



Speed Improvement for the RSA Encryption Method

Hsiu-Chiung Wang , Thesis for master
Philosophy of computer department in
Napier University



Outline

- Introduction
- Introduction to cryptology
- Number theory
- RSA algorithm / Rapid Module Exponentiation
- How to get greater prime number / Choice of RSA parameter
- New and old system / Novel Implementation of RSA Algorithm
- Conclusion



Introduction

- Research Motive
- Research purpose
- Main contribution
- Research method



Research Motive / purpose



- Now , Internet communication , Communication network E_commerce are more popular in our real life,So , there is a great need for privacy and security transmitted data. Then , the public key cryptosystem have been created in my thesis. I also design a public key cryptosystem to improve the encryption, decryption and key generation time



Main Contribution

- This system was spent a lots time to do it and already implement complete and also apply in the real life.
- This system used four skills to improve speed
- The speed improvement as following:

Items		New system (Block: number of bits)	Old system
Module exponentiation		M-ary Sliding window method	Right to left binary method
Number of bits		256-2048 bits	512 bits
The improvement Of efficient	Key generation	13.0 seconds	58.0 seconds
	(estimated) Encryption	5.5 ms /block	95 ms/seconds
	Decryption	251.4 ms/ block	420 ms/ block



Research Method

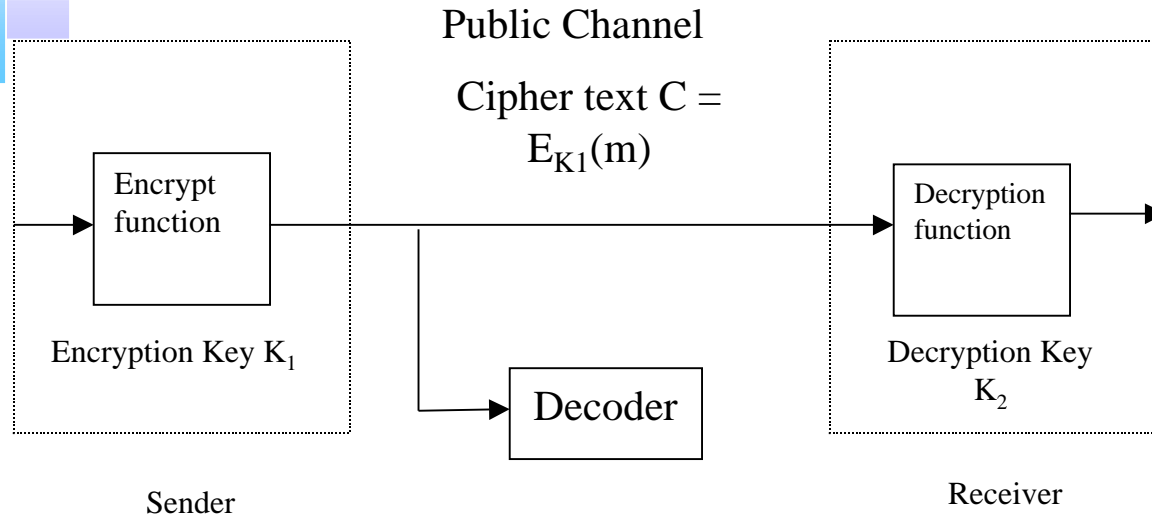
- RSA Algorithm
 - SPM (Single precision Multiplication
 - Tabulation Method
 - M-ary sliding window method
 - CRT (Chinese Remainder Theorem)



Introduction to cryptology

- Typical cryptosystem
- Classic cryptosystem (Symmetric Cryptosystem)
 - Permutation Encryption method
 - Substitution Encryption method
- Public key cryptosystem (Asymmetric cryptosystem)

Typical Cryptosystem



- If $k_1 = k_2$ and transfer key from sender to receiver by private channel, it is called *classic cryptosystem*
- If $k_1 \neq k_2$ and transfer k_1 from sender to receiver by public channel, It is called *public key cryptosystem*



Permutation Encryption method

- Geometric Permutation
- Column and Row Permutation
- Mixing Permutation



Geometric permutation

Plaintext : ABCDEF.....Z

- A F K P U Z

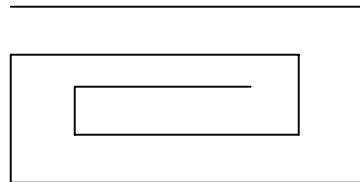
B G L Q V

C H M R W

D I N S X

Z J O T Y

- Following order $k = \textit{rec5 t2}$



- Cipher text

AFKPUZBGLQVCHMRWDINSXEJOTY¹⁰

Column / Row permutation

- Plaintext

ABC Z

1 2 3 4 5 6

A F K P U Z

B G L Q V

C H M R W

D I N S X

E J O T Y

- Key = Col 5 (4,2,5,1,6,3)

- 4 2 5 1 6 3

P F U A Z K

Q G V B L

R H W C M

S I X D N

T J Y E O

- Cipher text

PQRSTUVWXYZYABCD
EJKLMNO



Mixing Permutation

- Plaintext

ABCD.....Z

- Key ---Rec5 t2 + Row(4,1,3,2)

1. Rec5 t2- please refer to Geometric permutation

Cipher text

AFKPUZBGLQVCHMRWDINSXEJOTY

2. Row (4,1,3,2)

AUOCQSM	1	PTDLXH	4
---------	---	--------	---

FZJBVNR	2	AUOCQSM	1
---------	---	---------	---

KYEGWI	3	KYEGWI	3
--------	---	--------	---

PTDLXH	4	FZJBVNR	2
--------	---	---------	---

- Cipher text

PTDLXHAUOCQSMKYEGWIFZJBVNR




Substitution Encryption method

- Caesar shift
- Substitution Related to the Environment



Caesar Shift

- A • 0 , B • 1 , C • 2 , D • 3 , E • 4 , F • 5 , G • 6 , H • 7 , I • 8 ,
J • 9 , K • 10 , L • 11 , M • 12 , N • 13 , O • 14 , P • 15 , Q • 16 , R • 17 ,
S • 18 , T • 19 , U • 20 , V • 21 , W • 22 , X • 23 , Y • 24 ,
Z • 25
- Encoded $T_k(P) = (P+K) \bmod 26$
- Decoded $T_k^{-1}(C) = (C-K) \bmod 26$
-  **Example**
Plain text = " SECURITY "
Key = " 18,4,2,20,17,8,19,24 "
Follow Encoded $T_k(P) = (P+3) \bmod 26$
Cipher key : " 21,7,5,23,20,11,1 " Cipher text : " VHFXULWB "
Follow Decoded $T_k^{-1}(C) = (C-3) \bmod 26$
so Plain Text : " SECURITY "



The Defect of Private key Cryptosystem

- Key Distribution

How does the sender and receiver obtains the respective encryption and decryption key ?

- The number of keys is substantial

 **Example :**

n peoples in network , so need $n(n-1)/2$ keys,


if $n = 1000$ people

then key = $(1000 * 999) / 2 = 499500$

- Unable to achieve the service of non-repudiation



One way function / One way trapdoor function

- $Y = f(x)$ for all x , It is easily to solve y
- It is hard to solve $x = f^{-1}(y)$
-  **Example**
 - a. a mail box
 - b. putting mail in a mail box – public activity
 - c. opening the mail box – secret activity



RSA Algorithm / Rapid Modular Exponentiation

- Fermat ' s theorem / Euler ' s Theorem
- RSA algorithm
- Rapid Modular Exponentiation



Fermat's theorem / Euler's theorem

- Fermat's theorem :

If p is a prime number and $\text{GCD}(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$.

-  **Example** :

$$a=7, p=19$$

$$7^2 = 49 \equiv 11 \pmod{19}$$

$$7^4 = 121 \equiv 7 \pmod{19}$$

$$7^8 = 49 \equiv 11 \pmod{19}$$

$$7^{16} = 121 \equiv 7 \pmod{19}$$

$$a^{p-1} = 7^{18} = 7^{16} * 7^2 \equiv 7 * 11 \equiv 1 \pmod{19}$$



Fermat's theorem / Euler's theorem

- Euler's theorem
- $\text{GCD}(a, n) = 1$, and $a^{\phi(n)} \equiv 1 \pmod{n}$.
- $\phi(n)$ is the positive integer, less than n and relative prime to n . $\phi(n)$ is called Euler's totient number.

Example

$a=2$, $n=11$, So $\text{GCD}(2, 11) = 1$, $\phi(11) = 10$

$$2^{\phi(n)} = 2^{10} = 1024 \equiv 1 \pmod{11}$$



RSA Algorithm



- **RSA Algorithm cryptography as the following.**

1. Key generation:

- a. Choose two prime numbers p and q .
- b. Calculate $n = p \times q$
- c. Calculate $\phi(n) = \phi(p \times q) = (p-1)(q-1)$
- d. Select one e , which satisfies $\gcd(\phi(n), e) = 1$. $1 < e < \phi(n)$
- e. As $ed = 1 \pmod{\phi(n)}$, utilize Euclid's Algorithm to get $d = e^{-1} \pmod{\phi(n)}$.
- f. A pair of public keys is obtained, which is $\{e, n\}$.
A pair of secret keys is obtained, which is $\{d, n\}$.

2. encrypting:

- a. Plain text M and $M < n$.
- b. Encoded text $C = M^e \pmod{n}$

3. decrypting:

- a. Cryptographic text is C .
- b. Plain text can be obtained through decoding, which gets
 $M = C^d \pmod{n}$

RSA Algorithm

Example :

1. Key Generation:

- a. Choose two prime numbers p and q , $p=7$, $q=17$
- b. Calculate $n = p \times q = 7 \times 17 = 119$
- c. Calculate $\phi(n) = (p-1)(q-1) = 6 \times 16 = 96$
- d. Select one e , which makes $\gcd(\phi(n), e) = 1$. $1 < e < \phi(n)$, $e=5$
- e. Utilize Euclid's Algorithm to get $d = e^{-1} \bmod \phi(n) = 5^{-1} \bmod 96 = 77$.
- f. A pair of public keys is obtained, which is $\{e, n\} = \{5, 119\}$.
A pair of secret keys is obtained, which is $\{d, n\} = \{77, 119\}$.

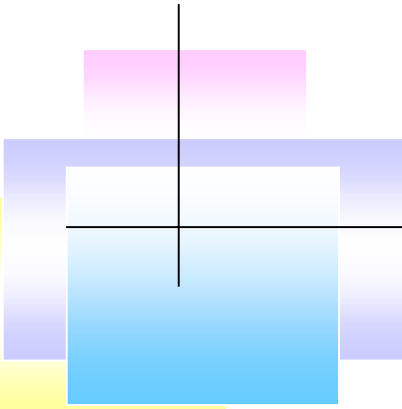
2. Encrypting:

The public key is $\{e, n\} = \{5, 119\}$. We choose the plain text $M = 20$, then $C = M^e \pmod{n} = 20^5 \bmod 119 = 3200000 / 119 = 26890 \dots 90$ (remainder). Therefore, cryptographic text is 90.

3. Decrypting:

The secret key is $\{d, n\} = \{77, 119\}$. Then $M = C^d \pmod{n} = 90^{77} \bmod 119$
 $= (8^{38} * 90) \bmod 119 = (8^{36} * 8 * 90) \bmod 119 = (36^{12} * 48) \bmod 119$
 $= (106^6 * 48) \bmod 119 = (50^3 * 48) \bmod 119 = 20$ (remainder)

Rapid Modular Exponentiation



- **Binary method**
- Input M.e.n
- Output: $C = M^e \text{ mod } n$
- 1. if $e_{k-1} = 1$, then $C := M$, else $C := 1$
- 2. for $i = k-2$ down to 0
 - 2a: $C := C * C \text{ (mod } n)$ (square)
 - 2b: if $e_i = 1$, then $C := C * M \text{ (mod } n)$ (multiply)
- 3. Return

-  **Example**

$e = 506 = 111111010$, $k = 9$. As the initial value is 1, $C := M$, $e_{k-1} = e_{9-1} = e_8 = 1$

I	e_i	Step 2a	Step 2b
7	1	$(M)^2 = M^2$	$M^2 * M = M^3$
6	1	$(M^3)^2 = M^6$	$M^6 * M = M^7$
5	1	$(M^7)^2 = M^{14}$	$M^{14} * M = M^{15}$
4	1	$(M^{15})^2 = M^{30}$	$M^{30} * M = M^{31}$
3	1	$(M^{31})^2 = M^{62}$	$M^{62} * M = M^{63}$
2	0	$(M^{62})^2 = M^{126}$	M^{126}
1	1	$(M^{126})^2 = M^{252}$	$M^{252} * M = M^{253}$
0	0	$(M^{252})^2 = M^{504}$	M^{506}

Rapid Modular Exponentiation

- **M-ary Method:**

The algorithm of the m-ary method is as follows.

Input = M.e.n

Output: $C = M^e \pmod n$

1. calculate and save $M^w \pmod n$ for all; $w = 2, 3, 4, \dots, m-1$
2. Decompose e into r-bit words F_i for $i=0, 1, 2, \dots, s-1$
3. $C := M^{F_{s-1}} \pmod n$
4. for $i = s-2$ down to 0
 - 4a: $C := C^{(2)**r} \pmod n$
 - 4b: if $F_i \neq 0$, then $C := C * M^{F_i} \pmod n$
5. Return

(Data Source: Cetin Kaya Koc, 1994, " High Speed RSA Implementation", pp.12)

Rapid Modular Exponentiation

- **Quaternary Method:**

bits	w	M^w
00	0	1
01	1	M
10	2	$M * M = M^2$
11	3	$M^2 * M = M^3$

-  **Example**

$E = 506 = \underline{01} \underline{11} \underline{11} \underline{10} \underline{10}$

$s = k \text{ (2-based bit)} / r \text{ (2 bits scanned one time)} = 10 / 2 = 5$
(groups).

I	F_i	Step 4a	Step 4b
3	11	$((M)^2)^2 = M^4$	$M^4 * M^3 = M^7$
2	11	$((M^7)^2)^2 = M^{28}$	$M^{28} * M^3 = M^{31}$
1	10	$((M^{31})^2)^2 = M^{124}$	$M^{124} * M^2 = M^{126}$
0	10	$((M^{126})^2)^2 = M^{504}$	$M^{504} * M^2 = M^{506}$

Rapid Modular Exponentiation

Octal Method:

Example

$$e = 506 = 0\ 111\ 111\ 010 = \underline{111}\ \underline{111}\ \underline{010}.$$

$$s \text{ (group number)} = k \text{ (number of bits)} / r \text{ (one group of } r \text{ bits)} = 9 / 3 = 3$$

bits	w	M^w
000	0	1
001	1	M
010	2	$M * M = M^2$
011	3	$M^2 * M = M^3$
100	4	$M^3 * M = M^4$
101	5	$M^4 * M = M^5$
110	6	$M^5 * M = M^6$
111	7	$M^6 * M = M^7$

Accordingly, $C := M^{F2} = M^7 \text{ mod } n$

i	F_i	Step 4a	Step 4b
1	111	$((M^7)^2)^2 = M^{56}$	$M^{56} * M^7 = M^{63}$
0	010	$((M^{63})^2)^2 = M^{504}$	$M^{504} * M^2 = M^{506}$

Rapid Modular Exponentiation

- The algorithm of the **sliding window method** is as follows.

Input: M, e, n

Output: $C = M^e \bmod n$

- 1. Calculate and save M^w , which results in $w = 3, 5, 7, \dots, 2^d - 1$.
- 2. Divide e into zero and nonzero windows F_i .
The length of F_i is $L(F_i)$ for $i = 0, 1, 2, \dots, p-1$.
- 3. $C := M^{F_{k-1}} \bmod n$
- 4. for $i = p-2$ down to 0
 - 4a. $C := C^{2^{L(F_i)}} \bmod n$
 - 4b. If $F_i \neq 0$, then $C := C * M^{F_i} \pmod n$
- 5. Return C

(Data source : Cetin Kaya Koc, High-Speed RSA Implementation 1994, pp16)




Rapid Modular Exponentiation

- Sliding window method (Constant Length nonzero window)
- **ZW**: Examine the bit from the right to the left. When the entered single bit is 0, remain at ZW; otherwise, go to NW.
- **NW**: When the entered bit is 1, stay at the nonzero window until d bit is collected. Resume checking. If the entered single bit is 0, go to ZW; otherwise, stay at NW.
- **CLNW** partitioning strategy
 1. produces zero windows of arbitrary length
 2. Produces nonzero windows with length of d

(Data Source: Cetin Kaya Koc, 1994, " High-Speed RSA Implementation",pp.17.)

 -

Rapid Modular Exponentiation

- Sliding window method with CLNW(Constant Length Nonzero Window)
-  **Example** : We presume $d = 3$ for the following example and $E = 3828227 = 111\ 01\ 00\ 11\ 0101\ 000000\ 011$.

In compliance with the division strategy of the constant variable nonzero windows, it comes out like:

$$\begin{array}{cccccccc} \underline{011} & \underline{101} & 0 & \underline{011} & 0 & \underline{101} & 000000 & \underline{011} \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 = I \end{array}$$

i	Fi	L(fi)	Step4a	Step 4b
6	101	3	$(x^3)^8 = x^{24}$	$x^{24} \cdot x^5 = x^{29}$
5	0	1	$(x^{29})^2 = x^{58}$	x^{58}
4	011	3	$(x^{58})^8 = x^{464}$	$x^{464} \cdot x^3 = x^{467}$
3	0	1	$(x^{467})^2 = x^{934}$	x^{934}
2	101	3	$(x^{934})^8 = x^{7472}$	$x^{7472} \cdot x^5 = x^{7477}$
1	000000	6	$(x^{7477})^{64} = x^{478528}$	x^{478528}
0	011	3	$(x^{478528})^8 = x^{3828224}$	$x^{3828224} \cdot x^3 = x^{3828227}$



Rapid Modular Exponentiation

- Sliding window method with VLNW (Variable Length Nonzero Window)
- The VLNW has two integer parameters as follows.
 - d: maximum nonzero window length
 - q: The minimum number of zeros required to switch to ZW.
- The algorithm process of VLNW is shown as the following.

ZW: Check each single entered bit. If the bit is 0, then stay at ZW; otherwise, skip to NW.

NW: Check entered q bits. If q bits are all 0s, then skip to ZW; otherwise, remain stay at NW.

(Data Source: Cetin Kaya Koc, 1994, " High-Speed RSA Implementation",pp.18-19, 1994.)

VLNW method

▪  **Example :**

We can describe the VLNW method through the previous example, which is $E = 3828227$.

$E = 3828227 = 111\ 01\ 00\ 11\ 01\ 01\ 0000000\ 11$; $d = 3$ and $q = 1$

Divided into: 111 0 1 00 11 0 101 0000000 11

8 7 6 5 4 3 2 1 0 = i

Therefore, $y = x^{F8} = x^7$ and the algorithm process of $x^{3828227}$ is as follows.

i	Fi	L(fi)	Step4a	Step 4b
7	0	1	$(x^7)^2 = x^{14}$	x^{14}
6	1	1	$(x^{14})^2 = x^{28}$	$x^{28} \cdot x = x^{29}$
5	00	2	$(x^{29})^4 = x^{116}$	x^{116}
4	11	2	$(x^{116})^4 = x^{464}$	$x^{464} \cdot x^3 = x^{467}$
3	0	1	$(x^{467})^2 = x^{934}$	x^{934}
2	101	3	$(x^{934})^8 = x^{7472}$	$x^{7472} \cdot x^5 = x^{7477}$
1	0000000	7	$(x^{74717})^{128} = x^{957056}$	x^{957056}
0	11	2	$(x^{957056})^4 = x^{3828224}$	$x^{3828224} \cdot x^3 = x^{3828227}$

According to the paper published by Koc in 1995 , when the q are between 1 and 3 for $4 \cdot d < 8$ and $128 \cdot k \cdot 2048$, the **VLNW reduces 5 – 8% of multiplications than the m-ary method**



How to get greater prime number / Choice of RSA parameter

How to get greater prime number

1. Use the c-standard function `rand`, creating a random number of k bits – `Void randl()`
2. When random number generate then put then into a array, so it can get a great random number – `Void keygen(void)`
3. Use Miller Robin method to do Probabilistic primality test – `word primetest()`
4. Creating prime numbers whose length is level 3 of $plen$ – `void getprime()`
5. Creating prime numbers whose length is the level2 of $12plen$ – `void level2_prime()`
6. Creating prime numbers whose length is level 1 of $splen$ – `void greater prime()`

How to get greater prime number / Choice of RSA parameter

How to get greater prime number p

[Theorem]: From the above, we know

$$\begin{aligned} p_1 &= 2k_1 r_1 + 1 & p_2 &= 2k_3 r_2 + 1 \\ p_1 &= 2k_2 s_1 - 1 & p_2 &= 2k_4 s_2 - 1 \end{aligned}$$

When $ss^{-1} = 1 \pmod r$ (that is, $ss^{-1} = k_5 r + 1$)

$$\text{Then } p_1 = 2ss^{-1} - 1 + 2k_5 r s$$

From $p_1 = 2ss^{-1} - 1 + 2k_5 r s$ then

$$\begin{aligned} p_1 \pmod{2s} & \\ &= (2ss^{-1} - 1 + 2k_5 r s) \pmod{2s} \\ &= -1 \pmod{2s} \end{aligned}$$

$$\begin{aligned} p_1 \pmod{2r} & \\ &= (2ss^{-1} - 1 + 2k_5 r s) \pmod{2r} \\ &= [2(k_5 r + 1) - 1] \pmod{2r} \\ &= (2k_5 r + 2 - 1) \pmod{2r} \\ &= 1 \pmod{2r} \end{aligned}$$

So, $p_1 = 1 \pmod{2r} = -1 \pmod{2s}$ ----- (4-1)

So, using $r_1 s_1$ get p_1 from 4-1,
using $r_2 s_2$ get p_2 from 4-1,
using $p_1 p_2$ get greater prime number P

Example :

$r_1 = 4e64575$	$384d29e3$		
$s_1 = 672551a$	$122f7495$		
$p_1 = e205d24$	$b2da69b4$	$eb4d61fe$	$07f9e701$
$r_2 = 6817125$	$16e62879$		
$s_2 = 69043c9$	$2d1f4f23$		
$p_2 = 16fb6edd$	$7abdabcc$	$364932d6$	$e1b08683$
$p = 946af43f$	$a33b2c54$	$9f4ecda7$	$bc715cbc$
$8a243c7c$	$88fa2977$	87733623	$a5005c6f$



Choice of RSA parameter

- **Choice the parameter e**

BRUCE SCHNEIER, recommend $e = 3$ or 17 , or 65537

I choice $e = 65537 (2^{16} + 1)$ in this new program, It only have two one so only need 17 multiplications



Novel Implementation of RSA Algorithm



- Old system

software: C language

machine : pentium 3 500

bit length: 512 bits

applied method :

Multiplication(bit by bit)

Long division

Binary method

Euclid algorithm

- New system

software: C language

machine : pentium 3 500

bit length: 256-2048 bits

applied method :

SPM (16 bits as one block)

Tabulation method

m-ary sliding window method

CRT

Euclid algorithm

Novel Implementation of RSA Algorithm

- Main system flow path



Novel Implementation of RSA Algorithm

(3) Decryption by CRT(function crt_decrypt):

Step 1: read secret key data $m_1, p, q, d_p, d_q, d, n$ from secret key file

Step 2: open the source file for decryption

Step 3: open the target file for storing the decrypted data

Step 4: evaluate the source file length($flen$) & the number of blocks($blen$)

Step 5: build the table of $p, 2p, 4p, 8p, \dots, 2^{16}p$ & $q, 2q, 4q, 8q, \dots, 2^{16}q$

Step 6: pre-compute $r = p^{-1} \bmod q$

Step 7: read one block message(msg) from the source file

Step 8: evaluate $msg = msg_1$

Step 9: evaluate $msg = (msg \bmod p)^{d_p} \bmod p$

Step 10: evaluate $msg_1 = (msg_1 \bmod q)^{d_q} \bmod q$

Step 11: evaluate $msg = ((msg_1 - msg)r \bmod q)p + msg$

Step 12: write msg to the target file.

Step 13: if it remained data for decryption goto step 6

Step 14: close the source and target files

The Comparison of two systems

The Speed Comparison of New system and old system as following :

Items		New system (Block: number of bits)	Old system
Module exponentiation		M-ary Sliding window method	Right to left binary method
Number of bits		256-2048 bits	512 bits
The improvement Of efficient	Key generation (estimated)	13.0 seconds	58.0 seconds
	Encryption	5.5 ms /block	95 ms/seconds
	Decryption	251.4 ms/ block	420 ms/ block



The Comparison of two systems

- **With adoption of the following methods, this new program has made a great improvement in key generation, Encryption and Decryption speed. The methods include:**
 - (1) Single Precision Multiplication, SPM
 - (2) Use tabulation method to complete the modulus calculation in the modulus multiplication
 - (3) Use m -ary sliding window to complete exponent calculation
 - (4) Use Chinese Remainder Theorem

Tabulation method

- $Z = X * Y \text{ mod } n$

$X : X_{32} \dots\dots\dots X_2 X_1$ (every X is 16 bits)

$Y : \text{Large number}$

$Z_i = X_i * Y \text{ mod } n$

Pre-calculate and save $2^0n \dots\dots\dots 2^{16}n$ into an array a[17]

Listing method to save a lot of time spent on shift and calculation

Repeat

when $(X_i * Y) \cdot 2^{16}n$ then $(X_i * Y) - 2^{16}n$

when $(X_i * Y) \cdot 2^{15}n$ then $(X_i * Y) - 2^{15}n$

..... •..... •..... •..... •..... •..... •.....


when $(X_i * Y) \cdot 2^0n$ then $(X_i * Y) - 2^0n$

until $(X_i * Y) < n$

Array a[17]:

2^0n	a_0
2^1n	a_1
•	•
•	•
•	•
$2^{16}n$	a_{16}

M-ary Sliding Window

-  **Example** : $E = 3828227 = 111\ 01\ 00\ 11\ 01\ 01\ 0000000\ 11$; $d = e$ and $q = 1$

Divided into: $\underline{111}\ \underline{0}\ \underline{1}\ \underline{00}\ \underline{11}\ \underline{0}\ \underline{101}\ \underline{0000000}\ \underline{11}$
 $8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0 = i$

Therefore, $y = x^{F8} = x^7$ and the algorithm process of $x^{3828227}$ is as follows.

i	Fi	L(Fi)	Step 4a	Step 4b
7	0	1	$(x^7)^2 = x^{14}$	x^{14}
6	1	1	$(x^{14})^2 = x^{28}$	$x^{28} \cdot x = x^{29}$
5	00	2	$(x^{29})^4 = x^{116}$	x^{116}
4	11	2	$(x^{116})^4 = x^{464}$	$x^{464} \cdot x^3 = x^{467}$
3	0	1	$(x^{467})^2 = x^{934}$	x^{934}
2	101	3	$(x^{934})^8 = x^{7472}$	$x^{7472} \cdot x^5 = x^{7477}$
1	0000000	7	$(x^{7477})^{128} = x^{957056}$	x^{957056}
0	11	2	$(x^{957056})^4 = x^{3828224}$	$x^{3828224} \cdot x^3 = x^{3828227}$

According to the paper published by Koc in 1995²⁴, when the q is between 1 and 3 and $4 \cdot d < 8$, the number of bits, k , is **128 • k • 2048**, the **VLNW reduces 5 – 8% of multiplications than the m-ary method.**

Chinese Remainder Theorem

- $M = C^d \bmod n$
- Let p_i for $i = 1, 2, \dots, k$ be pairwise relatively prime integers, i.e.,

$$\gcd(p_i, p_j) = 1 \text{ for } i \neq j$$

Given $u_i \rightarrow [0, p_i - 1]$ for $i = 1, 2, \dots, k$ the Chinese remainder theorem states that there exists a unique integer u in the range $[0, P - 1]$ where $P = p_1 p_2 \dots p_k$ such that

$$u = u_i \bmod p_i$$

The Chinese remainder theorem tells us that the computation of

$$M = C^d \bmod n$$

Can be broken into two parts as

$$M_1 = C^d \bmod p$$

$$M_2 = C^d \bmod q$$

$$M_1 = C^d \bmod p$$

$$M_2 = C^d \bmod q$$

Computation M_1 : $3/2 (k/2)$ $(k/2)$ – bit multiplications

Computation M_2 : $3/2 (k/2)$ $(k/2)$ – bit multiplications

Computation M : One $(k/2)$ -bit subtraction
 Two $(k/2)$ -bit multiplications
 One k -bit addition

$$2 * 3k/4 * (k/2)^2 + 2(k/2)^2 + (k/2) + k = 3k^3/8 + (k^2 + 3k)/2$$

also, binary method need $3k^3/2$

$$(3k^3/8) / (3k^3/2) = 1/4$$

Thus, the CRT method based algorithm will be approximately 4 times faster than Binary method.

Data Source: Cetin Kaya Koc, "High-Speed RSA Implementation", 1994, pp53.



New Program Demo

Program implementing file is " new rsa," and if implemented directly, you will see the following message from the screen

Please using `rsaw -e` for encryption by RSA.

Please using `rsaw -d` for decryption by RSA.

Please using `rsaw -kg` for RSA key generation.

Or press 'e' for encryption.

press 'd' for decryption.

press 'k' for key generation.

press other key for exit.