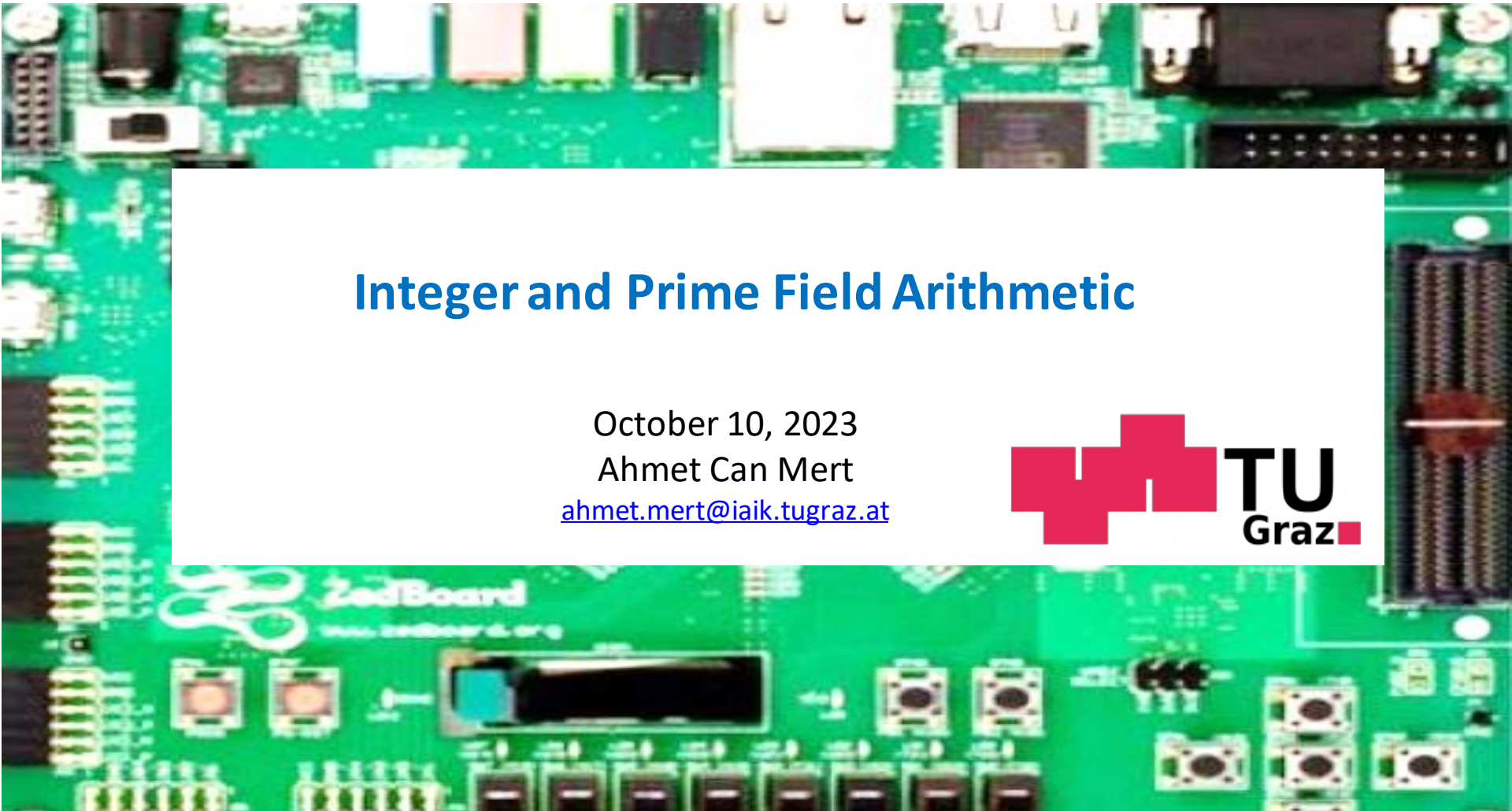


Integer and Prime Field Arithmetic

October 10, 2023

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

TU Graz

Modular Reduction Algorithms

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

- Modular reduction for special primes

- Modular reduction using Table/LUT

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

- Barrett Reduction: $a \pmod{q}$
  - $a < q^2$
  - $2^{k-1} < q < 2^k$ *(k-bit modulus)*

Barrett Modular Reduction Algorithm

- Barrett Reduction: $a \pmod{q}$
  - $a < q^2$
  - $2^{k-1} < q < 2^k$ *(k-bit modulus)*

- Main idea:

$$a \pmod{q} = a - \lfloor a/q \rfloor \cdot q$$

Barrett Modular Reduction Algorithm

- Barrett Reduction: $a \pmod{q}$
  - $a < q^2$
  - $2^{k-1} < q < 2^k$ *(k-bit modulus)*

- Main idea:

$$a \pmod{q} = a - \lfloor a/q \rfloor \cdot q$$

Approximate a/q as $(a \cdot (R/q) / R) \approx (a \cdot \lfloor R/q \rfloor / R)$

Barrett Modular Reduction Algorithm

- Barrett Reduction: $a \pmod{q}$
  - $a < q^2$
  - $2^{k-1} < q < 2^k$ *(k-bit modulus)*

- Main idea:

$$a \pmod{q} = a - \lfloor a/q \rfloor \cdot q$$

Approximate a/q as $(a \cdot (R/q) / R) \approx (a \cdot \lfloor R/q \rfloor / R)$

$$a \pmod{q} = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$$

Barrett Modular Reduction Algorithm

- Barrett Reduction: $a \pmod q$
  - $a < q^2$
  - $2^{k-1} < q < 2^k$ *(k-bit modulus)*

- Main idea:

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

Approximate a/q as $(a \cdot (R/q) / R) \approx (a \cdot \lfloor R/q \rfloor / R)$

$$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$$

Division by q is converted to division by R.

Barrett Modular Reduction Algorithm

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

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

Division is expensive

Barrett Modular Reduction Algorithm

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

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

Division is expensive

$$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$$

Approximate a/q

Barrett Modular Reduction Algorithm

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

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

Division is expensive

$$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$$

Approximate a/q

Select $R = 2^{2k}$, $\lfloor R/q \rfloor$ can be precomputed

Barrett Modular Reduction Algorithm

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

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ Division is expensive

$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$ Approximate a/q

Select $R = 2^{2k}$, $\lfloor R/q \rfloor$ can be precomputed

Select $R = 2^{2k}$, division by R is simple shift

Barrett Modular Reduction Algorithm

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

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ Division is expensive

$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$ Approximate a/q

Select $R = 2^{2k}$, $\lfloor R/q \rfloor$ can be precomputed

Select $R = 2^{2k}$, division by R is simple shift

$$
\begin{aligned}
a \pmod q &= a - \lfloor a \cdot s \rfloor \cdot q \\
&= a - \lfloor a \cdot (\lfloor R/q \rfloor / R) \rfloor \cdot q \\
&= a - \lfloor a \cdot (\lfloor 2^{2k}/q \rfloor / 2^{2k}) \rfloor \cdot q
\end{aligned}
$$

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

Barrett Modular Reduction Algorithm

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

$a \pmod q = a - \lfloor a/q \rfloor \cdot q$ — Division is expensive

$a \pmod q = a - \lfloor a \cdot \lfloor R/q \rfloor / R \rfloor \cdot q$ — Approximate a/q

Select $R = 2^{2k}$, $\lfloor R/q \rfloor$ can be precomputed

Select $R = 2^{2k}$, division by R is simple shift

$$
\begin{aligned}
a \pmod q &= a - \lfloor a \cdot s \rfloor \cdot q \\
&= a - \lfloor a \cdot (\lfloor R/q \rfloor / R) \rfloor \cdot q \\
&= a - \lfloor a \cdot (\lfloor 2^{2k}/q \rfloor / 2^{2k}) \rfloor \cdot q
\end{aligned}
$$

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

...

$a \pmod q = q \cdot (e_1 . q/R + e_2)$ — $a \pmod q < 2 \cdot q$ (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, μ
Output: $C = D \pmod{q}$
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)
```

Barrett Modular Reduction Algorithm (Reducing Multiplier Size)[1]

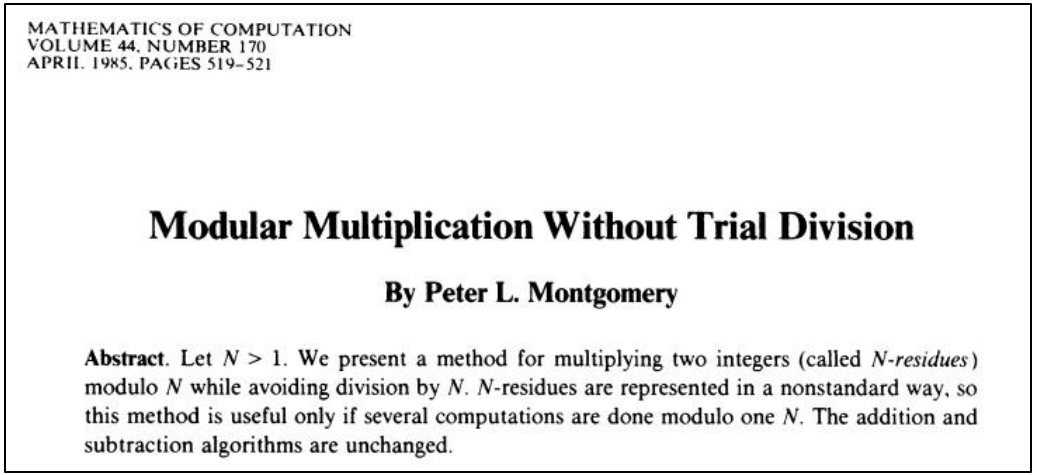
- Takes $D = A \cdot B$ as input and generates $C = D \pmod{q}$
  - $A, B < q$, $D = A \cdot B < q^2$
  - $2^{k-1} < q < 2^k$
  - $\mu = \lfloor 2^{2k+3}/q \rfloor$

Input: $D = A \cdot B$, q, μ
Output: $C = D \pmod{q}$
1: $t = D >> (k-2)$
2: $s = \mathbf{t \cdot \mu}$
2: $r = s >> (k+5)$
3: $u = D - \mathbf{r \cdot q} \pmod{2^{k+1}}$
4: if $(u \geq q)$ **then** $C = u - q$ **else** $C = u$
5: return C

[1] https://uwspace.uwaterloo.ca/bitstream/handle/10012/15191/Roma_Crystal.pdf?sequence=3&isAllowed=y

Montgomery Modular Reduction Algorithm

- An algorithm for computing $C = A \cdot B \pmod{q}$ where A, B, and q are k-bit numbers
- It computes the resulting k-bit number C without performing a division by q
  - Division by q is replaced by division by a power of 2



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.

Montgomery Modular Reduction Algorithm: Classical Montgomery

- Takes $C = A \cdot B$ as input and generates $D = C \cdot R^{-1} \pmod{q}$
  - $A, B < q$, $C = A \cdot B < q^2$
  - $2^{k-1} < q < 2^k$
  - $R = 2^k$, $R^{-1} = 2^{-k} \pmod{q}$
  - $\gcd(q, R) = 1$, q is odd

- Montgomery reduction also requires a quantity $\mu = (-q)^{-1} \pmod{R}$.

$$D = \frac{C + (C.\mu \pmod{R}).q}{R}$$

This only guarantees that D is less than 2.q, we need a final conditional subtraction to bring result into [0,q).

Montgomery Modular Reduction Algorithm: Classical Montgomery

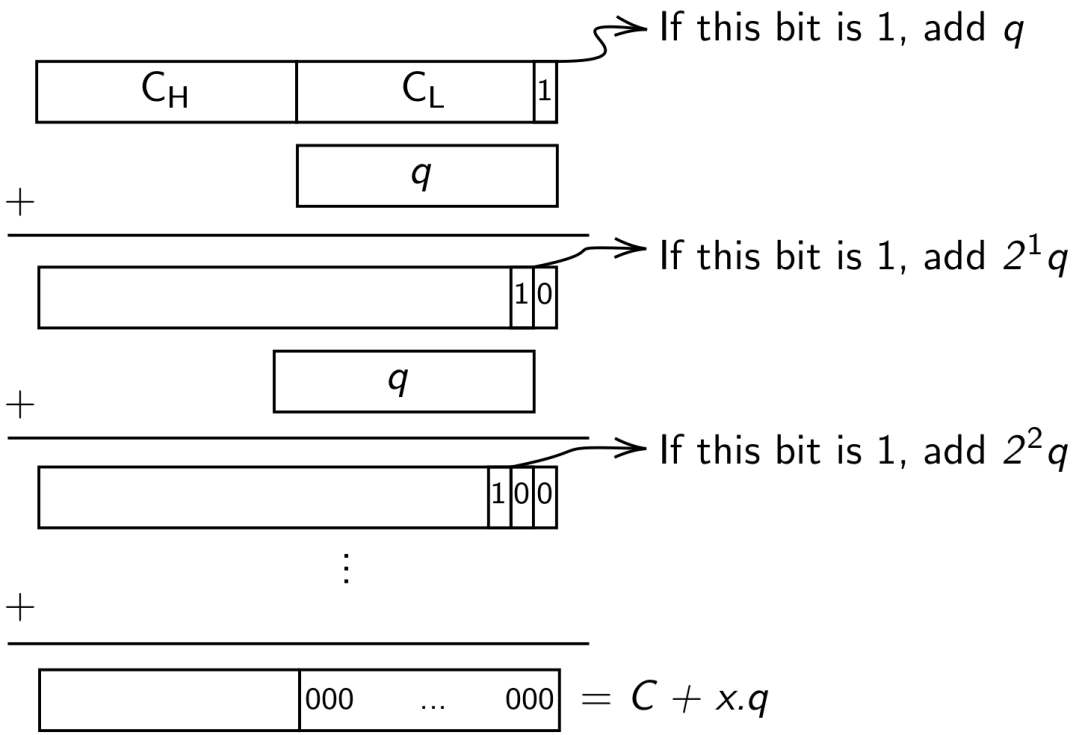
- Takes $C = A \cdot B$ as input and generates $D = C \cdot R^{-1} \pmod{q}$
  - $A, B < q$, $C = A \cdot B < q^2$
  - $2^{k-1} < q < 2^k$
  - $R = 2^k$, $R^{-1} = 2^{-k} \pmod{q}$
  - $\gcd(q, R) = 1$, q is odd

- Montgomery reduction also requires a quantity $\mu = (-q)^{-1} \pmod{R}$.

Input: $C = A \cdot B$, q, μ, $R=2^k$
Output: $D = C \cdot R^{-1} \pmod{q}$
1: $m = C \cdot \mu \pmod{R}$
2: $u = C + m \cdot q$
3: $u = u/R$
4: if $(u \geq q)$ **then** $D = u - q$ **else** $D = u$
5: return D

Montgomery Modular Reduction Algorithm: Classical Montgomery

- Montgomery Algorithm for an input C



Montgomery Modular Reduction Algorithm: Classical Montgomery

- What is x?

$$C + x \cdot q \equiv 0 \pmod{2^k} \text{ where } C = C_H \cdot 2^k + C_L$$

$$C_H \cdot 2^k + C_L + x \cdot q \equiv 0 \pmod{2^k}$$

$$\cancel{C_H \cdot 2^k} + C_L + x \cdot q \equiv 0 \pmod{2^k}$$

$$x \equiv C_L \cdot (-q)^{-1} \pmod{2^k}$$

We define $R = 2^k$ and $\mu = (-q)^{-1} \pmod{R}$. Then,

$$x = C_L . \mu \pmod{R}$$

$$x = C . \mu \pmod{R}$$

- What is reduction result?

$$D = \frac{C + x.q}{2^k} \longrightarrow D = \frac{C + (C.\mu \pmod{R}).q}{R}$$

Montgomery Modular Reduction Algorithm: Classical Montgomery

- There is R^{-1} in the result. Therefore, output needs to be multiplied by R. Alternatively, one of the inputs may be converted to Montgomery form by multiplying with R.
  - Not suitable for a single modular multiplication

In#1	In#2	Output	Correction
A	B	$C = (A.B).R^{-1} \pmod{q}$	$C.R \pmod{q}$
$A.R$	B	$C = (A.B.R).R^{-1} \pmod{q}$	–
A	$B.R$	$C = (A.B.R).R^{-1} \pmod{q}$	–
$A.R$	$B.R$	$C = (A.B.R^2).R^{-1} \pmod{q}$	$C.R^{-1} \pmod{q}$

Montgomery Modular Reduction Algorithm: Classical Montgomery

- There is R^{-1} in the result. Therefore, output needs to be multiplied by R. Alternatively, one of the inputs may be converted to Montgomery form by multiplying with R.
  - Not suitable for a single modular multiplication

In#1	In#2	Output	Correction
A	B	$C = (A.B).R^{-1} \pmod q$	$C.R \pmod q$
$A.R$	B	$C = (A.B.R).R^{-1} \pmod q$	–
A	$B.R$	$C = (A.B.R).R^{-1} \pmod q$	–
$A.R$	$B.R$	$C = (A.B.R^2).R^{-1} \pmod q$	$C.R^{-1} \pmod q$

When to use Montgomery Modular 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
$a \bmod N$	$A = a*R \bmod N$
$b \bmod N$	$B = b*R \bmod N$

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

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

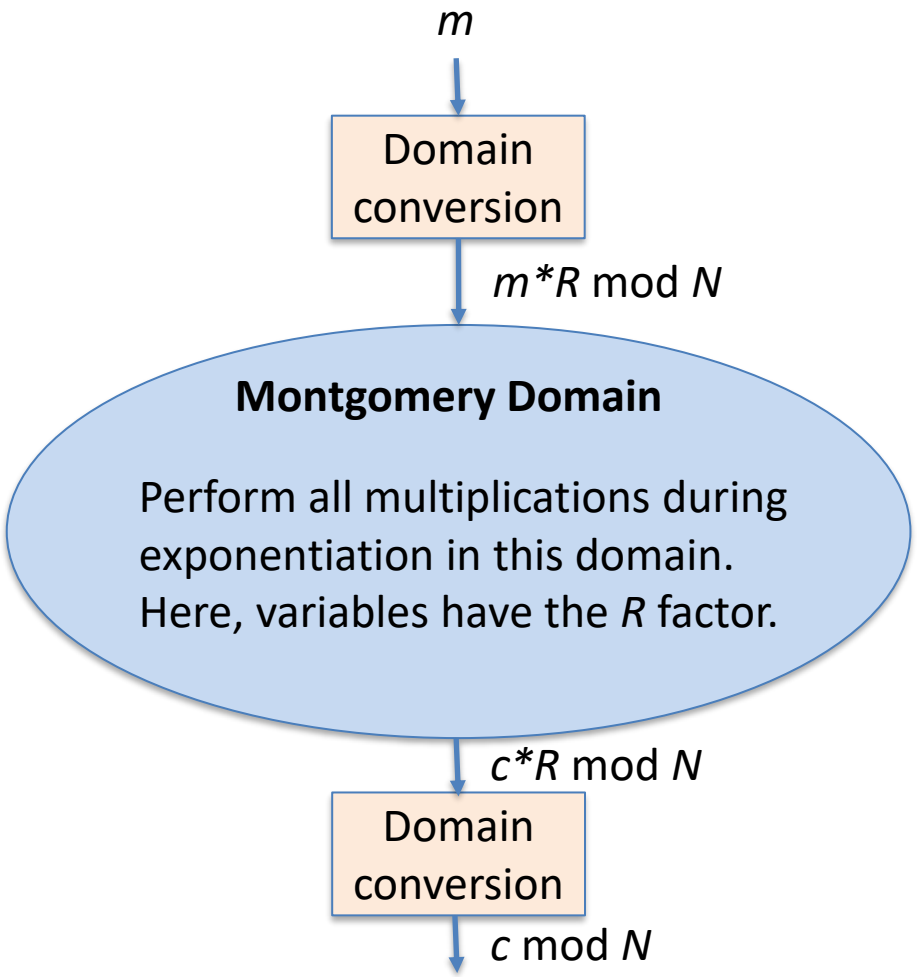
$$C*R^{-1} \bmod N = a*b*R \bmod N$$

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

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

Modular exponentiation in RSA

$c = m^e \bmod N$



Montgomery Modular Reduction Algorithm: Classical Montgomery

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

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

D = 129

u = (D + (D*qp % R)*q)/R
u = u-q if(u >= q) else u

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

Run the code.

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., Some ZKP/HE applications use $2^{64} - 2^{32} + 1$
  - E.g., NIST-selected PQC scheme Kyber uses $13 \cdot 2^{8} - 1$

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

Modular Reduction for Special Primes

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

  $q = 0 \pmod{q}$

Modular Reduction for Special Primes

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

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

Modular Reduction for Special Primes

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

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

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

  $A = A_1 \cdot 2^k + A_0 \pmod q$
  $A = A_1 \cdot c + A_0 \pmod q$ (using $2^k = c \pmod q$)
  ...

Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r

Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r

Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r

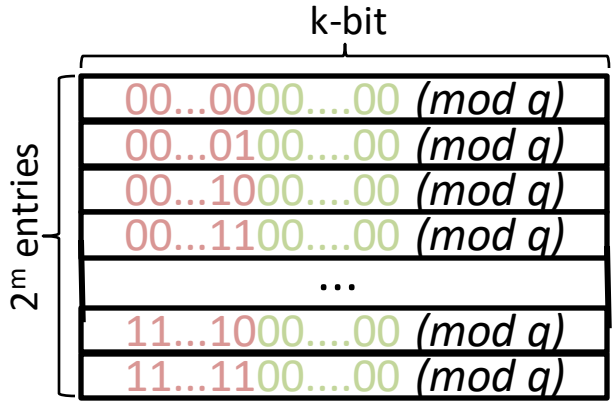


Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r

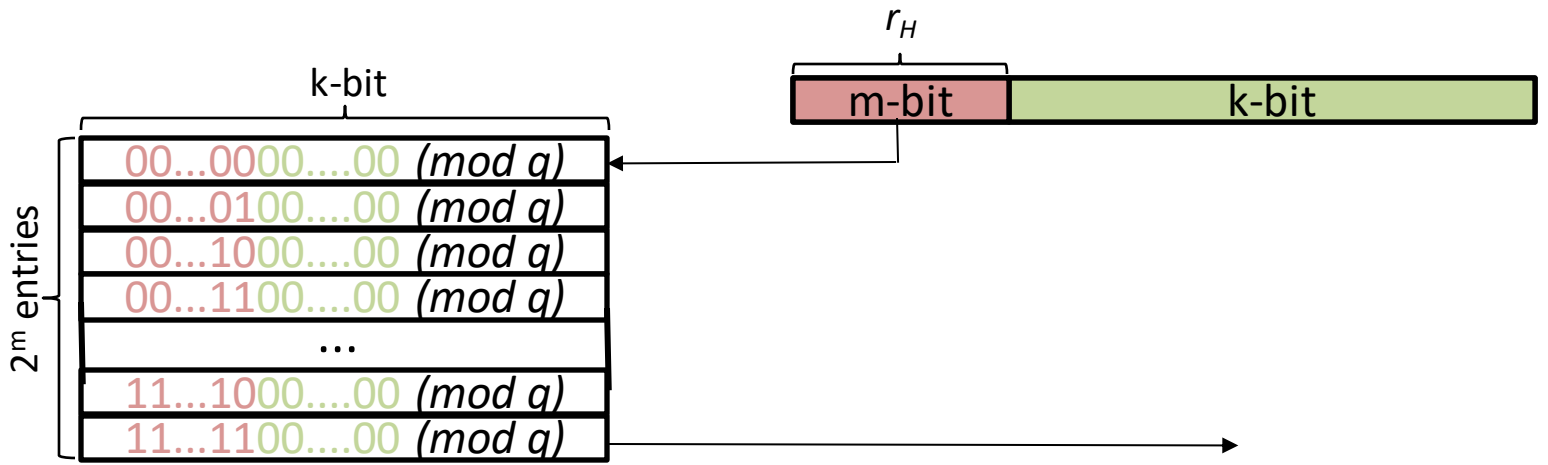


Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r

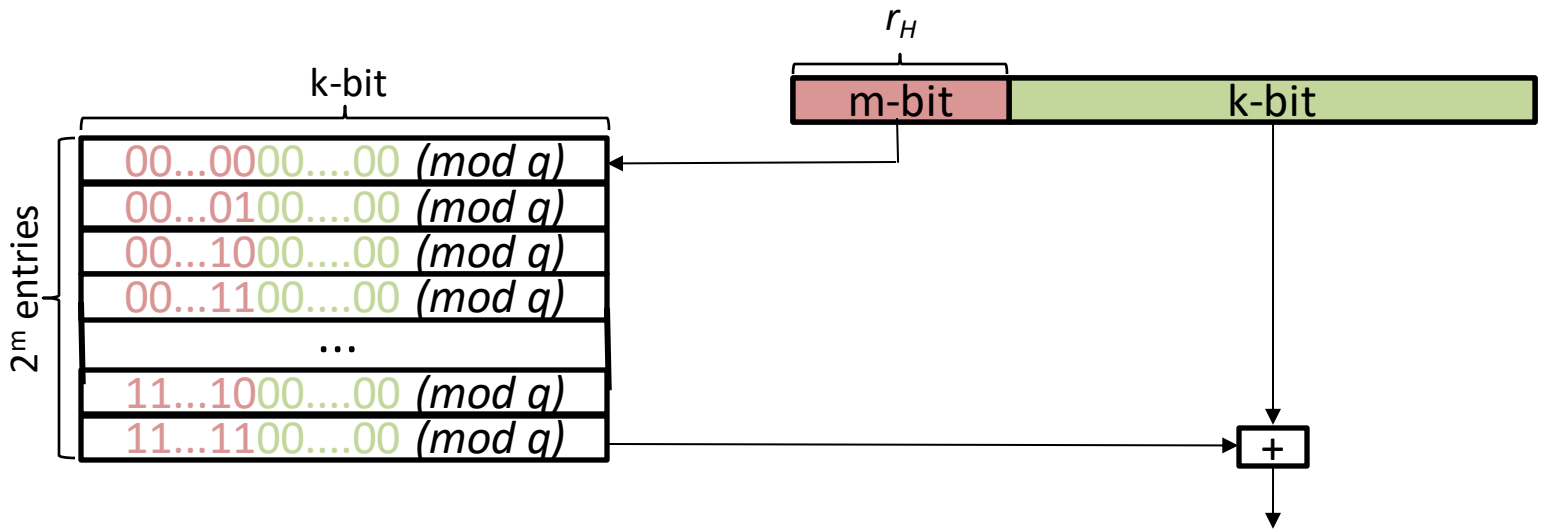


Table-based Reduction

- Table-based reduction method uses a table to store pre-computed values for most-significant bits of input in *(mod q)*.
  - Store pre-computed values $r_H \cdot 2^k \pmod q$ for input r