Implementation of RSA and Related Cryptographic Algorithms

Zahrasadat Dehnavi

29 August

1 Introduction

This project implements fundamental cryptographic algorithms in Python, focusing on:

- Generating prime numbers with the Miller-Rabin primality test.
- Computing greatest common divisors (GCD) and modular inverses using Euclidean algorithms.
- Implementing RSA encryption and decryption for educational purposes.

The project demonstrates key concepts of **public-key cryptography**, modular arithmetic, and secure message transmission in an accessible and educational manner.

2 Miller-Rabin Primality Test

The `Miller` class implements a probabilistic method for checking if a number is prime. It also includes a prime number generator.

2.1 Key Methods

- `power(base, exp, mod)`: Computes ($\text{base}^{\text{exp}} \bmod \text{mod}$) efficiently using square-and-multiply.
- `miller_test(d, n)`: Performs one iteration of the Miller-Rabin test.
- `is_prime(n, k)`: Runs k iterations to determine whether n is probably prime.
- `generate_prime(bits)`: Generates a random prime number of a specified bit length.

2.2 Example Output

```
Generated 512-bit prime numbers:
p = 13407807929942597099574024998205846127479365820592393З
q = 134078079299425970995740249982058461274793658205923959
```

3 Euclidean Algorithms

The `Euclid` class provides implementations for computing:

- GCD (greatest common divisor)
- Extended GCD for solving equations $a \cdot x + b \cdot y = \gcd(a, b)$
- Modular inverse, essential for RSA key generation

3.1 Example of Extended Euclidean Algorithm

```
e = 17, phi = 3120
Extended GCD result: gcd = 1, x = 2753, y = -15
Private exponent d = 2753
```

4 RSA Implementation

The `RSA` class generates keys, encrypts, and decrypts messages using the `Miller` and `Euclid` classes.

4.1 Steps

1. Generate two distinct primes p and q using Miller-Rabin.
2. Compute modulus $n = p \cdot q$ and Euler's totient $\phi = (p-1) \cdot (q-1)$.
3. Choose public exponent e such that $\gcd(e, \phi) = 1$.
4. Compute private exponent d as the modular inverse of e modulo ϕ.

4.2 Encryption/Decryption

- Messages are converted to numbers ($A = 1, ..., Z = 26$).
- Encryption: $cipher = m^e \bmod n$
- Decryption: $m = cipher^d \bmod n$

4.3 Example

```
Original message: HELLO
Message as numbers: [8,5,12,12,15]
Encrypted numbers: [203, 54, 92, 92, 199]
Decrypted numbers: [8,5,12,12,15]
Decrypted message: HELLO
```

5 RSA Educational Game

An additional **educational game** was implemented to help users manually understand RSA.

5.1 Highlights

- The RSA class uses both Miller and Euclid classes for prime generation and modular inverse computation.
- The game uses very small numbers so the reader can calculate encryption and decryption by hand.

5.2 Game Setup

```
p = 5
q = 13
n = p * q             # 65
phi = (p-1)*(q-1)     # 48
e = 5                 # public exponent
d = 29                # private exponent
c = 8                 # encrypted number (h -> 8)
```

5.3 Gameplay

1. User sees ciphertext c and keys (e, n) and (d, n).
2. User guesses the letter corresponding to c.
3. Program decrypts using $m = c^d \bmod n$ and converts to a letter.

5.4 Example Output

```
Welcome to the RSA letter-guessing game!
We have c=8, public key=(5,65), private key=(29,65)!
So can you guess the letter? h
Correct! You decoded the letter successfully!
```

6 Conclusion

This project demonstrates:

- Probabilistic prime generation with Miller-Rabin
- GCD and modular inverse computation with Euclidean algorithms
- RSA encryption/decryption workflow with educational examples
- An interactive RSA game for hands-on learning

Overall, this project provides a solid foundation for learning public-key cryptography and modular arithmetic principles.