# A Generalised Successive Resultants Algorithm

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

[1] Univeristy of Bath, Bath BA2 7AY, UK
J.H.Davenport@bath.ac.uk
b.i.pring@bath.ac.uk
[2] University of Oxford, Oxford OX2 6GG, UK
Christophe.Petit@maths.ox.ac.uk

**Abstract.** The Successive Resultants Algorithm (SRA) is a root-finding algorithm for polynomials over $\mathbb{F}_{p^n}$ and was introduced at ANTS in 2014 [19]. The algorithm was designed to be efficient when the characteristic $p$ is small and $n > 1$. In this paper, we abstract the core SRA algorithm to arbitrary finite fields and present three instantiations of our general algorithm, one of which is novel and makes use of a series of isogenies derived from elliptic curves with sufficiently smooth order.

**Keywords:** root finding, finite fields, algorithms, elliptic curves

## 1 Introduction

The factorization of polynomials over finite fields is an important problem in computer algebra, both from theoretical and practical points of view [11]. An important subcase of this problem is the root-finding problem, which given a polynomial over a finite field, asks for one, several or all roots of this polynomial over the field. It is well-known that factoring polynomials is deterministically reducible to root-finding [3], so in this paper we will mostly focus on the root-finding problem.

### 1.1 Finding roots of polynomials over finite fields

From now on in this paper, let $\mathbb{F}_{p^n}$ be a finite field of size $p^n$ and $f$ be a polynomial of degree $d$ with coefficients in $\mathbb{F}_{p^n}$. As it is often the case in the literature, we will assume that $f$ is entirely split over $\mathbb{F}_{p^n}$ and that it has no repeated roots. One can reduce the general case to this one by computing $\gcd(f(x), x^{q^n} - x)$, for example using a variant of the square and multiply algorithm. We allow the notation $x$ and $f(x)$ to denote the variable and polynomial in $\mathbb{F}_{p^n}[x]$ with $\hat{x} \in \mathbb{F}_{p^n}$ and $f(\hat{x})$ to represent the evaluation of the polynomial $f$ at the value $\hat{x}$.

Since the seminal work of Berlekamp in the seventies [3], the root-finding problem can be solved in probabilistic polynomial time (in the degree of $f$ and in $n \log p$). Significant practical and theoretical improvements have been made since then, with the current best probabilistic algorithm for the general factorization

of a polynomial being due to Kedlaya and Umans [16]. In practice, one will often use either Berlekamp's trace algorithm [3] or Cantor-Zassenhaus algorithm [5], depending on the parameters.

Berlekamp's trace algorithm in fact provides a polynomial time reduction from the root-finding problem over $\mathbb{F}_{p^n}$ to the root-finding problem over $\mathbb{F}_p$. The reduction can be made deterministic, leading to a polynomial time deterministic algorithm for fields of small characteristic.

In contrast, Shoup's algorithm [23] is still the best unconditional deterministic algorithm over $\mathbb{F}_p$ today, with a complexity in $\tilde{O}(d^2\sqrt{p})$. Designing a deterministic polynomial time algorithm in that setting is an important open problem, even in the case of degree 2 polynomials. Evdokimov has provided a quasi-polynomial time algorithm when a quadratic non-residue is provided together with the field as an input to the algorithm [8]. Under the Generalised Riemann hypothesis (GRH), this element can be computed in polynomial time, removing the need for an extra input. Polynomial time algorithms have also been suggested under additional assumptions on the polynomial [21], other conjectures [9, 1] or for specific families of primes [2, 21, 24], still under GRH.

In 2014, Petit introduced a new algorithm in the small characteristic case, called the Successive Resultants Algorithm [19].

### 1.2 Our contributions

In this paper, we introduce a generalisation of the Successive Resultants Algorithm to arbitrary finite fields. Our generalisation covers both the original SRA algorithm for finite fields $\mathbb{F}_{p^n}$ with small characteristic and the generalised Graeffe transform approach of Grenet et al. [12] when $p^n - 1$ is smooth. We also present a third instance using an elliptic curve with smooth order over $\mathbb{F}_{p^n}$, leading to a new algorithm of independent interest.

Our initial observation is that the linearized polynomials used in SRA can be replaced by any set of polynomials, and in fact even rational maps $K_i$, such that the image of the composed map $K_t \circ K_{t-1} \circ \ldots \circ K_2 \circ K_1$ under a restricted domain is sufficiently small. Similar generalisations were made in different contexts in [20].

Like the original SRA, our generalisation reduces the root-finding problem for large degree polynomials to the same problem for "small" degree polynomials. The original SRA has two stages, a resultant stage and a gcd stage. We show how to adapt both stages to the case of arbitrary rational maps, and how to overcome the technical difficulties introduced by the denominators of the maps.

Recently, De Feo, Petit and Quisquater showed that the Successive Resultants Algorithm and Berlekamp's celebrated Trace Algorithm (BTA) [3] are in a certain sense dual of each other [18]. We remark that the generalised Graeffe transform algorithm mentioned above can similarly be seen as a dual of Shoup's algorithm when $p-1$ is smooth [24], and our new algorithm as a dual of a slight variant of an algorithm due to Ronyai [21, Section 7]. (See Table 1.)

In the algorithm of Section 3.3, we have used Icart's embeddings [15] to map $\mathbb{F}_p$ elements to the $x$-coordinates of $\mathbb{F}_p$-rational points of a smooth order elliptic

curve over $\mathbb{F}_p$, where Ronyai's algorithm would map them to a smooth curve over $\mathbb{F}_{p^2}$. Our approach has some efficiency advantages and more importantly it leads to a larger set of suitable parameters in the algorithm. We remark that a similar improvement can be brought to Ronyai's algorithm.

We remark that all our algorithms can be made deterministic after some precomputation depending on the field, assuming the Generalised Riemann Hypothesis. These deterministic versions can be seen in the continuity of [2, 21, 24], providing polynomial time deterministic algorithms under GRH for special fields.

**Table 1.** Special instances of our Successive Resultants Algorithm and corresponding "dual" algorithms

| | $p$ small | $p^n - 1$ smooth | Elliptic Curves |
|---|---|---|---|
| Resultant-based | [19] | [12] | Section 3 |
| GCD-based | [3] | [24] | [21, Section 4] |

## 2 A generalised form of the Successive Resultants Algorithm

The Successive Resultants Algorithm (SRA) is a root finding algorithm which exploits the properties of an ordered set of rational mappings in order to extract roots by computing the roots of polynomials of small degree. As explained in the introduction, we will be considering the problem of finding the roots of a polynomial $f \in \mathbb{F}_{p^n}[x]$ whose splitting field is $\mathbb{F}_{p^n}$.

The generation of the rational maps is a key factor in the efficiency and utility of the SRA algorithm. These maps may be considered as input to the algorithm and the existence of a useful set of maps currently depends upon the structure of $\mathbb{F}_{p^n}$. We note that the rational maps are independent of $f$ and may be performed as precomputation.

Given a polynomial $f$ of degree $d$ and a sequence of rational maps $K_1, \ldots, K_t$ the SRA algorithm involves computing finite sequences of length $j \leq t+1$ obtained by successively transforming the roots of $f$ by application of the rational maps. In other words, the sequences $(x_1, \ldots, x_j)$ of length 1 to $t+1$ fulfilling

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

where $K_i : \mathbb{F}_{p^n} \setminus N_i \to \mathbb{F}_{p^n}$, $N_i := \{x_i \in \mathbb{F}_{p^n} \ : \ b_i(x_i) = 0\}$ and $a_i, b_i \in \mathbb{F}_{p^n}[X]$ such that $\gcd(a_i(x_i), b_i(x_i)) = 1$.
We define the composed map

$$\begin{aligned} &K^{[j]}(x_1) : \mathbb{F}_{p^n} \setminus N^{[j]} \to \mathbb{F}_{p^n} \\ &K^{[j]}(x_1) = K_j \circ \ldots \circ K_0(x_1) \end{aligned} \tag{2}$$

where $N^{[j]} := \{x_1 \in \mathbb{F}_{p^n} \ : \ \forall\, i \in \{1,\ldots,j\}\ K_i \circ \ldots \circ K_1(x_1) \notin N_i\}$. For notation purposes, we take $K^{[0]}$ to be the identity map $\mathrm{Id}_{\mathbb{F}_{p^n}} : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ and $N^{[0]} = \emptyset$.

The SRA algorithm consists of two separate stages, the Resultant Stage and the GCD stage. In the Resultant stage, a series of polynomials $f^{(1)}(x_1), \ldots, f^{(t)}(x_t)$ are computed with $f^{(i+1)}(x_{i+1})$ relying on $f^{(i)}(x_i)$ and $K_i(x_i)$. The roots of $f^{(i)}(x_i)$ lying in $\mathbb{F}_{p^n}$ correspond to the existence of a sequence $(x_1, \ldots, x_j)$ with the root as the $i^{\text{th}}$ value in the sequence. In the GCD stage, the roots of $f^{(i+1)}(x_{i+1})$ are used to find the roots of $f^{(i)}(x_i)$ by computing roots of polynomials whose degree is constrained by the $K_i$ maps.

**Theorem 1.** *Given the maps $K_i : \mathbb{F}_{p^n} \setminus N_i \to \mathbb{F}_{p^n}$ for $i = 1, \ldots, t$ we have that each distinct root of $f^{(1)}(x_1)$ produces a unique sequence $(x_1, \ldots, x_j)$ where $j \leq t+1$, obtained by computing $K_i(x_i) := x_{i+1}$ while $x_i \notin N_i$.*

Proof: This may be seen by successively applying the maps $K_i$ to each root. If for some $j \in \{1, \ldots, t\}$ we have that $K_{j-1} \circ \ldots \circ K_0(x_1) \in N_j$, then the sequence is of length $j$. Otherwise the sequence is of length $t+1$. The sequence is unique for any distinct root of $f^{(1)}(x_1)$ by the fact that the root is the first value in any sequence. □

### 2.1 The Resultant Stage

We will use the basic result that the resultant posesses the property that for $f, g \in \mathbb{F}_{p^n}[x]$ we have that $\mathrm{Res}_x(f(x), g(x)) = 0$ if and only if the polynomials $f(x)$ and $g(x)$ share a common factor in $\mathbb{F}_{p^n}[x]$. The resultant of two polynomials $f, g \in \mathbb{F}_{p^n}[x]$ may be calculated via naively taking the determinant of the Sylvester matrix of $f$ and $g$ or a more specialised method depending upon the structure of the $K_j$ maps. We will use the standard result [17, Definition 1.93] that

$$\mathrm{Res}_x(f, g) = \mathrm{lc}(f)^{\deg(g)} \prod_{x : f(x) = 0} g(x) \tag{3}$$

where the roots are taken over the splitting field of $f$ and $\mathrm{lc}(f)$ is the leading coefficient of $f$.

We will use the resultant on polynomials in two variables $x_i, x_{i+1}$, with respect to the $x_i$ variable to create a series of univariate polynomials, $f^{(i+1)}(x_{i+1})$, whose roots correspond to a non-empty subset of roots of $f^{(i)}(x_i)$.

In the Resultant stage we clear the denominator of the $K_i$ rational function representation of the map and successively compute resultants of the resulting equation and the previously computed polynomial with respect to the variable $x_i$, defining $f^{(1)}(x_1) := f(x_1)$ as the first such polynomial. This results in the generation of an ordered list of polynomials $f^{(1)}(x_1), \ldots, f^{(t)}(x_t)$ via the procedure

$$\begin{cases} f^{(1)}(x_1) & := f(x_1) \\ f^{(i+1)}(x_{i+1}) & := \mathrm{Res}_{x_i}(f^{(j)}(x_i),\ a_i(x_i) - b_i(x_i) \cdot x_{i+1}) \text{ for } i = 1, \ldots, t-1 \end{cases} \tag{4}$$

This results in a sequence of polynomials $f^{(1)}, \ldots, f^{(t)}$ whose roots encode the potential values any $x_i$ may take for the sequences described in Theorem 1.

**Theorem 2.** *The polynomials $f^{(1)}, \ldots, f^{(t)}$ have the following properties:*

(*i*) *If $f^{(1)}$ splits over $\mathbb{F}_{p^n}$ then all $f^{(i)}$ split over $\mathbb{F}_{p^n}$.*
(*ii*) *The degree of any $f^{(i+1)}$ is less than or equal to the degree of $f^{(i)}$.*
(*iii*) *The degree of $f^{(i+1)}$ is strictly less than the degree of $f^{(i)}$ if and only if the gcd of $f^{(i)}$ and $b_i$ is non-trivial.*

Proof: We have that by formula (3), that

$$\begin{aligned} f^{(i+1)}(x_{i+1}) &= \mathrm{Res}_{x_i}(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot x_{i+1}) \\ &= \mathrm{lc}(f^{(i)})^{\deg(a_i(x_i) - b_i(x_i) \cdot x_{i+1})} \prod_{x_i : f^{(i)}(x_i) = 0} (a_i(x_i) - b_i(x_i) \cdot x_{i+1}) \end{aligned} \tag{5}$$

For $(i)$ we note that for any polynomials $a_i, b_i \in \mathbb{F}_{p^n}[x]$ and $\hat{x}_i \in \mathbb{F}_{p^n}$ we have that $a_i(\hat{x}_i), b_i(\hat{x}_i) \in \mathbb{F}_{p^n}$. For $(ii)$ & $(iii)$ as $f^{(i)}$ splits over $\mathbb{F}_{p^n}$ it is clear that the degree of $f^{(i+1)}(x_{i+1}) \leq d$ with equality holding if and only if $b_i(x_i)$ and $f^{(i)}(x_i)$ share no roots in $\mathbb{F}_{p^n}$. □

We note that in the case of polynomial maps, we have that $\deg(f^{(i)}) = d$ as $b_i(x_i) = 1$ and therefore possesses no roots.

**Theorem 3.** *If $f^{(1)}(x_1)$ splits over $\mathbb{F}_{p^n}$ then the union of all $i^{th}$ values for valid sequences $(x_1, \ldots, x_j)$ as produced in Theorem 1 is equal to the set of roots of each $f^{(i)}(x_i)$.*

Proof: We use induction to prove the result. As $f^{(1)}$ splits over $\mathbb{F}_{p^n}$, we have that the roots of $f^{(1)}(x_1)$ comprise exactly the first values of the sequences as we have defined $f^{(1)}(x_1) := f(x_1)$. Assuming that the set of roots of $f^{(i)}$ is equal to the set of possible $x_i$ sequence values, we have that, by the computation of the resultant and equation (5), the roots of $f^{(i+1)}(x_{i+1})$ are those values such that $f^{(i)}(x_i) = 0$ and $a_i(x_i) - b_i(x_i) \cdot x_{i+1} = 0$. If $x_i \in N_i$ then by the product equation of the resultant as in (5) we have that if $b_i(x_i) = 0$, there is no root of $f^{(i+1)}(x_{i+1})$ corresponding to a solution of $a_i(x_i) - b_i(x_i) \cdot x_{i+1} = 0$. If $x_i \notin N_i$, we have that $b_i(x_i) \neq 0$ and so the root $x_{i+1}$ satisfies both $f^{(i)}(x_i) = 0$ and $\frac{a_i(x_i)}{b_i(x_i)} = x_{i+1}$ and is therefore the corresponding point in a sequence. □
We note that if some $f^{(i)}$ does not split over $\mathbb{F}_{p^n}$ then if $x_i$ is a root of an irreducible factor of $f^{(i)}$ in the splitting field of $f^{(i)}$ we have the potential for $f^{(i)}(x_i) = 0$ and $a_i(x_i) - b_i(x_i) \cdot x_{i+1} = 0$ with $x_{i+1} \in \mathbb{F}_{p^n}$. This would lead to sequences of the form $(x_{i+1}, \ldots, x_j)$, but as we assume that $f^{(i)}$ splits over $\mathbb{F}_{p^n}$ we have that all $f^{(i)}$ split over $\mathbb{F}_{p^n}$ by Theorem 1.

**Theorem 4.** *The roots of $f^{(i)}$ lie in the image of $K^{[i-1]}$.*

Proof: By the property of the resultant, the roots of $f^{(i)}(x_i)$ possess the property that $\frac{a_{i-1}(x_{i-1})}{b_{i-1}(x_{i-1})} = x_i$. This successively constrains the potential values

that the roots of each $f^{(i)}(x_i)$ may take. $\square$

By Theorem 4, we have that by sensible choice of the $K_i$ maps, we may obtain a small set which contains our potential $x_{t+1}$ values. This will be useful in the GCD stage.

It is clear that as the roots are successively constrained by Theorem 4, the sequences may be considered as a series of trees of depth $j \leq t+1$ with the $\hat{x}_j$ forming the root nodes and the distinct subsets of the roots of $f$ corresponding to sequences of length $j$ forming the leaves. The SRA algorithm may be considered as a means of generating this tree with relation to the $K_i$ rational maps by first encoding the information concerning the nodes at each level with the Resultant stage and then computing the root and children nodes with the GCD stage.

### 2.2 The GCD stage

In the GCD stage we use the $f^{(1)}, \ldots, f^{(t)}$ polynomials computed in the Resultant stage to locate the final values of all sequences $(x_1, \ldots, x_j)$ as created by the procedure in Theorem 1. Once we have the final value, we recursively determine the sequence by application of the gcd algorithm and by computing roots of bounded degree polynomials until we have found the roots of $f^{(1)}(x_1) = f(x_1)$.

**Theorem 5.** *Given any $\hat{x}_{i+1} \in \mathbb{F}_{p^n}$ which forms part of a sequence as computed by successive application of the $K_i$ maps on the roots of $f$, we may compute all $i^{th}$ values of sequences which possess $\hat{x}_{i+1}$ as the $i+1^{th}$ value.*

PROOF: If $\hat{x}_{i+1} \in \mathbb{F}_{p^n}$ is such that there exists a sequence $(x_1, \ldots, x_i, \hat{x}_{i+1}, \ldots, \hat{x}_j)$ we have that all potential values of $x_i$ such that $K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} = \hat{x}_{i+1}$ are contained in the roots of $f^{(i)}(x_i)$ by Theorem 1.
As $\mathsf{Res}_{x_i}(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \hat{x}_{i+1}) = 0$, we have that

$$g^{(i)}_{\hat{x}_{i+1}}(x_i) := \gcd(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot \hat{x}_{i+1}) \tag{6}$$

is non-trivial and as $f^{(i)}(x_i)$ is split over $\mathbb{F}_{p^n}$, we have that (6) is a product of linear factors whose roots are exactly those such that $f^{(i)}(x_i) = 0$ and $K_i(x_i) = \frac{a_i(x_i)}{b_i(x_i)} = \hat{x}_{i+1}$. We may therefore extract all $x_i$ values whose next value in the sequence is $\hat{x}_{i+1}$ by finding the roots of a split polynomial of degree bounded by $\deg g^{(i)}_{\hat{x}_{i+1}} \leq \max\{\deg a_i, \deg b_i\}$. $\square$

**Theorem 6.** (*i*) *We may detect that there exists a sequence $(\hat{x}_1, \ldots, \hat{x}_j)$ of length $j < t+1$ and may compute $\hat{x}_j \in \mathbb{F}_{p^n}$ by computing the roots of a polynomial of degree no larger than* $\deg b_j$.

(*ii*) *Given* $\mathrm{Image}(K^{[t]})$ *we may detect that there exists a sequence $(\hat{x}_1, \ldots, \hat{x}_{t+1})$ of length $t+1$ and its final value $\hat{x}_{t+1} \in \mathbb{F}_{p^n}$ by computing the roots of* $|\mathrm{Image}(K^{[t]})|$ *polynomials of degree no larger than* $\max\{\deg a_t, \deg b_t\}$.

PROOF: (i) If a sequence is of length $j < t+1$, then $x_j \in N_j$. If this is the case, then it is detected by observing that $\deg f^{(j+1)} < \deg f^{(j)}$ as in Theorem 2. As $x_j \in N_j$, we have that $b_j(x_j) = 0$ and $f^{(j)}(x_j) = 0$, so we may compute

$$g^{(j)}(x_j) := \gcd(f^{(j)}(x_j), b_j(x_j)) \tag{7}$$

whose degree is $\leq \deg b_j$ and whose roots are the final values of all sequences of length $j$. If a sequence of length $t+1$ exists then we may compute

$$g^{(t)}_{\hat{x}_{t+1}}(x_t) := \gcd(f^{(t)}(x_t), a_t(x_t) - b_t(x_t) \cdot \hat{x}_{t+1}) \tag{8}$$

for each $\hat{x}_{t+1} \in \text{Image}(K^{[t]})$. For sequences of length $t$ there will be no indication of degree drop and so we must compute $g^{(t)}(x_t)$. In the case where a sequence of length $t+1$ exists, we have by Theorem 4 that the potential values of $x_t$ lie in the roots of $f^{(t)}(x_t)$ and that $b_t(x_t) \neq 0$. The $x_t$ for which $K_t(x_t) = \frac{a_t(x_t)}{b_t(x_t)} = \hat{x}_{t+1}$ may therefore be extracted by computing the roots of (8). In the case where no such sequence exists for a chosen $\hat{x}_{i+1}$ then by Theorem 4 we have that (8) is trivial. □

Taken together with a sensible choice of mappings to constrain each $|\text{Image}(K^{[i]})|$ for $i = 1, \ldots, t$, Theorems 4, 5 and 6 allow us to find the length and final value of all sequences. For sequences of length $j < t+1$, we may use Theorem 4, whilst for sequences of length $t+1$ Theorem 5 constrains the possible values which $x_{t+1}$ may take. Ideally, we will wish $\text{Image}(K^{[t]})$ to contain only one element.

Together the Resultant stage and the GCD stage give us the generalised SRA.

**Algorithm 1:** The generalised Successive Resultants Algorithm

**Data**: $f \in \mathbb{F}_{p^n}[x]$ – the polynomial whose roots we wish to find
$\frac{a_1}{b_1}, \ldots, \frac{a_t}{b_t}$ – a set of rational maps
$B \subseteq \mathbb{F}_{p^n}$ – a set of points contained in Image$(K^{[t]})$
ParSeq $\in \{\mathrm{T}, \mathrm{F}\}$ – whether to extract roots not in ${K^{[t]}}^{-1}(\mathbb{F}_{p^n})$

**Result**:
The roots of $f$ in ${K^{[t]}}^{-1}(B)$ and optionally all roots not in ${K^{[t]}}^{-1}(\mathbb{F}_{p^n})$.

**begin**
- $f^{(1)}(x_1) \longleftarrow f(x_1)$
- **for** $i = 1, \ldots, t-1$ **do**
  - $f^{(i+1)}(x_{i+1}) \longleftarrow \mathsf{Resultant}_{x_i}(f^{(i)}(x_i), a_i(x_i) - b_i(x_i) \cdot x_{i+1})$
- CandidateRoots $\longleftarrow B$
- **for** $i = t, \ldots, 1$ **do**
  - TempRoots $\longleftarrow \{\}$
  - **for** $y \in$ CandidateRoots **do**
    - $g_y(x_i) \longleftarrow \mathsf{gcd}(f^{(t)}(x_i), a_i(x_i) - b_i(x_i) \cdot y)$
    - TempRoots $\longleftarrow$ TempRoots $\cup\, \mathsf{Roots}(g_y(x_i))$
  - **if** PartialSeq **and** $(i < t$ **and** $\deg(f^{(i+1)}) < \deg(f^{(i)})$ **or** $i == t)$ **then**
    - $g_b(x_i) \longleftarrow \mathsf{gcd}(f^{(i)}(x_i), b_i(x_i))$
    - TempRoots $\longleftarrow$ TempRoots $\cup\, \mathsf{Roots}(g_b(x_i))$
  - CandidateRoots $\longleftarrow$ TempRoots
- **return** CandidateRoots

We note that SRA may also be used to explicity extract roots possessing certain properties. The algorithm may specifically pick out only roots corresponding to sequences of specific length or roots which intersect with ${K^{[j]}}^{-1}(B)$ for $B \subseteq$ Image$(K^{[j]})$.

### 2.3 Generic complexity analysis

Complexity is given in terms of basic operations in $\mathbb{F}_p$. We allow the notation that $h(n)$ is $\tilde{O}(h(n))$ if $h(n)$ is $O(h(n)\log^c h(n))$. The cost of these operations is $O(\log p)$ for addition, $O((\log p)^2)$ for multiplication and inversion with classical arithmetic or $\tilde{O}(\log p)$ for addition, multiplication and $\tilde{O}((\log p)^2)$ for exponentiation with fast FFT-style arithmetic, such as via the Schönhage-Strassen algorithm [10]. We write $\mathbf{a}(n)$ and $\mathbf{m}(n)$ to represent the cost of addition and multiplication over $\mathbb{F}_{p^n}$. We let $\mathbf{M}(d)$ and $\mathbf{G}(d)$ represent the cost of performing the multiplication and taking the gcd of two polynomials of degree $d$ in $\mathbb{F}_{p^n}[x]$. We allow $\mathbf{R}(d)$ to be the cost of computing the resultant of two polynomials in $\mathbb{F}_{p^n}[y][x]$ with respect to $x$ where the maximum degree of either polynomials in

$x$ is $d$ and the maximum degree of $y$ is 1. We represent the cost of finding the roots in $\mathbb{F}_{p^n}$ of a degree $d$ polynomial to be $\mathbf{P}(d)$.

The following table summarises the cost of performing these with regard to $\mathbb{F}_{p^n}$ in terms of basic operations over $\mathbb{F}_p$ [10].

| | $\mathbf{a}(n)$ | $\mathbf{m}(n)$ | $\mathbf{A}(d)$ | $\mathbf{M}(d)$ | $\mathbf{G}(d)$ | $\mathbf{R}(d)$ | $\mathbf{P}(d)$ |
|---|---|---|---|---|---|---|---|
| Classical | $O(n)$ | $O(n^2)$ | $O(dn)$ | $O(d^2n^2)$ | $O(d^2n^2 \log d)$ | $O(d^3n^2 \log d)$ | $O(d^2n^3)$ |
| Fast | $O(n)$ | $\tilde{O}(n)$ | $O(dn)$ | $\tilde{O}(dn)$ | $\tilde{O}(dn)$ | $\tilde{O}(d^2n))$ | $\tilde{O}(dn^2)$ |

**Table 2.** Cost of operations over $\mathbb{F}_{p^n}$ in terms of basic operations over $\mathbb{F}_p$.

We assume that we are attempting to find the roots of the polynomial $f \in \mathbb{F}_{p^n}[x]$ of degree $d$ such that $f$ possesses $d$ distinct roots, all defined over $\mathbb{F}_{p^n}$. Whilst the complexity of the SRA algorithm depends upon the choice of $K_i$ rational maps, we may assume there are $t$ maps and that these have been provided by a precomputation. We assume that the maximum degree of any $a_i, b_i \in \mathbb{F}_{p^n}[x]$ is $B$, that $|\text{Image}(K^{[t]})| = L$ and that $d \geq \max\{B, L\}$. Computing purely the linear factors of a polynomial of degree $d$ may be done at a cost of $O(\mathbf{m}(n)\mathbf{M}(d) \log d \log(p^n d))$ operations in $\mathbb{F}_p$ [10]. We also assume that the maps $K_i$ have been precomputed and exclude this cost.

We have that the resultant stage will consist of taking $t$ resultants of bivariate polynomials where the maximum degree of $x$ is $d$ and the degree of $y$ is always 1. After noting if $\deg f^{(i)} < \deg f^{(i+1)}$, we may compute the square-free part by a cost bounded by $O(\mathbf{M}(n) \log d)$ - this cost is overwhelmed by the computation of the first resultant. We therefore have that the resultant stage generically costs $O(td^3n^2 \log d)$ with classical arithmetic or $\tilde{O}(td^2n)$ with fast arithmetic.

Each of the $t$ steps in the GCD consists of taking a maximum of $d$ gcds of maximum degree $d$ resulting in polynomials of degree bounded by $B$ which require solving. At any stage, at most $\frac{d}{2}$ of these will be of degree $\geq 2$ and will require solving. This results in a total cost of $O(td^3n^2 \log d + td\mathbf{P}(B))$ for classical arithmetic and $\tilde{O}(td^2n + td\mathbf{P}(B))$ for fast arithmetic.

We therefore have the generic cost of the SRA algorithm is $O(td^3n^2 \log d + td\mathbf{P}(B))$ with classical arithmetic or $\tilde{O}(td^2n + td\mathbf{P}(B))$ with fast arithmetic. As the $f^{(i)}$ are square-free, we may obtain the linear factors by using equal-degree splitting for a cost of $O(\mathbf{m}(n)\mathbf{M}(B) \log d \log(p^n B))$. This gives us a total complexity cost of $\tilde{O}(td^3n^2 + tBd^3n^5)$ for classical arithmetic or $\tilde{O}(td^2n + tBd^2n^3)$ with fast arithmetic. We note that optimized versions of the separate stages may be constructed by exploiting the structure of the $K_i$ maps [13, 19].

## 3 Instantiations of SRA

In this section we present three instantiations of our generalised SRA algorithm. We first show how the original SRA algorithm fits into our generalised version. We then present an algorithm for efficiently determining the roots of a polynomial $f \in \mathbb{F}_p[x]$ when $p - 1$ is smooth; this algorithm is equivalent to the one in [12]. We conclude with a description of a method to transform polynomials

with roots in $\mathbb{F}_p^*$ into a polynomial whose roots correspond to $x$-coordinates of an elliptic curve defined over $\mathbb{F}_p$ when $p = 2 \mod 3$. This method demonstrates a procedure with which we can exploit the structure of $\mathbb{F}_p$ to generate the required rational maps. Proof of concept code written in SageMath [7] for all three cases may be found at `https://github.com/bip20/SRA`.

### 3.1 Original SRA algorithm

In this section we show how the original SRA algorithm fits into the framework of our generalised algorithm. The original SRA algorithm was designed for extension fields $\mathbb{F}_{p^n}$. It uses $n$ polynomial maps $K_i$ of degree $p$, hence the algorithm is only efficient for small characteristic fields. The polynomials $K_i$ are chosen as follows. For any $\{v_1, \ldots, v_n\}$ a basis of $\mathbb{F}_{p^n}$ over $\mathbb{F}_p$, we define the system of *linearized polynomials*

$$\begin{cases} L_0(z) & = z \\ L_i(z) & = \prod_{i \in \mathbb{F}_p} L_{i-1}(z - iv_i) \qquad \text{for } i = 1, \ldots, n \end{cases} \tag{9}$$

whose composition is $L(z) = L_n \circ \cdots \circ L_0(z) = z^{p^n} - z$. The system

$$K_i(x_i) = x_i^p - c_i x_i = x_{i+1} \qquad \text{for } i = 1, \ldots, n \tag{10}$$

may be derived from system (9) by means of the setting $x_i = L_i(z)$ and the $c_i$ may be precomputed for a cost of $O(n^4)$ with classical arithmetic or $\tilde{O}(n^3)$ with fast arithmetic.

From system (9) we may deduce that $\text{Image}(K^{[n]}) = \{0\} \subset \mathbb{F}_{p^n}$ and we may call the SRA algorithm as described in (2) with the precomputed $K_i$ polynomials and $\{0\}$. As the maps are polynomials, all sequences will be of full length, so there is no need to check for partial sequences.

A straight forward application of the SRA algorithm as described in section 2 would lead to a cost of $\tilde{O}(d^3n^3 + pd^3n^6)$ with classical arithmetic and $\tilde{O}(d^2n^2 + pd^2n^4)$ for fast arithmetic. The algorithm possesses several optimizations for both the Resultant stage and the GCD stage which utilise the structure of the polynomial maps (10), fast arithmetic, multipoint evaluation and reuse calculations. This leads to a cost of $O(d^2n^3)$ with classical arithmetic or $\tilde{O}(dn^2)$ with fast arithmetic excluding the precomputation of the $c_i$ values [19]. These optimizations have allowed it outperform traditional algorithms such as Berlekamp's Trace Algorithm for certain parameters [19].

### 3.2 SRA maps for $\mathbb{F}_p^*$ of smooth order

In this section we explore the use of the SRA algorithm in the field $\mathbb{F}_p$ where $\mathbb{F}_p^*$ is of smooth order. We note that this is equivalent to an algorithm based on *Generalised Graeffe transforms* [12, 13]

**Definition 1.** *For any integer $n$ we will denote the* smoothness function *$S : \mathbb{N} \longrightarrow \mathbb{N}$ by $S(n) = \max\{p \ : \ p \text{ is a prime factor of } n\}$.*
*We say that an integer $n \in \mathbb{N}$ is* $B$-smooth *if $S(n) \leq B$.*

We will assume that $f \in \mathbb{F}_p[x]$ is of degree $d$ and that $f$ splits over $\mathbb{F}_p$ and is square-free. From Fermat's little theorem we have that $x^{p-1} - 1 = 0$ for $x \in \mathbb{F}_p^*$. As we have that $p - 1 = n_1 \cdots n_t$ we exploit this structure to to create the following system of maps

$$K_i(x_i) = x_i^{n_i} = x_{i+1} \qquad \text{for } i = 1, \ldots, t \tag{11}$$

with $K^{[t]}(x_1) = K_t \circ \cdots \circ K_1(x_1) = x_1^{n_1 \cdots n_t} = x_1^{p-1}$ and $\text{Image}(K^{[t]}) = \{0, 1\}$. We may therefore use the $K_i$ maps and $\{0, 1\}$ as input to the SRA algorithm. As with the original SRA, the maps are polynomials and so all sequences will be of length $t + 1$.

**An optimized resultant stage**
We may assume that $t \leq \log p$ and that $B = S(p - 1)$. A straightforward adaptation of the algorithm would cost $O(Bd^3)$ with classical arithmetic or $\tilde{O}(Bd^2)$ with fast arithmetic.

After checking whether 0 is a root of $f$, we may use an improved method of computing the resultant which requires the precomputation of a root of unity. Full details of procedure is described in [13]. This method replaces taking each resultant and is based upon the result that

$$\begin{aligned} f^{(i+1)}(x_{i+1}) &:= \mathsf{Res}_{x_i}(f^{(i)}(x_i), x_i^{n_i} - x_{i+1}) \\ f^{(i+1)}(x_{i+1}^{n_i}) &= \prod_{k \in \{1, \ldots, n_i\}} f^{(i)}(\zeta_{n_i}^k x_{i+1}) \end{aligned} \tag{12}$$

where $\zeta_{n_i}$ is an $n_i{}^{\text{th}}$ root of unity. After computing $f^{(i+1)}(x_{i+1}^{n_i})$ this way and shifting the coefficients to obtain $f^{(i+1)}(x_{i+1})$ the total cost of taking each resultant costs $O(\mathbf{M}(dB) \log p)$ instead of $O(d\mathbf{M}(d) \log d)$. This optimization results in a total cost for the Resultant stage of $O(d^2 B^2 \log p)$ with classical arithmetic and $\tilde{O}(dB)$ with fast arithmetic.

**Complexity analysis**
We assume that we wish to find the roots of a polynomial $f \in \mathbb{F}_p[x]$ of degree $d$ lying in $\mathbb{F}_p$ and that $S(p-1) = B$. Using the standard GCD stage and the optimized resultant gives us a complexity of $\tilde{O}(d^2 B^2 + Bd^3)$ with classical arithmetic and $\tilde{O}(Bd^2)$ with fast arithmetic.

### 3.3 SRA map in conjunction with hashing to an elliptic curve

We now generalise the previous instance by working with the group of rational points of an elliptic curve over $\mathbb{F}_p$ instead of the multiplicative group $\mathbb{F}_p^*$. This generalisation is analogous to Lenstra's generalisation of Pollard's $p - 1$ factorization method as the elliptic curve factorization method.

We assume that the field $\mathbb{F}_p$ is provided with an elliptic curve $E_{a,b}$ (in reduced Weierstass coordinates $Y^2 = X^2 + aX + b$) of smooth order $N = \prod_{i=1}^{t} n_i$ over

$\mathbb{F}_p$ and a sequence of isogenies $\varphi_i : E_i \to E_{i+1}$, where $E_1 = E_{a,b}$ and $\varphi_i$ has degree $n_i$. It is well-known that $\varphi_i$ can be defined as $\varphi_i(x,y) = \left(\frac{\xi_i(x)}{\psi_i^2(x)}, y\frac{\omega_i(x)}{\psi_i^3(x)}\right)$ where $\xi_i, \omega_i, \psi_i$ are polynomials [28]. From this data we define the rational maps $K_i : \mathbb{F}_p \to \mathbb{F}_p : x \to \frac{\xi_i(x)}{\psi_i^2(x)}$. The composition map $K^{[t]}(x) = K_t \circ \ldots \circ K_1$ clearly maps to infinity all $\mathbb{F}_p$ elements that are the $x$-coordinate of some $P \in E_{a,b}(\mathbb{F}_p)$. By Hasse's theorem, this set covers roughly half of the elements in $\mathbb{F}_p$ [25].

Applying the SRA algorithm with those maps on $f$, we would not be able to separate roots that are not the $x$-coordinates of a point in $E$. At this point, one could try to split the remaining factor $f'$ by applying SRA again on $f'(x-\alpha)$, for $\alpha$ randomly chosen in $\mathbb{F}_p$. This algorithm, somehow reminiscent of Berlekamp's trace algorithm, would probably work well in practice but it is not clear how it could be rigorously analyzed. An alternative approach would be to assume that $E$ has a smooth order over $\mathbb{F}_{p^2}$ instead of a smooth order over $\mathbb{F}_p$. This approach would be more satisfactory from a theoretical point of view but it would also put more severe restrictions on the set of parameters, requiring that the curve order is smooth over $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_p$. This is essentially the approach taken by Ronyai [21] for a different algorithm.

In this paper, we use recent progress on hashing into elliptic curves [15] to solve this problem in a different way, which moreover fits nicely within our generalised SRA framework. We first recall the following results from Icart [15].

**Lemma 1 ([15]).** *Let $p = 2 \bmod 3$ be an odd prime. For any $z \in \mathbb{F}_p$, there is a unique cube root of $z$ defined over $\mathbb{F}_p$, which we write $z^{1/3}$. For any $a, b \in \mathbb{F}_p$ let $E_{a,b}$ be the elliptic curve defined by the equation $y^2 = x^3 + ax + b$. The map $f_{a,b} : \mathbb{F}_p \to E_{a,b}$ sending 0 to the point at infinity and $u \in \mathbb{F}_p^*$ to $(x,y) \in E_{a,b}(\mathbb{F}_p)$ where*

$$x = \left(v^2 - b - \frac{u^6}{27}\right)^{\frac{1}{3}} + \frac{u^6}{3}, \qquad y = ux + v, \qquad v = \frac{3a - u^4}{6u}, \tag{13}$$

*is a well-defined surjective map. Reciprocally, if $P = (x,y)$ is a point on the curve $E_{a,b}$, then the solutions $u_s$ of $f_{a,b}(u_s) = P$ are the solutions of the polynomial equation $u^4 - 6u^2x + 6uy - 3a = 0$.*

The map $f_{a,b}$ defined in Lemma 1 is in fact an algebraic map as $z^{1/3} = z^{(2p-1)/3}$.

Let $K_{a,b} : \mathbb{F}_p \to \mathbb{F}_p$, where

$$u \to \left(v^2 - b - \frac{u^6}{27}\right)^{\frac{1}{3}} + \frac{u^6}{3} \tag{14}$$

be the composition of $f_{a,b}$ with a projection on the $x$-coordinate of the curve. Lemma 1 implies that the modified composition map $K'(x) = K_t \circ \ldots \circ K_1 \circ K_{a,b}$ maps all $\mathbb{F}_p$ elements to infinity. However, the degree of $K_{a,b}$ is prohibitively large to run SRA efficiently. We therefore modify our algorithm as follows.

In the first resultant step instead of computing $\mathrm{Res}_u(f(u), x - K_{a,b}(u))$, we compute

$$f^{(1)}(x) = \mathrm{Res}_u(f(u), \tilde{K}_{a,b}(u,x))$$

where

$$\tilde{K}_{a,b}(u,x) := (u^4 - 6u^2x - 3a)^2 - 36u^2(x^3 + ax + b).$$

Note that $\deg f^{(1)} = 3 \deg f$ as $\tilde{K}_{a,b}$ has degree 3 with respect to variable $x$. In fact, for every root $u$ of $f$ the three values

$$\xi^i \Big(v^2 - b - \frac{u^6}{27}\Big)^{\frac{1}{3}} + \frac{u^6}{3}, \quad i = 1, 2, 3,$$

where $\xi$ is a primitive cube root of unity, are roots of the polynomial $f^{(1)}$. Since only one value in each triple is defined over $\mathbb{F}_p$, one can eliminate the other "parasitic" roots by replacing $f^{(1)}(x)$ by $\gcd(f^{(1)}(x), x^p - x)$ at this stage. Alternatively, one can just ignore this issue and work with bigger polynomials, and eventually the SRA algorithm will only produce $\mathbb{F}_p$ roots anyway. The choice of computing a gcd or not may depend on the parameters; we will not explore this further here.

After this conversion is completed we may call the original SRA algorithm to find the roots of $f^{(1)}(x)$ via the rational maps derived from the isogenies. As we know that the composed map of isogenies maps all elements in $E_{a,b}(\mathbb{F}_p)$ to the point at infinity, we know that all roots of $f^{(1)}(x)$ will result in partial sequences. Therefore we do not have to calculate or supply SRA with the set of points in the image of $K^{[t]}$. Once the roots are returned from the SRA algorithm, we compute the potential corresponding $y$ coordinates on $E_{a,b}(\mathbb{F}_p)$ for each root $x$ and use the final equation from Lemma 1 to recover the roots by taking gcds with our original polynomial.

We conclude the section with a comment on the existence and computation of suitable parameters for this variant of SRA. The existence of a curve of order $N$ over $\mathbb{F}_p$ is equivalent to the existence of an integer solution to the equation

$$(N + 1 - p)^2 - Df^2 = 4N$$

with $D < 0$ (see [4, Equation 4.3]). Once this solution is known, the curve can be constructed using the complex multiplication algorithm [4, p.30] provided that the *reduced discriminant* $D$ is small enough. Finally, computing small degree isogenies can be done efficiently with Vélu's formulae [27]. In order to find suitable parameters for the algorithm of this section, one can therefore for example first fix $D$ small, then choose $N$ randomly among a set of numbers of the desired smoothness, and finally solve the above equation for $p$ and $f$ using Cornacchia's algorithm [6].

**Complexity analysis**

The first step requires that we take one resultant of two bivariate polynomials, where the degree of $u$ is $d$ and the degree of $x_1$ is 3. We must then normalise the resulting polynomial $f^{(1)}(x_1)$ by removing the irreducible factors. This step costs $O(d\mathbf{M}(d) \log d + \mathbf{M}(p) \log p)$. SRA is called with $f^{(1)}$ and the rational maps derived from the isogenies. There will be a maximum of $\lceil \log p \rceil$ rational maps with their degree bounded by the smoothness of $|E_{a,b}|$ and denoted $B$.

Finally the roots must be converted back, costing $O(\mathbf{G}(d))$. We therefore have the total complexity of the algorithm is $\tilde{O}(p^2 + Bd^3)$ with classical arithmetic and $\tilde{O}(p + Bd^2)$ with fast arithmetic.

**Algorithm 2:** The SRA algorithm in conjunction with Icart's map

**Data**: $f \in \mathbb{F}_p[u]$ – the polynomial whose roots we wish to find
$M := \{\frac{a_1}{b_1}, \ldots, \frac{a_t}{b_t}\}$ – rational map representation of the isogenies
$a, b \in \mathbb{F}_p$ – description the smooth order curve $E_{a,b}$
**Result**: The roots of $f$
**begin**
  $f^{(1)}(x_1) \longleftarrow \mathsf{Res}_u(f(u), (u^4 - 6u^2x_1 - 3a)^2 - 36u^2(x_1^3 + ax_1 + b))$
  $f^{(1)}(x_1) \longleftarrow \mathsf{gcd}(f^{(1)}(x_1), x_1^p - x_1)$
  UnconvertedRoots $\longleftarrow \mathsf{SRA}(f^{(1)}(x_1), M, \{\}, True)$
  Roots $\longleftarrow \{\}$
  **for** $x_1 \in$ UnconvertedRoots **do**
    $y_1 \longleftarrow \mathsf{Sqrt}(x_1^3 + ax + b)$
    $y_2 \longleftarrow -y_1$
    Roots $\longleftarrow$ Roots $\cup\, \mathsf{gcd}(f(u), u^4 - 6u^2x_1 + 6uy_1 - 3a)$
    Roots $\longleftarrow$ Roots $\cup\, \mathsf{gcd}(f(u), u^4 - 6u^2x_1 + 6uy_2 - 3a)$
  **return** Roots

## 4 Conclusions and open problems

In this paper, we provided a framework to extend the Successive Resultants Algorithm of [19] to arbitrary finite fields. As it stands we have three sets of maps for which the SRA algorithm works in an efficient manner. The maps for SRA in the $p = 2 \mod 3$ case exploit the structure of the rational map framework introduced in section 2 and additionally require Icart's map to transform the polynomial before converting our solutions back into roots of our original polynomial. We believe that the creation of suitable maps for specific finite fields exploiting the structure of the rational map framework remains an interesting open problem.

We remark that the SRA algorithm may be used to solve the problem of deterministic root finding for polynomials in $\mathbb{F}_p[x]$ under the *Generalised Riemann Hypothesis* (GRH), which gives us the result that finding a generator of may be done in time $O(\ln^6 p)$. Provided with such a generator, we may use deterministic algorithms to find square roots [22, 26] and cubic roots [14] allowing us to exploit the standard formula for finding roots of quadratic, cubic and quartic equations. As long as $\max\{\deg a_i, \deg b_i\} \leq 4$, we may therefore use SRA in a deterministic manner.

Given the various ways that the algorithm may operate, in terms of choosing either to include elements which produce partial sequences or not and restriction of the input $B \subseteq \text{Image}(K^{[t]})$ to contain specific elements, there also exists the possibility of using SRA to extract only roots with specific properties as defined

by the $K_i$ maps. Preliminary experiments also suggest the possibility that the SRA algorithm may be used for general-purpose factoring directly, as opposed to root finding.

## 5 Acknowledgements

Christophe Petit is supported by a GCHQ research grant and Benjamin Pring is supported by an EPSRC doctoral research grant. The authors would like to thank the anonymous reviewers both for their time and for their helpful advice, much of which was incorporated into the final paper.

## References

[1] Arora, M., Ivanyos, G., Karpinski, M., Saxena, N.: Deterministic polynomial factoring and association schemes. Electronic Colloquium on Computational Complexity 19, 68 (2012)

[2] Bach, E., von zur Gathen, J., Lenstra, H.: Deterministic factorization of polynomials over special finite fields. University of Wisconsin-Madison, Computer Sciences Department (1988)

[3] Berlekamp, E.: Factoring polynomials over large finite fields. Mathematics of computation 111, 713–735 (1970)

[4] Bröker, R.: Constucting elliptic curves of prescribed order. Ph.D. thesis, University of Leiden (2006)

[5] Cantor, D.G., Zassenhaus, H.: A new algorithm for factoring polynomials over finite fields. Mathematics of Computation 36 (154), 587–592 (1981)

[6] Cornacchia, G.: Su di un metodo per la risoluzione in numeri interi dell' equazione $\sum_{h=0}^{n} c_h x^{n-h} y^h = p$. Giornale di Matematiche di Battaglini 46, 33–90 (1903)

[7] Developers, T.S.: SageMath, the Sage Mathematics Software System (Version 6.8) (2015), `http://www.sagemath.org`

[8] Evdokimov, S.: Factorization of polynominals over finite fields in subexponential time under GRH. In: Adleman, L.M., Huang, M.A. (eds.) Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings. Lecture Notes in Computer Science, vol. 877, pp. 209–219. Springer (1994)

[9] Gao, S.: On the deterministic complexity of factoring polynomials. Journal of Symbolic Computation 31, 19 – 36 (2001)

[10] von zur Gathen, J., Gerhard, J.: Modern computer algebra. Cambridge university press (2013)

[11] von zur Gathen, J., Panario, D.: Factoring polynomials over finite fields: A survey. Journal of Symbolic Computation 31(1/2), 3–17 (2001)

[12] Grenet, B., van der Hoeven, J., Lecerf, G.: Randomized Root Finding over Finite FFT-fields Using Tangent Graeffe Transforms. In: Proceedings of ISSAC. pp. 197–204. ACM (2015)

[13] Grenet, B., van der Hoeven, J., Lecerf, G.: Deterministic root finding over finite fields using Graeffe transforms. Applied Algebra in Engineering, Communication and Computing 27(3), 237–257 (2016)

[14] Harasawa, R., Sueyoshi, Y., Aichi, K.: Root computation in finite fields. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 96(6), 1081–1087 (2013)
[15] Icart, T.: How to hash into elliptic curves. In: Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. pp. 303–316 (2009)
[16] Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM Journal on Computing 40(6), 1767–1802 (2011)
[17] Lidl, R., Niederreiter, H.: Finite fields, vol. 20. Cambridge university press (1997)
[18] Luca De Feo, Christophe Petit, M.Q.: Application of the affine geometry of $GF(q^n)$ to root finding. Poster presented at International Symposium on Symbolic and Algebraic Computation (2015)
[19] Petit, C.: 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)
[20] Petit, C., Kosters, M., Messeng, A.: Algebraic approaches for the elliptic curve discrete logarithm problem over prime fields. In: Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II (2016)
[21] Rónyai, L.: Galois groups and factoring polynomials over finite fields. SIAM Journal on Discrete Mathematics 5(3), 345–365 (1992)
[22] Shanks, D.: Five number-theoretic algorithms. In: Proceedings of the second Manitoba conference on numerical mathematics. vol. 5170 (1972)
[23] Shoup, V.: On the deterministic complexity of factoring polynomials over finite fields. Information Processing Letters 33(5), 261–267 (1990)
[24] Shoup, V.: Smoothness and factoring polynomials over finite fields. Information processing letters 38(1), 39–42 (1991)
[25] Silverman, J.H.: Heights and elliptic curves. In: Arithmetic geometry, pp. 253–265. Springer (1986)
[26] Tonelli, A.: Bemerkung über die auflösung quadratischer congruenzen. Nachrichten von der Königl. Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen 1891, 344–346 (1891)
[27] Vélu, J.: Isogénies entre courbes elliptiques. Communications de l'Académie royale des Sciences de Paris 273, 238–241 (1971)
[28] Washington, L.C.: Elliptic curves: number theory and cryptography. CRC press (2008)

# 6 Appendix

## 6.1 Example of SRA with $|\mathbb{F}_p^*|$ smooth

We demonstrate the $p-1$ instantiation of SRA with a toy example. We use the finite field $\mathbb{F}_{37}$, where $36 = 2 \cdot 2 \cdot 3 \cdot 3$ is 3-smooth. Precomputation for $\mathbb{F}_{37}$ gives us the series of rational maps (in fact polynomials)

$$\begin{cases} K_1(x_1) & = x_1^2 = x_2 \\ K_2(x_2) & = x_2^2 = x_3 \\ K_3(x_3) & = x_3^3 = x_4 \\ K_4(x_4) & = x_4^3 = x_5 \end{cases} \tag{15}$$

The composed map is $K^{[4]}(x_1) = x_1^{36}$, which gives us $B = \text{Image}(K^{[t]}) = \{0, 1\}$. We wish to find the roots of

$$f(x) = x^{10} + 21x^9 + 22x^8 + 7x^7 + 12x^6 + 25x^5 + 35x^4 + 4x^3 + 25x \tag{16}$$

We first compute $f^{(i+1)}(x_{i+1}) = \mathsf{Res}_{x_i}(f^{(x_i)}, x_i^{n_i} - x_{i+1})$ with $f^{(1)}(x_1) = f(x_1)$.

$$f^{(1)}(x_1) = x_1^{10} + 19x_1^9 + 25x_1^8 + 6x_1^7 + 22x_1^6 + 32x_1^5 + 13x_1^4 + 32x_1^3 + 6x_1^2 + 24x_1$$
$$f^{(2)}(x_2) = x_2^{10} + 22x_2^9 + 34x_2^8 + 22x_2^7 + 27x_2^6 + 32x_2^5 + 21x_2^4 + x_2^3 + 17x_2^2 + 16x_2$$
$$f^{(3)}(x_3) = x_3^{10} + 28x_3^9 + 20x_3^8 + 23x_3^7 + 36x_3^6 + 36x_3^4 + 22x_3^3 + 35x_3^2 + 3x_3$$
$$f^{(4)}(x_4) = x_4^{10} + 28x_4^9 + 28x_4^8 + 25x_4^7 + 18x_4^6 + 18x_4^5 + 21x_4^4 + 28x_4^3 + 28x_4^2 + 27x_4$$

We then compute $g^{(i)}(x_i) = \mathsf{gcd}(f^{(i)}(x_i), x_i^{n_i} - \hat{x}^{i+1})$ for $i = 4, 3, 2, 1$, where $\hat{x}_5 \in B$ for $i = 4$ and $\hat{x}^{i+1}$ is a root of $g^{(i+1)}(x_i)$ for $i = 3, 2, 1$.
We will note the solutions of these polynomials to the right of each equation.

$$g^{(4)}(x_4) = \mathsf{gcd}(f^{(4)}(x_4), x_4^3 - 0) \quad = x_4 \quad \{0\}$$
$$g^{(4)}(x_4) = \mathsf{gcd}(f^{(4)}(x_4), x_4^3 - 1) \quad = x_4^3 + 36 \quad \{1, 10, 26\}$$

giving us the values for $\hat{x_4}$: $\{0, 1, 10, 26\}$. We use these roots to calculate

$$g^{(3)}(x_3) = \mathsf{gcd}(f^{(3)}(x_3), x_3^3 - 0) \quad = x_3, \quad \{0\}$$
$$g^{(3)}(x_3) = \mathsf{gcd}(f^{(3)}(x_3), x_3^3 - 1) \quad = x_3^3 - 1, \quad \{10\}$$
$$g^{(3)}(x_3) = \mathsf{gcd}(f^{(3)}(x_3), x_3^3 - 10) \quad = x_3^2 + 9x_3 + 7, \quad \{7, 33, 34\}$$
$$g^{(3)}(x_3) = \mathsf{gcd}(f^{(3)}(x_3), x_3^3 - 26) \quad = x_3^2 + 16x_3 + 34, \quad \{9, 12\}$$

giving us the values for $\hat{x}_3$: $\{0, 7, 9, 10, 12, 33, 34\}$. We use these roots to calculate

$$\begin{aligned}
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 0) && = x_2, && \{0\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 7) && = x_2 + 28, && \{9\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 9) && = x_2 + 34, && \{3\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 10) && = x_2 + 26, && \{11\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 12) && = x_2 + 30, && \{7\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 33) && = x_2 + 25, && \{12\} \\
g^{(2)}(x_2) &= \mathsf{gcd}(f^{(2)}(x_2), x_2^2 - 34) && = x_2 + 16, && \{21\}
\end{aligned}$$

giving us the values for $\hat{x}_2$: $\{0, 3, 7, 9, 11, 12, 21\}$. We use these roots to calculate

$$\begin{aligned}
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 0) && = x_1, && \{0\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 3) && = x_1^2 + 34, && \{15, 22\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 7) && = x_1 + 9, && \{28\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 9) && = x_1 + 34, && \{3\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 11) && = x_1^2 + 26, && \{14, 23\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 12) && = x_1^2 + 25, && \{7, 30\} \\
g^{(1)}(x_1) &= \mathsf{gcd}(f^{(1)}, x_1^2 - 21) && = x_1 + 13, && \{24\}
\end{aligned}$$

whose union is the set of roots $\hat{x}_1$: $\{0, 3, 7, 14, 15, 22, 23, 24, 28, 30\}$ which are the roots of of our original polynomial $f(x)$.

## 7 Example of SRA with $p = 2 \mod 3$

We provide the toy example for the case of finding solutions for $h(u) \in \mathbb{F}_{41}[x]$, which fulfils our initial condition that $p = 41 = 2 \mod 3$.

We first perform the precomputation stage for the given $p$. A value of $N$ is computed so that a large enough proportion of $N$ is smooth and allows a suitable curve to be constructed. We find that $N = 32$ is a such a value and compute the auxillary curve

$$E_{1,0}(\mathbb{F}_{41}) := \{(x, y) \in \mathbb{F}_{41} \times \mathbb{F}_{41} \ : \ y^2 = x^3 + x\} \tag{17}$$

whose rational points we will convert our points in $\mathbb{F}_{41}$ to via Icart's map [15] as in equation 18,

$$\begin{aligned} &K_0 : \mathbb{F}_{41}[u, x] \to E_{1,0}(\mathbb{F}_{41}) \\ &(u, x) \mapsto -8u^8 + 14u^6x - u^4x^2 + u^2x^3 + 7u^4 + 10 \end{aligned} \tag{18}$$

The final step of the precomputation is to compute suitable elliptic curves and successive isogenies between them such that their degree is bounded by our smoothness bound.We will only use the rational map representations of the $x$-coordinate for these maps. The following series of isogenies with their rational-map representations of the mappings from $x$-coordinate to $x$-coordinates give rise to the following system of equations

$$\begin{aligned} K_1(x) : E_{1,0}(\mathbb{F}_p) \to E_{37,0}(\mathbb{F}_p), & \qquad \frac{x_1^2 + 1}{x_1} & = x_2 \\ K_2(x) : E_{37,0}(\mathbb{F}_p) \to E_{38,11}(\mathbb{F}_p) & \qquad \frac{x_2^2 - 2x_2 + 8}{x_2 - 2} & = x_3 \\ K_3(x) : E_{38,11}(\mathbb{F}_p) \to E_{25,8}(\mathbb{F}_p) & \qquad \frac{x_3^2 + 12x_3 + 19}{x_3 + 12} & = x_4 \\ K_4(x) : E_{25,8}(\mathbb{F}_p) \to E_{34,7}(\mathbb{F}_p) & \qquad \frac{x_4^2 + 17x_4 - 10}{x_4 + 17} & = x_5 \\ K_5(x) : E_{34,7}(\mathbb{F}_p) \to E_{1,0}(\mathbb{F}_p) & \qquad \frac{x_5^2 + 16x_5 - 18}{x_5 + 16} & = x_6 \end{aligned} \tag{19}$$

After this precomputation is completed, we may begin the process of calculating the roots of $h(u)$. We seek to find the roots of the polynomial

$$h(u) = u^5 + 19u^4 + 6u^3 + 37u^2 + 38u + 30 \tag{20}$$

We first use the Icart map $K_0$ to create our polynomial $f^{(1)}(x_1)$, whose roots represent solutions of both $h(u)$ and $K_0(u, x)$ by means of taking the resultant with regards to $u$.

$$\begin{aligned} f(x) &= \mathsf{Res}_u(h(u), -8u^8 + 14u^6x - u^4x^2 + u^2x^3 + 7u^4 + 10) \\ &= 39x^{15} + x^{14} + 22x^{13} + 30x^{12} + 4x^{11} + 33x^{10} + 33x^9 \\ &\quad + 32x^8 + 9x^7 + 4x^6 + 33x^5 + 40x^4 + 12x^3 + x + 2 \end{aligned} \tag{21}$$

we note that we now have a polynomial three times the degree of our original one, but we are only interested in linear factors hence we may obtain

$$\begin{aligned} f(x) &= \gcd(x^p - x, f^{(1)}(x)) \\ f(x) &= x^4 - 15x^3 - 5x^2 + 14x + 14 \end{aligned} \tag{22}$$

which is of degree bounded by $deg(h)$. We then compute the roots of $f$ using the SRA algorithm with the maps $M = \{K_i\}_{i=1}^{5}$, the set $B = \emptyset$ and the flag ParSeq = True.

As described in the generic case of SRA, we now apply the resultant stage to obtain our $f^{(2)}(x_2), f^{(3)}(x_3), f^{(4)}(x_4), f^{(5)}(x_5)$ polynomials using the map structure we have derived from the rational maps of the isogenies as described in system 19. To do this we successively compute

$$f^{(i+1)}(x) = \mathsf{Res}_u(f^{(i)}(x_i), a_i(x_i) - x_4 + 17 \cdot x_{i+1}) \qquad \text{for } i = 1, \ldots, t-1. \tag{23}$$

This results in the series of polynomials

$$\begin{aligned} f^{(1)}(x_1) &= x^4 - 15x^3 - 5x^2 + 14x + 14 \\ f^{(2)}(x_2) &= 14x^4 + 9x^3 - 13x^2 - 5x + 11 \\ f^{(3)}(x_3) &= -14x^4 - 4x^3 - 16x^2 - 13x - 16 \\ f^{(4)}(x_4) &= -3x^4 + 7x^3 - x^2 - 11x + 11 \\ f^{(5)}(x_5) &= -2x^4 - 5x^3 + 3x^2 - 9x + 5 \end{aligned} \tag{24}$$

We may then begin the gcd stage of the algorithm. We note that at each stage we must repeatedly extract those values in the kernel as these values are not picked up by the root merging process. Our first set of roots is therefore calculated via

$$g^{(5)}(x_5) = \gcd(-2x^4 - 5x^3 + 3x^2 - 9x + 5, x_5 + 16)$$

giving us the candidate roots $\hat{x}_5$: $\{25\}$. We use this to compute the polynomial

$$\begin{aligned} g^{(4)}(x_4) &= \gcd(-3x_4^4 + 7x_4^3 - x_4^2 - 11x_4 + 11, x_4^2 + 17x_4 - 10 - (x_4 + 17) \cdot 25) \\ &= x_4^2 + 33x_4 + 16 \end{aligned}$$

These give us $\hat{x}_4$: $\{4\}$. We perform the same procedure to compute

$$\begin{aligned} g^{(3)}(x_3) &= \gcd(-14x_3^4 - 4x_3^3 - 16x_3^2 - 13x_3 - 16, x_3^2 + 12x_3 + 19 - (x_3 + 12) \cdot 4) \\ &= x_3^2 + 8x_3 + 12 \end{aligned}$$

Giving us the candidate solutions $\hat{x}_3$: $\{35, 39\}$. We perform the same procedure to compute

$$\begin{aligned} g^{(2)}(x_2) &= \gcd(14x_2^4 + 9x_2^3 - 13x_2^2 - 5x_2 + 11, x_2^2 - 2x_2 + 8 - (x_2 - 2) \cdot 35) \\ &= x_2 + 9 \\ g^{(2)}(x_2) &= \gcd(14x_2^4 + 9x_2^3 - 13x_2^2 - 5x_2 + 11, x_2^2 - 2x_2 + 8 - (x_2 - 2) \cdot 39) \\ &= x_2^2 + 4 \end{aligned}$$

Giving us the candidate solutions $\{32\}$ and $\{18, 23\}$ respectively. Finally we compute

$$\begin{aligned} g^{(1)}(x_1) &= \gcd(x_1^4 - 15x_1^3 - 5x_1^2 + 14x_1 + 14, x_1^2 + 1 - (x_1) \cdot 18)) \\ &= x_1 + 21 \\ g^{(1)}(x_1) &= \gcd(x_1^4 - 15x_1^3 - 5x_1^2 + 14x_1 + 14, x_1^2 + 1 - (x_1) \cdot 23)) \\ &= x_1^2 + 18x_1 + 1 \\ g^{(1)}(x_1) &= \gcd(x_1^4 - 15x_1^3 - 5x_1^2 + 14x_1 + 14, x_1^2 + 1 - (x_1) \cdot 32)) \\ &= x_1 + 28 \end{aligned}$$

Solving these provides us with solutions $\{20\}, \{2, 21\}, \{13\}$ for $f(x)$.

We then must convert these back into solutions for $h(u)$. We now possess $x$-coordinate solutions and may retrieve the corresponding $y$-coordinates via substitution of $x$ into the auxiliary curve and taking square roots. These lead to the solutions

$$(13, 18), (13, 23), (21, 4), (21, 37), (2, 16), (2, 25), (20, 5), (20, 36)$$

each of which we substitute into the precomputed map $L(x, y) = u^4 - 6xu^2 + 6uy - 3$ and take the gcd with $h(u)$ to obtain the list of equations whose roots are precisely those of $h(u)$ (minus 0, which may be specially checked for).

| | | |
|---|---|---|
| $(13, 18)$ | $1$ | $\{\}$ |
| $(13, 23)$ | $u + 17$ | $\{24\}$ |
| $(21, 4)$ | $1$ | $\{\}$ |
| $(21, 37)$ | $u^2 + 22u + 23$ | $\{34, 26\}$ |
| $(2, 16)$ | $1$ | $\{\}$ |
| $(2, 25)$ | $u + 1$ | $\{40\}$ |
| $(20, 5)$ | $u + 20$ | $\{21\}$ |
| $(20, 36)$ | $1$ | $\{\}$ |

The roots of $h(u)$ are therefore $\{21, 24, 26, 34, 40\}$.