Reduction of correction in Barrett's algorithm

P. Stoianov[1] R. Dimova[2]

Abstract—**Modular operations are the basis for the most commonly used asymmetrical cryptographic algorithms. Data processing speed depends mainly on the high-precision operations – exponentiation operation and modulus. Performance time is of great importance for SMART cards where microcontrollers of lower clock frequency and of less computational capacity are used. Montgomery's and Barrett's algorithms are implemented for modular reduction. The paper presents Barrett's algorithm modified in terms of change of the size of the calculations depending on the value of the modulus. This leads to a reduction in the error resulting from rounding and to a reduction in the number of correction operations to 1.**

Keywords - **algorithm, Barrett, modification**

I. Introduction

Cryptographic algorithms are based on a certain mathematical function for ciphering (encryption) and deciphering (decryption). A code known to the communicating parties using it or to the public (public key) is used in ciphering. Cryptographic algorithms are called "symmetrical" when the codes for ciphering and deciphering are identical. Most commonly block ciphers (known as Feietel ciphers) are implemented, e.g. Triple DES, IDEA [1]. When the codes are different, the algorithms are "asymmetrical" referred to as "public-key algorithms". The advantage of asymmetrical algorithms is that they do not require a secure channel for session key exchange. As the processing time in asymmetrical algorithms is considerably greater in comparison to symmetrical ones [6], public-key algorithms are used for secret codes and symmetrical ones are implemented for data communication.

The most popular asymmetrical algorithm RSA [2] was created in 1977. As in other asymmetrical algorithms (e.g. Diffie-Hellman for code exchange, ELGamal, etc), in RSA the main operation is of the A^E mod M type where A is the radix, E is the power and M is the modulus. For correct deciphering it is necessary that A<M. For RSA [2], it was initially recommended that the digit capacity of M was not greater than 200 decimal digits.

At present, owing to the increased computational capabilities of cryptoanalysis, a code with a digit capacity of up to 2048 bits (more than 600 decimal digits) is used.

For faster computation of A^E mod M algorithms for multi-digit multiplication (Karatsuba – Ofman [7], Comba) and algorithms for exponentiation (K-ary, Sliding Window [8], etc) are implemented. Due to the availability of hardware multipliers, single multi-digit multiplication takes less time than division by an arbitrary divisor. An effective way of reducing the time for multi-digit integer division is by substituting the modulus with one divisible by the digital word (2^n at $n > 2$). In this case division is substituted only with the operation of shifting and rotation.

Since (R,N) = 1 is a necessary condition and R is chosen to be 2^n, the algorithm cannot be used with an even number. A way of solving the problem is by decomposition of the modulus and by the use of the Chinese remainder theorem [4].

With Barrett's algorithm a pre-calculation of b^{2n}/m is performed, where n is the digit capacity of the modulus. First, the integer part is calculated and then the remainder. A disadvantage of this algorithm is that in some cases the exact integer part cannot be obtained. Correction in the result is necessary by maximum two comparisons and two subtractions causing an increase in the processing time.

It is worth noting that these algorithms are effective in performing multiple operations with an arbitrary radix and identical modulus, all this being typical of asymmetrical cryptographic algorithms. With single operations they are relatively slower than traditional division because of the pre-calculations. A more detailed analysis and comparison of three algorithms, i.e. classical, Montgomery's and Barrett's, is suggested by A. Bosselaers et al. [10].

II. OVERVIEW OF MODULAR REDUCTION ALGORITHMS

The modular reduction operation, *a* mod *m*, is conventionally accomplished by dividing *a* by *m* to obtain the remainder. The steps of the division algorithm can be modified in order to speed up the process. Reducing the time and memory complexities of this operation is a problem on which relies the practical feasibility of the cryptosystem's signature and encryption methods.

The classical algorithm for modular reduction is a formalization of the ordinary pencil-and-paper method for finding the quotient and the remainder. Each step of this algorithm consists of estimating one digit quotient *q*, using the most significant digits of *z* and *n*, subtracting *qn* from *z* and correcting the resulting error. Thus the core of this algorithm is to estimate the quotient *q* as accurately as possible.

When $n_{k-1} \geq \left\lfloor \frac{b}{2} \right\rfloor^3$ we can see that the estimation of *q* by dividing the three most significant digits of *z* by the two most

[1] Plamen Stojanov - Technical University of Varna, Telecommunications Department, Studentska 1., Varna, Bulgaria, Email: pl63@abv.bg
[2] Rozalina Dimova - Technical University of Varna, Telecommunications Department, Studentska 1,Varna, Bulgaria, rdim@abv.bg

significant digits of n results in at most one error. This error occurs with approximate probability $\frac{2}{b}$

There are several variations of the classical algorithm with slightly different ways of quotient estimation [11].

Barret presented a method for estimating the whole quotient q $= \left\lfloor \frac{z}{n} \right\rfloor$ at a time.

Let $\mu = \left\lfloor \frac{b^{2k}}{n} \right\rfloor$, which can be recomputed for a given n.

Then the quotient can be estimated as

$\hat{q} = \left\lfloor \frac{u}{b^{k+1}} \right\rfloor$ with $u = \mu \left\lfloor \frac{z}{b^{k-1}} \right\rfloor = \mu z[l-1:k-1]$,

which can be viewed as an integer counterpart of the floating point division $q = \left\lfloor \frac{z}{b^{k-1}} \frac{b^{2k}}{n} \frac{1}{b^{k+1}} \right\rfloor$.

The estimate $\hat{q}$ is at most two smaller than the correct q. This can be seen from the inequalities

$$\begin{aligned} \frac{z}{n} \geq \hat{q} &> \frac{1}{b^{k+1}}\left(\frac{z}{b^{k-1}}-1\right)\left(\frac{b^{2k}}{n}-1\right)-1 \\ &= \frac{z}{n} - \frac{z}{b^{2k}} - \frac{b^{k-1}}{n} + \frac{1}{b^{k+1}} - 1 \\ &\geq q - \left(\frac{z}{b^{2k}} + \frac{b^{k-1}}{n} - \frac{1}{b^{k+1}} + 1\right) \\ &> q-3, \end{aligned}$$

where is used the fact that $\frac{y}{x} - 1 < \left\lfloor \frac{y}{x} \right\rfloor \leq \frac{y}{x}$.

Therefore, we can obtain z mod n by subtracting qn mod b^{k+1} from $z[k:0]$ and then adjusting the result with at most two subtractions of n. There is no need of full multiplication of μ and $z[l-1:k-1]$, since we do not need the lower k+1 digits of the product. The total number of multiplications required by Barret's algorithm is at most k(k+4).

In Montgomery's algorithm [3] for T mod N computation a radix of $R=b^n$ is chosen, with (R, N) = 1 и R>N. N' is pre-calculated so that RR^{-1} - NN' = 1 ($RR^{-1} \equiv 1 \bmod N$). By using the function REDC(T) one can find:

$X=TR^{-1} \bmod N = (T+(TN' \bmod R)N)/R$.

For determining the actual T mod N the function is repeated once again using substitution REDC ((X mod N)R^2 mod N).

Suggested and analyzed are diverse variations based on this method [9].

Obviously, Montgomery's algorithm is not efficient for just a few modular multiplications, due to the relatively large overhead involved in argument transformations. When is used for modular exponentiation, this algorithm is efficient enough to compensate for such overhead. To compute $y = x^d \bmod n$ with Montgomery reduction, we first compute $y' = x^d \bmod n$ and then do Montgomery exponentiation with this number in a usual way.

Montgomery reduction can be performed in k(k+1) multiplications, superior to Barret's algorithm which requires k(k+4) multiplications. However, we have to note, that this number independent of the size of the target number z. This means that with Montgomery's algorithm we have to carry out the same number multiplications even for one digit reduction. It is thus obvious that we had better use the classical algorithm or Barret's algorithm for modular multiplications involving small numbers.

III. MODIFIED BARRETT'S ALGORITHM

Paul Barrett suggests an original algorithm for modulus reduction at X = A mod M calculation. The authors suggest replacing the integer division with operations requiring less processing time.

The expression for obtaining the remainder

$$X = A - M * \left\lfloor \frac{A}{M} \right\rfloor$$

is changed into X = A – M * (A * R).

For calculating the result two multiplication operations and one subtraction are performed. The problem is that R is less than 1 and it is necessary to work with a floating point. To represent R as an integer a pre-calculation of R = $\left\lfloor \frac{b^{2n}}{M} \right\rfloor$ is carried out, where b is the radix and n is the digit capacity of the modulus.

The integer part is calculated by the expression

$$\bar{Y} = \left\lfloor \frac{A}{b^{n-1}} \right\rfloor * \left\lfloor \frac{b^{2n}}{M} \right\rfloor * \frac{1}{b^{n+1}} \qquad (1)$$

According to Barrett, the result $\bar{X}$ is always less than (3M – 1). Therefore, at most two subtractions are necessary for result correction. In 90% of the cases $\bar{X}$ is less than M and only in 1 % $\bar{X}$ is greater than 2M [5,10].

Inaccuracy is caused by the elimination of the remainder in the integer divisions $\left\lfloor \frac{A}{b^{n-1}} \right\rfloor$ and $\left\lfloor \frac{b^{2n}}{M} \right\rfloor$.

The expression of calculating $\frac{A}{M}$ may be written in the following way:

$$\left(\frac{A}{b^{n-1}} * \frac{b^{2n}}{M}\right) * \frac{1}{b^{n+1}} \text{ at } b^{n-1} < M < b^n .$$

For $\frac{A}{b^{n-1}} = Q_1 + \frac{R_1}{b^{n-1}}$ and $\frac{b^{2n}}{M} = Q_2 + \frac{R_2}{M}$

we obtain $(Q_1 + \frac{R_1}{b^{n-1}}) * (Q_2 + \frac{R^2}{M}) * \frac{1}{b^{n+1}}$

$$= (Q_1 Q_2 + Q_1 \frac{R_2}{M} + (Q_2 + \frac{R_2}{M}) \frac{R_1}{b^{n-1}})* \frac{1}{b^{n+1}}$$

$$= (Q_1 Q_2 + Q_1 \frac{R_2}{M} + \frac{b^{2n}}{M} \frac{R_1}{b^{n-1}})* \frac{1}{b^{n+1}}$$

$$= (Q_1 Q_2 + Q_1 \frac{R_2}{M} + \frac{R_1}{M} b^{n+1})* \frac{1}{b^{n+1}} \qquad (2)$$

It is necessary to correct the result a

$$Q_1 Q_2 \bmod b^{n+1} + (Q_1 \frac{R_2}{M} + \frac{R_1}{M} b^{n+1})* \frac{1}{b^{n+1}} > 1 \qquad (3)$$

As $A < M^2$ and $b^{n-1} < M < b^n$,
then

$$Q_1 \frac{R_2}{M} < Q_1 \le \frac{M^2}{b^{n-1}} \frac{R_2}{M} < \frac{M^2}{b^{n-1}} \qquad (4)$$

and

$$\frac{R_1}{M} b^{n+1} < Q_2 < \frac{b^{2n}}{M} \qquad (5)$$

The maximum possible error in computing the integer part is $Y - \bar{Y} \le 3 - \frac{2}{b^{n+1}}$.

Therefore, at most 2 corrections of the result are necessary. The ratio between the maximum values of

$Q_1 \cdot \frac{R_2}{M}$ and $\frac{R_2}{M} b^{n+1}$ depends on the modulus value.

For $M > \frac{b^n}{2}$, $Q_1 \frac{R_2}{M} < b^{n+1}$ and

$\frac{R_1}{M} b^{n+1} < 2b^n$ (^{n} ($R_1 < b^{n-1}$ and replacing M).

For $M < \frac{b^n}{2}$, $Q_1 \frac{R_2}{M} < \frac{b^{n+1}}{4}$ and $\frac{R_1}{M} b^{n+1} <$ b^{n+1} (seen from equation 5).

In the general case it could be written as follows:

$$\bar{Y} = \left\lfloor \frac{A}{b^{n-p}} \right\rfloor * \left\lfloor \frac{b^{2n-p+s}}{M} \right\rfloor * \frac{1}{b^{n+s}} \qquad (6)$$

For $M > \frac{b^n}{2}$, and for reducing the error $Q_1 \frac{R_2}{M}$ is substituted with p=1 and s=2. In this case we obtain

$$(Q_1 Q_2 + Q_1 \frac{R_2}{M} + \frac{R_1}{M} b^{n+1})* \frac{1}{b^{n+2}}.$$

The maximum error probability in calculating the integer part is

$$Y - \bar{Y} < (b^{n+2} + b^{n+1} + 2b^{n+1})* \frac{1}{b^{n+2}} = 1 + \frac{3}{b} < 2$$

for b>3 (7)

For $M < \frac{b^n}{2}$, and reducing the error in $\frac{R_1}{M} b^{n+1}$, in (5) substitution with p=2 and s=2 is performed.

In this case the error is

$$Y - \bar{Y} < (b^{n+2} + \frac{b^{n+2}}{4} + b^{n+1})* \frac{1}{b^{n+2}}$$

$$= 1 + \frac{1}{4} + \frac{1}{b} < 2 \qquad (8)$$

Therefore, when increasing the digit capacity of Q_1 or Q_2 b times, the correction of the result of the modulus calculation at most 1. The original Barrett's choice (1) is p =1 and s =1.

For computing X = A mod M the following algorithm is suggested:

```
Input : $A = (a_{2n-1} \ldots a_1 a_0)_b$ and
$M = (m_{n-1} \ldots m_1 m_0)_b$ $0 \le A < M^2$ , $3 < b$
Output : $X = (x_{n-1} \ldots x_1 x_0)_b$
1. Pre-calculation of R
    1.1 If $M > \frac{b^n}{2}$ then $R \leftarrow \left\lfloor \frac{b^{2n+1}}{M} \right\rfloor$, $f \leftarrow 0$ else $R \leftarrow \left\lfloor \frac{b^{2n}}{M} \right\rfloor$, $f \leftarrow 1$
2. Calculation of the integer part Y
    2.1 If $f = 0$ then $Y_1 \leftarrow \left\lfloor \frac{A}{b^{n-1}} \right\rfloor$ else
        $Y_1 \leftarrow \left\lfloor \frac{A}{b^{n-2}} \right\rfloor$
    2.2 $Y_2 \leftarrow Y_1 * R$
    2.3 $Y \leftarrow \left\lfloor \frac{Y_2}{b^{n+2}} \right\rfloor$
3. Computation of the remainder $\bar{X}$
    3.1 $X_1 \leftarrow A \bmod b^{n+1}$
    3.2 $X_2 \leftarrow (Y * M) \bmod b^{n+1}$
    3.3 $\bar{X} \leftarrow X_1 - X_2$
4. Correction of the result X
    4.1 If $\bar{X} \ge M$ then $X \leftarrow \bar{X} - M$
```

In step 1 R is calculated, the floating point being substituted with a fixed point. The value of M is verified only once at the beginning and the result is stored in the flag f.

Step 2 determines the integer part, with the verification in 2.1 referring to the condition of only one bit. In step 3 the remainder is determined.

It is possible that $X_1 < X_2$ (for example 990000 mod 999), but since always $A \geq (Y * M)$ a verification may not be performed (algorithm 14.42 step 3 [11] and in the subtraction in 3.3 the borrow of n+2 digits is neglected. In step 4 the actual value of X is determined.

According to (7) and (8) there are two possibilities: either $X = \bar{X}$ or $X = \bar{X} + 1$.

A. *Examples*

Given are two examples with a radix b=10 and n=3 with values of A and M deliberately chosen to yield a greater error. As in Barrett's algorithm the calculation of Y does not depend on A mod b^{n-1}, 10^2 values of A = $(a_5 .. a_2 00)_{10} \div (a_5 .. a_2 99)_{10}$ are analyzed.

Example 1: for $M < \frac{b^n}{2}$ $A = (165a_1 a_0)_{10}$, $M = 129$,

$$\left\lfloor \frac{16500}{129} \right\rfloor = 127, \left\lfloor \frac{16599}{129} \right\rfloor = 128$$

In the original Barrett's algorithm

$R = \frac{10^6}{129} = 7751 + \frac{121}{129}$ is pre-calculated.

For all values of A the result obtained is

$$Y = \left\lfloor \frac{165a_1a_0}{100} \right\rfloor * \left\lfloor \frac{1000000}{129} \right\rfloor * \frac{1}{10000} = 127.$$

The maximum error (3) is $\frac{16599}{129} - Y < 1.7$.

We can see that 88 values of A ($16512 \div 16599$) necessitate a result correction by 1 and 12 values of ($16500 \div 16511$) give $Y = \bar{Y}$, no correction being required.

In the algorithm suggested, since $M < \frac{10^3}{2}$ (f=1 in step 2.1),

$$Y = \left\lfloor \frac{165a_1a_0}{10} \right\rfloor * \left\lfloor \frac{1000000}{129} \right\rfloor * \frac{1}{10000}.$$

Only for 8 values of A ($16512 \div 16519$) a result correction by 1 is required; for the remaining 92 values, $Y = \bar{Y}$ without any correction.

Example 2: for $M > \frac{b^n}{2}$ $A = (9388a_1 a_0)_{10}$,

$$M = 969, \left\lfloor \frac{938800}{969} \right\rfloor = 968, \left\lfloor \frac{938899}{969} \right\rfloor = 968$$

According to Barrett's algorithm we obtain

$R = \frac{10^6}{969} = 1031 + \frac{961}{969}$ For all values of A,

$$Y = \left\lfloor \frac{9388a_1a_0}{100} \right\rfloor * \left\lfloor \frac{1000000}{969} \right\rfloor * \frac{1}{10000} = 967.$$

The maximum error (3) is $\frac{938899}{969} - Y < 1.1$.

For all values of A ($16500 \div 16599$) correction of the result by 1 is necessary. In the algorithm suggested, as $M > \frac{10^3}{2}$ (f=0 in step 1.1), then

$$R = \left\lfloor \frac{9388a_1a_0}{100} \right\rfloor * \left\lfloor \frac{10000000}{969} \right\rfloor * \frac{1}{10000}.$$

For all values of A, result is $Y = \bar{Y}$ without any correction.

IV. CONCLUSIONS

The modified algorithm suggested has the basic advantage of correction reduction for obtaining the actual value of A mod M in Barrett's algorithm. Depending on the modulus value the digit capacity of the computations is increased. It was proved that the maximum error in this case is 1.25 + 1/b. The examples given show the difference between the original algorithm and the one suggested in the paper.

REFERENCES

[1] P.Antonov, S. Malchev. "Cryptography in computer communications", Varna 2000.

[2] R. Rivest, A. Shamir and A. Adleman , "A method for obtaining for digital signatures and public-key cryptosystems", CACM, vol.21, pp. 120-126, 1978.

[3] P. Montgomery, "Modular multiplication without trial division", Mathematics of Computation", vol.44, pp519-521, 1985.

[4] C. Koc, "Montgomery Reduction with Even Modulus", IEE Proceedings : Computers and Digital Techiques, 141(5) , pp.314-316, 1985.

[5] P. Barrett, "Implementing the Rivest Shamir Adleman public-key encryption algorithm on a standart digital signal processor", Advances in Cryptology –CRYPTO'86, pp.311-323, 1987

[6] R. Wolfgang, E. Wolfgang, "Smart card handbook" 3rd edition, November 2003.

[7] L. Hars, "Applications of Fast Truncated Multiplication in Cryptography", Edinburgh CHES 2005.

[8] C. Koc, "Analysis of Sliding Window Techniques for Exponentiation", Computers and Mathematics with Applications, vol.30, no.10, pp. 17-24, 1995.

[9] C. Koc, T. Acar and B. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms" , IEEE Micro, no.16, pp. 26-33, 1996.

[10] A. Bosselaers, R. Govaerts and J. Vandewalle, "Comparison of three modular reductions", Advances in Cryptology – Crypto '93 (LNCS 773), Springer-Verlag, pp. 175-186, 1994.

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