Integer and Prime Field Arithmetic

October 17, 2022

Ahmet Can Mert
ahmet.mert@iaik.tugraz.at

Modular Reduction Algorithms

- Well-known modular reduction methods:
  - Barrett reduction
  - Montgomery reduction

- Reduction for special primes

Barrett Modular Reduction Algorithm

- An algorithm for computing $C = A \cdot B \pmod{q}$ where A, B, and q are k-bit numbers

IMPLEMENTING THE
RIVEST SHAMIR AND ADLEMAN
PUBLIC KEY ENCRYPTION ALGORITHM
ON A
STANDARD DIGITAL SIGNAL PROCESSOR

Paul Barrett, MSc (Oxon)
COMPUTER SECURITY LTD
August 1986

ABSTRACT

A description of the techniques employed at Oxford University to obtain a high speed implementation of the RSA encryption algorithm on an "off-the-shelf" digital signal processing chip. Using these techniques a two and a half second (average) encrypt time (for 512 bit exponent and modulus) was achieved on a first generation DSP (The Texas Instruments TMS 32010) and times below one second are achievable on second generation parts. Furthermore the techniques of algorithm development employed lead to a provably correct implementation.

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod{q}$, $a < q^2$, $2^{k-1} < q < 2^k$

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod{q}$, $a < q^2$, $2^{k-1} < q < 2^k$

$a \pmod{q} = a - \lfloor a/q \rfloor \cdot q$ $(s = 1/q)$ Division is expensive

$a \pmod{q} = a - \lfloor a \cdot s \rfloor \cdot q$ Result will be exact with s having enough prec.

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod{q}$, $a < q^2$, $2^{k-1} < q < 2^k$

$a \pmod{q} = a - \lfloor a/q \rfloor \cdot q$ $(s = 1/q)$ Division is expensive

$a \pmod{q} = a - \lfloor a \cdot s \rfloor \cdot q$ Result will be exact with s having enough prec.

$s = 1/q = m/2^{2k}$, so $m = 2^{2k}/q$ Based on rounding $2^{2k}/q$, $(m/2^{2k} > 1/q)$ could hold.

So, there's chance of underflow.

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod q$, $a < q^2$, $2^{k-1} < q < 2^k$

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ $(s = 1/q)$ Division is expensive

$a \pmod q = a - \lfloor a \cdot s \rfloor \cdot q$ Result will be exact with s having enough prec.

$s = 1/q = m/2^{2k}$, so $m = 2^{2k}/q$ Based on rounding $2^{2k}/q$, $(m/2^{2k} > 1/q)$ could hold. So, there's chance of underflow.

$\mathrm{m} = \lfloor 2^{2k}/q \rfloor$, so $s = \lfloor 2^{2k}/q \rfloor / 2^{2k}$

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod q$, $a < q^2$, $2^{k-1} < q < 2^k$

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ $\quad (s = 1/q)$ Division is expensive

$a \pmod q = a - \lfloor a \cdot s \rfloor \cdot q$ Result will be exact with s having enough prec.

$s = 1/q = m/2^{2k}$, so $m = 2^{2k}/q$ Based on rounding $2^{2k}/q$, ($m/2^{2k} > 1/q$) could hold. So, there's chance of underflow.

$m = \lfloor 2^{2k}/q \rfloor$, so $s = \lfloor 2^{2k}/q \rfloor / 2^{2k}$

$$a \pmod q = a - \lfloor a \cdot s \rfloor \cdot q$$

$$= a - \lfloor a \cdot (\lfloor 2^{2k}/q \rfloor / 2^{2k}) \rfloor \cdot q$$

$\lfloor x \rfloor = x - e, \quad 0 \le e < 1$

Barrett Modular Reduction Algorithm

- Reduction: $a \pmod q$, $a < q^2$, $2^{k-1} < q < 2^k$

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ $\quad (s = 1/q)$ Division is expensive

$a \pmod q = a - \lfloor a \cdot s \rfloor \cdot q$ Result will be exact with s having enough prec.

$s = 1/q = m/2^{2k}$, so $m = 2^{2k}/q$ Based on rounding $2^{2k}/q$, $(m/2^{2k} > 1/q)$ could hold. So, there's chance of underflow.

$m = \lfloor 2^{2k}/q \rfloor$, so $s = \lfloor 2^{2k}/q \rfloor / 2^{2k}$

$a \pmod q = a - \lfloor a \cdot s \rfloor \cdot q$

$= a - \lfloor a \cdot (\lfloor 2^{2k}/q \rfloor / 2^{2k}) \rfloor \cdot q$ $\quad \lfloor x \rfloor = x - e, \quad 0 \le e < 1$

…

$a \pmod q = e_1 \cdot (a/2^{2k}) \cdot q - e_2 \cdot q$ $\quad a \pmod q < 2 \cdot q$ (final subtraction is needed)

Barrett Modular Reduction Algorithm

- Takes $D = A \cdot B$ as input and generates $C = D \pmod{q}$
  - $A, B < q, D = A \cdot B < q^2$
  - $2^{k-1} < q < 2^k$
  - $\mu = \lfloor 2^{2k}/q \rfloor$

Input: $D = A \cdot B, q, \mu$
Output: $C = D$ (mod q)
1: $s = (D \cdot \mu) >> 2k$
2: $r = s \cdot q$
3: $u = D - r$
4: if $(u \geq q)$ **then** $C = u - q$ **else** $C = u$
5: return C

Barrett Modular Reduction Algorithm

- Try Barrett algorithm in `sage`.
  - https://sagecell.sagemath.org/

```
k = 5
q = 19
mu= 2^(2*k) // q

D = 120

u = D - ((D*mu) >> 2*k)*q
u = u-q if(u >= q) else u

print("D mod q:", D%q)
print("BR(D,q):", u)
```

Run the code.

Montgomery Modular Reduction Algorithm

MATHEMATICS OF COMPUTATION
VOLUME 44, NUMBER 170
APRIL 1985, PAGES 519–521

Modular Multiplication Without Trial Division

By Peter L. Montgomery

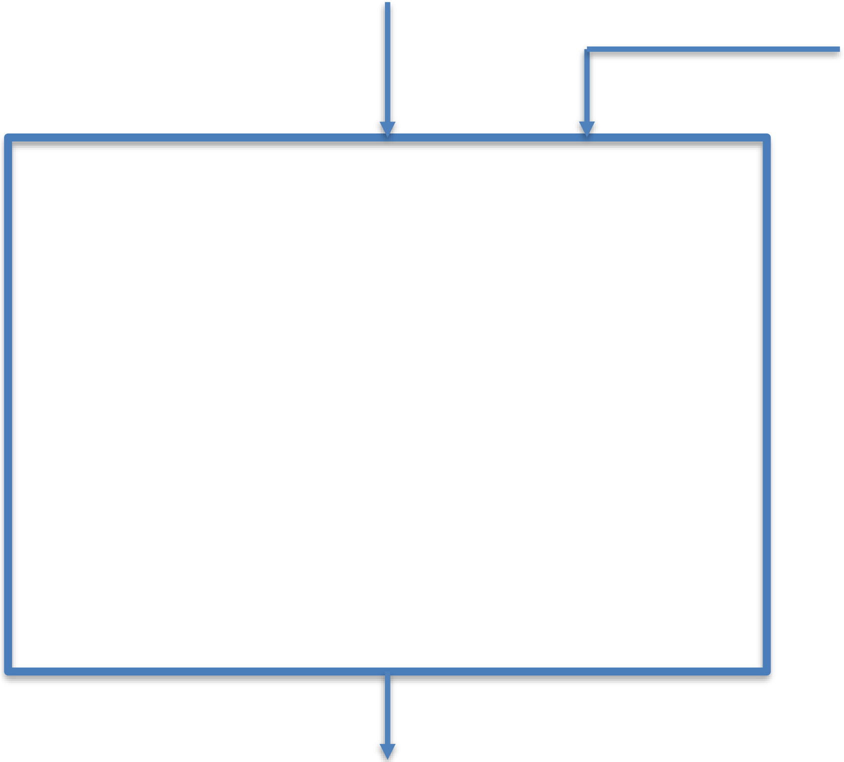
Abstract. Let $N > 1$. We present a method for multiplying two integers (called *N-residues*) modulo N while avoiding division by N. N-residues are represented in a nonstandard way, so this method is useful only if several computations are done modulo one N. The addition and subtraction algorithms are unchanged.

Replaces division by *q* by cheaper division by power-of-2

Montgomery Modular Reduction Algorithm

Some integer T in $[0, q^2-1]$ and q is an odd

R is a power-of-2 and just larger than q i.e., $R/2 < q < R$



Result is $T*R^{-1} \bmod q$

Montgomery Modular Reduction Algorithm

Some integer T in $[0, q^2\text{-}1]$ and q is an odd

R is a power-of-2 and just larger than q i.e.,
$R/2 < q < R$

1. $m \leftarrow ((T \bmod R) * q') \bmod R$

$q' = -q^{-1} \bmod R$
is precomputed constant

Result is $T*R^{-1} \bmod q$

Explanation(1):
This step is like performing a 'division' by $-T/q$ in the mod R domain.

Montgomery Modular Reduction Algorithm

Some integer T in $[0, q^2\text{-}1]$ and q is an odd

R is a power-of-2 and just larger than q i.e., $R/2 < q < R$

$q' = -q^{-1} \bmod R$ is precomputed constant

1. $m \leftarrow ((T \bmod R)*q') \bmod R$
2. $t \leftarrow (T + m*q) / R$

Result is $T*R^{-1} \bmod q$

Explanation(2):

$(T + m*q)$ is divisible by q because

$T + m*q = T + T*(-q^{-1})*q \pmod R$

$= 0 \pmod R$.

Hence, t is an integer $\approx T/R$.

Montgomery Modular Reduction Algorithm

Some integer T in $[0, q^2-1]$ and q is an odd

R is a power-of-2 and just larger than q i.e.,
$R/2 < q < R$

1. $m \leftarrow ((T \bmod R) * q') \bmod R$
2. $t \leftarrow (T + m*q) / R$
3. If $t > q$ then return $(t-q)$

   Else return t

$q' = -q^{-1} \bmod \mathrm{R}$
is precomputed constant

Result is $T*R^{-1} \bmod q$

Explanation(3):
As q and R are coprime, taking
$t \pmod q = (T + m*q)*R^{-1} \bmod q$
$= T*R^{-1} \bmod q$

Montgomery Modular Reduction Algorithm: Classical Montgomery

- Try Montgomery algorithm in `sage`.
  - https://sagecell.sagemath.org/

```
q = 19
R = 2^5
q_inv= -q^(-1) % R

T = 129

u = (T + (T*q_inv % R)*q)/R
u = u-q if(u >= q) else u

print("D mod q:", T%q)
print("MR(D,q):", u)
print("u*R mod q:", u*R % q)
```

Run the code.

When to use Montgomery Reduction Algorithm?

Given T as input it produces $T*R^{-1}$ mod q

→ We need an ***additional*** multiplication by R to get T mod q.

→ More expensive than Barrett reduction

Modular exponentiation in RSA

$$c = m^e \bmod N$$

… here we do all operations in the mod *N* ring.

Efficiency trick:

Let *a* and *b* are two integers modulo *N*.

Modular exponentiation in RSA

$$c = m^e \bmod N$$

… here we do all operations in the mod *N* ring.

Efficiency trick:
Let *a* and *b* are two integers modulo *N*.

Instead of multiplying *a* and *b* directly, first bring them to the ‘Montgomery domain’.

Normal domain	Montgomery domain
$a \bmod N$	$A = a*R \bmod N$
$b \bmod N$	$B = b*R \bmod N$

Modular exponentiation in RSA

$$c = m^e \bmod N$$

… here we do all operations in the mod *N* ring.

Efficiency trick:

Normal domain	Montgomery domain
a mod *N*	*A* = *a***R* mod *N*
b mod *N*	*B* = *b***R* mod *N*

Now multiply them: C = A*B = (*a***R*)*(*b***R*) mod *N*.

Modular exponentiation in RSA

$$c = m^e \bmod N$$

… here we do all operations in the mod *N* ring.

Efficiency trick:

Normal domain	Montgomery domain
$a \bmod N$	$A = a*R \bmod N$
$b \bmod N$	$B = b*R \bmod N$

Now multiply them: $C = A*B = (a*R)*(b*R) \bmod N$.

Now perform Montgomery reduction. It produces

$$C*R^{-1} \bmod N = a*b*R \bmod N$$

$= c*R \bmod N$ where $c = a*b$ is the 'normal domain' multiplication.

Modular exponentiation in RSA

$$c = m^e \bmod N$$

… here we do all operations in the mod *N* ring.

Efficiency trick:

Normal domain	Montgomery domain
$a \bmod N$	$A = a*R \bmod N$
$b \bmod N$	$B = b*R \bmod N$

Now multiply them: $C = A*B = (a*R)*(b*R) \bmod N$.

Now perform Montgomery reduction. It produces

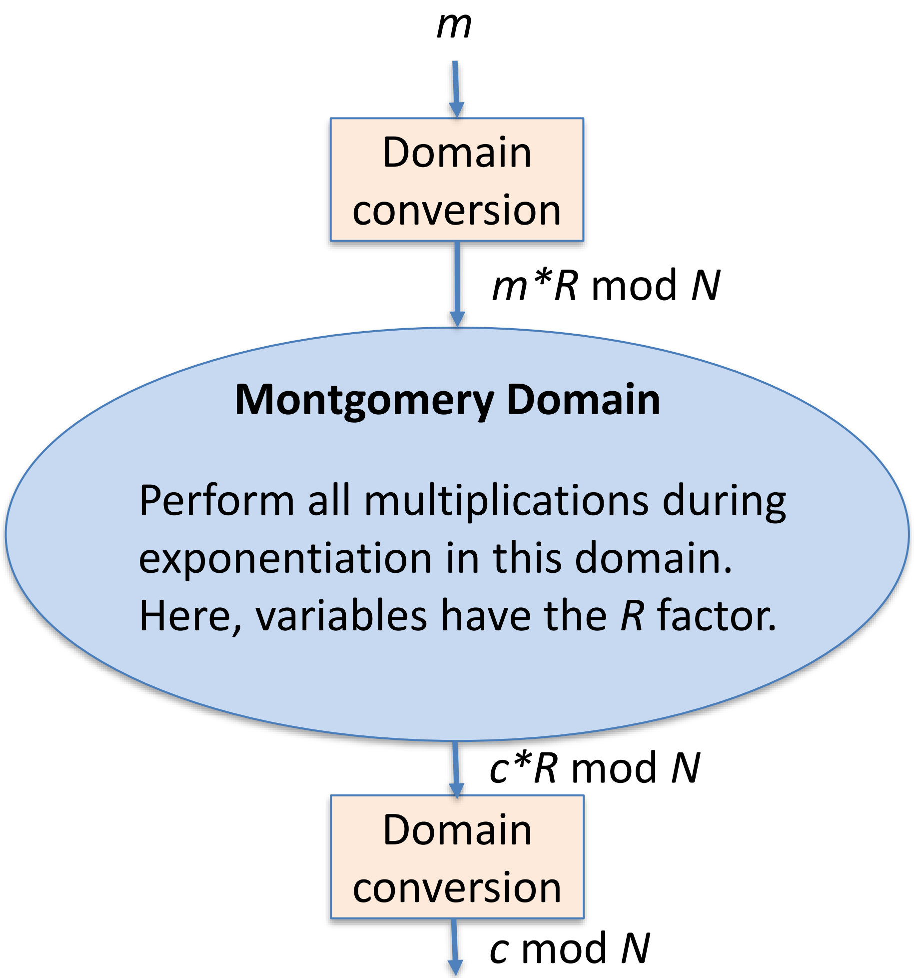
$$C*R^{-1} \bmod N = a*b*R \bmod N$$

$= c*R \bmod N$ where $c = a*b$ is the 'normal domain' multiplication.

Note that the result the Montgomery domain representation of *c*.

Modular exponentiation in RSA

$c = m^e \bmod N$



Modular Reduction for Special Primes

- Barrett and Montgomery are generics algorithms
  - Might not be optimum for numbers with special form

- Some cryptographic protocols use primes with special form:
  - E.g., ECC uses $2^{192} – 2^{64} – 1$
  - E.g., ZKP applications use $2^{64} – 2^{32} + 1$

- Mersenne primes: $2^k – 1$
- Generalized Mersenne primes (Solinas primes): $2^k – c$

Modular Reduction for Special Primes

- Modular reduction for $q = 2^k - c$

$$q = 0 \pmod{q}$$
$$2^k - c = 0 \pmod{q}$$
$$2^k = c \pmod{q}$$

Modular Reduction for Special Primes

- Modular reduction for $q = 2^k - c$

$$q = 0 \pmod{q}$$
$$2^k - c = 0 \pmod{q}$$
$$2^k = c \pmod{q}$$

- Perform $A \pmod{q}$ for $2k$-bit A

$$A = A_1 \cdot 2^k + A_0 \pmod{q}$$
$$A = A_1 \cdot c + A_0 \pmod{q} \quad (\text{using } 2^k = c \pmod{q})$$
…