Efficient Algorithms to Solve the Elliptic Curve Discrete Logarithm Problem over Finite Fields of Characteristic Two

黄, 筠茹

https://doi.org/10.15017/1500510

出版情報：九州大学, 2014, 博士（機能数理学）, 課程博士
バージョン：

Graduate School of Mathematics, Kyushu University

Ph.D. Thesis

九州大学大学院数理学府数理学専攻

博士論文

Efficient Algorithms to Solve the Elliptic Curve Discrete Logarithm Problem over Finite Fields of Characteristic Two

標数2の有限体上の楕円曲線離散対数問題の効率的な解法アルゴリズム

黃　筠茹

Yun-Ju Huang

指導教員: 高木 剛 博士

Advisor：Tsuyoshi Takagi, Ph.D.

平成 27 年 1 月

January, 2015

Acknowledgement

Abstract

In the last two decades, elliptic curves have become increasingly important. In 2009, the American National Security Agency (NSA) to advocate the use of elliptic curves for public key cryptography [Nat09], which are based on the hardness of elliptic curve discrete logarithm problem (ECDLP) or other hardness problem on elliptic curves. Elliptic curves used in practice are defined either over a prime field $\mathbb{F}_p$ or over a binary field $\mathbb{F}_{2^n}$. Like any other discrete logarithm problem, ECDLP can be solved with generic algorithms such as Baby-step Giant-step algorithm, Pollard's ρ method and their variants [Sha71, Pol75, Bre80, Pol00]. These algorithms can be parallelized very efficiently, but the parallel versions still have an exponential complexity in the size of the parameters. Better algorithms based on the *index calculus framework* have long been known for discrete logarithm problems over multiplicative groups of finite fields or hyperelliptic curves, but generic algorithms have remained the best algorithms for solving ECDLP until recently.

A key step of an index calculus algorithm for solving ECDLP is to solve the *point decomposition problem*. In 2004, Semaev introduced the *summation polynomials* (also known as Semaev's polynomials) to solve this problem. Solving Semaev's polynomials is not a trivial task in general, in particular if K is a prime field. At Eurocrypt 2012, Faugère, Perret, Petit and Renault re-analized Diem's attack [Die11] in the case $\mathbb{F}_{2^n}$ (denoted as FPPR in this work), and showed that the systems arising from the Weil descent on Semaev's polynomials are much easier to solve than generic systems [FPPR12]. Later at Asiacrypt 2012, Petit and Quisquater provided heuristic evidence that ECDLP is subexponential for that very important family of curves, and would beat generic algorithms when n is larger than about 2000 [PQ12]. In 2013, Shantz and Teske provided further experimental results using the so-called ``delta method'' with smaller factor basis to solve the FPPR system. [ST13b, FPPR12].

Even though these recent results suggest that ECDLP is weaker than previously expected

for binary curves, the attacks are still far from being practical. This is mainly due to the large memory and time required to solve the polynomial systems arising from the Weil descent in practice. In particular, the experimental results presented in [PQ12] for primes n were limited to $n = 17$. In order to validate the heuristic assumptions taken in Petit and Quisquater's analysis and to estimate the exact security level of binary elliptic curves in practice, experiments on larger parameters are definitely required.

In this paper, we introduced several variants to solve ECDLP. In our first approach, we focus on Diem's version of index calculus for ECDLP over a binary field of prime extension degree n [Die11, FPPR12, PQ12]. In that case, the Weil descent is performed on a vector space that is not a subfield of $\mathbb{F}_{2^n}$, and the resulting polynomial system cannot be re-written in terms of symmetric variables only. We introduce a different method to take advantage of symmetries even in the prime degree extension case [HPST13]. While Shantz and Teske use the same multivariate system as FPPR [ST13b, FPPR12], in this work we re-write the system with both symmetric and non-symmetric variables. The total number of variables is increased compared to [FPPR12, PQ12], but we limit this increase as much as possible thanks to an appropriate choice of the vector space V. On the other hand, the use of symmetric variables in our system allows reducing the degrees of the equations significantly. Our experimental results show that our systems can be solved faster than the original systems of [FPPR12, PQ12] as long as n is large enough.

In our second approach, we focus on the new method to calculate the Semaev's summation polynomial by breaking it down into several pieces of smaller Semaev's summation polynomial. In this case, we limited the degree of Semaev's summation polynomial as well as the degree of regularity of the new system by introducing more intermediate variables. By this new method, we can solve larger Semaev's summation polynomial to at least seven variables, while the first approach [HPST13] can only solved the Semaev's summation polynomial with

at most four variables. Our experimental results show that our systems can be solved faster than the first approach.

For a further discussion in the related topic, we also introduced more variants to solve ECDLP based on the idea to break the Semaev's summation polynomial down into several smaller pieces. We claim that the new algorithm can solve elliptic curve subset sum problem (ECSSP) as well by showing the reduction to ECSSP. The efficiency is also shown by the given experimental results.

Keywords: *elliptic curve cryptography, discrete logarithm problem, index calculus method, multivariate polynomial system, Gröbner basis*

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

In the last two decades, elliptic curves have become increasingly important. In 2009, the American National Security Agency (NSA) to advocate the use of elliptic curves for public key cryptography [Nat09], which are based on the hardness of elliptic curve discrete logarithm problem (ECDLP) or other hardness problem on elliptic curves. Given an elliptic curve E defined over a finite field K, some rational point P of E and a second point $Q \in \langle P \rangle \subset E$, elliptic curve discrete logarithm problem (ECDLP) requires finding an integer k such that $Q = [k]P$. Elliptic curves used in practice are defined either over a prime field $\mathbb{F}_p$ or over a binary field $\mathbb{F}_{2^n}$. Like any other discrete logarithm problem, ECDLP can be solved with generic algorithms such as Baby-step Giant-step algorithm, Pollard's ρ method and their variants [Sha71, Pol75, Bre80, Pol00]. These algorithms can be parallelized very efficiently, but the parallel versions still have an exponential complexity in the size of the parameters. Better algorithms based on the *index calculus framework* have long been known for discrete logarithm problems over multiplicative groups of finite fields or hyperelliptic curves, but generic algorithms have remained the best algorithms for solving ECDLP until recently.

A key step of an index calculus algorithm for solving ECDLP is to solve the *point decomposition problem*. Given a predefined *factor basis* $\mathcal{F} \subset E$ and a random point $R \in E$, this

problem asks the existence of points $P_i \in \mathcal{F}$ such that $R = \sum_i P_i$. In 2004, Semaev introduced the *summation polynomials* (also known as Semaev's polynomials) to solve this problem. The Semaev's polynomial s_r is a polynomial in r variables such that $s_r(x_1, \ldots, x_r) = 0$ if and only if there exist r points $P_i := (x_i, y_i) \in E$ such that $\sum_{i=1}^{r} P_i = O$. For a factor basis $\mathcal{F}_V := \{(x, y) | x \in V\}$ where $V \subset K$, the point decomposition problem now amounts to computing all x_i satisfying $s_{r+1}(x_1, \cdots, x_r, x(R)) = 0$ for the x-coordinate $x(R)$ of the given point R. Semaev's polynomials therefore reduce the decomposition problem on the elliptic curve E to algebraic problem over the base field K. Solving Semaev's polynomials is not a trivial task in general, in particular if K is a prime field. For extension fields $K = \mathbb{F}_{q^n}$, Gaudry and Diem [Die06, Gau09] independently proposed to define V as the subfield $\mathbb{F}_q$ and to apply a *Weil descent* to further reduce the resolution of Semaev's polynomials to the resolution of a polynomial system of equations over $\mathbb{F}_q$. Diem generalized these ideas by defining V as a vector subspace of $\mathbb{F}_{q^n}$ [Die11]. Using generic complexity bounds on the resolution of polynomial systems, these authors provided attacks that can beat generic algorithms and can even have subexponential complexity for specific families of curves [Die06]. At Eurocrypt 2012, Faugère, Perret, Petit and Renault re-analized Diem's attack [Die11] in the case $\mathbb{F}_{2^n}$ (denoted as FPPR in this work), and showed that the systems arising from the Weil descent on Semaev's polynomials are much easier to solve than generic systems [FPPR12]. Later at Asiacrypt 2012, Petit and Quisquater provided heuristic evidence that ECDLP is subexponential for that very important family of curves, and would beat generic algorithms when n is larger than about 2000 [PQ12]. In 2013, Shantz and Teske provided further experimental results using the so-called ``delta method'' with smaller factor basis to solve the FPPR system. [ST13b, FPPR12].

Even though these recent results suggest that ECDLP is weaker than previously expected for binary curves, the attacks are still far from being practical. This is mainly due to the large memory and time required to solve the polynomial systems arising from the Weil descent in

practice. In particular, the experimental results presented in [PQ12] for primes n were limited to $n = 17$. In order to validate the heuristic assumptions taken in Petit and Quisquater's analysis and to estimate the exact security level of binary elliptic curves in practice, experiments on larger parameters are definitely required.

Hybrid methods (involving a trade-off between exhaustive search and polynomial system solving) have been proposed to practically improve the resolution of the polynomial systems [JV11a]. More importantly, the special structure of these systems can be exploited. When n is composite and the Weil descent is performed on an intermediary subfield, Gaudry already showed in [Gau09] how the symmetry of Semaev's polynomials can be exploited to accelerate the resolution of the polynomial system in practice. In that case, the whole system can be re-written with new variables corresponding to the fundamental symmetric polynomials, therefore reducing the degrees of the equations and improving their resolution. In the particular cases of twisted Edward curves and twisted Jacobi curves, Faugère *et al.* also exploited additional symmetry coming from the existence of a rational 2-torsion point to further reduce the degrees of the equations [FGHR12].

In this paper, we introduced several variants to solve ECDLP. In our first approach, we focus on Diem's version of index calculus for ECDLP over a binary field of prime extension degree n [Die11, FPPR12, PQ12]. In that case, the Weil descent is performed on a vector space that is not a subfield of $\mathbb{F}_{2^n}$, and the resulting polynomial system cannot be re-written in terms of symmetric variables only. We introduce a different method to take advantage of symmetries even in the prime degree extension case. While Shantz and Teske use the same multivariate system as FPPR [ST13b, FPPR12], in this work we re-write the system with both symmetric and non-symmetric variables. The total number of variables is increased compared to [FPPR12, PQ12], but we limit this increase as much as possible thanks to an appropriate choice of the vector space V. On the other hand, the use of symmetric variables in our system allows reducing the

degrees of the equations significantly. Our experimental results show that our systems can be solved faster than the original systems of [FPPR12, PQ12] as long as n is large enough.

In our second approach, we focus on the new method to calculate the Semaev's summation polynomial by breaking it down into several pieces of smaller Semaev's summation polynomial. In this case, we limited the degree of Semaev's summation polynomial as well as the degree of regularity of the new system by introducing more intermediate variables. By this new method, we can solve larger Semaev's summation polynomial to at least seven variables, while the first approach can only solved the Semaev's summation polynomial with at most four variables. Our experimental results show that our systems can be solved faster than the first approach, especially in the case that the number of variables similar to the number of extension degree .

For a further discussion in the related topic, we also introduced more variants to solve ECDLP based on the idea to break the Semaev's summation polynomial down into several smaller pieces. We claim that the new algorithm can solve elliptic curve subset sum problem (ECSSP) as well by showing the reduction to ECSSP. The efficiency is also shown by the given experimental results.

1.1 Thesis Organization

The rest of this thesis is organized as follows. In the next chaper, Chapter 2, we will define the common notation used frequently in this thesis. In Chapter 3, we will go through the required preliminary, including the basic introdution of the elliptic curve, the basic idea of the index calculus method as well as the Semaev's summation polynomial, the FPPR method which we are trying to improve in our first approach in this thesis, and the F_4 algorithm for computing Gröwner basis. In Chapter 4, we introduce our first approach to improve the FPPR method using the symmetry property of the Semaev's summation polynomial and the specific vector

space. We also show the experimental evidence that our approach is more efficient than FPPR method. In Chapter 5, we further improve the efficiency of solving Semaev's summation polynomial by rewrite the Semaev's summation polynomial into several small pieces with intermediate unknown variables. We also show the experimental evidence that our approach is more efficient than the first approach. In Chapter 6, we give further discussion by proposing the new algorithm based on our second approach and claim it can solve the ECSSP as well. Last, we conclude this thesis and point out some future work in Chapter 7.

Chapter 2

Notation

In this work, we are interested in solving the elliptic curve discrete logarithm problem on a curve E defined over a finite field $\mathbb{F}_{2^n}$, where n is a prime number. We denote by $E_{\alpha,\beta}$ the elliptic curve over $\mathbb{F}_{2^n}$ defined by the equation $y^2 + xy = x^3 + \alpha x^2 + \beta$. For a given point $P \in E$, we use $x(P)$ and $y(P)$ to indicate the x-coordinate and y-coordinate of P respectively. From now on, we use the specific symbols P, Q and k for the parameters and solution of the elliptic curve discrete logarithm problem (ECDLP): $P \in E$, $Q \in \langle P \rangle$, and k is the smallest non-negative integer such that $Q = [k]P$. Without loss of generality, we assume that the order of $\langle P \rangle$ is prime here. We identify the field $\mathbb{F}_{2^n}$ as $\mathbb{F}_2[\omega]/h(\omega)$, where h is an irreducible polynomial of degree n. Any element $e \in \mathbb{F}_{2^n}$ can then be represented as $poly(e) := c_0 + c_1\omega + ... + c_{n-1}\omega^{n-1}$ where $c_i \in \mathbb{F}_2$. For any set S, we use the symbol $\#S$ to mean the order of S. We use s_m to denote the Semaev's summation polynomial with m variables. We denote the degree of regularity as D_{reg}.

Chapter 3

Preliminary

Before going through the detailed work of this paper, in order to have deeper insight of our research, sufficient background study about the elliptic curve discrete logarithm problem we are working on are necessary. In this chapter, we will introduce the preliminary knowledge required for our thesis. In the first Section 3.1, we will introduce the basic definitions and operations of elliptic curve and the discrete logarithm problem on the curve. And in Section 3.2, we will introduce the generic index calculus method to solve the elliptic curve discrete logarithm problem. In Section 3.3, we will introduce the FPPR method which is proposed by Faugère, Perret, Petit and Renault on Eurocrypt 2012 [FPPR12] and proved to be sub-exponential by Petit and Quisquater [PQ12]. In Section 3.4, we introduce some extra knowledge about the Gröbner basis.

3.1 Elliptic Curve Discrete Logarithm Problem

Public-key cryptography, also known as asymmetric cryptography, is one of the most important part of cryptography nowadays. It allows two person encrypt and decrypt a message with a pair of keys, a public key and a private key, which public key is not a secret. In such manner,

a secret channel to share the symmetric key is not required anymore. Not only an encryption scheme, many well-known application were therefore developed based on the public-key system, for example, the key exchange scheme and the signature scheme. Before the development of elliptic curve cryptography, denoted as ECC, the most famous public-key scheme was RSA. However, the large key size becomes a problem of RSA scheme. An elliptic curve cryptography was therefore introduced to meet such requirement. Compared to the RSA scheme with the same security level, the key size of a elliptic-curve-based RSA scheme is much smaller. For example, for the 112-bit security level, it requires 2048-bit key size for RSA while it takes rough 224-bit key size for an ECC system. The development of elliptic curve cryptography not only provided an option for the public-key cryptography and the pseudo-number generator, but also makes a new application using the pairing system embedded on the elliptic curve, for example, the identity-based cryptography. Nowadays, the elliptic curve cryptography is one of the most important research area in cryptography. We will simply introduced the definition of the elliptic curve used in the cryptography and the group and arithmetic operation on the curve in this section.

In cryptography, an elliptic curve is usually defined as an smooth plane curve over a finite field instead of the real numbers in the algebraic mathematics for the calculation purpose. In such manner, the abelian group defined on the curve are finite as well. Although there are many families of elliptic curves, for the generic Galois field F_q that the characteristic is not 2 or 3, the most popular equation for a curve is the Weierstrass equation:

$$E_{\alpha,\beta} : y^2 = x^3 + \alpha x + \beta,$$

where α and $\beta \in F_q$ are the parameters of the curve such that $-16(4\alpha^3 + 27\beta^2) \neq 0$.

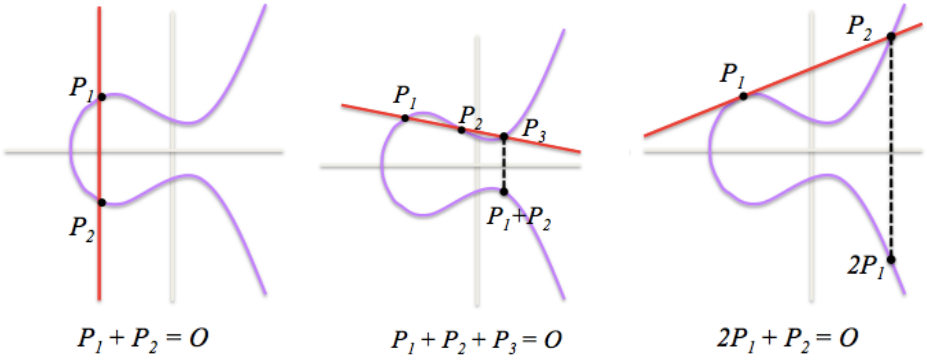
For curve with characteristic 2 or 3, to avoid the singularity, the curve is commonly defined by the corresponding curve equation respectively:

$$E_{\alpha,\beta} : y^2 + xy = x^3 + \alpha x^2 + \beta,$$

and

$$E_{\alpha,\beta} : y^2 = 4x^3 + \alpha x^2 + \beta,$$

where α and $\beta \in F_q$ are the parameters such that the curve is non-singular. In this paper, we are focus on the elliptic curve with characteristic 2.



(a) Inverse (b) Addition (c) Double multiplication

Figure 3.1: Illustration of the arithmetic operation on elliptic curve

The abelian group defined on the curve are the point set of those points $P(x, y)$ where $x, y \in F_q$ satisfied the curve equation along with an additive identity point O, which is also known as the point of infinity. If two points P_1, P_2 on the curve have the same x-coordinate, as shown in Figure 3.1(a), we said that $P_2 = -P_1$. Suppose we have three points P_1, P_2, P_3 line up in the line, as shown in Figure 3.1(b), we defined that $P_1+P_2+P_3 = O$ and therefore $P_1+P_2 = -P_3$. For a point P_1, $2P_1$ is defined by the tangent line of P_1, which meets another point P_2 on the curve, as shown in Figure 3.1(c). Therefore, $2P_1 + P_2 = O$ and $2P_1 = -P_2$. From the definition above, we can give the formula of $-P_1$, $P_1 + P_2$ and $2P_1$ respectively corresponding to the corresponded curve equation.

As long as we define the arithmetic operation on the curve, for any point P on the elliptic curve, we can define the scalar operation kP, where k is an integer. Figure 3.2 shows the exam-

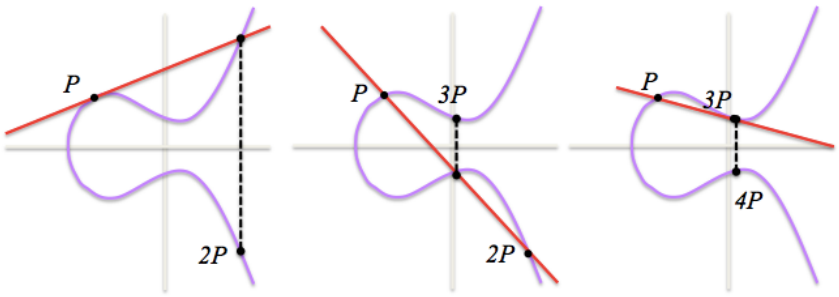


Figure 3.2: Illustration of the scalar multiplication on elliptic curve

ple of the scalar multiplication. As shown in the figure, we can figure out that the point kP generated by P for different k is irregular and non-predictive. This fact induces the famous hard problem on the elliptic curve, the elliptic curve discrete logarithm problem (ECDLP), which is the foundation of the security of the elliptic curve cryptography.

Let E be a curve defined over a base field F, and P is a point on the curve. Let Q is the point generated by P, the elliptic curve discrete logarithm problem is to find out the smallest non-negative integer $[k]$ such that $Q = [k]P$. There are several famous algorithm to solved ECDLP, for example, Baby-step Giant-step algorithm, Pollard's ρ method and their variants [Sha71, Pol75, Bre80, Pol00]. These algorithms are efficiently and easily parallelized, however, they still have an exponential complexity in the size of the parameters. Better algorithms based on the *index calculus framework* have long been known for discrete logarithm problems over multiplicative groups of finite fields or hyperelliptic curves, but generic algorithms have remained the best algorithms for solving ECDLP until recently. We are going to introduce the index calculus method in the next section.

3.2 Index Calculus Method

Index calculus method had been well-known in solving the generic discrete logarithm problem (DLP) over generic Galois field. There were also many variants of index calculus method such as number field sieve (NFS) or function field sieve. However, these well-known algorithm do not work with the ECDLP. Let $E_{\alpha,\beta}$ be a curve defined over F_q, a point $P \in E_{\alpha,\beta}$, and a point $Q \in \langle P \rangle$. Our target is to find out the smallest non-negative integer k such that $Q = [k]P$. Like the other algorithms in the family of the index calculus method, the most important part in the index calculus method solving ECDLP is how to find out the equivalency between the combination of a factor base and the combination of P and Q as soon as possible and as much as possible. In this section, we make more detailed introduction of index calculus to solve ECDLP as well as how to decompose a point $R = aP + bQ$ with random integers a and b on the elliptic curve $E_{\alpha,\beta}$ into m points belongs to the factor base F_v using the Semaev's summation polynomial, which is the most important and difficult part of the algorithm.

3.2.1 Generic Index Calculus Method

First we give a rough shape of the index calculus method. The pseudo code of the index calculus algorithm is in Algorithm 3.1.

The inputs are the elliptic curve $E_{\alpha,\beta}$, the point $P \in E$ and another point $Q \in \langle P \rangle$. For the first step in Algorithm 3.1, we select a subset of the point on the curve to be the factor base F_v.

The sieving step between line 3 to line 7 is the most important part of this algorithm. It dominates the efficiency of the computation of the algorithm. In this step, first we computed two random integers a and b in the range $[0, \#\langle P \rangle)$ where $ab \neq 0$, then computed $R = aP + bQ$. Later, we try to decompose R into m elements in the factor base F_v if R is $F_v^{(m)}$-smooth. That is, we want to find $P'_j \in F_v$, $1 \leq j \leq m$ such that $P'_1 + P'_2 + ... + P'_m + R = O$. The function **Decompose** will return all the possible decomposition solutions of R with respect to F_v.

Algorithm 3.1 Index Calculus of ECDLP [Sem04]

Input: elliptic curve $E_{\alpha,\beta}$, point $P \in E_{\alpha,\beta}$, point $Q \in \langle P \rangle$

1 $F_v \longleftarrow$ a subset of $E_{\alpha,\beta}$

2 $M \longleftarrow$ matrix with $\#F_v + 2$ columns

3 **while** $Rank(M) < \#F_v + 1$ **do**

4 $\quad R \longleftarrow aP + bQ$, a, b are random integers in $(0, \#P)$

5 $\quad sol(Set)_m \longleftarrow$ Decompose(R, F_v)

6 $\quad M \longleftarrow$ AddRelationToMatrix$(sol(Set)_m)$

7 **end**

8 $M\_ \longleftarrow$ ReducedRowEchelonForm(M)

9 $a', b' \longleftarrow$ last two column entries of last row

10 $k \longleftarrow -a'/b'$

Output: k, where $Q = [k]P$

After we get the decomposition of R with respect to factor base F_v, we may put the row meaning the decomposition into the matrix M using **AddRelationToMatrix** function. What this function do is illustrated in the Matrix in Figure 3.3. We use the first column to present the coefficient of P_1, the second column as coefficient of P_2, and so on. The last two columns present the coefficients of P and Q respectively. Each row present a relation $P'_1 + P'_2 + ... + P'_m + R = O$, that is $P'_1 + P'_2 + ... + P'_m + aP + bQ = O$. We put the coefficients into the corresponding entries and left other entries 0.

If the rank of M is $\#F_v + 1$, then we can break the loop of sieving. The **ReducedRowEchelonForm** function in line 8 makes M into a reduced row echelon form $M\_$ as shown in the lower matrix in Figure 3.3. The last two columns of the last row in $M\_$ would be $a'P + b'Q = O$. This means that $k = -a'/b'$.

There are so many ways to implement the **Decompose** function. The simplest way is doing the exhaustive search with F_v and find out all the solutions. But it is not a practicable idea.

$$
\begin{array}{c}
\begin{array}{cccccc} P_1 & P_2 & \dots & P_{\#F_v} & P & Q \end{array} \\
\begin{pmatrix}
1 & 0 & & 1 & a_1 & b_1 \\
0 & 1 & & 0 & a_2 & b_2 \\
\vdots & & \ddots & \vdots & & \vdots \\
\\
1 & 1 & \dots & 1 & a_j & b_j
\end{pmatrix} \\
\Downarrow \quad \text{reduced row echelon form} \\
\begin{pmatrix}
1 & 0 & & 0 & & \\
0 & 1 & & 0 & & \\
\vdots & & \ddots & \vdots & & \\
\\
0 & 0 & \dots & 1 & & \\
0 & 0 & & 0 & a' & b'
\end{pmatrix}
\end{array}
$$

Figure 3.3: Example of Matrix M and $M\_$ in Algorithm 3.1

In the next subsection, we will show how to decompose a point on the curve using Semaev's summation polynomial.

3.2.2 Semaev's summation polynomial

No matter how to define an elliptic curve $E_{\alpha,\beta}$, the fact that the operation over $E_{\alpha,\beta}$ is much slower than the operation over its base field will not change. If we can represent the decomposition of a point on $E_{\alpha,\beta}$ as the computation consisting of only operations of a finite field, it will help saving a lot of efforts in computing definitely. Semaev's proposed Semaev's summation polynomial to fulfill such property [Sem04].

Definition 1 *The m-th Semaev's summation polynomial s_m for $E_{\alpha,\beta}$ with characteristic 2 is defined as follows:*

$s_2 = x_1 + x_2$,

$s_3 = (x_1x_2 + x_1x_3 + x_2x_3)^2 + x_1x_2x_3 + \beta$,

for $m \geq 4$,

$s_m = Res_X(s_{j+1}(x_1, ..., x_j, X), s_{m-j+1}(x_{j+1}, ..., x_m, X))$,

$2 \leq j \leq m-2$.

Semaev's summation polynomials have the following property:

Property 1 *We said that a polynomial s_m with m inputs is a Semaev's summation polynomial of $E_{\alpha,\beta}$ if*

$$s_m(x_1, x_2, ..., x_m) = 0 \text{ iff } P'_1 + P'_2 + ... + P'_m = O,$$

where $P'_j \in E$, $x(P'_j) = x_j$.

According to Property 1, to find the decomposition of a given point $R = aP + bQ$, such that $P'_1 + P'_2 + ... + R = O$, we just need to find the solution of $s_{m+1}(x_1, x_2, ..., x_m, x(R)) = 0$ in F_{2^n} at first and find the corresponding points $P'_j \in F_v$ later.

We mention the computation of the Semaev's summation polynomials. The s_2 and s_3 are decided by Definition 1. For $m \geq 4$, we use the recursive definition. For example,

$$\begin{aligned} s_4 &= Res_X(S_3(x_1, x_2, X), S_3(X, x_3, x_4)) \\ &= (x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4)^4 + \\ &\quad (x_1x_2x_3x_4)^2(x_1 + x_2 + x_3 + x_4)^2 + \\ &\quad x_1x_2x_3x_4((x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4)^2 + \\ &\quad \beta(x_1 + x_2 + x_3 + x_4)^2) + \\ &\quad \beta(x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4)^2 + \\ &\quad \beta^2(x_1 + x_2 + x_3 + x_4)^4. \end{aligned}$$

The degree of each variable in s_m is 2^{m-2}. For the detail of the construction of Semaev's summation polynomial, readers can read the paper of Semaev [Sem04].

According to Definition 1, for $m \geq 4$, there are several ways to divide m into two parts $[1, ..., j], [j + 1, ..., m]$. For different j, we may have different s_{j+1} and s_{m-j+1} for the resultant calculation. Nevertheless, the s_m would always be the same for fixed $E_{\alpha,\beta}$ despite of the inputs of the resultant are different. This fact can be easily proved by Property 1. However, different j of s_m costs different computing time. The best choice of j is $\lceil m/2 \rceil$, due to the size of Sylvester matrix used to calculate the resultant of two polynomials. Considering the recursively construction of the Semaev's summation polynomial, the degree and the number of monomials of the polynomials grows rapidly. Using the fact that the Semaev's polynomial is symmetric helps to decrease the complexity in computing Semaev's summation polynomials. Antoine Joux and Vanessa Vitse proposed such technique in calculating s_m [JV11a]. Since this is not the main topic of this work, we left the rest details for readers.

The **Decompose** function with Semaev's summation polynomial is in Algorithm 3.2. Notice that in elliptic curve $E_{\alpha,\beta}$, there are at most two points shared the same x-coordinate. Thus, there are 2^m possible solution in $E_{\alpha,\beta}$ corresponding to one solution of $s_{m+1}(x_1, x_2, ..., x_m, x(R)) = 0$. We have to check this in line 5. The naive method to perform the computation in line 4 is

Algorithm 3.2 **Decompose** function with s_{m+1}

Input: $R = aP + bQ$, factor base F_v

$Set_m \longleftarrow \{F'_v \mid F'_v \subset F_v, \#F'_v = m\}$

$sol(Set)_m \longleftarrow \{\}$

for $e = \{P'_1, P'_2, .., P'_m\} \in Set_m$ **do**

  if $s_{m+1}(x(P'_1), x(P'_2), ..., x(P'_m), x(R)) = 0$ **then**

    if $P'_1 + P'_2 + ... + P'_m + R = O$ **then**

      $sol(Set)_m \longleftarrow sol(Set)_m \; join \; e$

    end

  end

end

Output: $sol(Set)_m$ conatains the decomposition elements of R w.r.t. F_v

is exhaustive search. There is still no systematic method to find the solution of s_{m+1} until the FPPR method using Gröbner basis was proposed in Eurocrypt 2012 [FPPR12]. We are going to give a simple introduction of the method in the following section.

3.3 FPPR method

In Eurocrypt 2012, Faugère, Perret, Petit and Renault gave a new vision in solving Semaev's summation polynomials over elliptic curves $E_{\alpha,\beta}$ specifically with characteristic 2 [FPPR12]. The main idea is that transforming Semaev's summation polynomial s_{m+1} to a binary multivariable polynomial system F.

The transformation is doing as followed. First, we can easily transformed an element $x(P_i)$ in F_{2^n}, where P_i is an arbitrary point in F_v, into a binary polynomial form :

$$poly(x(P_i)) = \bar{c}_0 + \bar{c}_1\omega + ... + \bar{c}_{n-1}\omega^{n'-1}, \; \bar{c}_l \; is \; known.$$

Equivalently, we can rewrite all the variables x_j in $s_{m+1} \mid_{x_{m+1}=x(R)}$ with their binary polynomial notation $poly(x_j)$ as followed:

$$x_j = c_{j,0} + c_{j,1}\omega + ... + c_{j,n'-1}\omega^{n'-1},\ c_{j,l}\ are\ variables.$$
$$x_{m+1} = r_0 + r_1\omega + ... + r_{n-1}\omega^{n-1},\ r_l\ is\ known,$$

where $1 \leq j \leq m$.

Describing s_{m+1} with variables $c_{j,l}$ by the above equation between x_j and $c_{j,l}$, we have that

$$s_{m+1} = f_0 + f_1\omega + ... + f_{n-1}\omega^{n-1},$$

where $f_0, f_1, ..., f_{n-1}$ are binary polynomials, $1 \leq j \leq m$, $0 \leq l \leq n' - 1$.

Thus, $s_{m+1} \mid_{x_{m+1}=x(R)} = 0$ if and only if each binary coefficient polynomial f_l with respect to ω is equal to 0. Now we have a binary multivariable polynomial system

$$F : \begin{cases} f_0(c_{1,0}, ..., c_{j,l}, ..., c_{m,n'-1}) = 0, \\ f_1(c_{1,0}, ..., c_{j,l}, ..., c_{m,n'-1}) = 0, \\ \qquad \vdots \\ f_m(c_{1,0}, ..., c_{j,l}, ..., c_{m,n'-1}) = 0, \end{cases}$$

where $1 \leq j \leq m, 0 \leq l \leq n' - 1$.

To solve the Semaev's summation polynomial s_{m+1} is equivalent to solve the binary multivariable polynomial system F now.

The **Decompose** function using binary multivariable polynomial system is described in Algorithm **??**. First, we substitute $x(R)$ into the variable x_{m+1} in s_{m+1}, which is the known information. The **TransFromSemaevToBinaryWithSym** function transform $s_{m+1} \mid_{x_{m+1}=x(R)}$ to F as mentioned above.

To solve the system F, we compute Gröbner basis for F with respect to lexicographic order by an algorithm such as F_4 [Fau99] or F_5 [Fau02]. In lexicographic order, there is always a univariate polynomial $gb_t(c_{j,l})$ in $GB(F)$ for some t. From the roots of $gb_t(c_{j,l})$, we obtain all

Algorithm 3.3 **Decompose** function with binary multivariable polynomial system [FPPR12]

Input: $R = aP + bQ$, factor base F_v

1 $F \longleftarrow$ TransFromSemaevToBinary($s_{m+1} \mid_{x_{m+1}=x(R)}$)

2 $GB(F) \longleftarrow$ GroebnerBasis($F, \prec_{lex}$)

3 $sol(F) \longleftarrow$ GetSolutionFromGroebnerBasis($GB(F)$)

4 $sol(Set)_m \longleftarrow \{\}$

5 **for** $e = \{P'_1, P'_2, .., P'_m\} \in sol(F)$ **do**

6 | **if** $P'_1 + P'_2 + ... + P'_m + R = O$ **then**

7 | | $sol(Set)_m \longleftarrow sol(Set)_m \; join \; e$

8 | **end**

9 **end**

Output: $sol(Set)_m$ contains the decomposition elements of R w.r.t. F_v

the solutions of F. However, since it is much efficient for a Gröbner basis solver running in the graded-reversed lexicographic order than in lexicographic order, user can solve the system F in graded-reverse lexicographic order first and change the computed Gröbner basis $GB(F)$ to lexicographic order by FGLM algorithm [FGLM93] later. After getting the solutions of F, we will find the corresponding solution over $E_{\alpha,\beta}$. Again, we have to check the statement $P'_1 + P'_2 + ... + P'_m + R = O$ in line 6 due to the same reason in Section 3.2.2. In the next section, we will mention more detail about the Gröbner basis.

3.4 Groebner Basis

Solving a multivariate polynomial system is always an improtant topic, no matter in mathematics, or in the cryptography, for example, the algebraic attack for the cryptanalysis. Suppose we have a set of variables $X = \{x_1, x_2, ..., x_n\}$ and a multivariate polynomial system

$F(X) = \{f_1(X), f_2(X), ..., f_m(X)\}$. To solve a system F is equivalent to solve $F(X)$ to get X, or any $x_i \in X$. Indeed, if we get the value of some x_i, then we can substitute x_i with its value and get a smaller multivariate polynomial system. Thus, we can solve the systems recursively and get X. Now, the problem is that how can we get the solution of some specific x_i.

The F_4 algorithm we are going to use in this thesis is one of the famous algorithms in solving a multivariate polynomials system. Basically, the F_4 algorithm is an algorithm calculating Gröbner basis of an ideal. In the multivariate polynomial system F, consider the ideal I spanned by the polynomials F and its Gröbner basis G of I. Due to the property of Groebner basis, there is usually a univariate polynomial in G, which is easier to solve than multivariate polynomial equations. Thus, we can solve $F(X)$ and get each x_i by calculating its Groebner basis repeatedly.

In the text of the following subsections, we will introduce the the basic definition and theorem of polynomial ring, Gröbner basis as well as the F_4 algorithm.

First we define some symbols. Let $X = \{x_1, x_2, ..., x_n\}$, R = $F_{p^k}[X]$ be a multivariate polynomial ring, and $F = \{f_1, f_2, ..., f_m\}$ be the set of polynomials in R. Ideal $I \subset R = \langle f_1, f_2, ..., f_m \rangle$ is an ideal finitely generated by F.

3.4.1 Order

It is easy to give an order between monomials in a univariate polynomial ring, for example, $x^7 > x^5 > 1$. However, the problem is how to determine the order in a multivariate polynomial ring, for example, $xy > z$ or $xy < z$ for $xy, z \in$ GF$_2[x, y, z]$.

To give an order in a multivariate polynomial ring, we need to define an order among x_1 to x_n first. Without loss of generality, we define $x_1 > x_2 > ... > x_n$. After having the order among variables, now we can construct the order among monomials.

An order is a relation satisfying:

- $x = x$
- $x > y$ then $y < x$
- $x > y$ and $y > z$ then $x > z$
- $x > y$ then $c^*x > c^*y$, $c \neq 0$

We now can define order among monomials in R. There are several orders in a multivariate polynomial ring. We shows some common orders here. Note that we denote the head term of a polynomial f as HT(f).

- **Lexical order** (lex)

  This order is what we usually do with words in dictionary. For a monomial m_1, m_2, if the degree of x_1 in m_1 is more than m_2, then $m_1 > m_2$. If the degree of x_1 in m_1 equal to m_2, then we compare the degree of x_2, and so on.

  For example, ${x_1}^2 > x_1{x_3}^7 > {x_2}^3{x_3}^5 > {x_2}^2{x_3}^6 > {x_2}^2{x_3}^4$

- **Graded Lexical order** (grlex)

  This order is a little different with lexical order. First we compare the total degree of m_1 and m_2. If deg(m_1) > deg(m_2), $m_1 > m_2$. If deg(m_1) = deg(m_2) then we check the order of m_1 and m_2 using lexical order.

  For example, $x_1{x_3}^7 > {x_2}^3{x_3}^5 > {x_2}^2{x_3}^6 > {x_2}^2{x_3}^4 > {x_1}^2$

- **Graded Reverse Lexical order** (grvlex)

  Like graded lexical order, we compare the total degree of m_1 and m_2. If deg(m_1) > deg(m_2), $m_1 > m_2$. If deg(m_1) = deg(m_2) then we check the degree of x_n in m_1 and m_2. The higher the degree of x_n is, the less it is. If the degree of x_n in m_1 equal to m_2, then we compare the degree of x_{n-1}, and so on.

  For example, ${x_2}^3{x_3}^5 > {x_2}^2{x_3}^6 > x_1{x_3}^7 > {x_2}^2{x_3}^4 > {x_1}^2$

Notice that we do not care about the coefficient of monomials, that is, two terms are the same if they have the same monomials, no matter what the coefficients are. For example, $3{x_1}^2$ have the same order with ${x_1}^2$.

After defining order in monomial, then we can easily tell the order among polynomials. Let f_1 and f_2 be two polynomials in R. First we make the monomials in f_1 and f_2 in order and compare the head term of these two polynomials. If HT(f_1) $>$ HT(f_2), then $f_1 > f_2$. If HT(f_1) equal to HT(f_2), then we check the next monomials. For example, ${x_2}^3{x_3}^5+1 > {x_2}^2{x_3}^6+{x_3}^2+1 > {x_2}^2{x_3}^6+x_3+1$ in graded reverse lex order.

3.4.2 Groebner basis

Definition 2 *Groebner basis*

Let G be a basis of I = <f_1, f_2, ..., f_m>, then G is a Groebner Basis if and only if $\forall f \in I$, $\exists\, g \in G$, such that HT(g) | HT(f)

For example, <8, 6> = <2> in Z. $\{8, 6\}$ is not a Groebner basis, while $\{2\}$ is. Another example in GF$_2$[x, y], <$x^2+y, x+y$> and <$x+y$,y^2+y> represent the same ideal, however $\{x^2+y, x+y\}$ is not a Groebner basis while $\{x+y, y^2+y\}$ is.

In Figure 3.4, every circle represents a HT(f), $f \in R$. The arrow from circle HT(f_i) to circle HT(f_j) represents HT(f_i) | HT(f_j). Let $HT(f_i) <_{HT} HT(f_j)$ iff HT(f_i) | HT(f_j). Figure 3.4 shows the partial order $<_{HT}$ defined on the set of head terms in ideal I. We can say that, by definition of Groebner Basis, a Groebner Basis G must contains all those minimal polynomials g_i with HT(g_i) being the leftmost node.

Next, to calculate Groebner Basis, we have to calculate SPolynomials first.

Definition 3 *SPolynomial*

Let $f_i, f_j \in I$, $c_i = \frac{\text{lcm}(HT(f_i,f_j))}{HT(f_i)}$, $c_j = \frac{\text{lcm}(HT(f_i,f_j))}{HT(f_j)}$, then $c_i f_i - c_j f_j$ is the SPolynomial of (f_i,

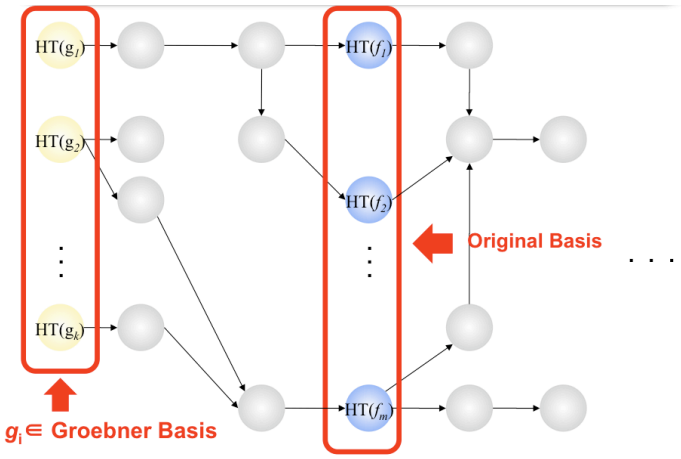


Figure 3.4: Simple sketch of Groebner Basis

f_j), denoted as SPoly(f_i, f_j). The degree of SPoly(f_i, f_j) is the degree of $\mathrm{lcm}(f_i, f_j)$*, denoted as deg(SPoly(f_i, f_j)).*

Definition 4 *Pair*

Let f_i, f_j, c_i, c_j be the same defintion in the Definition 3, then we said that <$\mathrm{lcm}(f_i, f_j)$, c_i, f_i, c_j, *f_j> is a pair which gives the information of SPoly(f_i, f_j). Let p be a pair of (f_i, f_j), the contents of p is denoted in order as* $p.lcm$, $p.f_imult$, $p.f_i$, $p.f_jmult$, $p.f_j$.

Definition 5 *pre-SPolynomial*

Let f_i, f_j, c_i, c_j be the same defintion in the Definition 3, c_if_i, c_jf_j are the pre-SPolynomials of (f_i, f_j), denoted as pre-SPoly$_i$(f_i, f_j) and pre-SPoly$_j$(f_i, f_j) respectively.

By definition of SPolynomial, SPoly(f_i, f_j) eliminates the head term of pre-SPoly$_i$ and pre-SPoly$_j$, which is divisible by head term of f_i and f_j respectively. Thus, we will find out new head terms that may not be divided by the head terms in G.

For example, let I = <$x^2y + 1, xy^2 + 1$>, SPoly($x^2y + 1, xy^2 + 1$) = $-x + y$, it is a new head term not divided by x^2y and xy^2 .

The next question is, how we determine whether a basis G is Groebner basis or not. Buchberger then gave a theorem as followed :

Theorem 1 *Buchberger Theorem*

Let G be a basis of I, then G is a Groebner Basis of I if and only if $\forall f_i, f_j \in I,\ SPoly(f_i, f_j)$ can be reduced to zero by G.

This theorem states that if all the SPolynomials of G can be reduced to zero by G, then G is Groebner Basis, and vice versa. One direction of the proof is straightforward. If there exists a SPoly(f_i, f_j), denoted as $spoly$, that cannot be reduced to zero by G, that is, $spoly$ is reduced to p by G, $p \neq 0$, then $\forall g \in G$, HT(g) $\nmid$ HT(p). This conflicts with the definition of Groebner Basis. In other hand, if G satisfied the statement, then every arithmetic combination of g_i in G will have a head term that is a multiple of g_i (Standard Representation). Thus, it is a Groebner Basis. The detail of proof can be found in [BKW93, 1993].

Now we have a systematic method to examine whether a basis G is a Groebner Basis or not. Buchberger also stated some special cases for SPoly(f_i, f_j) that do not need to be checked.

Criteria 1 *Buchberger First Criteria*

Let $f_i, f_j \in I$, if gcd$(f_i, f_j) = 1$*, then $SPoly(f_i, f_j)$ can be reduced to zero by $\{f_i, f_j\}$.*

Criteria 2 *Buchberger Second Criteria*

Let $f_i, f_j, f_k \in I$, and lcm$(f_i, f_j)|$ lcm(f_j, f_k)*,* lcm$(f_i, f_k)|$ lcm(f_j, f_k)*, then if both SPoly(f_i, f_j) and SPoly(f_i, f_k) can be reduced to zero by G, so is SPoly(f_j, f_k).*

The Buchberger First Criteria stated that if we have f_i, f_j with gcd$(f_i, f_j) = 1$, then we do not need to check SPoly(f_i, f_j), since it must be reduced to zero by $\{f_i, f_j\} \subseteq G$, thus must be reduced by G. The Buchberger Second Criteria stated that if we have f_i, f_j, f_k satisfying the condition specified, then we only need to check SPoly(f_i, f_j) and SPoly(f_i, f_k), since if both the

two SPolynomial can be reduced to zero by G, then SPoly(f_j, f_k) can be reduced to zero by G, too. The detail of proof also can be found in [BKW93, 1993].

The following three section shows three different algorithms finding out Groebner Basis of <F>, where F is a set of polynomials that we want to solve.

For convenience, when we said about divisibility of polynomials in this thesis, it means the divisibility of their head terms. For example, when we said $f \mid g$, it means $HT(f) \mid HT(g)$.

3.4.3 Buchberger Algorithm

Buchberger algorithm is based on Buchberger Theorem 1 stated by Bruno Buchberger in 1976. It is the first algorithm to compute a Groebner Basis of an Ideal systematically. Buchberger Theorem really told us how to examine whether a Basis is a Groebner Basis or not. According to the theorem, what the algorithm does is very simple:

1. Set the input F to be a Groebner Basis G
2. Check if G satisfies the condition stated in Buchberger Theorem
3. If G does not satisfy, modify the elements in G and go to step 1

We now introduce more detail as follows.

We can see in Algorithm 3.4 the whole flow of Buchberger algorithm. The input of the algorithm is F, the polynomial system we would like to solve, while the output of the algorithm is G, the Groebner Basis of F, which spans the same ideal and contains at least one univariate polynomial.

The algorithm takes the input F and set it as G first (line 3). At the same time, it also maintains a pool of Pairs P to store those pairs that haven't been checked. After the initialization, it starts to examine if G is really a Groener Basis by checking all the pairs generated by G. The pool P stores all these pairs; those which have been examined are removed from P.

Algorithm 3.4 Pseudocode of Buchberger Algorithm

Input: F

REM Initialization

$F \longleftarrow$ ReducedRowEchelon(F)

$G, P \longleftarrow$ UpdateGP(G, P, F)

REM Main program - check pairs and let G satisfied the condition

while $P \neq \{\}$ **do**

  $spoly, P \longleftarrow$ SelectPair(P)

  $r \longleftarrow$ MultivariateDivision($spoly, G$)

  if $r \neq 0$ **then**

    $G, P \longleftarrow$ UpdateGP($G, P, \{r\}$)

  end

end

Output: G

Algorithm 3.5 Pseudocode of SelectPair function in Buchberger Algorithm

Input: P

$pair \longleftarrow$ SelectStrategy(P)

$P \longleftarrow P \backslash \{pair\}$

$spoly \longleftarrow pair.imult^{*} pair.f_i - pair.jmult^{*} pair.f_j$

Output: $spoly, P$

In the while loop (line 5), we pick and check pairs from P one by one. First, SelectPair function picks a pair from P with a select strategy, as Algorithm 3.5 shows. The select strategy is left open, so the user can select pairs in any way she likes, as long as all the pairs are selected and checked eventually. Nevertheless, it is suggested that the user select the pair with the least degree of $pair.lcm$. After SelectPair function, we get a SPolynomial $spoly$. We check if it can be reduced to zero by G at line 7. The MulltivariateDivision function reduces $spoly$ by G and gets r either equal to zero or each monomial mn in r with $g_i \nmid mn, g_i \in G$. By the Buchberger Theorem, if there is one SPolynomial that cannot be reduced to zero by G, then G is not a Groebner Basis. Thus, if $r \neq 0$, then it means we need to modify G to make it satisfy the condition of Buchberger Theorem. The simplest and valid way to do this is to add r into G (line 9). This operation would not change the ideal spanned by G since r is a linear combination of $g_i \in G$. By doing this, $spoly$ is reduced to zero by G since its remainder r is in G now. By keeping checking the pairs and adding new remainders if needed, finally we will arrive at the Groebner Basis.

Algorithm 3.6 Pseudocode of UpdateGP function

Input: G, P, F

1 **forall the** $f \in F$ **do**

2 $\quad P \longleftarrow P \cup \{\text{getPair}(f, g_i) \mid g_i \in G\}$

3 $\quad P \longleftarrow$ BuchbergerCriteria(P)

4 $\quad G \longleftarrow G \cup \{f\} \backslash \{g_i \mid g_i \in G, f | g_i\}$

5 **end**

Output: G, P

The UpdateGP function appears in line 3 and line 9 is indeed taking a new set F, adding its elements into Basis G, and putting the new pairs generated into P, as shown in Algorithm 3.6.

$$F = \left\{ \begin{array}{l} 3x^2 + 2xy + y^2 + 3 \\ 2x^2 + 4xy + y^2 + 1 \end{array} \right\} \iff M = \left[\begin{array}{cccc} x^2 & xy & y^2 & 1 \\ 3 & 2 & 1 & 3 \\ 2 & 4 & 1 & 1 \end{array} \right]$$

Figure 3.5: Transformation between matrix and polynomials

It also discards useless pairs from P in the function BuchbergerCriteria based on Buchberger Criteria 1 and Buchberger Criteria 2. It also removes useless $g_i \in G$ if there is another $g_j \in G$ such that $g_j | g_i$. Notice that the pair of SPoly(g_i, g_j) is in P now according to the design of UpdateGP. Consider a new remainder r to be added into G; then g_i, g_j, r satisfy the condition of Buchberger Criteria 2, which means that we just have to check SPoly(g_i, g_j) and SPoly(g_j, r), and then SPoly(g_i, r) would also satisfy. Indeed, that means G always keeps the leftmost Head Term of every line shown in Figure 3.4.

What the RedcuedRowEchelon function in line 2 does is that it transforms the polynomials F into a matrix M, computes the reduced row echelon form of M, transforms it to polynomials again, and puts them into F. The transformation between matrix and polynomials is as figure 3.5 shows.

The Buchberger algorithm will terminate deterministically since the basis must be finitely generated based on Hilbert's basis theorem and the fact that we keep finding smaller polynomials in the ideal spanned by F. However, it is too slow to be practical in solving a larger system. If there are m polynomials in F, then we have to check almost $\frac{m^*(m-1)}{2}$ pairs, and more as new remainders are added into G. Doing this checking one by one is not very efficient. There are a lot of duplicated work that may be done together to save time. That's how F_4 improved the Buchberger algorithm, which we will show later in Section **??**. More detail of the Buchberger algorithm can be found in [Buc76, 1976].

3.4.4 Faugère's algorithm (F_4)

F_4 algorithm was given by Faugère in 1999. It was based on the idea of Buchberger algorithm with some improvement. The improvement makes it amazingly fast. The changes are summarized as follows.

- Check instead of a pair but a set of pairs at one time
- Do the reduction of SPolynomial in matrix form

We can see that how F_4 works in Algorithm 3.7. The initialization of F_4 is the same as Buchberger algorithm. Both Buchberger algorithm and F_4 use the same ReduceRowEchelon and UpdateGP function. However, the Select Pair step is a little different. Unlike Buchberger, SelectPairs here selects a set of pairs to check and return a set of pre-SPolynomials of those pairs stored in $Poly$, as shown in Algorithm 3.8. The SelectStrategy is left open, so the user can select pairs any way she likes. It is recommended that those pairs with lcm of the least degree be selected first. The improvement here is that the F_4 algorithm increases the throughput of the pairs checking. The reason to select a set of pairs is not only to increase the throughput, but also to eliminate some duplicated work. For example, if there is a monomial m in both f_i, f_j that can be divided by g, in Buchberger algorithm, g must to extend to m twice to do the reduction, whereas in F_4 algorithm, the extension only needs to be done once since we check f_i, f_j together.

The Reduction step here is a little bit complicated. What from line 9 to line 15 in Algorithm 3.7 do is to make sure that every monomial $m \in Poly$, if there exists a g in G such that $g|m$, then g will be extended to m and contained in $Poly$. Monomials($Poly$) is the set of monomials in $Poly$, and Headterms($Poly$) is the set of head terms of $Poly$. The reason to do this step is to ensure that for any $r \in R$ (line 17), r cannot be reduced by G anymore. We can then reduce the $Poly$ to $Poly'$ by Gaussian Elimination after discarding those p in $Poly'$ that can

Algorithm 3.7 Pseudocode of F_4 Algorithm

Input: F

REM Initialization

$F \longleftarrow$ ReducedRowEchelon(F)

$G, P \longleftarrow$ UpdateGP(G, P, F)

REM Main program - check pairs and let G satisfied the condition

while $P \neq \{\}$ **do**

  REM step : Select Pair

  $Poly, P \longleftarrow$ SelectPairs(P)

  REM step : Reduction

  $M \longleftarrow$ Monomials($Poly$)

  $Done \longleftarrow$ Headterms($Poly$)

  while $\exists\, m \in M \backslash Done,\ \exists\, g \in G,\ g|m$ **do**

    $Done \longleftarrow Done \cup \{m\}$

    $Poly \longleftarrow Poly \cup \{g * \frac{m}{\mathrm{HT}(g)}\}$

    $M \longleftarrow$ Monomials($Poly$)

  end

  $Poly' \longleftarrow$ ReducedRowEchelon($Poly$)

  $R \longleftarrow \{p \mid p \in Poly', \mathrm{HT}(p) \notin \mathrm{HT}(Poly)\}$

  REM step : UpdateGP

  $G, P \longleftarrow$ UpdateGP(G, P, R)

end

Output: G

Algorithm 3.8 Pseudocode of SelectPairs function in F_4

Input: P

1 $Pair \longleftarrow$ SelectStrategy(P)

2 $P \longleftarrow P \backslash Pair$

3 $Poly \longleftarrow \{pair.f_i mult^* pair.f_i \ , \ pair.f_j mult^* pair.f_j \mid pair \in Pair\}$

Output: $Poly, P$

be divided by G, after which we get the new remainders R and finish the Reduction step. The UpdateGP step is the same as in Buchberger algorithm. More detailed proof of the correctness of the algorithm, please refer to [Fau99].

Chapter 4

First Approach

Although the approach of FPPR method provides a systematic way to solve Semaev's polynomials, their algorithm is still not practical. Petit and Quisquater estimated that the method could beat generic algorithms for extension degrees n larger than about 2000 [PQ12]. This number is much larger than the parameter $n = 160$ that is currently used in applications. In fact, the degrees of the equations in F grow quadratically with m, and the number of monomial terms in the equations is exponential in this degree. In practice, the sole computation of the Semaev's polynomial s_{m+1} seems to be a challenging task for m larger than 7. Because of the large computation costs (both in time and memory), no experimental result has been provided yet when n is larger than 20.

In this chapter, we provide a variant of the FPPR method that practically improves its complexity. Our method exploits the symmetry of Semaev's polynomials to reduce both the degree of the equations and the number of monomial terms appearing during the computation of a Gröbner basis of the system F.

At first, in the Section 4.1, we will give some example of the previous work using similar idea, and in the Section 4.2, we exploits the same idea by overcoming some structure disadvantages of the specific field with prime extension degree (F_{2^n} with n is prime). We show some

theoretical analysis here to illustrate the improvement of our first approach. At last, in Section 4.3, we show the experimental evidence both in solving single relation search and solving the entire ECDLP problem to indicates that our first approach is more efficient in practice.

4.1 Use of Symmetries in Previous Works

The symmetry of Semaev's polynomials has been exploited in previous works, but always for finite fields $\mathbb{F}_{p^n}$ with *composite* extension degrees n. The approach was already described by Gaudry [Gau09] as a mean to accelerate the Gröbner basis computations. The symmetry of Semaev's polynomials has also been used by Joux and Vitse's to establish new ECDLP records for composite extension degree fields [JV11a, JV12]. Extra symmetries resulting from the existence of a rational 2-torsion point have also been exploited by Faugère *et al.* for twisted Edward curves and twisted Jacobi curves [FGHR12]. In all these approaches, exploiting the symmetries of the system allows reducing the degrees of the equations and the number of monomials involved in the Gröbner basis computation, hence it reduces both the time and the memory costs.

To exploit the symmetry in ECDLP index calculus algorithms, we first rewrite Semaev's polynomial s_{m+1} with the elementary symmetric polynomials.

Definition 6 *Let $x_1, x_2, ..., x_m$ be m variables, then the elementary symmetric polynomials are defined as*

$$\begin{cases} \sigma_1 := \sum_{1 \leq j_1 \leq m} x_{j_1} \\ \sigma_2 := \sum_{1 \leq j_1 < j_2 \leq m} x_{j_1} x_{j_2} \\ \sigma_3 := \sum_{1 \leq j_1 < j_2 < j_3 \leq m} x_{j_1} x_{j_2} x_{j_3} \\ \quad \vdots \\ \sigma_m := \prod_{1 \leq j \leq m} x_j \end{cases} \tag{4.1}$$

Any symmetric polynomial can be written as an algebraic combination of these elementary

symmetric polynomials. We denote the symmetrized version of Semaev's polynomial s_m by s'_m. For example for the curve $E_{\alpha,\beta}$ in characteristic 2, we have

$$s_3 = (x_1x_2 + x_1x_3 + x_2x_3)^2 + x_1x_2x_3 + \beta,$$

where x_3 is supposed to be fixed to some $x(R)$. The elementary symmetric polynomials are

$$\sigma_1 = x_1 + x_2,$$

$$\sigma_2 = x_1x_2.$$

The symmetrized version of s_3 is therefore

$$s'_3 = (\sigma_2 + \sigma_1x_3)^2 + \sigma_2x_3 + \beta.$$

Since x_3 is fixed and the squaring is a linear operation over $\mathbb{F}_2$, we see that symmetrization leads to a much simpler polynomial.

Let us now assume that n is a composite number with a non-trivial factor n'. In this case, we can fix the vector space V as the subfield $\mathbb{F}_{p^{n'}}$ of $\mathbb{F}_{p^n}$. We note that all arithmetic operations are closed on the elements of V for this special choice. In particular, we have

$$if\ x_i \in V\ then\ \sigma_i \in V\ . \tag{4.2}$$

Let now $\{1, \omega_2, \ldots, \omega_{n/n'}\}$ be a basis of $\mathbb{F}_{p^n}/\mathbb{F}_{p^{n'}}$. We can write

$$\sigma_j = d_{j,0}\ for\ 1 \leq j \leq m,$$

$$x_{m+1} = r_1 + r_2\omega_2 + \ldots + r_{n/n'}\omega_{n/n'},$$

where $r_\ell \in \mathbb{F}_{p^n}$ are known and the variables $d_{j,0}$ are defined over $\mathbb{F}_{p^{n'}}$. These relations can be substituted in the equation $s'_{m+1}\ |_{x_{m+1}=x(R)} = 0$ to obtain a system of n/n' equations in the m variables $d_{j,0}$ only. Since the total degree and the degree of s'_m with respect to each symmetric variable σ_i are lower than those of s_m with respect to all non-symmetric variables x_i, the degrees of the equations in the resulting system are also lower and the system is easier to solve. As long as $n/n' \approx m$, the system has a reasonable chance to have a solution.

Given a solution $(\sigma_1, \ldots, \sigma_m)$ for this system, we can recover all possible corresponding values for the variables $x_1, \ldots, x_m$ (if there is any) by solving the system given in Definition 6, or equivalently by solving the symmetric polynomial equation

$$x^m + \sum_{i=1}^{m} \sigma_i x^{m-i} = x^m + \sigma_1 x^{m-1} + \sigma_2 x^{m-2} + \ldots + \sigma_m.$$

Note that the existence of a non-trivial factor of n and the special choice for V are crucial here. Indeed, they allow building a new system that only involves symmetric variables and that is significantly simpler to solve than the previous one.

4.2 Using Symmetries with Prime Extension Degrees

When n is prime, the only subfield of $\mathbb{F}_{2^n}$ is $\mathbb{F}_2$, but choosing $V = \mathbb{F}_2$ would imply to choose $m = n$, hence to work with Semaev's polynomial s_{n+1} which would not be practical when n is large. In Diem's and Faugère *et al.*'s attacks [FPPR12, Die11], the set V is therefore a generic vector subspace of $\mathbb{F}_{2^n}/\mathbb{F}_2$ with dimension n'. In that case, Implication (4.2) does not hold, but we now show how to nevertheless take advantage of symmetries in Semaev's polynomials.

4.2.1 A New System with both Symmetric and Non-Symmetric Variables

Let n be an arbitrary integer (possibly prime) and let V be a vector subspace of $\mathbb{F}_{2^n}/\mathbb{F}_2$ with dimension n'. Let $\{v_1, \ldots, v_{n'}\}$ be a basis of V. We can write

$$\begin{cases} x_j = c_{j,1}v_1 + c_{j,2}v_2 + ... + c_{j,n'}v_{n'}, \ for \ 1 \leq j \leq m \\ x_{m+1} = r_0 + r_1\omega + ... + r_{n-1}w^{n-1}, \end{cases}$$

where $c_{j,\ell}$ with $1 \leq j \leq m$ and $1 \leq \ell \leq n'$ are variables but r_ℓ, $1 \leq \ell \leq n$ are known elements in $\mathbb{F}_2$.

Like in the composite extension degree case, we can use the elementary symmetric polynomials to write Semaev's polynomial s_{m+1} as a polynomial s'_{m+1} in the variables σ_j only. How-

ever since V is not a field anymore, constraining x_j in V does not constrain σ_j in V anymore. Since $\sigma_j \in \mathbb{F}_{2^n}$, we can however write

$$\begin{cases} \sigma_1 = d_{1,0} + d_{1,1}\omega + \ldots + d_{1,n-1}\omega^{n-1}, \\ \sigma_2 = d_{2,0} + d_{2,1}\omega + \ldots + d_{2,n-1}\omega^{n-1}, \\ \qquad\qquad \vdots \\ \sigma_m = d_{m,0} + d_{m,1}\omega + \ldots + d_{m,n-1}\omega^{n-1}. \end{cases}$$

where $d_{j,\ell}$ with $1 \leq j \leq m$ and $1 \leq \ell \leq n$ are binary variables. Using these equations, we can substitute σ_j in s'_{m+1} to obtain

$$s'_{m+1} = f'_0 + f'_1\omega + \ldots + f'_{n-1}\omega^{n-1}$$

where $f'_0, f'_1, ..., f'_{n-1}$ are polynomials in the binary variables $d_{j,\ell}$. Applying a Weil descent on the symmetrized Semaev's polynomial equation $s'_m = 0$, we therefore obtain a polynomial system

$$f'_0 = f'_1 = \ldots = f'_{n-1} = 0$$

in the mn binary variables $d_{j,\ell}$.

The variables $d_{j,\ell}$ must also satisfy certain constraints provided by System (4.1). More precisely, substituting both the x_j and the σ_j variables for binary variables in the equation

$$\sigma_j = \sum_{\substack{I \subset \{1,\ldots,m\} \\ \#I = j}} \prod_{k \in I} x_k \,,$$

we obtain

$$\begin{aligned} d_{j,0} + d_{j,1}\omega + ... + d_{j,n-1}\omega^{n-1} &= \sigma_j = \sum_{\substack{I \subset \{1,\ldots,m\} \\ \#I = j}} \prod_{k \in I} \sum_{\ell=1}^{n'} c_{k,\ell} v_\ell \\ &= g_{j,0} + g_{j,1}\omega + ... + g_{j,n-1}\omega^{n-1} \end{aligned}$$

where $g_{j,\ell}$ are polynomials in the mn' binary variables $c_{i,\ell}$ only. In other words, applying a Weil descent on each equation of System (4.1), we obtain mn new equations

$$d_{j,\ell} = g_{j,\ell}$$

in the $mn + mn'$ binary variables $c_{j,\ell}$ and $d_{j,\ell}$. The resulting system

$$\begin{cases} f'_j = 0, & 1 \leq j \leq n, \\ d_{j,\ell} = g_{j,\ell}, & 1 \leq j \leq m, 1 \leq \ell \leq n, \end{cases}$$

has $mn + n$ equations in $mn + mn'$ binary variables. As before, the system is expected to have solutions if $mn' \approx n$, and it can then be solved using a Gröbner basis algorithm.

In comparison with the simpler FPPR method [FPPR12], the number of variables is multiplied by a factor roughly $(m + 1)$. However, the degrees of our equations are also decreased thanks to the symmetrization, and this may decrease the degree of regularity of the system. In order to compare the time and memory complexities of both approaches, let D_{FPPR} and D_{Appr1} be the degrees of regularity of the corresponding systems. The time and memory costs are respectively roughly $N^{2D_{reg}}$ and $N^{3D_{reg}}$, where N is the number of variables and D_{reg} is the degree of regularity. Assuming that neither D_{FPPR} nor D_{Appr1} depends on n (as suggested by Petit and Quisquater's experiments [PQ12]), that $D_{Appr1} < D_{FPPR}$ (thanks to the use of symmetric variables) and that m is small enough, then the extra $(m+1)$ factors in the number of variables will be a small price to pay for large enough parameters. In practice, experiments are limited to very small n and m values. For these small parameters, we could not observe any significant advantage of this variant with respect to FPPR method. However, the complexity can be improved even further in practice with a clever choice of vector space.

4.2.2 A Special Vector Space

In the prime degree extension case, V cannot be a subfield, hence the symmetric variables σ_j are not restricted to V. This led us to introduce mn variables $d_{j,\ell}$ instead of mn' variables only in the composite extension degree case. However, we point out that some vector spaces may be ``closer to a subfield'' than other ones. In particular if V is generated by the basis $\{1, \omega, \omega^2, \ldots, \omega^{n'-1}\}$, then we have

$$if\ x_j \in V\ then\ \sigma_2 \in V'$$

where $V' \supset V$ is generated by the basis $\{1, \omega, \omega^2, \ldots, \omega^{2n'-2}\}$.

More generally, we can write

$$\begin{cases} \sigma_1 = d_{1,0} + d_{1,1}\omega + ... + d_{1,n'-1}\omega^{n'-1}, \\ \sigma_2 = d_{2,0} + d_{2,1}\omega + ... + d_{2,2n'-2}\omega^{2n'-2}, \\ \qquad \vdots \\ \sigma_m = d_{m,0} + d_{m,1}\omega + ... + d_{m,n-m}\omega^{n-m}. \end{cases}$$

Applying a Weil descent on $s'_{m+1} \mid_{x_{m+1}=x(R)}$ and each equation of System (4.1) as before, we obtain a new polynomial system

$$\begin{cases} f'_j = 0, & 1 \leq j \leq n, \\ d_{j,\ell} = g_{j,\ell}, & 1 \leq j \leq m, 0 \leq \ell \leq j(n'-1), \end{cases}$$

in $n + (n'-1)\frac{m(m+1)}{2} + m$ equations and $n'm + (n'-1)\frac{m(m+1)}{2} + m$ variables.

When m is large and $mn' \approx n$, the number of variables is decreased by a factor 2 if we use our special choice of vector space instead of a random one. For $m = 4$ and $n \approx 4n'$, the number of variables is reduced from about $5n$ to about $7n/2$. For $m = 3$ and $n \approx 3n'$, the number of variables is reduced from about $4n$ to about $3n$ thanks to our special choice for V. In practice, this improvement turns out to be significant.

	s_{m+1}	s'_{m+1}	s'_{m+1} with specific V
variables number	mn'	$mn' + mn$	$mn' + (n' - 1)\frac{m(m+1)}{2} + m$
polynomials number	n	$n + mn$	$n + (n' - 1)\frac{m(m+1)}{2} + m$
D_{reg}	7 or 6	4 or 3	4 or 3

Table 4.1: Comparison for different multivariate polynomial system

4.2.3 New Decomposition Algorithm

Our new algorithm for the decomposition problem is described in Algorithm 4.1. It is denoted as Imp_{Appr1} in this work. The only difference between Imp_{FPPR} and Imp_{Appr1} comes from a different transformation function in the line 1 of Algorithm 4.1. Although the system solved in Imp_{Appr1} contains more variables and equations than the system solved in Imp_{FPPR}, the degrees of the equations are smaller and they involve less monomial terms. We now describe our experimental results.

4.3 Experimental Results

To validate our analysis and experimentally compare our method with FPPR method, we implemented both algorithms in Magma. All our experiments were conducted on a CPU with four AMD Opteron Processor 6276 with 16 cores, running at 2.3GHz with a L3 cache of 16MB. The Operating System was CentOS 6.3 with Linux kernel version 2.6.32-279.14.1.el6.x86_64 and 512GB memory. The programming platform was Magma V2.18-9 in its 64-bit version. Gröbner basis were computed with the *GroebnerBasis* function of Magma. Our implementations of Imp_{FPPR} and Imp_{Appr1} share the same program, except that the former uses Algorithm 3.3 and the latter uses Algorithm 4.1 to set up the binary multivariate system. We first focus on the relation search, then we describe experimental results for a whole ECDLP computation.

Algorithm 4.1 **Decompose** function with binary multivariable polynomial system and symmetric elementary functions (Imp$_{Appr1}$)

Input: $R = [a]P + [b]Q$, factor base F_V

$F \longleftarrow$ TransFromSemaevToBinaryWithSym($s_{m+1} \mid_{x_{m+1}=x(R)}$)

$F\_ \longleftarrow$ GroebnerBasis(F)

$sol(F) \longleftarrow$ GetSolutionFromGroebnerBasis($F\_$)

$sol_m \longleftarrow \{\}$

for $e = \{P_1, P_2, .., P_m\} \in sol(F)$ **do**

  if $P_1 + P_2 + ... + P_m + R = O$ **then**

    $sol_m \longleftarrow sol_m \cup \{e\}$

  end

end

Output: sol_m contains the decomposition elements of R w.r.t. F_V

4.3.1 Relation Search

The relation search is the core of both FPPR method and our variant. In our experiments, we considered a fixed randomly chosen curve $E_{\alpha,\beta}$, a fixed ECDLP with respect to P, and a fixed $m = 3$ for all values of the parameters n and n'. For random integers a and b, we used both FPPR method and our first approach to find factor basis elements $P_j \in F_V$ such that $P_1 + \cdots + P_m = [a]P + [b]Q$.

We focused on $m = 3$ (fourth Semaev's polynomial) in our experiments. Indeed, there is no hope to solve ECDLP faster than with generic algorithms using $m = 2$ because of the linear algebra stage at the end of the index calculus algorithm[1]. On the other hand, the method appears unpractical for $m = 4$ even for very small values of n because of the exponential increase with m of the degrees in Semaev's polynomials.

[1] In fact, even $m = 3$ would require a double large prime variant of the index calculus algorithm described above in order to beat generic discrete logarithm algorithms [Gau09].

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	17	3	6	3.95	1.08	21.51	6	4.50	0.09	23.40
Imp_{Appr1}	17	3	3	0.67	1.14	14.86	3	0.72	0.24	16.26
Imp_{FPPR}	19	3	6	4.44	1.08	27.55	6	4.97	0.11	29.59
Imp_{Appr1}	19	3	3	0.75	1.13	19.75	3	0.79	0.31	20.90
Imp_{FPPR}	23	3	6	5.47	1.06	29.10	6	6.18	0.12	32.25
Imp_{Appr1}	23	3	3	0.91	1.04	15.59	3	0.97	0.14	16.68
Imp_{FPPR}	29	3	6	6.94	1.02	38.85	6	7.75	0.07	43.14
Imp_{Appr1}	29	3	3	1.15	0.95	17.16	3	1.22	0.17	18.43
Imp_{FPPR}	31	3	6	7.38	1.03	41.12	5	8.38	0.06	46.33
Imp_{Appr1}	31	3	3	1.24	0.90	17.59	3	1.30	0.04	18.87
Imp_{FPPR}	37	3	6	8.90	1.00	48.88	6	9.99	0.06	54.81
Imp_{Appr1}	37	3	3	1.48	0.88	19.23	3	1.58	0.05	20.85
Imp_{FPPR}	41	3	6	9.81	0.98	54.35	6	11.17	0.06	61.70
Imp_{Appr1}	41	3	3	1.64	0.87	20.58	3	1.75	0.05	22.60
Imp_{FPPR}	43	3	6	10.47	0.99	57.69	6	11.73	0.06	64.74
Imp_{Appr1}	43	3	3	1.76	0.87	21.28	3	1.86	0.05	23.24
Imp_{FPPR}	47	3	6	11.29	1.00	63.77	5	12.85	0.06	72.47
Imp_{Appr1}	47	3	3	1.92	0.83	23.17	3	2.02	0.06	25.32
Imp_{FPPR}	53	3	6	12.86	1.03	72.06	5	14.57	0.07	81.22
Imp_{Appr1}	53	3	3	2.12	0.79	24.89	2	2.28	0.04	27.52

Table 4.2: Comparison of the relation search ($m = 3$, $n' = 3$) with two strategies, Imp_{FPPR} and Imp_{Appr1}. D_{reg}, var, poly and mono are the degree of regularity, the number of variables, the number of polynomials and the number of monomials in the system. t_{trans} and t_{groe} are the transformation time and solving Gröbner basis time (seconds). men is the memory consumptions for solving the system (MB).

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	17	4	7	15.47	6.81	58.16	7	16.53	1.20	55.37
Imp_{Appr1}	17	4	3	1.31	3.91	31.52	3	1.33	2.23	24.88
Imp_{FPPR}	19	4	7	17.04	6.88	67.24	7	17.85	1.54	64.78
Imp_{Appr1}	19	4	3	1.51	3.26	32.97	3	1.46	1.57	27.11
Imp_{FPPR}	23	4	6	21.06	6.83	95.66	6	22.31	4.67	91.23
Imp_{Appr1}	23	4	3	1.83	3.19	29.63	3	1.81	1.72	22.75
Imp_{FPPR}	29	4	6	26.63	6.56	125.32	6	27.80	1.37	129.78
Imp_{Appr1}	29	4	3	2.30	3.11	32.95	3	2.29	1.06	27.88
Imp_{FPPR}	31	4	6	28.94	3.37	136.23	6	30.19	1.56	142.69
Imp_{Appr1}	31	4	3	2.49	3.20	35.30	3	2.48	1.24	29.22
Imp_{FPPR}	37	4	6	35.03	2.43	172.56	6	35.68	0.88	176.13
Imp_{Appr1}	37	4	3	2.93	2.45	31.32	3	2.96	0.49	32.45
Imp_{FPPR}	41	4	6	37.58	2.79	189.16	6	39.80	0.84	201.77
Imp_{Appr1}	41	4	3	3.24	2.23	33.84	3	3.33	0.56	35.49
Imp_{FPPR}	43	4	6	40.59	2.24	207.05	6	41.39	0.85	210.59
Imp_{Appr1}	43	4	3	3.41	2.23	35.02	3	3.48	0.60	36.51
Imp_{FPPR}	47	4	6	43.37	2.10	225.73	6	46.01	0.66	239.89
Imp_{Appr1}	47	4	3	3.73	2.12	37.93	3	3.84	0.67	39.78
Imp_{FPPR}	53	4	6	50.63	1.86	272.55	6	52.26	0.37	279.83
Imp_{Appr1}	53	4	3	4.19	1.75	40.46	3	4.36	0.46	42.63

Table 4.3: Comparison of the relation search ($m = 3$, $n' = 4$) with two strategies, Imp_{FPPR} and Imp_{Appr1}. D_{reg}, var, poly and mono are the degree of regularity, the number of variables, the number of polynomials and the number of monomials in the system. t_{trans} and t_{groe} are the transformation time and solving Gröbner basis time (seconds). men is the memory consumptions for solving the system (MB).

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	17	5	7	46.53	218.87	723.08	7	48.06	59.82	725.07
Imp_{Appr1}	17	5	4	2.21	485.10	596.46	4	2.16	136.93	492.88
Imp_{FPPR}	19	5	7	50.50	91.61	401.17	7	54.03	41.80	348.01
Imp_{Appr1}	19	5	4	1.97	516.67	619.63	4	2.67	182.92	492.82
Imp_{FPPR}	23	5	7	64.67	70.46	475.55	7	65.07	55.75	381.39
Imp_{Appr1}	23	5	4	3.01	157.86	323.60	4	3.07	17.83	253.16
Imp_{FPPR}	29	5	7	81.75	109.40	587.39	7	80.99	50.75	530.53
Imp_{Appr1}	29	5	4	3.67	140.47	372.59	4	3.82	20.03	278.07
Imp_{FPPR}	31	5	7	84.08	70.64	547.86	7	85.50	53.56	410.47
Imp_{Appr1}	31	5	4	3.99	130.07	362.76	4	4.13	20.98	279.23
Imp_{FPPR}	37	5	7	101.06	158.23	828.44	7	103.29	88.29	690.51
Imp_{Appr1}	37	5	3	4.85	11.68	118.00	3	4.87	6.85	57.52
Imp_{FPPR}	41	5	6	113.85	230.40	889.70	7	114.09	69.12	930.24
Imp_{Appr1}	41	5	3	5.33	13.26	126.19	3	5.34	8.53	58.99
Imp_{FPPR}	43	5	6	118.87	75.46	600.95	6	118.31	39.69	615.72
Imp_{Appr1}	43	5	3	5.41	11.35	89.33	3	5.74	8.21	56.86
Imp_{FPPR}	47	5	6	128.63	65.03	674.87	6	131.95	45.34	693.31
Imp_{Appr1}	47	5	3	6.07	9.57	109.38	3	6.26	4.71	60.15
Imp_{FPPR}	53	5	6	147.66	80.76	810.08	6	150.41	23.31	814.76
Imp_{Appr1}	53	5	3	6.83	6.68	59.58	3	6.96	1.36	59.91

Table 4.4: Comparison of the relation search ($m = 3$, $n' = 5$) with two strategies, Imp_{FPPR} and Imp_{Appr1}.

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	23	6	7	163.45	3888.70	6656.13	7	156.11	3309.43	5025.06
Imp_{Appr1}	23	6	4	4.36	5150.12	4791.31	4	4.42	3082.15	4428.07
Imp_{FPPR}	29	6	7	198.99	4511.74	6685.01	7	204.07	1681.27	6528.03
Imp_{Appr1}	29	6	4	5.67	2848.46	3368.01	4	5.76	932.65	2681.20
Imp_{FPPR}	31	6	7	209.98	4664.25	7336.11	7	206.29	1205.29	7276.85
Imp_{Appr1}	31	6	4	5.82	2811.99	3257.82	4	6.09	1049.14	2616.21
Imp_{FPPR}	37	6	7	248.24	4733.79	9777.27	7	256.90	1126.29	9812.93
Imp_{Appr1}	37	6	4	6.77	1101.04	1327.00	4	7.10	146.14	927.36
Imp_{FPPR}	41	6	7	279.05	1045.53	4416.99	7	266.44	653.92	3062.68
Imp_{Appr1}	41	6	4	7.87	953.60	1361.59	4	8.31	87.61	896.38
Imp_{FPPR}	43	6	7	298.13	1444.41	4288.28	7	280.46	787.02	3796.57
Imp_{Appr1}	43	6	4	8.02	920.13	1340.39	4	8.33	82.37	918.05
Imp_{FPPR}	47	6	7	326.22	1278.79	4524.33	7	326.08	463.62	3287.07
Imp_{Appr1}	47	6	4	9.06	858.66	1296.09	4	9.24	80.54	919.39
Imp_{FPPR}	53	6	7	366.92	2967.03	7311.44	7	359.03	1857.65	6677.92
Imp_{Appr1}	53	6	3	10.48	34.82	151.04	3	10.70	31.21	151.02

Table 4.5: Comparison of the relation search ($m = 3, n' = 6$) with two strategies, Imp_{FPPR} and Imp_{Appr1}.

The experimental results are given in Table 4.2, Table 4.3, Table 4.4 and 4.5. For most values of the parameters n and n', the experiment was repeated 200 times and average values are presented in the table. For large values $n' = 6$, the experiment was only repeated 3 times due to the long execution time.

We noticed that the time required to solve one system varied significantly depending on whether it had solutions or not. Table 4.2, Table 4.3, Table 4.4 and 4.5 therefore present results for each case in separate columns. The table contains the following information: D_{reg} is the maximum degree appearing when solving the binary system with Magma's Gröbner basis routine; var is the number of $\mathbb{F}_2$ variables of the system; poly and mono are the number of polynomials and monomials in the system; rel is the average number of solutions obtained (modulo equivalent solutions through symmetries); t_{trans} and t_{groe} are respectively the time (in seconds) needed to transform the polynomial s_{m+1} into a binary system and to compute a Gröbner basis of this system; mem is the memory required by the experiment (in MB).

The experiments show that the degrees of regularity of the systems occurring during the relation search are decreased from values between 6 and 7 in FPPR method to values between 3 and 4 in our first approach. This is particularly important since the complexity of Gröebner basis algorithms is exponential in this degree. However, this huge advantage of our method comes at the cost of a significant increase in the number of variables, which itself tends to increase the complexity of Gröbner basis algorithms. Our experimental results confirm the analysis of Section 4.2: while our method may require more memory and time for small parameters (n, n'), it becomes more efficient than FPPR method when the parameters increase. We remark that although the time required to *solve* the system may be larger with our method than with FPPR method for small parameters, the time required to *build* this system is always smaller. This is due to the much simpler structure of s'_{m+1} compared to s_{m+1} (lower degrees and less monomial terms).

n	$\#E_{\alpha,\beta}$	Imp_{FPPR}	Imp_{Appr1}
7	4*37	1.574	0.864
11	4*523	8.625	6.702
13	4*2089	49.698	31.058
17	4*32941	2454.470	1364.742
19	4*131431	22474.450	9962.861

Table 4.6: Comparison of two ECDLP strategies, Imp_{FPPR} and Imp_{Appr1}. The last two columns are computing time in seconds.

	probability to get an answer $\frac{2^{mn'}}{m!2^n}$	complexity $N^{\omega D_{reg}}$
m increases	probability increases	D_{reg} increases, N increases.
n' increases	probability increases	N increases.

Table 4.7: Trade-off for choosing m and n'. N: total number of variables. D_{reg}: degree of regularity.

4.3.2 Whole ECDLP Computation

In a next step, we also implemented the whole ECDLP algorithm with the two strategies Imp_{FPPR} and Imp_{Appr1}. For n in $\{7, 11, 13, 17, 19\}$, we ran the whole attack using $m = 3$ and several values for n'. The orders of the curves we picked in our experiments are shown in Table 4.6 together with the experimental results for the best value of n', which turned out to be 3 in all cases. Timings provided in the table are in seconds and averaged over 20 experiments. Table 4.6 clearly shows that our method (Imp_{Appr1}) is more efficient than FPPR method (Imp_{FPPR}).

It may look strange that $n' = 3$ leads to optimal timings at first sight. Indeed, the ECDLP attacks described above use $mn' \approx n$ and a constant value for n' leads to a method close to

exhaustive search. However, this is consistent with the observation already made in [FPPR12, PQ12] that exhaustive search is more efficient than index calculus for small parameters. Table 4.7 also shows that while increasing n' increases the probability to have solutions, it also increases the complexity of the Gröebner basis algorithm. This increase turns out to be significant for small parameters.

Chapter 5

Second Approach

In our first approach, though using the symmetry property and the specific vector space looks efficient, we still encounter the difficulty in calculating larger parameter for n. This is either because of the rapidly growing number of variables, if you always calculate the decomposition with $s_4 \mid_{x_4=x(R)}$ and behaves rather like an exhaustive search, or because of the rapidly growing degree and size of polynomials, due to the recursive construction of Semaev's summation polynomial, if you calculate with larger s_{m+1}. That is, the complicated structure of Semaev's summation polynomial itself makes the problem difficult. Therefore, in this chapter, we focus on the particularity of the Semaev's summation polynomial, namely their resultant structure, and try to make the computation of Semaev's summation polynomial easier.

In the next Section 5.1 and Section 3.1, we will show the new idea to solve the resultant $\mathrm{Res}_X(f^{(1)}, f^{(2)})$ of two polynomials $f^{(1)}$ and $f^{(2)}$ without really calculating the resultant by adding more intermediate extra variables. In the Section 5.3, we will showing the experimental results of the new idea.

5.1 Splitting up the Resultant

Let n be a positive integer and let $p = 2$. Let $f^{(1)}, f^{(2)}$ be two multivariate polynomials over the field $\mathbb{F}_{p^n}$, respectively with $m_1 + 1$ and $m_2 + 1$ variables. To simplify the exposition of our idea, we will assume that $m_1 = m_2 = m$ and that the degrees of $f^{(1)}$ and $f^{(2)}$ are bounded by D with respect to all variables individually. Let $f(x_1, \ldots, x_m, y_1, \ldots, y_m) :=$ $\mathrm{Res}_z X(f^{(1)}(x_1, \ldots, x_m, z), f^{(2)}(y_1, \ldots, y_m, z))$. The polynomial f has degree bounded by $2D$ with respect to all its variables.

Let $n' \approx n/2m$ and let $V = \langle v_1, \ldots, v_{n'} \rangle \subset \mathbb{F}_{p^n}$ be a vector space of dimension n' over $\mathbb{F}_p$, generated by $\{v_1, \ldots, v_{n'}\}$. For ``random'' polynomials $f^{(1)}$ and $f^{(2)}$ with the above characteristics, we expect the equation $f(x_1, \ldots, x_m, y_1, \ldots, y_m) = 0$ to have about one solution such that $x_i \in V$. As recalled in Chapter 4, the standard way to find such a solution is to introduce small field variables to model the vector space constraints and to apply a Weil descent on f.

To exploit the resultant structure of f, we suggest to introduce n additional small field variables z_i such that $z = \sum_{i=1}^{n} z_i \theta_i$, and to perform a Weil restriction on $f^{(1)}$ and $f^{(2)}$ separately instead of doing the Weil restriction on their resultant. This leads to a polynomial system

$$\begin{cases} f_\ell^{(1)}(x_{ij}, z_j) = 0 & \ell = 1, \ldots, n \\ f_\ell^{(2)}(y_{ij}, z_j) = 0 & \ell = 1, \ldots, n \\ x_{ij}^p - x_{ij} = 0 & 1 \leq i \leq m, 1 \leq j \leq n' \\ y_{ij}^p - y_{ij} = 0 & 1 \leq i \leq m, 1 \leq j \leq n' \\ z_j^p - z_j = 0 & 1 \leq j \leq n. \end{cases} \tag{5.1}$$

Note that the polynomials f_ℓ^k have degree bounded by $\lceil \log_p(D) \rceil$ in each block of variables, and by $(m+1)\lceil \log_p(D) \rceil$ in total. We will apply this technique to ECDLP in the next section.

5.2 Application to ECDLP

For an elliptic curve with equation $y^2 + xy = x^3 + \alpha x^2 + \beta$ over $\mathbb{F}_{2^n}$, the second and third summations polynomials are defined by $S_2(x_1, x_2) = x_1 + x_2$ and $S_3(x_1, x_2, x_3) = x_1^2 x_2^2 + x_1^2 x_3^2 + x_2^2 x_3^2 + x_1 x_2 x_3 + \beta$, as mentioned in Section 3.2.2. The next summation polynomials can be recursively defined by

$$S_r(x_1, \ldots, x_r) = \text{Res}_X(S_{r-k}(x_1, \ldots, x_{r-k-1}, X), S_{k+2}(x_{r-k}, \ldots, x_r, X))$$

for an arbitrary integer $k \in [1, r-3]$.

Although the asymptotic analysis in [PQ12] suggests using $m = n^{1/3}$ for optimal efficiency, the huge memory requirements following from a (predicted) degree of regularity approximately equal to m^2 have so far limited m to either 3 or 4 in all experiments. In fact even when additional symmetries coming from composite extension degrees are exploited, the maximal value of m reportedly used in experiments is 6, and the largest summation polynomial computed is S_8 [FHJ$^+$14]!

Splitting the resultant as above can potentially allow for much larger m values, without even needing to computing the corresponding summation polynomial. To compute a relation in Diem's algorithm with $m = 3$, one can apply a Weil descent on both polynomials in the set $\{S_3(x_1, x_2, w), S_3(w, x_3, X)\}$ and solve the resulting system using Gröbner basis algorithms.

This reasoning can be extended to larger m values. By recursively splitting down all the resultant, we finally split the $(m+1)$th summation polynomial into $m-1$ summation polynomials S_3. The method will introduce $m-2$ variables w_i that are unconstrained over $\mathbb{F}_{2^n}$ (corresponding to x-coordinates of partial sums of the first points involved in the summation), so solving the $(m+1)$th summation polynomial.

Assuming that the first fall degree heuristics works in all cases, it is tempting to increase m as much as possible. This will decrease the size of the vector spaces, hence the cost of linear

algebra in the second phase of index calculus. The parameter m can however not be increased too much (say not up to $m \approx n$) since the factor $m!$ arising from the symmetry of Semaev's polynomials will then significantly decrease the probability that one random point can be decomposed as a sum of factor basis elements.

In practice, we could solve $(m+1)$th summation polynomials with m up to 6 using this splitting strategy. For smaller values of m, we could solve them for n values larger than in previous works. Although we cannot give a rough theoretical analysis of complexity of the new idea we proposed here so far, we will shows the experimental evidence of the efficiency of the second approach in the next section. Even the experiments are limited by their memory and time requirements, they do provide some support for efficiency of the new method. Detailed experimental results are provided in Sections 5.3.1 to 5.3.3.

5.3 Experimental results

In order to investigate the validity of our generalized first fall degree assumptions, we first performed experiments on Weil descent systems coming from splitting strategies of Semaev's polynomials. We then also studied the Weil descent of ``generic resultant polynomials''. All algorithms were implemented in Magma, and we used the *GroebnerBasis* function of Magma to compute Gröbner bases.

The experiments of Sections 5.3.1 to 5.3.4 were conducted on a CPU with four 16-cores AMD Opteron Processor 6276, running at 2.3GHz with a L3 cache of 16MB. The Operating System was Linux Mint 14 Nadia with kernel version GNU/Linux 3.5.0-17-generic x86_64 and 512GB memory. The programming platform was Magma V2.18-9 in its 64-bit version. We usually provide average results on at least 10 experiments, except when an experiment takes more than 1000 seconds, in which case we provide average numbers on at least 2 experiments. The time unit is the second, and the memory unit is a MB.

5.3.1 Splitting up Semaev 4

We first applied our splitting strategy to compute relations in Diem's algorithm when $m = 3$. We chose a random curve, a random point (X, Y) on the curve and a random vector space V of appropriate size, and we solved $S_4(x_1, x_2, x_3, X) = 0$ by applying a Weil descent to two polynomials $S_3(x_1, x_2, w)$ and $S_3(w, x_3, X)$ where we left the variable w unconstrained. Table 5.1 and Table 5.2 compares the experimental results obtained using this approach with previous works denoted as Imp_{FPPR} [FPPR12] and Imp_{Appr1} [HPST13]. The table suggests that the degree of regularity remains at 4 in our splitting method. The new method is also faster than both Imp_{FPPR} and Imp_{Appr1} by an order of magnitude, and it requires less memory in most cases. Empty lines in the table occur when we could not finish the computation: we observe that our splitting strategy allows for larger parameters.

5.3.2 Splitting up Semaev 5

We then applied our splitting strategy to compute relations in Diem's algorithm when $m = 4$. We repeated the same steps as the previous section, and we split $s_5(x_1, x_2, x_3, x_4, X)$ into three parts $\{s_3(x_1, x_2, w_1), s_3(w_1, X, w_2), s_3(w_2, x_3, x_4)\}$, where the new variables w_1 and w_2 were left unconstrained. We remark that it is better to split the polynomial in this way than as $\{s_3(x_1, x_2, w_1), s_3(w_1, x_3, w_2), s_3(w_2, x_4, X)\}$ to obtain a better balance of variables in the three equations.

Table 5.3 summarizes the experimental results obtained using this approach. Note that we could not finish the computation using previous methods (neither Imp_{FPPR} [FPPR12] nor Imp_{Appr1} [HPST13]), so we only show the experiments results for the new method here. The table shows that D_{reg} remains at 4 in this case.

5.3.3 Splitting up Semaev 6 and 7

We then considered larger summation polynomials. We followed the same steps as in the previous two sections, and we solved $s_{m+1}(x_1, x_2, ..., x_m, X) = 0$ by applying a Weil descent to split polynomials $s_3(x_1, x_2, w_1)$, $s_3(w_1, x_3, w_2)$, ..., and $s_3(w_{m-2}, x_m, X)$ where we left the variables $w_1, w_2, ..., w_{m-2}$ unconstrained. The results are shown in Table 5.4.

The table shows that we could solve Semaev's polynomials up to $m = 6$ (s_7) using our splitting method. The degree of regularity remains reasonably small, but unlike for smaller m values it does seem slightly larger than 4 when $m = 5$ and $m = 6$. We point out that the experiments here were only repeated twice due to the longer computing time, so the average results may still be affected by variability effects such as the number of solutions.

5.3.4 Generic resultant polynomials

We then studied the case of ``generic resultant polynomials''. To this aim, we generated two random polynomials $f^{(1)}$ and $f^{(2)}$ with $m+1$ variables over the base field F_{2^n}, with degree 2^t-1 in each variables, such that f_1 and f_2 share exactly one variable. We applied a Weil descent on both f and the set $\{f^{(1)}, f^{(2)}\}$, we solve both systems $F^{(res)}$ and $F^{(1+2)}$ using Gröbner basis algorithms, and we compare the timings and the degree of regularity obtained. The experiment results with $t = 1, 2, 3$, $m = 1, 2$ and $7 \leq n \leq 29$ are shown in Fig. 5.1, Fig. 5.2, Fig. 5.3 and Fig. 5.4.

For the largest parameters t and m we tested, solving the system was actually faster than generating it. We believe this is because we did not optimize the Weil descent process (to focus on its resolution) but we did not investigate this further. As an example in the case $(t = 2, m = 1)$, it takes more than 20,000 seconds to setup the system and seconds to solve it. We also noted that the computation sometimes ran out of memory or took too much time to solve $F^{(res)}$ system in the step Gröbner walk no matter how small t and m were, while it worked

well in solving $F^{(1+2)}$. The numbers we report are average values on the cases that could be completed.

The results obtained for generic polynomials are not as spectacular as for Semaev polynomials. On the one hand, we consistently observe smaller degrees of regularity with our splitting method, on the other hand these smaller degrees of regularity do not compensate for the larger number of variables at the parameters we could test, and for most parameters the splitting method was actually slower than solving the Weil descent of the resultant polynomial. We point our that this does not contradict to our claim, which predicts that the new method becomes better for sufficiently large n values.

Another important remark is that for $(t = 1, m = 1)$, the resultant polynomial has actually the same degrees as both $f^{(1)}$ and $f^{(2)}$, so it definitely makes sense that $F^{(res)}$ can be solved faster than $F^{(1+2)}$ in this case. In fact for larger parameters such as $(t = 3, m = 2)$ shown in Fig 5.4, we observed that the splitting method became more efficient.

Interestingly, the splitting strategy performs much better for s_4 than for generic resultant polynomials with $(t = 2, m = 2)$ (arguably the closest case to s_4). We believe this is because of the sparsity and the symmetric structure of the Semaev's polynomial system. The sparsity of a system is quite important in solving the system, as it affects the matrix size occurring in F_4 calculation, hence the run time.

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	17	4	7	15.47	6.81	58.16	7	16.53	1.2	55.37
Imp_{Appr1}	17	4	3	1.31	3.91	31.52	3	1.33	2.23	24.88
Imp_{Appr2}	17	4	4	0.66	4.96	37.38	4	0.65	3.16	39.63
Imp_{FPPR}	17	5	7	46.53	218.87	723.08	7	48.06	59.82	725.07
Imp_{Appr1}	17	5	4	2.21	485.1	596.46	4	2.16	136.93	492.88
Imp_{Appr2}	17	5	4	0.84	10.32	68.55	4	0.85	9.19	61.67
Imp_{FPPR}	19	5	7	50.5	91.61	401.17	7	54.03	41.8	348.01
Imp_{Appr1}	19	5	4	1.97	516.67	619.63	4	2.67	182.92	492.82
Imp_{Appr2}	19	5	4	1.19	18.6	91.91	4	1.11	16.19	86.61
Imp_{FPPR}	19	6								
Imp_{Appr1}	19	6								
Imp_{Appr2}	19	6	4	1.46	135.17	400.89	4	1.44	68.60	380.98
Imp_{FPPR}	23	5	7	64.67	70.46	475.55	7	65.07	55.75	381.39
Imp_{Appr1}	23	5	4	3.01	157.86	323.6	4	3.07	17.83	253.16
Imp_{Appr2}	23	5	4	1.37	49.14	136.1	4	1.41	34.61	108.24
Imp_{FPPR}	23	6	7	163.45	3888.7	6656.13	7	156.11	3309.43	5025.06
Imp_{Appr1}	23	6	4	4.36	5150.12	4791.31	4	4.42	3082.15	4428.07
This Work	23	6	4	1.77	168.12	794.3	4	1.69	146.35	767.47
Imp_{FPPR}	23	7								
Imp_{Appr1}	23	7								
Imp_{Appr2}	23	7	4	2.07	538.77	1981.37	4	2.09	476.41	1936.13

Table 5.1: Comparison FPPR [FPPR12], the first approach [HPST13] and the second approach when $m = 3$

	n	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
Imp_{FPPR}	29	6	7	198.99	4511.74	6685.01	7	204.07	1681.27	6528.03
Imp_{Appr1}	29	6	4	5.67	2848.46	3368.01	4	5.76	932.65	2681.2
Imp_{Appr2}	29	6	4	2.53	636.97	2195.45	4	2.44	489.95	2201.95
Imp_{FPPR}	29	7								
Imp_{Appr1}	29	7								
Imp_{Appr2}	29	7	4	3.02	1907.13	5555.25	4	2.83	1519.04	5545.99
Imp_{FPPR}	29	8								
Imp_{Appr1}	29	8								
Imp_{Appr2}	29	8	4	3.45	5450.35	12297.73	4	3.36	4868.43	11449.37
Imp_{FPPR}	31	6	7	209.98	4664.25	7336.11	7	206.29	1205.29	7276.85
Imp_{Appr1}	31	6	4	5.82	2811.99	3257.82	4	6.09	1049.14	2616.21
Imp_{Appr2}	31	6	4	2.79	1088.8	2984.91	4	2.61	827.2	2988.82

Table 5.2: Comparison FPPR [FPPR12], the first approach [HPST13] and the second approach when $m = 3$

n	m	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
13	4	3	4	1.05	246.38	803.26	4	0.99	178.89	514.82
17	4	3	4	1.77	427.45	2538.02	4	1.47	545.94	2258.43
17	4	4	4	1.90	4252.28	10987.02	4	1.72	2460.63	10483.68
23	4	3	4	3.49	3132.48	14068.59	4	4.46	2061.40	11451.40
23	4	4	4	3.94	27501.20	57511.55	4	4.56	18092.68	57493.99
23	4	5	4	3.73	24890.21	57657.05	4	5.36	16672.26	57553.95
29	4	3	4	7.38	17053.36	57943.28	4	8.05	11804.60	44238.16
29	4	4	4	8.74	122559.83	220688.44	4	7.00	431414.36	671703.84

Table 5.3: Experiment results for the splitting strategy when $m = 4$

n	m	n'	sol: yes				sol: no			
			D_{reg}	t_{trans}	t_{groe}	mem	D_{reg}	t_{trans}	t_{groe}	mem
13	5 (s_6)	2	4.00	1.55	42.39	395.97	4.00	1.45	31.80	276.18
13	5 (s_6)	3	4.33	1.87	686.53	2784.53	4.00	1.74	668.06	3711.62
13	6 (s_7)	2	5.00	2.08	213.70	887.97	4.00	2.00	238.17	932.53
17	5 (s_6)	3	4.33	2.53	4378.38	12444.42	4.00	2.64	572.30	4518.55
17	6 (s_7)	2	4.50	3.15	1295.83	3618.48	4.00	4.00	305.57	2164.90
17	6 (s_7)	3	5.00	3.70	25543.69	25533.75	4.00	3.79	14361.97	15185.44
23	5 (s_6)	3	4.67	4.84	8780.20	28556.62	4.00	5.66	3602.93	23407.68
23	6 (s_7)	3	5.00	6.27	88161.21	43730.85	4.00	5.97	4666.71	16058.17
29	5 (s_6)	3	4.33	10.39	48122.94	102263.43	4.00	9.63	16588.69	81579.91

Table 5.4: Experiment results for the splitting strategy when $m > 4$

n	$F^{(1+2)}$		$F^{(res)}$	
	D_{reg}	time	D_{reg}	time
12	3.60	0.03	3.00	0.00
13	3.20	0.20	3.00	0.00
14	3.00	0.21	3.00	0.00
15	3.00	0.38	3.00	0.01
16	3.20	0.44	3.00	0.01
17	3.60	0.68	3.00	0.02
18	4.00	0.85	3.00	0.01
19	3.20	1.07	3.20	0.03
20	3.20	1.31	3.00	0.02
21	3.60	1.82	3.00	0.11
22	3.20	2.15	3.00	0.03
23	3.80	3.09	3.00	0.17
24	3.60	3.08	3.00	0.16
25	4.00	3.78	3.00	0.27
26	3.20	4.79	3.00	0.24
27	3.20	33.85	3.00	0.39
28	3.50	7.14	3.25	0.34
29	4.00	58.44	3.20	0.60

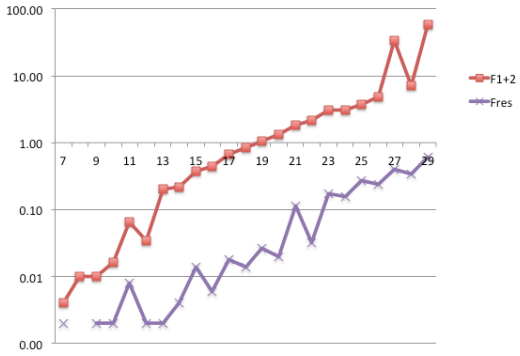


Figure 5.1: Experiments for the splitting strategy with generic resultant polynomial ($t = 1, m = 1$).

n	$F^{(1+2)}$		$F^{(res)}$	
	D_{reg}	time	D_{reg}	time
12	4.20	1.38	5.00	0.06
13	4.20	1.60	5.00	0.10
14	4.80	30.47	5.00	1.36
15	4.60	22.38	5.00	1.62
16	4.00	30.95	5.00	2.23
17	4.20	41.86	5.00	2.64
18	5.00	2599.71	5.00	53.18
19	5.00	3419.45	5.00	47.66
20	5.00	4666.46	5.20	46.81

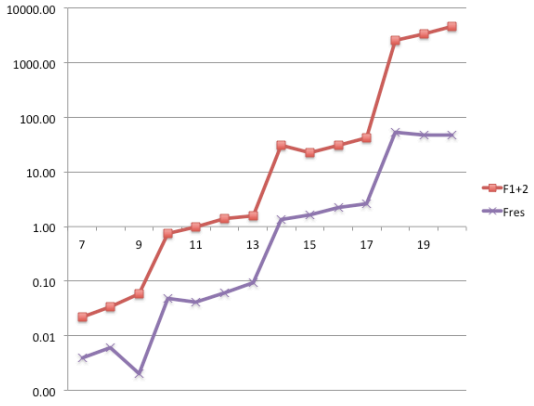


Figure 5.2: Experiments for the splitting strategy with generic resultant polynomial ($t = 1, m = 2$).

n	$F^{(1+2)}$		$F^{(res)}$	
	D_{reg}	time	D_{reg}	time
7	5	0.10	7	0.01
8	5	0.13	7	0.01
9	5	0.84	7	0.09
10	5	1.34	7	0.09
11	5	4.38	7	0.39
12	5	6.91	7	0.54
13	5	221.04	7	4.87
14	5	327.19	7	5.60
15	5	1620.49	7	37.19

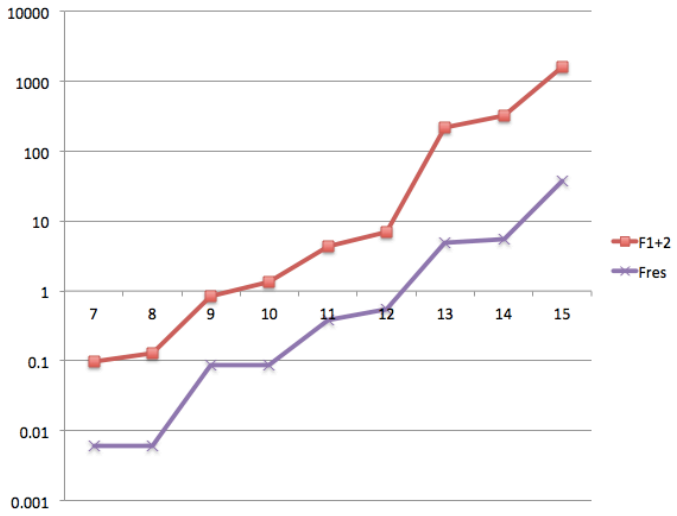


Figure 5.3: Experiments for the splitting strategy with generic resultant polynomial ($t = 2, m = 1$).

t	m	n	$F^{(1+2)}$		$F^{(res)}$	
			D_{reg}	time	D_{reg}	time
2	2	7	7	0.119	8	0.01
2	2	13	5	10.612	9	3.406
3	1	7	5	0.115	7	0.113
3	1	13	5	219.813	7	13.534
3	2	13	7	39.359	12	625.939

Figure 5.4: Experiments for the splitting strategy with generic resultant polynomial for larger t and m.

Chapter 6

Extension Discussion

In this chapter, we will showing more potential idea relating to both ECDLP and other work, based on our previous splitting method in Section 5.1. In the first Section 6.1, we will showing a new algorithm ``binary ECDLP'' to solving ECDLP, and in the later Section 6.1.2 and Section 6.1.3, we will show its variants which is more efficient in solving the problem. In the Section 6.2, we are going to introducing a reduction from the binary ECDLP to ECSSP(elliptic curve subset sum problem), which had been proved to be an NP-complete problem [Che08]. Also, it is shown by [Che08] that an ECDLP is not hard than ECSSP. We will then actually show that we are not only proposed a new algorithm to solving ECDLP, but also solving ECSSP problem.

we implemented the Algorithms of Sections 6.1.1, 6.1.2 and 6.1.3 and showing the experimental results in Section 6.3. Although there are not sufficient experimental results, we still give the evidence that the new method would be more efficient than the previous approach [FPPR12, HPST13] in some cases.

6.1 New binary ECDLP and DLP Algorithms

In this section, we provide several new algorithms for solving ECDLP arising from the new splitting idea showing in Chapter 5. Although some of them are partly backed by small experimental results, we believe that we may extend it in the future work and get good news.

6.1.1 Binary ECDLP Algorithm

We first focus on binary ECDLP. Let n be a prime number, let $K := \mathbb{F}_{2^n}$ and let E be the elliptic curve given by the equation $y^2 + xy = x^3 + \alpha x^2 + \beta$ with $\alpha, \beta \in K$. The second and third summation polynomials are given by $S_2(z_1, z_2) := z_1 + z_2$ and $S_3(z_1, z_2, z_3) := z_1^2 z_2^2 + z_1^2 z_3^2 + z_2^2 z_3^2 + z_1 z_2 z_3 + \beta$. Let $P \in E$ and let $Q \in < P >$. Let N be the order of P.

For any $(P_1, \ldots, P_m) \in E^m$, let x_i be the x-coordinate of P_i and let w_i be the x-coordinate of $Q_i := \sum_{j=1}^{i} P_j$. Our attack goes as follows

1. Let $P_i = (x_i, y_i) := a_i P$ for a_i randomly generated in $\{0, \ldots, N-1\}$, $i = 1, \ldots, n$.

2. Let $R = (X, Y) := aP + bQ$ for a, b randomly generated in $\{0, \ldots, N-1\}$

3. Build the system
$$\begin{cases} S_3(x_1, x_2, w_2) = 0 \\ S_3(w_2, x_3, w_3) = 0 \\ S_3(w_3, x_4, w_4) = 0 \\ \quad \cdots \\ S_3(w_{n-1}, x_n, X) = 0. \end{cases}$$

4. Apply a Weil descent on each equation of this system, restricting x_i in V_i and w_i in the whole field $\mathbb{F}_{2^n}$. Add the field equations to this system. This results in a system with

$n(n-1)$ variables over $\mathbb{F}_2$ (corresponding to the variables $w_2, \ldots, w_{n-1}$ over $\mathbb{F}_{2^n}$ and the variables $x_i \in \mathbb{F}_2$) and $2n(n-1)$ equations, each of them of degree at most 2.

5. Solve this system with F_4. If there is no solution, go back to Step 2.

6. Reconstruct the corresponding solutions w_i over $\mathbb{F}_{2^n}$.

7. Deduce the signs $s_i \in \{\pm 1\}$ in the relation $\sum_{i=1}^{m} s_i P_i = aP + bQ$, one value at the time.

8. Deduce the discrete logarithm $Q = b^{-1}\left(-a + \sum_{i=1}^{n} a_i s_i\right) P$

By heuristic probabilistic arguments, each random choice of a and b is expected to produce one relation on average. Note that we saved the factor $m!$ present in other attacks by constraining each relation term in a distinct ``factor basis'' $\{\pm P_i\}$, similarly to Galbraith and Gebregiyorgis's [GG14].

If the first fall degree assumption holds for the system generated in Step 4, then the degree of regularity of this system should remain very small even for large n. This implies that the runtime complexity of the attack is roughly

$$\left(n^{2\omega D_{reg}}\right).$$

6.1.2 Binary DLP Algorithm, First Variant

We now consider the discrete logarithm problem over the multiplicative group of a finite field of characteristic 2. Let $K := \mathbb{F}_{2^n}$, let g a generator of K^* and let $h = g^k$. We suggest the following algorithm to compute h:

1. Let a_i be randomly chosen elements in $\mathbb{F}_{2^n}^*$, $i = 1, \ldots, n$.

2. Let $\mathcal{F} := \{g, h\} \cup \{e + a_i, e \in \mathbb{F}_2, i = 1, \ldots, n\}$.

3. Let $\mathcal{R} := \emptyset$

4. Relation search: until $\#\mathcal{R} > 2m + 1$ do

   (a) Let $r = g^a h^b$ for a, b randomly generated in $\{0, \ldots, 2^n - 2\}$

   (b) Build the system

$$\begin{cases} w_2 = (x_2 + a_2)(x_1 + a_1) \\ w_3 = (x_3 + a_3)w_2 \\ \quad \ldots \\ w_{n-1} = (x_{n-1} + a_{n-1})w_{n-2} \\ r = (x_n + a_n)w_{n-1} \end{cases}$$

   (c) Apply a Weil descent on each equation of this system, restricting x_i in $\mathbb{F}_2$ and w_i in the whole field $\mathbb{F}_{2^n}$. Add the field equations to this system. This results in a system with $n(n-1)$ variables over $\mathbb{F}_2$ (corresponding to the variables $w_2, \ldots, w_{n-1}$ over $\mathbb{F}_{2^n}$ and the variables x_i over $\mathbb{F}_2$) and $2n(n-1)$ equations, each of them of degree at most 2.

   (d) Solve this system with F4. If there is no solution, go back to Step 4a.

   (e) Add the relation $g^a h^b = \prod(x_i + a_i)$ in $\mathcal{R}$.

5. Do linear algebra on the relations to recover one relation $g^a h^b = 1$, and deduce the discrete logarithm.

The algorithm uses a very small factor basis $\mathcal{F}$. Unlike in the previous algorithm, the discrete logarithms of the factor basis elements are not known a priori, so $2n + 1$ relations are needed and a (very small) linear algebra step is performed at the end.

The runtime complexity of the algorithm can be estimated as

$$n^{2\omega D_{reg}+1}$$

where D_{reg} is the degree of regularity of the above system.

In the next subsection, we provide another DLP algorithm which is closer in spirit to the algorithm of Section 6.1.2. Note that a similar generalization to hyperelliptic curves is straightforward given previous works such as [iN13a].

6.1.3 Binary DLP Algorithm, Second Variant

Semaev's polynomials project the addition law on point elements into an addition law constraint on their x-coordinate. The projection from one point to its x-coordinate is mostly 2-to-1 since the x-coordinates of P and $-P$ are identical.

To extend the algorithm of the previous section to finite fields, we suggest to project any field element x onto the element $z := x+x^{-1}$, which ensures that every element and its inverse have the same image. The second and third summation polynomials then naturally become

$$S_2(z_1, z_2) := z_1 + z_2, \qquad S_3(z_1, z_2, z_3) := z_1^2 + z_2^2 + z_3^2 + z_1 z_2 z_3.$$

Indeed, $S_2 = 0$ if and only if $x_1^{e_1} x_2^{e_2} = 1$ for some $(e_1, e_2) \in \{-1, 1\}^2$, and similarly for S_3. The remaining summation polynomials are then defined inductively using resultants. At this point, generalizing the algorithm of Section 6.1.1 is straightforward, and it is also clear that it has expected polynomial complexity $O(\left(n^{2\omega D_{reg}}\right))$ under an appropriate generalization of the first fall degree assumption.

6.2 Reduction to binary ECSSP

A subset sum problem (SSP) is a classical hard problem related to decoding theorem. An elliptic curve subset sum problem (ECSSP) is the specific problem on the elliptic curve. In the paper of Cheng [Che08], it is stated that the ECSSP is NP-complete and the ECDLP is not harder than ECSSP. We are going to reduce our algorithm to solve ECSSP in this section.

First, we define the elliptic curve subset sum problem. Let E be a curve and $P_1, P_2, ..., P_n, R \in E$. The elliptic curve subset sum problem is to find out that if $R = \sum_{1 \leq i \leq n} e_i P_i$, where $e_i \in \{0, 1\}$.

In this work, we specified $E_{\alpha,\beta}$ be a curve over F_{2^n}, and $P \in E_{\alpha,\beta}$. Then we modified the binary ECDLP method as followed:

1. Let $P_i = (x_i, y_i) := a_i P$ for a_i randomly generated in $\{0, \ldots, N-1\}$, $i = 1, \ldots, n$.

2. Let $R = (X, Y) := (\sum_{1 \leq i \leq n} a_i + a)P$ for a, b randomly generated in $\{0, \ldots, N-1\}$

3. Build the system
$$\begin{cases} S_3(x_1, x_2, w_2) = 0 \\ S_3(w_2, x_3, w_3) = 0 \\ S_3(w_3, x_4, w_4) = 0 \\ \quad \cdots \\ S_3(w_{n-1}, x_n, X) = 0. \end{cases}$$

4. Apply a Weil descent on each equation of this system, restricting x_i in V_i and w_i in the whole field $\mathbb{F}_{2^n}$. Add the field equations to this system. This results in a system with $n(n-1)$ variables over $\mathbb{F}_2$ (corresponding to the variables $w_2, \ldots, w_{n-1}$ over $\mathbb{F}_{2^n}$ and the variables $x_i \in \mathbb{F}_2$) and $2n(n-1)$ equations, each of them of degree at most 2.

5. Solve this system with F_4. If there is no solution, go back to Step 2.

6. Reconstruct the corresponding solutions w_i over $\mathbb{F}_{2^n}$.

7. Deduce the signs $s_i \in \{\pm 1\}$ in the relation $\sum_{i=1}^{m} s_i P_i = (\sum_{1 \leq i \leq n} a_i + a)P$, one value at the time.

8. Deduce the subset sum $aP = \sum_{1 \leq i \leq n} e_i (2a_i P)$.

We modified Step 2 and get a new equation in Step 7. By this reduction, we can solve the problem whether aP is a subset sum of $\{2a_iP\}$ as well as the solution in Step 8. Now we claim that the binary ECDLP is also a solution of ECSSP, hence fore, for the modified version we may like to name it as binary ECSSP. In the next section, we then show some experiments that the new method would be much efficient than the previous approach at least in some cases.

6.3 Experimental results

In order to investigate the efficiency of the new binary ECDLP method (or binary ECSSP method), we implemented the Algorithms of Sections 6.1.1, 6.1.2 and 6.1.3. All algorithms were implemented in Magma, and we used the *GroebnerBasis* function of Magma to compute Gröbner bases. Although we cannot give a rough theoretical analysis of complexity of the new idea and algorithm we proposed here so far, we will shows the experimental evidence of the efficiency of the new algorithm in this section.

The experiments of Sections 6.3.1 to 6.3.2 were conducted on a CPU with four 4-cores Intel Xeon Processor 5550, running at 2.67GHz with 8MB cache. The Operating System was Linux Ubuntu 12.04.5 LTS with kernel version GNU/Linux 3.5.0-17-generic x86_64 and 24GB memory. The programming platform was Magma V2.18-5 in its 64-bit version. The time unit is the second, and the memory unit is a MB.

6.3.1 New Binary ECDLP Algorithm

We implemented the binary ECDLP algorithm of Section 6.1.1. We observed the timings and degrees of regularity obtained and we compared them with previous methods proposed in the Chapter 4 [HPST13]. The results are provided in Table 6.1. Perhaps surprisingly, the degree of regularity remained below 4 in all experiments we did, but we point out that the parameters we could test remained quite small. Table 6.1 also reports the number of times Step 2 of the

n	binary ECDLP in 6.1.1				FPPR	HPST
	D_{reg}	loop	time	mem	time	time
7	4	3	0.180	11.030	1.57	0.86
8	3	3	0.210	11.190		
9	4	2	0.360	13.380		
10	4	2	1.180	35.030		
11	4	1	1.120	24.160	8.63	6.70
12	4	1	4.440	48.500		
13	4	1	19.010	88.410	49.70	31.06
14	4	2	213.350	231.380		
15	4	1	597.570	364.380		
17					2454.47	1364.74

Table 6.1: Comparison of ECDLP algorithms

algorithm was executed.

6.3.2 New Binary DLP Algorithms

We finally implemented the binary DLP algorithms of Sections 6.1.2 and 6.1.3, for which we report results in Table 6.2. As for ECDLP algorithm, the degree of regularity surprisingly remained below 4 in all experiments we did, but again the parameters we could test were quite small, particularly with respect to the latest DLP records. For the second variant, we also report the second step of the algorithm was executed.

n	binary DLP 1st variant			binary DLP 2nd variant			
	D_{reg}	time	mem	D_{reg}	loop	time	mem
7	3	0.14	11.03	4	1	0.15	11.03
8	4	0.18	13.25	3	1	0.17	11.50
9	4	0.47	15.81	4	1	0.35	14.72
10	3	0.71	17.28	3	2	1.69	22.06
11	3	1.53	35.84	4	3	13.60	242.66
12	4	23.62	94.75	4	1	96.19	864.59
13	4	126.53	235.62	4	4	1150.25	3330.28
14	4	503.40	1713.34	4	1	1105.41	5308.00
15	4	2286.90	6486.28	4	1	2647.24	8130.47
16	4	5550.42	2901.28				
17	4	22589.91	12900.94				
18	3	15497.95	15763.69				

Table 6.2: Comparison of the DLP variants

Chapter 7

Conclusion

In this thesis, we proposed two approaches to improve the efficiency in solving the binary elliptic curve discrete logarithm problem (ECDLP). In the first approach [HPST13], our variant takes advantage of the symmetry of Semaev's polynomials to compute relations more efficiently. While symmetries had also been exploited in similar ECDLP algorithms for curves defined over finite fields with *composite* extension degrees, our method is the first one in the case of extension fields with *prime* extension degrees, which is the most interesting case for applications. In our second approach, we further give a new idea in solving the Semaev's summation polynomial using splitting method. As for an extension discussion, we also proposed several new variants of ECDLP algorithm which seems to be even powerful which can solve ECSSP and be more efficient in solving ECDLP.

At Asiacrypt 2012, Petit and Quisquater estimated that FPPR method would beat generic discrete logarithm algorithms for any extension degree larger than roughly 2000. Though we cannot give the accurate theoretical complexity analysis, We provided heuristic arguments and experimental data showing that our method reduces both the time and the memory required to compute a relation in FPPR method, unless the parameters are very small. Our results therefore imply that Petit and Quisquater's bound can be lowered a little.

7.1 Future work

Our work raises several interesting questions. At first, we give two different approach to solving ECDLP, without combing the two ideas into one approach. This could be an extension work in the future of this thesis.

Besides, on a theoretical side, it would be interesting to prove that the degrees of regularity of the systems appearing in solving the Semaev's summation polynomial will not rise rapidly when n increases (in most our experiments for various parameter sizes, they were equal to either 3 or 4). It would also be interesting to provide a more precise analysis of our propose and to precisely estimate for which values of the parameters it will become better than FPPR method.

On a practical side, it would also be interesting to improve the resolution of the systems even further. One idea in that direction is pre-computation. Solving the Semaev's summation polynomial involves solving a large number of closely related systems, where only the value $x(R)$ changes from one system to the other. The transformation of Semaev's polynomial into a binary multivariate system could therefore be done in advance, and its cost be neglected. In fact, even the resolution of the system could potentially be improved using special Gröebner basis algorithms such as F_4 trace [JV11b, Fau99]. A second direction on the practical side is parallelization. A powerful feature of Pollard's ρ method and its variants is their highly-parallelized structure. Since our method saves memory compared to FPPR method, it is also more suited to parallelization. A new solver instead of Gröbner basis solver would be also an option. For example, SAT solver is one of the practical choices, though it is hard to make a complexity analysis. Also, an exhaustive search solver seems be more efficient than a Gröbner basis solver in practice, though its complexity is exponential.

Also, the work we showed in the extension discussion is also an interesting part to work on. Since ECSSP is also an well-known hard problem, we hope our idea could be extended to a

new research direction of such problem.

Using Gröbner basis algorithms to solve ECDLP is a very recent idea. We expect that the index calculus algorithms that have recently appeared in the literature will be subject to further theoretical improvements and practical optimizations in a close future.

Bibliography

[Abe10] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, New York, 2010. Springer.

[BCC+10] DanielJ. Bernstein, Hsieh-Chung Chen, Chen-Mou Cheng, Tanja Lange, Ruben Niederhagen, Peter Schwabe, and Bo-Yin Yang. Ecc2k-130 on nvidia gpus. In Guang Gong and KishanChand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 328--346. Springer Berlin Heidelberg, 2010.

[BKW93] T. Becker, H. Kredel, and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, London, UK, 0 edition, 4 1993.

[Bre80] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*, 20:176--184, 1980.

[BSSC05] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.

[Buc76] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19--29, 1976.

[CCK+00] Nicolas Courtois, Nicolas Courtois, Er Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. *IN ADVANCES IN CRYPTOLOGY, EUROCRYPT'2000, LNCS 1807*, 1807:392--407, 2000.

[Che08] Qi Cheng. Hard problems of algebraic geometry codes. *IEEE Transactions on Information Theory*, 54(1):402--406, 2008.

[CLO07] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[Die06] Claus Diem. An index calculus algorithm for plane curves of small degree. In Hess et al. [HPP06], pages 543--557.

[Die11] Claus Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147:75--104, 2011.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1-3):61--88, 1999.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75--83, New York, NY, USA, 2002. ACM.

[FGHR12] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. Using symmetries in the index calculus for elliptic curves discrete logarithm. *IACR Cryptology ePrint Archive*, 2012:199, 2012.

[FGLM93] J.C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329 -- 344, 1993.

[FHJ+14] Jean-Charles Faugère, Louise Huot, Antoine Joux, Guénaël Renault, and Vanessa Vitse. Symmetrized summation polynomials: Using small order torsion points to speed up elliptic curve index calculus. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 40--57, 2014.

[FK13] Marc Fischlin and Stefan Katzenbeisser, editors. *Number Theory and Cryptography - Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday*, volume 8260 of *Lecture Notes in Computer Science*, New York, 2013. Springer.

[FPPR12] J.-C. Faugère, L. Perret, C. Petit, and G. Renault. Improving the complexity of index calculus algorithms in elliptic curves over binary field. In *Proceedings of Eurocrypt 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 27--44, London, 2012. Springer Verlag.

[Gau09] Pierrick Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *Journal of Symbolic Computation*, 44(12): 1690 -- 1702, 2009.

[GG14] Steven D. Galbraith and Shishay W. Gebregiyorgis. Summation polynomial algorithms for elliptic curves in characteristic two. Personal communication, 2014.

[Gra10] Robert Granger. On the static Diffie-Hellman problem on elliptic curves over extension fields. In Abe [Abe10], pages 283--302.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[HPP06] Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors. *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, volume 4076 of *Lecture Notes in Computer Science*, New York, 2006. Springer.

[HPST13] Yun-Ju Huang, Christophe Petit, Naoyuki Shinohara, and Tsuyoshi Takagi. Improvement of Faugère et al.'s method to solve ECDLP. In Sakiyama and Terada [ST13a], pages 115--132.

[iN13a] Koh ichi Nagao. Decomposition formula of the jacobian group of plane curve. Cryptology ePrint Archive, Report 2013/548, 2013. http://eprint.iacr.org/.

[iN13b] Koh ichi Nagao. Equations system coming from weil descent and subexponential attack for algebraic curve. Cryptology ePrint Archive, Report 2013/549, 2013. http://eprint.iacr.org/.

[JMS12] Lyndon Judge, Suvarna Mane, and Patrick Schaumont. A hardware-accelerated ECDLP with high-performance modular multiplication. *International Journal of Reconfigurable Computing*, 2012, 2012.

[JV11a] Antoine Joux and Vanessa Vitse. Elliptic curve discrete logarithm problem over small degree extension fields. *Journal of Cryptology*, pages 1--25, 2011.

[JV11b] Antoine Joux and Vanessa Vitse. A variant of the F4 algorithm. In Kiayias [Kia11], pages 356--375.

[JV12] Antoine Joux and Vanessa Vitse. Cover and decomposition index calculus on elliptic curves made practical - application to a previously unreachable curve over $\mathbb{F}_{p^6}$. In Pointcheval and Johansson [PJ12], pages 9--26.

[Kia11] Aggelos Kiayias, editor. *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, New York, 2011. Springer.

[Nat09] National Security Agency. The case for elliptic curve cryptography. http://www.nsa.gov/business/programs/elliptic_curve.shtml, January 2009.

[PJ12] David Pointcheval and Thomas Johansson, editors. *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, New York, 2012. Springer.

[Pol75] J. M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15 (3):331--334, 1975.

[Pol00] J. M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13:437--447, 2000.

[PQ12] Christophe Petit and Jean-Jacques Quisquater. On polynomial systems arising from a Weil descent. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 451--466. Springer Berlin Heidelberg, New York, 2012.

[Sem04] Igor Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)*, pages 415--440. Providence, R.I., 1971.

[Sil99] Joseph H. Silverman. The $xedni$ calculus and the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 20:5--40, 1999.

[ST13a] Kazuo Sakiyama and Masayuki Terada, editors. *Advances in Information and Computer Security - 8th International Workshop on Security, IWSEC 2013, Okinawa, Japan, November 18-20, 2013, Proceedings*, volume 8231 of *Lecture Notes in Computer Science*, New York, 2013. Springer.

[ST13b] Michael Shantz and Edlyn Teske. Solving the elliptic curve discrete logarithm problem using semaev polynomials, weil descent and gröbner basis methods - an experimental study. In Fischlin and Katzenbeisser [FK13], pages 94--107.

[SYKY11] Tsunekazu Saito, Shun'ichi Yokoyama, Tetsutaro Kobayashi, and Go Yamamoto. Some relations between $semaev's$ summation polynomials and stange's elliptic nets. *Journal of Math-for-Industry*, 3 (2011A-9):89--92, 2011.

[YC04] Bo-Yin Yang and Jiun-Ming Chen. All in the xl family: Theory and practice. In *ICISC*, pages 67--86, 2004.