Computational Reduction of Wilson's Primality Test for Modern Cryptosystems

Chia-Long Wu
Department of Aviation & Communication Electronics, Chinese Air Force Institute of Technology
Kaohsiung, Taiwan, R.O.C.
E-mail: chialongwu@seed.net.tw

Der-Chyuan Lou and Te-Jen Chang
Department of Electrical Engineering, Chung Cheng Institute of Technology, National Defense University
Taoyuan, Taiwan, R.O.C.
E-mail: {dclouprof, karl591218}@gmail.com

Keywords: information security, cryptography, modular arithmetic, primality test, number theory

Received: March 18, 2008

In this paper, a method of diminishing computational reduction to improve Wilson's primality test method is proposed. Basically, the RSA algorithm entails a modular exponentiation operation on large integers, which is considerably time-consuming to implement. Since ancient time, number theory has been an important study subject and modular arithmetic has also been widely used in cryptography. The Wilson's primality test method is one of the most well-known deterministic prime number test methods. It states that n *is a prime number if and only if* $(n-1)! \equiv -1 \bmod n$. *In this paper, we compare two primality test algorithms for implementing the Wilson's method, which need* $2*[(\frac{n-1}{2})!]$ *and* $n(\log_2 n)^2$ *multiplications, respectively. However, by using the proposed reduction algorithm, only* $\frac{n+1}{2}*[1-(\frac{1}{2})^k]+1$ *multiplications are needed for the Wilson's primality test method, where* $k=\left\lceil \log_2 \frac{n-1}{2} \right\rceil$ *and the "n" means a prime number. The proposed computational reduction method can efficiently perform Wilson's deterministic primality test, and it is faster than other proposed methods. By using the proposed method, it can not only reduce the overall computational complexity of the original Wilson's primality test method but also reduce the computational space.*

Povzetek: Opisana je metoda redukcije za moderne kriptografske sisteme.

1 Introduction

Modular exponentiation (ME) is the cornerstone computations performed in public-key cryptosystems. Taking the RSA cryptosystem [1] for example, the public and private keys are functions of a pair of large prime numbers, and the encryption and decryption operations are accomplished by modular exponentiation.

This modular exponentiation problem can be described as follows. Given M (message), E (public key), and N (the product of two large primes), compute ciphertext C ≡ ME mod N. For the computation of modular exponentiation, the very intuitive way is to break the modular exponentiation operation into a series of modular multiplications.

Meganet corporation [2] has announced its 13-year research results in the prime number testing area. Meganet corporation has implemented the algorithm in an ANSI C application running on a single CPU 450 MHZ PC. Some results of Meganet corporation are depicted in Table 1.

Wilson's primality test method states that n is a prime number if and only if $(n-1)! \equiv -1 \bmod n$. In this paper we compare two algorithms by its multiplication numbers for implementing the Wilson's primality method: Naccache-Donio's needs $2*[(\frac{n-1}{2})!]$ multiplications from a little trick about factories [3] and Rosen's method needs $[n(\log_2 n)^2]$ multiplications [4-9].

To design a fast primality test for finding a prime number is necessary and very important. We apply a method based on the modular arithmetic to advance the Wilson's primality test. The motivation of this paper is to reduce the numbers of multiplication, modular multiplication, and square. Besides, we will describe how to have a better space usage by using the proposed method.

The paper is organized as follows. In Section 2 we describe modern primality test methods such as probabilistic and deterministic primality test methods. The Wilson’s primality test method and some mathematical preliminaries are introduced in Section 3, and the proposed method using modular arithmetic is described in details. In Section 4 we analyze the computational complexity and area usage for our proposed improved Wilson’s primality test method and compare the performance with Naccache-Donio’s method [3] and Rosen’s method [4-9]. In Section 5 we draw some figures and tables to compare the above different methods.

Table 1. Experiments of Meganet corporation.

Bit number of the primality test	Time (in second)
1,000	0.5
2,000	1
3,000	3
4,000	8
5,000	15
6,000	26
7,000	41
8,000	62
9,000	87
10,000	118

2 Modern primality test methods

For the modern primality test theory [10-15], two fields of test methods have been published. They can enhance the security in public key cryptosystem such as probabilistic primality tests and deterministic primality tests. They are Solovay-Strassen, Lehman, Lucas, Miller-Rabin methods and so on, which have been issued in the probabilistic primality test field [16, 17]. We also have other primality test papers which have been issued in the deterministic test field [18-20] such as Demytko, Wilson, Proth methods etc.

2.1 Probabilistic primality test methods

2.1.1Fermat probabilistic primality test method:

This theorem assures us that if n is a prime number then $b^{n-1} \equiv 1 \bmod n$ for every integer b co-prime to n. In contrast, if n is a composite number, it is quite rare for the above congruence to be satisfied with b.

2.1.2Lucas probabilistic primality test method:

For any two nonzero integers, this equation is $D = a^2 - 4b \neq 0$. We define Lucas sequence as $U_k = \frac{\alpha^k - \beta^k}{\alpha - \beta}$, for $k \geq 0$, and α, β are two roots of the equation $x^2 - ax + b = 0$. If p is a prime number, p cannot divide b, and p will satisfy this equation $\frac{D}{p} = -1$, where $\frac{D}{p}$ is Jacobi symbol. We can get p |Up+1. So we use this principle to presume that if n is a positive odd number and n can not divide Un+1, then n is a composite number.

2.1.3 Miller-Rabin probabilistic primality test method:

Given a positive odd integer n and let $n = 2^r s + 1$, where s is an odd number. Then follow the testing numbers: choose a random positive integer a with $1 \leq a \leq n-1$. If $a^s \equiv 1 \bmod n$ or $a^{2^j s} \equiv -1$ mod N for some $0 \leq j \leq r-1$, then n passes the test. A prime number will pass the test for all a.

2.2 Deterministic primality test methods

Compared with probabilistic primality test methods, the output results of deterministic primality test methods are absolutely correct. In other words, when a positive odd number is tested, the output result has only two possible situations by using deterministic primality test methods. Either this number is a prime number or this number is a composite number. By using this method, the found number can be assumed as whether this number is a prime number or not.

2.2.1 Demytko deterministic primality test method:

If “ $p_{i+1} = h_i * p_i + 1$ ” meets the four following conditions, then pi+1 is sure to be a prime number.

(a) Input a positive odd prime number p_i . Let it be regarded as a seed generating prime number. We also look for them by using Look-Up Table (LUT) or other primality test methods.

(b) For h_i<4(p_i+1) *Hi*, *hi* is an even number, so we must use all of the even numbers from 2 to h_i during the test procedures.

(c) $2^{h_i p_i} \equiv 1 \bmod p_{i+1}$.

(d) $2^{h_i} \neq 1 \bmod p_{i+1}$.

2.2.2 Wilson deterministic primality test method:

If and only if n is a prime number, then (n-1)! +1 is a multiple of n, that is

$(n-1)! \equiv -1 \bmod n$.

This theorem was proposed by John Wilson and published by Edward Waring in 1770 though it was previously known for Leibniz [19]. It was proved by Lagrange [20] in 1773. Unlike the Fermat probabilistic primality test method, the Wilson's theorem is not only necessary but also sufficient for the primality test.

2.2.3 Proth deterministic primality test method:

For $N = k * 2^n + 1$ with k odd and $2^n > k$, if there exists an integer α such that

$\alpha^{(\frac{N-1}{2})} \equiv -1 \bmod N$,

then N is prime.

The difference between probabilistic primality test methods and deterministic primality test methods is that the result of the later methods can be precisely accurate. Namely, we are sure the number we calculate is a prime number.

3 Wilson’s primality test method

Number theory has played an important role in the public key cryptosystems [7, 21]. In this section we will review modular arithmetic in number theory. Then we will introduce the proposed improved primality test method.

3.1 Mathematical preliminaries

As what is introduced in the above section, the Wilson’s primality test method could provide an absolutely correct result, but it needs too much space and time if we calculate the “n!” directly. If our proposed algorithm is not used for Wilson’s primality test method, it will need enormous operation digits and take much time. For example, when the number n = 1,597, it can nearly produce 4,500 digits decimal number result. More tests are shown in Table 2.

In this section, we review some definitions and theorems [21] which are well-suited for our proposed primality test method.

Theorem 1:

If a, b, c, and m are integers with m > 0 so that $a \equiv b \bmod m$, then

(a) $(a+b) \bmod m \equiv [(a \bmod m)+(b \bmod m)] \bmod m$,

(b) $(a-b) \bmod m \equiv [(a \bmod m)-(b \bmod m)] \bmod m$,

(c) $(a*b) \bmod m \equiv [(a \bmod m)*(b \bmod m)] \bmod m$,

(d) $(a+c) \equiv (b+c) \bmod m$,

(e) $(a-c) \equiv (b-c) \bmod m$,

(f) $(a*c) \equiv (b*c) \bmod m$.

Theorem 2:

If $a \equiv b \bmod m_1, a \equiv b \bmod m_2, \dots, a \equiv b \bmod m_k$, where a, b, m1, m2, …, mk are integers with m1, m2, …, mk positive, then $a \equiv b \bmod [m_1, m_2, \dots, m_k]$, where [m1, m2, …, mk] is the least common multiple of m1, m2, …, mk.

Theorem 3:

A geometric series $\Sigma_k a_k$ is a series for which the ratio of each two consecutive terms $\frac{a_{k+1}}{a_k}$ is a constant function of the summation index k. For the case of the ratio $\frac{a_{k+1}}{a_k} = r$ equals to a constant r, the terms a_k are the form $a_k = ar^k$. Sn is a summation of $a + ar + ar^2 + ar^3 + \dots + ar^n$.

$$S_n = \sum_{k=0}^{n} ar^k = a + ar + ar^2 + ar^3 + \dots + ar^n \quad (1)$$

Multiplying both sides by r gives

$$rS_n = r\sum_{k=0}^{n} ar^k = ar + ar^2 + ar^3 + \dots + ar^n + ar^{n+1} \quad (2)$$

and subtracting (2) from (1) gives $(1-r)S_n = a - ar^{n+1}$,

so

$$S_n = \frac{a(1-r^{n+1})}{1-r} \quad (3)$$

Theorem 4:

If $(a*c) \equiv (b*d) \bmod m, c \equiv d \bmod m$, and (c, m) = 1, then $a \equiv b \bmod m$ where (c, m) represents the greatest common divisor. (c, m) = 1 means they don’t have any factors between c and m except 1.

Definition:

A complete system of residues modulo *m* is a set of integers so that every integer is congruence modulo *m* to exactly one integer of the set.

Table 2. Comparison decimal digits.

Test number	The decimal digits of test number	Decimal digits in original Wilson's primality test method (*n*-1)!	Decimal digits in the proposed Wilson's primality test method
97	2	150	4
127	3	212	5
251	3	493	5
367	3	781	6
499	3	1,129	6
541	3	1,243	6
677	3	1,622	6
727	3	1,764	6
877	3	2,200	6
977	3	2,496	6
1,009	4	2,592	7
1,103	4	2,876	7
1,213	4	3,213	7
1,301	4	3,486	7
1,423	4	3,868	7
1,597	4	4,421	7

3.2 Improved Wilson’s primality test method

The Wilson’s primality test method is described as:

$$(n-1)! \equiv -1 \bmod n, \quad (4)$$

where *n* represents a prime number.

It can be written as:

$$[(n-1)*(n-2)*(n-3)*(n-4)*(n-5)*(n-6)*(n-7)*\dots*5*4*3*2*1] \equiv -1 \bmod n \quad (5)$$

Equations (5) can be rewritten in details as following:

$$[(n-1)(n-2)*\dots*(\frac{n-1}{2}+2)*(\frac{n-1}{2}+1)*(\frac{n-1}{2})*(\frac{n-1}{2}-1)*\dots*5*4*3*2*1] \equiv -1 \bmod n \quad (6)$$

The following equations are based on [4] following:

$X_m \equiv b \bmod r, X_m = \prod X_i X_{i+1}$, i = 1, 3, …, m-1 where m, i, and r represent each integer. (7)

$[\prod (X_i X_{i+1} \bmod r)] \bmod r \equiv b \bmod r$, i = 1, 3, .., m-1 where m, i, and r represent each integer. (8)

If $a \equiv b \bmod m$, then $(a-m) \equiv b \bmod m$ (9)

Based on Equation (9), we subtract n from item (n-1) to item $\frac{n-1}{2}+1$, and other items remain the same in Equation (6) to obtain Equation (10) as follows.

$$\{[(n-1)-n][(n-2)-n]*...*[(\frac{n-1}{2}+2)-n]*[(\frac{n-1}{2}+1)-n]*(\frac{n-1}{2})*...*2*1\} \equiv -1 \bmod n \quad (10)$$

The result of Equation (10) can be rewritten as:

$$[(-1)*(-2)*(-3)*...*(\frac{3-n}{2})*(\frac{1-n}{2})*(\frac{n-1}{2})*(\frac{n-3}{2})*...*5*4*3*2*1] \equiv -1 \bmod n \quad (11)$$

Because *n* is an odd number, we can skip the minus symbol, and Equation (11) can be transformed to be:

$$[1*2*3*4*5...*(\frac{n-3}{2})*(\frac{n-1}{2})]^2 \equiv -1 \bmod n \quad (12)$$

In other words, we can recompose the original Wilson's primality test method [Equation (4)] as Equation (12).

Based on Equation (8), Equation (12) can be recomposed as follows.

$$\{[(1*2) \bmod n][(3*4) \bmod n]*...*[(\frac{n-3}{2}*\frac{n-1}{2}) \bmod n]\}^2 \bmod n \equiv -1 \bmod n \quad (13)$$

Based on Equation (13), we can now process each item in our proposed Wilson's primality test method, which involves multiplications and modulus inside each square bracket entry. Then we can square them in the last step. At last we use this modulus n to get the final result.

The proposed method is depicted as follows. These items inside the square bracket entries in Equation (12) can be represented in the following form:

$\prod A_{k'}A_{k'+1} = A_1 * A_2 * A_3 * A_4 * A_5 * A_6 * A_7 *...* A_{\frac{n-1}{2}-1} * A_{\frac{n-1}{2}}$; k' = 1, 3, 5, 7, 9, ……, $\frac{n-1}{2}-1$.

Hence, Equation (13) can also be represented by using a different form as follows.

$$(\prod A_{k'}A_{k'+1}) \bmod n \equiv [(A_1 * A_2) \bmod n]*[(A_3 * A_4) \bmod n]*...*[(A_{\frac{n-1}{2}-1} * A_{\frac{n-1}{2}}) \bmod n]$$

,for k' = 1,3,5,7,9, ……, $\frac{n-1}{2}-1$ (14)

In this modification, based on the fundamental modular arithmetic, we need some variables during the following computational procedures to solve Equation (12).

a = 1, 3, 5, 7, 9, ……, ($\left\lceil \frac{n+1}{4} \right\rceil$-1),

b = 1, 3, 5, 7, 9, ……, ($\left\lceil \frac{n+1}{8} \right\rceil$-1),

$\vdots$

k' = 1, 3, 5, 7, 9, ……, ($\left\lceil \frac{n+1}{2^{k+1}} \right\rceil$-1),

k =1, 2, 3, 4, 5, ……, $\left\lceil \log_2 \frac{n-1}{2} \right\rceil$.

Here we use the following procedures to evaluate Equation (12).

The first procedure:

$$A_{1a} \equiv \prod A_{k'}A_{k'+1} \bmod n$$
$$\equiv [(A_1 * A_2) \bmod n]*[A_3 * A_4) \bmod n]*...*[(A_{\frac{n-1}{2}-1} * A_{\frac{n-1}{2}}) \bmod n] \quad (15)$$

The second procedure:

$$A_{2b} \equiv \prod A_{1a}A_{1(a+1)} \bmod n$$
$$\equiv [(A_{11} * A_{12}) \bmod n]*[A_{13} * A_{14}) \bmod n]*...*[(A_{1(\left\lceil \frac{n+1}{4} \right\rceil-1)} * A_{1(\left\lceil \frac{n+1}{4} \right\rceil)}) \bmod n] \quad (16)$$

$\vdots$

The kth procedure:

$$A_{kk'} \equiv \prod A_{(k-1)k'}A_{(k-1)(k'+1)} \bmod n$$
$$\equiv [(A_{(k-1)1} * A_{(k-1)2}) \bmod n]*[A_{(k-1)3} * A_{(k-1)4}) \bmod n]*...*[(A_{(k-1)(\left\lceil \frac{n+1}{2^k} \right\rceil-1)} * A_{(k-1)(\left\lceil \frac{n+1}{2^k} \right\rceil)}) \bmod n] \quad (17)$$

The final procedure:

We assure $B \equiv (A_{kk'})^2 \bmod n$ (18)

If B = -1, n is a prime number.

If B ≠ -1, n is a composite number.

Complexity analyses

Now we generalize the above procedures from Equation (15) to Equation (18), and analyze the complexity of the proposed algorithm in detail as follows.

The first procedure:

($\left\lceil \frac{n+1}{4} \right\rceil$ -1) modular multiplications are needed to evaluate Equation (15).

The second procedure:

($\left\lceil \frac{n+1}{8} \right\rceil$ -1) modular multiplications are needed to evaluate Equation (16).

$\vdots$

The kth procedure:

($\left\lceil \frac{n+1}{2^{k+1}} \right\rceil$-1) modular multiplications are needed to evaluate Equation (17).

In the above procedures, we proceed k' numbers for each procedure, where

k' = 1, 3, 5, 7, 9, ……, ($\left\lceil \frac{n+1}{2^{k+1}} \right\rceil$-1),

and we need to execute k procedures, where

k =1, 2, 3, 4, 5, ……, $\left\lceil \log_2 \frac{n-1}{2} \right\rceil$.

The final procedure:

One modular square is needed to evaluate Equation (18).

To simplify the discussions in this paper, the modular operation is ignored and only the multiplication and the square are referred to [20, 23]. So we can sum up the computational amounts (the number of modular multiplication and modular square) in all of the above procedures below.

$$(\left\lceil \frac{n+1}{4} \right\rceil-1)+(\left\lceil \frac{n+1}{8} \right\rceil-1)+(\left\lceil \frac{n+1}{16} \right\rceil-1)+(\left\lceil \frac{n+1}{32} \right\rceil-1)+(\left\lceil \frac{n+1}{64} \right\rceil-1)+...+(\left\lceil \frac{n+1}{2^{k+1}} \right\rceil-1)+1 \quad (19)$$

where k = 1, 2, 3, …,$\left\lceil \log_2 \frac{n-1}{2} \right\rceil$

We rearrange the above equation as follows.

$$[(\frac{n+1}{4}+1-1)+(\frac{n+1}{8}+1-1)+(\frac{n+1}{16}+1-1)+(\frac{n+1}{32}+1-1)+...+(\frac{n+1}{2^{k+1}}+1-1)]+1 \quad (20)$$

where "+1" inside each parenthesis means we get the maximum for each item, which marks ceiling symbol in Equation (19).

Based on Theorem 3, we calculate Equation (20) to obtain the final result as follows.

$\frac{1}{2}*(n+1)*[1-(\frac{1}{2})^k]+1$, where $k = \left\lceil \log_2 \frac{n-1}{2} \right\rceil$. (21)

The original Wilson's primality test method is (*n*-1)! ≡ -1 mod *n*. From Table 2, we know the larger the test number is, the larger the decimal-digit size is. However, by using the proposed algorithm, the maximum decimal-digit size is generated by the (n-1)*(n-2) item. Note, this item should be bounded to 2q if we assume that the test number "n" has q decimal-digit size. Some experimental results are shown in Table 2.

4 Example

Let us take n = 29 to depict our proposed Wilson's primality test method and show the correctness of the proposed method. The Wilson's primality test method is based on Equation (4):

$(n-1)! \equiv -1 \bmod n$, where n represents a prime number. Based on Equation (5) to Equation (6), and Equation (10) to Equation (12), the original Wilson's primality test method can be changed as follows.

$$(1*2*3*4*5*6*...*11*12*13*14)^2 \bmod 29 \equiv -1 \bmod 29 \quad (22)$$

Based on Equation (8) and Equation (22), our proposed method executes basically the following steps:

The first step,

$\{[(1*2) \bmod 29]*[(3*4) \bmod 29]*...*[(11*12) \bmod 29]*[(13*14) \bmod 29]\}^2 \bmod 29 \equiv -1 \bmod 29$

$\Rightarrow \{[2*12*1*27*3*16*8] \bmod 29\}^2 \bmod 29 \equiv -1 \bmod 29$.

The second step,

$\{[(2*12) \bmod 29]*[(1*27) \bmod 29]*[(3*16) \bmod 29]*[(8) \bmod 29]\}^2 \bmod 29 \equiv -1 \bmod 29$

$\Rightarrow \{[24*27*19*8] \bmod 29\}^2 \bmod 29 \equiv -1 \bmod 29$.

The third step,

$\{[(24*27) \bmod 29]*[(19*8) \bmod 29]\}^2 \bmod 29 \equiv -1 \bmod 29$

$\Rightarrow \{[10*7] \bmod 29\}^2 \bmod 29 \equiv -1 \bmod 29$.

The fourth step,

$\{[10*7] \bmod 29\}^2 \bmod 29 \equiv -1 \bmod 29$

$\Rightarrow \{12 \bmod 29\}^2 \bmod 29 \equiv -1 \bmod 29$.

The fifth step,

$\{12\}^2 \bmod 29 \equiv -1 \bmod 29$

$\Rightarrow \{144 \bmod 29\} \equiv -1 \bmod 29$.

From the first step to the fifth step, the proposed Wilson's primality test method requires 7, 4, 2, 1 modular multiplication and one modular square, respectively. To sum up, the whole evaluation of the proposed Wilson's primality test method requires 14 modular multiplications and one modular square.

5 Conclusions and future works

In this paper, we apply modular arithmetic to improve the original Wilson's primality test method for reducing the computational complexity and getting a better area usage. Compared these criterions depicted in [3] [4] with the proposed algorithm in this paper, we can clearly understand that the test number "n" becomes larger and the other two methods will require much space and time as shown in Figure 1 and Table 3. They become infeasible. By using our proposed algorithm, even though n grows larger, the space and time we require can be still under control and save much more.

In the future, we will try to further effectively improve the Wilson's primality test method by transforming integer n from decimal number system into binary system [21, 27, 28] and reduce the redundant computational complexity [29-31]. Secondly, starting from many studies emphasized in this field [32-33], we will further study and search for more efficient methods and useful mathematical theorem to speed up the Wilson's primality test method. To sum up, we can therefore perform this deterministic primality test method more effectively when applying it in modern cryptosystem.

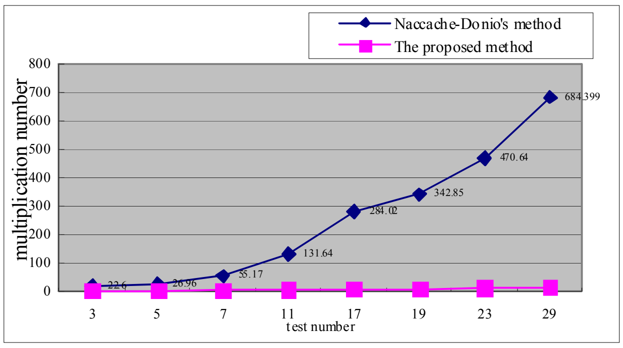

Figure 1. Complexity comparisons between Naccache-Donio's method and the proposed method.

Table 3. Complexity comparisons using smaller test number "n".

n	Naccache-Donio's method	Rosen's method	The proposed method
3	2	22.6	2
5	4	26.96	2.5
7	12	55.17	3
11	240	131.64	6.25
17	80,640	284.02	8.875
19	725,760	342.85	10.375
23	79,833,600	470.64	12.25
29	174,356,582,400	684.399	15.0625

Naccache-Donio's method: $2*[(\frac{n-1}{2})!]$.

Rosen's method: $[n(\log_2 n)^2]$.

The proposed method: $\frac{n+1}{2}*[1-(\frac{1}{2})^k]+1$,where $k = \left\lceil \log_2 \frac{n-1}{2} \right\rceil$.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, Oct. 1978.

[2] http://www.meganet.com/primality.htm.

[3] D. Naccache and M. Donio, "Accelerating Wilson's Primality Test," *Revue Technique Thomson-CSF*, vol. 23, no. 3, pp. 595-599, 1991. http://library.wolfram.com/search/.

[4] K.-H. Rosen, *Elementary Number Theory and Its Applications*, 3rd Ed., Addison-Wesley, 1988.

[5] M.-R. Schroeder, *Number Theory Science and Communication with Applications in Cryptography*, Berlin, N. Y. Springer-Verlag, 1986.

[6] R. Kumanduri and C. Romero, *Number Theory with Computer Applications*, Upper Saddle River, N. J. Prentice Hall, 1998.

[7] W. Stallings, *Cryptography and Network Security Principles and Practice*, 3rd Ed., Prentice-Hall, 2003.

[8] I. Koren, A.-K. Peters, and M.-A. Natick, *Computer Arithmetic Algorithms*, 2nd Ed., 2002.

[9] S.-S. Wagstaff, *Cryptanalysis of Number Theoretic Ciphers*, CRC Press Chapman & Hall, 2003.

[10] M. Agrawal, "On derandomizing tests for certain polynomial identities," *Proceedings of 18th IEEE Annual Conference on Computational Complexity*, 2003, vol. 7-10, pp. 355-359.

[11] E.-W. Weisstein, "Primality testing is easy," *MathWorld Headline News*, Aug. 7, 2002. http://mathworld.wolfram.com/news/2002-08-07/primetest/.

[12] M. Agrawal, N. Kayal, and N. Saxena, "Primes in *P*," *Preprint*, Aug. 6, 2002.

[13] D.-J. Bernstein, "An exposition of the Agrawal-Kayal-Saxena primality-proving theorem," http://cr.yp.to/papers/aks.ps, 2002.

[14] D. Mukhopadhyay and D. Roy Chowdhury, "An efficient end to end design of Rijndael cryptosystem in 0.18 μ CMOS", *Proceedings of the 18th International Conference on VLSI Design*, pp. 405-410, Jan. 2005.

[15] J. Linn, "Technology and web user data privacy: a survey of risks and countermeasures," *IEEE Security & Privacy Magazine,* vol. 3, no. 1, pp. 52-58, Jan.-Feb. 2005.

[16] R. Silverman, "Massively distributed computing and factoring large integers," *Communications of the ACM*, vol. 34, no. 11, pp. 95-103, 1991.

[17] M. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12, pp. 128-138, 1980.

[18] http://mathworld.wolfram.com/

[19] N. Demytko, "Generating multi-precision integers with guaranteed primality," *IFIP*, Elsevier Science publishers, North-Holland, 1989.

[20] http://scienceworld.wolfram.com/

[21] D.-E. Knuth, *The Art of Computer Programming*, vol. 2: Seminumerical Algorithms, 3rd Ed., Addison-Wesley, 1998.

[22] D.-C. Lou, C.-L. Wu, and R.-Y. Ou, "Application of parallel virtual machine framework to the strong prime problem," *International Journal of Computer Mathematics*, vol. 79, no. 7, pp. 797-806, June 2002.

[23] D.-C. Lou and C.-C. Chang, "Fast exponentiation method obtained by folding the exponent in half," *IEE Electronics Letters*, vol. 32, no. 11, pp. 984-985, May 1996.

[24] C.-W. Chou, "Parallel implement of the RSA public-key cryptosystem," *International Journal Computer Mathematics*, vol. 78, no.5, pp. 153-155, 1993.

[25] M. Joye and S.-M. Yen, "Optimal left-to-right binary signed-digit recoding," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 740-748, July 2000.

[26] A. Arora and R. Telang, "Economics of software vulnerability disclosure", *IEEE Security & Privacy Magazine,* vol. 3, no. 1, pp. 20-25, Jan.-Feb. 2005.

[27] X. Ruan and R.-S. Katti, "Left-to-right optimal signed-binary representation of a pair of integers," *IEEE Transactions on Computers,* vol. 54, no. 2, pp. 124-131, Feb. 2005.

[28] M.-E. Kaihara and N. Takagi, "A hardware algorithm for modular multiplication/division," *IEEE Transactions on Computers,* vol. 54, no. 1, pp. 12 – 21, Jan. 2005.

[29] D.-C. Lou and C.-L. Wu, "Parallel exponentiation using common multiplicand multiplication and signed-digit-folding techniques," *International Journal of Computer Mathematics* , vol. 81 , no. 10 pp. 1187-1202, June 2004.

[30] C.-L. Wu, D.-C. Lou, and T.-J. Chang, "An efficient Montgomery exponentiation algorithm for cryptographic application," *Informatica – An International Journal*, vol. 16, no. 3, pp. 449-468, Sept. 2005.

[31] C.-L. Wu, D.-C. Lou, T.-J. Chang, and S.-Y. Chen, "Integer factorization for RSA cryptosystem under a PVM environment," *International Journal of Computer Systems Science & Engineering*, vol. 1, no. 2, pp. 25-35, Jan. 2007.

[32] C.-L. Wu, D.-C. Lou, and T.-J. Chang, "Fast parallel exponentiation algorithm for RSA public-key cryptosystem," *Informatica–An International Journal*, vol. 17, no. 3, pp. 445-462, Sept. 2006.

[33] D.-C. Lou,, J.-C. Lai, C.-L. Wu, and T.-J. Chang, "An efficient Montgomery exponentiation algorithm by using signed-digit-recoding and folding techniques," *Applied Mathematics and Computation*, vol. 185, no. 1, pp. 31-44, Feb. 2007.