**RSA Encryption and Decryption Algorithm Implementation**

**Experiment Objective:** Learn and deepen understanding of public key encryption systems, and implement a public key encryption and decryption algorithm based on RSA.

**Experiment Content:**

1.Prime number calculation: Based on the definition of prime numbers, create a program to calculate prime numbers starting from 2. Output the prime numbers from 0 to 500 in text format to "primer.txt".

2.Based on the previous step, use the program to select the largest prime number smaller than 300 and the smallest prime number larger than 300. These two prime numbers will be denoted as p and q. Generate the public key (e, n) and the private key (d, n) automatically.

3.Using the keys obtained in step 2, implement the RSA algorithm for encryption and decryption. Encrypt and decrypt the plaintext using the last four digits of your student ID.

Experiment Report:

a) Please explain in writing the method for determining whether a number is prime. How can the search for prime numbers be effectively accelerated? Can known small prime numbers be used to determine larger prime numbers?

To determine whether a number num is prime, we can divide num by numbers smaller than it, denoted as n, and obtain the remainder (n being less than or equal to the square root of num). If the remainder is 0, it means num is divisible by n and thus not prime. If the remainder is not 0 for any n, num is prime.

To accelerate the search for prime numbers, we can skip even numbers between 2 and the square root of num, as they cannot be prime. By checking only odd numbers, we can speed up the process.

One method to find prime numbers is to start with a small prime number in the range of 2 to num, and then eliminate its multiples. Repeat this process with subsequent small prime numbers, continuously narrowing down the range from 2 to num. If num remains in the range, it is a prime number; otherwise, it is not.

b) Given three integers B, n, and i, if B > n, how can we avoid generating a large integer Bi when calculating Bi mod n?

When B and i are large, Bi can become very large. To avoid this situation, we can use the property of taking the modulus of B by n to simplify the calculation.

If B > M and B = c\*M + r (0 <= r < M), then Bi mod M = ri mod M.

c) Given three integers B, n, and i, if B < n but i is very large, how can we avoid generating a large integer Bi when calculating Bi mod n?

By using the property of remainders:

If B < M and B2 = c\*M + r (0 <= r < M), then Bi+2 mod M = [(Bi mod M) \* r] mod M.

To find the value of B2 mod M, we can convert Bi mod M to (Bi-2 mod M) \* r mod M. By continuously applying this transformation, we can avoid generating a large integer Bi.

d) Divide the program algorithm into multiple steps and list the function declarations for each step. Write about each step's implementation, parameter meanings, and return value results.

bool isPrime(int num)

Check if a number is prime.

Parameters: num - a positive integer.

Return value: bool - true if num is prime, false otherwise.

int exgcd(int a, unsigned long int b, unsigned long int &x, unsigned long int &y)

Uses the extended Euclidean algorithm to find a set of solutions (x, y) for ax + by = gcd(a, b).

Here, it is used to find the values of e and d.

a: The value of e to be solved.

b: The value of (p-1)\*(q-1).

x: The value of d to be solved, also x in the aforementioned formula.

y: The value of y involved in the extended Euclidean algorithm.

Return value: ans - the greatest common divisor of a and b.

unsigned long int Plus(unsigned long int a, unsigned long int b, unsigned long int c)

Calculate (a + b) % c, as a preparation for calculating (a^b) % c.

Parameters:

a: unsigned long int.

b: unsigned long int.

c: unsigned long int.

Return value: unsigned long int - the result of (a + b) % c.

unsigned long int Multiplies(unsigned long int a, unsigned long int b, unsigned long int c)

Calculate (a \* b) % c, as a preparation for calculating (a^b) % c.

Parameters:

a: unsigned long int.

b: unsigned long int.

c: unsigned long int.

Return value: unsigned long int - the result of (a \* b) % c.

unsigned long int PowerMod(unsigned long int a, unsigned long int b, unsigned long int c)

Calculate (a^b) % c.

Parameters:

a: unsigned long int.

b: unsigned long int.

c: unsigned long int.

Return value: unsigned long int - the result of (a^b) % c.

e) Write down all the prime numbers within 500.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499.

f) How do you plan to use the RSA algorithm for encryption and decryption of Chinese and English strings? What is the efficiency of encryption?

Convert the Chinese and English strings into a sequence of numeric representations (grouping them, each group representing a fixed number of characters, then converting each character within the group into its ASCII code representation for English characters, and using unsigned char for Chinese characters). Then, use the RSA algorithm to encrypt the numeric representation of each group.

The encryption efficiency depends on the size of the message and the key length used in the RSA algorithm. Generally, RSA encryption is slower compared to symmetric encryption algorithms, especially for larger messages. However, it provides the advantage of secure key exchange and the use of public and private keys for encryption and decryption.

