MATHEMATICS OF COMPUTATION
Volume 73, Number 246, Pages 899–906
S 0025-5718(03)01543-6
Article electronically published on May 7, 2003

NEW FRAMEWORKS FOR MONTGOMERY'S MODULAR MULTIPLICATION METHOD

PHILIP B. MCLAUGHLIN, JR.

Abstract. We present frameworks for fast modular multiplication based on a modification of Montgomery's original method. For (fixed) large integers, our algorithms may be significantly faster than conventional methods. Our techniques may also be extended to modular polynomial arithmetic.

1. Introduction

In [3], Montgomery shows how to compute modulo an integer $N > 1$ which avoids division by N. He also gives a multiprecision algorithm which, for n-bit integers, runs in $O(n^2)$ time. We offer here frameworks for Montgomery's method which are more suitable for large integers. Our techniques are especially suited for use with Discrete Fourier Transform (DFT) methods for multiplication, since they require no zero-padding, i.e., no computation of any double-length products using $2n$-bit transforms.

Note: In this paper we use the notation $a = b \pmod{c}$ to mean $a \equiv b \pmod{c}$ and $0 \le a < c$.

2. Review and extension of the method

Review. We paraphrase Montgomery's main results from [3], while recasting his algorithm as a complete modular multiplication (as opposed to reduction) method:

Given $N > 1$, choose $R > N$ with $\gcd(R, N) = 1$. Find integers R^{-1} and N' such that $0 < R^{-1} < N$, $0 < N' < R$, and $RR^{-1} - NN' = 1$. Let $\mathbf{Z}_N$ denote the usual ring of integers modulo N, and let $\mathbf{Z}'_N$ denote a second ring over the same set with addition unchanged but multiplication defined by $a*b = abR^{-1} \pmod{N}$. Define $f : \mathbf{Z}_N \to \mathbf{Z}'_N$ by $f(x) = xR \pmod{N}$. It is easy to check that f is a ring isomorphism. Montgomery proved that $a*b$ can be computed as follows:

Function REDC(a, b). (Given $a, b \in \mathbf{Z}'_N$, returns $a*b$.)
(0) $T = ab$.
(1) $m = (T \pmod{R})N' \pmod{R}$.
(2) $t = (T + mN)/R$.
(3) If $t < N$, then return t, else return $t - N$.

It is easy to modify algorithms involving $\mathbf{Z}_N$ to use $\mathbf{Z}'_N$. See [3] for other details.

Received by the editor May 5, 2000 and, in revised form, July 2, 2002.
2000 *Mathematics Subject Classification.* Primary 11-04, 11Y16; Secondary 11A07, 11T55.
Key words and phrases. Modular arithmetic, multiplication, factorization, primality testing, polynomial arithmetic, public-key cryptography.

Extension. Observe that t in REDC satisfies the constraint

$$t \leq ((N-1)^2 + (R-1)N)/R = N + (N^2 - 3N + 1)/R.$$

Choose an integer Q which satisfies

$$N + (N^2 - 3N + 1)/R < Q/\gcd(Q, R). \tag{2.1}$$

We can now eliminate step (0) and replace steps (1) and (2) of REDC with

(1′) $m = abN' \pmod R$.
(2′) $t = ((ab + mN)/R) \pmod{(Q/\gcd(Q, R))}$.

In the next section we discuss particular choices for Q and R which will allow us to compute both m and t efficiently.

3. Computational frameworks

Variation 1. Let $R = 2^{gk} - 1$ and $Q = 2Q'$, where $Q' = 2^{g(k+1)} - 1$, g is a small constant (whose choice is discussed below), and k is sufficiently large to force $R > N$. In this case $\gcd(Q, R) = 2^g - 1$, $Q' - 2^g R = 2^g - 1$, and $Q'/(2^g - 1) > R$. Check that $\gcd(R, N) = 1$ and compute N'. We can multiply in $\mathbf{Z}'_N$ using:

Function $*$_operator_v1(a, b). (Given $a, b \in \mathbf{Z}'_N$, returns $a*b$.)
(1) $m = abN' \pmod R$.
(2) $S = (ab + mN) \pmod{Q'}$.
(3) $s = -2^g S \pmod{Q'}$.
(4) If $(ab + mN) \equiv s \pmod 2$, then $t = s/(2^g - 1)$,
else $t = (s + Q')/(2^g - 1)$.
(5) If $t < N$, then return t, else return $t - N$.

Proof. Q satisfies (2.1) since $2Q'/(2^g - 1) > 2R > 2N > (N + (N^2 - 3N + 1)/R)$. Now $Rt = (ab + mN)$, so $Rt \pmod{Q'} = S$. Then $-2^g Rt \equiv -2^g S \pmod{Q'} = s$. But $-2^g R \equiv (2^g - 1) \pmod{Q'}$, so $(2^g - 1)t \pmod{Q'} = s$. By (2.1), $(2^g - 1)t < 2Q'$, so $(2^g - 1)t = s$ or $(2^g - 1)t = s + Q'$. Since $Rt \equiv (2^g - 1)t \equiv (ab + mN) \pmod 2$, we must have $(2^g - 1)t = s$ when $(ab + mN) \equiv s \pmod 2$, and $(2^g - 1)t = s + Q'$ otherwise. □

One approach to steps (1) and (2) of Variation 1 is modular multiplication based on algebraic factorizations of R and Q'. For each step, we can compute (or, for N and N', precompute) factor residues for each argument, multiply/add modulo the relevant factors, and then reconstruct the result.

We can choose g so that one or more of the familiar identities

$$(x^2 - 1) = (x + 1)(x - 1), \qquad (x^3 - 1) = (x - 1)(x^2 + x + 1), \tag{3.1}$$
$$(x^3 + 1) = (x + 1)(x^2 - x + 1), \quad (4x^4 + 1) = (2x^2 + 2x + 1)(2x^2 - 2x + 1) \tag{3.2}$$

apply to Q' and R and, in turn, to their factors and subfactors. Also, either $2|k$ or $2|(k + 1)$, so one of the exponents will be a multiple of $2g$ and the other will be of the form $2gk' + g$. For example, if we choose $g = 4$, we can use the identities

$$(x^8 - 1) = (x^4 + 1)(x^2 + 1)(x + 1)(x - 1), \tag{3.3}$$
$$(16x^8 - 1) = (2x^2 + 2x + 1)(2x^2 - 2x + 1)(2x^2 + 1)(2x^2 - 1) \tag{3.4}$$

to factor R and Q' (or vice versa.) If we choose $g = 6$, we can use

$$(x^{12} - 1) = (x^2 + 1)(x^4 - x^2 + 1)(x \pm 1)(x^2 \pm x + 1), \tag{3.5}$$
$$(64x^{12} - 1) = (2x^2 + 1)(4x^4 - 2x^2 + 1)(2x^2 - 1)(4x^4 + 2x^2 + 1). \tag{3.6}$$

The referee notes the factorization

$$(64x^{12}+1) = (2x^2 \pm 2x + 1)((4x^4 + 2x^2 + 1) \pm (4x^3 + 2x)) \tag{3.7}$$

which follows from the two identities (3.2) above. If we set $g = 12$, we can use

$$(x^{24} - 1) = (x^4 + 1)(x^8 - x^4 + 1)(\text{factors of } x^{12} - 1), \tag{3.8}$$

$$(2^{12}x^{24} - 1) = (\text{factors of } 64x^{12} + 1)(\text{factors of } 64x^{12} - 1) \tag{3.9}$$

to factor Q' and R. Decomposition or reduction modulo the factors in these identities is well known to require only addition, subtraction, and shift operations. Recovery of residues modulo R and Q' is also straightforward; see Appendix C for some applicable reconstruction formulas. We offer a simple numerical example of Variation 1 in Appendix A.

Variation 2. Choose $R = 2^k - 1 > N$ and $Q = 2Q'$, where $Q' = 2^k + 1$. In this case $\gcd(Q, R) = 1$. Verify $\gcd(R, N) = 1$ and find N'. Multiply in $\mathbf{Z}'_N$ using:

Function $*$_operator_v2(a, b). (Given $a, b \in \mathbf{Z}'_N$, returns $a*b$.)

(1) $m = abN' \pmod R$.
(2) $S = (ab + mN) \pmod{Q'}$.
(3) $w = -S \pmod{Q'}$.
(4) If $2|w$, then $s = w/2$, else $s = (w + Q')/2$.
(5) If $(ab + mN) \equiv s \pmod 2$, then $t = s$, else $t = s + Q'$.
(6) If $t < N$, then return t, else return $t - N$.

Proof. As above, Q satisfies (2.1) and $Rt \pmod{Q'} = S$. $R \equiv -2 \pmod{Q'}$, so $2t \equiv -S \pmod{Q'} = w$. Step (3) removes the 2, so $t \pmod{Q'} = s$. Since $t < 2Q'$, either $t = s$ or $t = s + Q'$. But $Rt \equiv t \equiv (ab + mN) \pmod 2$, so $t = s$ when $s \equiv (ab + mN) \pmod 2$; otherwise $t = s + Q'$. □

As before, we could apply the algebraic factorization method to compute m and S. For example, choose $k = 12k' + 6$ and use identities (3.6) and (3.7). But we suggest a different approach to Variation 2 — Discrete Fourier Transforms (DFT's).

Suppose $a = a_0 + a_1B + \cdots + a_{P-1}B^{P-1}$ and $b = b_0 + b_1B + \cdots + b_{P-1}B^{P-1}$, with $0 \le a_i, b_i < B$, $0 \le i \le P - 1$. As discussed by Crandall and Fagin in [1], we can compute $ab \pmod{B^P - 1}$ from the cyclic convolution (without zero-padding) of the sequences $\{a_i\}$ and $\{b_i\}$. Similarly, we can use the negacyclic convolution to compute $ab \pmod{B^P + 1}$. But this is all that we need; compute m using cyclic convolutions and S using negacyclic convolutions. Set $k = k'P$, where k' and P are chosen so that transforms are as fast as possible for a given size of N.

Note that we can compute m either sequentially or "all at once". In the latter option, we multiply the transforms of a, b, and N' together in one step. This requires that we maintain enough precision to recover convolution sums of size $P^2(B-1)^3$, rather than the usual $P(B-1)^2$. For the S computation, if we maintain one extra bit of precision, we can recover sums of size $2P(B-1)^2$. This allows us to add the transform components of ab and mN prior to the final reverse transform. See Appendix B for a numerical example of Variation 2.

A practical speedup. In many algorithms (such as computing $b^c \pmod N$) a given argument b may appear in more than one modular multiplication. In the algebraic approach, we can compute the residues of bN' and b modulo the factors of R and Q', respectively, and save them for subsequent operations. Likewise, in

the DFT approach, we can reuse the cyclic transform of bN' (mod R) and the negacyclic transform of b.

The same idea extends to sums and differences of arguments. These occur, for example, in Montgomery's elliptic curve (ECM) factorization algorithm from [4]. Suppose that we know aN' (mod R) and bN' (mod R) (or their factor residues) and we need to compute $((a \pm b) \pmod N)N' \pmod R$. When $a < b$, we can use $(a - b + N)N' \equiv (aN' - bN' - 1) \pmod R$. When $a + b \geq N$, we can use $(a + b - N)N' \equiv (aN' + bN' + 1) \pmod R$. Keep in mind when using DFT's that the underlying sums or differences are not carry-propagated, which will affect the required precision and constraints.

4. Estimated running times

Algebraic approach. For either variation, it is easy to check that all of the overhead required to compute m and S (obtaining factor residues, product reductions, and reconstructions), as well as the final steps to recover t, is $O(n)$, where n is the number of bits in Q'. Let τ be the time required to compute a conventional $O(n^2)$ $n \times n$ multiplication. Then the time for an $n/2 \times n/2$ multiplication is about $\tau/4$, for $n/4 \times n/4$ about $\tau/16$, etc. Similarly, let τ' be the time for an $n \times n$ Karatsuba multiplication (an $O(n^{\lg 3})$ method; see Knuth [2], pp. 294-295). Then the $n/2 \times n/2$ time is about $\tau'/3$, $n/4 \times n/4$ about $\tau'/9$, etc.

Suppose $Q' = x^8 - 1$ and we use (3.3) to factor it. Then one set of $O(n^2)$ factor residue products will take about $\tau(1/4 + 1/16 + 2/64) = 11\tau/32 \approx 0.34\tau$. The corresponding $O(n^{\lg 3})$ time is about $\tau'(1/3 + 1/9 + 2/27) = 14\tau'/27 \approx 0.52\tau'$. We computed similar estimates for identities (3.4) through (3.9); the results (normalized using division by τ or τ') are summarized in Table 1.

Recall from §3 (Variation 1) that the identities (3.3) and (3.4) apply when $g = 4$, (3.5) and (3.6) when $g = 6$, and (3.8) and (3.9) when $g = 12$. In each case, the first identity will be used for R when k is even. For any pair of identities, we need two sets of factor residue products each to compute m and S, unless an argument is reused, which eliminates one set of products modulo R. Reusing an argument will save the most time when the slower identity within each pair, namely (3.3), (3.6), or (3.8), is used to compute modulo R. Thus we should choose k to be even when $g = 4$ or $g = 12$ and odd when $g = 6$, unless doing so somehow imposes a significant time penalty.

Estimated total multiplication times (again normalized) to compute $a * b$ are summarized in Table 2. When an argument is reused, the time (for optimal parity of k in the Variation 1 cases) is shown in parentheses.

Note that the Variation 2 case has multiplication times only marginally faster than the $g = 4$ case of Variation 1, but we included it for comparison. All of the $O(n^2)$ cases compare favorably with the standard multiplication/long division method, as well as Montgomery's original algorithm, which requires about 2τ to compute $a*b$. The $O(n^{\lg 3})$ times all appear to be faster than a conventional three-multiply approach. Of course in practice the overhead will be significant, and we could use different multiplication methods for factors of different sizes. In fact, N would have to be fairly large (perhaps several thousand bits) to apply the Karatsuba method to all factor residue products. For such N the DFT approach may be faster. Clearly, for any given machine, experiments will be needed to find optimal choices for g, methods, etc., for N of various sizes.

TABLE 1. Approximate time to compute one set of products.

	(3.3)	(3.4)	(3.5)	(3.6)	(3.7)	(3.8)	(3.9)
$O(n^2)$	0.344	0.250	0.208	0.278	0.278	0.191	0.139
$O(n^{\lg 3})$	0.519	0.444	0.390	0.467	0.467	0.364	0.312

TABLE 2. Approximate multiplication time required to compute $a*b$.

	V1, $g=4$	V1, $g=6$	V1, $g=12$	V2, $k=12k'+6$
$O(n^2)$	1.188 (0.844)	0.972 (0.694)	0.660 (0.469)	1.111 (0.833)
$O(n^{\lg 3})$	1.926 (1.407)	1.714 (1.247)	1.350 (0.987)	1.870 (1.402)

DFT approach. Suppose that the time required for an $n\times n$ multiplication to form a $2n$-bit product using DFT's is $3T$, where T is the time required for one forward (or reverse) transform, and overhead time is negligible. Such a calculation generally requires zero-padding, so that the transform components can accommodate $2n$ bits. Here we never need a $2n$-bit product, only n-bit convolutions. In our case one convolution should take about $1.5T$. Our function requires four convolutions, but the transforms of N and N' can be precomputed. And, as mentioned previously, we can add the transform components of ab and mN prior to the final reverse transform. So the total time required to compute $a*b$ is about $4.5T$ ($3.5T$ for squares). When an argument is reused, the time drops to only $2.5T$. In contrast, a standard three-multiply approach to compute $ab \pmod N$ using $2n$-bit DFT's will require time of about $7T$ ($6T$ for squares or when an argument is reused) since the transforms of N and a scaled estimate of $1/N$ can be precomputed.

5. Modular multiplication of polynomials

The referee has observed that our methods may be extended to polynomial arithmetic. Here we outline the algorithm analogous to Variation 2 in §3.

Suppose $p(x)$ is a polynomial with $\deg(p(x)) \le h$, over a field of odd characteristic, where h has been chosen so that $\gcd(p(x), x^h - 1) = 1$ and arithmetic modulo $x^h \pm 1$ is easy. Precompute $\tilde{p}(x)$ such that $p(x)\tilde{p}(x) \equiv 1 \pmod{x^h - 1}$ and $\deg(\tilde{p}(x)) < h$. Given $a(x)$ and $b(x)$ with $\deg(a(x)) < \deg(p(x))$ and $\deg(b(x)) < \deg(p(x))$, suppose we want

$$c(x) \equiv a(x)b(x)(x^h - 1)^{-1} \pmod{p(x)}, \quad \text{with } \deg(c(x)) < \deg(p(x)).$$

We can compute

$$m(x) \equiv a(x)b(x)\tilde{p}(x) \pmod{x^h - 1}, \quad \text{with } \deg(m(x)) < h.$$

Follow this with

$$s(x) \equiv (m(x)p(x) - a(x)b(x)) \pmod{x^h + 1}, \quad \text{with } \deg(s(x)) < h.$$

Finally, $c(x) = 2^{-1}s(x)$.

Proof. Set $S(x) = m(x)p(x) - a(x)b(x)$. Then $S(x) \equiv s(x) \pmod{x^h + 1}$, $S(x) \equiv 0 \pmod{x^h - 1}$, $\deg(S(x)) < \deg(p(x)) + h$, and $S(x) \equiv -a(x)b(x) \pmod{p(x)}$. Set $S(x) = (x^h - 1)t(x)$. Since $t(x) \equiv -a(x)b(x)(x^h - 1)^{-1} \pmod{p(x)}$ and $\deg(t(x)) < \deg(p(x))$, we have $t(x) = -c(x)$. Thus $(x^h - 1)t(x) \equiv 2c(x) \equiv s(x) \pmod{x^h + 1}$. But $\deg(c(x)) < h$ and $\deg(s(x)) < h$, so $2c(x) = s(x)$. □

Nussbaumer ([5], Appendix B) has many sequences for multiplying two polynomials modulo some fixed univariate or bivariate polynomial, especially when one of the input polynomials will be reused.

Appendix A: A simple numerical example of Variation 1 using algebraic factorization

Let $N = 4000000003$. Choose $g = 4$ and $k = 8$, so that $R = 2^{32} - 1$. Compute $R^{-1} = 2767083981$ and $N' = 2971133798$. Finally, $Q' = 2^{36} - 1$ and $\gcd(Q', R) = 2^4 - 1$. Factor R and Q' using (3.3) and (3.4):

$$R = (2^{16} + 1)(2^8 + 1)(2^4 + 1)(2^4 - 1) = r_1 \cdot r_2 \cdot r_3 \cdot r_4,$$
$$Q' = (2^9 + 2^5 + 1)(2^9 - 2^5 + 1)(2^9 + 1)(2^9 - 1) = q_1 \cdot q_2 \cdot q_3 \cdot q_4.$$

Note that both R and Q' have more algebraic factors, but the above is sufficient to illustrate the method. Compute

$$\begin{array}{llll} N' \bmod r_1 = 13903, & N' \bmod r_2 = 231, & N' \bmod r_3 = 6, & N' \bmod r_4 = 8, \\ N \bmod q_1 = 298, & N \bmod q_2 = 155, & N \bmod q_3 = 493, & N \bmod q_4 = 335. \end{array}$$

Now suppose that $a = 3987997002$ and $b = 3796466986$. Decomposing a and b modulo the factors of R and Q', we have

$$\begin{array}{llll} a \bmod r_1 = 5015, & a \bmod r_2 = 16, & a \bmod r_3 = 16, & a \bmod r_4 = 12, \\ a \bmod q_1 = 377, & a \bmod q_2 = 28, & a \bmod q_3 = 153, & a \bmod q_4 = 213, \\ b \bmod r_1 = 39650, & b \bmod r_2 = 21, & b \bmod r_3 = 7, & b \bmod r_4 = 1, \\ b \bmod q_1 = 256, & b \bmod q_2 = 364, & b \bmod q_3 = 226, & b \bmod q_4 = 151. \end{array}$$

Multiplying and reducing modulo the factors of R, we get

$$abN' \bmod r_1 = 4671, \quad abN' \bmod r_2 = 2, \quad abN' \bmod r_3 = 9, \quad abN' \bmod r_4 = 6.$$

Reconstructing modulo R, $m = abN' \pmod R = 3527206011$. Decomposing m modulo the factors of Q', we get

$$m \bmod q_1 = 346, \quad m \bmod q_2 = 303, \quad m \bmod q_3 = 126, \quad m \bmod q_4 = 406.$$

Multiplying, summing terms, and reducing, we have

$$\begin{array}{ll} (ab + mN) \bmod q_1 = 150, & (ab + mN) \bmod q_2 = 399, \\ (ab + mN) \bmod q_3 = 252, & (ab + mN) \bmod q_4 = 54. \end{array}$$

Reconstructing, $S = 40860178875$. Then $s = -2^4 S \pmod{Q'} = 33431905350$. Since $(ab + mN) \not\equiv s \pmod 2$, $t = (s + Q')/(2^4 - 1) = 6810092139$. Finally, $t \geq N$, so $a * b = t - N = 2810092136$. As a check, we computed $abR^{-1} \pmod N$ directly and obtained the same result.

Appendix B: A simple numerical example of Variation 2 using DFT's

Suppose $R = B^P - 1$ and $Q' = B^P + 1$, where $B = 2^u$ and $P = 2^v$. We choose here to use transforms over a ring $\mathbf{Z}_M$. To avoid overflow, we require $M > 2P(B-1)^2$. (We include an extra factor of 2 so that we can add the transforms of ab and mN before applying the final reverse transform to obtain $(ab + mN) \pmod{Q'}$.) Let $M = 2^{cP} + 1$, where $c \geq (v + 2u + 1)/P$. Then $A = 2^c$ and $\omega = 2^{2c}$ will be primitive

Pth roots of -1 and 1, respectively, in $\mathbf{Z}_M$. As shown in [1], for cyclic convolutions the forward and reverse transforms are given by

$$X_n = \sum_{k=0}^{P-1} x_k \omega^{-kn} \pmod{M}, \qquad x_n = P^{-1} \sum_{k=0}^{P-1} X_k \omega^{kn} \pmod{M},$$

respectively, for $0 \le n \le P-1$. Also from [1], for negacyclic convolutions the forward and reverse transforms are given by

$$X'_n = \sum_{k=0}^{P-1} x_k A^k \omega^{-kn} \pmod{M}, \qquad x_n = P^{-1} A^{-n} \sum_{k=0}^{P-1} X'_k \omega^{kn} \pmod{M},$$

respectively, for $0 \le n \le P-1$. Also, the result of the final reverse negacyclic transform to obtain $(ab + mN) \pmod{Q'}$ will be subject to the constraints

(B1) $$-2(P-1-n)(B-1)^2 \le x_n \le 2(n+1)(B-1)^2, \quad 0 \le n \le P-1.$$

Now suppose $N = 3141592661$. Then we can choose $R = 2^{32} - 1$, $Q' = 2^{32} + 1$, $P = 2^2$, and $B = 2^8$. We then have $c \ge (2 + 16 + 1)/4$, so $c = 5$ and $M = 2^{20} + 1$, $A = 2^5$, and $\omega = 2^{10}$. We find $R^{-1} = 660970531$ and $N' = 903633004$. Let $T(x)$ and $T'(x)$ denote the cyclic and negacyclic transforms, respectively, of an integer $x = x_0 + x_1 B + x_2 B^2 + x_3 B^3$, or $x = (x_0, x_1, x_2, x_3)$ for short. Precompute

$$N = (85, 230, 64, 187), \qquad T'(N) = (957712, 780541, 222107, 137134),$$
$$N' = (108, 88, 220, 53), \qquad T(N') = (469, 1012625, 187, 35728).$$

Now let $a = 519910555$ and $b = 2438952723$. First, compute $m = abN' \pmod{R}$:

$a = (155, 52, 253, 30)$,
$b = (19, 119, 95, 145)$,
$T(a) = (490, 1025951, 326, 22430)$,
$T(b) = (378, 26548, 1048427, 1021877)$,
$T(b) \cdot T(N') = (177282, 799951, 1020527, 267470)$,
$T^{-1}(T(b) \cdot T(N')) = (42019, 51459, 32597, 51207)$,
$bN' \pmod{R} = (235, 167, 30, 135)$ (after carries mod R),
$T(bN' \pmod{R}) = (567, 1016014, 1048540, 32973)$,
$T(a) \cdot T(bN' \pmod{R}) = (277830, 669384, 1036515, 337605)$,
$T^{-1}(T(a) \cdot T(bN' \pmod{R})) = (56045, 73160, 76839, 71786)$,
$m = abN' \pmod{R} = (6, 164, 69, 151)$ (after carries mod R).

Now compute $(ab + mN) \pmod{Q'}$:

$T'(a) = (195354, 133341, 323100, 397402)$,
$T'(b) = (658159, 193015, 585016, 661040)$,
$T'(m) = (829570, 842028, 360331, 65249)$,
$T'(a) \cdot T'(b) = (627277, 539227, 82426, 719424)$,
$T'(m) \cdot T'(N) = (176749, 847895, 446469, 348825)$,
$T'(a) \cdot T'(b) + T'(m) \cdot T'(N) = (804026, 338545, 528895, 19672)$,
$(T')^{-1}(T'(a) \cdot T'(b) + T'(m) \cdot T'(N)) = (-101504, -27349, 37102, 98415)$
(after constraints (B1) applied),
$S = (ab + mN) \pmod{Q'} = (0, 157, 129, 255) = 4286684416$
(after carries mod Q').

Next, compute $t \pmod{Q'}$:

$2t \equiv -S \pmod{Q'} = Q' - 4286684416 = 8282881$,
$t \pmod{Q'} = (8282881 + Q')/2 = 2151625089$.

Finally, $(ab + mN) \equiv 1 \pmod 2$, so $t = 2151625089$. Since $t < N$, $a*b = t$. As a check, we computed $a*b = abR^{-1} \pmod N$ directly and obtained the same result.

Appendix C: Algebraic reconstruction formulas

The following formulas can be used to reconstruct $P \pmod{p(x)}$ from P modulo factors of $p(x)$, where $x = 2^h$. In each formula, m_1 and m_2 are given factors, $m_1 > m_2$, $P \pmod{m_1} = A$ and $P \pmod{m_2} = B$.

$(x^2 - 1)$: $m_1, m_2 = (x \pm 1)$. Compute $C = (B - A) \pmod{m_2}$ and $r = C \pmod 2$. Then $P \pmod{(x^2 - 1)} = A + m_1((C + m_2 r)/2)$.

$(x^6 + 1)$: $m_1 = (x^4 - x^2 + 1)$ and $m_2 = (x^2 + 1)$. Compute $C = (B - A) \pmod{m_2}$ and $r = C \pmod 3$. Then $P \pmod{(x^6 + 1)} = A + m_1((C + m_2 r)/3)$.

$(4x^4 + 1)$: $m_1 = (2x^2 + 2x + 1)$ and $m_2 = (2x^2 - 2x + 1)$. Find C_0 and C_1, where $(2xC_1 + C_0) = (A - B) \pmod{m_2}$. Compute $D = ((x - 1)C_0 - C_1) \pmod{m_2}$ and $r = D \pmod 2$. Then $P \pmod{(4x^4 + 1)} = A + m_1((D + m_2 r)/2)$.

$(16x^8 - 4x^4 + 1)$: $m_1, m_2 = ((4x^4 + 2x^2 + 1) \pm (4x^3 + 2x))$. Find C_0 and C_1, where $((4x^3+2x)C_1+C_0) = (A-B) \pmod{m_2}$. Compute $D = ((x-1)C_0-C_1) \pmod{m_2}$ and $r = D \pmod 2$. Then $P \pmod{(16x^8 - 4x^4 + 1)} = A + m_1((D + m_2 r)/2)$.

$(8x^6 - 1)$: $m_1 = (4x^4 + 2x^2 + 1)$ and $m_2 = (2x^2 - 1)$. Find $C = (B - A) \pmod{m_2}$ and $r = 2C \pmod 3$. Then $P \pmod{(8x^6 - 1)} = A + m_1((C + m_2 r)/3)$.

$(8x^6 + 1)$: $m_1 = (4x^4 - 2x^2 + 1)$ and $m_2 = (2x^2 + 1)$. Here $3 \parallel m_1$ and $3^{k-1} \parallel m_2$ for some $k \geq 2$. Since $\gcd(m_1, m_2) = 3$, we need to know $P \pmod{3^k} = C$ in addition to A and B. We also need to precompute $u = (m_2/3^{k-1}) \pmod 3$. Find $D = (B-A) \pmod{m_2}$, $E = (A+D) \pmod{3^k}$, and $r = u((C-E)/3^{k-1}) \pmod 3$. Then $P \pmod{(8x^6 + 1)} = A + m_1((D + m_2 r)/3)$.

Acknowledgment

The author would like to thank the referee for suggesting many valuable improvements to this paper.

References

1. Richard Crandall and Barry Fagin, *Discrete weighted transforms and large-integer arithmetic*, Math. Comp. **62** (1994), 305-324. MR **94c:**11123
2. Donald E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (3rd ed.), Addison-Wesley, Boston, MA, 1998. MR **83i:**68003
3. Peter L. Montgomery, *Modular multiplication without trial division*, Math. Comp. **44** (1985), 519-521. MR **86e:**11121
4. Peter L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. **48** (1987), 243-264. MR **88e:**11130
5. H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, (2nd ed.), Springer-Verlag, New York, 1982. MR **83e:**65219

237 N. Harris Avenue, Tucson, Arizona 85716
E-mail address: pbmcl@netscape.net