Accelerated Modular Multiplication Algorithm of Large Word Length Numbers with a Fixed Module

Nikolaos Bardis[1], Athanasios Drigas[1], Alexander Markovskyy[2], and John Vrettaros[1]

[1] National Centre for Scientific Research "Demokritos",
DTE/YE Applied Technologies Department & Net Media Lab,
Ag.Paraskevi - Athens, 15310, Greece
bardis@ieee.org, dr@imm.demokritos.gr, jvr@imm.demokritos.gr
[2] National Technical University of Ukraine,
Department of Computer Engineering, 37, Peremohy,
pr. Kiev 252056, KPI 2003, Ukraine
markovskyy@rgcds.org

Abstract. A new algorithm is proposed for the software implementation of modular multiplication, which uses pre-computations with a constant module. The developed modular multiplication algorithm provides high performance in comparison with the already known algorithms, and is oriented at the variable value of the module, especially with the software implementation on micro controllers and smart cards with a small number of bits.

Keywords: Modular multiplication, Montgomery, Barrett, embedded cryptographic software, efficient implementations.

1 Introduction

The operations of modular multiplication and modular exponentiation are the computational basis of an important category of the contemporary information security algorithms, which are based on the number theory [8]. In particular, the computational implementation of such important operations for the contemporary technologies where information security mechanisms are provided (Public Key Algorithms for encryption (RSA, ECC), Diffie-Hellman key exchange algorithm, Digital Signature Algorithms and Digital Signature Standard), are based on the operations of modular involution and modular multiplication. In order to provide the acceptable security level concerning attacks, the aforementioned algorithms utilize numbers with word lengths of thousands of bits. Thus, for the majority of applications, information security under contemporary conditions requires that the length of the numbers utilized in the algorithms (based on elliptical curves (ECC)), should range from 128 to 256 bits, while the length of the numbers utilized in the algorithms (based on exponential transformation) should range from 1024 to 2048 bits [5].

Software implementation of modular multiplication for such long bit numbers on general-purpose processors or micro-controllers with a fixed number of bits (from 8 to 64) is very time consuming. Accordingly, the software implementation of information security, which is based on public key algorithms, on the universal processors, is

M.D. Lytras et al. (Eds.): WSKS 2010, Part II, CCIS 112, pp. 497–505, 2010.

carried out several orders slower in comparison with the symmetric algorithms (such as DES or AES), with the condition of ensuring adjacent levels of security attacks. The problem of the speed performance of the modular arithmetic operations of large numbers during the cryptographic algorithms implementation, is especially grave for low-length bit micro-controllers [2].

The basic operation of the modular arithmetic utilized in algorithms of that class, is the modular involution, i.e., the calculation $A^K \ mod \ M$. In this case, the majority of the cases of modular involution are carried out by the method of "squaring and multiplications" [1], which uses a number of multiplications close to the theoretical minimum. Based on this, in order to increase the performance of the algorithms implementation by software, it is important to decrease the time needed for the modular multiplication.

Consequently, searching for possibilities to increase the performance of modular multiplications with the use of software implementation on the general purpose processors and micro-controllers, is important and vital.

2 Principal Notations and Effectiveness Estimation Model of the Modular Multiplication Algorithms

The basic operation of the modular arithmetic, utilized in the information security algorithms is modular multiplication, i.e., the calculation $R=A \cdot B \ mod \ M$.

It is assumed that the result R, coefficient A, multiplicand B and module M are n-bits binary numbers, and that the high-order bit of the module is equal to one: $2^{n-1} \le M < 2^n$, and that the co-factors are lower than the module: A<M, B<M.

It is also assumed that the operation of modular multiplication is performed on the k- bits general purpose processors, microprocessors or micro-controllers.

Accordingly, each of the numbers, which participate in the operation of modular multiplication can be represented in the form of s = n/k - bits words:

$$A = \sum_{j=0}^{s-1} a_j \cdot 2^{j \cdot k}, B = \sum_{j=0}^{s-1} b_j \cdot 2^{j \cdot k}, M = \sum_{j=0}^{s-1} m_j \cdot 2^{j \cdot k} \quad (1)$$

where a_j, b_j, m_j – k-bits word, $j \in \{0,\ldots,s-1\}$.

In contrast to the classical modular multiplication algorithm [1], contemporary algorithms [3], [6], do not use the operation of division, which is ineffectively realized on the general purpose processors. Based on this, as a criterion of the productivity evaluation of the software implementation of modular multiplication algorithms, the total operation time of multiplication and addition, which are the basic operations of the contemporary algorithms of modular multiplication, is usually examined [5].

It is assumed that the result of the multiplication command of two k- bits numbers is a 2k - bits representation. By denoting:

q_m – as the number of the required multiplication commands

t_m – as the execution time needed for each command

q_a – as the number of additional commands

t_a - as the execution time of each additional command the estimation time of modular multiplication calculation of n- bits numbers, which is acceptable for the comparative accuracy analysis is: $q_m \cdot t_m + q_a \cdot t_a$.

If the execution time of the multiplication and addition commands on the processor is $w = t_{mul}/t_a$,, then the execution time of the modular multiplication can be represented as $t_a \cdot (w \cdot q_m + q_a) \cdot$

3 Brief Analysis of the Contemporary State of the Acceleration Problem: The Software Implementation of the Modular Multiplication

The classical algorithm for the modular multiplication software implementation [1] without detailing the method of the Reduce(X) procedure execution (which returns the modular reduction of X), is described as follows in the C++ language, in Algorithm 1.

Algorithm 1. Classical scheme of word-by-word Modular Multiplication

```
R=0;
for(i=0; i<s; i++)
{
   Y=0;
   for (j=0; j<s; j++)
   Y+= (a_i*b_j)<<(j*k);
   R += Reduce(Y);
   if (i<s–1)
   {
    B<<=k;
    Reduce(B);
   }
}
Reduce(R);
```

The operation of multiplication is performed word-by-word: each j -th (j=0,…, s-1) the k- bits word of coefficient a_j is multiplied by shifting each of s words of multiplicand in B. The obtained product 2k- bits are added, forming (n+k) - bits, which is a partial representation of the product:

$$a_j \cdot B = \sum_{i=0}^{s-1} a_j \cdot b_i \cdot 2^{i \cdot k} \tag{2}$$

Following this, the modular reduction of the partial expression is carried out, obtaining j-th partial residual $R_j = a_j \cdot B \bmod M$. The result of the modular multiplication $R=A \cdot B \bmod M$ is formed as the sum of the modular reductions of the partial expression of the product: $R=(R_0+R_1+\ldots+R_{s-1}) \bmod M$.

The classical modular reduction algorithm is achieved with the use of the operation of the integer division of 2k bits divisible to the k- bits divider, obtaining a quotient and a residual.

Since the division of the n- bits numbers on the k- bits processor ($n>>k$) is carried out very ineffectively, the calculation of the reduction in the classical algorithm requires $s(s+2\triangleright 5)$ operations of multiplication and $s$operations of the integer division [3].

At present, various algorithms are proposed [2], [3], [5], which increase the performance of the software implementation of the modular multiplication operation. The largest part of the aforementioned algorithms realize the increase in the performance of modular multiplication due to the acceleration of modular reduction by the exception of the operation of integer division, which is used in the classical algorithm [1].

Nowadays, the most effective method of modular multiplication is the Montgomery algorithm [6], which is well adjusted to the architecture of universal processors. The Montgomery algorithm substitutes the operation of division into the random module M by the divisions into power of 2, which effectively are realized by SHIFTS. The operation of modular reduction in Montgomery's algorithm requires s(s+1) operations of multiplication.

The general computational complexity of the implementation of the Montgomery modular multiplication algorithm on a k - bits processor is determined by $2s^2+s$ operations of multiplication and by $4s^2+4s+2$ additions. Accordingly, the calculation time T_M of Montgomery's algorithm on the k- bits processor can be calculated approximately as follows:

$$T_M = (2\cdot s^2 + s)\cdot t_m + (4\cdot s^2 + 4\cdot s + 2)\cdot t_a = t_a\cdot(s^2\cdot(2\cdot w+4) + s\cdot(w+4) + 2) \qquad (3)$$

Known algorithms assume that each calculation of modular multiplication is produced with the new values of co-factors A, B and of the module M. However, the analysis of the practical application of information security algorithms, which use modular multiplication, shows that both their keys, and respectively the module change relatively rarely. This offers the potential possibilities of further decrease of the computational complexity of modular multiplication by simplification in the reduction. The practical implementation of such possibilities requires special research and development. On this basis new modular multiplication algorithms should be developed, which will contain a constant module.

The purpose of this work is the development of an effective modular multiplication algorithm of large numbers on general purpose processors with a constant module.

4 Analysis of the Possibilities of Accelerating the Modular Multiplication in the Information Security Systems

The information security algorithms are based on cryptographic properties. This particular cryptographic property has to do with the non solution using an analytical method of "number theory" tasks. These algorithms require the special complex procedures of the generation of keys.

In particular, the widely used (in practice) algorithm RSA [5] uses a complex procedure to obtain the three numbers d, e and M, of length n from 1024 to 2048 bits, which satisfy the identity $A^{de}\equiv A$. The process of the coding of the block A of a certain message consists of the calculation of $C=A^e \bmod M$, and the decoding of block A is realized with the calculation of $A=C^d \bmod M$. The pair of numbers $<d,M>$ composes the public key, while the pair $<e,M>$ composes the private key.

One of the aforementioned keys depending on the protocol that the RSA uses is public, while the other is private. The analysis of the practical use of an RSA algorithm shows that the keys change relatively rarely so that with the use of the same key, tens of thousands of information blocks are processed. This makes it possible to consider that in the process of computational implementation, the RSA key and consequently the module are both in effect constant. Analogous reasonings can also be applied to a number of other, standardized and widely utilized in practice information security algorithms and in particular to the Digital Signature Standard algorithm [5].

The constancy of the module M makes it possible to simplify the calculation of modular reduction in the multiplication process due to the use of precomputational results. Such pre-computations depend only on the value of the module M and therefore, they are carried out once with a change in the module. The results of the pre-computations remain in the tabular memory and are used repeatedly with each modular multiplication calculation.

In the modular multiplication implementation, the part of the computational resources is strictly used for the calculation of multiplication and the other part for the modular reduction implementation.

In different modular multiplication algorithms [6], [7] the specific weight of expenditures for these two procedures varies. Table 1 gives the quantity of the multiplication operations and the word divisions, which are utilized in the most known modular multiplication algorithms for the calculation of the product A·B and the modular reduction implementation [3].

It is obvious that the possibilities of decreasing the number of operations for the calculation of the product A·B due to the pre-computations with a constant module, are completely limited, since the module itself is not used directly in such calculations.

Therefore, the basic reserve for increasing the speed of the software implementation of modular multiplication, is the use of pre-computations for decreasing the computational complexity of modular reduction.

Data analysis, given in Table I shows that with the use of pre-computations, the greatest effect of the decrease in the implementation time of modular multiplication, is achieved. This takes place due to the reduction of the time expenditures for the modular reduction.

Table 1. Quantity of operations of multiplication

Algorithm	Quantity multiplications k–bits word for calculation A·B	Quantity multiplications k–bits word for modular reduction	Quantity divisions word for modular reduction
Classical	s^2	$s^2+2.5\cdot s$	s
Barrett	s^2	$s^2+4\cdot s$	0
Montgomery	s^2	s^2+s	0

Quantity of multiplication operations and division above the k- bits words, utilized by different modular multiplication algorithms and calculation of the multiplication of the expression $A \cdot B$ and the residual of module M.

5 Modular Multiplication Organization Based on Pre-computations with the Fixed Module

In the classical algorithm (Algorithm 1) the modular reduction procedure, Reduce(X), is carried out from the partial products $a_j \cdot B$ and by shifting the code of multiplicand B by k bits to the left. In both cases, the length of the reduced number X is not more than $(s+1)\,k$- - bits words or more than $(s+1)\cdot k=(n+k)$ bits $x_0, x_1, x_2,\ldots,x_{n+k-1}$:

$$X = \sum_{j=0}^{n+k-1} x_j \cdot 2^j, x_j \in \{0,1\} \tag{4}$$

The number X can be represented in the form of the sum of two components: (n−1)- bits number X'', which coincides with (n−1) low-order digits X and (n+k)- bits number X', which consists of (k+1) high-order digits, coinciding with the similar bits of $X$and $(n-1)$ low-order digits, equal to zero:

$$X = \sum_{j=0}^{n+k-1} x_j \cdot 2^j = X'+X'', X' = \sum_{j=n-1}^{n+k-1} x_j \cdot 2^j, X'' = \sum_{i=0}^{n-2} x_i \cdot 2^i \tag{5}$$

In accordance with the property of congruence for the modular reduction, the residual $X \bmod M$ can be represented in the form of the modular reduction as the sum of the residuals of the components X composing X' and X'':

$$X \bmod M = (\sum_{j=0}^{n+k-1} x_j \cdot 2^j) \bmod M = (X'+X'') \bmod M = \tag{6}$$
$$= (X' \bmod M + X'' \bmod M) \bmod M$$

Since the high order digits, (n−1)- bits of module M are equal to one, and the X'' is equal to (n−1)- bits number, then $X''<M$ and, accordingly $X'' \bmod M = X''$. Number X' contains only k+1 significant digits. The rest n−1 low-order digits are equal to zero. Consequently, X' and accordingly $X' \bmod M$ assume only 2^{k+1} different values. All possible n- bits values of $X' \bmod M$ for the appropriate X', can be pre-computed and stored in the memory as tables.

If we designate through Z the binary code, which consists of (k+1) high order significant digits X':

$$Z = \sum_{j=n-1}^{n+k-1} x_j \cdot 2^{j-n+1}$$

and with T(Z) – n- bits code of the tabular value $T(Z) = X' \bmod M$, then the modular reduction procedure Reduce(X) is realized in accordance with the following expression:

$$\mathrm{Re}duce(X) = X \bmod M = (T(Z) + X'') \bmod M \tag{7}$$

In this case, the computational complexity of the modular reduction implementation is determined by maximum two operations of addition between (n+k)- bits numbers: the first for the calculation of $T(Z)+X''$ and the second for executing the subtraction $(T(Z)+X'')-M$, if $T(Z)+X'' \geq M$.

Since $0 \leq T(Z)+X'' < 2 \cdot M$, for the modular reduction $(T(Z)+X'') \bmod M$ not more than one subtraction of n- bits numbers, is required. Therefore, the execution time of the

procedure will not exceed $2\cdot(s+1)\cdot t_a$, with the average value of $1.5\cdot s\cdot t_a$. The storage memory, which is required for storing all the pre-computed possible values of T(Z) comprises $2^{k+1}\cdot n$ bits or $2^{k+1}\cdot s$ of k- bits words.

The proposed approach is especially effective in the implementation of modular multiplication on the low-bits microprocessors, micro-controllers and smart cards. In this case, the memory size for storing the results of pre-computations with a constant module, proves to be completely acceptable for the majority of applications.

For example, for the accelerated multiplication implementation of the 1024- bits numbers on the 8- bits micro-controller, the capacity of the required storage memory will compose of $(2^9\cdot 128)=2^{16}$ of bytes (64 Kbyte).

For the accelerated modular multiplication implementation on the 16-bits processor r, the above capacity requires storage memory, which substantially grows and therefore decreases the effectiveness of the application of pre-computations.

In order to decrease the capacity of the memory required for storing the results of pre-computations T(Z), its multi-section organization, is proposed.

The essence of the proposed tables organization of pre-computations lies in the fact that the value X' is divided into q components:

$$X' = X'_1 + X'_2 + \ldots + X'_q$$

with lengths $n+r_1$, $n+r_1+r_2$, …, $n+r_1+\ldots+r_q$, since $r_1+r_2+\ldots+r_q=k+1$.

Each i- th constituting $X'_i\ (i=1,\ldots,q)$ contains ri high order significant digits, which coincide with the digits x_{n+h}, x_{n+h+1}, . . . , x_{n+h+ri} of number X (h=0 for i=1 and h = r_1 + . . . + r_{i-1} for i>1), and the rest of the low-order digits are equal to zero:

$$X'_i = \sum_{h=g_i}^{g_i+r_i} x_{n+h}\cdot 2^{n+h},\ g_i = \sum_{t=1}^{i-1} r_t,\ \forall i \in \{2,\ldots,q\},\ g_1 = 0 \tag{8}$$

Following this, in accordance with the property of congruence the modular reduction X mod M can be represented in the form:

$$X \bmod M = (X'_1 \bmod M + X'_2 \bmod M + \ldots + X'_q \bmod M + X'') \bmod M$$

In order to determine each of the values of $X'_i\ mod\ M$, the use of the precomputations results is proposed, where the results are previously calculated for all possible codes X'_i. Since a quantity of significant (non zero) bits in code X'_i is equal to r_i , then the number of different values X'_i will comprise 2^{r_i} , and the memory capacity for storing all possible values of $X'_i\ mod\ M$ respectively will be $(2^{r_i}\cdot n)$ bit.

If we denote through Z_i the binary r_i - bits code, which contains only r_i high order significant digits X'_i, then,

$$Z_i = \sum_{h=g_i}^{g_i+r_i} x_{n+h}\cdot 2^{h-g_i},\ g_i = \sum_{t=1}^{i-1} r_t,\ \forall i \in \{2,\ldots,q\},\ g_1 = 0 \tag{9}$$

If we denote through $T_i(Z_i)$ – n bits code of the tabular value $T_i(Z_i) = X'_i\ mod\ M$, then the modular reduction procedure is realized in accordance with the following expression:

$$X \bmod M = (\sum_{i=1}^{q} T_i(Z_i) + X'') \bmod M \tag{10}$$

The total volume of the tabular memory for storing $T_1(Z_1),T_2(Z_2),\ldots,T_q(Z_q))$ comprises $n \cdot \sum_{i=1}^{q} 2^{r_i}$ bits. The aforementioned example of storing T(Z) in one table, can be examined as a special case of the results organization within subdivided tables, with pre-computations for q=1.

The use of multi-section tables makes it possible to substantially decrease the memory capacity of their storage. For example, under the conditions of the example given above, for the accelerated multiplication implementation of 1024- bits numbers on the 8- bits micro-controller with the two-section memory (q=2, r_1 =5, r_2=4=2), the required memory capacity will compose of $1024 \cdot (2^5+2^4) = 10^{10} \cdot 48$ bits or 6144 bytes or 10.67 times less than during the single-section organization of tabular memory.

From another point of view, the use of multi-section organization of the tabular memory is combined with the increase of the execution time of the modular reduction. The calculation of the sum of expression (10) requires q(s+1) operations of summing up k- bits words.

The number of significant digits of the sum code will not exceed in this case n+q, so that, if $r_1 \geq q+1$, then for executing the modular reduction of sum with the use of the first table $T_1(Z_1)$, 1.5(s+1) addition operations are required, on average. The total number of the additional operations is: $(q+1.5) \cdot (s+1)$.

6 Conclusions

The problem of increasing the performance of the modular multiplication software implementation, which is the basic computational operation used in a wide circle of information security algorithms was researched. It is shown that during the practical application of information security algorithms, based on the analytically insoluble tasks of the "number theory", the keys and consequently the module, change relatively rarely.

Based on the conducted research, a new algorithm was proposed for the modular multiplication that differs from the classical organization of the modular reduction execution. Reduction in the computational complexity of the software implementation is achieved by the use of pre-computations results, which depend only on the module and which are stored in the tabular memory. The performance estimation of the proposed algorithm and memory use for storing the pre-computations tables are theoretically substantiated.

The executed analysis showed that the speed of the software implementation of modular multiplication on the micro-controllers with the use of the proposed algorithm grows 1.5-2, in comparison with the most effective algorithm today, the Montgomery algorithm.

References

1. Bosselaers, A., Govaerts, R., Vandewalle, J.: Comparison of three modular reduction functions. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 175–186. Springer, Heidelberg (1994)

2. Dhem, J.-F., Quisquater, J.-J.: Resent results on modular multiplications for smart cards. In: Schneier, B., Quisquater, J.-J. (eds.) CARDIS 1998. LNCS, vol. 1820, pp. 350–366. Springer, Heidelberg (2000)
3. Hars, L.: Long Modular multiplication for Cryptographic Applications. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 45–61. Springer, Heidelberg (2004)
4. Hong, S.M., Oh, S.Y., Yoon, H.: New modular multiplication algorithms for fast modular exponentiation. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 166–177. Springer, Heidelberg (1996)
5. Menezer, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
6. Montgommery, P.L.: Modular multiplication without trial division. // Mathematics of Computation 44, 519–521 (1985)
7. Barrett, P.: Implementing the River Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987)
8. Cohen, H.: A Course in Computational Algebraic Number Theory, 2nd edn. Graduate Texts in Mathematics. Springer, Heidelberg (1995)