

Encryption and Speed-up of RSA

CAN304 Group-28

Letian Xie Longyang Wang Xinzi Li
Xiaoyue Ma Tianlei Shi



CONTENT

- 01 | Introduction
- 02 | Problems Define
- 03 | Large Primes Generation
- 04 | Faster Decryption
- 05 | Pre-processed Plaintext
- 06 | Code Demo
- 07 | Experimental Result

Introduction

0
1

PART ONE

RSA Algorithm Process:

- Select Two large prime numbers p and q
- Calculate the product $n = pq$ with $\varphi(n) = (p - 1)(q - 1)$
- Select a random integer e ($\varphi(n) > e > 1$) with $\gcd(e, \varphi(n)) = 1$
- Calculate d such with $de = 1 \bmod \varphi(n)$
- For each key $k = (n, p, q, d, e)$, the encryption transformation is defined as $C = M^e \bmod n$, the decryption transformation is defined as $M = C^d \bmod n$, when C means the cipher text and M means the original message
- $\{e, n\}$ is the public key and $\{d, n\}$ is the private key.

Problems Define

0
2

PART ONE

- Inconvenient to use:
 - Require two large prime numbers
 - Limitations of prime number generation
- Slow:
 - Key length
 - Standardization of the data format

Large Primes Generation

0
3

PART ONE

- According to the prime number theorem, let $\pi(n)$ denote the number of primes less than n , then we have

$$\pi(n) \approx \frac{n}{\ln n} (n \rightarrow \infty)$$

- Assume integer r be chosen randomly in the interval $1 \leq r \leq 10^{100}$, and then we can get the following corollaries:

$$\text{Corollary I: } \Pr(r \text{ is prime}) = \frac{\pi(n)}{n} = \frac{1}{\ln n}$$

$$\text{Corollary II: } \Pr(r \text{ is prime} \mid r \text{ is odd}) = \frac{\pi(n)}{n/2} = \frac{2}{\ln n}$$

$$\text{Corollary III: } \Pr(r \text{ is prime} \mid r \text{ is odd and } \text{GCD}(r, 3) = 1) = \frac{\pi(n)}{n/2 \cdot (1 - 1/3)} = \frac{3}{\ln n}$$

Large Primes Generation

9

- The probability of generating prime numbers can become no lower than $\frac{3}{\ln n}$ by only generating odd integers and using small primes to verify (trial division), and this can be improved at least 200% compared to $\frac{1}{\ln n}$ in previous.

The procedure of large prime generation:

1. randomly generate an integer r in the interval $[n/2, n]$ based on the specified large integer n , and perform $r = r \times 2 + 1$
2. conduct trial division to check r by using all prime numbers less than 100, and set $r = r + 2$ and repeat step 2 if r failed in checking
3. conduct M-R test. If r failed in the test, set $r = r + 2$ and return to step 2, and we produce a prime otherwise

Faster Decryption

0
4

PART TWO

- Through THE CHINESE REMAINDER THEOREM, the problem of speeding up the decryption time can be well solved.
- THE CHINESE REMAINDER THEOREM :

premise: recipient knows the two prime numbers p and q

If $p < q$ then define A , which $0 < A < q-1$ and $A \cdot p \equiv 1 \pmod{q}$

Calculate: $d_p \equiv d \pmod{p-1}$, $d_q \equiv d \pmod{q-1}$

$$C_p \equiv C \pmod{p}, \quad C_q \equiv C \pmod{q}$$

$$M_p \equiv C^{d_p} \pmod{p}, \quad M_q \equiv C^{d_q} \pmod{q}$$

To decryption: $\tilde{M} = [((M_q + q - M_p) \cdot A) \pmod{q}] \cdot p + M_p$

Pre-processed Plaintext

0

5

PART TWO

- Encryption

Step1:

Convert every character of message M to binary representation according to Ascii table (every character must consist of 8 of 0s and 1s) from left to right.

Step2:

Convert whole representation(S) of M to decimal representation M_1

Step3:

Apply RSA encrypting function on the number M_1 and get number M_2

- Decryption

Step1:

Apply RSA decrypting function on the number M_2 and will get number M_1

Step2:

Convert M_1 to binary representation and ensure the number of digits must divided by 8 (we can add zeros to the left)

Step3:

Divide that representation into bytes (8 digits) from left to right

Step4:

Convert every bytes into character according to Ascii table

Code Demo

06
PART TWO

- The code demo is shown as below.

The screenshot shows the 'MainWindow' application. It features a grid of buttons for generating prime numbers and processing plaintext. The 'Plaintext' field contains the character 'I'. The 'Ciphertext' and 'Decryption text' fields are empty. The 'Encryption time' and 'Decryption time' fields are also empty.

Generate Prime Numbers	Plaintext process	Method
Big Prime Generation	Improve by Ascii	Quick power
	No treatment	Chinese remainder theorem
Big prime generation	No treatment	Quick power
Encryption	Decryption	Refresh
Python RSA		
Plaintext: I		
Ciphertext:		
Decryption text:		
Encryption time:		
Decryption time:		

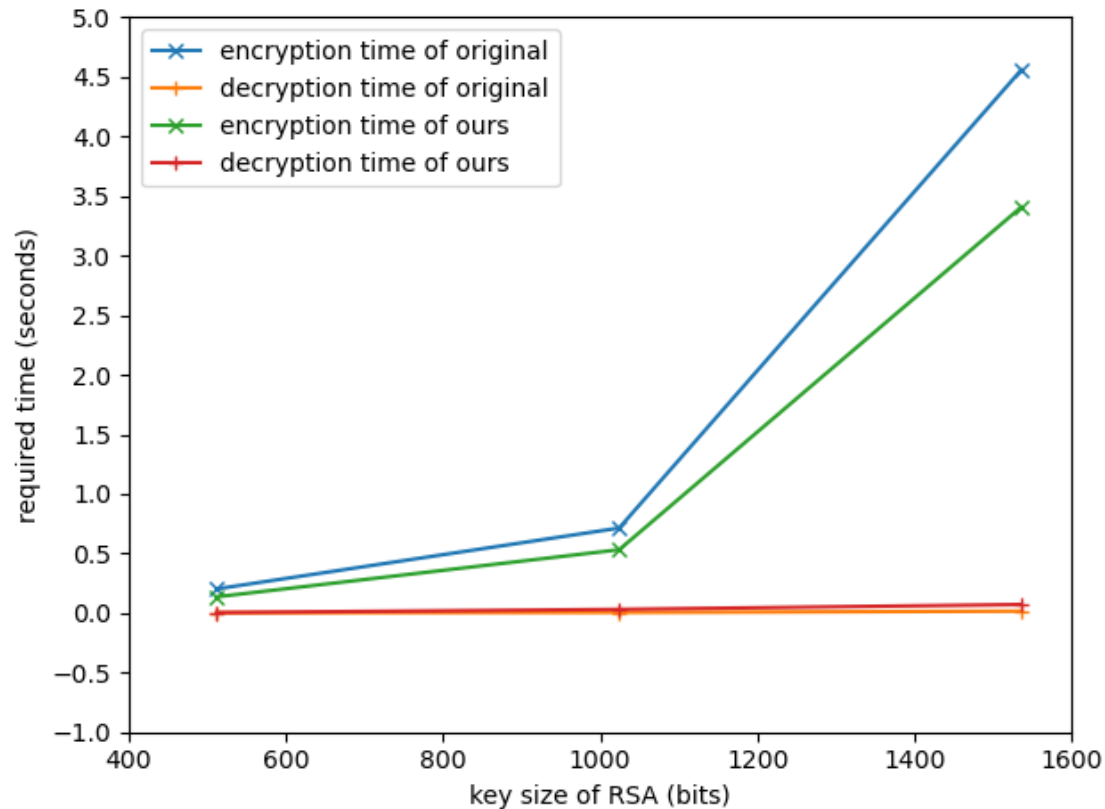
The screenshot shows the 'MainWindow' application after decryption. The 'Plaintext' field now contains 'hello'. The 'Ciphertext' field displays a long hexadecimal string. The 'Decryption text' field contains 'hello'. The 'Encryption time' field shows '0:00:00.181099' and the 'Decryption time' field shows '0:00:00.001993'.

Generate Prime Numbers	Plaintext process	Method
Big Prime Generation	Improve by Ascii	Quick power
	No treatment	Chinese remainder theorem
Python RSA	Python RSA	Python RSA
Encryption	Decryption	Refresh
Python RSA		
Plaintext: hello		
Ciphertext: b'\x91U\x84\nWes\xfa\xab\x8f\x04\xa4\xe1\xc2\xddQM\xe4\x94E\xac\xa97\xd3\x8a\xd1\xb0'		
Decryption text: hello		
Encryption time: 0:00:00.181099		
Decryption time: 0:00:00.001993		

Experimental Result

07 PART TWO

- The results show that the improved version has significantly improved the encryption speed for key sizes larger than 1000 bits.



Thank You