International Journal of Advanced Computer Research (ISSN (Print): 2249-7277 ISSN (Online): 2277-7970)
Volume-4 Number-4 Issue-17 December-2014

Modular Arithmetic in RSA Cryptography

Sridevi[1*] and Manajaih.D.H[2]

Received: 15-December-2014; Revised: 18-January-2015; Accepted: 18-January-2015

Abstract

Faster implementations of public-key cryptography and in particular of RSA are of uttmost importance nowadays. Performing fast modular multiplication for large integers is of special interest because it provides the basis for performing fast modular exponentiation, which is the key operation of the RSA cryptosystem. Currently, it seems that in a radix representation, all major performance improvements have been achieved. Nevertheless, the use of Residue Number System(RNS) proves to be a promising alternative for achieving a breakthrough. All these aspects are detailed throughout this research paper. Also presents an overview of the RSA cryptosystem, followed by a short proof of why the encryption-decryption mechanism works. With considerations regarding the employed key-sizes and with an example of a small RSA cryptosystem.

Keywords

Residue Number Systems (RNS), Public key, Cryptosystem, encryption, RSA.

1. Introduction

Generally cryptography falls into two main categories: secret and public key cryptography. Secret-key cryptography is based on a prior exchange of a common secret key. Since a single key is used for both encryption and decryption, the major issue associated with symmetric-key systems is the key distribution problem that is an efficient method has to be devised for the parties to agree upon and then exchange keys securely. In 1970, W. Diffie and M. E. Hellman proposed in [2][4] an efficient method of exchanging a shared secret key over an unsecured communications channel and thus setting up the basis of a new type of cryptography: the public-key cryptography. The assymetric-key cryptography uses a key (public) for encryption, which is made available to everyone at the sending end, and another one (secret) for decryption that is known only by the recipient of the message. In 1977, R. Rivest, A. Shamir, and L. Adleman introduced the RSA cryptosystem [7], which became the most widely used public-key cyptosystem in the world. Its security depends upon the intractability of the integer factorization problem and it can be used to provide both data encryption and digital signatures.

2. RSA algorithm

Prior to the execution of the encryption-decryption protocol, outlined in Algorithm 2, each party that wants to communicate should generate first its own public/private key pair, as described in Algorithm 1.

Algorithm 1 Public key generation
Ouput: a public key (n,e) and a private key d.

1. Generate randomly two large primes p and q, which are kept secret.
2. Compute the modulus $n = p \bullet q$ and Euler's totient function $\Phi = (p - 1)(q - 1)$.
3. Select a random integer e, $1 < e < \Phi$, coprime with Φ.
4. Compute the multiplicative inverse of e with respect to modulus Φ ($d \bullet e = 1 \pmod{\Phi}$)

Algorithm 2 RSA encryption-decryption protocol
B encrypts a message m and sends it to A; A decrypts the message

1. Encryption
   a. B should obtain the public key (n, e) of A.
   b. B represents the message m as an integer between 0 and n - 1.
   c. B computes $c = m^e \bmod n$.
   d. B sends the cipher text to A.

2. Decryption

*Author for correspondence

Sridevi, Assistant Professor, Department of Computer Science, Karnatak University Dharwad, Karnataka State, India.
Manajaih.D.H, Professor,Department of Computer Science, Managlore University, Mangalore, Karnataka State, India.

a. A should use its private key d to recover the message m form the cipher text m = $c^d \bmod n$

2.1. RSA proof

The RSA encryption system is based on Euler's theorem and its generalization, the Carmichael's theorem.

Theorem 1 (Euler's Theorem) If n *and a are two* positive, relatively prime integers, then it holds

$$\left| a^{\Phi(n)} \right|_n = 1$$

where Φ(*n*) *is the Euler's totient function (the number of integers less than n, and relatively prime with n).*

Theorem 2 (Carmichael's theorem) *If* n *and* a *are two positive, relatively prime integers, then*

$$\left| a^{\lambda(n)} \right|_n = 1$$

where λ(n) *is the Carmichael function (the least common multiple of the factors of* Φ(*n*)).

Theorem 3 *If* n *is a product of distinct primes, then, for all integers* a

$$\left| a^{\lambda(n)+1} \right|_n = |a|_{n.}$$

The correctness of the RSA scheme, i.e., the fact that the encryption and decryption are inverse operations, relies on the fact that

$$\left| m^{e\text{-}d} \right|_n = m, \quad \text{for} m \in [\,0, n-1\,].$$

There are two cases to consider.
Case 1.*gcd(m,n)* = 1
We have $|d \bullet e|\Phi_{(n)} = 1$, relation which rewritten for an integer $a > 1$ becomes

$$d \bullet e = a \bullet \Phi(n) + 1$$

$$\begin{aligned} |c^d|_n &= |(m^e)^d|_n \\ &= |m^{1+a \bullet \Phi(n)}|_n \\ &= |m \bullet (m^{\Phi(n)})^a|_n \\ &= |m \bullet 1|_n \\ &= |m|_n \end{aligned}$$

Case 2.*gcd(m, n)*> 1

For n, a product of two odd distinct primes,

$$\lambda(n) = \frac{\Phi(n)}{gcd(p-1, q-1)}$$

will always be a divisor of Φ(n). Since $\lambda(n) | \Phi(n)$, the equality $|e \bullet d|_{\Phi(n)} = 1$ implies that $|e \bullet d|_{\lambda(n)} = 1$. Using a derivation similar to case 1, and based on Theorem 3, we obtain

$$|m^{d-e}|_n = |m|_n ,$$

2.2. RSA key sizes

As far as the operands sizes are concerned, the following remarks can be made.
The stochastic prime's *p* and q should be chosen such that they have approximately the same bit length to ensure that any attempts to factor the modulus are computationally infeasible. For instance, for 1024-bit modulus *n*, *p* and *q* should be chosen about 512-bits each.

The exponent e is usually chosen small and preferably with a small Hamming weight (the number of 1's in its binary representation), in order to increase the efficiency of the exponentiation. One exponent currently used in practice is $e = 2^{16} + 1 = 65537$. The exponentiation algorithm would require in this case 16 modular squaring and 2 modular multiplications (since the Hamming weight is 2).
For security reasons, the bit length of the modulus n is typically in the range 5122048 bits or even more, and thus efficient long integer modular arithmetic is required for achieving high throughput rates at these bit precisions.

3. Modular Exponentiation

Modular exponentiation ($a^b \bmod m$) and its key constituent operation, modular multiplication ($a \bullet b \bmod m$), are the fundamental operations underlying cryptographic algorithms. Since modular multiplications account for most of the time spent for encryption and decryption, their optimization is crucial. This can be achieved either by reducing the number of modular multiplications or by reducing the latency of each modular multiplication. Assuming m and e have a bit length of 1024 each, $c = m^e$ would require a total number

$1og_2(m^e) = e \bullet log_2 m \approx 2^{1024} \bullet 1024 = 2^{1034}$
bits in order to store its value. Therefore *c* cannot be obtained by performing first the modular exponentiation m^e and only after that the reduction modulo *n*. Thus these operations have to be interleaved at each step. A straight forward way of performing exponentiation is

$$\mathrm{m} \xrightarrow{SQ} m^2 \xrightarrow{MUL} m^3 \xrightarrow{MUL} m^4 \xrightarrow{MUL} \ldots\ldots$$

However, this naive approach requires *e*-1 1 modular multiplications, which would be infeasible for large exponents. Taking into consideration that not all powers of m need to be computed in order to obtain m^e, a faster method would be: a faster method would be:

$$\mathrm{m} \xrightarrow{SQ} m^2 \xrightarrow{MUL} m^3 \xrightarrow{SQ} m^6 \xrightarrow{MUL} m^7 \ldots\ldots$$

This method is called the square-and-multiply algorithm. The algorithm provides a systematic way for finding the exact sequence in which squarings and multiplications by m have to be performed in order to efficiently compute m^e.

4. Binary Exponentiation Method

The method is based on scanning bit-by-bit the exponent. At each step (i.e., for every scanned bit) a squaring is performed. If and only if the currently scanned exponent bit is 1, then a subsequent multiplication is performed. Depending on the direction of processing the exponent bits (i.e., from MSB to LSB, or vice-versa), there exist two versions of the algorithm: the left-to-right binary method which is described below and the right-to-left binary method that is similar but requires an extra variable to keep the powers of *m.* Let $k = \lfloor log_2 e \rfloor + 1$ denote the bit length of the exponent e whose binary expansion

$$\mathrm{e} = (\mathrm{e}_{k-1}\, \mathrm{e}_{k-2} \ldots\ldots \mathrm{e}_1\, \mathrm{e}_0) = \sum_{i=0}^{k-1} e_i 2^i$$

The left-to-right binary exponentiation algorithm computes the exponentiation starting from the most significant bit position of the exponent *E* and proceeding to the right, as described in Algorithm 3.

Algorithm 3: Left-to-right binary exponentiation algorithm

Input: m, e, n

Output: $c = |m^e|_n$

1. **if** $e_{k-1} = 1$ **then** $c := m$ *else* $c := 1$
2. **for** $i = k$ -2 **downto**0

2a. $c := |c \bullet c|_n$
2b. **if** $ei = 1$ **then** $c := |c \bullet m|_n$

3. **return** *c*

Assuming e_{k-1}=1,the algorithm requires $k-1$ squaring (step 2a.) and $H(e)-1$ multiplications (step 2b.), where H(e) is the Hamming weight of the exponent (the number of ones in its binary representation). Since $0 < H(e) - 1 < k - 1$, we have a total maximum number of multiplications of 2 ‧ (k–1), a minimum of k–1, while in the average case $(H(e) = 0.5$ ‧ k, that is half of the bits of e are 1), 1.5 ‧ (k– 1) multiplications are needed. For instance, for 1024-bit exponents, the square-and-add algorithm has a logarithmic computational complexity, requiring on average only 1.5 * 1023 = 1535 multiplications, while the straightforward exponentiation needs a linear amount of 2^{1024} multiplications.

The binary method is used frequently in smart cards and embedded devices, due to its simplicity and low memory requirements.

This method can be generalized by scanning multiple bits of the exponent at a time. Generally, if *log2m* bits are scanned, the method is called m-ary. When compared to the binary method, it requires fewer iterations (clock cycles), but at the expense of higher memory resources. Usually, this method is used for software implementations on processors which have access to bigger memory resources.

5. Modular Multiplication

The modular multiplication operation may be decomposed in two parts: a normal multiplication followed by a reduction. In its simple form, modular reduction requires trial division for finding the multiple of the modulus that has to be subtracted from the result and thus it is inherently slow. For this reason, faster alternative algorithms are utilized (Fast-Fourier Transforms, Karatsuba-Ofman algorithm Barret reduction and Quisquater's modification, redundant-digit division, etc.).

6. RNS Modular Multiplication

RNS exhibits several advantages over commonly employed fixed-radix, weighted number

representations, that facilitate fast, parallel implementations of long integer arithmetic. This makes RNS a good candidate for supporting the long multiplications involved in Montogomery method. The Montgomery multiplication algorithm relies on the following two relations, repeated here for convenience:

$$\begin{cases} |t + |t \cdot n'|_r \cdot n|_r = 0, \\ |t + |t \cdot n'|_r \cdot n|_n = t. \end{cases}$$

The rationale behind choosing the value of r is to easily compute the operation modulo *r* in the first aforementioned equation. For weighted, binary number systems, this is achieved by letting *r* be an integer power of the radix 2, but for RNS representation, it is preferable to take *r* as the dynamic range of the RNS base. This way, all numbers less than *r* are already reduced modulo *r*. However, there is a drawback induced by this choice: all numbers greater than *r* need a larger RNS base to be represented. Therefore, costly base extension operations are required in order to compute these additional RNS residues.

The Montgomery algorithm was first adapted to RNS by Posch & Posch in [6]. In what follows, the RNS version of Montgomery multiplication Algorithm 4, is presented according to [1].

Algorithm 4: Montgomery Multiplication for RNS
Input: two RNS bases $B = (m_1, \ldots, m_k)$ and $B' = (mi, \ldots, m'_k)$ with gcd$(M, M') = 1$ a positive integer N represented in both bases such that gcd$(N, M) = 1$ and $0 < 4N < M < M'$ two integers a and b, represented in both bases, with $a \bullet b < M \bullet N$
Output: $r' = |a \bullet b \bullet \mathrm{M}^{-1}|_{\mathrm{N}}$, represented in both bases

1. $t = a \bullet b$ in base $B \cup B'$
2. $q = t \bullet (-N^{-1})$ in base B
3. base extension: q from base $B \rightarrow q'$ in base B'
4. $r' = (t + q' \bullet N) M^{-1}$ in base B'
5. base extension: r' from base $B' \rightarrow r$ in base B

In order to derive the RNS Montgomery multiplication method, we consider two RNS bases B $= (m_1, \ldots, m_k)$ and $B' = (mi, \ldots, m'_k)$ of relatively prime moduli which implies that gcd$(M, M') = 1$, where M and M' are the dynamic ranges of the two bases. Even though it is not mandatory, it is assumed an equal number of elements for both bases, for regularity purposes of the correspondent hardware architecture or software implementation. Next we take $\mathrm{M}=\prod_{i=1}^{k} m_i$ as the Montgomery constant *r*. Therefore the Montgomery method yields
$|a \,.b \,.\, M^{-1}|_{\mathrm{N}}$
where a, b, and the modulus N are positive integers represented in the predefined bases *B* and *B* , as

$(a)_B = (a_i, \ldots, a_k) \qquad (a)_{B'} = (a_i, \ldots, a_k)$

$(b)_B = (bi, \ldots, b_k) \qquad (b)_{B'} = (bi, \ldots, b_k)$

$(N)_B = (Ni, \ldots, Nk) \quad (N)_{B'} = (Ni, \ldots, Nk)$

In step 1. we compute the product $a \cdot b$ in both RNS bases. This can be done in parallel for all moduli, in constant time, according to Equation as follows:
$t=|a \wedge b|_M=(|a_1 \wedge b_1|_{m1} > \ldots > |a_k \wedge b_k|_{mk})$
$t' =|a \wedge b|_{M'}=(|a^1 \wedge b^1|_{m1} > \ldots > |a'_k \wedge b'_k|_{mk})$

In step 2. we have to compute q such that $r = a \cdot b + q \cdot N$ is a multiple of M. The term $a \cdot b + q \cdot N$ represented in base *B*, composed solely of 0, since any multiple of *M*, represented modulo *M* equals 0. Thus, we have
$(r)_B = 0 \quad \Leftrightarrow \quad ri = |ai \cdot bi + qi \cdot Ni|_m = 0$ for $i = 1, \ldots, k$.

The value of q is then given by the solutions of the previous equations
$qi = |a_i \,.\, bi \,.| - N^{-1}|_{mi}|_{mi} \quad$ *for i=1,….,k*

Step3. Since r is a multiple of *M*, as pointed out previously, it is composed only of 0 in base *B*, which further divided by *M* yields 0. Moreover, we have to perform the division by *M*, that reduces in RNS to a multiplication by the multiplicative inverse of *M* which unfortunately does not exist modulo *M*. Thus, we need *a* larger dynamic range to accommodate *r*. Also the new base moduli should be prime to M for the inverse $|\mathrm{M}^{\sim 1}|_{M'}$ to exist.

In step 3, a base extension is performed to obtain q', the RNS representation of *q* in base

Step4. Now we can evaluate $r' = |(t' + q' \bullet N') \bullet |M^{-1}|_m \; |_{M'}$. The term $t' = |a \bullet b|_{M'}$ is already computed from step 1, the multiplicative inverse $|\mathrm{M}^{-1}|_{M'}$ is a pre computed constant, hence we obtain

$r_i = | (t_i + q_i \bullet N_i) \bullet || M^{\sim 1}|_{M'} \,|_{M'} \,|_{M'}$

Step5. In order to be able to perform modular exponentiation by repeating the Montgomery

multiplication, the range of the output should be made compatible with the range of the input. Hence we need to convert the result *r* back to base *B*, which is achieved by applying again a base extension.

We note that if $a \bullet b < M \bullet N$ we have the output in the desired range that is < 2N, as suggested by the following derivation:

$$\frac{a.b+q'.N}{M} < \frac{M.N+M.N}{M} = 2N < M.$$

If only one multiplication is desired, then the condition 2N < M suffices for the algorithm to give the correct result. Otherwise, it is required that 4N < M < M′ in order to reuse the output as input for a subsequent modular multiplication without any preceding reduction

$$(2N)^2 < M \bullet N \quad \Rightarrow \quad 4N < M.$$

There exist several RNS Montgomery multiplication methods in the literature. The main difference between them is the base extension algorithm, which induces different conditions for the range of the input and output values, or necessitates additional operations to counterbalance other introduced effects. In the following chapter, a survey of existing base extension methods is presented.

7. RNS Moduli Choice

The Montgomery method works for RNS basis with moduli of any form. However, if we choose the moduli of a particular form, the overall performance can be improved. The number of moduli and their form both affect the complexity of the algorithm to be performed and the efficiency of the representation. Thus, selecting the proper set of moduli is a major issue. An important remark is that usually, we tend to choose the moduli such that they are comparable in magnitude to the largest one, due to the fact that the computation speed is dictated by the largest modulus. In the following. we present an overview of different possible moduli and highlight their advantages and disadvantages.

7.1 Mersenne numbers

Among these special forms, moduli of the form $m = 2^k - 1$ are of special interest [5][9]. Based on the property $|2^k|_m = 1$ the modulo reduction operation $|a|_m$ is greatly simplified. It requires at most two *k*-bit operations, for $a < m^2$, obtained by writing $a = a_1 \cdot 2^k + a_0$ and then reducing both sides modulo *m* and observing that:

$$|a|_m = \begin{cases} a1 + a2 & if\ a1 + a2 < m. \\ a1 + a2 - m & if\ a1 + a2 \geq m \end{cases}$$

When these moduli are prime numbers, they are called Mersenne numbers. Nevertheless, this approach is rather unpractical. Since there exists only one Mersenne number of *k* bits, choosing as the bases moduli only relatively prime moduli of this form would result in a base with moduli not comparable in magnitude and a dynamic range wider than required.

7.2 Pseudo-Mersenne numbers

Crandall [3] enlarged the class of Mersenne numbers and proposed in 1992 the pseudo-Mersennenumers. The pseudo-Mersenne moduli have the form $m = 2^k - c$, where k ∈ N and $c < 2^{k/2}$. The operation $|a|_m$ costs in this case 2 · *H*(*c*) + 2 *k*-bit additions, where $a < 2^{2k}$ and *H(c)* denotes the Hamming weight of *c*. Thus, it is preferable to choose *c*, such that *H(c)* is minimized.

7.3 Generalized Mersenne numbers

Solinas introduced in 1999, [8] another class of moduli, the Generalized Mersenne numbers. A Generalized Mersenne number has the following polynomial form

$m = f(2^k)$, where $f(X) = X^n - C(X)$ with *C* polynomial of degree $\leq n/2$ and $||C||_\infty = 1$.

In the above definition $||C||_\infty = 1$ means that the coefficients f_i of the polynomial f belong to the set {—1, 0, , 1}. Among these numbers, one particular form that allows smaller RNS base dynamic ranges when compared to Mersenne numbers, while still maintaining the efficiency of the modulo reduction, is $m = 2^{kl} - 2^{k2} - 1$. The operation $|a|_m$, where $a < m^2$, requires at most 6 k_1-bit additions for $0 < k_2 < k$ l+12. Furthermore, the number of additions can be reduced to 4, if *k*2is taken bigger than 1. However, there is a drawback when using these numbers: there exists only one number for a given bit length of the modulus and a fixed *f(X).* This implies that each generalized Mersenne number necessitates a dedicated implementation.

8. Conclusion

This research paper provides an overview of the RSA cryptosystem, emphasizing its computational requirement, the modular exponentiation that is broken into a sequence of modular multiplications

with respect to a large modulus. The binary exponentiation method and the Montgomery multiplication are scrutinized. By combining the efficient modulo reduction operations provided by the Montgomery method with RNS, a powerful arithmetic tool is obtained, which improves the global performance. We have also given guidelines regarding the moduli form and their influence on the overall performance. Then, the bottleneck of the RNS Montgomery multiplication algorithm is identified, most of computational effort is due to the two required base extensions.

References

[1] J.C. Bajard, L.S. Didier, and P. Kornerup, Modular Multiplications and Base Extension in ResidueNumber Systems, Proceedings ARITH15, the 15th IEEE Symposium on Computer Arithmetic, June 2001, pp.59–65.

[2] J. Chung and A. Hassan, More Generalized Mersenne Numbers, Selected Areas in Cryptography SAC 2003, volume 3006 of LNCS, August 2003.

[3] R.Crandall, 1992, Method and Apparatus for Public Key Exchange in a Cryptographic System.U.S. Patent number 5159632.

[4] W. Diffie and M. E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, IT-22(6), November 1976, pp. 644–654.

[5] D. E. Knuth, The art of computer programming: Seminumerical algorithms–volume2, Addison-Wesley, 1981.

[6] Modulo Reduction in Residue Number System, IEEE Transactions on Parallel and Distributed Systems, vol. 6, May 1995, pp. 449–454.

[7] A. Shamir R. L. Rivest and L. Adleman, A Method for Obtaining Digital Signatures and Public-key Cryptosystems, Communications of the ACM (1978), pp. 120 – 126.

[8] J. Solinas, 1999, Generalized Mersenne numbers. Research Report CORR-99-39, Center forApplied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada.

[9] J. C. Bajard, M. Kaihara, and T. Plantard, Selected RNS Bases for Modular Multiplication, 18th IEEE International Symposium on Computer Arithmetic, 2009, pp. 25–32.

Sridevi, at present working as Assistant Professor at karnatak University, Dharwad,Karnataka state. And persuing part time Ph.D at Mangalore University, under the guidance of Dr.Manjaih.D.H. Her research areas are Computer Networks and Mobile ommunication systems and network security.
Email:devisris@yahoo.com

Dr. Manjaiah D.H., is currently Professor and BOS Chairman of the Dept. of Computer Science.,Mangalore University, and Mangalore. He is also the BoE and BoS Member of all Universities of Karnataka and other reputed universities in India. He received PhD degree from University of Mangalore, M.Tech. fromNITK, Surathkal and B.E. from Mysore University. Dr. Manjaiah D.H has more than 15years extensive academic, Industry and Research experience. He has worked at many technical bodies like CSI [AM IND 00002429], ISTE [LM - 24985], ACS, IAENG, WASET, IACSIT and ISOC. He has authored more than - 50 research papers in International / National reputed journals and conferences. He is the recipient of the several talks for his area of interest in many public occasions. He had written Kannada text book, with an entitled, “COMPUTER PARICHAYA”, for the benefits of all teaching and Students Community of Karnataka. Dr. Manjaiah D.H ’s areas interest are Advanced Computer Networking, Mobile / Wireless Communication, Wireless Sensor Networks, Operations Research, E-commerce, Internet Technology and Web Programming. He is the expert committee member of various technical bodies like AICTE, various technical Institutions and Universities in INDIA. He sessions of various International & National conference and reviewer of the Journals. He successfully completed Major Research project on “Design Tool of IPv6 Mobility for 4G Networks”, around Rs.12lakhs worth funded by UGC, New Delhi from year 2009 - 2012. He is recognized as a Ph.D. guide in Computer Science at Mangalore University, Mangalore and currently five students are doing their Ph.D., under the guidance of him. He is recognized as advisory editorial board member of the International Journal of Advanced Computing [IJAC], International Journal of Computer Science and Application [IJCSA], and Journal of Intelligent System Research and Journal of Computing. He visited most of the countries in the World.