EFFICIENT ROOT FINDING OF POLYNOMIALS OVER FIELDS OF CHARACTERISTIC 2

Vincent Herbert (Joint work with Bhaskar Biswas)

WEWoRC 2009

INRIA Paris Rocquencourt

Agenda

1. Motivation for code-based cryptography
2. Algorithms & Complexities
3. Speed Up McEliece Decryption
4. Results & Analysis

Why do we study Polynomial Root Finding ?

We face this problem in code-based cryptography.

Indeed, McEliece-type cryptosystems are often based on Binary Goppa codes.

Root finding is the most time-consuming step, in the implementation of algebraic decoding of Binary Goppa codes.

R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. JPL DSN Progress Report, pages 114 - 116, 1978.

What is McEliece Public Key Cryptosystem ?

Let us have an insight of the original version of McEliece.

Public key : A binary linear [n,k] code C, *i.e.* a k-dimensional linear $\mathbb{F}_2$-subspace of $\mathbb{F}_2^n$, described by a generator matrix G.
Private key : An efficient decoding algorithm for C up to the error correcting capacity t.

Encryption : Map the k bits plaintext x to the codeword $x.G$, add e, an uniformly random error of length n and weight t.
Decryption : Correct the t errors, unmap to get the message. This process is also called decoding.

What is a Binary Goppa Code ?

Let $m > 0$, $n \leq 2^m$ and $a = (a_1, ..., a_n) \in \mathbb{F}_2^n$.

The n-length binary Goppa code $\Gamma(L, g)$ is defined by :

- Support $L = (\alpha_1, ..., \alpha_n)$ n-tuple of distinct elements of $\mathbb{F}_{2^m}$;
- Goppa polynomial $g(z) \in \mathbb{F}_{2^m}[z]$, square-free, monic of degree $t > 0$ with no root in L.

$\Gamma(L, g)$ is a subfield subcode over $\mathbb{F}_2$ of a particular Goppa code over the binary field $\mathbb{F}_{2^m}$.

We have $a \in \Gamma(L, g)$ if and only if :

$$R_a(z) := \sum_{i=1}^{n} \frac{a_i}{z - \alpha_i} = 0 \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$

How to decode Binary Goppa Codes ?

Let e, x, y be n-length binary vectors. We have to find x, the sent codeword knowing $y = x + e$ where y is the received word and e the error word. We can correct up to t errors.

Algebraic decoding is carried out in three steps :

1. Syndrome computation

$$R_y(z) = R_e(z) = \sum_{i=1}^{n} \frac{e_i}{z - \alpha_i} \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$

2. Solving the Key Equation to obtain the error locator polynomial

$$R_e(z) \cdot \sigma_e(z) = \sigma'_e(z) \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$

3. Error Locator Polynomial Root Finding

$$\sigma_e(z) := \prod_{i=1}^{n} (z - \alpha_i)^{e_i} \; ; \; \sigma_e(\alpha_i) = 0 \Leftrightarrow e_i \neq 0.$$

How to find the roots efficiently ?

Several approaches are possible, their efficiency depends on the size of parameters m and t.

- Chien search computes roots by evaluating artfully the polynomial in all points of L. This method is recommended for hardware implementations and coding theory applications in which m is small.
- BTA is a recursive algorithm using trace function properties. It is a faster method for secure parameters in McEliece-type cryptosystems.

What is the cost of the decryption ?

Let us recall, in practice, $n = 2^m$ and $mt \leq n$.

Theoretical Complexity = number of binary operations required to decrypt in the worst case.

- Syndrome computation $\mathcal{O}(mnt)$
- Key Equation Solving (w/ Patterson algorithm) $\mathcal{O}(mt^2)$
- Error Locator Polynomial Root Finding
  - Chien search $\mathcal{O}(mnt)$
  - Berlekamp Trace Algorithm (abbr. BTA) $\mathcal{O}(m^2t^2)$

Experimental Complexity = average running time for the decryption.

For recommended parameters (*i.e.* $m = 11$, $t = 32$), root finding with BTA (resp. Chien search) takes 72% (resp. 86%) of the total decryption time.

How does BTA work ?

Trace function $\mathrm{Tr}(\cdot) : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2$

$$\mathrm{Tr}(z) := z + z^2 + z^{2^2} + \ldots + z^{2^{m-1}}.$$

The function $\mathrm{Tr}(\cdot)$ is $\mathbb{F}_2$-linear and onto.

We know that :

$$\forall i \in \mathbb{F}_2 \ , \ \mathrm{Tr}(z) - i = \prod_{\gamma \ s.t. \ \mathrm{Tr}(\gamma)=i} (z - \gamma).$$

Moreover, we have : $z^{2^m} - z = \mathrm{Tr}(z) \cdot (\mathrm{Tr}(z) - 1)$.

How does BTA work ? (contd)

Let $B = (\beta_1, \ldots, \beta_m)$ a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$.

Every $\alpha \in \mathbb{F}_{2^m}$ is uniquely represented by the m-tuple :

$$(\mathrm{Tr}(\beta_1 \cdot \alpha), \ldots, \mathrm{Tr}(\beta_m \cdot \alpha)).$$

BTA splits any $f \in \mathbb{F}_{2^m}[z]$ s.t. $f(z)|(z^{2^m} - z)$ into linear factors by computing iteratively on $\beta \in B$ and recursively on f :

$$g(z) := \gcd(f(z), \mathrm{Tr}(\beta \cdot z)) \text{ and } h(z) := \frac{f(z)}{g(z)}.$$

BTA always successfully returns the linear factors of f.

First call : $f = \sigma_e$ and $\beta = \beta_1$.

How to reduce time complexity ?

The drawback of BTA is the large number of recursive calls when the system parameters grow.

We reduce it by mixing BTA and Zinoviev's algorithms which are ad-hoc methods for finding roots of polynomials of degree ≤ 10 over $\mathbb{F}_{2^m}$.

We call this process BTZ in the following.

BTZ depends on a parameter d_{max} which is the maximum degree up to which we use Zinoviev's methods.

V.A. Zinoviev, On the solution of equations of degree ≤ 10 over finite fields GF(2^m), Research Report INRIA n° 2829, 1996

Pseudocode of a simplified version of BTZ

Algorithm 1 - BTZ(f, d, i)

First call : $f \leftarrow \sigma_e$; $d \leftarrow d_{max} \in \{2, \ldots, 10\}$; $i \leftarrow 1$.
if degree$(f) \leq d$ **then**
  return ZINOVIEV(f, d);
else
  $g \leftarrow \gcd(f, \mathrm{Tr}(\beta_i \cdot z))$;
  $h \leftarrow f/g$;
  return BTZ$(g, d, i+1) \cup$ BTZ$(h, d, i+1)$;
end if

What are Zinoviev's algorithms ?

Zinoviev's methods find an affine multiple of any polynomial of degree ≤ 10 over $\mathbb{F}_{2^m}$. The methods differ according to this degree.

Affine Polynomial

$A(z) = L(z) + c$ where L is a linearized polynomial, $c \in \mathbb{F}_{q^m}$.

Linearized Polynomial

$$L(z) = \sum_{i=0}^{n} l_i \cdot z^{q^i}$$

with q a prime power, $l_i \in \mathbb{F}_{q^m}$ and $l_n = 1$. In our case, $q = 2$.

After that, finding roots of affine polynomial is easier than in the general case.

Get an affine multiple of a polynomial of degree 2 or 3

Let us have an equation : $z^2 + \alpha z + \beta = 0, \qquad \alpha, \beta \in \mathbb{F}_{2^m}$.

Notice $z^2 + \alpha z$ is already a linearized polynomial. Nothing to do here.

Now consider the equation : $z^3 + az^2 + bz + c = 0, \qquad a, b, c \in \mathbb{F}_{2^m}$

We have to decimate the non-linear terms.

For this, we add one particular root by multiplying the left side by $(z + a)$.

We obtain $z^4 + dz^2 + ez + f = 0$ with $d = a^2 + b$, $e = ab + c$, $f = ac$.

We get what we want, an affine multiple of a polynomial of degree 3.

What results do we obtain ?

We specify a recurrence complexity formula for BTZ.

We then use dynamic programming to estimate its theoretical complexity in the worst case.

We thus determine the best d_{max} to use to have the optimal efficiency on the following range of parameters :

$m = 8, 11, 12, 13, 14, 15, 16, 20, 30, 40$; $t = 10..300$; $d_{max} = 2..10$.

Let K be the cost function of any operation over $\mathbb{F}_{2^m}$.
We take $K(+) = 1$; $K(\times) = 1$ or $K(\times) = m$.

Conclusions & Perspectives

For $m = 11$, $t = 32$, theory recommends $d_{max} = 5$.

Theoretical gain, in terms of number of operations over $\mathbb{F}_{2^m}$, of BTZ with $d_{max} = 5$ over BTA is 46%, the one over Chien method is 93%.

The higher is t, the higher is the optimal d_{max}, according to the theory.

Practice confirms theory up to degree 3 at least.

For instance with $m = 11$, $t = 32$ and $d_{max} = 2$, BTZ takes 65% of the total time decryption against 72% for BTA and 86% for Chien.

Implementation is in progress for greater parameters d_{max}.

Danke schön WEWoRC 2009 !

Any questions or comments ?

Any further remarks or suggestions can be adressed at :

Vincent.Herbert@inria.fr

Slides will be available in a short time on :

`http://www-roc.inria.fr/secret/Vincent.Herbert/`

Why is it easier to find roots of an affine polynomial ?

Let us have an affine polynomial $A(z) = L(z) + c = \sum_{i=0}^{m-1} l_i \cdot z^{2^i} + c$.

Consider $(\alpha_1, \cdots, \alpha_m)$ is a $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, $(l_i)_{1 \leq i \leq m}$, c and x are elements of $\mathbb{F}_{2^m}$.

Guess x is a root of A.

$$
\begin{aligned}
A(x) = 0 \quad &\Leftrightarrow \quad L(x) = c \\
&\Leftrightarrow \quad \sum_{i=1}^{m} x_i \cdot L(\alpha_i) = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(using linearity of } L\text{)} \\
&\Leftrightarrow \quad \sum_{i=1}^{m} \sum_{j=1}^{m} x_i \; l_{i,j} \cdot \alpha_i = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(linear system in } x_i\text{)}
\end{aligned}
$$

How does Chien search operate ?

Chien search is a recursive algorithm. We can say it's a clever exhaustive search.

Let α be a generator of $\mathbb{F}_{2^m}^*$ and let $f(x) = a_0 + a_1 \cdot x + \cdots + a_t \cdot x^t$ be a polynomial over $\mathbb{F}_{2^m}$.

$$\begin{aligned} f(\alpha^i) &= a_0 + a_1 \cdot \alpha^i + \cdots + a_t \cdot (\alpha^i)^t \\ f(\alpha^{i+1}) &= a_0 + a_1 \cdot \alpha^{i+1} + \cdots + a_t \cdot (\alpha^{i+1})^t \\ &= a_0 + a_1 \cdot \alpha^i \cdot \alpha + \cdots + a_t \cdot (\alpha^i)^t \cdot \alpha^t \end{aligned}$$

Set $a_{i,j} = a_j(\alpha^i)^j$. It is easy to obtain $f(\alpha^{i+1})$ from $f(\alpha^i)$ since we have that $a_{i+1,j} = a_{i,j} \cdot \alpha^j$.

Moreover, if $\sum_{j=0}^{t} a_{i,j} = 0$, then α^i is a root of f.

Second description of a Binary Goppa Code

Let $m > 0$ and $n \leq 2^m$.

The n-length binary Goppa code $\Gamma(L, g)$ is defined by :

- Support $L = (\alpha_1, ..., \alpha_n)$ n-tuple of distinct elements of $\mathbb{F}_{2^m}$;
- Goppa polynomial $g(z) \in \mathbb{F}_{2^m}[z]$, square-free, monic of degree $t > 0$ with no roots in L ;

$\Gamma(L, g)$ is a subfield subcode over $\mathbb{F}_2$ of a particular Goppa code over binary field $\mathbb{F}_{2^m}$ which have parity-check matrix H.

$$H := \begin{pmatrix} \frac{1}{g(\alpha_1)} & \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_1^{t-1}}{g(\alpha_1)} \\ \frac{1}{g(\alpha_2)} & \frac{\alpha_2}{g(\alpha_2)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} \\ \vdots & \vdots & \vdots \\ \frac{1}{g(\alpha_n)} & \frac{\alpha_n}{g(\alpha_n)} & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{pmatrix} \in \mathscr{M}_{n,t}(\mathbb{F}_{2^m}).$$

Thus, we have $a \in \Gamma(L, g)$ if and only if $a.H = 0$ and $a \in \mathbb{F}_2^n$.