Applications of The Montgomery Exponent

Shay Gueron [1,3]
[1] Dept. of Mathematics,
University of Haifa, Israel
(shay@math.haifa.ac.il)

Or Zuk [2,3]
[2] Dept. of Physics of Complex Systems,
Weizmann Institute of Science, Israel
(or.zuk@weizmann.ac.il)

[3] Discretix Technologies, Israel

Abstract

We define here the Montgomery Exponent of order s, modulo the odd integer N, by $\mathrm{MEXP} = \mathrm{MEXP}(A, X, N, s) = A^X 2^{-s(X-1)} \pmod N$, and illustrate some properties and usage of this operator. We show how $A^X \pmod N$ can be obtained from $\mathrm{MEXP}(A, X, N, s)$ by one Montgomery multiplication. This provides a new modular exponentiation algorithm that uses one Montgomery multiplication less than the number required with the standard method. The resulting reduction in the computation time and code size is significant when the exponent X is short (e.g., modular squaring and RSA *verification). We also illustrate the potential advantage in performance and code size when known cryptographic applications are modified to allow for using* MEXP *as the analogue of modular exponentiation.*

Keywords: Montgomery multiplication, modular exponentiation, efficient implementations.

1 Introduction

The Montgomery multiplication ([9]), defined as $\mathrm{MMUL}(A, B, N) = AB2^{-n} \pmod N$, where n is the bitlength of the odd modulus N, is considered to be one of the most efficient methods for implementing modular arithmetic operations. It is therefore widely used in hardware implementations of public key cryptosystems, such as smartcards.

The Montgomery multiplication method has attracted substantial attention in attempts to improve the algorithm, or to efficiently use it as a building block for long operations (e.g., [14] [7]). Attacks that exploit the final (conditional) reduction step of the Montgomery multiplications have been proposed (e.g., [12]). Of particular interest were methods for chaining Montgomery multiplications in a way that the output and input share the same bounds, and methods for avoiding the final reduction step. Few examples are [1], [2], [13], [10], [5]. In a different context, the Montgomery inverse, $a^{-1}2^n \pmod N$, was defined in [4], and later revisited in [6]. This operator has some applications, and can also be viewed as the analogue of modular inversion, performed in the Montgomery domain.

The advantage of using Montgomery multiplication is greatly appreciated in real applications that compute modular exponents. The related procedure (details are given in Section 2) involves an initial conversion to the Montgomery domain and a final conversion to the real domain after the computations are carried out. It also requires precomputation and storage for a particular conversion constant. For long exponents (e.g., RSA private key operations), the overhead is negligible, but in other situations (e.g., RSA public key operations) it may not be the case.

The purpose of this paper is (a) to define the Montgomery Exponent operator which is analogous to modular exponentiation, but carried out completely in the Montgomery domain, (b) to demonstrate the potential applications of this operator.

2 Preliminaries and Notations

All the discussed quantities in this paper are positive integers, where the binary representation of an integer M is denoted by the $m = 1 + \lfloor \log_2 M \rfloor$ bits string $[M_{m-1}...M_1M_0]$, with M_0 being the least significant bit. For two integers X, Y, we write $X = Y \ \ (+N)$ if either $X = Y$ or $X = Y + N$, and the notation $X = Y \ \ (-N)$ is analogous. N denotes an odd integer (modulus).

Definition 1 (Non Reduced Montgomery Multiplication). *The Non Reduced Montgomery Multiplication of order s, of A, B, modulo (odd) N is* $\mathrm{NRMM} = \mathrm{NRMM}(A, B, N, s) = \big(AB + \big(-ABN^{-1} \pmod{2^s N}\big)\big)/2^s$.

Definition 2 (Conversions to/from Montgomery domain). *The image in the (order s) Montgomery domain, of an integer X (given in the real domain) is the integer $X' = \mathrm{NRMM}(X, H, N, s)$, where $H = 2^{2s} \pmod N$ is called the conversion constant. The image in the real domain, of an integer Y' (given in the Montgomery domain) is $Y = \mathrm{NRMM}(Y', 1, N, s)$*

Property 1 (see [5]). *Assume that $s \geq n+2$,*
(a) If $X < 2N$ then $X' = X2^s \pmod N$ $(+N)$.
(b) If $A, B < 2N$, then $\mathrm{NRMM}(A, B, N, s) < 2N$.
(c) If $Y' < 2N$ then $Y = \mathrm{NRMM}(Y', 1, N, s) = Y'2^{-s} \pmod N$.

From Property 1a, it follows that the conversion to the Montgomery domain, of inputs that are bounded by $2N$, maintains the same upper bound. Property 1c shows that the conversion back to the real domain, of inputs that bounded by $2N$, returns a reduced result bounded by N. Using the fact that $(AB)' = \mathrm{NRMM}(A', B', N, s)$, together with Property 1b it follows that modular exponentiation can be computed by the following steps:

```
Algorithm 1: (Modular exponentiation using NRMM)
    Input: $A$, $X$, $N$.
    Pre-Calculation: Choose $s \geq n+2$;
    compute $H = 2^{2s} \pmod N$.
    1.1 Convert $A$ to the Montgomery domain:
        $A' = \mathrm{NRMM}(A, H, N, s)$.
    1.2 Perform appropriate sequence of NRMM's (e.g.,
        square-and-multiply).
    1.3 Convert back to the real domain:
        $T = \mathrm{NRMM}(result, 1, N, s)$.
    1.4 Return $T$.
```

The advantage of using Algorithm 1 to compute $A^X \pmod N$ stems from the fact that the bit level algorithm for computing NRMM involves only additions and divisions by 2, and relies only on the least significant bit of the accumulator (S). This allows for computations using carry-save (redundant) arithmetic, which makes the implementation (especially in hardware) very efficient.

```
Algorithm 2: (A bit level algorithm for computing
    $\mathrm{NRMM}(A, B, N, s)$)
    Input: $N$ (odd), $A, B < 2N$, $s \geq n+2$
    2.1 $S = 0$
    2.2 For $i$ from 0 to $s-1$ do
    2.3     $S = \big(S + A_iB + (S_0 + A_iB_0)N\big)/2$
    2.4 End For
    2.5 Return $S$
```

Remark 1 (Modular exponentiation via classical MMUL). *The classical way of computing modular exponents by means of MMUL uses Algorithm 1 with $s = n$, and MMUL operations instead of NRMM. In this case, the MMUL's are computed by Algorithm 2 with the inclusion of the following reduction step after step 2.4: $if\ S > N\ then\ S = S - N$. This reduction step guarantees that $\mathrm{MMUL}(A, B, N) < N$ if $A, B < N$, so the result of MMUL has the same bound as its input, and the exponentiation algorithm can be applied. The drawback of this method is the need for repeated (conditional) reduction steps after each multiplication. These affect the performance and the cost of the related hardware implementation. Further, the conditional reductions turn out to leak undesired information which can be used for cryptographic attacks (e.g., [12]).*

3 The Montgomery Exponent

We are interested here in implementations of public key cryptosystems (performing modular exponentiation) that operate in environments having extremely limited hardware and software resources (e.g., smartcards). We note that although Algorithm 1 is a very efficient approach, it carries the overhead involved with the conversion to and from the Montgomery domain, at the performance and code size cost of two NRMM operations. Further, dedicated code is needed for computing the conversion constant H, and storage space is required for storing it.

In some situations, particularly when the exponent is short, and when code size is of major concern, this overhead is not necessarily negligible. This motivates the definition and use of an operator which is analogous to modular exponentiation, but carried out completely in the Montgomery domain, in a way that domain conversions are eliminated.

Definition 3 (The Montgomery Exponent). *(a) The Montgomery Exponent of order s, of A to the power X modulo N, is*

$$\mathrm{MEXP} = \mathrm{MEXP}(A, X, N, s) = A^X 2^{-s(X-1)} \pmod N = (A2^{-s})^X 2^s \pmod N. \tag{1}$$

(b) An algorithm for computing the Montgomery Exponent is the following:

```
Algorithm 3: (Computing MEXP)
    Input: N (odd), $A < 2N$, $X$, $x = 1 + \lfloor \log_2 X \rfloor$, $s \geq n+2$.
    Output: $\mathrm{MEXP}(A, X, N, s)$
    3.1 $T = A$
    For $i$ from $x-2$ to $0$ do
    3.2     $T = \mathrm{NRMM}(T, T, N, s)$
    3.3     If $X_i = 1$ then $T = \mathrm{NRMM}(T, A, N, s)$
    End For
    3.4 If $(T > N)$ $T = T - N$ (Final reduction step)
    3.5 Return $T$
```

(c) The Nonreduced Montgomery Exponent $\mathrm{NRMEXP} = \mathrm{NRMEXP}(A, X, N, s)$ is the result of Algorithm 3, when

the final reduction step (3.4) is omitted. It follows that $\mathrm{NRMEXP}(A, X, N, s) = \mathrm{MEXP}(A, X, N, s) \ (+N)$.

Lemma 1. *Algorithm 3 returns* $\mathrm{MEXP}(A, X, N, s)$.

Proof. We show the correctness of Algorithm 3 by induction. From steps 3.2 and 3.3 we conclude

$$T = T^{2+X_i} 2^{-s(1+X_i)} \pmod N \ (+N). \tag{2}$$

Since by definition, $X_{x-1} = 1$, we have, after step $(x-2)$,

$$T \equiv A^{2+X_{x-2}} 2^{-s(1+X_{x-2})} = A^{2X_{x-1}+X_{x-2}} 2^{-s(2X_{x-1}+X_{x-2}-1)} \pmod N. \tag{3}$$

Using 2 we can get by induction, after step i,

$$T \equiv A^{\sum_{j=i}^{x-1} 2^{j-i} X_j} 2^{-s(\sum_{j=i}^{x-1} 2^{j-i} X_j - 1)} \pmod N. \tag{4}$$

Taking $i = 0$ gives $T \equiv A^X 2^{-s(X-1)} \pmod N$. From Property $1b$ it follows that $T < 2N$ throughout the computations, thus the final reduction step 3.4 yields the desired result. ■

Remark 2 (A classical definition for MEXP**).** *The choice* $s \geq n+2$ *in Algorithm 3 guarantees that the reduction step is required only once, at the end of the* NRMM *sequence. Clearly, the same result is obtained if a reduction step is applied at the end of each multiplication (i.e., steps 3.2 and 3.3). With repeated reductions one can choose* $s = n$ *(and not* $s \geq n+2$*) and replace all* NRMM *operations by standard* MMUL*'s. We call the result of this computation, namely* $A^X 2^{-n(X-1)} \pmod N = (A2^{-n})^X 2^n \pmod N$*, a "classical Montgomery exponent". For obvious practical reasons, we use here* $s \geq n+2$ *for computing* MEXP.

Example 1. NRMEXP *and* MEXP *can be different even when* $A < N$*:*

$$\mathrm{MEXP}(111, 34, 119, 9) = 15$$

$$\mathrm{NRMEXP}(111, 34, 119, 9) = 134 = 15 \ (+119)$$

Remark 3 (Comparing Algorithms 1 and 3). *Compared with the cost of computing modular exponents (via Algorihtm 1),* MEXP *computation has the following advantages:*

1. *No need to (pre)compute a conversion constant (H).*
2. *No need to store H.*
3. *No need for conversions to/from the Montgomery domain (two* NRMM*'s).*
4. *Code size and HW control are reduced.*

4 Modular exponentiation with MEXP

Lemma 2 below, shows how to use the MEXP operator for modular exponentiation.

Lemma 2. *If* $G = 2^{sX} \pmod N$ *then*

$$A^X \pmod N = \mathrm{NRMM}(\mathrm{NRMEXP}(A, X, N, s), G) \ (-N). \tag{5}$$

Proof. Denote

$$T = \mathrm{NRMEXP}(A, X, N, s) = A^X 2^{-s(X-1)} \pmod N \ (+N). \tag{6}$$

Using the definition, we have

$$\mathrm{NRMM}(T, G, N, s) = A^X 2^{-s(X-1)} 2^{sX} 2^{-s} \pmod N \ (+N) = A^x \pmod N \ (+N). \tag{7}$$

■

Lemma 2 and Algorithm 3 provide an alternative exponentiation algorithm. Compared with Algorithm 1, the new method requires one less NRMM operation.

Note that a final reduction step is required because the result in (7) is not necessarily reduced. For example, $N = 119$ $A = 109$ $X = 26$ gives $A^X \pmod N = 109^{26} \pmod{119} = 2$. Using NRMEXP, we have $s = n+2 = 9$, $G = 106$, $\mathrm{NRMEXP}(109, 26, 119, 9) = 86$, and finally $\mathrm{NRMM}(86, 106, 119, 9) = 121 = 2 \ (+119)$. Indeed, one subtraction is required to obtain the reduced result. We point out that the cost of the (single) reduction step is much smaller than the cost of an NRMM operation.

The relative advantage in code size and performance depends on the length of the exponent. For example, RSA private key operations use long exponents and the saving is negligible. However, the related RSA public key operations use short exponents (typically $2^{16}+1$ having 17 bits), and eliminating one multiplication may be significant. The extreme case is modular squaring $A^2 \pmod N$.

4.1 Application 1: Modular Multiplication (and modular squaring)

Using lemma 2, the algorithm for computing modular multiplication reduces from three multiplications (Algorithm 1) to only two as follows: a. $T = \mathrm{NRMM}(A, B, N, s)$, b. $T = \mathrm{NRMM}(T, G, N, s)$. Note also that in this case, the pre-computed constant $G = 2^{2s}$, equals to the conversion constant H associated with the standard method.

The applications: various cryptographic applications use modular multiplication (typically modular squaring) as a core operation, and our proposed method can be more efficient. Two such examples are the Fiat-Shamir identification scheme [3], and the Rabin encryption system [11].

5 Using MEXP with Appropriately Changed Protocols

Here we show how the MEXP operator can be used as the analogue of modular exponentiation in cryptographic protocols, if these are modified appropriately. This provides a protocol which is more efficient than the original one. Also, we show in the examples that the appropriately changed protocols can be defined in a way that some of the final reductions of the NRMM operations can be eliminated.
We use the following lemma:

Lemma 3. *$\forall A \leq 2N, X, Y, s$, and $\Phi = \Phi(N)$ being Euler's Phi function, we have :*
a.

$$\mathrm{NRMEXP}(\mathrm{NRMEXP}(A, X, N, s), Y, N, s) =$$
$$\mathrm{NRMEXP}(A, XY, N, s) \ \ (\pm N)$$

b.

$$\mathrm{NRMEXP}(A, \Phi(N), N, s) = A \ \ (+N)$$

Proof. a.

$$\mathrm{NRMEXP}(\mathrm{NRMEXP}(A, X, N, s), Y, N, s) =$$
$$\mathrm{NRMEXP}(A^X 2^{-s(X-1)} \pmod N), Y, N, s) =$$
$$(A^X 2^{-s(X-1)})^Y 2^{-s(Y-1)} \pmod N =$$
$$A^{XY} 2^{-s(XY-1)} \pmod N =$$
$$\mathrm{NRMEXP}(A, XY, N, s) \pmod N \qquad (8)$$

b.

$$\mathrm{MEXP}(A, \Phi(N), N, s) = A^{\Phi(N)} 2^{s(1-\Phi(N))} \pmod N =$$
$$(A2^{-s})^{\phi(N)} 2^s \pmod N = A2^{-s}2^s \pmod N =$$
$$A \pmod N \qquad (9)$$

■

Corollary 1.

$$\mathrm{MEXP}(\mathrm{NRMEXP}(A, X, N, s), Y, N, s) =$$
$$\mathrm{MEXP}(\mathrm{NRMEXP}(A, Y, N, s), X, N, s) \qquad (10)$$

5.1 Application 2 : Fiat-Shamir Scheme

The FS identification scheme can be changed as follows.

Scheme 1: (Montgomery-Fiat-Shmair Identification Scheme)

(Step 0) Public key : $N = PQ$ for some primes P and Q, and $X_i = f(I, j_i) \quad i = 1, \dots k$, where I represents Alice's identity (e.g., biometric signatures), and the j_i's are chosen small numbers such that the X_i's are quadratic residues modulo N. Private key (known to Alice) is P, Q, and $S_i = \sqrt{X_i} \pmod N, \quad i = 1, \dots k$

Repeat the following t times:

Step 1: Alice chooses (uniformly) a random number $R \in [0, N-1]$, computes $Z = \mathrm{NRMM}(R, R, N, s)$ and sends Z to Bob.

Step 2: Bob sends Alice k Bits $e_1, \dots, e_k$.

Step 3: Alice sends $\alpha = \mathrm{NRMM}(R, \prod_{e_i=1} S_i^{e_i}, N, s)$ where here $\prod$ denotes a chain of $\mathrm{NRMM}(*, *, N, s)$ operations

Step 4: Upon receiving α, Bob verifies that $\mathrm{NRMM}(\mathrm{NRMM}(\alpha, \alpha, N, s), \prod_{e_i=1} X_i, N, s) = Z \ (+N)$

If all t verifications are correct, Bob accepts Alice's identity.

The computational cost of this proposed scheme is smaller than that of the FS scheme. Step 1 is done using only one Montgomery multiplications. Further, steps 3 and 4 become cheaper (compared with the original FS scheme) by using the MEXP algorithm, since these use up to k multiplications, where k is typically small. Step 4 is the only instance where (conditional) reduction is needed (i.e., there is no need for reduction by Alice). The saving in code size could be significant for Alice, since she does not have to perform conversions to and from the Montgomery domain. This could be of major importance, in situations when Alice has very limited resources (e.g. smartcard).

5.2 Application 3 : Rabin Encryption System

The Rabin encryption system [11] (used for encryption and signature) is similar to RSA. Its advantages are: efficient encryption, requiring only one modular square (decryption time is similar to that of RSA), and guaranteed security (i.e., breaking the Rabin system is as hard as factoring $N = PQ$, a fact that was not proved for RSA). The drawback of this system is that decryption is not unique (proper decryption needs to be chosen out of four results, and this can be achieved by some additional identification bits).

The Rabin encryption system can be changed to a Montgomery variant as described below.

Encryption system 1: (Montgomery-Rabin Encryption)

(Step 0) Public key : $N = PQ$ for some primes P and Q. Private key (known to Alice) is P and Q.

Step 1 : (Encrypt) : Alice takes her message, M, computes $C = \mathrm{NRMM}(M, M, N, s)$ and sends to Bob.

Step 2 : (Decrypt) Bob receives C and computes :
$M_P = \mathrm{NRMEXP}(C, (P + 1)/4, P, s)$
$M_Q = \mathrm{NRMEXP}(C, (Q + 1)/4, Q, s)$.
Then, Bob solves the four congruence systems:
$M = \pm M_Q \pmod P, M = \pm M_Q \pmod Q$,
each one giving a unique solution in the range $[0, N - 1]$ (according to the Chinese Remainder Theorem). One of these four solutions is the correct M (and can be identified, for example, by a sample byte).

Efficiency: the saving obtained by using $NRMEXPs$ (instead of modular exponent) is not significant in the decryption phase (Step 2). However, the encryption phased (Step 1) is computed using only one Montgomery multiplications. This can be useful in situations, where the verifying party has limited resources.

6 Conclusion

We have shown here how the MEXP operator can be applied, and illustrated a few useful examples. It should be understood that performance is not the only consideration when comparing between the standard and the proposed methods. In fact, code size saving may even be more important in our context. For example, the use of the shortened method for modular squaring, can save about one third of the required code. This can be significant in cases where the code for performing the necessary steps is stored in ROM.

We point out that in fact, all public key schemes that are based on modular arithmetic (e.g., DSA, or ECC over $GF(p)$) can be implemented in the Montgomery domain in similar ways, using Lemma 3. However, modified protocol can be used only with proprietary implementations. Such protocols not only skip the conversion steps to and from the Montgomery base, but also do not require storage the conversion constant (H).Again, code size is significantly reduced.

Finally, it is important to note that since there exists a (simple) one-to-one mapping between the real and the Montgomery domain, it is often easy to show that the security of the “Montgomery domain protocols” is equivalent to that of the original protocols.

References

[1] Blum, T., and Paar, C.: Montgomery modular exponentiation on reconfigurable hardware. In Proceedings of the 14th symposium on computer arithmetic, (1999) 70–77.

[2] Dusse, S. R. and Kaliski, B. S.: A Cryptographic Library for the Motorola DSP56000. Lect. Not. Comp. Sci. (EUROCRYPT 1990) **473** (1990) 230–244.

[3] Fiat A. and Shamir A. : How to Prove Yourself: Practical Solutions to Identification and Signature Problems. CRYPTO 1986: 186-194

[4] Kaliski, B. S.: The Montgomery Inverse and its applications. IEEE Transactions on Computers **44** (1995) 1064-1065.

[5] Gueron, S. Enhanced Montgomery Multiplication. Lect. Not. Comp. Sci. (CHES 2002) **2523** (2002) 46-56.

[6] Koç, Ç. K.: The Montgomery modular Inverse - Revisited. IEEE Transactions on Computers **49** (2000) 763–766.

[7] Tenca, A. F., and Koç, Ç. K.: A Scalable Architecture for Montgomery Multiplication. Lect. Not. Comp. Sci. (CHES 1999) **1717** (1999) 94–108.

[8] Menezes, A. J. Oorschot, P.C., and Vanstone,S.A.: Handbook of Applied Cryptography. CRC Press, New York (1997).

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

[10] Hachez, G., and Quisquater J. J.: Montgomery Exponentiation with no Final Subtractions: Improved Results. Lect. Not. Comp. Sci. (CHES 2000) **1965** (2000) 293–301.

[11] Rabin M. : Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Massachussets Institute of Technology, Cambridge, Technical Report, 1979.

[12] Schindler, F.: A timing attack against RSA with Chinese Remainder Theorem. Lect. Not. Comp. Sci. (CHES 2000) **1965** (2000) 110–124.

[13] Walter, C. D.: Montgomery exponentiation needs no final subtractions. Electronics Letters **35** (1992) 1831 –1832.

[14] Walter, C. D.: Montgomery’s Multiplication Technique: How to Make It Smaller and Faster.Lect. Not. Comp. Sci. (CHES 1999) **1717** (1999) 80–93.