The Montgomery Modular Inverse - Revisited

E. Savaş, Ç. K. Koç

Abstract— **We modify an algorithm given by Kaliski to compute the Montgomery inverse of an integer modulo a prime number. We also give a new definition of the Montgomery inverse, and introduce efficient algorithms for computing the classical modular inverse, the Kaliski-Montgomery inverse, and the new Montgomery inverse. The proposed algorithms are suitable for software implementations on general-purpose microprocessors.**

Keywords— **Modular arithmetic, modular inverse, almost inverse, Montgomery multiplication, cryptography.**

I. Introduction

The basic arithmetic operations (i.e., addition, multiplication, and inversion) modulo a prime number p have several applications in cryptography, for example, the decipherment operation in the RSA algorithm [9], the Diffie-Hellman key exchange algorithm [1], the US Government Digital Signature Standard [8], and also recently elliptic curve cryptography [5], [6]. The modular inversion operation plays an important role in public-key cryptography, particularly, to accelerate the exponentiation operation using the so-called addition-subtraction chains [2], [4] and also in computing point operations on an elliptic curve defined over the finite field $GF(p)$ [5], [6].

The modular inverse of an integer $a \in [1, p-1]$ modulo the prime p is defined as the integer $x \in [1, p-1]$ such that $ax = 1 \pmod p$. It is often written as $x = a^{-1} \pmod p$. This is the classical definition of the modular inverse [4]. In the sequel, we will use the notation

$$x := \text{ModInv}(a) = a^{-1} \pmod p \tag{1}$$

to denote the inverse of a modulo p. The definition of the modular inverse was recently extended by Kaliski to include the so-called Montgomery inverse [3] based on the Montgomery multiplication algorithm [7]. In this paper, we introduce a new definition of the Montgomery inverse, and also give efficient algorithms to compute the classical modular inverse, the Kaliski-Montgomery inverse, and the new Montgomery inverse of an integer a modulo the prime number p.

II. The Montgomery Inverse

The Montgomery multiplication [7] of two integers $a, b \in [0, p-1]$ is defined as $c = ab2^{-n} \pmod p$ where $n = \lceil \log_2 p \rceil$. We denote this multiplication operation using the notation

$$c := \text{MonPro}(a, b) = ab2^{-n} \pmod p \ , \tag{2}$$

Electrical and Computer Engineering, Oregon State University, Corvallis, Oregon 97331, E-mail: savas@ece.orst.edu, koc@ece.orst.edu

This work is supported by Secured Information Technology, Inc.

where p is the prime number and n is its bit-length. The Montgomery inverse of an integer $a \in [1, p-1]$ is defined by Kaliski [3] as the integer $x = a^{-1}2^n \pmod p$. Similarly, we will use the notation

$$x := \text{MonInv}(a) = a^{-1}2^n \pmod p \tag{3}$$

to denote the Montgomery inversion as defined by Kaliski. The algorithm introduced in [3] computes the Montgomery inverse of a. We give this algorithm below. The output of Phase I is the integer r such that $r = a^{-1}2^k \pmod p$, where $n \le k \le 2n$. This result is then corrected using Phase II to obtain the Montgomery inverse $x = a^{-1}2^n \pmod p$.

Phase I
Input: $a \in [1, p-1]$ and p
Output: $r \in [1, p-1]$ and k, where $r = a^{-1}2^k \pmod p$ and $n \le k \le 2n$

1: $u := p$, $v := a$, $r := 0$, and $s := 1$
2: $k := 0$
3: while $(v > 0)$
4: if u is even then $u := u/2$, $s := 2s$
5: else if v is even then $v := v/2$, $r := 2r$
6: else if $u > v$ then $u := (u-v)/2$, $r := r+s$, $s := 2s$
7: else if $v \ge u$ then $v := (v-u)/2$, $s := s+r$, $r := 2r$
8: $k := k+1$
9: if $r \ge p$ then $r := r-p$
10: return $r := p-r$ and k

Phase II
Input: $r \in [1, p-1]$, p, and k from Phase I
Output: $x \in [1, p-1]$, where $x = a^{-1}2^n \pmod p$

11: for $i = 1$ to $k-n$ do
12: if r is even then $r := r/2$
13: else then $r := (r+p)/2$
13: return $x := r$

III. The Almost Montgomery Inverse

As shown above, Phase I computes an integer $r = a^{-1}2^k \pmod p$, where $n \le k \le 2n$. The Montgomery inverse of a is defined as $x = a^{-1}2^n \pmod p$ where $n = \lceil \log_2 p \rceil$. We will call the output of the Phase I the *almost Montgomery inverse* of a, and denote it as

$$(r, k) := \text{AlmMonInv}(a) = a^{-1}2^k \pmod p \ , \tag{4}$$

where $n \le k \le 2n$, in the sequel. We note that a similar concept, the almost inverse of elements in the Galois field $GF(2^m)$, was introduced in [11] and some implementation issues were addressed in [10].

Since k is an output of the Phase I, we will include it in the definition of the AlmMonInv function as an output

value. We also propose to make an additional change in the way the almost Montgomery inverse algorithm is being used. Instead of selecting the Montgomery radix as $R = 2^n$ where $n = \lceil \log_2 p \rceil$, we will select it as $R = 2^m$, where m is an integer multiple of the wordsize of the computer w, i.e., $m = iw$ for some positive integer i. The Montgomery product algorithm would work with any m as long as $m \geq n$, where n is the bit-length of the prime number p. For efficiency reasons, we select the smallest i which makes m larger than n, in other words, $iw = m \geq n$, but $(i-1)w < n$. It turns out that the almost Montgomery inverse algorithm (Phase I) works for this case as well. Furthermore, it even works for an input a which may be larger than p as long as it is less than 2^m, as proven below in Theorem 1. The second issue is the value of k after the almost Montgomery inverse algorithm terminates. We show in Theorem 2 below that $n \leq k \leq m + n$.

Theorem 1: If $p > 2$ is a prime and $a \geq 1$ (a might be larger than p), then the intermediate values r, s, and u in the almost Montgomery inverse algorithm are always in the interval $[0, 2p-1]$.

Proof: If $a < p$, then the proof given in [3] is applicable here. If $a > p$ and a is not an integer multiple of p, then only Step 5 and Step 7 are executed in the while loop until v becomes smaller than u. Until then, the variables u, r, and s keep their initial values. They start changing when $v < u$, and after this point, the algorithm proceeds as in the case $a < p$. Thus, the intermediate values remain in the interval $[0, 2p-1]$ for $a > p$ as well. ■

Theorem 2: If $p > 2$ is a prime and $a \geq 1$, then the index k produced at the end of the almost Montgomery inverse algorithm takes a value between n and $m+n$, where $n = \lceil \log_2 p \rceil$ and $m = sw$ with $sw \geq n$ with $(s-1)w < n$.

Proof: The reduction of uv and $u+v$ at each iteration (at Steps 4-7) is illustrated below:

	uv	$u+v$
Step 4	$(uv)/2$	$(u+2v)/2$
Step 5	$(uv)/2$	$(2u+v)/2$
Step 6	$u^2/2-(uv)/2$	$(u+v)/2$
Step 7	$(uv)/2-v^2/2$	$(u+v)/2$

Note that these steps are mutually exclusive, i.e., at an iteration only one of the four cases occurs. At each iteration, the value uv is at least halved while the value $u+v$ is at most halved, and furthermore, both u and v are equal to 1 before the last iteration. Since the initial values of the product uv and the sum $u+v$ are ap and $a+p$, respectively, the index value k (i.e., the number of iterations) satisfies

$$(a+p)/2 \leq 2^{k-1} \leq ap \ .$$

Since $2^{n-1} < p < 2^n$ and $0 < a < 2^m$, we have

$$\begin{aligned} 2^{n-2} &< 2^{k-1} < 2^m \cdot 2^n \ , \\ 2^{n-1} &\leq 2^{k-1} \leq 2^{m+n-1} \ . \end{aligned}$$

Thus, we obtain the result: $n \leq k \leq m+n$. Furthermore, we note that $m - n \leq w - 1$, where w is the word size of the machine. This implies that $m - w + 1 \leq k \leq m + n$. ■

IV. Using the Almost Montgomery Inverse

The Montgomery inverse algorithm computes $x = a^{-1}2^n \pmod p$. The Kaliski algorithm [3] uses the bit-level operations in Phase II in order to achieve its goal. It uses $k-n$ steps in Phase II, where at each step a bit-level right shift operation is performed. Additionally, if r is odd, an addition operation $r+p$ needs to be performed.

As suggested earlier, we will use the definition $x = a^{-1}2^m \pmod p$. Furthermore, it is possible to eliminate the bit-level operations completely, and use the Montgomery product algorithm to obtain the same result. In our approach, we replace these bit-level operations by word-level Montgomery product operations which are intrinsically faster on microprocessors, particularly when the wordsize of the computer is large (i.e., 16, 32, or 64).

The new Phase II is based on the pre-computed Montgomery radix $R = 2^m \pmod p$, however, we only need $R^2 \pmod p$. This value can be pre-computed and saved, and used as necessary. Another issue is the range of input variables to the AlmMonInv and MonPro functions. For both of these functions, any input cannot exceed $2^m - 1$.

A. The Modified Kaliski-Montgomery Inverse

This algorithm computes $x = \text{MonInv}(a) = a^{-1}2^m \pmod p$ given the integer a. Thus, it finds the inverse of the integer a modulo p and also converts it to the Montgomery domain. The modified Kaliski-Montgomery inverse algorithm is given below.

Input: a, p, n, and m, where $a \in [1, 2^m - 1]$.
Output: $x = a^{-1}2^m \pmod p$, where $x \in [1, p-1]$.

1: $(r, k) := \text{AlmMonInv}(a)$ where $r = a^{-1}2^k \pmod p$ and $n \leq k \leq m+n$.
2: If $n \leq k \leq m$ then
2.1: $r := \text{MonPro}(r, R^2) = (a^{-1}2^k)(2^{2m})(2^{-m}) = a^{-1}2^{m+k} \pmod p$
2.2: $k := k + m > m$
3: $r := \text{MonPro}(r, 2^{2m-k}) = a^{-1} \cdot 2^k \cdot 2^{2m-k} \cdot 2^{-m} = a^{-1}2^m \pmod p$
4: Return $x = r$, where $x = a^{-1}2^m \pmod p$

The inputs to the MonPro function in Step 2.1 are r and R^2, which are both in the correct range. The input 2^{2m-k} to MonPro in Step 3 is also in the correct range since k is adjusted to be larger than m in Step 2.2 when $k \leq m$, thus, $0 < 2^{2m-k} < 2^m$.

B. The Classical Modular Inverse

In some cases, we are only interested in computing $x = \text{ModInv}(a) = a^{-1} \pmod p$ without converting to the Montgomery domain. One way to achieve this is first to compute the Kaliski-Montgomery inverse of a to obtain $b = a^{-1}2^m \pmod p$, and then re-convert the result back to the residue (non-Montgomery) domain using the Montgomery product as

$$\begin{aligned} b &:= \text{MonInv}(a) = a^{-1}2^m \pmod p \ , \\ x &:= \text{MonPro}(b, 1) = (a^{-1}2^m)(1)2^{-m} = a^{-1} \pmod p \ . \end{aligned}$$

Another way of computing the classical inverse is by reversing the order of MonInv and MonPro operations, and using the constant $R^2 = 2^{2m} \pmod p$ as follows:

$$\begin{aligned} b &:= \text{MonPro}(a, R^2) = (a)(2^{2m})2^{-m} = a2^m \pmod p , \\ x &:= \text{MonInv}(b) = (a2^m)^{-1}2^m = a^{-1} \pmod p . \end{aligned}$$

However, either one of these approaches requires 2 or 3 Montgomery product operations in addition to the AlmMonInv function. Instead, we can modify the Kaliski-Montgomery inverse algorithm so that it directly computes the classical modular inverse after the AlmMonInv function with 1 or 2 Montgomery product operations.

Input: a, p, n, and m, where $a \in [1, 2^m - 1]$
Output: $x = a^{-1} \pmod p$, where $x \in [1, p-1]$

1: $(r, k) := \text{AlmMonInv}(a)$ where $r = a^{-1}2^k \pmod p$ and $n \le k \le m + n$.
2: If $k > m$ then
2.1: $r := \text{MonPro}(r, 1) = (a^{-1}2^k)(2^{-m}) = a^{-1}2^{k-m} \pmod p$
2.2: $k := k - m < m$
3: $r := \text{MonPro}(r, 2^{m-k}) = (a^{-1})(2^k)(2^{m-k})(2^{-m}) = a^{-1} \pmod p$
4: Return $x = r$, where $x = a^{-1} \pmod p$

C. The New Montgomery Inverse

We propose the following new definition of the Montgomery inverse: $x = a^{-1}2^{2m} \pmod p$ given the input $a \pmod p$. According to this new definition, we compute the Montgomery inverse of an integer which is already in the Montgomery domain, producing the output x which is also in the Montgomery domain. We will denote the new Montgomery inverse computation by

$$x := \text{NewMonInv}(a2^m) = (a2^m)^{-1}2^{2m} = a^{-1}2^m \pmod p.$$

The Kaliski-Montgomery inverse of a is defined as $\text{MonInv}(a) = a^{-1}2^m \pmod p$, which has the following property

$$\begin{aligned} \text{MonPro}(a, \text{MonInv}(a)) &= \text{MonPro}(a, a^{-1}2^m) \\ &= a(a^{-1}2^m)2^{-m} = 1 \pmod p. \end{aligned}$$

In other words, according to the Kaliski-Montgomery inverse, the multiplicative identity is equal to 1, which is an incorrect assumption if we are operating in the Montgomery domain where the image of 1 is $2^m \pmod p$. On the other hand, the new Montgomery inverse has the following property

$$\begin{aligned} \text{MonPro}(a2^m, \text{NewMonInv}(a2^m)) &= a2^m(a^{-1}2^m)2^{-m} \\ &= 2^m \pmod p . \end{aligned}$$

This new definition of the inverse is more suitable for computing expressions using the Montgomery multiplication since it computes the result in the Montgomery domain.

The new Montgomery inverse cannot be directly computed using the MonInv algorithm by giving the input as $a2^m \pmod p$, since we would obtain

$$\text{MonInv}(a2^m) = (a2^m)^{-1}2^m = a^{-1} \pmod p .$$

However, this can be converted back to the Montgomery domain using a single Montgomery product with $R^2 \pmod p$. Thus, we obtain a method of computing the new Montgomery inverse as

$$\begin{aligned} b &:= \text{MonInv}(a2^m) = (a2^m)^{-1}2^m = a^{-1} \pmod p , \\ x &:= \text{MonPro}(b, R^2) = a^{-1}2^{2m}2^{-m} = a^{-1}2^m \pmod p . \end{aligned}$$

Similarly, another method to obtain the same result is by reversing the order of the operations:

$$\begin{aligned} b &:= \text{MonPro}(a2^m, 1) = (a2^m)(1)(2^{-m}) = a \pmod p , \\ x &:= \text{MonInv}(a) = a^{-1}2^m \pmod p . \end{aligned}$$

The new algorithm uses the pre-computed value $R^2 \pmod p$, and it is more efficient: It uses only 2 or 3 Montgomery product operations after the AlmMonInv function.

Input: $a2^m \pmod p$, p, n, and m
Output: $x = a^{-1}2^m \pmod p$, where $x \in [1, p-1]$

1: $(r, k) := \text{AlmMonInv}(a2^m)$ where $r = a^{-1}2^{-m}2^k \pmod p$ and $n \le k \le m + n$
2: If $n \le k \le m$ then
2.1: $r := \text{MonPro}(r, R^2) = (a^{-1}2^{-m}2^k)(2^{2m})(2^{-m}) = a^{-1}2^k \pmod p$
2.2: $k := k + m > m$
3: $r := \text{MonPro}(r, R^2) = (a^{-1}2^{-m}2^k)(2^{2m})(2^{-m}) = a^{-1}2^k \pmod p$
4: $r := \text{MonPro}(r, 2^{2m-k}) = (a^{-1}2^k)(2^{2m-k})(2^{-m}) = a^{-1}2^m \pmod p$
5: Return $x = r$, where $x = a^{-1}2^m \pmod p$

V. Conclusions and Applications

We have proposed a new definition of the Montgomery inverse, and have given efficient algorithms to compute the classical modular inverse, the Kaliski-Montgomery inverse, and the new Montgomery inverse. The new algorithms are based on the almost Montgomery inverse function and require 2 or 3 Montgomery product operations thereafter, instead of using the bit-level operations as in [3].

Table 1: Implementation results.

Bit	PhI	Algorithm	Old PhII	New PhII	PhII Spd	All Spd
160	138 μs	MonInv	30 μs	9 μs	3.34	1.14
		ModInv	85 μs	10 μs	8.50	1.51
		NewMonInv	57 μs	10 μs	5.70	1.32
192	192 μs	MonInv	39 μs	11 μs	3.54	1.14
		ModInv	128 μs	13 μs	9.85	1.56
		NewMonInv	87 μs	13 μs	6.69	1.36

We have performed some experiments by implementing all three inversion algorithms using both classical (shift and add) and newly proposed Montgomery product based

Phase II steps. These algorithms were coded using Microsoft Visual C++ 5.0 development system. The timing results are obtained on a 450-MHz Pentium II processor running the Windows NT 4.0 operating system. In Table 1, we summarize the timing results. The table contains the old and new Phase II timings (Old PhII and New PhII) in microseconds for operands of length 160 and 192 bits. The last two columns (PhII Spd and All Spd) give the speedup in Phase II only and the overall speedup, which illustrates the efficiency of the algorithms introduced.

An application of the new Montgomery inverse is found in computing eP, where e is an integer and P is a point on an elliptic curve defined over the finite field $GF(p)$. This computation requires that we perform elliptic curve point addition $P + Q$ and doubling $P + P = 2P$ operations, where each point operation requires a few modular additions and multiplications, and a modular inversion. The inverse operation is used to compute the variable $\lambda := (y_2 - y_1)(x_2 - x_1)^{-1} \pmod{p}$, which is required in computing elliptic curve point addition of $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in order to obtain $P + Q = (x_3, y_3)$. Assuming the input variables are given in the Montgomery domain, we would like to obtain the result in the Montgomery domain. If the Kaliski-Montgomery inverse is used, it will compute the classical inverse which is in the residue (non-Montgomery) domain, and cannot be readily used in subsequent operations. We need to perform a Montgomery product with $R^2 \pmod{p}$ in order to convert back to the Montgomery domain. However, with the help of the new Montgomery inverse, we can perform the above computation in a single step. Since these operations are performed for every bit of the exponent e, the new Montgomery inverse is more efficient and highly useful in this context.

References

[1] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, November 1976.

[2] Ö. Eğecioğlu and Ç. K. Koç. Exponentiation using canonical recoding. *Theoretical Computer Science*, 129(2):407–417, 1994.

[3] B. S. Kaliski Jr. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.

[4] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, Third edition, 1998.

[5] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[6] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA, 1993.

[7] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[8] National Institute for Standards and Technology. Digital signature standard (DSS). Federal Register, 56:169, August 1991.

[9] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, October 1982.

[10] M. Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications, Greenwich, CT, 1999.

[11] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. In D. Coppersmith, editor, *Advances in Cryptology — CRYPTO 95*, Lecture Notes in Computer Science, No. 973, pages 43–56. Springer-Verlag, Berlin, Germany, 1995.