Algorithm for modular exponentiation in public key cryptosystems

Plamen Stoianov[1]

Abstract **– The operational speed of most public key cryptosystems is largely determined by the modular exponentiation operation. The required modular exponentiation is computed by a series of modular multiplications. Optimized algorithms are required for various platforms, especially for lower-end platforms. These require the algorithms to be efficient and consume as little resources as possible. This article presents algorithm for calculating modular exponentiation using less precomputation without division. The aim is to improve computational efficiency of modular exponentiation based public key cryptosystems**.

Keywords – **Cryptography, modular reduction, modular exponentiation, long integers.**

I. INTRODUCTION

The word cryptography comes from the Greek words crypto (hidden) and graphy (writing), hence cryptography is the art of secret writing. More formally cryptography is the study of mathematical techniques related to the security services of information security. The ITU-T X.800 standard defines the security services provided by a system to give a specific kind of protection to system resources. The standard divides security services into the following four categories:

- Confidentiality is a service used to keep the content of information accessible to only those authorized to have it. This service includes both of protection of all user data transmitted between two points over a period of time as well as protection of traffic flow from analysis.
- Integrity is a service that requires that computer system assets and transmitted information be capable of modification only by authorized users. Modification includes writing, changing, changing the status, deleting, creating, and the delaying or replaying of transmitted messages. It is important to point out that integrity relates to active attacks and therefore, it is concerned with detection rather than prevention. Moreover, integrity can be provided with or without recovery, the first option being the more attractive alternative.
- Authentication is a service that is concerned with assuring that the origin of a message is correctly identified. That is, information delivered over a channel should be authenticated as to the origin, date of origin, data content, time sent, etc. For these reasons this service is subdivided into two major classes: entity authentication and data origin authentication. Notice that the second class of authentication implicitly provides data integrity. An important part of almost all modern security protocols is public-key algorithms.
- Non-repudiation is a service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. A procedure involving a trusted third party is needed to resolve the dispute.

Specifically, unauthorized access to information must be prevented, privacy must be protected, and the authenticity of electronic documents must be established. Cryptography, or the art and science of keeping messages secure, allows us to solve these problems.

These security services are provided by using cryptographic algorithms. There are two major classes of algorithms in cryptography: Symmetric algorithms and Public-Key algorithms.

Symmetric algorithms are algorithms where the encryption and decryption key is the same, or where the decryption key can easily be calculated from the encryption key. The main function of these algorithms, which are also called secret-key algorithms, is encryption of data, often at high speeds. Private-key algorithms require the sender and the receiver to agree on the key prior to the communication taking place. The security of private-key algorithms rests in the key; divulging the key means that anyone can encrypt and decrypt messages. Therefore, as long as the communication needs to remain secret, the key must remain secret.

There are two types of symmetric-key algorithms: block ciphers and stream ciphers. Block ciphers are a function which maps n-bit plaintext to n-bit ciphertext blocks (n is called the blocklength). The most used secret-key algorithms are DES, 3DES, AES, RC5 etc. Stream ciphers operate on a single bit of plaintext at a time. They are useful because the encryption transformation can change for each symbol of the message being encrypted. They can be used when the data must be processed one symbol at a time because of lack of equipment memory or limited buffering.

One of the major issues with symmetric key systems is the need to find an efficient method to agree on and exchange the secret keys securely. This is known as the key distribution problem.

A major advance in cryptography came in 1976 with the publication by Diffie and Hellman (New Directions of Cryptography) [1] of a new concept of cryptography. This new concept was called public-key cryptography. Public-Key Cryptography (PKC) is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. Pair of matched keys is used, termed “public” and “private”

[1]Plamen Stoianov is with the Electronic Engineering Faculty at Technical University of Varna, Communications Equipment and Technology Department, Studentska 1, Varna , Bulgaria,
E-mail: pl_stoianov@tu-varna.bg.

keys. Anyone that wants to send a message to party A can encrypt that message using A's public key but only A can decrypt the message using her private key. In implementing a public-key cryptosystem, it is understood that A's private key should be kept secret at all times. Furthermore, even though A's public key is publicly available to everyone, including A's adversaries, it is impossible for anyone, except A, to derive the private key.

In general, one can divide practical public-key algorithms into three families [2]:

- Algorithms based on the integer factorization problem: given a positive integer n, find its prime factorization. RSA [3], the most widely used public-key encryption algorithm, is based on the difficulty of solving this problem. RSA problem: given a positive integer *n* that is a two distinct odd primes *p* and *q*, a positive integer *e* such that gcd(*e*,(*p*-1)(*q*-1))=1, and an integer c, find an integer *m* such that $m^e \equiv c \bmod n$.
- Algorithms based on the discrete logarithm problem: given α and β find the integer *x* such that $\alpha^x \equiv \beta \bmod p$. The Diffie-Hellman key exchange protocol is based on this problem: given a prime *p*, a generator α and elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$.
- Algorithms based on Elliptic Curves. Elliptic curve cryptosystems are the most recent family of practical public-key algorithms, but are rapidly gaining acceptance. Due to their reduced processing needs, elliptic curves are especially attractive for embedded applications. Despite the differences between these mathematical problems, all three algorithm families have something in common: they all perform complex operations on very large numbers, typically 1024 bits in length for the RSA and discrete logarithm systems or 160 bits in length for the elliptic curve systems.

II. Overview of Algorithms for Modular Reduction and Exponentiation

The most common operation performed in public-key schemes is modular exponentiation, i.e., the operation $A^E \bmod M$. Performing computation of numbers of this large size (e.g., 2048 bit) with multiple precisions is not easy or fast to implement. Modular exponentiations are typically calculated using repeated square-and-multiply algorithms with modular reductions in between. In [2] this method is called binary exponentiation. A similar algorithm is also used for point multiplication in ECC. The basic idea of binary method is to compute modular exponentiation using the binary expression of exponent *E*. The exponentiation operation is broken into a series of squaring and multiplication operations by the use of the binary method. There are two variations of the algorithm: left to right (LRB) and right to left binary exponentiation (RLB). LRB algorithm computes the exponentiation starting from the most significant bit position of the exponent *E* and proceeding to the right, which is depicted as follows.

Input : integers A, M, $E = (e_n e_{n-1} \ldots e_1 e_0)_2$

Output: $X = A^E \bmod M$

1. $X \leftarrow 1$
2. for i = n to 0 do

   $X \leftarrow X^2 \bmod M$

   If $e_i = 1$, then $X \leftarrow X.A \bmod M$
3. Return (X)

Let n + 1 be the bitlength of the binary representation of E, and let w(e) be the number of 1's in this representation. Algorithm LRB performs t + 1 modular squarings and w(e) − 1 modular multiplications by A. Different from the LRB , the RLB algorithm computes the exponentiation starting from the last significant bit position of the exponent *E* and proceeding to the left. Each multiplication (or squaring) operation requires a large number of clock cycles due to the long operand length depending on the implementation. The binary method is frequently used in smartcards and embedded devices, due to its simplicity and low resource consumption.

Mostly mentioned are various windowing techniques as a generalization of the basic algorithm in which more than one bit of the exponent is processed per iteration. The basic idea is as follows: the exponent is divided into digits (windows). Algorithm LRB can thus be considered as a special case where the window size is equal to 1.

The k-ary method (fixed window) is an optimization of the binary method. Bits of the exponent are scanned in groups as against the binary method in which a bit is scanned per iteration. The algorithm for this technique is shown below.

Input : integers A, M, $E = (e_n e_{n-1} \ldots e_1 e_0)_b$ where

$b = 2^k$ for $k \geq 1$

Output: $X = A^E \bmod M$

1. precomputation

   1.1 $a_0 \leftarrow 1$

   1.2 for i = 1 to 2^k -1 do : $a_i \leftarrow a_{i-1}.A \bmod M$
2. $X \leftarrow 1$
3. for i = n down to 0 do

   3.1 $X \leftarrow X^{2^k} \bmod M$

   3.2 $X \leftarrow X.a_{e_i} \bmod M$
4. return (X)

Most methods rely on modular reduction algorithm functions to reduce the size and complexity of the required arithmetic operations to carry out their public-key cryptosystem implementations more efficiently [4] [5].

The Classical, Barrett, and Montgomery algorithms are well-known modular reduction algorithms for large integers used in public-key cryptosystems

Montgomery Reduction can be implemented in two ways: word-serial and bit-serial. For a software implementation, the bit-serial algorithm becomes too slow because the processor is built on word-level arithmetic. Therefore, software implementations typically utilize the word-level Montgomery Reduction algorithm. If we assume a word-level length of n,

to reduce a 2n-bit number to an n-bit number, 2 full multiplications and 2 full addition operations are required. Thus, a full modular multiplication requires 3 multiplication and 2 addition operations [2]. This also applies to large digit approaches such as ours, where the multiplication/addition operations on large digits are further decomposed into word-size operations. The approach of Montgomery avoids the time consuming trial division that is the common bottleneck of other algorithms. His method is proven to be very efficient and is the basis of many implementations of modular multiplication in hardware as well as software such as high-radix design [6][7], scalable design [8], parallel calculation quotient and partial result and signed-digit recoding [9].

The notation is as follows: MONT(X, Y) = XY R^{-1} mod N

For a word base b = 2^a , R is usually chosen such that R = 2^r = $(2^a)^l$ > N.

To compute Z = xyRmodN, one first has to compute the Montgomery function of x and R^2 modM to get Z' = xR modM. Mont(Z', y) gives the desired result. When computing the Montgomery product T = XY R^{-1} modM, the following procedure was proposed:

INPUT: Integers M(odd), x ⊂ [0,M − 1], y ⊂ [0,M − 1],

R = 2^r ,

and M' = $-M^{-1}$ mod 2^r

OUTPUT: xyR^{-1} modM

1. T ← 0
2. For i from 0 to (l -1) do:

   2.1 m_i ← $(t_0 + x_i y)M'$ mod 2^a

   2.2 T ← $(T + + x_i y + m_i M)/ 2^a$
3. If T ≥ M, then T ← T − M
4. Return (T)

An architecture based on Montgomery's algorithm[10] is probably the best studied architecture in hardware. Differences appeared because of a different approach for avoiding long carry chains.

The Barrett reduction [11] requires the pre-computation of one parameter, μ = $\left\lfloor \frac{b^{2k}}{M} \right\rfloor$, where M is the modulus of the multiplication operation. Since this is a parameter that only depends on the modulus, it remains unchanged throughout the entire exponentiation operation, thus the calculation time of this parameter is not significant. If the number to be reduced is N, the reduction then takes the form Q= $\lfloor (A/b^{2k-l}).\mu / b^l \rfloor$ by integer division which requires two n-bit multiplies and one n-bit subtract, leaving the total at three multiplications and one subtraction. In [12], the authors proposed a method called folding to reduce the amount of computations for a Barrett's reduction scheme.Their method relies on the precomputation of the constant M' = 2^{3s} mod M.

III. PROPOSED ALGORITHM

The algorithm proposed related to computing A^E mod M uses a combination between sliding windows exponentiation and an improvement of the reduction method for moduli of special form b^n – c [2][13]. By reduction method for moduli of special form the time of execution depends on the value of the radix. Before involution the radix is being checked first. If A>M/2, the modular operation calculates by

$$(M-A)^l \bmod M.\ (M-A)^2 \bmod M =$$
$$= M^2 \bmod M - 2M.A \bmod M + A^2 \bmod M =$$
$$A^2 \bmod M \qquad (1)$$

Equation (1) is valid for all the even powers

$$(M-A)^{2n} \bmod M = A^{2n} \bmod M \text{ for } n \geq 1.$$

By odd powers a correction

$$(M-A)^{2n+1} \bmod M = M - A^{2n+1} \bmod M \qquad (2)$$

(1) and (2) could be used by modular multiplication of two integers : AB mod M.

By A>M/2 and B>M/2 (M-A)(M-B) mod M = AB mod M.

By A>M/2 and B<M/2 (M-A) B mod M = M-AB mod M.

The modular squaring algorithm is described in Algorithm 1.

Algorithm 1 . EXPMOD(A,M)

Input : Integers A, M=($m_{n-1} \ldots m_1 m_0$), m_{n-1} =1

Output: Y=A^2 mod M

1. if A>M/2 then A ← M-A
2. P ← 2^n -M, Y ← A^2
3. while Q>0 do

   Q ← $\lfloor Y / 2^n \rfloor$

   Y ← Q.P + Y mod 2^n
4. if Y ≥ M then X ← Y-M
5. Return (Y)

The modular multiplication algorithm is presented in Algorithm 2.

Algorithm 2 . MULMOD(A,B,M)

Input : Integers A,B, M=($m_{n-1} \ldots m_1 m_0$), m_{n-1} =1

Output: Y=ABmod M

1. 0 ← j if A>M/2 then A ← M-A, j ← j+1
   if B>M/2 then A ← M-B, j ← j+1
2. P ← 2^n -M, Y ← A.B
3. while Q>0 do
   a. Q ← $\lfloor Y / 2^n \rfloor$
   b. X ← Q.P + Y mod 2^n
4. if Y ≥ M then Y ← Y-M
5. if j=1 them X ← M-Y
6. Return (Y)

For the sliding window algorithm the window size may be of variable length and hence the partitioning may be performed so that the number of zero-windows is as large as possible, thus reducing the number of modular multiplication

necessary in the squaring and multiplication phases. Furthermore, as all possible partitions have to start (i.e. in the right side) with digit 1, the pre-processing step needs to be performed for odd values only.

Algorithm 3. Sliding window with EXPMOD and MULMOD

Input: Integers A, M, $E=(e_t e_{t-1}...e_1 e_0)_2$, $k \geq 1$

k is called window size

Output: $X=A^E \bmod M$

1. precomputation : compute and store A_i

$A_1 \leftarrow A$, EXPMOD(A,M), $A_2 \leftarrow Y$

for i=1 to 2^{k-1}-1 do MULMOD(A_{2i-1},A_2,M), $A_{2i+1} \leftarrow Y$

for i = 0 to p, decompose E into zero and nonzero windows f_i of length $L(f_i) \leq k$

2. $X \leftarrow A_{f_p}$
3. for i = p-1 down to 0 do

for j=1 to $L(f_i)$ do EXPMOD(X,M), $X \leftarrow Y$; $X^{2^{L(f_i)}}$

if $f_i \neq 0$ then MULMOD(X,A_{f_i},M), $X \leftarrow Y$

4. Return (X)

IV. Conclusion

Modular exponentiation is the main operation to RSA-based public-key cryptosystems. It is performed using uccessive modular multiplications. This operation is time consuming for large operands, which is always the case in cryptography. For software or hardware fast cryptosystems, one needs thus reducing the total number of modular multiplications required. The proposed algorithm for modular exponentiation is effective by transmission of short messages. It is faster then the classical algorithm why because it does not use integer division. The check in step 1 of EXPMOD and MULMOD reduces the execution time, because always A<M/2. The execution time for step 3 is less, as smaller is the value of $\lfloor X / 2^n \rfloor$. With multiplicity of the modulus different than 8 is selected n=8k and step 2 is being executed while P<M. This permits canceling of rotation within steps 3.1 и 3.2 and operating with bytes only.

References

[1] W. Diffie, M. Hellman, “New direction in cryptography”, IEEE Trans., Inform. Theory IT-22, pp. 644-654, Nov.1976.

[2] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography”, CRC Press, first ed. 1997.

[3] R. Rivest, A. Shamir and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM, 21(2), pp. 120-126, 1978.

[4] C. Koc, “Analysis of Sliding Window Techniques for Exponentation”, Computers and Mathematics with Applications, 30(10), pp.17-24, 1995.

[5] B. Moller, “Improved Techniques for Fast Exponentiation”, Information Security and Cryptology-ICIST 2002, Springer-Verlag LNCS 2587, pp.298-312, 2003.

[6] N. Pinckney, D. Harris, “Parralelized radix-4 scalable Montgomery multipliers”, Journal of Integrated Circuits and Systems, vol.3, no.1, pp.39-45, 2008.

[7] L. Tawalbeh, A. Tenca and C. Koc, “A radix-4 scalable design”, IEEE Potentials, vol.24, pp.16-18, 2005.

[8] A. Tenca, C. Koc, “A scalable architecture for modular multiplication based on Montgomery’s algorithm, IEEE Trans. On computer, vol.52, no.9, pp.1215-1221, 2003.

[9] N. Pinckney, P. Amberg and D. Harris, “Parallelized Booth-encoded radix-4 Montgomery multipliers”, proceeding of 16th IFIP/IEEE International Conference on Very Large Scale Integration, 2008.

[10] P. Montgomery, “Modular multiplication without trial division”, Mathematics of Computation, vol.44, pp.519-521, 1985.

[11] P. Barrett, “Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor”, Advances in Cryptology- CRYPTO’86, pp.313-323, 1987.

[12] W. Hasenplaugh, G. Gaubatz and V. Gopal, “Fast Modular Reduction”, 18th IEEE Symposium on Computer Arithmetic (ARITH’07), pp.225-229, 2007.

[13] П.Стоянов, В. Димов, “Модулна редукция за криптографски алгоритми без предварителни изчисления”, Научни трудове на Русенски университет, том 47, серия 3.2, стр.80-83, 2008.