IPHV1I20012X
International Journal Of Advanced Research and Innovations Vol.1, Issue .2
ISSN Online: *2319 – 9253*
Print: *2319 – 9245*

AN ENHANCED RSA ALGORITHM FOR LOW COMPUTATIONAL DEVICES

Maheswari Losetti[1] Kanaka Raju Gariga[2]

1. *Assistant Professor, Department of IT, Vardhaman College of Engineering, Hyderabad.*
2. *Associate Professor, Department of IT, Vardhaman College of Engineering,Hyderabad.*

ABSTRAC*T*

Authentication and Confidentiality can be realized by using RSA Signature algorithm .RSA is widely used in public-key cryptosystem. But running this algorithm needs lots of time and memory. So, running of this algorithm become difficulty low computational devices such as PCs to laptops, mobile phones, digital television sets, and so on.

In this paper a RSA signature algorithm to fit for the devices with low computational power. The new signature algorithm is based on complex numeric operation function. This paper expounds the fundamental principles of RSA algorithm. The realization of RSA algorithm includes the generation of RSA cryptographic key and the encryption and decryption of data. By using RSA algorithm, we can use the private key of the sender to sign the plaintext and the public key of the receiver to encrypt. For the receiver, he can use his private key to decrypt and the public key of the sender to verify the signature.

Key words- Low computational devices, Signature algorithms, RSA, Complex numeric operations.

I. INTRODUCTION

The emergence and development of data encryption technology provides an important guarantee for global Ecommerce, which makes the Internet based electrical tradeoff be feasible. The main purpose of digital signature technology in E-Commerce is guarantees the confidentiality, authentication, data integrity and undeniable of the information that transformed in insecurity and unreliable network. Until now, there are many digital signature algorithm established, in which the Hash, DSA and RSA are the common ones. The main restriction of Hash algorithm is the receiver must have a copy of the sender's private key to verify the signature, so the encryption system is easy to be attacked and the signature can be forged. DSA (DigitalSignature Algorithm) is another common public key algorithm, but it has no data encryption function, it can only be used for digital signature. Compared with Hash algorithm, the private key for generating signature in RSA system only stores in user's computer, it is more secure than Hash algorithm. While compared with DSA algorithm, RSA can either be used for data encryption or digital signature, which makes it, be the most widely used public key algorithm. In order to guarantee the security of a RSA system, the length of public and private keys is usually greater than 1024 bits in current commerce use, thus the key generation and data encryption/decryption process are all large number operations, which makes the speed of a RSA algorithm about 100 times slower than DES algorithm. The processing speed is a major flaw of RSA algorithm either for hardware or software implementation, so how to design an effective large number operation scheme is an important question [1,3]. In this paper, an *n* carry array based large number denotation approach is proposed to reduce the complexity of large number operation. The simulation results indicate that the speed of proposed algorithm has an efficient improvement over classic string based large number operation method.

This paper is organized as follows in Section II, Exploring the details for RSA algorithms and it mathematical foundation. Section III, is presenting problem in RSA algorithm running in low computational devices. Then IV section, illustrate the details of a n carry

array storage structure and it followed by conclusion in section V.

II. THE RSA ALGORITHM AND ITS MATHEMATICAL FOUNDATION

A. The Mathematical Foundation for RSA Algorithm: The RSA digital signature has precise mathematical foundations, which are as follows:

Theorem 1 (fundamental theorem of mathematics) Any positive integer *a* can be denoted as $a=P_1^{a1} ..P_n^{an}$
which $P_1 > P_2 > P_3 \ldots > P_T$ are all prime numbers, $ai > 0$.

Theorem 2 (Euclid theorem) Any two integers a and b has a greatest common factor d, in which d can be expressed as the linear combination of a and b with integer coefficient, namely $s, t \in Z$, which satisfies $d = sa + tb$.

Theorem 3 (Fermat theorem) If *p* is a prime number, then for any positive integer a that prime to *p*, $a^{(p-1)} \equiv 1 \pmod{p}$.

Definition 1 (Euler function (*n*)) When $n = 1$, $\phi(1) = 1$, when $n > 1$, the value of $\phi(n)$ is the amount of positive integer that less than *n* and prime to *n*.

Theorem 4 If *p* and *q* are all prime numbers and $p \neq q$, then $\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$

Theorem 5 (Euler theorem) If integer *a* is co-prime to integer n, then $a^{\phi(n)} \equiv \pmod{m}$.

Above theorem have the following 3 deductions:

(1) If p is prime number and $n = p$, then $a^{(p-1)} \equiv 1 \pmod{p}$, namely the Fermat theorem.

(2) $a^{\phi(n+1)} \equiv a \pmod{p}$.

(3) If $n = pq$, p and q are prime numbers and $p \neq q$, for $0<m<n$, if $(m, n) = 1$, then $(n)\ 1\ m\ m\ \phi + \equiv \pmod{n}$, namely $m^{(p-1)(q-1)+1} \equiv \pmod{n}$.

Above five theorems will be used in the feasibility proof of RSA digital signature algorithm in the following section.

Theorem 6 If *p* and *q* are prime numbers and $p \neq q$,
$rm \equiv 1 \pmod{(p-1)(q-1)}$, *a* is any positive integer,
$b \equiv am \bmod pq$, $c \equiv br \bmod pq$, then $c \equiv a \bmod pq$.

B. RSA Key Generation Algorithm:

1. Generate two large random primes, *p* and *q*, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits.
2. Compute n = pq and (φ) phi = (p-1)(q-1) [Theorem 4].
3. Choose an integer *e*, 1 < e < phi, such that gcd(e, phi) = 1. [Theorem 2].
4. Compute the secret exponent *d*, 1 < d < phi, such that ed ≡ 1 (mod phi). [Theorem 6].
5. The public key is (n, e) and the private key is (n, d). Keep all the values d, p, q and phi secret.
   - n is known as the *modulus*.
   - e is known as the *public exponent* or *encryption exponent* or just the *exponent*.
   - d is known as the *secret exponent* or *decryption exponent*.

C. Encryption Algorithm:

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m*
3. Computes the ciphertext $c = m^e \bmod n$.
4. Sends the ciphertext *c* to B.

D. Decryption Algorithm:

Recipient B does the following:-

1. Uses his private key (n, d) to compute $m = c^d \bmod n$.
2. Extracts the plaintext from the message representative *m*.

E. Digital Signature Algorithm(DSA):

Digital Signing

Sender A does the following:-

1. Creates a *message digest* of the information to be sent.
2. Represents this digest as an integer *m* between 0 and *n*-1.
3. Uses her *private* key (n, d) to compute the signature $s = m^d \bmod n$.
4. Sends this signature *s* to the recipient, B.

Signature Verification

Recipient B does the following:-

1. Uses sender A's public key (n, e) to compute integer $v = s^e \bmod n$.

2. Extracts the message digest from this integer.
3. Independently computes the message digest of the information that has been signed.
4. If both message digests are identical, the signature is valid.

III. COMPLEXITY OVER RUNNING RSA IN LOW COMPUTATIONAL DEVICES

To improve usability of the RSA algorithm we need to increase the adaptability of the RSA algorithms to low computational devices and has to increase the security like confidentiality and authentication features. Why because present days e-commerce application and online banking transaction are running on low computational devices also. Here processing speed is very less compare to be super computers where we deploying servers to done high speed computation. This lead to be biggest problem to the RSA algorithm in term of reducing processing time. In RSA algorithm, to promise security levels we need bigger length of prime number ex: 1024 bit, more than that. So, it takes much processing time to complete those computations such as Generating Keys, Encryption, Decryption and DSA. This can be proved by current implementation of RSA algorithm in java programming language *jdk1.7.0 versions packages java.math.BigInteger and java.security.SecureRandom* having platform of,

Hardware configuration:
Processor - Intel(R) Core(TM) 2 Duo 2.20 GHzs.
RAM - 3.00GB
Software Configuration:
Running Platform - JDK1.6.0
Operating system - Windows 7 Ultimate
32-bit OS.

For that implementation time taking to the bit length on above platform as show in flowing graph

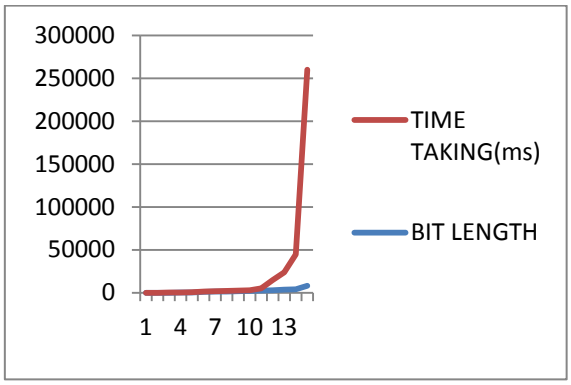

Figure 1. Time variation graph with bit length.

From the above graph time taking is varying based on bit length, but not in same order, from 32bit to 1024 time is varying in sequential order after that 2048 to 8196 and so on. is in exponential order. Above time factor can rapidly various to Mobile phone where processing speed less than 1.00GHzs. To overcome this drawback there two possible solutions are they i.e

1. Reduce the bit length.
2. Invent new processing mechanisms.

If we are looking for first possibility it will reduce system security, why because security is directly propositional to key length. So, we need to introduce an advanced computational mechanism that can solve our problem this computational mechanism we can implement using n carry array storage structure for large numbers .

IV. AN EFFICIENT IMPLEMENTATION OF RSA ALGORITHM FOR LOW COMPUTATIONAL DEVICES

A. A n carry array storage structure for large number

The most import stage in RSA application is the key generation process, the selection of large prime number *p* and *q* to construct modulo *n* is crucial. In practice application, n must be large enough in order to guarantee the security of a RSA system. So the efficiency of a RSA system depends on the large number calculation speed. Most compilers support 64 bits integer operation at present, the integers calculated must equal or less than 64bits, namely 0xffffffffffffffff (18446744073709551615), which is far short

than the RSA algorithm requires. The classic large number storage method is string based, a large number is stored in a character type array, and then we can construct the corresponding function to perform add, subtract, multiply and divide operation based on the array. But the efficiency of this scheme is very low because for a 1024 bits number, the length of decimal form is about several hundreds, any numeric operation should do multiple nested loop on two long character array, besides a large extra space is needed to store the carry flag and middle results, which leads to the heavy system resource occupy and low efficiency [4,6].

This paper proposes an n carry array beads large number storage scheme. For a 32 bits system, n can be 2 powers of 32, namely 0x10000000. If a 1024 bits large number converts to 0x10000000 carry, its length should be 32, the range of each digit is 0~0xffffffff rather than 0~1 or 0~9 Each digit of a n carry number just can be stored in a unsigned long integer type unit, so a 1024 bits large number can be stored in a unsigned long array with 32 elements. Then we can implement numeric operation on the array, the loop time is reduced to 32, the calculation speed is greatly improved.

B. Complex numeric operation function

In order to deal with the complex numeric mathematical operation efficiently, the following 3 functions are constructed in RSA system implemented.

1 Miller-Rabin Function

Function : prime number judgment.
Invoking method : N.Rab().
Return value : if *N* is prime, return 1, else return 0.
Algorithm description
Step 1:Calculating M, which makes
$N = (2r) M + 1$
Step 2: Selecting a random number $A < N$;
Step 3: If $AM \bmod N = 1$ or for any $i < r$,
$A2i\ M \bmod N = N-1$,then *N* passes the testing of random number *A*;
Step 4:Testing *N* 5 times, each time with different *A*, if all the 5 times testing are passed, then *N* is prime.

If *N* passes one testing, then the probability that *N* is not a prime number is 25%. If t times testing are passed, then the probability that *N* is not a prime number is $(1/4)^t$. When $t = 5$, the probability that *N* is a prime number is 99.99%. The worst case is that *N* contains smaller prime factor. In this function, 600 small prime numbers is used for testing, so the reliability is guaranteed.

2 Extended Euclid Function

Function : calculation the multiplicative inverse of integer *N*.
Invoking method : N.Euc(A).
Return value : *Y*2, which satisfies NY2 mod *A* = 1.
Algorithm description
Step 1: Let $c = \gcd(a, b)$ denotes the greatest common factor of *a* and *b*;
Step 2: (*X*1, *X*2 , *X*3) ←(1,0, *A*) ;
(*Y*1, *Y*2 , *Y*3) ←(0,1, *N*);
Step 3: If *Y*3 = 1 then return *Y*3 = gcd(*A*, *N*); no inverse;
Step 4: If *Y*3 = 1 then return *Y*3 = gcd(*A*, *N*);
*Y*2 = *N* -1 mod *A*;
Step 5: $Q=[X_3/Y_3];$
Step 6:
(*T*1, *T* 2 , *T* 3) ←(*X*1- *QY*1, *X*2 - *QY*2, *X*3 - *QY*3) ;
Step 7: (*X*1, *X*2 , *X*3) ← (*Y*1, *Y*2 , *Y*3);
Step 8: (*Y*1, *Y*2 , *Y*3) ←(*T*1, *T* 2 , *T* 3) ;
Step 9: go to step 2.

The variables used in the algorithm have the following relationships:
*AT*1+ *NT*2= *T*3; *AX*1+ *NX*2= *X*3; *AY*1+ *NY*2= *Y*3;

3 Montgomery Function

Function : calculation the modulo of power.
Invoking method: N.Mon (A, B).
Return value : $X = N^A \bmod B$.
Algorithm description
Step 1: *X* = 1;
Step 2: If *A* is odd, then *A* = *A* −1,
X = *X* * *N* mod *B* ;
Step 3: If *A* is even, then *A* = *A* / 2，
N = *N* * *N* mod *B* ;

Step 4: Repeating step (2) and (3) to reduce the order *A*, until *A* = 0;
Step 5: The final calculated X is the result we want.

V. CONCLUSION

The random RSA public and private key pair with arbitrary length can be generated effectively by using the java large number library design by the algorithm proposed in this paper. A 1024 bits RSA key can be generated within 4 seconds on common PC platform, while the encryption/decryption operation on data less than 1024 bits can be done within 2 seconds, the efficiency of RSA system is greatly improved, which provides important guarantees for implementation high security RSA algorithm with long keys on PC platform.

REFERENCES:

[1]. MJ Wiener. (1990), "*Cryptanalysis of short RSA secret exponents*", IEEE Transactions on Information Theory, Vol 36, No 3, pp 553-558.
[2]. R Gennaro. (2000), "*RSA-Based Undeniable Signatures*", Journal of Cryptology, Vol 13, No. 4, pp 397-416.
[3]. R Cramer, V Shoup. (2008), "*Signature schemes based on the strong RSA assumption*", ACM Transactions on Information and System Security, Vol 3, No 3, pp 161-185.
[4]. R Gennaro. (2008), "*Robust and Efficient Sharing of RSA Functions*", Journal of Cryptology, Vol 13, No 2, pp 273-300.
[5]. D Boneh, M Franklin. (2001), "*Efficient generation of shared RSA keys*", Journal of the ACM, Vol 48, No. 4, pp 702-722.
[6]. D Boneh, M Franklin. (2001), "*Efficient generation of shared RSA keys*", Journal of the ACM, Vol 48, No. 4, pp 702-722.

Author's Profile

Maheswari Losetti received the B.Tech degree from JNTUH Hyderabad, the M.Tech degree in computer science and engineering from JNTUH Hyderabad.

She is currently an Assistant Professor of computer science and engineering with the Department of Computer Science and engineering. Her current research interests in networks include cloud computing, information security, hacking.

Kanakaraju Gariga received the B.Tech degree from JNTUH Hyderabad, the M.Tech degree in computer science and engineering from JNTUH Hyderabad.

He is currently an Associate Professor of computer science and engineering with the Department of Computer Science and engineering. He current research interests in service oriented operating systems include networking, cloud computing, green computing.