Efficient on-Board J_2-RSA Key Generation with Smart Cards

L. Sreenivasulu Reddy

Abstract*— Public key cryptography gained increasing attention from both companies and the end users who wish to use this emerging technology to secularize a wide variety of applications. A major consequence of this trend has been the growing significance of the public-key smart cards. A smart card is a tiny secure crypto processor embedded within a credit card-size or smaller(like the GSM SIM) card which provide encryption, decryption as well as key generation within it's security perimeter. J_2-RSA is a simple and easy to implement public key cryptographic algorithm. Today J_2-RSA key keys range from 512 bits to 2048 bits and some bodies envision 4096-bit J_2-RSA keys in near future, like RSA key. In this paper, I will present a study of efficient algorithms involved in on-board J_2-RSA key generation[1].

Key Words : J_2-RSA , Jordan arithmetic function, Prime Number and Co-primes .

I. INTRODUCTION

It has been said that smart cards will one day be as important as computers are today**.** This statement contains a bit of an error because it implies that smart cards are not computers, when in fact, they are a smart card-a type of chip card- is a plastic card embedded with a computer chip that stores and transacts data between users. This data is associated with either value or information or both and is stored and processed within the card's chip, either a memory or microprocessor. The card data is transacted via a reader that is part of a computing system. Smart card-enhanced systems are in use today throughout several key applications, including healthcare, banking, entertainment and transportation. To various degrees ,all applications can benefit from the added features and security that smart cards provide. Because smart cards are indeed tiny computers, it's difficult to predict the variety of applications that will be possible with them in the future. It's quite possible that smart cards will follow the same trend of rapid increases in processing power that computers have, following "Moore's Law" and doubling in performance while halving in cost every eighteen months. Smart cards have proven to be quite useful as a transaction/ authorization/ identification medium in European countries. As their capabilities grow, they could Become the ultimate thin client, eventually replacing all of the things we carry around in our wallets ,including credit cards, Licenses, cash ,and even family photographs.(The photographs could be viewed and/or exchanged by capable terminals or personal computers).By containing various identification certificates, smart Cards could be used to voluntarily identify attributes of ourselves no matter where we are or to which computer network we are attached. RSA was widely used by the manufactures of the microcontrollers of smart cards .

However, the computational power of smart cards is very limited and the on-card implementations are much slower than that on desktops. This paved a way to for high-end smart card controllers to have special hardware, called a J_2-crypto-coprocessor[1]. The J_2-crypto-coprocessor is a specialized circuitry that is able to perform fast modular addition and multiplication which in turn accelerates encryption and decryption of public key cryptographic algorithms that use the computing intensive modular multiplication and addition like J_2-RSA . J_2-RSA key generation depends upon the efficient generation of prime numbers quick and correct like RSA . This paper describes some efficient prime generation algorithms and prime number testing algorithm that are relatively fast and explain about Jordan function. The rest of this paper is organized as follows. Section II gives a brief introduction to J_2-RSA cryptosystem, J_2 -RSA algorithm and J_2 -RSA key generation. Section III presents the prime number generation tests. Section IV contains the an efficient Prime number generation algorithms . Section V presents a new method of generating J_2 -RSA keys –on-board key generation. An algorithm to describe the working of this key generation is mentioned. This paper is concluded with the acknowledgements and a conclusion finally.

II. J_2 - RS CRYPTOSYSTEM

J_2-RSA is the new kind of public-key cryptosystem .The functioning's of this algorithm mostly all are similar to that of RSA functioning's. It is also used for both public key encryption and digital signatures. The security of J_2-RSA relies on the integer factorization like RSA .

A. J_2-RSA ALGORITHM

Two large primes p and q are chosen by the each individual user and the integer

Manuscript received on April 14, 2012.

Dr. L.Sreenivasulu Reddy ,Department of Computer Science and Engineering ,Madanapalle Institute of Technology and Science , P.B.No.14, Angallu, Madanapalle-517325 ;A.P; India.
(E - mail: sreenivasulureddy.svu@gmail.com).

$J_2(n)$ where $n = pq$ is published. Next, each user chooses a public key e that is relatively prime to $(P^2 - 1)$ and $(q^2 - 1)$. Finally, each user computes the secret key d according to $ed \equiv 1(\mathrm{mod}\, J_2(n))$. The public parameters are $(J_2(n), e)$ while the secret parameters are $(p; q; d)$. To send a message M to Bob, Alice locates Bob's name in the directory and obtains his public key $(J_2(n), e)$ and computes the cipher text $C = M.e(\mathrm{mod}\, J_2(n))$. Next, to recover the plain text M , Bob uses his private key and the modulus and computes $M = C.d(\mathrm{mod}\, J_2(n))$. J_2-RSA algorithm provides a procedure for signing a digital document, and verifying whether the signature is indeed authentic. A digital signature cannot be a constant; it is a function of the document for which it was produced. Suppose Alice wants to sign a message, and Bob would like to obtain a proof that this message is indeed signed by Alice. Alice takes the message M , uses her secret key and computes $\mathrm{S} = \mathrm{Md}(\mathrm{mod}(J_2(n))$ .Next, she sends M and S to Bob. Then Bob can verify that S corresponds to Alice's signature on the message M by checking whether $\mathrm{Se}(\mathrm{mod})\ (J_2(n)) = \mathrm{M}$ where e is the public key of Alice. Otherwise, either the original message M or the signature S is modified, thus, the signature is not valid.

B. J_2-RSA Key Generation

As mentioned in the above section, $J_2(n)$-RSA keys is a pair of matching public/private keys. $J_2(n)$-RSA uses a key for encryption that is different from the decryption key. The key generation requires two primes p and q so that

(i) $(p^2 - 1)$ and $(\mathrm{q}^2 - 1)$ are co-prime to public component e and

(ii) $\mathrm{J}_2(\mathrm{n}) = ((\mathrm{p}^2 - 1).(\mathrm{q}^2 - 1))$ is exactly an l bit integer, where l is the bit length of n . The advantage in this approach is the running time which is the less expensive operation in generating these keys . Another solution consists in pre-computing values for p and q for various pairs (e, l) and to store those values in a non-volatile memory of crypto-coprocessor. Non-volatile memory is the drawback here as it is more expensive. Using very efficient algorithms, 1024 bit modulus $\mathrm{J}_2(\mathrm{n})$ along with d can be generated in few seconds on current smart cards.

III. PRIME NUMBER GENERATION

The generation of prime numbers underlies the use of the most public-key schemes, essentially as a major primitive needed for the creation of key pairs or as a computation stage appearing during various cryptographic setups. Despite decades of intense mathematical studies on primality testing and an observed progressive intensification of cryptographic usages, prime number generation algorithms remain scarcely investigated. Common generators typically require $\mathrm{O}((\mathrm{J}_2(\mathrm{n} = \mathrm{pq}))^4)$ or $\mathrm{O}(\mathrm{J}_2(\mathrm{n} = \mathrm{pq})^4 / l\mathrm{og}\, \mathrm{J}_2(\mathrm{n} = \mathrm{pq}))$ bit operations where $\mathrm{J}_2(\mathrm{n} = \mathrm{pq})$ is the bit-length of the expected prime number [3]. Thus, there is a need for algorithms that substantially reduce the value of the hidden constants, therefore providing much more efficient prime generation algorithms. It is also necessary to optimize the performance of the algorithm used for finding primes in order to optimize the performance of the algorithm for generating the $\mathrm{J}_2(\mathrm{n} = \mathrm{pq})$-RSA key pair [3].

For $\mathrm{J}_2(\mathrm{n} = \mathrm{pq})$-RSA, it is very important to choose prime numbers p and q forming the modulus randomly. $\mathrm{J}_2(\mathrm{n} = \mathrm{pq})$-RSA based on the difficulty to factorize the modulus $\mathrm{J}_2(\mathrm{n} = \mathrm{pq})$ in the form $\mathrm{J}_2(\mathrm{n} = \mathrm{pq}) = ((\mathrm{p}^2 - 1)\ (\mathrm{q}^2 - 1))$ also means that someone will not be able to infer the numbers $(\mathrm{p}^2 - 1)$ and $(\mathrm{q}^2 - 1)$, by another way than the factorization,

A. Primality and Compositeness Tests

Primality tests declare whether a number is prime with probability with 1. Sieve of Eratosthenes and Modular Search Method fall in this category. Compositeness test declares whether a number is composite with a probability 1 or prime with probability ! 1. Fermat test, Solovay-Strassen test and Miller-Rabin test fall under this category. Sieve of Eratosthenes was the first known means to test for primality and to factorize numbers. It simply verifies the divisibility of the number $\mathrm{J}_2(\mathrm{n} = \mathrm{pq}) = (\mathrm{p}^2 - 1)(\mathrm{q}^2 - 1)$ to test by all the primes starting from 2 to $\sqrt{\mathrm{J}_2(\mathrm{n=pq})}$. It is practically unthinkable to use this kind of algorithm for generating primes like in $\mathrm{J}_2(\mathrm{n})$-R.SA key generation. Modular Search Method has excellent execution time when using an arithmetic processor but the memory space needed to store the numbers appears dissuasive, in particular on smart cards where memory is subjected to strong size constraint. Compositeness test also called probabilistic primality tests seems to be much faster than primality tests. A number is a prime number with a high probability if it passes a probabilistic primality certain number of times. In the next section, we will see many different prime generation algorithms. To improve the speed of generating prime numbers it is possible to use some properties of numbers to build probabilistic tests such that all prime numbers pass the tests and the other ones pass with a probability x .

B. Fermat Test

The (little) Fermat theorem says that $a^{n-1} \equiv 1 \pmod n$,for every prime n and base a relatively prime with n. Hence the algorithm for the test goes like this

1 : for $i = 1$ to t

2: Choose randomly a with $2 \le a \le n-2$

3: $r = a^{n-1} \pmod n$

4: if $r \neq 1$ return composite

5: end for

6: return n n prime

There exist composite numbers n always passing Fermat 's pseudo-primality test called Carmichael Numbers [4].

C. Solovay-Strassen Test

The test is a further improvement of Fermat' s test.

$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod n$ where $\left(\frac{a}{n}\right)$ is Jacobi Symbol.

The probability for a composite number to pass this test is less than $1/2^t$ if t is the number of randomly chosen base a [4].

B. Miller-Rabin Test

Miller-Rabin test is further more simple and easy to implement test than the previous ones. This is a primality test that provides an efficient probabilistic algorithm for determining if a given number is prime. It is based on the properties of strong pseudo primes. The algorithm proceeds as follows. Given an odd integer n let $n = 2^r s + 1$ with s odd. Then choose a random integer a with $1 \le a \le n-1$ .If $a^s \equiv a^{2^j} s \equiv -1 \pmod n$ for some $0 \le j \le r-1$, then n passes the test. A prime will pass the test for all a. The test is very fast and requires no more than (1+o(1) log n) multiplications (mod n),where log is the logarithm base 2. Unfortunately, a number which passes the test is not necessarily prime. Monier and Rabin have shown that a composite number passes the test for at most 1/4 of the possible bases a. If N multiple independent tests are performed on a composite number, then the probability that it passes each test is $1/4^N$or less [4]. The algorithm (n odd) is shown below.

1: find t and q such that n = 2^t + 1

2:for i=1 tot

3: choose randomly a with $2 \le a \le n - j2$

4: x = a^q modn

5: if $(X \neq 1)$

6: i = 1

7: while (x ≠ n-1)do

8: if(i = t) return composite

9: x = x^2 mod n

10: if(x =1) return composite

11: i++

12: end while

13: endif

14: endfor

15: return n pseudo-prime

C. Phmality testing with Elliptic Curves

Let n be a suspected prime. If it is really a prime, then | E(a, b)/n lies in the interval (n + 1 -2 $\sqrt{n}$, n + 1 + 2 $\sqrt{n}$).Furthermore, it is known that if n is prime, then there are always many elements of high order. If we can factor | E(a; b)=n, say | E(a, b)=n| = a_1x ----n a_r then we choose a point P = (X_0, Y_0, Z_0) at random and find its order. The order must be divisor of/E(a; b)=n/and hence we can start by verifying P#(|E(a, b)/ n| qi) for each i. If then Z coordinate is relatively prime to n for each / and if n divides the Z coordinate of P# \ E(a, b)/n, then the order of P in E(a, b)/p is | E(a, b)/n≥ n+1-2 $\sqrt{n}$ for any prime p dividing n which implies that n is prime. There is still hope even if one or more of the Z coordinates is divisible by n. For each such /, we find the smallest positive integer bi, such that the Z coordinate of P#(|E(a, b)/n | / qi^bi,)
is relatively prime to n.

IV. PRIME GENERATION ALGORITHMS

1) Generating J_2(n=pq)- RSA Primes: This is a simple algorithmic outline to generate J_2(n=pq)- RSA primes. 1. Pick a k bit odd m uniformly at random from [m/2,m] 2. Apply test division on m by all primes less than a certain small prime 3. If m passes trial division test, then apply the (strong) pseudoprimality test for r different bases 2, 3, 5, 7... 4. If m passes all r (strong) pseudoprimality tests, then m is a prime number with high probability 5. If m fails, take m := m + 2gotoStep2

2) Naive prime finding algorithm: The naive prime generation algorithm is sketched in the following table. Neglecting calls to the random number generator, the expected number os trials here is asymptotically equal to (In 2"/2). 89 trials are required to generate a 256 prime number. A naive approach to find an n bit prime number is to randomly choose a b bit odd number and call a compositeness test function fusing the odd number as input. If the function f returns the result that the number is not a prime, another random number is chosen and the procedure of testing it with a function is repeated until a prime number is found, which would the prime used for key generation. Hence, the implementation of the primality test function f must be optimized and the numbers of tests (number of calls to /) should be minimum in order to optimize the performance of the prime finding algorithm [3].Table 1 below describes the algorithm clearly. The T(q) function mentioned in the algorithm is nothing but the test of primality.

Table: l Naive Prime Finding Algorithm

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

3) Sieve of Eratosethenes: The sieve of eratosethenes was the first known means to test for primality and to factorize numbers. It verifies the divisibility of the number n to test by all primes starting from 2 to Vn. It is very fast with the first small primes but its computation time essentially grows linearly ith n.

Sieve is always used to rapidly eliminate the randomly chosen prime candidates having very small factors. The function Q(x) where Px is the set of all primes < x may be defined as Q(x) = rip€px(1-1/p) [5]. For a n larger than x, this function may be interpreted as the probability for n to be relatively prime with all primes in Px. Practically, a sieve can be implemented by evaluating the GCD of the product all the lements of Px with the number n to test.Table 2 lists out the algorithm for this test. The function S(k) is the set of prime numbers.

Table:2 Sieve of Eratosethenes

1. Let p, be the i- th smallest odd prime (p_1=3, p_2=5,...)
2. Let S(k) be a set of prime small prime such that S(k) = (p_ij$p_i \leq$ k, i € N), where k can be any positive integer.
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.

4) Prime finding algorithm using trial division**:** One of the most used sieve methods is the trial division method. If we are given an integer less than a million, we can find its prime factors fairly quickly just by using the fact that if it is not a prime, then it must have a factor less than its square root. Thus, take a list of all primes and try dividing them into the 4 number to be factored. If none of them divide evenly, then the original number was prime. Each time we find a prime divisor, we divide it out. Once the unfactored portion that remains is less than the square of the last prime tested, it can be known that the unfactored portion has to be prime [5]. The algorithm is described in the Table.3.

Table:3 Prime finding algorithm using trial division

1.Choose a set S(k). Pick a n- bit odd random number q and let $q_{(o)=}$ q, i=0
2.Let $W^j_{(j)} = q^{(i)} \bmod p_j$. If M/'(y) = 0, for any 7, $1 \leq j \leq k$, goto 4
3.If T($q^{(i)}$)= TRUE, output q^i and halt. 4. $q^{i+1}=q^{i+2}$, i := i+ 1, goto 2

5) Prime finding algorithm using table look - up: The modular reductions are very expensive. Hence, it is desirable to keep the number of modular reduction operations on the sieve procedure to minimum. The modular reduction in trial division algorithm is shown in Table 3. Although modular reduction can be used to compute ^{wj} j, there is an efficient way to improve . If $q_{i+1} = q_{i+2}$ and $w^i_j = q_i \bmod p$, then W^i j = W^i j + 2mod p. Finally, one can calculate wi+1 j from wi j . In addition, if pj isassumed to be 8-bits long, so is w^j j. So the computation of w^{i+1} j, only uses two 8-bil operands, resulting in a performance much faster Than modular reduction operations. The algorithm is described in 1 able 4 [5].

Tftblc:4 Prime finding algorithm using table look-up

1. Choose a set S(k). Pick a n-j bit odd random number q and let q(0)=q. i=0
2. Compute $W^0{}_{(j)} = q^{(0)} \bmod p_j; 1 \leq j \leq k$
3. If W^i(j) = 0, for any j, $1 \leq j \leq k$, goto 5
4.. If T($q^{(i)}$) = TRUE, output q^i ad halt
5. W^{i+1}(j) = $W^{(i)}$ +2 mod p_j; $1 \leq j \leq k$
6. q ($^{(i+1)}$ = q^i, I = I +1, goto 3

The algorithm requires a table [w_1;w_2,:::] to keep all the residues w^ij of the previous iteration. Hence this algorithm is referred to as table look-up algorithm.

6) Prime finding algorithm using bit array: Let us consider the interval of /test candidates, say. $q^{(0)}, q^{(1)}.....q^{(i-1)}$. A bit array A is defined[$a_0 a_1 ::: a_{i-1}$] where a_1, is thc l –th bit, initially set to 0 with a, representing the number $q^{(i)}$ .For each prime p from the set S(K), we find a starting point g[p] =min(i/q^i) is divisible by p. i in N. If g(p) is less than /, set $a_{i(p)}$ to land then every p - th bit of A is set to 1 since the values of that are represented by these bits would also be divisible by p. After this sieve, a_i is zero if and only if $q^{(i)}$ is not divisible by any of the numbers in S(k).The sieve is then concluded and the prime finding algorithm must only scan the hit array A and try the primality test T for each $q^{(i)}$ such that a_i is zero. This algorithm is called bit array algorithm. The bit array algorithm will find a probable prime if there is one in the chosen interval. In the case that there is no probable prime on the chosen interval, once can either randomly choose another odd q and let q(01 = c/Mt 1-2. Table 5 describes the algorithm [5].

Table:5 Prime firiding algorithm using bj $\leq$array

l Set a_i = 0, for $0 \leq j \leq l-1$
2. Pick a n-j bit odd random number q and let $q_{(0)}$=q, i=0
3. For each p_j €S(k),do
3.1 Compute w0j =q(0) mod pj
3.2 Compute g(pj)
3.3 Set $a_{g(pj)}$+m = 1; $0 \leq m \leq [(l - g(p_j))/pj]$
4. for i: = 0 to l - 1, do
4.1 If (a_i=0) and T($q^{(i)}$)= true, output $q^{(i)}$ and halt.
4.2 $q^{(i+1)}=q^{(i)}+2$
4.3 $q^{(0)}=q^{(i)}$, M), goto 3

V. ONBOARD GENERATION

Onboard key generation is preferable as the keys are not imposed by the card manufacturer. This is a new method for the generation of J_2(n=pq)- RSA keys. This algorithm is divided into two phases. The first phase is performed offline before the values of (l;e) are even known. The second phase is performed online by the cryptographic device once (l; e) are known. This method is supposed to be very fast when compared any other method of J_2(n=pq)- RSA key generation .

Table: 6 Onboard J_2(n=pq)-RSA prime generation algorithm

Input: parameters l_0; e and a (of large order) in $Z^*_{j2(n=Pq)}$
Output: a number (q^2 – 1) /n [$2^{/0-1/2}$]
1. Compute v =f [$2^{/0-1}$=2] / J_2(n=pq) and w = b^{t0} / J_2(n=pq)
2. randomly choose i € R v; :::;w i 1 and set / ← J_2(n=pq)
3. randomly choose k€R $Z^*_{J2(n=pq)}$
4. Set (q^2-1) $\leftarrow k+l$
5. If ((q2-1) is not prime) or (gcd(e, q^2 -1) $\neq$ 1) then
5.1 Set kak(mod J_2(n=pq))
5.2 Goto step 4
6. Output (q^2-1)

The produced(q2-!) (p2-!) by this algorithm has the form of (q2-!) = $a^{f-1}k_{(0)}$ mod $J_2(n=pq)$+ i
$J_2(n-pq)$

where $K_{(0)}$ denotes the initial value of k and f is the number of failures of the test in Step 5 [7].Memory requirements are reduced radically if/O bit prime q can be stored as a pair (l;f) where i is a unique indexed identifier (i is used as the input of a pseudo random number generator for constructing v;w; j; $K_{(0)}$).This phase is performed offline. Then the online stage consists in reconstructing primes from pairs (i; f). Moreover, in addition to be a fast, this method allows the online generation of $J_2(n=pq)$-RSA modulii $J_2(n=pq)$ of arbitrary length from a very small set of values computed at the offline hase. Furthermore, parameters can be chosen so that keys are guarenteed to work for the usual public exponent e which is usually let at the discretion of the end user [7].

VI. CONCLUSION

A basis for understanding the key generation on smart cards is presented in this paper. Time constraint in generating $J_2(n=pq)$- RSA keys on smart cards is clearly pointed and algorithms that can optimize the amount of time required to generate the keys on smart cards are highlighted. Most of these algorithms are implemented with varying results. There are several factors that must be taken into account when designing cryptographic algorithms for smart cards. There are several other algorithms that can efficiently generate $J_2(n=pq)$- RSA key pairs but they couldn't overcome the specified constraint on the smart cards. At present, a cheaper but effective solution may be to have an on-board key generation. So, sets of keys will be generated only if they will be used. Furthermore, they is more memory available and this method is more secure as private keys are only owned by the end user. Recent study shows that on-board generation takes few seconds on an average. Larger key sizes present new challenges since smart card crypto coprocessor have fixed register sizes that will not be able to accommodate large numbers. Elliptic Curve Cryptography (ECC) can provide the same level of security using smaller $J_2(n=pq)$- RSA key lengths. A future work would be the designing of key generation algorithms using the Elliptic Curve Cryptography.

REFERENCES

1. J. J. Quisquater and B. Schneier, Smart Card Crypto- Coprocessors for Public-Key Cryptography, vol. 1820 of Lecture Notes in Computer Science, Springer Verlag, 2000.
2. C,. K. Koc,, "High-Speed RSA Implementation," Tech. Rep. TR 201, RSA Laboratories, 73 pages, November 1994.
3. M. Joye, P. Paillier, and S. Vaudenay, "Efficient Generation of Prime Numbers," Cryptographic Hardware and Embedded Systems, pp. 340-354, Aug. 2000.
4. J. F. Dhem, Design of an Efficient Public-Key Cryptographic Library in RISC-based Smart Cards, Ph.D. thesis, Unbiversit Catholique de Louvain, May 1998.
5. Chenghuai Lu, A. L. M. Santos, and F. R. Pimentel, "Implementation of Fast RSA Key Generation in Smart Cards," in Proceedings of the 2002-ACM Symposium on Applied computing. 2002, pp. 214-220, ACM Press.
6. N. Feyt and M. Joye, "A better use of smart cards in pkis," Gemplus Developer Conference, Nov. 2002.
7. N. Feyt, M. Joye, and P. Paillier, "Off-line/on-line generation of RSA keys with smart cards." 2nd International Workshop for Asian Public Key Infrastructures, pp. 153-158, Oct. 2002.

AUTHORS PROFILE

Dr. L.Sreenivasulu Reddy ,Department of Computer Science and Engineering ,Madanapalle Institute of Technology and Science ,P.B.No.14 ;Angallu, Madanapalle-517325 ;A.P; India.
email: sreenivasulureddy.svu@gmail.com