The Montgomery Inverse and Its Applications

Burton S. Kaliski Jr.

***Abstract*— The *Montgomery inverse* of b modulo a is $b^{-1}2^n$ mod a, where n is the number of bits in a. The right-shifting binary algorithm for modular inversion is shown naturally to compute the new inverse in fewer operations than the ordinary modular inverse. The new inverse facilitates recent work by Koç on modular exponentiation and has other applications in cryptography.**

I. Introduction

Modular arithmetic plays an increasingly important role in cryptography. Many public-key cryptosystems [2], [10], [13] involve modular exponentiation. Modular inversion, the computation of b^{-1} mod a, has a part in exponentiation based on addition-subtraction chains [6], as well as other applications in such public-key systems.

Montgomery [9] describes a method for arithmetic modulo an odd integer a in which one represents an integer b as $b' = b2^n$ mod a, where n is the number of bits in a. (That is, where n satisfies $2^{n-1} \le a < 2^n$.) The *Montgomery product* of two integers b', c' is $b'c'2^{-n}$ mod a; this equals the Montgomery representation of the ordinary product bc mod a.

Montgomery multiplication involves ordinary multiplication, addition and subtraction, and division by 2^n, the latter being just a shift. The following algorithm computes the Montgomery product.

```
$d' \leftarrow b'c'$
$t \leftarrow wd'$ mod $2^n$, where $w = -a^{-1}$ mod $2^n$
$d' \leftarrow (d' + ta)/2^n$
if $d' \ge a$ then $d' \leftarrow d' - a$
```

The value w, which depends only on the modulus a, can be precomputed and therefore need not be counted among the operations to compute the product. Montgomery multiplication is generally faster than ordinary modular multiplication.

Though there is some overhead in converting into and out of the Montgomery representation, in a series of multiplications such as an exponentiation, the overhead occurs only once for the series, not for each multiplication. Several efficient implementations are based on this representation [1], [3], [7], [14].

II. Montgomery Inverse Algorithm

The right-shifting greatest common divisor algorithm [15] computes the gcd of two integers a and b with halvings (right shifts) and subtractions. A related algorithm, attributed to M. Penk ([5], exercise 4.5.2.35) also computes the modular inverse b^{-1} mod a. In Penk's algorithm, both even and odd values are "halved"; the halving of an odd value, working modulo a, is accomplished by first adding the modulus a, which is assumed to be odd. (A similar approach is described by J.L. Massey in his cryptography seminar [8].)

Another approach is to defer halvings related to the modular inverse until the end of the algorithm [4]. In such a case, the algorithm first computes $b^{-1}2^k$ mod a, where k is the number of deferred halvings, then halves k times modulo a. If $k \ge n$, interestingly, one of the intermediate values is the Montgomery representation of the modular inverse, i.e., $b^{-1}2^n$ mod a, or the *Montgomery inverse* for short.

Algorithm MontInverse (see Fig. 1) demonstrates this observation. The first phase of the algorithm computes gcd(a, b) and, assuming that a and b are relatively prime, an intermediate value $b^{-1}2^k$ mod a, where k is the number of iterations. The second phase halves the intermediate value $k - n$ times modulo a, then negates the result. If a and b are relatively prime, the final result is the Montgomery inverse. Fig. 2 gives an example.

```
Algorithm MontInverse
Inputs: $a$, $b$, $n$, where $a$ is odd, $a > b > 0$, and $n$ is the number of bits in $a$
Output: "Not relatively prime," or $b^{-1}2^n$ mod $a$

First phase
$u \leftarrow a$, $v \leftarrow b$, $r \leftarrow 0$, $s \leftarrow 1$
$k \leftarrow 0$
while $v > 0$ do
  if $u$ is even then $u \leftarrow u/2$, $s \leftarrow 2s$
  else if $v$ is even then $v \leftarrow v/2$, $r \leftarrow 2r$
  else if $u > v$ then $u \leftarrow (u - v)/2$, $r \leftarrow r + s$, $s \leftarrow 2s$
  else $v \leftarrow (v - u)/2$, $s \leftarrow r + s$, $r \leftarrow 2r$
  $k \leftarrow k + 1$
if $u \ne 1$ then return "Not relatively prime"
if $r \ge a$ then $r \leftarrow r - a$

Second phase
for $i \leftarrow 1$ to $k - n$ do
  if $r$ is even then $r \leftarrow r/2$
  else $r \leftarrow (r + a)/2$
return $a - r$
```

Fig. 1. Algorithm MontInverse computes the Montgomery inverse $b^{-1}2^n$ mod a.

First phase

u	v	r	s	k
17	10	0	1	0
17	5	0	1	1
6	5	1	2	2
3	5	1	4	3
3	1	2	5	4
1	1	7	10	5
1	0	14	17	6

Second phase

i	r
1	14
—	7

return $17 - 7 = 10$

Fig. 2. Algorithm MontInverse with a = 17, b = 10, and n = 5. One easily verifies that 10 is the Montgomery inverse of 10 modulo 17, since $10 \times 10 \equiv 2^5$ (mod 17).

Assuming that $k > n$ (see Theorem 2), Algorithm MontInverse computes the Montgomery inverse faster than the algorithms from which it is derived compute the ordinary modular inverse, since it has n fewer halvings. Thus, Algorithm MontInverse is significantly faster than the conventional approach of computing the ordinary modular inverse and converting the result to the Montgomery representation.

III. Analysis

Theorem 1 bounds the intermediate values in Algorithm MontInverse, Theorem 2 bounds the number of iterations, and Theorem 3 shows that the algorithm is correct.

Manuscript received June 28, 1993; revised June 10, 1994.
The author is with RSA Laboratories, RSA Data Security, Inc., 100 Marine Parkway, Suite 500, Redwood City, CA 94065; e-mail: burt@rsa.com.
IEEECS Log Number C95084.

THEOREM 1. *If $a > b > 0$, then the intermediate values r, s, u, and v in Algorithm* MONTINVERSE *are always between* 0 *and* $2a - 1$.

PROOF. The following invariants of the first phase's **while** loop can be verified by induction:

$$a = us + vr, \quad s \geq 1, \quad u \geq 1, \quad \text{and} \quad 0 \leq v \leq b.$$

It follows directly that $s \leq a$ and $u \leq a$. Until the last iteration, it is also the case that $v > 0$, so that $r < a$. The last iteration always computes $r \leftarrow 2r$, so after the last iteration, $r < 2a$.

During the second phase, r remains less than a, and the result follows. □

THEOREM 2. *If a and b are relatively prime, a is odd, and $a > b > 0$, then the number of iterations in the first phase of Algorithm* MONTINVERSE *is at least n and at most $2n$, where n is the number of bits in a.*

PROOF. Each iteration reduces the product uv by at least half and the sum $u + v$ by at most half. Also, since a and b are relatively prime, u and v are both 1 before the last iteration.

Since the product is initially ab and the sum is initially $a + b$, the number of iterations k satisfies

$$\frac{a+b}{2} \leq 2^{k-1} \leq ab.$$

Since $a > b > 0$ and $2^{n-1} \leq a < 2^n$, the number of iterations satisfies $2^{n-2} < 2^{k-1} < 2^{2n}$. Thus $n - 1 \leq k - 1 \leq 2n - 1$, and the result follows. □

THEOREM 3. *If a and b are relatively prime, a is odd, and $a > b > 0$, then Algorithm* MONTINVERSE *returns* $b^{-1}2^n \bmod a$.

PROOF. It can be verified by induction that at each iteration of the first phase, $\gcd(u, v) = \gcd(a, b)$. After the last iteration, $v = 0$, so $u = \gcd(a, b) = 1$.

The following invariants can also be verified by induction:

$$br \equiv -u2^k \pmod{a}$$

$$bs \equiv v2^k \pmod{a}.$$

Thus, after the last iteration of the first phase, $r \equiv -b^{-1}2^k \pmod{a}$. By Theorem 1, it takes at most one subtraction to bring r into the range 0 to $a - 1$. By Theorem 2, $k \geq n$, so the second phase is well defined. It is easily verified that the final value of r is $b^{-1}2^n \bmod a$. □

IV. APPLICATIONS

Modular exponentiation by addition-subtraction chains computes $b^c \bmod a$ by multiplying intermediate values starting with b and $b^{-1} \bmod a$. With Montgomery multiplication, one starts with $b2^n \bmod a$ and $b^{-1}2^n \bmod a$. Both approaches assume that $b^{-1} \bmod a$ exists, which is generally the case in cryptographic applications.

Koç [6] gives heuristics for addition-subtraction chains based on recoding the exponent as signed bits. For instance, the exponent $119 = (1110111)_2$ is recoded as $119 = (1000\bar{1}00\bar{1})_2$ where the $\bar{1}$ digits have weight -1. Scanning the exponent one bit at a time (the binary method of exponentiation), it takes 11 multiplications to compute $b^{119} \bmod a$ given only b, but only nine if $b^{-1} \bmod a$ is available.

In general, the improvement in performance is about eight percent with the binary method for exponentiation based either on ordinary multiplication or on Montgomery multiplication. Since Algorithm MONTINVERSE makes the Montgomery inverse more easily available, it facilitates exponentiation based on Montgomery multiplication.

RSA private-key operations with the Chinese Remainder Theorem [12], DSA signatures and verification [10], and ElGamal verification [2], which all involve modular inversion, can also benefit from the algorithm.

V. CONCLUSIONS

The right-shifting binary algorithm for modular inversion naturally computes the Montgomery inverse. This facilitates recent work on modular exponentiation, and has other applications in cryptography.

Further work will determine whether one can improve other modular inversion algorithms, such as those based on multiple-precision arithmetic [5] or on redundant representations [11], by similar techniques.

ACKNOWLEDGMENTS

Johnson C.-H. Yang helped me understand the binary modular inverse algorithm. The loop invariants were suggested to me by Peter Montgomery. Matt Robshaw, Benjamin Arazi, and the anonymous referees also offered helpful comments and suggestions on the article. I also thank God.

REFERENCES

[1] S.R. Dussé and B.S. Kaliski Jr., "A cryptographic library for the Motorola DSP56000," I.B. Damgård, ed., *Advances in Cryptology—Eurocrypt '90*, pp. 230–244, New York: Springer-Verlag, 1990.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Information Theory*, vol. 31, pp. 469–472, 1985.

[3] S. Even, "Systolic modular multiplication," A.J. Menezes and S.A. Vanstone, eds., *Advances in Cryptology'90.*, pp. 619–624, New York: Springer-Verlag, 1991.

[4] A. Guyot, "OCAPI: Architecture of a VLSI coprocessor for the GCD and the extended GCD of large numbers," *Proc. 10th IEEE Symp. Computer Arithmetic*, pp. 226–231, 1991.

[5] D.E. Knuth. *The Art of Computer Programming*, vol. 2. Reading, Mass.: Addison-Wesley, 2nd edition, 1981.

[6] Ç.K. Koç, "High-radix and bit recoding techniques for modular exponentiation," *Int'l J. Computer Mathematics*, vol. 40, nos. 3-4, pp. 139–156, 1987.

[7] D. Laurichesse and L. Blain, "Optimized implementation of RSA cryptosystem," *Computers & Security*, vol. 10, no. 3, pp. 263–267, May 1991.

[8] J.L. Massey, "Cryptography: Fundamentals and applications," Advanced Technology Seminars, Zurich, Switzerland, Feb. 1993.

[9] P.L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[10] Nat'l Inst. of Standards and Technology (NIST), *FIPS Publication 186: Digital Signature Standard*. May 19, 1994.

[11] S.N. Parikh and D.W. Matula, "A redundant binary Euclidean GCD algorithm," *Proc. 10th IEEE Symp. Computer Arithmetic*, pp. 220–225, 1991.

[12] J.-J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," *Electronics Letters*, vol. 18, no. 21, pp. 905–907, 1982.

[13] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Comm. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[14] M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography," *Proc. 11th IEEE Symp. Computer Arithmetic*, pp. 252–259, 1993.

[15] J. Stein, "Computational problems associated with Racah algebra," *J. Comp. Phys.*, vol. 1, pp. 397–405, 1967.