

# Finite Field Arithmetic

Nigel Smart

nigel@cs.bris.ac.uk

January 27, 2009

# Outline

Choices of Finite Field Arithmetic

Characteristic Two Fields

Large Prime Characteristic Fields

Large Integer Arithmetic

Montgomery Arithmetic

GM Primes

OEF Fields

## Choices of Finite Field Arithmetic

# Recap on Finite Fields

A **Field** is a set with two operations  $(G, \times, +)$  such that

- ▶  $(G, +)$  is an abelian group, identity denoted by 0.
- ▶  $(G \setminus \{0\}, \times)$  is an abelian group
- ▶  $(G, \times, +)$  satisfies the **distributive law**

## Distributive law

For all  $f, g, h \in (G, \times, +)$

$$f \times (g + h) = (f \times g) + (f \times h).$$

## Examples

Rational numbers, real numbers, complex numbers, integers modulo  $p$ .

# Fields

We define the set of invertible elements of  $\mathbb{Z}/N\mathbb{Z}$  as

$$(\mathbb{Z}/N\mathbb{Z})^* = \{a \in \mathbb{Z}/N\mathbb{Z} : \gcd(a, N) = 1\}.$$

The set  $(\mathbb{Z}/N\mathbb{Z})^*$  is always a group with respect to multiplication and clearly has size  $\phi(N)$ .

When  $N$  is a prime  $p$  we have

$$\mathbb{Z}/p\mathbb{Z}^* = \{1, \dots, p-1\}.$$

We define the sets

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, \dots, p-1\} \quad \text{and} \quad \mathbb{F}_p^* = (\mathbb{Z}/p\mathbb{Z})^* = \{1, \dots, p-1\}.$$

We call  $\mathbb{F}_p$  a **finite field** of **characteristic  $p$** .

Finite fields are of central importance in **coding theory** and **cryptography**.

# Finite Field Arithmetic

It is crucial to have a good finite field arithmetic.

A number of different choices have been proposed.

We shall now recap on the main two choices.

- ▶  $\mathbb{F}_{2^p}$ ,  $p$  prime
- ▶  $\mathbb{F}_p$ ,  $p$  prime

## Characteristic Two Fields

# Characteristic Two Fields

Of particular interest are fields of char 2.

Take an **irreducible** binary polynomial  $f$  of degree  $n$  and let  $\mathbb{F}_{2^n}$  denote all the binary polynomials of degree  $< n$ .

**Addition** in  $\mathbb{F}_{2^n}$  is defined as

- ▶  $a \oplus b = a + b \pmod{2}$
- ▶ Note this means  $-a = a$ .

**Multiplication** in  $\mathbb{F}_{2^n}$  is defined as

- ▶  $a \otimes b = a \cdot g \pmod{f}$ .
- ▶ Inversion is performed by a variant of the Euclidean algorithm for polynomials.

# Characteristic Two Fields

Often write

$$\mathbb{F}_{2^n} = \mathbb{F}_2[x]/f$$

to denote working modulo  $f$ .

Set of non-zero elements denoted by  $\mathbb{F}_{2^n}^*$

- ▶ This is the **multiplicative subgroup** of the field

## Char 2 Example

Let  $f = x^6 + x + 1$  (this is **irreducible**) The finite field of  $2^6$  elements can then be identified with

- ▶ Bit strings of length six bits
- ▶ Binary polynomials of degree less than or equal to five

$$a = 001101 = x^3 + x^2 + 1$$

$$b = 101011 = x^5 + x^3 + x + 1$$

$$a \oplus b = 100110 = x^5 + x^2 + x$$

- ▶ Since the two  $x^3$  and the two 1 terms cancel, as we are working mod two.
- ▶ Notice, we are simply taking the exclusive-or of the bit string representation.

## Char 2 Example

Recap  $f = x^6 + x + 1$ ,  $a = 001101 = x^3 + x^2 + 1$ ,  
 $b = 101011 = x^5 + x^3 + x + 1$ .

Since  $f$  is sparse reduction mod  $f$  done using rewriting, as  
 $x^6 = x + 1 \pmod{f}$ ,

$$\begin{aligned}a \otimes b &= (x^3 + x^2 + 1) \cdot (x^5 + x^3 + x + 1) \\&= x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x + 1 \\&= x^6 \cdot (x^2 + x + 1) + x^4 + x^3 + x^2 + x + 1 \\&= (x + 1) \cdot (x^2 + x + 1) + x^4 + x^3 + x^2 + x + 1 \\&= (x^3 + 1) + (x^4 + x^3 + x^2 + x + 1) \\&= x^4 + x^2 + x.\end{aligned}$$

i.e.  $a \otimes b = 010110 = x^4 + x^2 + x$ .

## Char 2 Example

Since  $f$  is assumed irreducible, every polynomial  $a \neq 0$  is coprime to  $f$ .

Hence, using a binary polynomial version of the extended GCD algorithm we can find  $u$  and  $v$  so that

$$u \cdot a + v \cdot f = 1 \pmod{2}.$$

In which case  $a^{-1} = u$  in  $\mathbb{F}_{2^n}$ .

If  $a = x^3 + x^2 + 1$  and  $f = x^6 + x + 1$  then taking  $u = x^5 + x^3$  and  $v = x^2 + x + 1$  gives us

$$\blacktriangleright u \cdot a + v \cdot f = 1 \pmod{2}$$

and so

$$\blacktriangleright a^{-1} = u = x^5 + x^3 = 101000.$$

## Choice of Defining Polynomial

All char 2 fields of the same degree  $n$  are **isomorphic**.

- ▶ This means it does not depend on which polynomial  $f$  we take.
- ▶ Different  $f$ 's give different representations of the same thing.

Let  $f(x)$  and  $g(y)$  be irreducible polynomials of degree  $n$ . Then there are polynomials  $r(x)$  and  $s(y)$  such that one can map one field into the other via

- ▶  $x \pmod{f(x)} \longrightarrow s(y) \pmod{g(y)}$
- ▶  $y \pmod{g(y)} \longrightarrow r(x) \pmod{f(x)}$

This means we can select the best irreducible polynomial  $f$  for our own implementation.

- ▶ Requires the mapping  $s(y)$  only when talking to someone else's implementation which uses  $g(y)$  instead.

## Characteristic Two : Composite Extension

These are fields of the form  $\mathbb{F}_{2^{n \cdot m}}$

For a while some people proposed these (of course backed up by patents).

- ▶ The IETF standards include such finite fields.
- ▶ They provide a number of performance advantages

Problem is that such fields are susceptible to [Weil Descent](#) attacks.

- ▶ Whilst such attacks are often not practical they cast sufficient concern to mean we no longer use such fields.

## Characteristic Two : Prime Extension

$K = \mathbb{F}_{2^p}$  With  $p$  prime, eg  $p = 163, 191$ .

- ▶ Trinomial Bases
- ▶ Pentanomial Bases
- ▶ Normal Bases

All are very good in hardware, normal bases are very good.

All three occur in standards documents ANSI/NIST etc.

- ▶ SECG does not support Normal Bases so as to aid interoperability.
- ▶ This is despite Certicom having loads of patents on Normal Bases.

## Large Prime Characteristic Fields

# Large Prime Characteristic

$$K = \mathbb{F}_p$$

Can be implemented in a number of ways.

- ▶ Montgomery arithmetic (general prime)
- ▶ Barrett Reduction (general prime)
- ▶ Generalised Mersenne Primes (special prime)

Most popular method for a general modulus is Montgomery arithmetic.

## Large Integer Arithmetic

# Multi-Precision Arithmetic

We first have to explained to perform arithmetic on 160 – 512 bit numbers.

It is common

- ▶ to represent all integers in **little wordian** format and
- ▶ to operate on them using adaptations of school-book arithmetic.

For a 32 bit machine and 64 bit numbers  $x$  and  $y$  we obtain

- ▶  $x = x_1 2^{32} + x_0$  and
- ▶  $y = y_1 2^{32} + y_0$ .

# Multi-Precision Arithmetic

## Addition

This is easy using the processor's `addc` instruction.

The sum

$$z = x + y = z_2 2^{64} + z_1 2^{32} + z_0$$

is computed using the following code.

```
add    x0, y0, z0
addc   x1, y1, z1
addc   0, 0, z2
```

# Multi-Precision Arithmetic

## Multiplication

Note that two 32-bit words multiply together to form a 64 bit result. Most processors have an instruction which will do this operation:

$$w_1 \cdot w_2 = (\text{High}, \text{Low}) = (H(w_1 \cdot w_2), L(w_1 \cdot w_2)).$$

We use this to do school-book **long multiplication**.

$$\begin{array}{r} \phantom{H(x_1 \cdot y_1)} \phantom{L(x_1 \cdot y_1)} \phantom{H(x_1 \cdot y_0)} \phantom{L(x_1 \cdot y_0)} \phantom{H(x_0 \cdot y_1)} \phantom{L(x_0 \cdot y_1)} \phantom{H(x_0 \cdot y_0)} \phantom{L(x_0 \cdot y_0)} \\ \phantom{H(x_1 \cdot y_1)} \phantom{L(x_1 \cdot y_1)} \phantom{H(x_1 \cdot y_0)} \phantom{L(x_1 \cdot y_0)} \phantom{H(x_0 \cdot y_1)} \phantom{L(x_0 \cdot y_1)} \phantom{H(x_0 \cdot y_0)} \phantom{L(x_0 \cdot y_0)} \\ \phantom{H(x_1 \cdot y_1)} \phantom{L(x_1 \cdot y_1)} \phantom{H(x_1 \cdot y_0)} \phantom{L(x_1 \cdot y_0)} \phantom{H(x_0 \cdot y_1)} \phantom{L(x_0 \cdot y_1)} \phantom{H(x_0 \cdot y_0)} \phantom{L(x_0 \cdot y_0)} \\ \phantom{H(x_1 \cdot y_1)} \phantom{L(x_1 \cdot y_1)} \phantom{H(x_1 \cdot y_0)} \phantom{L(x_1 \cdot y_0)} \phantom{H(x_0 \cdot y_1)} \phantom{L(x_0 \cdot y_1)} \phantom{H(x_0 \cdot y_0)} \phantom{L(x_0 \cdot y_0)} \\ \hline H(x_1 \cdot y_1) \quad L(x_1 \cdot y_1) \quad H(x_1 \cdot y_0) \quad L(x_1 \cdot y_0) \quad H(x_0 \cdot y_1) \quad L(x_0 \cdot y_1) \quad H(x_0 \cdot y_0) \quad L(x_0 \cdot y_0) \end{array}$$

We add up the four rows to get the answer.

Remember, we need to take care of the carries!

# Multi-Precision Arithmetic

## Modular Arithmetic

This is all well and good

- ▶ but we are doing modular arithmetic not standard arithmetic.

For each result we then need to take the remainder on division by  $N$ .

But integer division, let alone multi-precision division, is very slow.

For GM-primes this is easy: Which is why they are used.

For general primes we need another technique.

- ▶ We use a form of arithmetic called Montgomery Arithmetic.

## Montgomery Arithmetic

# Montgomery Arithmetic

Let  $b$  denote the word size

▶  $b = 2^{32}$  or  $2^{64}$ .

Then choose

▶  $R = b^t > N$ .

Instead of holding the value of the integer  $x$  in memory we hold the value

$$x \cdot R \pmod{N}.$$

This is called the **Montgomery representation**.

Adding two elements is easy.

Suppose  $z = x + y \pmod{N}$ .

We compute  $zR$  from  $xR$  and  $yR$  as follows.

$$\begin{aligned} zR &= xR + yR \\ \text{if } (zR \geq N) \{ & zR -= N; \} \end{aligned}$$

## Example

Let us take a simple example

- ▶  $N = 1073741827$  and
- ▶  $b = R = 2^{32} = 4294967296$

The following is the map from the normal to the Montgomery representation.

- ▶  $1 \rightarrow R \pmod{N} = 1073741815$
- ▶  $2 \rightarrow 2R \pmod{N} = 1073741803$
- ▶  $3 \rightarrow 3R \pmod{N} = 1073741791$

Now lets check that addition works.

- ▶  $1 + 2 = 3$
- ▶  $1073741815 + 1073741803 = 1073741791 \pmod{N}$

# Montgomery Arithmetic

Now we look at multiplication of  $xR$  and  $yR$ :

$$(xR) \cdot (yR) = xyR^2$$

but we want  $xyR$ .

- ▶ We need to divide the result by  $R$ .
- ▶ This should be easy since  $R$  is a power of the word size.

**Montgomery reduction:** Given  $y$ , the following computes  $x = y/R \pmod{N}$  given the earlier choice of  $R$ .

Precompute  $N' = 1/N \pmod{R}$ , then compute the following steps.

```
u = -y * N' mod R; // Reduction mod R is easy Why ?
x = (y + u * N) / R; // Division by R is easy Why ?
if (x >= N) { x -= N; }
```

# Example

Again taking

- ▶  $N = 1073741827$  and
- ▶  $b = R = 2^{32} = 4294967296$ .

Take the numbers

- ▶  $2 \rightarrow 2R \pmod{N} = 1073741803 = x$  and
- ▶  $3 \rightarrow 3R \pmod{N} = 1073741791 = y$ .

Compute

$$z = x \times y = 1152921446624789173 = 2 \times 3 \times R^2 \pmod{N}.$$

## Example

We now need to **Montgomery reduce** this number to find the Montgomery representation of  $x \cdot y$ .

- ▶  $z = 1152921446624789173$
- ▶  $N' = (1/N) \pmod{R} = 1789569707$
- ▶  $u = -z \cdot N' \pmod{R} = 3221225241$
- ▶  $z = (z + u \cdot N)/R = 1073741755$

So the product of  $x$  and  $y$  in Montgomery representation should be

- ▶  $1073741755$ .

Check that

- ▶  $6 \cdot R \pmod{N} = 1073741755$ .

# Montgomery Arithmetic

Montgomery reduction requires

- ▶ **two** multi-precision multiplications and
- ▶ **zero** multi-precision divisions.

So, to multiply two numbers using Montgomery multiplication we only require

- ▶ **three** multi-precision multiplications and
- ▶ **zero** multi-precision divisions.

But we can do better...

# Montgomery Arithmetic

Suppose that  $y$  is given in little wordian format

$$y = (y_{2t-1}, y_{2t-2}, \dots, y_1, y_0).$$

A better way to perform **Montgomery reduction** as follows.

Precompute  $N' = -1/N \pmod{b}$ .

```
z=y;
for (i=0; i<t; i++)
  { u=z_i*N' mod b;
    z+=u*N*b^i;
  }
z/=R;
if (z>=N) { z-=N; }
```

Note the following.

- ▶ Multiplication and reduction by  $b$  are simple. (Why?)
- ▶ The above algorithm uses  $t$  multi-precision additions.

## Montgomery Arithmetic

We can also interleave the multiplication with the reduction; we perform a single loop to produce

$$Z = XY/R \pmod{N}.$$

If  $X = xR$  and  $Y = yR$  this will produce

$$Z = (xy)R.$$

This procedure is called **Montgomery multiplication**. It works as follows.

Precompute  $N' = -1/N \pmod{b}$ .

```
Z=0;
for (i=0; i<t; i++)
  { u=(Z_0+X_i*Y_0)*N' mod b;
    Z=(Z+X_i*Y+u*N)/b;
  }
if (Z>=N) { Z-=N; }
```

## Montgomery Arithmetic

This is all very well, but how do we get  $xR \pmod{N}$  from  $x$  without doing multi-precision division?

Suppose we have a Montgomery multiplier  $Mmult$  that takes as input  $X$  and  $Y$  and outputs

$$Mmult(X, Y) = XY/R \pmod{N}.$$

Suppose we also have  $R' = R^2 \pmod{N}$ .

Now, if we want the Montgomery representation of  $x$ , we compute

$$Mmult(x, R') = xR^2/R = xR \pmod{N}.$$

To get  $x$  back from  $xR \pmod{N}$  we do

$$Mmult(1, xR) = xR/R = x \pmod{N}.$$

# Montgomery Arithmetic

We still need to be able to compute  $R' = R^2 \pmod{N}$ .

We know that  $R = 2^a$  for some  $a$ .

We compute  $R'$  as follows.

```
R' = 1;
for (i=0; i<2a; i++)
  { R' *= 2;
    if (R' >= N) { R' -= N; }
  }
```

So, there you have it: division-free modular arithmetic!

## Montgomery Arithmetic

We also need to compute  $N' = 1/N \pmod{b}$  where  $b = 2^w$ .

This is done via the following simple algorithm

```
y=1;
for (i=2; i<=w; i++)
  { r=1<<(i-1);
    mask=(r<<1)-1;
    s=(y*N)&mask;
    if (r<s) { y=y+r; }
  }
return y;
```

Note,

- ▶  $1 \ll (i-1) = 2^{i-1}$ .
- ▶  $A \& ((1 \ll i) - 1) = A \pmod{2^i}$ .

## GM Primes

# GM Primes

$$K = \mathbb{F}_p$$

Standards bodies have settled on GM-Primes as the main recommend fields.

- ▶ GM-Primes give significant performance advantages.
- ▶ Their special form means one has **significant** performance improvements.
- ▶ Montgomery Mult takes about  $2n(n+1)$  word multiplications, whereas for GM-Primes this is only  $n^2$ .

A GM-prime is one of the form

$$p = f(2^{32}) \text{ or } f(2^{64}),$$

where  $f$  is a "low weight" polynomial.

- ▶ Eg.  $p = 2^{192} - 2^{64} - 1$  is popular.

# GM Primes

Suppose we take  $p = 2^{192} - 2^{64} - 1$  as an example and we want to compute

$$z = x \cdot y \pmod{p}.$$

To perform modular multiplication we first do a standard school book (or Karatsuba) multiplication

$$a = x \cdot y = a_1 2^{192} + a_0$$

where  $a_0, a_1 < 2^{192}$ .

- ▶ This requires at most 36 32-bit word multiplications plus some additions

# GM Primes

We now need to produce  $z = a \pmod{p}$ , but note that mod  $p$  we have

$$2^{192} = 2^{64} + 1$$

and so

$$\begin{aligned} a &\equiv a_1(2^{64} + 1) + a_0 \pmod{p} \\ &= b_1 2^{192} + b_0, \end{aligned}$$

where  $b_0 < 2^{192}$  but  $b_1 < 2^{65}$ .

We then repeat to obtain

$$\begin{aligned} b &\equiv b_1(2^{64} + 1) + b_0 \pmod{p} \\ &= z. \end{aligned}$$

# GM Primes

Notice the reduction stage just involved some shifting and addition  
▶ i.e. very cheap operations.

This total time is dominated by the 36 32-bit word multiplications.

If we did Montgomery arithmetic on similar size numbers we would require

$$2 \cdot 6 \cdot (6 + 1) = 84$$

32-bit word multiplications.

Hence the GM-Prime version will be around twice as fast.

## OEF Fields

# OEF Fields

$$K = \mathbb{F}_{p^n}$$

These have appeared in a number of recent papers.

## OEF Fields.

- ▶ OEF fields choose  $p$  close to the word size.
- ▶ The equation defining  $K$  over  $\mathbb{F}_p$  is chosen to be very simple,

$$x^n - 2.$$

OEF Fields give very good implementations.

- ▶ Very fast field inversion.

Not adopted by main standards bodies.

- ▶ P1363a thinking of doing so.
- ▶ Could Weil-descent work in this case ?