Journal of Theoretical and Applied Information Technology
15th December 2012. Vol. 46 No.1
© 2005 - 2012 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN OPTIMIZATION ALGORITHM OF RSA KEY GENERATION IN EMBEDDED SYSTEM

[1]LI DONGJIANG, [2]WANG YANDAN
[1]Department of Computer Science, North China Electric Power University, Beijing, China
[2]Department of Computer Science, North China Electric Power University, Beijing, China
Email: [1]lidongjiang100@126.com , [2]wyd418@126.com

ABSTRACT

To reduce the time complexity, this paper proposes an optimization algorithm to generate the large prime number. Before the primality test, an improved prescreening algorithm is used to get rid of most of odd composite numbers. Then we introduce the Fermat's little theorem to make a further judgement, which can decrease the times of using prescreening algorithm. At last, Miller-Rabin algorithm is used to make a final primality test. The results show that the speed of key generation has been greatly improved.

Keywords: *RSA Key Generation; Embedded System; Fermat's Little Theorem; Miller-Rabin Algorithm*

1. INTRODUCTION

RSA is an asymmetric encryption algorithm and also the first and most successful public key cryptosystem in theory. It is widely used in PKCS (Public Key Cryptography Standards) and Electronic Business. In this algorithm, there is always the computation of large numbers, so great time complexity is the greatest imperfection in either hardware or software implementation. Therefore, an efficient implementation of RSA key generation is very important for theoretical study and practical applications [2].

In the process of RSA key generation, the most time-consuming process is the generation of large prime number. So our main study is focused on the optimization. Generally, in order to get rid of partial odd composite numbers, prescreening is adopted before the final Miller-Rabin algorithm. Up to now, much research work has been taken to reduce key generation time.

In this paper, we propose a much faster prescreening algorithm based on the principle of modular arithmetic. Furthermore, to reduce the calling times of prescreening algorithm, the Fermat's little theorem is introduced between prescreening algorithm and Miller-Rabin algorithm. With those efforts, the generating time of key has been considerably reduced. Under the system clock frequency of 60MHz, a pair of 1024 bits RSA keys needs 2.129s and 2048 bits 6.896s.

Firstly, the paper proposes the principle of RSA key generation. Secondly, we mention the time consumption regularity of RSA key generation and probabilistic prime test and true prime test. Then we put emphasis up on the generation of large prime number. At last, the paper gives the results of optimization algorithm.

2. PRINCIPLE OF RSA KEY GENERATION

In RSA algorithm, there are two kinds of key, public key and private key. They are separately used for RSA encryption and decryption operations. Basic steps are as follows:

(1)Generate two large random positive integers p and q. p is unequal to q. Then compute N, which is equal to the value p multiplied by q;

(2)Select an integer e which is less than ϕ where $\phi = (p-1)*(q-1)$. Here ϕ and e should be relatively prime numbers;
(3)Use Expanded-Euclidean algorithm to calculate d. The formula is as follows:

(1) $d * e \equiv 1 \bmod \phi$

After the above process, (N, e) is public key and (N, d) is private key.

In embedded system, the number e is suggested to select 3, 5, 17, 257, or 65537 by IEEE [1]. In the practical applications, we always select a key length of 1024 or 2048 bits for security reasons. Usually p has the same bit width with q[2]. For example, if we want to get a key of 2048 bits, p and q are supposed to be 1024 bits.

3. TIME CONSUMPTION REGULARITY OF RSA KEY GENERATION

Before the optimization of key generation, we conducted runtime statistics of some critical blocks of key generation as follows.

Table 1: Average Time Consumption Of Critical Blocks In Key Generation. Unit: S Annotations: The Content In Parentheses Represents The Percentage Of Time.

	large prime number	private key	public key	others
1024 bits	5.599（86.0%）	0.845(13.0%)	0.008 (0.1%)	0.061（0.9%）
2048 bits	17.980（86.4%）	0.961(4.6%)	0.009 (0.04%）	1.849 (8.96%）

We can obviously see from Table 1 that the time of large prime number generation occupies the vast majority time of the key generation. Therefore, this paper focuses on the optimization of prime number generation.

4. PROBABILISTIC PRIME TEST AND TRUE PRIME TEST

When generating a prime number, prime test is a very important process. There are two kinds of prime test algorithm. One is probabilistic prime test, the other is true prime test. Probabilistic prime test is relatively fast and simple, but it has erroneous judgments to some extent which means the output number is just a possible prime number. On the contrary, the true prime test has no erroneous judgments, but it's not useful in practical because of its time-consuming calculations. Moreover, the probability of erroneous judgments of probabilistic prime test can be controlled at a very low and acceptable scale. Therefore, probabilistic prime test is used in most practical applications.

5. THE GENERATION OF LARGE PRIME NUMBER

In embedded system, there are three steps for the generation of large prime number.

(1)Use random number generator to generate a large odd number;

(2)Prescreening;

(3)Primality testing.

5.1 random number generation

The large random number is generated by random number generator which is integrated in single-chip. Table 2 shows the average time consumption of random number's generation in large prime number generation.

Table 2: Average Time Consumption Of Random Number's Generation In Large Prime Number Generation. Unit: S

Annotations: The Content In Parentheses Represents The Percentage Of Time.

	Random number generation	Others
1024 bits	0.272（9.7%）	2.527 (90.3%）
2048 bits	0.544（6.1%）	8.446 (93.9%)

From Table 2, we can see that generating a random number of 1024 bits takes up nearly 10% of the whole prime number generating time, while generating one of 2048 bits takes up 6%. It is time-consuming to use the generator anytime if we want a large number. Therefore, we use the following method to generate searching sequence [4].

(1)Use generator to generate the first number of the sequence. This number is named n which is odd.

(2)Increase 2 to n every time to form the sequence of n, n+2, n+4, n+6 …

If any number of the sequence passes the primality test, then drop the whole sequence to ensure the randomness of the data.

5.2 rapid prescreening algorithm

In order to improve the speed of prime number generation, we designed a rapid prescreening algorithm to get rid of the majority of odd composite numbers. Only those who pass the prescreening algorithm can enter the final test [9].

The probability that a large integer n has a small prime divisor is high, so the main idea of prescreening is to construct a table with some small primes. In the algorithm of this paper, the small primes table we construct includes all prime numbers less than 256. These numbers are named continuously as t_1, t_2, t_3 ….Then use prime numbers in the table to screen the candidate number

Journal of Theoretical and Applied Information Technology
15th December 2012. Vol. 46 No.1
© 2005 - 2012 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

p. If the remainder is 0, we can judge that p is a composite number.

Because the number in the sequence mentioned in 5.1 increases 2 every time, we design our improved algorithm, which is based on the following principle of modular arithmetic.

(2) (a+b) mod n= ((a mod n) + (b mod n)) mod n

Algorithm 1:

(1)Defining that R[i] =p mod ti (t_1=3, t_2=5, t_3=7,…);

(2)Use the formula (2) to make the following conversion. If any R[i] is equal to 0, p is a composite number. Then update R[i] = R[i]+2. If R[i] ≥ti, then R[i]= R[i]- ti and p= p+2;

(3)If all elements of R[i] is unequal to 0, then p passes the prescreening algorithm.

The results of prescreening for the improved algorithm above are saved in the array R. With this effort, all division procedures are very simple single precision division except the first time. So it largely reduces complexity and largely shortens the computing time [6].

5.3 Primality Testing

5.3.1 Fermat's little theorem

After the prescreening, the possibility, that the large number is a prime, is still not high. A further primality test is usually carried on. Before entering the final Miller-Rabin algorithm, we have designed the test with Fermat's little theorem [8].

In Fermat's little theorem, supposing that p is a prime number, *a* and *p* are coprime numbers. Then we can get the conclusion that $a^{p-1} \equiv 1 \pmod{p}$ .Table 4 shows the contrast key generation time before and after using Fermat's little theorem.

Table 3 The Contrast Key Generation Time Before And After Using Fermat's Little Theorem（The System Clock Is 60mhz）

Bits	1024	2048
Before(s)	2.780	9.620
After(s)	2.129	6.896

From the table above we can see that key generation time of either 1024 bits or 2048 bits has been largely shortened after we use Fermat's little theorem. This is because that it reduces the calling times of prescreening after the superposition of prescreening and Fermat's little theorem. For example, before using Fermat's little theorem, the average calling times is 230, and it is 152 after.

5.3.2 miller-rabin algorithm

At the end of the whole prime number's generation, Miller-Rabin algorithm is used to make a final test.

Miller-Rabin algorithm is the rapidest method for detecting primes by far. It is also the algorithm which is recommended to use in Digital Signature Standard (DSS) by National Institute of Standards and Technology (NIST).

This algorithm is just a kind of probabilistic primality test, so there exist erroneous judgments. But based on the probability theory, it can be accepted that if the probability of erroneous judgement is less than $(1/2)^{80}$. A security parameter t which represents the times of algorithm execution [7] is introduced to control the erroneous judgments. And in embedded applications, it can increase the computing time if the value of t is too large. So after comprehensive consideration, table 5 gives us the appropriate valve of security parameter t when meeting the condition of $p< (1/2)^{80}$ in practical applications [5].

Table 4 Security Parameter's Selection Of Miller-Rabin Algorithm

Bits	512	1024	2048
Security parameter t	6	3	2

6. TESTING RESULTS OF KEY GENERATION

After the optimization of RSA key generation above, we get the testing results as is shown in Table 5. Under the system clock of 60MHz, it takes 2.129s on average to generate a pair of keys with 1024 bits and 6.896s with 2048 bits. Compared with previous results, key generation time is largely shortened after using the optimization method designed in this paper.

Table 5 Average Time-Consuming Of Key Generating Before And After Optimization

	Before(s)	After(s)
1024 bits	6.513	2.129
2048 bits	20.799	6.896

7. CONCLUSION

In this paper, we propose a remarkable optimization algorithm of RSA key generation in detail. Table 6 shows that it takes up only one thirds of the previous key generation time.

In our algorithm, multiple precision divisions have been transformed into single precision divisions. And the improved small prime table includes all primes numbers whose value is less than 256. With these efforts, the speed of key generating has been increased 38.2%. What's more, with the usage of Fermat's little theory, the speed of key generation has been increased 66.8% finally.

RSA algorithm is regarded as one of the most excellent public key crypto schemes. Though it has many advantages, speed is always the largest drawback. The algorithm proposed in this paper has brought us a remarkable optimization result, but we need to optimize continually so as to adapt the increasing market demand.

REFERENCES

[1] IEEE Std 1363-2000, IEEE standard specifications for public-key cryptography, 2000.

[2] Alfred J Menezes, Paul C Oorschot, Scott A Vanstone, Handbook of Applied Cryptography, 2005.

[3] Knuth D E, Art of Computer Programming(Vol.2). the 3rd Edition. Su Yunlin, translation, National Defence Industry Press, Beijing, 2002.

[4]Cui Jingsong, Tu Hang, Peng Rong,etc.The quick generation of large prime number in embedded system.Computer engineering,2003,299(5):24-58.

[5]Qin Xiaodong, Xin Yunwei, Lu Guizhang. Research and optimization realization of Miller-Rabin algorithm.Computer engineering, 2002,28(10):55-57.

[6] Li Jialu, Zhou Yujie, Efficient implementation of RSA key generation in embedded system. Computer Engineering and Design, 2009.

[7]Yao Guoxiang, Lin Liangchao. Efficient Method of RSA Key-Pair Generation[A]. Computer Engineering, 2007.

[8] Bahadori, M.; Mali, M.R.; Sarbishei, O.; Atarodi, M.; Sharifkhani, M. A novel approach for secure and fast generation of RSA public and private keys on Smart Card. NEWCAS Conference (NEWCAS), 2010.

[9] Jingwei Hu; Wei Guo; Jizeng Wei; Yisong Chang; Dazhi Sun. A Novel Architecture for Fast RSA Key GenerationBased on RNS. Parallel Architectures, Algorithms and Programming (PAAP)