Analysis Implementation of Fast RSA Key Generation on Smartcards

Minte Chen
Electrical and Computer Engineering
Oregon State University
Corvallis, Oregon 97331–4501
Email: chenmin@ece.orst.edu

Abstract— **Smartcard, or smart card, is a plastic card with an embedded microcomputer chip. The dimension of a smartcard is the size of an ordinary credits card, which can be easily carried around in a wallet. Smartcards are much more difficult to duplicate than magnetic strip cards and cryptographic functions can be implemented inside these cards. With substantial cost reductions and ability to handle multiple applications on a single card, the smartcards are about to enter a period of the rapid growth. An individual bearing a single smartcard will be able to electrically and security interact with several servers or service provides. As a consequence, an entirely new type of commercial and educational landscape is being created. However, smartcard itself has limited computing power and memory capacity. This makes it a difficult job to efficiently implement the cryptographic functions inside smartcard. RSA is one of the most popular public key cryptographic algorithms. In this paper , we are going to analysis the fast RSA key implementation on smartcards using different algorithms.**

I. Introduction

The smartcard, an intelligent token, is a credit card sized plastic card embedded with an integrated circuit chip. It provides not only memory capacity, but computational capability as well.It has been said that smartcards will one day as important as computers are today[1],[2].Smartcards have proven to be quite useful as a transaction/authorization/identification medium in some countries.

As their capability grow, they could become the ultimate client, eventually replacing all of the things we carry around in our wallets, including credit cards, licenses and cash. By containing various identification certificates, smartcards could be used to identify attributes of ourselves no matter where we are. Smartcards are often used in different applications which require strong security protection and authentication. For example, smartcards can act as identification card which is used to prove the identity of the card holder. It also can be a medical card which stores the medical history of a person. Furthermore, the smartcard can be used as a credit/debit bank card which allows off-line transaction. All of these applications require sensitive data to be stored in the card, such as personal medical history and cryptographic key for authentication, etc.

The ability to compute and interact in a system gives the smart card access to powerful cryptographic algorithms and improves the flexibility, security and reliability of the system. The possibility of applications that required high degree of security.

As cryptography made great progress in the 1960s and security mechanism could be proved mathematically , smartcards proved to be an ideal medium for safely storing cryptographic keys and algorithms.In recent years, public key cryptography has gained increasing from both companies and end uses who wish to use this technology for secure a wide variety of applications. One consequence of this trend has been the growing importance of public key smart to store the user's private keys and provide a secure computing environment for private key operations. As a result, smartcards can be seen as a kind of security tokens.

Main industrial efforts concentrated on the performance of these algorithms with key sizes that were lying around the 1024-bit range. The new range of key sizes for RSA is now generally up to 2048-bit computations. However, the computational power of smart cards is very limited and the on-card implementations are usually much slower than that in desktops. Many chip manufactures are therefore proposing ever better and faster implementations of public key algorithms using their special hardware, called crypto-coprocessor, on their chip ,which can accelerate the crypto computations for a class of public key cryptographic algorithms. The crypto-coprocessor is a specialized circuitry that perform fast modular exponentiation. Like RSA uses modular exponentiation for encryption and decryption of data for both operations.We categorized RSA key generation algorithms into two types, generating the public and private key pairs in a single computer and generating a shared RSA key by multiple computers. Although crypto-coprocessor accelerate the RSA key generation for the modular exponentiation operation. However, it cannot let smartcards achieve the desired efficiency for on-card key generation.

II. KEY GENERATION

The RSA was proposed by Rivest, Shamir, and Adleman in 1977[3].RSA use a key for encryption that is different from the decryption key. The procedure for generating a key pair is as below:

1.Choose secret primes p and q and computes $n = pq$

2.Choose e with $gcd(e, (p-1)(q-1)) = 1$

3.Compute d with $de = 1(mod(p-1)(q-1))$

4.The pair (e, n) is published as the public key, and the pair (p, q, d) is kept secret as private key

III. LARGE PRIME FINDING ALGORITHMS

The total time for generating an RSA key pair is almost totally due to the time of finding two large primes. Therefore, it is important to optimize the algorithm used for finding large prime for generating the RSA key pairs.

A. Prime Distribution

The number of primes less than a natural number N is asymptotically equal to $N/logN$[6].We randomly choose x,the probability of x being a prime number is approximately $(N/logN)/N$.If x is an n-bit number,then $logx = n * log2 = 0.69n$. Table1 shows the number of n-bit numbers containing one prime on the average.

Table 1. Number of randomly generated numbers needed to obtain one prime on the average.

bit number	256	512	1024	2048
average number necessary	176	355	710	1420

B. Prime Test

Generating a random number, the generating number must be tested for primality in order to be useful for generation of a RSA key pair. Primality tests can be divided into two categories: primality test and probabilistic primality test[4].Primality test seems the appropriate technique when finding a prime number. However, they are more complex and computing power intensive than probabilistic primality test. In this paper ,we are going to use probabilistic primality test

C. Generating Primes Algorithm

We use naive algorithm to find an random n-bit odd prime number, which T use the odd number as input[5]. In this algorithm ,the probabilistic primality test returns that a number is not a prime number, another random odd number is chosen and the same procedure is repeated until a prime number is found. As Algorithm A is shown the naive algorithm.

Algorithm A: Naive prime algorithm

1: pick a random n-bit odd number q
2: if $T(q)$= false then goto 1
3: output q and halt

An average of 176 calls to the probabilistic primality test function T is required to find a 512-bit prime using the naive prime algorithm.An average of 355 calls to T is required to find a 1024-bit prime.
We can reduce the number of calls to the probabilistic primality test function T by using a variation of sieve of Eratosthense called before calling T.As shown in Algorithm B. The implementaiton of the primality test function T must be optimized and the number of calls of T must be the lowest as possible.

Algorithm B: Variation of sieve Eratosthenes

1: Let p_i be the i-th smallest odd prime ($p_1 = 3$, $p_2 = 5$, ...)
2: Let $S(k)$ be a set of small primes such that $S(k)$= $[p_i|p_i \leq k]$where k can be any positive number
3: For a given number q, divide q by all the elements in $S(k)$
4: If q is not divisible by all the elements in $S(k)$,q is said to survive the sieve.Otherwise q is said to fail the sieve.i.e.,q is a composite number

The sieve function is very time efficient for $S(k)$ when k is small, requiring much less processing time than T.Therefore, it increases the overall performance of the prime finding algorithm.

We can modify Navie algorithm by using the sieve as shown Algorithm C. In the new algorithm, $q^{(i)}$ for $i = 0, 1..$, are tested until a probable prime is found.Another optimized for the algorithm of Algorithm C is that a new $q^{(i)}$ is generated by adding $2d$ to $q^{(i-1)}$.Hence, the algorithm doesn't need to generate an n-bit random number on each iteration, saving the time required for random generation.

Algorithm C: Naive algorithm modified to use sieve

1: Pick a random n-bit odd number q and let $q^{(0)} = q$,$i = 0$
2: Call sieve procedure ,if $q^{(i)}$ fails,goto 4
3: If $T(q^{(i)})$ =TRUE,output $q^{(i)}$ and halt
4: $q^{(i+1)} = q^{(i)} + 2d$,$i = i + 1$,goto 2 (d is a chosen integr)

Only a small portion of the prime candidates is able to reach step 3 on the algorithm C because of the sieve procedure.Therefore,the average numberof calls to T is reduced, shown in Table2.

Table 2. Expected average number of calls to the probabilistic primality test T.

bit number	S(29)	S(256)	S(512)	S(2560)
512-bit	54.7	35.5	31.0	25.3
1024-bit	109.3	71.0	62.0	50.1

A bigger size of the set $S(k)$ results in fewer calls to primality test T. We design $S(k)$ as big as posssible.However, the bigger size of $S(k)$ will increase storage space and cost more time to process for the sieve procedure.

One of the most used sieve methods is the division method[7],[8].As shown in Algorithm D.The prime finding algorithm using the trial division method.

Algorithm D: Prime finding algorithm using trail division

1: Choose a set $S(k)$,Pick a random n-bit odd number q and let $q^{(0)} = q$,$i = 0$
2: Let $w_j^{(i)}$=$q^{(i)}$ mod p_j,If $w_j^{(i)}$=0, for any j,

$1 \le j \le k$, goto 4
3: If $T(q^{(i)})$ =TRUE,output $q^{(i)}$ and halt
4: $q^{(i+1)} = q^{(i)} + 2d, i = i + 1$,goto 2

The modular reduction operation of the trial division algorithm is shown on step 2 of Algorithm D ($w_j^{(i)} = q^{(i)} \bmod p_j$). If $q^{(i+1)} = q^{(i)} + 2$,and $w_j^{(i)} = q^{(i)} mod \mathrm{p}, then \mathrm{w}_j^{(i+1)} = w_j^{(i)} + 2 \bmod p$.

The computation of $w_j^{(i+1)}$ uses two 8-bit operands, resulting in a performance much faster than modular reduction operations.The algorithm as shown in Algorithm E. In the algorithm of Algorithm E require to keep all the residures $w_j^{(i)}$ of the previous iteration.

Algorithm E: Prime finding algorithm using table look-up

1: Choose a set $S(k)$.Pck a random n-bit odd number q and let $q^{(0)} = q, i = 0$
2: Compute $w_j^{(0)} = q^{(0)} \bmod p_j, 1 \le j \le k$
3: If $w_j^{(i)} = 0$,for any j, $1 \le j \le k$,goto 5
4: If $T(q^{(i)})$ =TRUE,output $q^{(i)}$ and halt
5: $w^{(i+1)} = w^{(i)} + 2 \bmod p_j, 1 \le j \le k$
6: $q^{(i+1)} = q^{(i)} + 2, i = i + 1$,goto 3

The bit-array algorithm will find a probable prime if there is one in the chosen interval.If there is no probable prime on the chosen interval ,one can either random choose another odd q and let $q^{(0)} = q$.The bit algorithm is shown in Algorithm F.

Algorithm F: Prime finding algorithm using bit arrray

1: Set $a_i = 0$,for $0 \le i \le l - 1$;
2: Pick a random n-bit odd number q and let $q^{(0)} = q, i = 0$
3: for each p_j, do
   a.Compute $w_j^{(0)} = q^{(0)} 4 mod \mathrm{p}_j$
   b.Compute $g(p_i)$
4: $B = B_0$
   a.If($a_i = 0$) and ($T(q^{(i)})$=TRUE),
   output $q^{(i)}$ and halt
   b.$q^{(i+1)} = q^{(i)} + 2$
   c.$q^{(0)} = q^{(i)}, i$=o,goto 3

D. Fast Implementaton of Probabilistic Primality Tests

Modular exponentiations are the most computing expensive operations of probabilistic primality test algorithms. In this implementation ,computing $B = A^E \bmod M, E$ is represented by$E = [e_{k-1}..e_0]$ with $e_{k-1} = 1$ .As shown in Algorithm G.

Algorithm G: Algorithm to compute $B = A^E \bmod M$

1: $B_k - 1 = A$;
2: for $i = k - 1$ to -,do
   a.$P_i = B_i * B_i \bmod M$;
   b.if $e_i = 1$ then $B_{i+1} = P_i * A \bmod M$;
   c.else $B_{i+1} = P_i$;
3: $B = B_0$

A smartcard crypto coprocessor implements the modular multiplication using n-bit multiplicand and m-bit multiplication. We can achieve the better speed from the calculation of step 2.b on Algorithm G. Using 2 as the witness can result in 33randomly picked up witness.

From crypto co-processor in Infineon SLE66CX160S supports two modes of operation:long mode for up to 1120-bit long operations number,and short mode for up to 560-bit long operations number.As a result,both the module and exponent are 512-bit long and witness is 2, the excuting time is 110 ms for the short mode and 220 ms for the long mode.

E. Timing

Most cost part of an RSA key generation is the prime finding procedure.The optimization on the performance of the prime finding algorithm is the sieve algorithm.See Table 3 shown the overhead on using the sieve algorithm.

The performance shown in Table 3,using 400 example measurements as each sieve implementaion.

Table 3. Overhead of sieve procedures using different algortihm

Different algorithm	Memory Space used	512-bit (sec)	1024-bit (sec)
Trail division S(256)	53 bytes code	2.00	20.00
Table look-up S(256)	53 bytes code	0.20	0.80
Bit array S(256)	53 bytes code 128 bytes RAM	0.11	0.17
Bit array S(512)	93 bytes code 128 bytes RAM	0.15	0.26

IV. Key Pair Generation

Finding the public key and private key after finding the prime numbers is trival.A very common method for finding modular inverse is the extend Euclidean algorithm.[7]Finding the public key and private key after finding the prime numbers is trival.From the extended Eucldidean Algorithm,shown in Algorithm G, the overtime to find a 1024-bit RSA key pair is less than 11 seconds with table look-up algorithm $S(256)$

Algorithm H: Euclid's extended algorithm for computing inverses

1: Compute $m = (p - 1)(q - 1)$;
2: Let $g_0 = m, g_1 = e, u_0 = 1, v_0 = 0, u_1 = 0, v_1 = 1, i = 1$;

3: While $g_i \neq 0$
   a.$y := g_{i-1}$ div g_i;
   b.$g_{i+1} := g_{i-1} - y * g_i$;
   c.$u_{i+1} := u_{i-1} - y * u_i$;
   d.$v_{i+1} := v_{i-1} - y * v_i$;
   e.$i = i + 1$;
4: $x := v_{i-1}$;
5: If $x = o$ Then inv=x else inv=$x + m$;
6: Output inv

V. Related Work and Conclusion

We detailly descript the steps on optimizing the performance of RSA key generation in smartcards.The on-card key generation problem is that a large prime finding problem.This paper we propose a protype for fast prime finding algorithms.And we also build fast implementation for the algorithm in smartcard with crypto-coprossor using Infineon SLE66CX160S microcontroller. As this paper shown that the algorithms presented are more efficient and space saving than other algorithms.It is suitable for on-card RSA key generation.

References

[1] Chung-Hung Yang , Hikaru Morita and Tatsuaki Okamoto, "Fast Implementation of Digital signature on Smartcards without coprocessor," *Communication of the IIMA*, vol. 2, no. 2, pp. 82–90, Oct. 2002.

[2] Chan,S. C. "An overview of Smart card Security," http://www.hkstar.com/ alanchan/papers/smartCardSecurity/

[3] Rivest, R., Shamit, A. and Adleman, L., "A Method for Obtaining digital Signatures and Public Key Cryptosystems," *communications of ACM*, vol. 21, no. 2, pp.158-164 , Feb. 1978.

[4] Chenghuai Lu, Andre L.M. dos Santos and Francisco R. Pimentel, "Implementation of Fast Key Generation on Smart Cards," *Proceedings of the 2002 ACM symposium on applied computing*,Mar. 2002.

[5] Joye, M., Palliar, P. and Vandeney, S. *Efficient Generation of Prime numbers*, CHES 2000, 340-354.

[6] knuth, D.E., "The Art of Computer Programming Seminumerial Algorithms of Computer Science and Information Processing," Addison-wesley, 3rd ed., Vol.2,1997.

[7] Stallings, W., *cryptography and Network Security:principle and practice,* 2nd ed., Prentice-Hall , New Jersey, 1999.

[8] Bressoud, D. M., *Factorizations and Primality Testing,* Springer-Verlag, New York, 1989.