prime number p: 61
Enter prime number q: 53

Calculated n = 3233 and φ(n) = 3120
Enter encryption key e (1 < e < φ(n)) such that gcd(e, φ) = 1: 17
Public Key (e, n): (17, 3233)
Private Key (d, n): (2753, 3233)

Enter message to encrypt: HELLO

Encrypted Message (ciphertext): [3000, 28, 2726, 2726, 1307]
Decrypted Message: HELLO
```

5. Conclusion

The RSA cryptosystem was successfully implemented and validated through a programmatic approach. The project covered all necessary components: prime validation, key generation, ASCII conversion, encryption, and decryption. By using the Extended Euclidean Algorithm, the system reliably calculated the modular inverse required for the decryption key. The encrypted messages were correctly converted back to their original plaintext, demonstrating the correctness of the cryptographic process. This project reinforces the theoretical foundation of RSA while also providing practical insight into its real-world application. Through secure key management and careful mathematical operations, it confirms that RSA remains a robust solution in the domain of public-key cryptography.

References

1. Rivest, R. L., Shamir, A., & Adleman, L. (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21(2), 120–126.

2. Wikipedia contributors. (2025, April). RSA (cryptosystem). Wikipedia. https://en.wikipedia.org/wiki/RSA_cryptosystem

3. Brilliant.org. *RSA Encryption*. https://brilliant.org/wiki/rsa-encryption/

4. GeeksforGeeks. (2025, January). *RSA Algorithm in Cryptography.* https://www.geeksforgeeks.org/rsa-algorithm-cryptography/