Effective Arithmetic in Finite Fields Based on Chudnovsky Multiplication Algorithm

Kevin Atighehchi [1] **Stéphane Ballet** [2]
Alexis Bonnecaze [2] Robert Rolland [2]

[1] Aix-Marseille Université, CNRS, Laboratoire d'Informatique Fondamentale de Marseille, UMR 7279

[2] Aix-Marseille Université, CNRS, Institut de Mathématiques de Marseille, UMR 7373

Crypto'puces 2017, 29 Mai–2 Juin

Summary

1. Introduction
2. Multiplication
   - Algorithm of Chudnovsky-Chudnovsky
   - Representation of type normal basis
   - Strategy of implementation
   - Product of 2 elements
   - Setup algorithm
3. Exponentiation
   - A first algorithm
   - A variant of von Zur Gathen algorithm
   - Using Coppersmith-Winograd multiplication

Introduction

Multiplication in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$:

Let $\mathcal{B} = \{e_1, ..., e_n\}$ be a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

If $x = \sum_{i=1}^{n} x_i e_i$ et $y = \sum_{i=1}^{n} y_i e_i$ then:

$$z = xy = \sum_{h=1}^{n} z_h e_h = \sum_{h=1}^{n} \left(\sum_{i,j=1}^{n} t_{ijh} x_i x_j \right) e_h, \qquad (1)$$

where

$$e_i e_j = \sum_{h=1}^{n} t_{ijh} e_h,$$

$t_{ijh} \in \mathbb{F}_q$ being constants.

Complexity of the multiplication in $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$:

Minimal number of elementary operations in $\mathbb{F}_q$ required to compute the product of two elements $x, y \in \mathbb{F}_{q^n}$.

Types of opérations :

- addition : $(\alpha, \beta) \mapsto \alpha + \beta$ où $\alpha, \beta \in \mathbb{F}_q$,
- scalar multiplication : $x_i \mapsto \alpha \cdot x_i$ where $\alpha, x_i \in \mathbb{F}_q$, and α is a constant,
- non-scalar or bilinear multiplication : $(x_i, y_j) \mapsto x_i \cdot y_j$ where $x_i, y_j \in \mathbb{F}_q$ depend on the elements x and y of $\mathbb{F}_{q^n}$ which are multiplied.

Problem and Aims

Problem: efficient multiplication and exponentation in $\mathbb{F}_{q^n}$.

Theorem (Couveignes and Lercier 2008)

There is a model $\Xi(q,n)$ for $\mathbb{F}_{q^n}$ such that the following holds:

- *Elements in $\mathbb{F}_{q^n}$ are represented by vectors of at most $Kn(\log n)^2(\log(\log n))^2$ components in $\mathbb{F}_q$.*
- *Product in $O(n(\log n)^4|\log(\log n)|^3)$ mult. in $\mathbb{F}_q$.*
- *Exponentiation by q consists in a circular shift of the coordinates.*

Bilinear complexity of Ξ: $O(n(\log n)^2(\log(\log n))^2)$.

Models used in this work (in terms of operations in $\mathbb{F}_q$):

- Non-Scalar model (bilinear multiplications),
- S1 model (scalar/bilinear mult.),
- S2 model (any operations).

News results

Chudnovsky and Chudnovsky Multiplication Algorithm (CCMA):

- Interpolation on algebraic curves
- Bilinear complexity in $O(n)$
- The use of matrices promotes parallel computation

This work:

- An explicit multiplication algorithm based upon CCMA and suitable to exponentiation
- New Design with integration of a representation of type normal basis in the Riemann-Roch spaces
- Establishing interesting asymptotic bounds for parallel multiplication and exponentiation

General Principles of CCMA

Theorem

Let $F/\mathbb{F}_q$ be an algebraic function field of genus g over $\mathbb{F}_q$. Let us suppose that there exists a place Q of degree n. Then, if $N_1(F/\mathbb{F}_q) > 2n + 2g - 2$ there is a divisor D of degree $n + g - 1$ such that:

1. *Q is not in the support of D,*
2. *the evaluation map E defined by*

$$\begin{array}{rccc} E: & \mathcal{L}(D) & \to & F_Q = O_Q/Q \simeq \mathbb{F}_{q^n} \\ & f & \mapsto & f(Q) \end{array}$$

   is an isomorphism of vector spaces over $\mathbb{F}_q$,
3. *there exist $2n + g - 1$ places of degree one P_i which are not in the support of D such that the multi-evaluation map T defined by*

$$\begin{array}{rccc} T: & \mathcal{L}(2D) & \to & (\mathbb{F}_q)^{2n+g-1} \\ & f & \mapsto & \left(f(P_1), \ldots, f(P_{2n+g-1})\right) \end{array}$$

   is an isomorphism.

Representation of type normal basis

Definition:
Let α be an element of $\mathbb{F}_{q^n}$ such that

$$\left(\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}\right)$$

is a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$. Such a basis is called a **normal basis** of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, constructed using a **normal polynomial** of degree n over $\mathbb{F}_q$.

Benefits:

- the qth power of an element is just a cyclic shift of its coordinates;
- allows the computation of x^k to be efficiently parallelized without prior storage of lookup tables for a fixed base x.

Strategy of implementation

For the construction of our spaces:

- D is a place of degree $n+g-1$.
- Q is a **normal place** of degree n.
- $D-Q$ is non-special of degree $g-1$, *i.e.* $dim\mathcal{L}(D-Q)=0$.
- Let a **normal basis** $\mathcal{B}_Q=(\phi_1,\ldots,\phi_n)$ of F_Q. The basis $\mathcal{B}_D$ of $\mathcal{L}(D)$ is such that

$$\mathcal{B}_D=(E^{-1}(\phi_1),\ldots,E^{-1}(\phi_n)).$$

- Let $\mathcal{M}=\{f\in\mathcal{L}(2D)\mid f(Q)=0\}$. Then,

$$\mathcal{L}(2D)=\mathcal{L}(D)\oplus\mathcal{M}.$$

Product of 2 elements

- Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, u_n)$ be two elements of $\mathbb{F}_{q^n}$.
- Since $\mathbb{F}_{q^n}$ is isomorphic to $\mathcal{L}(D)$, x and y are identified to elements of $\mathcal{L}(D)$

$$f_x = \sum_{i=1}^{n} x_i f_i \text{ and } f_y = \sum_{i=1}^{n} y_i f_i.$$

- f_x and f_y are elements of $\mathcal{L}(2D)$ where the $n + g - 1$ last components are 0.

Theorem

The product of x by y is such that

$$x \cdot y = P\left(T^{-1}\left(T(f_x) \odot T(f_y)\right)\right)$$

where P is a projection map from $\mathcal{L}(2D)$ onto $\mathcal{L}(D)$.

Product of 2 elements (ctd)

- After having computed $f_z = T(f_x)$ and $f_t = T(f_y)$, the product in $(\mathbb{F}_q)^{2n+g-1}$ is as follows:

$$f_z \odot f_t = (z_1, \ldots, z_{2n+g-1}) \odot (t_1, \ldots, t_{2n+g-1})$$

$$= (z_1 t_1, \ldots, z_{2n+g-1} t_{2n+g-1}).$$

- Then,

$$T^{-1}(f_z \odot f_t) = h = (h_1, \ldots, h_n, h_{n+1}, \ldots, h_{2n+g-1}).$$

- The result:

$$x \cdot y = (h_1, \ldots, h_n, 0, \ldots, 0).$$

Setup algorithm

INPUTS: $F/\mathbb{F}_q$, $Q, D, P_1, \ldots, P_{2n+g-1}$.
OUTPUTS: T and T^{-1}.

1. An element x of $\mathbb{F}_{q^n}$ is represented in a fixed basis: $x = (x_1, \ldots, x_n)$ (where $x_i \in \mathbb{F}_q$).
2. The function field $F/\mathbb{F}_q$, the place Q, the divisor D and the points $P_1, \ldots, P_{2n+g-1}$ are as in the theorem.
3. Construct a basis $(f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n+g-1})$ of $\mathcal{L}(2D)$ where $(f_1, \ldots, f_n)$ is the basis of $\mathcal{L}(D)$ and $(f_{n+1}, \ldots, f_{2n+g-1})$ a basis of $\mathcal{M}$.
4. Any element $x = (x_1, \ldots, x_n)$ in $\mathbb{F}_{q^n}$ is identified to the element $f_x = \sum_{i=1}^{n} x_i f_i$ of $\mathcal{L}(D)$.
5. Compute the matrices T and T^{-1}.

Exponentiation in $\mathbb{F}_{q^n}$

Computing x^k with the Square and Multiply algorithm, assuming the use of a normal basis?

According to Fermat's Little Theorem, we may assume that the exponent k satisfies $0 \leq k < q^n$. We take the q-ary representation of k:

$$k = \sum_{i=0}^{n-1} K_i q^i,$$

with $2 \leq K_i < q$.

Exponentiation in $\mathbb{F}_{q^n}$

Let $\sigma(x, i)$ a function that right shifts i times the vector x.

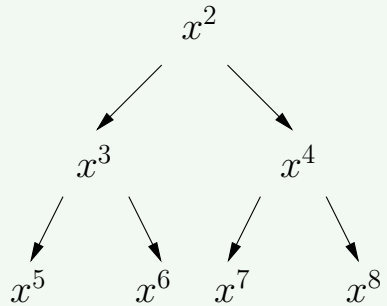
We could compute x^k as:

$$\prod_{i=0}^{n-1} \sigma(x^{K_i}, i).$$

Parellel time in terms of multiplications in $\mathbb{F}_{q^n}$:

- Computation of all x^j, for all $j < q$, in $O(\log q)$,

Example with $q = 9$



- A cyclic shift is free,
- The outer product is done in $O(\log n)$, using $O(n)$ processors.

A variant of von zur Gathen algorithm

How can we reduce the number of processors?

Let us set:

- $r = \lceil \log_q^2 n - 2\log_q(n)\log_q\log_q n \rceil$,
- $u = \lfloor \log_q n - 2\log_q\log_q n \rfloor$,

and rewrite the exponent:

$$k = \sum_{i=0}^{s-1} \left(\sum_{j=0}^{t-1} K_{i,j} q^{uj} \right) q^{ri}$$

with $s = \lceil n/r \rceil = \mathcal{O}(n/\log^2 n)$ and $t = \lceil r/u \rceil = \mathcal{O}(\log n)$.

A variant of von zur Gathen algorithm (ctd)

The computation of x^k consists of five steps:

1. for $2 \leq l < q^u$, compute x^l,
2. for $0 \leq i < s$ and $0 \leq j < t$, compute $y_{i,j} = \sigma\left(x^{K_{i,j}}, uj\right)$,
3. for $0 \leq i < s$, compute $y_i = \prod_{j=0}^{t-1} y_{i,j}$,
4. for $0 \leq i < s$, compute $z_i = \sigma\left(y_i, ri\right)$,
5. return $x^k = \prod_{i=0}^{s-1} z_i$.

Efficiency in terms of multiplications in $\mathbb{F}_{q^n}$:

- step 1 in $O(\log n)$ using $O(q^u) = O(n/\log^2 n)$ processors,
- step 2 is free,
- step 3 in $O(\log n)$ using $s = O(n/\log^2 n)$ processors,
- step 4 is free,
- step 5 in $O(\log n)$ using $s/2 = O(n/\log^2 n)$ processors,

A variant of von zur Gathen algorithm (ctd)

We scale up the number of processors to optimize the running time in terms of operations in $\mathbb{F}_q$:

- $\mathcal{O}(n/\log^2 n)$ sets of $\mathcal{O}(n)$ processors in the NS model,
- $\mathcal{O}(n/\log^2 n)$ sets of $\mathcal{O}(n^2)$ processors in the S1 model,
- $\mathcal{O}(n/\log^2 n)$ sets of $\mathcal{O}(n^2/\log n)$ processors in the S2 model.

Efficiency in terms of operations in $\mathbb{F}_q$:

- NS model: depth $\mathcal{O}(\log n)$ and width $\mathcal{O}(n^2/\log^2 n)$,
- S1 model: depth $\mathcal{O}(\log n)$ and width $\mathcal{O}(n^3/\log^2 n)$,
- S2 model: depth $\mathcal{O}(\log^2 n)$ and width $\mathcal{O}(n^3/\log^3 n)$.
  - $\mathcal{O}(\log n)$ mult. in $\mathbb{F}_{q^n}$, each of which being done in $\mathcal{O}(\log n)$ op. in $\mathbb{F}_q$.
  - These $\mathcal{O}(\log n)$ op. in $\mathbb{F}_q$ can be done using $O(n^2/\log n)$ processors.

Reducing the number of scalar multiplications

Matrix-vector products involved:

$$Tx,\ T \circ P \circ T^{-1}(x) \text{ and } T^{-1}x.$$

Among all the parallel steps, the maximum number of such products in a step is in $\Theta(n/\log^2 n)$. We regroup all the vectors involved in a matrix B, and compute a matrix product TB using Coppersmith-Winograd algorithm.

Theorem

Parallel complexity of the exponentiation, in terms of scalar operations in $\mathbb{F}_q$:

- *S1 model: $\mathcal{O}(\log n)$ mult. using $\mathcal{O}(n^{2.38}/\log^{0.76} n)$ proc.*
- *S2 model: $\mathcal{O}(\log^2 n)$ op. using $\mathcal{O}(n^{2.38}/\log^{1.76} n)$ proc.*

Thanks for your attention...

Questions ?