Design and Implementation of Fast Multiplication Algorithms in Public Key Cryptosystems for Smart Cards

G. Joseph and W.T. Penzhorn

Abstract—Most practical public-key cryptosystems are based on modular exponentiation. A modular exponentiation is composed of repeated modular multiplications. Several methods have been proposed to reduce the execution time of a modular exponentiation, essentially aiming to reduce the execution time of each modular multiplication. The goal of this paper is to investigate three different integer multiplication techniques, as used in conjunction with various public-key cryptographic algorithms as used a standard smart card. The aim is to obtain exact numerical results for standard multiplication algorithms used in industry.

Keywords—Public-key cryptosystems, Montgomery multiplication, Barrett reduction, Classical multiplication

I. INTRODUCTION

The need for security in e-commerce has lead to the study of public key cryptography. Since 1976, numerous public-key cryptosystems have been proposed, mostly based on modular exponentiation. The most well-known public-key cryptosystems in use today are based on the difficulty of factorizing large integers (RSA algorithm [1]) and on the difficulty of computing discrete logarithms (Diffie-Hellman key exchange [2], ElGamal [3], Digital Signature Standard(DSS) [4]).

In this paper we will introduce three different algorithms for the modular multiplication of large integers. The simplest algorithm for modular multiplication is utilizing normal integer multiplication, as well as some method for performing modular reduction [5]. Barrett introduced one of the first methods to perform modular multiplication without modular division [6]. It involves a pre-computation step for the reduction step before actually implementing the multiplication. Montgomery proposed an alternative method in 1985, which allows efficient implementation of modular multiplication without explicitly performing the reduction step [7]. These three approaches will be discussed in detail in Section III.

II. OVERVIEW OF PUBLIC-KEY CRYPTOSYSTEMS

A. The RSA Algorithm

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem [1]. It can provide both encryption, decryption and digital signatures. Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement. The security of the RSA cryptosystem depends on the problem of factoring large integers.

The RSA algorithm operates as follows: Two keys $p$ and $q$ are generated and the modulus $n = pq$ is computed. The encryption key $e$ is chosen such that $gcd(e, (p - 1)(q - 1)) = 1$. The decryption key $d$ is computed such that $ed \equiv 1 (mod (p - 1)(q - 1))$. The public key is $(e, n)$ and the private key is $(d)$. To encrypt a message $m$, compute $c_i = m^e \mod n$. Decryption is done as follows: $m_i = c_i^d \mod n$.

![Fig. 1. The RSA encryption and decryption algorithms](image-url)

B. The Diffie-Hellman Key Exchange

Diffie-Hellman was developed in 1976, and its security is based on the difficulty of calculating discrete logarithms in a finite field. The Diffie-Hellman algorithm is used in key distribution protocols. The basic version provides passive protection, but may be prone to interception or modification (man-in-the-middle attacks). However Diffie-Hellman can be extended, in order to prevent man-in-the-middle attacks [8,9].

The objective of the algorithm is that two parties A and B are enabled to derive a common secret key $(k)$ over an open, insecure channel, as shown in Fig. II-B.

C. The ElGamal Algorithm

This algorithm is an extension of the basic Diffie-Hellman algorithm, and also depends on the difficulty of computing discrete logarithms over finite fields. The ElGamal scheme can be used for encryption, decryption and digital signatures. Pretty good privacy (PGP), the well-known workhorse for encrypting and signing e-mail messages and documents, uses the ElGamal procedure for its key management.

The key generation requires each entity $A$ needs to create a public key and a corresponding private key. In order to create the keys, a large random prime $p$ and a generator $\alpha$ of the multiplicative group $Z_p$ modulo $p$ is generated. Then select a random integer $a$, $1 < a < p - 2$, and compute $\alpha^a \mod p$. Entity $B$'s public key is $(p, \alpha, \alpha^a)$ and the private key is $(a)$.
D. The Digital Signature Standard (DSS)

The U.S. National Institute of Standards and Technology (NIST), has proposed an algorithm for digital signatures. The algorithm is known as DSA, Digital Signature Algorithm. As a proposed standard it is now known as the Digital Signature Standard (DSS). The DSA algorithm is due to Kravitz [10] and was proposed as a Federal Information Processing Standard in August 1991 by NIST. It became the Digital Signature Standard (DSS) in May 1994, as specified in FIPS 186 [4]. DSS uses the ElGamal algorithm as its basis.

For the generation of DSA primes $p$ and $q$ in the algorithm below, it is required to select the prime $q$ first, and then try to find a prime $p$ such that $q$ divides $(p - 1)$. Each party creates a public key and corresponding private key.

III. MODULAR MULTIPLICATION TECHNIQUES

In recent years, considerable endeavours were invested in the design of efficient modular multiplication algorithms. Modular exponentiation is composed of a sequence of modular multiplication operations. Modular multiplication is more involved than $2k$-bit multiplication, requiring both $2k$-bit multiplication and a method to perform modular reduction.

In this paper we are looking at the complexity of the following operation: $a \times b = x \mod m$. Reduction of the time and memory complexity of the operation $x \mod m$, greatly influences the practical feasibility of a cryptosystem’s signature and encryption scheme.

A. Classical Modular Multiplication

Modular multiplication relies on both $2k$-bit multiplication and a method of modular reduction. The easiest way to perform modular reduction is to compute the remainder $r$ by division with the modulus $m$, using the division algorithm. This combination will be referred to as the classical algorithm for performing modular multiplication.

Parameter definition: Given $w = (w_0...w_{t-1}w_0)_b$ and $m = (m_1...m_t m_0)_b$ with $n ≤ t ≤ 1$, $m_0 \neq 0$. When $w$ is divided by $m$ we obtain the quotient $q$ and the remainder $r = (r_1...r_{t-1}r_0)_b$. The division step follows the equation $w = qm + r$, $0 ≤ r < m$.

B. Barrett Modular Multiplication

Barrett [6] introduced the idea of estimating the quotient $x/di\text{m}$ with operations that either are less expensive in time than a division by $m$, or can be done as a precalculation for a given $m$. The Barrett reduction computes $r = x \mod m$ given $x$ and $m$. The algorithm requires the precomputation of the quantity $\gamma = [b^{2k}/m]$, it is advantageous if many reductions are performed with a single modulus. The precomputation takes a fixed amount of work, which is negligible in comparison to modular exponentiation cost.

Parameter definition: Two positive integers $a$ and $b$ produces $x = (x_{2k-1}...x_1x_0)_b$ and $m = (m_{k-1}...m_1 m_0)_b$, with $m_{k-1} \neq 0$ and $\gamma = [b^{2k}/m]$. The output will be the remainder $r = x \mod m$. 

Figure 2. The Diffie-Hellman key exchange

Figure 3. The ElGamal encryption and decryption algorithms

Figure 4. DSS signing and verification
1. Compute \( w = x \cdot y \) (using multiple-precision multiplication).

2. Compute the remainder \( r \) when \( w \) is divided by \( m \).

    (a) While \( (w \geq mb^{n-1}) \) do \( w - mb^{n-t} \rightarrow w \).

    (b) For \( i \) from \( n \) down-to \((t + 1)\) do the following:
        i. If \( w_i = m \) then set \( b \rightarrow q \);
           else set \( q \rightarrow (w_i + w_{i-1}) / m_i \).
        ii. While \( (q(m_i b + m_{i-1}) > w_i b^2 + w_{i-1} b + w_{i-2}) \)
            do: \( q \rightarrow q - 1 \).
        iii. \( w - qmb^{n-t-1} \rightarrow w \).
        iv. If \( w < 0 \) then
            set \( w + mb^{n-t-1} \rightarrow w \) and \( q \rightarrow q - 1 \).
    (c) \( w \rightarrow r \).

3. Return \( r \).

Fig. 5. Classical modular multiplication

1. Compute \( x = a \cdot b \) (Using 2k-bit multiplication)

2. Compute remainder \( r \) using Barrett reduction

    (a) \( q_1 \rightarrow \lfloor x/b^{k-1} \rfloor \), \( q_2 \rightarrow q_2 / 2 \), \( q_3 \rightarrow \lfloor q_2 / b^{k+1} \rfloor \).

    (b) \( x \ \text{mod} \ b \rightarrow r_1, q_1 \ \text{mod} \ b \rightarrow r_2, r_1 \rightarrow r_2 \rightarrow r \).

    (c) If \( r < 0 \) then \( r + b^{k+1} \rightarrow r \).

    (d) While \( r \geq m \) do: \( r - m \rightarrow r \).

3. Return \( r = x \ \text{mod} \ m \).

Fig. 6. Barrett modular multiplication

C. Montgomery Modular Multiplication

The third approach to modular multiplication was suggested by Montgomery [7]. Montgomery reduction is a technique which does efficient implementation of modular multiplication without explicitly carrying out the classical modular reduction step. Montgomery multiplication combines Montgomery reduction and multiple-precision multiplication to compute the Montgomery reduction of the product of two integers. It is a generalization of a much older technique due to Hensel [11].

Montgomery reduction can be described as follows: Let \( m \) be a positive integer, and \( R \) and \( T \) are integers (such that \( R > m \), \( \gcd(m, R) = 1, \ 0 \leq T < mR \)). A method is described for computing \( TR^{-1} \mod m \) without using the classical method. \( TR^{-1} \mod m \) is called a Montgomery reduction of \( T \) modulo \( m \) with respect to \( R \). In other words, the \( m \)-residue of \( T \) with respect to \( R \). With a suitable choice of \( R \), a Montgomery reduction can be efficiently computed.

Mathematically, Montgomery’s reduction can be stated as follows: Given integers \( m \) and \( R \) let \( \tilde{m} = -m^{-1} \ mod \ R \). If \( \gcd(m, R) = 1 \), then for all integers \( T \) (where \( 0 \leq T < mR \)), \( U = T \tilde{m} \mod m \) is an integer satisfying:

\[
    \frac{T + Um}{R} \equiv TR^{-1} \mod m
\]

Parameter definition: Integers \( m = (m_{n-1} \ldots m_1 m_0)_b, x = (x_{n-1} \ldots x_1 x_0)_b, y = (y_{n-1} \ldots y_1 y_0)_b \) satisfying \( 0 \leq x, y < m, \ R = b^n \) and \( \gcd(m, b) = 1 \). To obtain \( xyR^{-1} \tilde{m} = -m^{-1} \mod b \) is computed.

1. \( 0 \rightarrow A \). Notation: \( A = (a_{2n-1} \ldots a_1 a_0)_b \)

2. For \( i \) from \( 0 \) to \((n - 1)\) do:
    (a) \( u_i \rightarrow (a_0 + x_i y_0) \mod b \).
    (b)  \( A \rightarrow (A + x_i y_i u_i) \mod b \).

3. If \( A \geq m \) then \( A \rightarrow A - m \).

4. Return \( A \).

Fig. 7. Montgomery modular multiplication

IV. EXPERIMENTAL RESULTS

A. A comparison of the multiplication algorithms

In this section we present experimental results of the performance of the various multiplication algorithms. The performance of the algorithm is attributed to the multiplications and divisions required to reduce an \( 2k \)-bit integer. The reason for these investigations is that the multiplication and division are the most time consuming operations in the inner loop of all three algorithms.

In Table I we show the theoretical number of multiplications and divisions required for the reduction operation only i.e. they do not include the multiplications and divisions of the precalculation, any transformation or postcalculation [5]. The results refers to the reduction of a \( 2k \)-digit number with a \( k \)-digit modulus \( m \).

In Table II we present the simulation results for the three algorithms, corresponding to the theoretical results in Table I. The three multiplication algorithms were implemented in ANSI C, utilizing some of the functions of the MIRACL library. These results were obtained on a 550MHz Pentium III based PC using the 32-bit Borland C++ 6.0 platform. All times are given in microseconds, unless otherwise stated.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classical</th>
<th>Barrett</th>
<th>Montgomery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplications</td>
<td>( k(k + 2.5) )</td>
<td>( k(k + 4) )</td>
<td>( k(k + 1) )</td>
</tr>
<tr>
<td>Divisions</td>
<td>( k )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Precalculations</td>
<td>Normalization</td>
<td>( b^{2k} / m )</td>
<td>(-m^{-1} \mod b )</td>
</tr>
<tr>
<td>Argument transformation</td>
<td>None</td>
<td>None</td>
<td>( m )-residue</td>
</tr>
<tr>
<td>Postcalculations</td>
<td>Unnormalization</td>
<td>None</td>
<td>Reduction</td>
</tr>
<tr>
<td>Restrictions</td>
<td>None</td>
<td>( x &lt; b^{2k} )</td>
<td>( x &lt; mb^{2k} )</td>
</tr>
</tbody>
</table>

B. Comparison of the Public-Key Cryptosystems

In this section we present simulation results for the four public-key cryptosystems. These public-key cryptosystem provide differing environments for the evaluation of the three

1 MIRACL is a portable C library which implements multi-precision integer and rational data-types, and provides routines to perform basic arithmetic on them.
multiplication algorithms. For the exponentiation process, each cryptosystem requires the repetition of modular multiplications. Table III implements a left-to-right binary exponentiation [12] for its modular exponentiation and Montgomery multiplication for the modular multiplication.

The implementation was written with the aid of the MIR-ACL library. The cryptosystems were optimized for speed not security in order to test time required per multiplication / exponentiation cycle with precomputation. A precomputation in an algorithm is the computation of required parameters before the algorithm is executed. A postcalculation is the performing a function on the result after the algorithm has been executed.

### Table II

<table>
<thead>
<tr>
<th>$k$</th>
<th>$m$</th>
<th>Classical ($\mu s$)</th>
<th>Barrett ($\mu s$)</th>
<th>Montgomery ($\mu s$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>256</td>
<td>17.21</td>
<td>16.12</td>
<td>14.82</td>
</tr>
<tr>
<td>256</td>
<td>512</td>
<td>67.81</td>
<td>61.23</td>
<td>52.26</td>
</tr>
<tr>
<td>512</td>
<td>1024</td>
<td>317.81</td>
<td>275.23</td>
<td>207.65</td>
</tr>
<tr>
<td>1024</td>
<td>2048</td>
<td>1017.81</td>
<td>815.23</td>
<td>783.83</td>
</tr>
</tbody>
</table>

If the precomputation and post-calculations and the $m$-residue transformation are compensated by a faster modular reduction An operation using Barrett’s or Montgomery’s modular reduction methods will only be faster than the corresponding operation using the classical modular reduction if the pre- and post-calculation and the $m$-residue transformation only for Montgomery are compensated by faster modular reductions (i.e. modular exponentiation).

From Table III it follows that the ElGamal algorithm’s execution time for decryption (with and without precomputation) is approximately the same as creating a DSS signature. This can explained due to the same amount of multiplications and exponentiations required by both algorithms. A similar kind of explanation applies to DSS signature verification and ElGamal encryption.

In terms of processing speed, RSA encryption for small and fixed exponents is preferred to ElGamal encryption. However, RSA decryption is much slower than encryption. The decryption time can be improved utilizing the Chinese remainder theorem as described in [13]. Precomputation reduces the execution time as shown in Table III.

An integration of multiplication techniques in its public-key cryptosystem environment will give a comprehensive comparison of multiplication algorithms utilizing a 8-bit processor (i.e. smart card). The results require an integration of the public-key cryptosystem with its multiplication counterparts. Further research will emphasis to obtain a detailed comparison between standard multiplication algorithms used in industry cryptosystems.

### Table III

<table>
<thead>
<tr>
<th>$k$</th>
<th>Decryption</th>
<th>Encryption $e = 3$</th>
<th>Encryption $e = 65537$</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>56.95</td>
<td>0.25</td>
<td>1.98</td>
</tr>
<tr>
<td>1024</td>
<td>399.37</td>
<td>0.51</td>
<td>3.98</td>
</tr>
<tr>
<td>2048</td>
<td>2997.12</td>
<td>1.69</td>
<td>12.91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$</th>
<th>Encryption</th>
<th>Decryption $e = 3$</th>
<th>Decryption $e = 65537$</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>160</td>
<td>9.58</td>
<td>3.35</td>
</tr>
<tr>
<td>1024</td>
<td>160</td>
<td>34.10</td>
<td>12.13</td>
</tr>
<tr>
<td>2048</td>
<td>256</td>
<td>199.12</td>
<td>61.44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$</th>
<th>Signing</th>
<th>Signing</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>160</td>
<td>9.67</td>
<td>3.37</td>
</tr>
<tr>
<td>1024</td>
<td>160</td>
<td>34.12</td>
<td>12.31</td>
</tr>
<tr>
<td>2048</td>
<td>256</td>
<td>198.12</td>
<td>62.44</td>
</tr>
</tbody>
</table>

### V. Discussion

Table I indicates that, if only the reduction operation is considered (i.e. without the precalculations, argument transformations and postcalculations) Montgomery’s algorithm is clearly faster than Barrett’s and the classical algorithm. Barrett’s algorithm is only slightly faster than the classical algorithm, and these observations are confirmed by the simulation results given in Table II.

### VI. Conclusion

A basic operation in modern public key cryptosystems is the modular multiplication of large integers. Since the initial multiple-precision multiplication is common, the difference comes in the reduction step. An efficient implementation of this operation is the key to high performance. Three well known multiplication algorithms are investigated, implemented and evaluated with respect to their software performance.

Four public-key algorithms implemented and compared. It was shown that they all have their specific behavior resulting in a specific field of application (i.e. public-key cryptosystem).

A theoretical and practical comparison has been made of three algorithms for the multiplication of large numbers. It has been shown that from the software implementation the three algorithms, Montgomery is the fastest between the three for $k$-bit modulus. However a good implementation will leave minor differences in performance between the three algorithms.

### References


George Joseph holds a B.Eng(2001) degree from the University of Pretoria with specialization in network security and public-key cryptography. He is currently busy with a M.Eng degree at the University of Pretoria, looking into methods to improve performance of public-key cryptosystems used for smart cards. He is an employee of Telkom SA Ltd. since 2002.

Walter Penzhorn holds degrees from University of Pretoria and University of London. After working for the CSIR for 17 years, he joined the Department of Electrical, Electronic and Computer Engineering at the University of Pretoria in 1990 as Associate Professor. Since 1999 he is the director of Telkom’s Centre of Excellence in Teletraffic at the University of Pretoria. He is a senior member of the IEEE, and member of the SAIEE and ECSA. He has more than 25 years of experience in the design and analysis of cryptosystems.