TU Graz

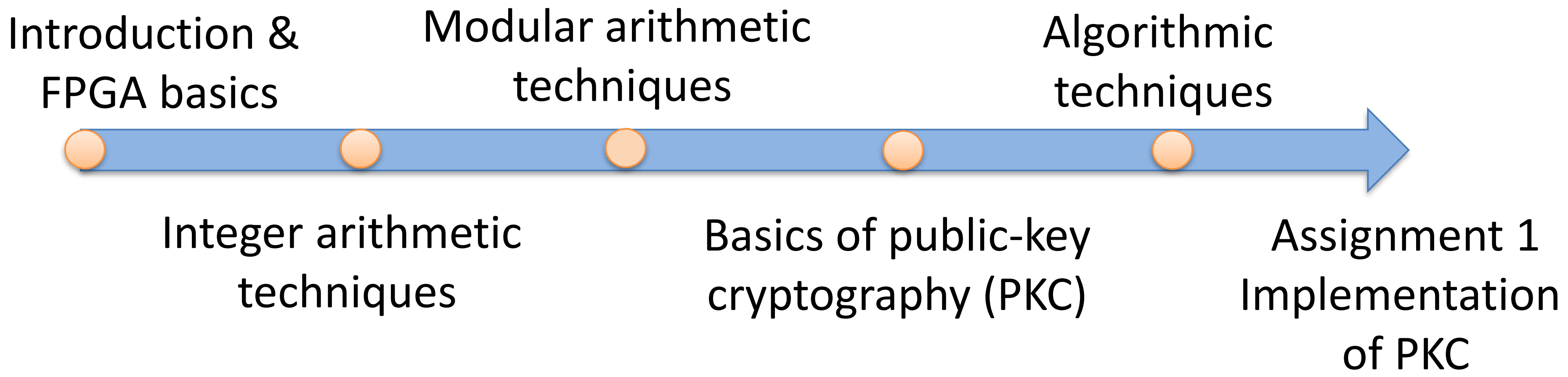
SCIENCE
PASSION
TECHNOLOGY

Cryptography on HW Platform
Modular Arithmetic Techniques

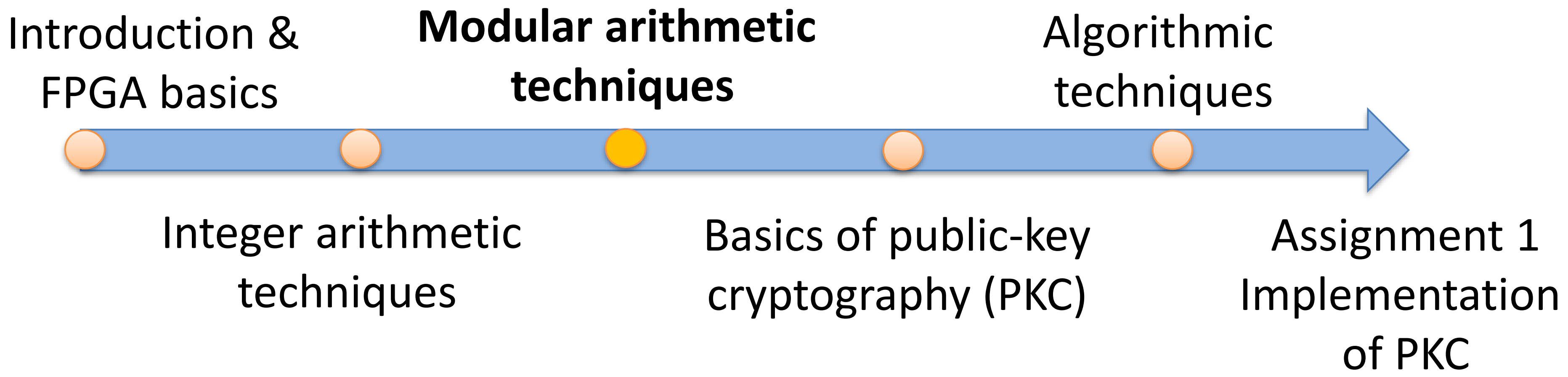
December, 2023

Sujoy Sinha Roy
sujoy.sinharoy@iaik.tugraz.at
Graz University of Technology

Roadmap


Introduction &
FPGA basics
Integer arithmetic
techniques
Modular arithmetic
techniques
Basics of public-key
cryptography (PKC)
Algorithmic
techniques
Assignment 1
Implementation
of PKC

Roadmap


Introduction &
FPGA basics
Integer arithmetic
techniques
Modular arithmetic
techniques
Basics of public-key
cryptography (PKC)
Algorithmic
techniques
Assignment 1
Implementation
of PKC

Reminder

Background on modular arithmetic

The "modulo" or mod operation

For any integer c, we want to compute the result of

$$c \bmod m$$

(we are interested in positive integers in cryptography)

The "modulo" or mod operation

For any integer *c*, we want to compute the result of

$$c \bmod m$$

(we are interested in positive integers in cryptography)

<u>Procedure:</u>

1. Divide *c* by *m* and obtain the quotient *q*

$$q = \lfloor c/m \rfloor$$

2. Compute the remainder $r = c - q*m$

3. Assign $r = c \bmod m$

The "modulo" or mod operation

Example 1:

$$23 \bmod 5 = ?$$

The “modulo” or mod operation

Example 2:

$$? \bmod 5 = 3$$

Congruence: definition

For modulus m, and two positive integers a and b, we say that a is congruent to b modulo m if

$$m \mid (a - b)$$

The notation is

$$a \equiv b \pmod{m}$$

The binary relationship is called “congruence”.
It indicates that a and b have the same remainder modulo m.

Example: $23 \equiv 3 \pmod{5}$, similarly $13 \equiv 3 \pmod{5}$, and $13 \equiv 23 \pmod{5}$.

Properties of congruence

The following relations hold

i. $a \equiv a \pmod{m}$

ii. $a \equiv b \pmod{m} \Rightarrow b \equiv a \pmod{m}$

iii. $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$

iv. $a \equiv a' \pmod{m}$ and $b \equiv b' \pmod{m}$

$\Rightarrow a + b \equiv a' + b' \pmod{m}$ and

$a * b \equiv a' * b' \pmod{m}$

Congruence class

The congruence class of a modulo m is the set of all integers that are congruent to a modulo m.

$$[a]_m = \{b \in \mathbb{Z} \text{ such that } b \equiv a \pmod{m}\}$$

Example:

$$[3]_5 = \{\ldots, -7, -2, 3, 8, 13, 18, 23, \ldots.\}$$

For more information on congruences, you may consider reading chapter-2 of the book:

“A Computational Introduction to Number Theory and Algebra”, by Victor Shoup. https://shoup.net/ntb/ntb-v2.pdf

Consider the problem of computing modular multiplication.

Input: $a, b \in [0, m\text{-}1]$
Output: $c = a * b \bmod m \in [0, m\text{-}1]$
1: $t = a * b \in [0, (m\text{-}1)^2]$
2: $r = t \bmod m$
3: return r

The number of bits in t is 2x larger than in m.

Consider the problem of computing modular multiplication.

Input: $a, b \in [0, m\text{-}1]$
Output: $c = a * b \bmod m \in [0, m\text{-}1]$
1: $t = a * b \in [0, (m\text{-}1)^2]$
$\boxed{\textbf{2:}\ r = t \bmod m}$
3: return r

The number of bits in t is 2x larger than in m.

How to compute the modular reduction of *t*?

Consider the problem of computing modular multiplication.

Input: $a, b \in [0, m\text{-}1]$
Output: $c = a * b \bmod m \in [0, m\text{-}1]$
1: $t = a * b \in [0, (m\text{-}1)^2]$
$\boxed{\textbf{2: } r = t \bmod m}$
3: return r

The number of bits in t is 2x larger than in m.

<u>Schoolbook method</u> for calculating r:

1. Perform division $q = \lfloor t/m \rfloor$
2. Calculate remainder $r = t - q*m$

Schoolbook modular reduction is very inefficient

Division is very expensive to compute.

See how long this PARI/GP code takes for division (/) and multiplication (*).

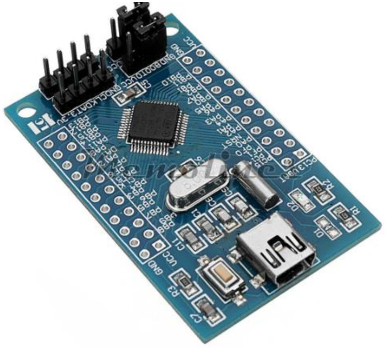
```
a=vector(100000);
b=vector(100000);
c=vector(100000);

for(i=1, 100000, a[i] = random(2^4096))
for(i=1, 100000, b[i] = random(2^2048))
for(i=1, 100000, c[i]=floor(a[i]/b[i]))
```

Schoolbook modular reduction is very inefficient

Division is very expensive to compute.

Low-end microcontrollers do not have division instructions.



Division is computed as repeated subtraction.

→ Extremely slow modular reduction

Efficient algorithms for modular reduction

In this course, we will study the following algorithms

- Barrett reduction
- Montgomery reduction
- Reduction for special modulus

Barrett reduction

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.

P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor". CRYPTO' 86.

Barrett reduction

Barrett's method optimizes reduction for fixed modulus m.

Main idea: Replace division by cheaper multiplication.

Precompute 1/m and multiply t*(1/m).

Calculating remainder r:

1. Perform division $q = \lfloor t/m \rfloor$
2. Calculate remainder $r = t - q*m$

Example: Let m = 7069 (m is a 13-bit number)

5044*6312 mod m = ?

Precomputed (1/m) =0.00014146272457207525816947234 …

t = 5044*6312 = 31837728

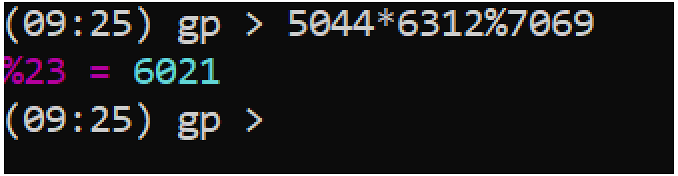
$t/m = t*(1/m)$ = 31837728 * 0.00014146272457207525816947234 …

≈ 4503.8517470646484 …

$q = \lfloor t/m \rfloor$ = 4503

$r = t - q*m$ = 31837728 - 4503*7069 = 6021

Matches with PARI/GP



Example: Let m = 7069 (m is a 13-bit number)

5044*6312 mod m = ?

Precomputed (1/m) =0.000141462724

What precision do we need for correctly computing quotient?

t = 5044*6312 = 31837728

$t/m = t*(1/m)$ = 31837728 * 0.00014146272457207525816947234 …

≈ 4503.8517470646484 …

$q = \lfloor t/m \rfloor = 4503$

$r = t - q*m$ = 31837728 - 4503*7069 = 6021

Example: Let m = 7069 (m is a 13-bit number)

5044*6312 mod m = ?

Precomputed (1/m) =0.00014146272457207525816947234 ...

t = 5044*6312 = 31837728

$t/m = t$*(1/m) = 31837728 * 0.00014146272457207525816947234 ...

≈ 4503.8517470646484 ...

$q = \lfloor t/m \rfloor = 4503$

$r = t - q*m$ = 31837728 - 4503*7069 = 6

Barrett's method takes 2k bits after the (.) where k is the length of m.

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$$
\begin{aligned}
1/m &= 0.00014146272457207525816947234\ldots_{10} \\
&= 0.0000000000001001010001010101100111\ldots_2 \\
&\approx 0.00000000000010010100010101_2 \qquad \text{(Truncate after } 2k\text{=26 bits)} \\
&= 0.00014145672321319580_{10}
\end{aligned}
$$

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$1/m = 0.000141462724572075258169472 34\ldots_{10}$
$= 0.0000000000001001010001010101100111\ldots_2$
$\approx 0.00000000000010010100010101_2$ (Truncate after $2k$=26 bits)
$= 0.00014145672321319 58_{10}$

Next, we can do like before

$t*(1/m) \approx 31837728 * 0.0001414567232131958$
$= 4503.660677433013891142 40$

$q = \lfloor t/m \rfloor = 4503$

$r = t - q*m = 31837728 - 4503*7069 = 6021$

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$$1/m = 0.000141462724572075258169472 34 \ldots_{10}$$
$$= 0.000000000000100101000101010110 0111 \ldots_2$$
$$\approx 0.00000000000010010100010101_2 \qquad \text{(Truncate after } 2k\text{=26 bits)}$$
$$= 0.0001414567232131958_{10}$$

Next, we can do like before

$$t^*(1/m) \approx 31837728 * 0.0001414567232131958$$
$$= 4503.6606774330138911424 0$$

Can we replace this real multiplication by integer multiplication?

$$q = \lfloor t/m \rfloor = 4503$$

$$r = t - q^*m = 31837728 - 4503^*7069 = 6021$$

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$$1/m = 0.0001414627245720752581694723 4 \ldots_{10}$$
$$= 0.0000000000001001010001010101100111 \ldots_2$$
$$\approx 0.00000000000010010100010101_2 \qquad \text{(Truncate after } 2k\text{=26 bits)}$$
$$= 0.00014145672321319 58_{10}$$

Replaces the real number by a $2k$ shifted value of $1/m$, which is integer.

$$\mu = 0.00000000000010010100010101_2 \ll 26 \qquad \text{(left shift is multiplication by } 2^{26}\text{)}$$
$$= 10010100010101_2$$
$$= 9493_{10}$$

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$1/m = 0.0001414627245720752581694723\ldots_{10}$
$= 0.00000000000010010100010101100111\ldots_2$
$\approx 0.00000000000010010100010101_2$ (Truncate after $2k$=26 bits)
$= 0.00014145672321319580_{10}$

Replaces the real number by a $2k$ shifted value of $1/m$, which is integer.

$\mu = 0.00000000000010010100010101_2 << 26$ (left shift is multiplication by 2^{26})
$= 10010100010101_2$
$= 9493_{10}$

$q' = (t*\mu) >> 2k = (31837728*9493) >> 26$ (Truncate $2k$=26 least bits)
$= 4503_{10}$

Barrett reduction

Modulus m = 7069 is 13 bits long. Hence k = 13.

$$
\begin{aligned}
1/m &= 0.000141462724572075258169472 34\ldots_{10} \\
&= 0.0000000000001001010001010101100111\ldots_{2} \\
&\approx 0.00000000000010010100010101_2 \qquad \text{(Truncate after } 2k\text{=26 bits)} \\
&= 0.00014145672321319 58_{10}
\end{aligned}
$$

Replaces the real number by a $2k$ shifted value of $1/m$, which is integer.

$$
\begin{aligned}
\mu &= 0.00000000000010010100010101_2 \ll 26 \qquad \text{(left shift is multiplication by } 2^{26}\text{)} \\
&= 10010100010101_2 \\
&= 9493_{10}
\end{aligned}
$$

$$
\begin{aligned}
q' &= (t*\mu) \gg 2k = (31837728*9493) \gg 26 \qquad \text{(Truncate } 2k\text{=26 least bits)} \\
&= 4503_{10}
\end{aligned}
$$

$r = t - q'*m = 31837728 - 4503*7069 = 6021$

Barrett reduction: conditional subtraction

Schoolbook method for $t=a*b$

1. Quotient $q = \lfloor t/m \rfloor$
2. Remainder $r = t - q*m$

Barrett method for $t=a*b$

1. Precomputes approximate $\mu = \lfloor 2^{2k}/m \rfloor$
2. Approximate quotient $q' = \lfloor (t*\mu) / 2^{2k} \rfloor$
3. Remainder $r = t - q'*m$

Barrett reduction: conditional subtraction

Schoolbook method for $t=a*b$

1. Quotient $q = \lfloor t/m \rfloor$
2. Remainder $r = t - q*m$

Barrett method for $t=a*b$

1. Precomputes approximate $\mu = \lfloor 2^{2k}/m \rfloor$
2. **Approximate** quotient $q' = \lfloor (t*\mu)/2^{2k} \rfloor$
3. Remainder $r = t - q'*m$

In the approximation process, we truncate $1/m$ at $2k$-th bit after (.)
→ This causes approximation error.

Because of this error, there are two possibilities:

$$q' = q \quad \text{or} \quad q' = q-1$$

Barrett reduction: conditional subtraction

Schoolbook method for $t = a*b$
1. Quotient $q = \lfloor t/m \rfloor$
2. Remainder $r = t - q*m$

Barrett method for $t = a*b$
1. Precomputes approximate $\mu = \lfloor 2^{2k}/m \rfloor$
2. **Approximate** quotient $q' = \lfloor (t*\mu) / 2^{2k} \rfloor$
3. Remainder $r = t - q'*m$

In the approximation process, we truncate $1/m$ at $2k$-th bit after (.)
→ This causes approximation error.

Because of this error, there are two possibilities:

$$q' = q \quad \text{or} \quad q' = q-1$$

If $q' = q-1$ happens, then r will be in $[m, 2m)$.
→ One additional subtraction of m from r will be needed.

Barrett reduction: conditional subtraction (proof)

For $\mu = \lfloor 2^{2k}/m \rfloor$, we have the relation

$$2^{2k}/m - 1 < \mu < 2^{2k}/m$$

Hence,

$$t/m - t/2^{2k} \leq \ t^{*}\mu/2^{2k} \leq t/m$$

Because $t/2^{2k} < 1$, we write

$$t/m - 1 < \ t^{*}\mu/2^{2k} \leq t/m$$

Now consider the floor $\lfloor (t^{*}\mu)\ /2^{2k} \rfloor$. There are two possibilities:

- $\lfloor (t^{*}\mu)\ /2^{2k} \rfloor > t/m - 1$ [e.g., floor(7.1) > 6.7]
- $\lfloor (t^{*}\mu)\ /2^{2k} \rfloor < t/m - 1$ [e.g., floor(6.9) > 6.7]

Hence,

$$t/m - 2 < \lfloor t^{*}\mu/2^{2k} \rfloor \leq t/m$$

Barrett reduction: conditional subtraction (proof)

… continuing

$$t/m - 2 < \lfloor t*\mu/2^{2k} \rfloor \leq t/m$$

or

$$t/m - 2 < q' \leq t/m$$

Hence,

$$t - 2m < q'*m \leq t$$

Or

$$0 \leq t - q'*m < 2m$$

Barrett method for $t=a*b$
1. Precomputes approximate $\mu = \lfloor 2^{2k}/m \rfloor$
2. Approximate quotient $q' = \lfloor (t*\mu) / 2^{2k} \rfloor$
3. Remainder $r = t - q'*m$

Hence, r is in [0, 2m].
$\Rightarrow$ Conditional subtraction $r - m$.

Complete Barrett reduction algorithm

Input: $t = a*b \in [0, (m\text{-}1)^2]$, $2^{k\text{-}1} < m < 2^k$, $\mu = \lfloor 2^{2k}/m \rfloor$
Output: $c = t \pmod{m}$
1: $q' = \lfloor (t*\mu)\ /2^{2k} \rfloor$
2: $r = t - q'*m$
4: if $(r \geq m)$ **then** $c = r - m$ **else** $c = r$
5: return c

Modulus m is fixed and μ is precomputed.

Complete Barrett reduction algorithm

Try Barrett algorithm in Sage.

https://sagecell.sagemath.org/

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

t = 120

r = t - ((t*mu) >> 2*k)*m
c = r-m if(r >= m) else r

print("t mod m:", t%m)
print("BR(t,m):", c)
```

Efficient algorithms for modular reduction

In this course, we will study the following algorithms

- Barrett reduction
- **Montgomery reduction**
- Reduction for special modulus

Montgomery reduction

Replaces expensive division by cheaper shift operation.

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.

P. Montgomery, "Modular Multiplication Without Trial Division". Mathematics of Computation, 1985.

Montgomery reduction procedure

Let, modulus m is a k-bit odd and $R = 2^k$

and $m' = (-m)^{-1} \bmod R$

Takes an input t in the range [0, $R*m$-1] and computes s in the range [0, $2m$-1]

$$s = \frac{t + (t*m' \bmod R)*m}{R}$$

s is in the range [0, $2m$].
After a conditional subtraction of m from s (when $m < s < 2m$), we get $s = t*R^{-1} \bmod m$

Why does this work?

Our $2k$ bit integer t

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Our 2*k* bit integer *t*

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Wish *t* is like this?

xxxxxxxxxxxxxxxx0000000000000

Least *k* bits are zeros

Our 2*k* bit integer *t*

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Wish *t* is like this?

xxxxxxxxxxxxxxxxx00000000000000

Least *k* bits are zeros

We divide by $R=2^k$ (right shift by *k* bits) to obtain

000000000000000xxxxxxxxxxxxxxxx

Becomes *k* bit integer

Our $2k$ bit integer t

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Wish t is like this?

xxxxxxxxxxxxxxxx0000000000000

Least k bits are zeros

We divide by $R=2^k$ (right shift by k bits) to obtain

00000000000000xxxxxxxxxxxxxxxx Becomes k bit integer

Division by R is equivalent to multiplication by R^{-1} mod m

→ We get the desired result $t*R^{-1}$ mod m

Our 2*k* bit integer *t* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

In real world these *k* bits may not be all 0s

Can we transform *t* into *t*' such that, $t \equiv t'$ (mod *m*) and

t' = xxxxxxxxxxxxxxxxxx0000000000000 **?**

Least *k* bits are zeros

t_H t_L

Our 2*k* bit integer *t* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

In real world these *k* bits may not be all 0s

Can we transform *t* into *t'* such that, $t \equiv t'$ (mod *m*) and

$t' =$ xxxxxxxxxxxxxxxxxx0000000000000 ?

Least *k* bits are zeros

Let, $t = t_H\, 2^K + t_L$

t_H t_L

Our $2k$ bit integer t xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

In real world these k bits may not be all 0s

Can we transform t into t' such that, $t \equiv t'$ (mod m) and

t' = xxxxxxxxxxxxxxxxxx0000000000000 ?

Least k bits are zeros

Let, $t = t_H\, 2^K + t_L$

We add $m*q$ to t such that, $t' = t_H\, 2^K + t_L + m*q \equiv 0 \pmod{2^k}$

Note that with $t' = t_H\, 2^K + t_L + m*q$ we have $t' \equiv t \pmod{m}$

t_H t_L

Our $2k$ bit integer t xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

In real world these k bits may not be all 0s

Can we transform t into t' such that, $t \equiv t'$ (mod m) and

t' = xxxxxxxxxxxxxxxxxx000000000000000 ?

Least k bits are zeros

Let, $t = t_H 2^K + t_L$

We add $m*q$ to t such that, $t' = t_H 2^K + t_L + m*q \equiv 0 \pmod{2^k}$

$\Rightarrow q = t_L*(-m^{-1}) \bmod 2^k$

$= t_L*m' \bmod 2^k$

t_H t_L

Our $2k$ bit integer t xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

In real world these k bits may not be all 0s

Can we transform t into t' such that, $t \equiv t'$ (mod m) and

t' = xxxxxxxxxxxxxxxxxxx0000000000000 ?

Least k bits are zeros

Let, $t = t_H\, 2^K + t_L$

We add $m*q$ to t such that, $t' = t_H\, 2^K + t_L + m*q \equiv 0 \pmod{2^k}$

$\Rightarrow q = t_L*(-m^{-1}) \bmod 2^k$

$= t_L*m' \bmod 2^k$

Our $t' = t + m*q$.

Montgomery reduction: conditional subtraction

Modulus m is a k-bit odd, $R = 2^k$ and $m' = (-m)^{-1} \bmod R$

For an input t in the range $[0, R*m-1]$ compute the following:

- $q = (t \bmod R) * m' \bmod R$ → k bits, in range $[0, R-1]$
- $t' = t + q*m$ → $2k + 1$ bits, in range $[0, 2R\mathrm{m})$
- $s = t'/R$ → $k + 1$ bits and $< 2\mathrm{m}$

Montgomery reduction: conditional subtraction

Modulus m is a k-bit odd, $R = 2^k$ and $m' = (-m)^{-1} \bmod R$

For an input t in the range $[0, R*m\text{-}1]$ compute the following:

- $q = (t \bmod R) * m' \bmod R$ → k bits, in range $[0, R\text{-}1]$
- $t' = t + q*m$ → $2k + 1$ bits, in range $[0, 2R\mathrm{m})$
- $s = t'/R$ → $k + 1$ bits and $< 2\mathrm{m}$
- *If* ($s >= m$) then output $s - m$

  else output s

Complete Montgomery reduction algorithm

Try Barrett algorithm in Sage.

https://sagecell.sagemath.org/

```
m = 19
k = 5
R = 2^(2*k)
mp= -m^(-1) % R          #m’

t = 120

s = (t + (t*mp % R)*m)/R
c = s-m if(s >= m) else s

print("t mod m:", t%m)
print("MR(t,m):", c)
print("c*R mod q:", c*R % m)
```

Montgomery reduction: input and output forms

In#1	In#2	$t=a*b$	Output	Adjustment
a	b	$a*b$	$s = a*b*R^{-1} \pmod{m}$	$s*R \pmod{m}$
$a*R$	b	$a*b*R$	$s = a*b*R*R^{-1} \pmod{m}$	*Not required*
a	$b*R$	$a*b*R$	$s = a*b*R*R^{-1} \pmod{m}$	*Not required*
$a*R$	$b*R$	$a*b*R$	$s = a*b*R^2*R^{-1} \pmod{m}$	$s*R^{-1} \pmod{m}$

To obtain $a*b$ mod m, removing the R factor is needed.

Montgomery reduction: when to use?

Consider computing a^5 mod m.

Montgomery reduction: when to use?

Consider computing a^5 mod m.

Usual way of computing,
e.g., with Barret reduction.

T = $a*a$ mod m
T = T*a mod m
T = T*a mod m
T = T*a mod m
= a^5 mod m

Montgomery reduction: when to use?

Consider computing a^5 mod m.

Usual way of computing, e.g., with Barret reduction.

T = $a*a$ mod m
T = T*a mod m
T = T*a mod m
T = T*a mod m
= a^5 mod m

If we use Montgomery **naively**

T = Mont($a*a$, m)
T = T*R mod m
T = Mont(T*a, m)
T = T*R mod m
…
T = Mont($a*a$, m)
= a^5R^{-1} mod m
T = T*R mod m


a mod m
b mod m
Montgomery Multiplier()
abR^{-1} mod m

Inputs and outputs all have R factor.

→ Gives us a closed representation called "Montgomery form"

Inputs and outputs all have R factor.
→ Gives us a closed representation called “Montgomery form”

Inputs and outputs all have R factor.

→ Gives us a closed representation called "Montgomery form"

Montgomery reduction: when to use?

Consider computing a^5 mod m.

Correct method of using Montgomery

$b = a$*R mod m. #Convert to Montgomery form

T = MontMultiplier(b, b)

T = MontMultiplier(T, b)

T = MontMultiplier(T, b)

T = MontMultiplier(T, b)

$= a^5R \bmod m$

Result = T*R^{-1} mod m #Convert to normal form

Efficient algorithms for modular reduction

In this course, we will study the following algorithms

- Barrett reduction
- Montgomery reduction
- **Reduction for special modulus**

Modular deduction for special modulus

- Some cryptographic primitives use moduli with sparse representation:
  E.g., ECC uses $m = 2^{192} - 2^{64} - 1$
  E.g., Some ZKP/FHE applications use $m = 2^{64} - 2^{32} + 1$

- Mersenne primes: $m = 2^k - 1$ with k a prime.
  E.g., $m = 2^{31} - 1$,
  $m = 2^{61} - 1$ is currently the largest known Mersenne prime.

- Pseudo Mersenne primes (Solinas primes): $m = 2^k - c$ with small c.

Can modular reduction be made fast utilizing sparse structure of m?

Modular deduction for special modulus

Example: modular reduction for $m = 2^k - c$

$\Rightarrow$

$$m \equiv 0 \pmod{m}$$

Modular deduction for special modulus

Example: modular reduction for $m = 2^k - c$

$\Rightarrow$

$$m \equiv 0 \pmod{m}$$
$$2^k - c \equiv 0 \pmod{m}$$

Modular deduction for special modulus

Example: modular reduction for $m = 2^k - c$

$\Rightarrow$

$$m \equiv 0 \pmod{m}$$

$$2^k - c \equiv 0 \pmod{m}$$

$$2^k \equiv c \pmod{m}$$

Modular deduction for special modulus

Example: modular reduction for $m = 2^k - c$

$\Rightarrow$

$$m \equiv 0 \pmod{m}$$
$$2^k - c \equiv 0 \pmod{m}$$
$$2^k \equiv c \pmod{m}$$

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

$$A = A_1 \cdot 2^k + A_0 \bmod m$$
$$A = A_1 \cdot c + A_0 \bmod m \quad \text{using } 2^k \equiv c \pmod{m}$$

…

References

V. Shoup, "A Computational Introduction to Number Theory and Algebra”. https://shoup.net/ntb/ntb-v2.pdf

P. Barrett, “Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor”. CRYPTO’ 86.

P. Montgomery, "Modular Multiplication Without Trial Division". Mathematics of Computation, 1985.

D. Hankerson, S. Vanstone, A. Menezes, "Guide to Elliptic Curve Cryptography".