A Generalised Successive Resultants Algorithm

James Davenport[1], Christophe Petit[2] and Benjamin Pring[2]

[1]University of Bath

[2]University of Oxford

July 14, 2016
WAIFI 2016 - Ghent, Belgium

Root Finding in Finite Fields (I))

Formally...

Definition (Root Finding Problem over Finite Fields)

Given $f \in \mathbb{F}_{p^n}[x]$, where f is a univariate, seperable polynomial which splits over $\mathbb{F}_{p^n}$,

$$f(x) = \prod_{i=1}^{d}(x - a_i) \qquad a_i \neq a_j \text{ for } i \neq j, a_i \in \mathbb{F}_{p^n},$$

the *Root Finding Problem* consists of computing $a_1, \ldots, a_d$.

- Deterministic and randomised algorithms.
- Reduction of factoring in to root finding .
- Reduction of root finding in $\mathbb{F}_{p^n}$ to root finding in $\mathbb{F}_p$ by Berklekamp (1970).
- Many different algorithms and strategies in literature: Direct root finding, factoring, Berlekamp, Cantor-Zassenhaus, Kedlaya and Umans, Graeffe Transforms,...

Root Finding in Finite Fields (I))

Formally...

Definition (Root Finding Problem over Finite Fields)

Given $f \in \mathbb{F}_{p^n}[x]$, where f is a univariate, seperable polynomial which splits over $\mathbb{F}_{p^n}$,

$$f(x) = \prod_{i=1}^{d}(x - a_i) \qquad a_i \neq a_j \text{ for } i \neq j, a_i \in \mathbb{F}_{p^n},$$

the *Root Finding Problem* consists of computing $a_1, \ldots, a_d$.

- Deterministic and randomised algorithms.
- Reduction of factoring in to root finding .
- Reduction of root finding in $\mathbb{F}_{p^n}$ to root finding in $\mathbb{F}_p$ by Berklekamp (1970).
- Many different algorithms and strategies in literature: Direct root finding, factoring, Berlekamp, Cantor-Zassenhaus, Kedlaya and Umans, Graeffe Transforms,...

Root Finding in Finite Fields (I))

Formally...

Definition (Root Finding Problem over Finite Fields)

Given $f \in \mathbb{F}_{p^n}[x]$, where f is a univariate, seperable polynomial which splits over $\mathbb{F}_{p^n}$,

$$f(x) = \prod_{i=1}^{d}(x - a_i) \qquad a_i \neq a_j \text{ for } i \neq j, a_i \in \mathbb{F}_{p^n},$$

the *Root Finding Problem* consists of computing $a_1, \ldots, a_d$.

- Deterministic and randomised algorithms.
- Reduction of factoring in to root finding .
- Reduction of root finding in $\mathbb{F}_{p^n}$ to root finding in $\mathbb{F}_p$ by Berklekamp (1970).
- Many different algorithms and strategies in literature: Direct root finding, factoring, Berlekamp, Cantor-Zassenhaus, Kedlaya and Umans, Graeffe Transforms,...

Root Finding in Finite Fields (I))

Formally...

Definition (Root Finding Problem over Finite Fields)

Given $f \in \mathbb{F}_{p^n}[x]$, where f is a univariate, seperable polynomial which splits over $\mathbb{F}_{p^n}$,

$$f(x) = \prod_{i=1}^{d}(x - a_i) \qquad a_i \neq a_j \text{ for } i \neq j, a_i \in \mathbb{F}_{p^n},$$

the *Root Finding Problem* consists of computing $a_1, \ldots, a_d$.

- Deterministic and randomised algorithms.
- Reduction of factoring in to root finding .
- Reduction of root finding in $\mathbb{F}_{p^n}$ to root finding in $\mathbb{F}_p$ by Berklekamp (1970).
- Many different algorithms and strategies in literature: Direct root finding, factoring, Berlekamp, Cantor-Zassenhaus, Kedlaya and Umans, Graeffe Transforms,...

Root Finding in Finite Fields (I))

Formally...

Definition (Root Finding Problem over Finite Fields)

Given $f \in \mathbb{F}_{p^n}[x]$, where f is a univariate, seperable polynomial which splits over $\mathbb{F}_{p^n}$,

$$f(x) = \prod_{i=1}^{d}(x - a_i) \qquad a_i \neq a_j \text{ for } i \neq j, a_i \in \mathbb{F}_{p^n},$$

the *Root Finding Problem* consists of computing $a_1, \ldots, a_d$.

- Deterministic and randomised algorithms.
- Reduction of factoring in to root finding .
- Reduction of root finding in $\mathbb{F}_{p^n}$ to root finding in $\mathbb{F}_p$ by Berklekamp (1970).
- Many different algorithms and strategies in literature: Direct root finding, factoring, Berlekamp, Cantor-Zassenhaus, Kedlaya and Umans, Graeffe Transforms,...

Root Finding in Finite Fields (II)

Another perspective

SRA: encode the roots of $f(x) \in \mathbb{F}_{p^n}$ into tree or forest structure and then compute the tree to obtain the roots.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (III)

Tschirnhaus Transformations (1683)

- Very simple tools - essentially GCD, Resultants and a generic method to find roots of small polynomials.
- Probably a good thing as this talk comes directly after lunch...

Definition (Tschirnhaus Transformation)

The *Tschirnhaus Transformation* of a polynomial $f(x) \in \mathbb{R}[x]$ is used to transform a polynomial $f(x)$ into a polynomial $g(y) \in \mathbb{R}[y]$, where the roots of $g(y)$ are of the form $y = \frac{a(x)}{b(x)}$, for all x such that $f(x) = 0$.

The Tschirnhaus Transformation may be computed by

$$g(y) := \mathrm{Res}_x(f(x), y \cdot a(x) - b(x)).$$

- Computable by the determinant of the Sylvester matrix.
- Specialised methods available — subresultants, characteristic polynomials, sparse resultants, modular approaches, etc.

Root Finding in Finite Fields (IV)

Graeffe Transforms in $\mathbb{R}[X]$ (1826, 1834, 1837)

$$f_1(x_1) = (x_1 - a_1)\cdots(x_1 - a_d)$$

The *Graeffe* method[a] finds roots of polynomials in $\mathbb{R}[x]$ via the transformation

$$\begin{aligned} f_1(x_1^2) &= (-1)^d \cdot f_0(x_1) \cdot f_1(-x_1) = (x_1^2 - a_1^2)\cdots(x_1^2 - a_d^2) \\ f_2(x_2) &= (x_2 - a_1^2)\cdots(x_2 - a_d^2) \end{aligned}$$

where $x_2 := x_1^2$.

- Step one - may be thought of as transforming the roots.
- Step two - solve using Vieta's formulas or solve the roots of the transformation and convert them.
- Step one of the Graeffe method is equivalent to computing a Tschirnhaus transform, or equivalently the resultant $(-1)^d \cdot \mathrm{Res}_{x_1}(f(x_1), x_1^2 - x_2)$.

---

a

Root Finding in Finite Fields (IV)

Graeffe Transforms in $\mathbb{R}[X]$ (1826, 1834, 1837)

$$f_1(x_1) = (x_1 - a_1) \cdots (x_1 - a_d)$$

The *Graeffe* method[a] finds roots of polynomials in $\mathbb{R}[x]$ via the transformation

$$\begin{aligned} f_1(x_1^2) = (-1)^d \cdot f_0(x_1) \cdot f_1(-x_1) &= (x_1^2 - a_1^2) \cdots (x_1^2 - a_d^2) \\ f_2(x_2) &= (x_2 - a_1^2) \cdots (x_2 - a_d^2) \end{aligned}$$

where $x_2 := x_1^2$.

- Step one - may be thought of as transforming the roots.
- Step two - solve using Vieta's formulas or solve the roots of the transformation and convert them.
- Step one of the Graeffe method is equivalent to computing a Tschirnhaus transform, or equivalently the resultant $(-1)^d \cdot \mathrm{Res}_{x_1}(f(x_1), x_1^2 - x_2)$.

[a] C. Graeffe. Die auflösung der höherer numerischen gleichungen, 1837

Root Finding in Finite Fields (IV)

Graeffe Transforms in $\mathbb{R}[X]$ (1826, 1834, 1837)

$$f_1(x_1) = (x_1 - a_1)\cdots(x_1 - a_d)$$

The *Graeffe* method[a] finds roots of polynomials in $\mathbb{R}[x]$ via the transformation

$$\begin{aligned} f_1(x_1^2) = (-1)^d \cdot f_0(x_1) \cdot f_1(-x_1) &= (x_1^2 - a_1^2)\cdots(x_1^2 - a_d^2) \\ f_2(x_2) &= (x_2 - a_1^2)\cdots(x_2 - a_d^2) \end{aligned}$$

where $x_2 := x_1^2$.

- Step one - may be thought of as transforming the roots.
- Step two - solve using Vieta's formulas or solve the roots of the transformation and convert them.
- Step one of the Graeffe method is equivalent to computing a Tschirnhaus transform, or equivalently the resultant $(-1)^d \cdot \mathrm{Res}_{x_1}(f(x_1), x_1^2 - x_2)$.

---

[a] C. Graeffe. Die auflösung der höherer numerischen gleichungen, 1837

Root Finding in Finite Fields (IV)

Graeffe Transforms in $\mathbb{R}[X]$ (1826, 1834, 1837)

$$f_1(x_1) = (x_1 - a_1) \cdots (x_1 - a_d)$$

The *Graeffe* method[a] finds roots of polynomials in $\mathbb{R}[x]$ via the transformation

$$\begin{aligned} f_1(x_1^2) = (-1)^d \cdot f_0(x_1) \cdot f_1(-x_1) &= (x_1^2 - a_1^2) \cdots (x_1^2 - a_d^2) \\ f_2(x_2) &= (x_2 - a_1^2) \cdots (x_2 - a_d^2) \end{aligned}$$

where $x_2 := x_1^2$.

- Step one - may be thought of as transforming the roots.
- Step two - solve using Vieta's formulas or solve the roots of the transformation and convert them.
- Step one of the Graeffe method is equivalent to computing a Tschirnhaus transform, or equivalently the resultant $(-1)^d \cdot \mathrm{Res}_{x_1}(f(x_1), x_1^2 - x_2)$.

[a] C. Graeffe. Die auflösung der höherer numerischen gleichungen, 1837

Root Finding in Finite Fields (IV)

Graeffe Transforms in $\mathbb{R}[X]$ (1826, 1834, 1837)

$$f_1(x_1) = (x_1 - a_1) \cdots (x_1 - a_d)$$

The *Graeffe* method[a] finds roots of polynomials in $\mathbb{R}[x]$ via the transformation

$$\begin{aligned} f_1(x_1^2) = (-1)^d \cdot f_0(x_1) \cdot f_1(-x_1) &= (x_1^2 - a_1^2) \cdots (x_1^2 - a_d^2) \\ f_2(x_2) &= (x_2 - a_1^2) \cdots (x_2 - a_d^2) \end{aligned}$$

where $x_2 := x_1^2$.

- Step one - may be thought of as transforming the roots.
- Step two - solve using Vieta's formulas or solve the roots of the transformation and convert them.
- Step one of the Graeffe method is equivalent to computing a Tschirnhaus transform, or equivalently the resultant $(-1)^d \cdot \mathrm{Res}_{x_1}(f(x_1), x_1^2 - x_2)$.

---

[a] C. Graeffe. Die auflösung der höherer numerischen gleichungen, 1837

Root Finding in Finite Fields (VI)

Graeffe Transforms in $\mathbb{F}_{p^n}[X]$ (2014, 2015)

$$f(x_1) = (x - a_1) \cdots (x - a_d)$$

Grenet et al.[b] define the *Generalised Graeffe Transform of order n*, for $n \in \mathbb{N}$ by

$$G_n(f)(x) := (-1)^{d \cdot n} \cdot \mathrm{Res}_z(f(z), z^n - x).$$

Our Resultant stage is essentially a Tschirnhaus transform.

$$G_{a(x)/b(x)}(f)(x) := (-1)^{d \cdot n} \cdot \mathrm{Res}_z(f(z), a(z) - b(z) \cdot x)$$

with $a, b \in \mathbb{F}_{p^n}[x]$ and $\gcd(a(x), b(x)) = 1$.

[b] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized Root Finding over Finite FFT-fields Using Tangent Graeffe Transforms, 2015

Root Finding in Finite Fields (VI)

Graeffe Transforms in $\mathbb{F}_{p^n}[X]$ (2014, 2015)

$$f(x_1) = (x - a_1) \cdots (x - a_d)$$

Grenet et al.[b] define the *Generalised Graeffe Transform of order n*, for $n \in \mathbb{N}$ by

$$G_n(f)(x) := (-1)^{d \cdot n} \cdot \mathrm{Res}_z(f(z), z^n - x).$$

Our Resultant stage is essentially a Tschirnhaus transform.

$$G_{a(x)/b(x)}(f)(x) := (-1)^{d \cdot n} \cdot \mathrm{Res}_z(f(z), a(z) - b(z) \cdot x)$$

with $a, b \in \mathbb{F}_{p^n}[x]$ and $\gcd(a(x), b(x)) = 1$.

[b] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized Root Finding over Finite FFT-fields Using Tangent Graeffe Transforms, 2015

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

---

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

---

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

---

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

The Successive Resultants Algorithm

- First introduced by Christophe Petit in 2014[c].
- A root finding algorithm for $\mathbb{F}_{p^n}$ which exploits the linear properties of certain polynomials in $\mathbb{F}_{p^n}$.
- Two distinct stages - the Resultant stage and the GCD stage.
- Theoretical complexity is on par with Berlekamp Trace Algorithm when fast arithmetic is used and p is fixed.
- Main idea: use the tree structure to compute the roots of many small degree polynomials instead of one large one.
- Efficiency dependent on fast FFT-style arithmetic and p small.

[c] C. Petit. Finding roots in $GF(p^n)$ with the successive resultant algorithm. *LMS Journal of Computation and Mathematics (special issue for ANTS XI)*, 17A:203–217, 2014

From Maps to Sequences (I)

SRA Ingredients

- $f \in \mathbb{F}_{p^n}[x]$ - separable polynomial of degree d, split over $\mathbb{F}_{p^n}$.
- $K_1(x_1), \ldots, K_t(x_t)$ - a precomputed set of rational maps.
- $\text{Image}(K_t \circ \cdots K_1) \subset \mathbb{F}_{p^n}$ is known and small.

We consider the system:

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \qquad \text{for } i = 1, \ldots, t \end{cases}$$

where $\gcd(a_i, b_i)$ is trivial and K_i non-constant for $i = 1, \ldots, t$.

From Maps to Sequences (I)

SRA Ingredients

- $f \in \mathbb{F}_{p^n}[x]$ - separable polynomial of degree d, split over $\mathbb{F}_{p^n}$.
- $K_1(x_1), \ldots, K_t(x_t)$ - a precomputed set of rational maps.
- Image$(K_t \circ \cdots K_1) \subset \mathbb{F}_{p^n}$ is known and small.

We consider the system:

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \qquad \text{for } i = 1, \ldots, t \end{cases}$$

where $\gcd(a_i, b_i)$ is trivial and K_i non-constant for $i = 1, \ldots, t$.

From Maps to Sequences (II)

Sequences

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \qquad \text{for } i = 1, \ldots, t. \end{cases}$$

Main idea:

- Successive application of the ordered $K_i(x_i)$ maps to the roots of f gives us d unique sequences of length up to $t+1$.

$$(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_j)$$

where

$$\begin{aligned} \bar{x}_1 &= \bar{x}_1 && \text{is a root of } f. \\ \bar{x}_2 &= K_1(\bar{x}_1) \\ \bar{x}_3 &= K_2(\bar{x}_2) && = K_2 \circ K_1(\bar{x}_1) \\ &\vdots \\ \bar{x}_j &= K_{j-1}(\bar{x}_{j-1}) = K_{j-1} \circ \cdots \circ K_2 \circ K_1(\bar{x}_1) \end{aligned}$$

From Maps to Sequences (III)

Facts about sequences

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

- If the K_i maps are chosen carefully, we may limit the number of potential values for the final points in the sequences.
- Leads to finite sequences of length $\leq t + 1$.
- If a sequences $(\bar{x}_1, \ldots, \bar{x}_i)$ exists and $b_i(\bar{x}_i) = 0$, the sequence is of length i.
- If Image$(K_{i-1} \circ \cdots \circ K_1)$ is known, we know all potential values $\bar{x}_i$ may take.

From Maps to Sequences (III)

Facts about sequences

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

- If the K_i maps are chosen carefully, we may limit the number of potential values for the final points in the sequences.
- Leads to finite sequences of length $\leq t + 1$.
- If a sequences $(\bar{x}_1, \ldots, \bar{x}_i)$ exists and $b_i(\bar{x}_i) = 0$, the sequence is of length i.
- If Image$(K_{i-1} \circ \cdots \circ K_1)$ is known, we know all potential values $\bar{x}_i$ may take.

From Maps to Sequences (III)

Facts about sequences

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \dots, t.$$

- If the K_i maps are chosen carefully, we may limit the number of potential values for the final points in the sequences.
- Leads to finite sequences of length $\leq t + 1$.
- If a sequences $(\bar{x}_1, \dots, \bar{x}_i)$ exists and $b_i(\bar{x}_i) = 0$, the sequence is of length i.
- If Image$(K_{i-1} \circ \cdots \circ K_1)$ is known, we know all potential values $\bar{x}_i$ may take.

From Maps to Sequences (III)

Facts about sequences

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

- If the K_i maps are chosen carefully, we may limit the number of potential values for the final points in the sequences.
- Leads to finite sequences of length $\leq t + 1$.
- If a sequences $(\bar{x}_1, \ldots, \bar{x}_i)$ exists and $b_i(\bar{x}_i) = 0$, the sequence is of length i.
- If $\mathrm{Image}(K_{i-1} \circ \cdots \circ K_1)$ is known, we know all potential values $\bar{x}_i$ may take.

From Maps to Sequences (IV)

Encoding sequences in polynomials

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

Assume $\bar{x}_i \in \mathbb{F}_{p^n}$.

If $b_i(\bar{x}_i) \neq 0$, $\exists\ \bar{x}_{i+1} \in \mathbb{F}_{p^n}$ st.

$$K_i(\bar{x}_i) = \frac{a_i(\bar{x}_i)}{b_i(\bar{x}_i)} = \bar{x}_{i+1} \iff a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} = 0$$

If $b_i(\bar{x}_i) = 0$,

$$a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} \neq 0$$

for any $\bar{x}_{i+1} \in \mathbb{F}_{p^n}$.

From Maps to Sequences (IV)

Encoding sequences in polynomials

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

Assume $\bar{x}_i \in \mathbb{F}_{p^n}$.

If $b_i(\bar{x}_i) \neq 0$, $\exists\ \bar{x}_{i+1} \in \mathbb{F}_{p^n}$ st.

$$K_i(\bar{x}_i) = \frac{a_i(\bar{x}_i)}{b_i(\bar{x}_i)} = \bar{x}_{i+1} \iff a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} = 0$$

If $b_i(\bar{x}_i) = 0$,

$$a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} \neq 0$$

for any $\bar{x}_{i+1} \in \mathbb{F}_{p^n}$.

From Maps to Sequences (IV)

Encoding sequences in polynomials

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \dots, t.$$

Assume $\bar{x}_i \in \mathbb{F}_{p^n}$.

If $b_i(\bar{x}_i) \neq 0$, $\exists\ \bar{x}_{i+1} \in \mathbb{F}_{p^n}$ st.

$$K_i(\bar{x}_i) = \frac{a_i(\bar{x}_i)}{b_i(\bar{x}_i)} = \bar{x}_{i+1} \iff a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} = 0$$

If $b_i(\bar{x}_i) = 0$,

$$a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} \neq 0$$

for any $\bar{x}_{i+1} \in \mathbb{F}_{p^n}$.

From Maps to Sequences (IV)

Encoding sequences in polynomials

$$\begin{cases} f(x_1) & = 0 \\ K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} & = x_{i+1} \end{cases} \qquad \text{for } i = 1, \ldots, t.$$

Assume $\bar{x}_i \in \mathbb{F}_{p^n}$.

If $b_i(\bar{x}_i) \neq 0$, $\exists\ \bar{x}_{i+1} \in \mathbb{F}_{p^n}$ st.

$$K_i(\bar{x}_i) = \frac{a_i(\bar{x}_i)}{b_i(\bar{x}_i)} = \bar{x}_{i+1} \iff a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} = 0$$

If $b_i(\bar{x}_i) = 0$,

$$a_i(\bar{x}_i) - b_i(\bar{x}_i) \cdot \bar{x}_{i+1} \neq 0$$

for any $\bar{x}_{i+1} \in \mathbb{F}_{p^n}$.

From Maps to Sequences (VI)

SRA - The Resultant stage

The SRA Resultant stage:

$$\begin{cases} f^{(1)}(x_1) & := f(x_1) \\ f^{(i+1)}(x_{i+1}) & := \mathrm{Res}_{x_i}(f^{(i)}(x_i),\ a_i(x_i) - b_i(x_i) \cdot x_{i+1}) \end{cases}$$

Output: univariate polynomials $f^{(1)}(x_1), \ldots, f^{(t)}(x_t)$.

Lemma

$f^{(i)}(\bar{x}_i) = 0 \iff$ *there exists a sequence* $(\bar{x}_1, \ldots, \bar{x}_i, \ldots)$.

- $\deg(f^{(i+1)}) < \deg(f^{(i)})$ if there exists a sequence of length i.
- $\deg(f^{(i+1)}) = \deg(f^{(i)})$ otherwise.

From Maps to Sequences (VII)

Working backwards from a known point in a sequence

The SRA Resultant stage:

$$\begin{cases} f^{(1)}(x_1) & := f(x_1) \\ f^{(i+1)}(x_{i+1}) & := \mathrm{Res}_{x_i}(f^{(i)}(x_i),\ a_i(x_i) - b_i(x_i) \cdot x_{i+1}) \end{cases}$$

Theorem

Given any $\bar{x}_{i+1} \in \mathbb{F}_{p^n}$ such that $f^{(i+1)}(\bar{x}_{i+1}) = 0$, we may extract all $\bar{x}_i \in \mathbb{F}_{p^n}$ such that there exists a sequence $(\bar{x}_1, \ldots, \bar{x}_i, \bar{x}_{i+1}, \ldots)$.

This procedure may be performed by computing the roots of polynomials of degree $\leq \max\{\deg(a_i), \deg(b_i)\}$.

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1})$$

$$g_{b_i}(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i))$$

From Maps to Sequences (VIII)

Locating all final points in sequences

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1}) \quad (1)$$

$$g_i(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i)) \quad (2)$$

- $\bar{x}_{t+1}$ - we may locate from Image$(K_t \circ \cdots K_1)$
- $\bar{x}_1, \ldots, \bar{x}_{t-1}$ - we compute (2) if $\deg(f^{(i+1)}) < \deg(f^{(i)})$.
- $\bar{x}_t$ - we always compute (2) as there exists no $f^{(t+1)}$.

Together, the Resultant stage and the GCD stage give us a way to encode and iteratively extract these sequences from the $K_1, \ldots, K_t$ rational maps, leading to the roots of $f^{(1)} = f$.

From Maps to Sequences (VIII)

Locating all final points in sequences

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1}) \tag{1}$$

$$g_i(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i)) \tag{2}$$

- $\bar{x}_{t+1}$ - we may locate from $\text{Image}(K_t \circ \cdots K_1)$
- $\bar{x}_1, \ldots, \bar{x}_{t-1}$ - we compute (2) if $\deg(f^{(i+1)}) < \deg(f^{(i)})$.
- $\bar{x}_t$ - we always compute (2) as there exists no $f^{(t+1)}$.

Together, the Resultant stage and the GCD stage give us a way to encode and iteratively extract these sequences from the $K_1, \ldots, K_t$ rational maps, leading to the roots of $f^{(1)} = f$.

From Maps to Sequences (VIII)

Locating all final points in sequences

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1}) \tag{1}$$

$$g_i(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i)) \tag{2}$$

- $\bar{x}_{t+1}$ - we may locate from Image$(K_t \circ \cdots K_1)$
- $\bar{x}_1, \ldots, \bar{x}_{t-1}$ - we compute (2) if $\deg(f^{(i+1)}) < \deg(f^{(i)})$.
- $\bar{x}_t$ - we always compute (2) as there exists no $f^{(t+1)}$.

Together, the Resultant stage and the GCD stage give us a way to encode and iteratively extract these sequences from the $K_1, \ldots, K_t$ rational maps, leading to the roots of $f^{(1)} = f$.

From Maps to Sequences (VIII)

Locating all final points in sequences

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1}) \quad (1)$$

$$g_i(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i)) \quad (2)$$

- $\bar{x}_{t+1}$ - we may locate from Image$(K_t \circ \cdots K_1)$
- $\bar{x}_1, \ldots, \bar{x}_{t-1}$ - we compute (2) if $\deg(f^{(i+1)}) < \deg(f^{(i)})$.
- $\bar{x}_t$ - we always compute (2) as there exists no $f^{(t+1)}$.

Together, the Resultant stage and the GCD stage give us a way to encode and iteratively extract these sequences from the $K_1, \ldots, K_t$ rational maps, leading to the roots of $f^{(1)} = f$.

From Maps to Sequences (VIII)

Locating all final points in sequences

The SRA GCD stage:

$$g_{\bar{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \bar{x}_{i+1}) \tag{1}$$

$$g_i(x_i) := \gcd(f^{(i)}(x_i), b_i(x_i)) \tag{2}$$

- $\bar{x}_{t+1}$ - we may locate from Image$(K_t \circ \cdots K_1)$
- $\bar{x}_1, \ldots, \bar{x}_{t-1}$ - we compute (2) if $\deg(f^{(i+1)}) < \deg(f^{(i)})$.
- $\bar{x}_t$ - we always compute (2) as there exists no $f^{(t+1)}$.

Together, the Resultant stage and the GCD stage give us a way to encode and iteratively extract these sequences from the $K_1, \ldots, K_t$ rational maps, leading to the roots of $f^{(1)} = f$.

Concerning Generic Complexity

Generic maps

Assumes precomputation of maps is performed:

- $f \in \mathbb{F}_{p^n}[x]$ of degree d, separable and split over $\mathbb{F}_{p^n}$.
- t rational maps $K_1, \ldots, K_t$ of maximum degree B.
- $L = \mathrm{Image}(K_t \circ \cdots \circ K_1) \subset \mathbb{F}_{p^n}$.
- $d \geq \max\{L, B\}$.
- Let $\mathrm{P}(d)$ denote a generic method of solving a degree d polynomial (any algorithm, or directly by formula if $d \leq 4$).

	Resultant Stage	GCD stage	Total[d]
Classical	$O(td^3n^2 \log d)$	$O(td^3n^2 \log d + td\mathrm{P}(B))$	$O(td^3n^2 \log d)$
Fast	$\tilde{O}(td^2n)$	$\tilde{O}(td^2n + td\mathrm{P}(B))$	$\tilde{O}(td^2n)$

Table : Costs for Generalised SRA in terms of operations over $\mathbb{F}_p$.

[d] assuming $\mathrm{P}(d)$ is performed by BTA and $B, n < d$.

Concerning Generic Complexity

Generic maps

Assumes precomputation of maps is performed:

- $f \in \mathbb{F}_{p^n}[x]$ of degree d, separable and split over $\mathbb{F}_{p^n}$.
- t rational maps $K_1, \ldots, K_t$ of maximum degree B.
- $L = \text{Image}(K_t \circ \cdots \circ K_1) \subset \mathbb{F}_{p^n}$.
- $d \geq \max\{L, B\}$.
- Let $\text{P}(d)$ denote a generic method of solving a degree d polynomial (any algorithm, or directly by formula if $d \leq 4$).

	Resultant Stage	GCD stage	Total[d]
Classical	$O(td^3n^2 \log d)$	$O(td^3n^2 \log d + td\text{P}(B))$	$O(td^3n^2 \log d)$
Fast	$\tilde{O}(td^2n)$	$\tilde{O}(td^2n + td\text{P}(B))$	$\tilde{O}(td^2n)$

Table : Costs for Generalised SRA in terms of operations over $\mathbb{F}_p$.

[d] assuming $\text{P}(d)$ is performed by BTA and $B, n < d$.

The Successive Resultants Algorithm (I)

Precomputing maps for $\mathbb{F}_{p^n}$

- Works with $f \in \mathbb{F}_{p^n}[x]$.
- Define an arbitrary basis for $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$: $\{\alpha_1, \ldots, \alpha_n\}$.

The Successive Resultants Algorithm (I)

Precomputing maps for $\mathbb{F}_{p^n}$

- Works with $f \in \mathbb{F}_{p^n}[x]$.
- Define an arbitrary basis for $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$: $\{\alpha_1, \ldots, \alpha_n\}$.

$$
(1) \begin{cases} L_0(z) = z \\ L_1(z) = \prod_{i \in \mathbb{F}_p} L_0(z - i \cdot \alpha_1) \\ \vdots \\ L_n(z) = \prod_{i \in \mathbb{F}_p} L_{n-1}(z - i \cdot \alpha_n) \end{cases}
$$

The Successive Resultants Algorithm (I)

Precomputing maps for $\mathbb{F}_{p^n}$

- Works with $f \in \mathbb{F}_{p^n}[x]$.
- Define an arbitrary basis for $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$: $\{\alpha_1, \ldots, \alpha_n\}$.

$$(1) \begin{cases} L_0(z) = z \\ L_1(z) = \prod_{i \in \mathbb{F}_p} L_0(z - i \cdot \alpha_1) \\ \vdots \\ L_n(z) = \prod_{i \in \mathbb{F}_p} L_{n-1}(z - i \cdot \alpha_n) \end{cases}$$

Lemma

(i) *Given system* (1)*, there exist constants* $a_1, \ldots, a_n \in \mathbb{F}_{p^n}$ *such that*

$$L_i(z) = L_{i-1}(z)^p - a_i \cdot L_{i-1}(z).$$

(ii) $L_n(z) = z^{p^n} - z$.

The Successive Resultants Algorithm (I)

Precomputing maps for $\mathbb{F}_{p^n}$

- Works with $f \in \mathbb{F}_{p^n}[x]$.
- Define an arbitrary basis for $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$: $\{\alpha_1, \ldots, \alpha_n\}$.

$$(1)\begin{cases} L_0(z) = z \\ L_1(z) = \prod_{i \in \mathbb{F}_p} L_0(z - i \cdot \alpha_1) \\ \vdots \\ L_n(z) = \prod_{i \in \mathbb{F}_p} L_{n-1}(z - i \cdot \alpha_n) \end{cases} \Longrightarrow \begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

Lemma

(i) *Given system* (1), *there exist constants* $a_1, \ldots, a_n \in \mathbb{F}_{p^n}$ *such that*

$$L_i(z) = L_{i-1}(z)^p - a_i \cdot L_{i-1}(z).$$

(ii) $L_n(z) = z^{p^n} - z$.

The Successive Resultants Algorithm (II)

Complexity

$$\begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

- Each $K_i(x_i)$ is of degree p.
- $\text{Image}(K_n \circ \cdots \circ K_1) = \{0\}$.

	Precomputation	Naive SRA Algorithm	Optimised SRA
Classical	$O(n^4)$	$O(d^2 n^3 \log d)$	—
Fast	$\tilde{O}(n^3)$	$\tilde{O}(d^2 n^2)$	$\tilde{O}(dn^2)$

Table : Costs for original SRA in terms of basic operations over $\mathbb{F}_p$.

- Optimisation of the resultant stage is based on exploiting Frobenius transformation and the prime degree of the maps.
- Optimisation of GCD stage comes from multipoint evaluation and reuse of computations performed in the resultant stage.

The Successive Resultants Algorithm (II)

Complexity

$$\begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

- Each $K_i(x_i)$ is of degree p.
- $\mathrm{Image}(K_n \circ \cdots \circ K_1) = \{0\}$.

	Precomputation	Naive SRA Algorithm	Optimised SRA
Classical	$O(n^4)$	$O(d^2 n^3 \log d)$	—
Fast	$\tilde{O}(n^3)$	$\tilde{O}(d^2 n^2)$	$\tilde{O}(dn^2)$

Table : Costs for original SRA in terms of basic operations over $\mathbb{F}_p$.

- Optimisation of the resultant stage is based on exploiting Frobenius transformation and the prime degree of the maps.
- Optimisation of GCD stage comes from multipoint evaluation and reuse of computations performed in the resultant stage.

The Successive Resultants Algorithm (II)

Complexity

$$\begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

- Each $K_i(x_i)$ is of degree p.
- $\mathrm{Image}(K_n \circ \cdots \circ K_1) = \{0\}$.

	Precomputation	Naive SRA Algorithm	Optimised SRA
Classical	$O(n^4)$	$O(d^2 n^3 \log d)$	—
Fast	$\tilde{O}(n^3)$	$\tilde{O}(d^2 n^2)$	$\tilde{O}(dn^2)$

Table : Costs for original SRA in terms of basic operations over $\mathbb{F}_p$.

- Optimisation of the resultant stage is based on exploiting Frobenius transformation and the prime degree of the maps.
- Optimisation of GCD stage comes from multipoint evaluation and reuse of computations performed in the resultant stage.

The Successive Resultants Algorithm (II)

Complexity

$$\begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

- Each $K_i(x_i)$ is of degree p.
- $\mathrm{Image}(K_n \circ \cdots \circ K_1) = \{0\}$.

	Precomputation	Naive SRA Algorithm	Optimised SRA
Classical	$O(n^4)$	$O(d^2 n^3 \log d)$	—
Fast	$\tilde{O}(n^3)$	$\tilde{O}(d^2 n^2)$	$\tilde{O}(dn^2)$

Table : Costs for original SRA in terms of basic operations over $\mathbb{F}_p$.

- Optimisation of the resultant stage is based on exploiting Frobenius transformation and the prime degree of the maps.
- Optimisation of GCD stage comes from multipoint evaluation and reuse of computations performed in the resultant stage.

The Successive Resultants Algorithm (II)

Complexity

$$\begin{cases} K_1(x_1) = x_1^p - a_1 \cdot x_1 = x_2 \\ \vdots \\ K_n(x_n) = x_n^p - a_n \cdot x_n = x_{n+1} \end{cases}$$

- Each $K_i(x_i)$ is of degree p.
- $\text{Image}(K_n \circ \cdots \circ K_1) = \{0\}$.

	Precomputation	Naive SRA Algorithm	Optimised SRA
Classical	$O(n^4)$	$O(d^2 n^3 \log d)$	—
Fast	$\tilde{O}(n^3)$	$\tilde{O}(d^2 n^2)$	$\tilde{O}(dn^2)$

Table : Costs for original SRA in terms of basic operations over $\mathbb{F}_p$.

- Optimisation of the resultant stage is based on exploiting Frobenius transformation and the prime degree of the maps.
- Optimisation of GCD stage comes from multipoint evaluation and reuse of computations performed in the resultant stage.

Maps when $|\mathbb{F}_p^*|$ is smooth (I)

Precomputing maps

- Works for $f \in \mathbb{F}_p[x]$.

Definition (Smooth number)

$n \in \mathbb{N}$ is B-smooth if $n = n_1 \cdots n_t$, where $1 \leq n_i \leq B$.

Theorem (Fermat's Little Theorem)

For any $x \in \mathbb{F}_p^*$, we have that

$$x^{p-1} = 1.$$

If $p - 1 = n_1 \cdots n_t$, we can construct the series of maps

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases}$$

with $\mathrm{Image}(K_t \circ \cdots \circ K_1) = \{0, 1\}$.

Maps when $|\mathbb{F}_p^*|$ is smooth (I)

Precomputing maps

- Works for $f \in \mathbb{F}_p[x]$.

Definition (Smooth number)

$n \in \mathbb{N}$ is B-smooth if $n = n_1 \cdots n_t$, where $1 \leq n_i \leq B$.

Theorem (Fermat's Little Theorem)

For any $x \in \mathbb{F}_p^*$, we have that

$$x^{p-1} = 1.$$

If $p - 1 = n_1 \cdots n_t$, we can construct the series of maps

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases}$$

with $\mathrm{Image}(K_t \circ \cdots \circ K_1) = \{0, 1\}$.

Maps when $|\mathbb{F}_p^*|$ is smooth (I)

Precomputing maps

- Works for $f \in \mathbb{F}_p[x]$.

Definition (Smooth number)

$n \in \mathbb{N}$ is B-smooth if $n = n_1 \cdots n_t$, where $1 \leq n_i \leq B$.

Theorem (Fermat's Little Theorem)

For any $x \in \mathbb{F}_p^*$, we have that

$$x^{p-1} = 1.$$

If $p - 1 = n_1 \cdots n_t$, we can construct the series of maps

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases}$$

with $\mathrm{Image}(K_t \circ \cdots \circ K_1) = \{0, 1\}$.

Maps when $|\mathbb{F}_p^*|$ is smooth (II)

Complexity

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases} \qquad \text{for } p-1 = n_1 \cdots n_t.$$

- Correctness holds for any finite field, but efficient when B small.
- Complexity of $O(td^2n^2 \log d + tdB^2n^3)$ for classical arithmetic.
- Complexity of $\tilde{O}(td^2n + tdBn^2)$ for fast arithmetic.
- Grenet et al.[e] algorithm posesses optimisations complexity of $\tilde{O}(B^{\frac{1}{2}}dn^2\log^2 p) + (dn^2\log^2 p)$ for $\mathbb{F}_{p^n}$.

---

e

Maps when $|\mathbb{F}_p^*|$ is smooth (II)

Complexity

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases} \qquad \text{for } p-1 = n_1 \cdots n_t.$$

- Correctness holds for any finite field, but efficient when B small.
- Complexity of $O(td^2n^2 \log d + tdB^2n^3)$ for classical arithmetic.
- Complexity of $\tilde{O}(td^2n + tdBn^2)$ for fast arithmetic.
- Grenet et al.[e] algorithm posesses optimisations complexity of $\tilde{O}(B^{\frac{1}{2}}dn^2 \log^2 p) + (dn^2 \log^2 p)$ for $\mathbb{F}_{p^n}$.

e

Maps when $|\mathbb{F}_p^*|$ is smooth (II)

Complexity

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases} \qquad \text{for } p-1 = n_1 \cdots n_t.$$

- Correctness holds for any finite field, but efficient when B small.
- Complexity of $O(td^2n^2 \log d + tdB^2n^3)$ for classical arithmetic.
- Complexity of $\tilde{O}(td^2n + tdBn^2)$ for fast arithmetic.
- Grenet et al.[e] algorithm posesses optimisations complexity of $\tilde{O}(B^{\frac{1}{2}}dn^2 \log^2 p) + (dn^2 \log^2 p)$ for $\mathbb{F}_{p^n}$.

---

e

Maps when $|\mathbb{F}_p^*|$ is smooth (II)

Complexity

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases} \qquad \text{for } p-1 = n_1 \cdots n_t.$$

- Correctness holds for any finite field, but efficient when B small.
- Complexity of $O(td^2n^2 \log d + tdB^2n^3)$ for classical arithmetic.
- Complexity of $\tilde{O}(td^2n + tdBn^2)$ for fast arithmetic.
- Grenet et al.[e] algorithm posesses optimisations complexity of $\tilde{O}(B^{\frac{1}{2}}dn^2 \log^2 p) + (dn^2 \log^2 p)$ for $\mathbb{F}_{p^n}$.

e

Maps when $|\mathbb{F}_p^*|$ is smooth (II)

Complexity

$$\begin{cases} K_1(x_1) & = x_1^{n_1} = x_2 \\ \vdots & \\ K_t(x_t) & = x_t^{n_t} = x_{t+1} \end{cases} \qquad \text{for } p-1 = n_1 \cdots n_t.$$

- Correctness holds for any finite field, but efficient when B small.
- Complexity of $O(td^2n^2 \log d + tdB^2n^3)$ for classical arithmetic.
- Complexity of $\tilde{O}(td^2n + tdBn^2)$ for fast arithmetic.
- Grenet et al.[e] algorithm posesses optimisations complexity of $\tilde{O}(B^{\frac{1}{2}}dn^2 \log^2 p) + (dn^2 \log^2 p)$ for $\mathbb{F}_{p^n}$.

---

[e] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized Root Finding over Finite FFT-fields Using Tangent Graeffe Transforms, 2015

Maps when $p = 2 \mod 3$ (I)

Definition (Isogeny)

We use the notation $E_{w,v}$ to signify the rational points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ which satisfy the Elliptic curve

$$y^2 = x^3 + wx + v \qquad \text{where } w, v \in \mathbb{F}_p.$$

An *Isogeny* between two elliptic curves $\phi : E_{w,v} \longrightarrow E_{w',v'}$ is a non-constant morphism which preserves the point at infinity.

Lemma

(i) Any *isogeny* $\phi_i : E_{w_i,v_i} \longrightarrow E_{w_{i+1},v_{i+1}}$ has a rational representation

$$\bar{\phi}_i(x_i, y_i) \mapsto \left(\frac{\xi_i(x_i)}{\psi_i^2(x_i)}, y_i \cdot \frac{\omega_i(x_i)}{\psi_i^2(x_i)}\right)$$

(ii) If $\phi_i(P_i) = \mathcal{O}$ with $P_i = (x_i : y_i : 1)$, then we have that $\psi_i(x_i) = 0$.

Maps when $p = 2 \mod 3$ (I)

Definition (Isogeny)

We use the notation $E_{w,v}$ to signify the rational points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ which satisfy the Elliptic curve

$$y^2 = x^3 + wx + v \qquad \text{where } w, v \in \mathbb{F}_p.$$

An *Isogeny* between two elliptic curves $\phi : E_{w,v} \longrightarrow E_{w',v'}$ is a non-constant morphism which preserves the point at infinity.

Lemma

(i) Any *isogeny* $\phi_i : E_{w_i,v_i} \longrightarrow E_{w_{i+1},v_{i+1}}$ has a rational representation

$$\bar{\phi}_i(x_i, y_i) \mapsto \left(\frac{\xi_i(x_i)}{\psi_i^2(x_i)}, y_i \cdot \frac{\omega_i(x_i)}{\psi_i^2(x_i)}\right)$$

(ii) If $\phi_i(P_i) = \mathcal{O}$ with $P_i = (x_i : y_i : 1)$, then we have that $\psi_i(x_i) = 0$.

Maps when $p = 2 \mod 3$ (I)

Definition (Isogeny)

We use the notation $E_{w,v}$ to signify the rational points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ which satisfy the Elliptic curve

$$y^2 = x^3 + wx + v \qquad\qquad \text{where } w, v \in \mathbb{F}_p.$$

An *Isogeny* between two elliptic curves $\phi : E_{w,v} \longrightarrow E_{w',v'}$ is a non-constant morphism which preserves the point at infinity.

Lemma

(i) Any *isogeny* $\phi_i : E_{w_i,v_i} \longrightarrow E_{w_{i+1},v_{i+1}}$ has a rational representation

$$\bar{\phi}_i(x_i, y_i) \mapsto \left(\frac{\xi_i(x_i)}{\psi_i^2(x_i)}, y_i \cdot \frac{\omega_i(x_i)}{\psi_i^2(x_i)}\right)$$

(ii) If $\phi_i(P_i) = \mathcal{O}$ with $P_i = (x_i : y_i : 1)$, then we have that $\psi_i(x_i) = 0$.

Maps when $p = 2 \mod 3$ (II)

Overview

- Works for $f \in \mathbb{F}_p[x]$ when $p = 2 \mod 3$.
- Uses SRA as a special subroutine.

Main idea:

1. Use a bijection $\phi_0 : \mathbb{F}_p^* \longrightarrow E_{w_0,v_0}$ to encode roots of f as x-coordinates of the mostly smooth order curve E_{w_0,v_0}.
2. Use rational maps derived from the x-coordinates of isogenies between mostly smooth curves such that $\text{Image}(\phi_t \circ \cdots \circ \phi_1) = \mathcal{O}$.
3. Convert the roots found in E_{w_0,v_0} back to $\mathbb{F}_p^*$.

Maps when $p = 2 \mod 3$ (II)

Overview

- Works for $f \in \mathbb{F}_p[x]$ when $p = 2 \mod 3$.
- Uses SRA as a special subroutine.

Main idea:

1. Use a bijection $\phi_0 : \mathbb{F}_p^* \longrightarrow E_{w_0,v_0}$ to encode roots of f as x-coordinates of the mostly smooth order curve E_{w_0,v_0}.
2. Use rational maps derived from the x-coordinates of isogenies between mostly smooth curves such that $\text{Image}(\phi_t \circ \cdots \circ \phi_1) = \mathcal{O}$.
3. Convert the roots found in E_{w_0,v_0} back to $\mathbb{F}_p^*$.

Maps when $p = 2 \mod 3$ (II)

Overview

- Works for $f \in \mathbb{F}_p[x]$ when $p = 2 \mod 3$.
- Uses SRA as a special subroutine.

Main idea:

1. Use a bijection $\phi_0 : \mathbb{F}_p^* \longrightarrow E_{w_0,v_0}$ to encode roots of f as x-coordinates of the mostly smooth order curve E_{w_0,v_0}.
2. Use rational maps derived from the x-coordinates of isogenies between mostly smooth curves such that $\text{Image}(\phi_t \circ \cdots \circ \phi_1) = \mathcal{O}$.
3. Convert the roots found in E_{w_0,v_0} back to $\mathbb{F}_p^*$.

Maps when $p = 2 \mod 3$ (II)

Overview

- Works for $f \in \mathbb{F}_p[x]$ when $p = 2 \mod 3$.
- Uses SRA as a special subroutine.

Main idea:

1. Use a bijection $\phi_0 : \mathbb{F}_p^* \longrightarrow E_{w_0,v_0}$ to encode roots of f as x-coordinates of the mostly smooth order curve E_{w_0,v_0}.
2. Use rational maps derived from the x-coordinates of isogenies between mostly smooth curves such that $\text{Image}(\phi_t \circ \cdots \circ \phi_1) = \mathcal{O}$.
3. Convert the roots found in E_{w_0,v_0} back to $\mathbb{F}_p^*$.

Maps when $p = 2 \mod 3$ (III)

Icart's Map

Given $\mathbb{F}_p^*$ where $p = 2 \mod 3$:

$$p = 2 \mod 3 \iff z \mapsto z^3 \mod p \qquad \text{is a bijection.}$$

Theorem (Hashing to Elliptic Curves (2009) - Icart)

(i) *Let $p = 2 \mod 3$ be an odd prime.*
The map $f_{w_0,v_0} : \mathbb{F}_p \longrightarrow E_{w_0,v_0}$ sending 0 to the point at infinity and $u \in \mathbb{F}_p^*$ to $(x, y) \in E_{w_0,v_0}$ where

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

is a well defined surjective map.
(ii) *If $P = (x, y)$ is a point on the curve E_{w_0,v_0}, then the solutions u_s of $f_{w_0,v_0}(u_s) = P$ are the solutions of the polynomial equation*

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Maps when $p = 2 \mod 3$ (III)

Icart's Map

Given $\mathbb{F}_p^*$ where $p = 2 \mod 3$:

$$p = 2 \mod 3 \iff z \mapsto z^3 \mod p \qquad \text{is a bijection.}$$

Theorem (Hashing to Elliptic Curves (2009) - Icart)

(i) *Let $p = 2 \mod 3$ be an odd prime.*
The map $f_{w_0,v_0} : \mathbb{F}_p \longrightarrow E_{w_0,v_0}$ sending 0 to the point at infinity and $u \in \mathbb{F}_p^*$ to $(x, y) \in E_{w_0,v_0}$ where

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

is a well defined surjective map.
(ii) *If $P = (x, y)$ is a point on the curve E_{w_0,v_0}, then the solutions u_s of $f_{w_0,v_0}(u_s) = P$ are the solutions of the polynomial equation*

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Maps when $p = 2 \mod 3$ (III)

Icart's Map

Given $\mathbb{F}_p^*$ where $p = 2 \mod 3$:

$$p = 2 \mod 3 \iff z \mapsto z^3 \mod p \qquad \text{is a bijection.}$$

Theorem (Hashing to Elliptic Curves (2009) - Icart)

(i) *Let $p = 2 \mod 3$ be an odd prime.*
The map $f_{w_0,v_0} : \mathbb{F}_p \longrightarrow E_{w_0,v_0}$ sending 0 to the point at infinity and $u \in \mathbb{F}_p^*$ to $(x, y) \in E_{w_0,v_0}$ where

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

is a well defined surjective map.
(ii) *If $P = (x, y)$ is a point on the curve E_{w_0,v_0}, then the solutions u_s of $f_{w_0,v_0}(u_s) = P$ are the solutions of the polynomial equation*

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Maps when $p = 2 \mod 3$ (IV)

Using Icart's map

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

Part (i) of this lemma gives us the polynomial transformation method

$$f^{(1)}(x_1) := \mathrm{Res}_u(f(u), u^4 - 6u^2x - 3w_0)^2 - 36u^2(x^3 + w_0x + v_0)$$

resulting in a polynomial of degree $3 \cdot \deg(f)$. Our maps are then the x-coordinate maps corresponding to the isogenies

$$\begin{cases} K_1(x_1) = \frac{\xi_1(x_1)}{\psi^2(x_1)} = x_2 \\ \vdots \\ K_t(x_t) = \frac{\xi_t(x_t)}{\psi^2(x_t)} = x_{t+1} \end{cases}$$

$\phi_t \circ \cdots \circ \phi_1(P) = \mathcal{O} \implies$ all sequences are of length $< t + 1$.

Maps when $p = 2 \mod 3$ (IV)

Using Icart's map

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

Part (i) of this lemma gives us the polynomial transformation method

$$f^{(1)}(x_1) := \mathrm{Res}_u(f(u), u^4 - 6u^2x - 3w_0)^2 - 36u^2(x^3 + w_0x + v_0)$$

resulting in a polynomial of degree $3 \cdot \deg(f)$. Our maps are then the x-coordinate maps corresponding to the isogenies

$$\begin{cases} K_1(x_1) = \frac{\xi_1(x_1)}{\psi^2(x_1)} = x_2 \\ \vdots \\ K_t(x_t) = \frac{\xi_t(x_t)}{\psi^2(x_t)} = x_{t+1} \end{cases}$$

$\phi_t \circ \cdots \circ \phi_1(P) = \mathcal{O} \implies$ all sequences are of length $< t + 1$.

Maps when $p = 2 \mod 3$ (IV)

Using Icart's map

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

Part (i) of this lemma gives us the polynomial transformation method

$$f^{(1)}(x_1) := \mathrm{Res}_u(f(u), u^4 - 6u^2x - 3w_0)^2 - 36u^2(x^3 + w_0x + v_0)$$

resulting in a polynomial of degree $3 \cdot \deg(f)$. Our maps are then the x-coordinate maps corresponding to the isogenies

$$\begin{cases} K_1(x_1) = \frac{\xi_1(x_1)}{\psi^2(x_1)} = x_2 \\ \vdots \\ K_t(x_t) = \frac{\xi_t(x_t)}{\psi^2(x_t)} = x_{t+1} \end{cases}$$

$\phi_t \circ \cdots \circ \phi_1(P) = \mathcal{O} \implies$ all sequences are of length $< t + 1$.

Maps when $p = 2 \mod 3$ (IV)

Using Icart's map

$$x = (r^2 - v_0 - \frac{u^6}{27})^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + r, \qquad r = \frac{3w_0 - u^4}{6u}$$

Part (i) of this lemma gives us the polynomial transformation method

$$f^{(1)}(x_1) := \mathrm{Res}_u(f(u), u^4 - 6u^2x - 3w_0)^2 - 36u^2(x^3 + w_0x + v_0)$$

resulting in a polynomial of degree $3 \cdot \deg(f)$. Our maps are then the x-coordinate maps corresponding to the isogenies

$$\begin{cases} K_1(x_1) = \frac{\xi_1(x_1)}{\psi^2(x_1)} = x_2 \\ \vdots \\ K_t(x_t) = \frac{\xi_t(x_t)}{\psi^2(x_t)} = x_{t+1} \end{cases}$$

$\phi_t \circ \cdots \circ \phi_1(P) = \mathcal{O} \implies$ all sequences are of length $< t + 1$.

Maps when $p = 2 \mod 3$ (V)

Converting back

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Part (ii) of the lemma gives us a method to convert back.

We now know the x-coordinates and may compute the y-coordinates from our original elliptic curve $y^2 = x^3 + w_0x + v_0$.

To recover the original roots of the polynomial $f(u)$:

$$\begin{aligned}
\bar{y}_1 &:= \mathrm{Sqrt}(\bar{x}^3 + w_0\bar{x} + v_0) \\
\bar{y}_2 &:= -y_1 \\
\mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_1 - 3w_0) \\
\mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_2 - 3w_0) \\
\text{Return} &\longleftarrow \mathrm{Roots1} \cup \mathrm{Roots2}
\end{aligned}$$

Maps when $p = 2 \mod 3$ (V)

Converting back

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Part (ii) of the lemma gives us a method to convert back.

We now know the x-coordinates and may compute the y-coordinates from our original elliptic curve $y^2 = x^3 + w_0x + v_0$.

To recover the original roots of the polynomial $f(u)$:

$$\begin{aligned} \bar{y}_1 &:= \mathrm{Sqrt}(\bar{x}^3 + w_0\bar{x} + v_0) \\ \bar{y}_2 &:= -y_1 \\ \mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_1 - 3w_0) \\ \mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_2 - 3w_0) \\ \mathrm{Return} &\longleftarrow \mathrm{Roots1} \cup \mathrm{Roots2} \end{aligned}$$

Maps when $p = 2 \mod 3$ (V)

Converting back

$$u^4 - 6u^2x + 6uy - 3w_0 = 0$$

Part (ii) of the lemma gives us a method to convert back.

We now know the x-coordinates and may compute the y-coordinates from our original elliptic curve $y^2 = x^3 + w_0x + v_0$.

To recover the original roots of the polynomial $f(u)$:

$$\begin{aligned}
\bar{y}_1 &:= \mathrm{Sqrt}(\bar{x}^3 + w_0\bar{x} + v_0) \\
\bar{y}_2 &:= -y_1 \\
\mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_1 - 3w_0) \\
\mathrm{Roots1} &:= \gcd(f(u), u^4 - 6u^2\bar{x} + 6u\bar{y}_2 - 3w_0) \\
\mathrm{Return} &\longleftarrow \mathrm{Roots1} \cup \mathrm{Roots2}
\end{aligned}$$

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Open problems

- Open problem - what other structures exist for which we may exploit to create low-degree maps for $\mathbb{F}_{p^n}$?
- Open problem - can we use SRA for factoring directly?
- Open problem - does there exist a generic method to create efficient maps for any $\mathbb{F}_{p^n}$?
- Open problem - are there any applications where we can use SRA to pick out specific roots with certain properties defined by the map structure?
- Efficiency - generic methods of computing Resultant and GCD stages may be improved.
- Test real-world performance of all cases with efficient implementations.
- Investigate fully how SRA may be made deterministic under GRH.

Conclusions

- Introduced an abstracted form of the original SRA algorithm.
- Extended the maps to exploit rational maps instead of polynomials.
- Introduced three different ways to construct maps for finite fields with different properties which are efficient when the finite field is
  - $\mathbb{F}_{p^n}$ and has small characteristic
  - $\mathbb{F}_p$ and $p - 1$ is smooth
  - $\mathbb{F}_p$ and $p = 2 \mod 3$
- Gained some insight into the original SRA algorithm, in regards to the tree-structure and aims for new designs for maps.
- Proof of concept code written in SageMath[f] may be found at `https://www.github.com/bip20/SRA`

[f]

Conclusions

- Introduced an abstracted form of the original SRA algorithm.
- Extended the maps to exploit rational maps instead of polynomials.
- Introduced three different ways to construct maps for finite fields with different properties which are efficient when the finite field is
  - $\mathbb{F}_{p^n}$ and has small characteristic
  - $\mathbb{F}_p$ and $p-1$ is smooth
  - $\mathbb{F}_p$ and $p = 2 \mod 3$
- Gained some insight into the original SRA algorithm, in regards to the tree-structure and aims for new designs for maps.
- Proof of concept code written in SageMath[f] may be found at `https://www.github.com/bip20/SRA`

[f]

Conclusions

- Introduced an abstracted form of the original SRA algorithm.
- Extended the maps to exploit rational maps instead of polynomials.
- Introduced three different ways to construct maps for finite fields with different properties which are efficient when the finite field is
  - $\mathbb{F}_{p^n}$ and has small characteristic
  - $\mathbb{F}_p$ and $p-1$ is smooth
  - $\mathbb{F}_p$ and $p = 2 \mod 3$
- Gained some insight into the original SRA algorithm, in regards to the tree-structure and aims for new designs for maps.
- Proof of concept code written in SageMath[f] may be found at `https://www.github.com/bip20/SRA`

[f]

Conclusions

- Introduced an abstracted form of the original SRA algorithm.
- Extended the maps to exploit rational maps instead of polynomials.
- Introduced three different ways to construct maps for finite fields with different properties which are efficient when the finite field is
  - $\mathbb{F}_{p^n}$ and has small characteristic
  - $\mathbb{F}_p$ and $p - 1$ is smooth
  - $\mathbb{F}_p$ and $p = 2 \mod 3$
- Gained some insight into the original SRA algorithm, in regards to the tree-structure and aims for new designs for maps.
- Proof of concept code written in SageMath[f] may be found at https://www.github.com/bip20/SRA

[f]

Conclusions

- Introduced an abstracted form of the original SRA algorithm.
- Extended the maps to exploit rational maps instead of polynomials.
- Introduced three different ways to construct maps for finite fields with different properties which are efficient when the finite field is
  - $\mathbb{F}_{p^n}$ and has small characteristic
  - $\mathbb{F}_p$ and $p - 1$ is smooth
  - $\mathbb{F}_p$ and $p = 2 \mod 3$
- Gained some insight into the original SRA algorithm, in regards to the tree-structure and aims for new designs for maps.
- Proof of concept code written in SageMath[f] may be found at `https://www.github.com/bip20/SRA`

[f] `http://www.sagemath.com`