Introduction to algebraic approaches for solving isogeny path-finding problems (Theory and Applications of Supersingular Curves and Supersingular Abelian Varieties)

AUTHOR(S):

FUKASAKU, Ryoya; IKEMATSU, Yasuhiko; KUDO, Momonari; YASUDA, Masaya; YOKOYAMA, Kazuhiro

CITATION:

FUKASAKU, Ryoya ...[et al]. Introduction to algebraic approaches for solving isogeny path-finding problems (Theory and Applications of Supersingular Curves and Supersingular Abelian Varieties). 数理解析研究所講究録別冊 2022, B90: 169-184

ISSUE DATE:

2022-06

URL:

http://hdl.handle.net/2433/276280

RIMS Kôkyûroku Bessatsu
B90 (2022), 169–184

Introduction to algebraic approaches for solving isogeny path-finding problems

By

Ryoya FUKASAKU [*], Yasuhiko IKEMATSU [**], Momonari KUDO [***], Masaya YASUDA [†] and Kazuhiro YOKOYAMA [‡]

Abstract

The isogeny path-finding is a computational problem that finds an isogeny connecting two given isogenous elliptic curves. The hardness of the isogeny path-finding problem supports the fundamental security of isogeny-based cryptosystems. In this paper, we introduce an algebraic approach for solving the isogeny path-finding problem. The basic idea is to reduce the isogeny problem to a system of algebraic equations using modular polynomials, and to solve the system by Gröbner basis computation. We report running time of the algebraic approach for solving the isogeny path-finding problem of 3-power isogeny degrees on supersingular elliptic curves. This is a brief summary of [16] with implementation codes.

§ 1. Introduction

Since proposals of the hash function of [3] and the key exchange of [11], supersingular isogeny-based cryptography has received attention as post-quantum cryptography (PQC). The National Institute of Standards and Technology (NIST) has proceeded PQC standardization since 2016. For the PQC standardization process, Jao et al. [10] submitted algorithms of supersingular isogeny key encapsulation, called SIKE, that is based

Received December 11, 2020. Revised April 19, 2021.
2020 Mathematics Subject Classification(s): Primary: 14G50, Secondary: 94A60
Key Words: *Elliptic curves*, *Isogenies*, *Isogeny problems*, *Gröbner basis computation*
Supported by JSPS KAKENHI Grant Numbers 19K22847 and 20K14301, Japan
[*] Faculty of Mathematics, Kyushu University, Fukuoka 819-0395, Japan.
e-mail: `fukasaku@math.kyushu-u.ac.jp`
[**] Institute of Mathematics for Industry, Kyushu University, Fukuoka 819-0395, Japan.
e-mail: `ikematsu@imi.kyushu-u.ac.jp`
[***] Department of Mathematical Informatics, The University of Tokyo, Tokyo 113-8654, Japan.
e-mail: `kudo@mist.u-tokyo.ac.jp`
[†] Department of Mathematics, Rikkyo University, Tokyo 171-8501, Japan.
e-mail: `myasuda@rikkyo.ac.jp`
[‡] Department of Mathematics, Rikkyo University, Tokyo 171-8501, Japan.
e-mail: `kazuhiro@rikkyo.ac.jp`

on [11]. In 2020, NIST selected 15 proposals for the third-round of the standardization process, of which SIKE was selected as an alternate candidate (see [13] for details).

The security of supersingular isogeny-based cryptography is based on the hardness of finding an isogeny connecting two given isogenous supersingular elliptic curves. The meet-in-the-middle approach is a practical way to solve the isogeny path-finding problem. Specifically, it builds two trees of isogenies of prime degrees from both the sides of E and $\widetilde{E}$, respectively, and it finds a collision between the two trees to find the shortest path from E to $\widetilde{E}$. In this paper, we introduce a new approach for solving the isogeny path-finding problem. The basic strategy is to reduce the isogeny problem to a system of algebraic equations using modular polynomials. In particular, we divide a system of algebraic equations into two parts like the meet-in-the-middle approach, and compute their Gröbner bases to efficiently find j-invariants of intermediate curves between given two isogenous elliptic curves E and $\widetilde{E}$. We report running time of the algebraic approach for solving the isogeny problem of 3-power degrees on supersingular elliptic curves over $\mathbb{F}_{p^2}$ with 503-bit prime p, extracted from SIKE-p503 parameters [10], in order to compare with the meet-in-the-middle approach.

§ 2. Mathematical background

We review basic definitions and properties of elliptic curves and their isogenies.

§ 2.1. Elliptic curves over finite fields

Let $p \geq 5$ be a prime, and q a p-power integer. An elliptic curve over the finite field $\mathbb{F}_q$ is given by the (short) Weierstrass form

$$E : y^2 = x^3 + ax + b \quad (a, b \in \mathbb{F}_q) \tag{2.1}$$

with discriminant $\Delta(E) = -16(4a^3 + 27b^2) \neq 0$. The j-invariant of E is defined as $j(E) = -1728\frac{(4a)^3}{\Delta(E)}$. There exists an elliptic curve over $\mathbb{F}_q$ with j-invariant equal to a given $j_0 \in \mathbb{F}_q$. Two elliptic curves are isomorphic over the algebraic closure $\overline{\mathbb{F}}_q$ of $\mathbb{F}_q$ if and only if they both have the same j-invariant. The set of $\mathbb{F}_q$-rational points on E

$$E(\mathbb{F}_q) = \left\{(x, y) \in \mathbb{F}_q^2 : y^2 = x^3 + ax + b\right\} \cup \{\mathcal{O}_E\}$$

forms an abelian group, where $\mathcal{O}_E$ denotes the infinity point on E (see [15, Chapter III] for the group law). The number of $\mathbb{F}_q$-rational points on E, denoted by $\#E(\mathbb{F}_q)$, is represented as $\#E(\mathbb{F}_q) = q + 1 - t$ where t denotes the trace of the q^{th}-power Frobenius map. The trace holds $|t| \leq 2\sqrt{q}$ by Hasse's theorem. An elliptic curve E over $\mathbb{F}_q$ is said *supersingular* if the characteristic p of the base field divides the trace t. Otherwise E is said *ordinary*. Every supersingular elliptic curve over $\overline{\mathbb{F}}_p$ has its j-invariant defined over $\mathbb{F}_{p^2}$ [15, Theorem 3.1, Chapter V], and it is isomorphic over $\overline{\mathbb{F}}_q$ to one defined over $\mathbb{F}_{p^2}$. There are about $\frac{p}{12}$ isomorphism classes of supersingular elliptic curves over $\overline{\mathbb{F}}_p$. For every $n \geq 2$, we let

$$E[n] = \left\{P \in E(\overline{\mathbb{F}}_q) : nP = \mathcal{O}_E\right\}$$

denote the subgroup of $E(\overline{\mathbb{F}}_q)$ of torsion points of order n.

§ 2.2. Isogenies between elliptic curves

A morphism $\phi : E \longrightarrow E'$ between two elliptic curves E and E' satisfying $\phi(\mathcal{O}_E) = \mathcal{O}_{E'}$ is called an *isogeny*. Two curves are called *isogenous* (over $\mathbb{F}_q$) if there is a non-zero isogeny (over $\mathbb{F}_q$) between them. Tate's theorem [17] states $\#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q)$ if two curves E and E' are isogenous over $\mathbb{F}_q$. Every non-zero isogeny $\phi : E \longrightarrow E'$ induces an injection between function fields $\phi^* : \overline{\mathbb{F}}_q(E') \longrightarrow \overline{\mathbb{F}}_q(E)$ [15, Chapter III]. The *degree* of a non-zero isogeny ϕ is defined as the extension degree between function fields:

$$\deg \phi = \left[\overline{\mathbb{F}}_q(E) : \phi^* \overline{\mathbb{F}}_q(E')\right].$$

A non-zero isogeny ϕ is said *separable* if the extension $\overline{\mathbb{F}}_q(E)/\phi^*\overline{\mathbb{F}}_q(E')$ is separable. In particular, a non-zero isogeny is separable if its degree $\deg \phi$ is not divisible by the characteristic p of the base field. A non-zero isogeny $\phi : E \longrightarrow E'$ also induces a surjective group homomorphism from $E(\overline{\mathbb{F}}_q)$ to $E'(\overline{\mathbb{F}}_q)$, and its kernel is a finite subgroup of $E(\overline{\mathbb{F}}_q)$, denoted by $E[\phi]$. It satisfies $\deg \phi = \#E[\phi]$ if ϕ is separable. Conversely, given a finite subgroup C of $E(\overline{\mathbb{F}}_q)$, there is a unique elliptic curve E' and a separable isogeny $\phi : E \longrightarrow E'$ with $E[\phi] = C$ [15, Proposition 4.12, Chapter III]. The target curve E' and the corresponding isogeny ϕ are denoted by E/C and ϕ_C, respectively.

2.2.1. Vélu's formula Given a finite subgroup C of $E(\overline{\mathbb{F}}_q)$, Vélu [18] gave an explicit representation of $\phi_C : E \longrightarrow E/C$ and the Weierstrass equation for E/C. Let $P = (x_P, y_P)$ be a point of prime order $\ell \neq p$ on an elliptic curve E defined by (2.1). For the cyclic subgroup $C = \langle P \rangle$, we present Vélu's formula for $\ell = 2$ and 3 below:

- For $\ell = 2$, let $v = 3x_P^2 + a, a' = a - 5v, b' = b - 7vx_P$. Then the Weierstrass equation for E/C is given by $Y^2 = X^3 + a'X + b'$. The image $\phi_C(x, y)$ of the isogeny ϕ_C for $(x, y) \in E(\overline{\mathbb{F}}_q)$ is given by

$$\left(x + \frac{v}{x - x_P},\ y - \frac{vy}{(x - x_P)^2}\right).$$

- For $\ell = 3$, let $u = 4y_P^3, v = 3x_P^2 + a, a' = a - 5v, b' = b - 7(u + vx_P)$. The Weierstrass equation for E/C is given by the same form as in the case $\ell = 2$. The image $\phi_C(x, y)$ of the isogeny ϕ_C for $(x, y) \in E(\overline{\mathbb{F}}_q)$ is given by

$$\left(x + \frac{v}{x - x_P} + \frac{u}{(x - x_P)^2},\ y\left\{1 - \frac{v}{(x - x_P)^2} + \frac{2u}{(x - x_P)^3}\right\}\right).$$

2.2.2. Modular polynomials For every $\ell \geq 2$, the *modular polynomial* $\Phi_\ell(X,Y)$ parameterizes pairs of elliptic curves with a cyclic isogeny of degree ℓ between them (see [14, Exercise 2.18, Chapter II]). For two curves E and E', there is an isogeny of degree ℓ from E to E' with cyclic kernel if and only if $\Phi_\ell(j(E), j(E')) = 0$. The modular polynomial is symmetric in each variable, and its integer coefficients become very large as ℓ increases. For a prime ℓ, the modular polynomial $\Phi_\ell(X,Y)$ is equal to the form

$$X^{\ell+1} - X^\ell Y^\ell + Y^{\ell+1} + \sum_{i,j \leq \ell, i+j<2\ell} a_{ij} X^i Y^j \quad (a_{ij} \in \mathbb{Z}), \tag{2.2}$$

since there are precisely $\ell + 1$ subgroups of the ℓ-torsion group of an elliptic curve E.

2.2.3. Supersingular isogeny graphs For each prime $\ell \neq p$, any two supersingular elliptic curves E and E' over $\mathbb{F}_{p^2}$ are connected by a chain of isogenies of degree ℓ. Such two curves can be connected by m isogenies of degree ℓ for $m = O(\log p)$ [12, Theorem 79]. The *supersingular ℓ-isogeny graph* over $\mathbb{F}_{p^2}$ is the graph $\mathcal{G}_\ell(\overline{\mathbb{F}}_{p^2}) := (V, G)$ whose vertices V is the set of the $\overline{\mathbb{F}}_{p^2}$-isomorphism classes of supersingular elliptic curves over $\mathbb{F}_{p^2}$ labeled by their j-invariants ($\#V \approx \frac{p}{12}$), and whose edges G are the pairs $(j(E), j(E'))$ for ℓ-isogenous curves E and E'. The ℓ-isogeny graph $\mathcal{G}_\ell(\overline{\mathbb{F}}_{p^2})$ is regular with regularity degree $\ell + 1$. When $p \equiv 1 \bmod 12$, it is a Ramanujan graph, an optimal expander graph on which random walks quickly reach the uniform distribution.

§ 3. Computational isogeny problems

There are a number of computational problems related to isogenies for the security of (supersingular) isogeny-based cryptography. A template is the *general isogeny problem* [8, Definition 1]; "Given two elements j, j' in $\mathbb{F}_q$, find an isogeny $\phi : E \longrightarrow E'$, if exists, such that $j(E) = j$ and $j(E') = j'$." There are a variety of representations of ϕ, such as a chain of isogenies of low degrees, a sequence of j-invariants of intermediate curves, and a path in an isogeny graph between E and E'. Below we present typical cryptographic schemes and their related isogeny problems:

§ 3.1. The isogeny path-finding problem

The first cryptographic construction over a supersingular isogeny graph is the hash function in [3]. For a large prime p and a small prime ℓ (e.g., $\ell = 2$ or 3), we consider the supersingular ℓ-isogeny graph over $\mathbb{F}_{p^2}$. We fix j_0 as the initial vertex in the graph, and determine the order of the edges at each vertex by sorting the j-invariants of $\ell + 1$ neighbours. Given a message $(m_0, m_1, \ldots, m_{N-1})$, the hash function proceeds as below:

1. We first choose the edge of j_0 according to the value of m_0, and compute the corresponding neighbour j_1.

2. For $1 \leq k < N$, we repeat to choose the edge of j_k according to m_k (excluding the edge between j_{k-1} and j_k), and compute the corresponding neighbour j_{k+1}.

3. We return the final invariant j_N as the output value of the hash function.

The security of the hash function is based on the hardness of the following problem: "Let p and ℓ be distinct prime numbers, e a positive integer, and E and E' two supersingular elliptic curves over $\mathbb{F}_{p^2}$. Suppose that there exists an isogeny of degree ℓ^e between E and E'. Find an isogeny of degree ℓ^e from E to E'." In terms of cryptography, it is preimage resistant if and only if, given two supersingular invariants j and j', it is computationally hard to compute an isogeny $\phi : E \longrightarrow E'$ of prime power degree ℓ^e with $j = j(E)$ and $j' = j(E')$. It is also collision resistant if and only if, given one supersingular invariant $j = j(E)$ for some elliptic curve E, it is computationally hard to compute an endomorphism $\varphi : E \longrightarrow E$ of prime power degree ℓ^e.

§ 3.2. The computational supersinglular isogeny problem

Let $p = \ell_A^{e_A} \ell_B^{e_B} - 1$ be a large prime, where ℓ_A and ℓ_B are distinct small primes satisfying $\ell_A^{e_A} \approx \ell_B^{e_B} \approx p^{1/2}$. Take a supersingular elliptic curve E over $\mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Then the group $E(\mathbb{F}_{p^2})$ has all torsion points of order $p+1$ and it contains two torsion groups $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$. Two bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ for $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ are fixed in the supersingular isogeny Diffie-Hellman (SIDH) key agreement scheme [11]. The procedure of SIDH between Alice and Bob is as below:

1. Alice randomly selects $m_A, n_A \in [0, \ell_A^{e_A} - 1]$, not both divisible by ℓ_A, and computes the isogeny $\phi_A : E \longrightarrow E_A = E/\langle R_A \rangle$ with $R_A = m_A P_A + n_A Q_A \in E[\ell_A^{e_A}]$. While keeping m_A, n_A, R_A and ϕ_A secret, she transmits $E_A, \phi_A(P_B)$ and $\phi_A(Q_B)$ to Bob.

2. Similarly, Bob randomly selects $m_B, n_B \in [0, \ell_B^{e_B} - 1]$, not both divisible by ℓ_B, and $\phi_B : E \longrightarrow E_B = E/\langle R_B \rangle$ with $R_B = m_B P_B + n_B Q_B \in E[\ell_B^{e_B}]$. While keeping m_B, n_B, R_B and ϕ_B secret, Bob transmits $E_B, \phi_B(P_A)$ and $\phi_B(Q_A)$ to Alice.

3. After that, Alice computes $m_A \phi_B(P_A) + n_A \phi_B(Q_A) = \phi_B(R_A)$ and $E_B/\langle \phi_B(R_A) \rangle$ whereas Bob computes $m_B \phi_A(P_B) + n_B \phi_A(Q_B) = \phi_A(R_B)$ and $E_A/\langle \phi_A(R_B) \rangle$. Then the two compositions of isogenies

$$E \xrightarrow{\phi_A} E_A \longrightarrow E_A/\langle \phi_A(R_B) \rangle \quad \text{and} \quad E \xrightarrow{\phi_B} E_B \longrightarrow E_B/\langle \phi_B(R_A) \rangle$$

have the common kernel $\langle R_A, R_B \rangle$, and thus the two target curves are isomorphic. Hence Alice and Bob can share the same j-invariant of these curves as a secret.

The security of SIDH relies on the hardness of the computational supersingular isogeny problem [6]; "Given two curves E, E_A and two points $\phi_A(P_B)$, $\phi_A(Q_B)$ on E_A, compute an isogeny $\phi_A : E \longrightarrow E_A$ of degree $\ell_A^{e_A}$."

Remark 3.1. As discussed in [4], there exists a reduction between the computational supersingular isogeny problem and the isogeny path-finding problem. In the below sections, we shall focus on the isogeny path-finding problem. The computational hardness of the problem assures the security of the hash function [3] as mentioned in this section, and it is also deeply connected with the security of SIDH.

§ 4. Algebraic approach for solving the isogeny path-finding problem

We introduce an algebraic approach for solving the isogeny path-finding problem. Let us define our setting of problem; "Let $\ell = \ell_0^e$ be a power of a small odd prime ℓ_0. Suppose that there exists an isogeny of degree ℓ between two elliptic curves E and $\widetilde{E}$ over $\mathbb{F}_q$. Given ℓ, $j = j(E)$ and $\tilde{\jmath} = j(\widetilde{E})$, find an isogeny $\phi : E \longrightarrow \widetilde{E}$ of degree ℓ."

§ 4.1. 2-section method: A basic approach

Consider a chain of isogenies ϕ_k of prime degree ℓ_0 from E to $\widetilde{E}$ as

$$E \xrightarrow{\phi_1} E_1 \xrightarrow{\phi_2} E_2 \xrightarrow{\phi_3} \cdots \xrightarrow{\phi_{e-1}} E_{e-1} \xrightarrow{\phi_e} \widetilde{E}.$$

Let j_k denote the j-invariant of E_k for every $1 \leq k < e$. We here regard j-invariants j_k's as *variables* (cf., two elements $j, \tilde{\jmath}$ are in $\mathbb{F}_q$). Then we consider a system of algebraic equations using modular polynomials

$$\begin{cases} \Phi_{\ell_0}(j, j_1) = 0, \\ \Phi_{\ell_0}(j_k, j_{k+1}) = 0 \quad (1 \leq k \leq e-2), \\ \Phi_{\ell_0}(j_{e-1}, \tilde{\jmath}) = 0. \end{cases} \tag{4.1}$$

A solution of this system gives all j-invariants j_k's of intermediate curves E_k. We introduce a method to solve the system (4.1) by Gröbner basis algorithms. (See textbooks [1, 5] for Gröbner basis computation.) For simplicity, assume that the exponent e of the isogeny degree ℓ is even with $e = 2e_0$ for a positive integer e_0. We divide the system (4.1) into two parts like the meet-in-the-middle approach. In terms of Gröbner basis computation, we consider two ideals in different multivariate polynomial rings

$$\begin{aligned} I &= \langle \Phi_{\ell_0}(j, j_1), \Phi_{\ell_0}(j_1, j_2), \ldots, \Phi_{\ell_0}(j_{e_0-1}, j_{e_0}) \rangle \subset \mathbb{F}_q[j_1, \ldots, j_{e_0}], \\ \widetilde{I} &= \langle \Phi_{\ell_0}(j_{e_0}, j_{e_0+1}), \Phi_{\ell_0}(j_{e_0+1}, j_{e_0+2}), \ldots, \Phi_{\ell_0}(j_{e-1}, \tilde{\jmath}) \rangle \subset \mathbb{F}_q[j_{e_0}, \ldots, j_{e-1}]. \end{aligned} \tag{4.2}$$

Both ideals I and $\widetilde{I}$ are zero-dimensional since $j, \tilde{\jmath} \in \mathbb{F}_q$. The dimensions of $\mathbb{F}_q$-vector spaces $\mathbb{F}_q[j_1, \ldots, j_{e_0}]/I$ and $\mathbb{F}_q[j_{e_0}, \ldots, j_{e-1}]/\widetilde{I}$ are both at most $(\ell_0 + 1)^{e_0}$ due to the form (2.2) of the modular polynomial. Moreover, the generators in (4.2) form a Gröbner basis for the ideal I (resp., the ideal $\widetilde{I}$) with the lex term order with respect to

$$j_{e_0} \succ \cdots \succ j_2 \succ j_1 \quad (\text{resp., } j_{e_0} \succ \cdots \succ j_{e-2} \succ j_{e-1}).$$

Then we can efficiently compute minimal polynomials g and $\tilde{g}$ of the variable j_{e_0} with respect to ideals I and $\widetilde{I}$, respectively, by using the FGLM algorithm [7]. Both degrees of g and $\tilde{g}$ are equal to $(\ell_0+1)\ell_0^{e_0-1}$, which is equal to the number of subgroups of exact order $\ell_0^{e_0}$ respectively in E and $\widetilde{E}$. By the GCD computation over the univariate polynomial ring $\mathbb{F}_q[j_{e_0}]$, we obtain a common root of two minimal polynomials g and $\tilde{g}$. Such a common root is a solution of j_{e_0}. Once a solution of j_{e_0} is found, the isogeny problem is divided into two isogeny problems of smaller degree $\ell_0^{e_0}=\sqrt{\ell}$ (i.e., a divide-and-conquer strategy). By repeating this procedure, we can solve the whole problem.

§ 4.2. 3-section method: An improvement

The idea is simple to divide the system (4.1) into *three parts.* Using two parameters e_1 and e_2 satisfying $1<e_1<e_0<e_2<e$ and $e_1\approx e-e_2$, consider two ideals in different multivariate polynomial rings

$$\begin{aligned} I_{[1:e_1]} &= \langle \Phi_{\ell_0}(j,j_1), \Phi_{\ell_0}(j_1,j_2), \dots, \Phi_{\ell_0}(j_{e_1-1}, j_{e_1})\rangle\,, \\ \widetilde{I}_{[e_2:e-1]} &= \langle \Phi_{\ell_0}(j_{e_2}, j_{e_2+1}), \Phi_{\ell_0}(j_{e_2+1}, j_{e_2+2}), \dots, \Phi_{\ell_0}(j_{e-1}, \tilde{\jmath})\rangle\,. \end{aligned}$$

As in the 2-section method, we use the lex term order with $j_{e_1}\succ\cdots\succ j_2\succ j_1$ (resp., $j_{e_2}\succ j_{e_2+1}\succ\cdots\succ j_{e-1}$) for the zero-dimensional ideal $I_{[1:e_1]}$ (resp., $\widetilde{I}_{[e_2:e-1]}$). Then a Gröbner basis for the ideal $I_{[1:e_1]}$ (resp., $\widetilde{I}_{[e_2:e-1]}$) includes a polynomial $g(j_{e_1})$ (resp., $\tilde{g}(j_{e_2})$) such that its roots coincide with those of $\Phi_{\ell'}(j,j_{e_1})$ of level $\ell'=\ell_0^{e_1}$ (resp., $\Phi_{\tilde{\ell}'}(j_{e_2},\tilde{\jmath})$ of level $\tilde{\ell}'=\ell_0^{e-e_2}$). In other words, the polynomials g and $\tilde{g}$ are minimal polynomials for zero-dimensional ideals $I_{[1:e_1]}$ and $\widetilde{I}_{[e_2:e-1]}$, respectively. We then consider a new ideal

$$J_{[e_1:e_2]} = \langle g(j_{e_1}), \Phi_{\ell_0}(j_{e_1}, j_{e_1+1}), \dots, \Phi_{\ell_0}(j_{e_2-1}, j_{e_2}), \tilde{g}(j_{e_2})\rangle\,.$$

For this zero-dimensional ideal, we use the grevlex term order with

$$j_{e_1}\prec j_{e_2}\prec j_{e_1+1}\prec j_{e_2-1}\prec\cdots\prec j_{e_0}$$

in order to find intermediate j-invariants from j_{e_1} to j_{e_2}. With these j-invariants, we can recover the other j-invariants as in the 2-section method.

§ 5. Implementation and experiments

We report experimental results of the algebraic approach and the meet-in-the-middle approach for solving the isogeny path-finding problem of 3-power degrees.

§ 5.1. Implementation

We describe details of our implementation for the algebraic approach and the meet-in-the-middle approach with Magma [2], a computational algebra system. See Appendix for Magma codes of the algebraic approach.

5.1.1. For the algebraic approach We used a combination of the modular polynomials $\Phi_N(X,Y)$ for $N = 3, 3^2, 3^3$, which are pre-computed in Magma, in order to obtain the minimal polynomials $g, \tilde{g}$. For example, for $\ell = 3^{10}$, we computed Gröbner bases for $I = \langle \Phi_{3^3}(j, j_3), \Phi_{3^2}(j_3, j_5)\rangle$, $\widetilde{I} = \langle \Phi_{3^2}(j_5, j_7), \Phi_{3^3}(j_7, \tilde{j})\rangle$ with the Magma command `GroebnerBasis` to obtain $g, \tilde{g} \in \mathbb{F}_{p^2}[j_5]$, then computed the GCD of $g, \tilde{g}$ with the Magma commands `GCD` for the 2-section method.

5.1.2. For the meet-in-the-middle approach We constructed two sets J and $\widetilde{J}$ of sequences $(j, j_1, \ldots, j_{e_0})$ and $(\tilde{j}, \tilde{j}_1, \ldots, \tilde{j}_{e_0})$ of j-invariants of elliptic curves E_k and $\widetilde{E}_k$, respectively. Here E_{k-1} and E_k (resp., $\widetilde{E}_{k-1}$ and $\widetilde{E}_k$) are isogenous of degree 3 for every $1 \le k \le e_0$, where we set $E_0 = E$ (resp., $\widetilde{E}_0 = \widetilde{E}$) for convenience. To construct sequences in J and $\widetilde{J}$, we used the modular polynomial of level 3. For example, we added each solution of $\Phi_3(j_{k-1}, x)$ to a sequence $(j, j_1, \ldots, j_{k-1})$ of length k. (We used the Magma command `Roots` to find a solution.) In constructing such sequences, we removed a sequence whose ending point already appeared in the other sequences, in order to reduce the sizes of two sets J and $\widetilde{J}$. In our experiments, it terminated when we find a pair of two sequences of J and $\widetilde{J}$ satisfying $j_{e_0} = \tilde{j}_{e_0}$ (i.e., a collision).

§ 5.2. Experiments

5.2.1. Input parameters We fix parameters $p = 2^{250} \cdot 3^{159} - 1$, $q = p^2$ and $E : y^2 = x^3 + x$, extracted from SIKE-p503 parameters [10]. (The quadratic extension field $\mathbb{F}_{p^2}$ is represented as $\mathbb{F}_p[z]/(z^2+1)$.) The initial curve E is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ having $\#E(\mathbb{F}_{p^2}) = (p+1)^2 = (2^{250} \cdot 3^{159})^2$ and $j = j(E) = 1728$. We also take 3-powers $\ell = 3^e$ (i.e., $\ell_0 = 3$) as isogeny degrees for even exponents $e = 2e_0$. (cf., SIKE uses a combination of isogenies of degrees 2 and 3.) We follow the method in [10] to generate the target supersingular curve $\widetilde{E}$, isogenous to E of degree ℓ.

5.2.2. Experimental results In Table 1, we summarize average running times of the algebraic approach and the meet-in-the-middle approach for solving the isogeny problem of degrees $\ell = 3^e$ with even e from $e = 6$ up to 14. We measured the running time of every approach until it finds the j-invariant or the Weierstrass coefficients of an intermediate curve between two isogenous curves E and $\widetilde{E}$. We also experimented 5 times for every parameter set. All the experiments were performed using Magma 2.24-5 on 4.20 GHz Intel Core i7 CPU with 16 GByte memory.

5.2.3. Discussion From Table 1, the algebraic approach by the 3-section method is the fastest for isogeny degrees up to $\ell = 3^{10}$. For isogeny degrees larger than $\ell = 3^{12}$, the meet-in-the-middle approach is faster than the algebraic approach. With respect to the memory usage, the algebraic approach by the 2-section method requires about 64, 221 and 526 MByte for $\ell = 3^{10}$, 3^{12} and 3^{14}, respectively, while the meet-in-the-middle approach requires about 24, 26 and 32 MByte for the same isogeny degrees. The

Table 1. Average running times (seconds) of the algebraic and the meet-in-the-middle approaches for solving the isogeny problem of degrees $\ell = 3^e$ on supersingular elliptic curves over $\mathbb{F}_{p^2}$ with 503-bit prime $p = 2^{250} \cdot 3^{159} - 1$, extracted from SIKE-p503 [10]

Isogeny degree $\ell = 3^e$	Algebraic approach 2-section	Algebraic approach 3-section	Meet-in-the-middle approach
$3^6 = 729$	0.11	0.14	2.93
$3^8 = 6561$	0.19	0.40	9.61
$3^{10} = 59049$	7.98	2.72	31.09
$3^{12} = 531441$	287.77	58.43	96.75
$3^{14} = 4782969$	5071.16	1775.45	292.82

algebraic approach is applicable in the collision search step of the meet-in-the-middle approach. Such combination could make it faster in practice and reduce the memory size of the meet-in-the-middle approach. See a subsequent work [9] for such combination. On the other hand, the algebraic approach would be (much) slower for degrees $\ell = \ell_0^e$ with larger prime ℓ_0, since the ℓ_0-th modular polynomial has total degree $\ell_0 + 1$ (see the form (2.2)).

References

[1] Thomas Becker and Volker Weispfenning. *Gröbner bases*, volume 141 of *Graduate Texts in Mathematics*. Springer, 1993.

[2] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[3] Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, 2009.

[4] Anamaria Costache, Brooke Feigon, Kristin Lauter, Maike Massierer, and Anna Puskás. Ramanujan graphs in cryptography. *IACR ePrint 2018/593*, 2018.

[5] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms: An introduction to computational algebraic geometry and commutative algebra*. Springer Science & Business Media, 2013.

[6] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[7] Jean-Charles Faugere, Patrizia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[8] Steven D Galbraith and Frederik Vercauteren. Computational problems in supersingular elliptic curve isogenies. *Quantum Information Processing*, 17(10):265, 2018.

[9] Yasuhiko Ikematsu, Ryoya Fukasaku, Momonari Kudo, Masaya Yasuda, Katsuyuki Takashima, and Kazuhiro Yokoyama. Hybrid meet-in-the-middle attacks for the isogeny

path-finding problem. In *Proceedings of the 7th ACM Workshop on ASIA Public-Key Cryptography–APKC 2020*, pages 36–44, 2020.

[10] D Jao, R Azarderakhsh, M Campagna, C Costello, L DeFeo, B Hess, A Jalali, B Koziel, B LaMacchia, P Longa, et al. SIKE: Supersingular isogeny key encapsulation. submission to the NIST standardization process on post-quantum cryptography, 2017.

[11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography–PQCrypto 2011*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[12] David Kohel. *Endomorphism rings of elliptic curves over finite fields.* PhD thesis, University of California at Berkeley, 1996.

[13] Dustin Moody, Gorjan Alagic, Daniel C Apon, David A Cooper, Quynh H Dang, John M Kelsey, Yi-Kai Liu, Carl A Miller, Rene C Peralta, Ray A Perlner, et al. Status report on the second round of the NIST post-quantum cryptography standardization process. *NISTIR 8309*, 2020.

[14] Joseph H Silverman. *Advanced topics in the arithmetic of elliptic curves*, volume 151 of *Graduate Texts in Mathematics.* Springer-Verlag New York, 1994.

[15] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics.* Springer Science & Business Media, second edition, 2009.

[16] Yasushi Takahashi, Momonari Kudo, Ryoya Fukasaku, Yasuhiko Ikematsu, Masaya Yasuda, and Kazuhiro Yokoyama. Algebraic approaches for solving isogeny problems of prime power degrees. *Journal of Mathematical Cryptology*, 15:31–44, 2021.

[17] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, 1966.

[18] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:238–241, 1971.

Appendix: Magma codes of the algebraic approach

In this appendix, we present a part of Magma codes of the algebraic approach for solving the isogeny path-finding problem of 3-power isogeny degrees. Please visit

`https://www2.math.kyushu-u.ac.jp/~fukasaku/software/Isogeny_Problem`

to obtain the full codes.

```
/*-------------------------------------------------------*/
/*** solve 3^e-degree isogeny problem                 ***/
/***       by computing 3-isogenous j-invariant sequence ***/
/*-------------------------------------------------------*/

/*** generate the ideals I, \tilde{I} ***/
function ideal_I_0(e0, F_p2, mod_poly_3, jE)
  /* Step 1: generate the polynomial ring */
    W := [];
    for i in [1..e0] do
      Wi := [];
      for j in [1..e0] do
        if i eq j then Wi[j] := 1;
        else Wi[j] := 0; end if;
      end for;
```

```
      W := Wi cat W;
    end for;
    Rj := PolynomialRing(F_p2,e0,"weight",W);
  /* Step 2: generate j-invariants */
    j := [];  j[1] := Rj!jE;
    for i in [1..e0] do
      j[1+i] := Rj.i;
    end for;
  /* Step 3: generate the ideal */
    G := [];
    for i in [1..e0] do
      G[i] := Rj!Evaluate(mod_poly_3,[j[i],j[i+1]]);
    end for;
    I := ideal<Rj|G>;
    return j, G, I, Rj;
end function;

/*** generate the ideal J, \tilde{J}  ***/
function ideal_J_0(e0, F_p2, mod_poly_3, g, R1)
  /* Step 1: generate a polynomial ring */
    W := [];
    for i in [1..e0] do
      Wi := [];
      for j in [1..e0] do
        if i eq j then Wi[j] := 1;
        else Wi[j] := 0; end if;
      end for;
      W := Wi cat W;
    end for;
    Rj := PolynomialRing(F_p2,e0,"weight",W);
  /* Step 2: generate j-invariants */
    j := [];
    for i in [1..e0] do
      j[i] := Rj.i;
    end for;
  /* Step 3: generate the ideal */
    G := [Rj!Evaluate(g,Rj.1)];
    for i in [1..e0-1] do
      G[i+1] := Rj!Evaluate(mod_poly_3,[j[i],j[i+1]]);
    end for;
    I := ideal<Rj|G>;
    return j, G, I, Rj;
end function;
function ideal_J_append(F_p2, mod_poly, g, g_tilde, R1, middle_ord)
    Rj := PolynomialRing(F_p2,2,middle_ord);
    j := [Rj.1,Rj.2];
    p  := Evaluate(g,Rj.1);
    p_tilde := Evaluate(g_tilde,Rj.2);
    m  := Evaluate(mod_poly, [Rj.1,Rj.2]);
    j := [Rj.1,Rj.2];
    G := [p,m,p_tilde];
    I := ideal<Rj|G>;
    return j, G, I, Rj;
end function;
```

```
/*** compute minimal polynomials ***/
function mini_pol_GB(e0, F_p2, R1, j, I, Rj)
  /* Step 0: generate a polynomial ring */
    MarkGroebner(I);
    W := [];
      for i in [1..(e0-1)] do
      Wi := [];
      for j in [1..e0] do
        if i eq j then Wi[j] := 1;
        else Wi[j] := 0; end if;
      end for;
      W := Wi cat W;
    end for;
    for i in [1..e0] do
      if i eq e0 then W[e0*(e0-1)+i] := 1;
      else W[e0*(e0-1)+i] := 0; end if;
    end for;
    RjN := PolynomialRing(F_p2,e0,"weight",W);
    I := ChangeOrder(I, RjN);
  /* Step 1: compute a GB of I */
    print "compute a GB of I ..."; print ""; print "";
    Subtimer1 := Cputime();
    G := GroebnerBasis(I);
    print ""; print "";
    print "GB computation time:", Cputime(Subtimer1);
  /* Step 2: compute the minimal polynomial of I w.r.t. j_{e0} */
    for i in [1..#G] do
      boolean,Gi_uni := IsUnivariate(G[i],e0);
      if boolean then
        print "non-sq-deg: ", Degree(R1!Gi_uni);
        g := SquarefreePart(R1!Gi_uni);
        print "sq-deg: ", Degree(g);    break i;
      end if;
      if i eq #G then
        error("Fail to compute forward poly.");
      end if;
    end for;
    print "the minimal polynomial: ", g;  return(g);
end function;
function append_GB(k, H, J, Rk, R1)
  /* Step 1: compute a GB of I w.r.t. j_1 >_{lex}...>_{lex} j_{e0} by FGLM */
    print "compute a GB of I w.r.t. j_1 <_{grevlex} j_{2} ...";
    print ""; print "";
    Subtimer1 := Cputime();
    G := GroebnerBasis(J);
    print ""; print ""; print "GB computation time:",Cputime(Subtimer1);
    print "GB: ", G;  return(G);
end function;

/*** 2-SECTION ***/
// mini_pol_2: a parameter "GB", skip_2: a parameter 0, gf_2: 0 or GF(p^2)
functionj_invariant_2_section(e,jE,jE_tilde:mini_pol_2:="GB",
skip_2:=0,gf_2:=0,cycle_2:=0,Al_2:=0)
```

```
  /* Step 1: generate the modular polynomials */
    if e lt 2 then error("Require e >= 2"); end if;
    if e mod 2 ne 0 then error("Require e is even"); end if;
    Timer := Cputime();
    if Type(gf_2) eq Type(0) then
      p := 2^250*3^159-1;
      F_p := ResidueClassRing(p);
      R1<t> := PolynomialRing(F_p);
      P := ideal<R1 | t^2+1>;
      F_p2<t> := quo<R1 | P>;
    else
      F_p2 := gf_2;
    end if;
    R1<Z> := PolynomialRing(F_p2);  R2<X,Y> := PolynomialRing(F_p2,2);
  /* Step 2: generate the ideal I, \tilde{I} */
    e0 := e div 2;
    if skip_2 eq 0 then
      mod_poly_3 := R2!ClassicalModularPolynomial(3);
      j, G, I, Rj := ideal_I_0(e0, F_p2, mod_poly_3, jE);
      j_tilde, G_tilde, I_tilde, Rj_tilde := ideal_I_0(e0, F_p2,
      mod_poly_3, jE_tilde);
    end if;
  /* Step 3: compute the minimal polynomials of I, \tilde{I} */
    print "I: ", I;  print "tilde{I}: ", I_tilde;
    if mini_pol_2 eq "GB" then
      g := mini_pol_GB(e0, F_p2, R1, j, I, Rj);
      g_tilde := mini_pol_GB(e0, F_p2, R1, j_tilde, I_tilde, Rj_tilde);
    end if;
  /* Step 2: compute the GCD of g, \tilde{g} */
    print "compute GCD(g,\tilde{g})...";
    Subtimer := Cputime();  gcd_mod := R1!Gcd(g,g_tilde);
    print "the GCD: ", gcd_mod;
    print "GCD computation time:",Cputime(Subtimer);
    /* Step 3: output a j-invariant */
    if Type(gf_2) eq Type(0) then boolean,J_half := HasRoot(gcd_mod,F_p);
    else                          boolean,J_half := HasRoot(gcd_mod,F_p2);
    end if;
    if boolean eq false then
      error("Fail to compute J as a root of GCD");
    end if;
    print "a", ((e-1) div 2)+2, "-th j-invariant: ", J_half;  print"";
    print "total time:",Cputime(Timer);  return J_half,((e-1) div 2)+2;
end function;

/*** 3-SECTION ***/
// mini_pol_3: "GB", skip_3/skip_mid_3: 0,  gf_3: 0 or GF(p^2), middle_ord_3:
a term order
function j_invariant_3_section(e,jE,jE_tilde:
         mini_pol_3:="GB",skip_3:=0,skip_mid_3:=0,gf_3:=0,
         middle_ord_3:="grevlex",cycle_3:=0,A1_3:=0)
  /* Step 0: test */
    if e lt 2 then error("Require e >= 2"); end if;
    if e mod 2 ne 0 then error("Require e is even"); end if;
    if e eq 2 then
```

```
    print "use j_invariant_2_section since e = 2";
    return j_invariant_2_section(e,jE,jE_tilde:
    mini_pol_2:=mini_pol_3,skip_2:=skip_3,Al_2:=Al_3);
  end if;
/* Step 1: generate the modular polynomials */
  Timer := Cputime();
  if Type(gf_3) eq Type(0) then
    p := 2^250*3^159-1;
    F_p := ResidueClassRing(p);
    R1<t> := PolynomialRing(F_p);
    P := ideal<R1 | t^2+1>;
    F_p2<t> := quo<R1 | P>;
  else  F_p2 := gf_3;  end if;
  R1<Z> := PolynomialRing(F_p2);  R2<X,Y> := PolynomialRing(F_p2,2);
  mod_poly_3 := R2!ClassicalModularPolynomial(3);
  mod_poly_9 := R2!ClassicalModularPolynomial(9);
  mod_poly_27:= R2!ClassicalModularPolynomial(27);
  mod_poly_81:= R2!list_mod_poly(81,F_p2);
/* Step 2: generate the ideal I, \tilde{I} */
  e0 := e div 2;
  if   skip_mid_3 eq 0 or e0 le 2 then e1 := e0 - 1;
  elif skip_mid_3 eq 1 or e0 le 3 then e1 := e0 - 2;
  elif skip_mid_3 eq 2 or e0 le 4 then e1 := e0 - 3;
  else                                 e1 := e0 - 4; end if;
  if skip_3 eq 0 or e0 le 1 then
    j, G, I, Rj := ideal_I_0(e0, F_p2, mod_poly_3, jE);
  end if;
  if skip_3 eq 0 or e1 le 1 then
    j_tilde, G_tilde, I_tilde, Rj_tilde := ideal_I_0(e1, F_p2, mod_poly_3, jE_tilde);
  end if;
  print "I: ", I;  print "tilde{I}: ", I_tilde;
/* Step 3: compute the minimal polynomials of I, \tilde{I} w.r.t. j_{e_0} */
  if mini_pol_3 eq "GB" then
    g := mini_pol_GB(e0, F_p2, R1, j, I, Rj);
    g_tilde := mini_pol_GB(e1, F_p2, R1, j_tilde, I_tilde, Rj_tilde);
  end if;
/* Step 4: compute the minimal polynomial w.r.t. e_0 */
/* Step 4-1: generate the ideal J */
  if skip_mid_3 eq 0 or ((e-1)-(e1-1))-e0 le 1 then
    k, H, J, Rk := ideal_J_append(F_p2, mod_poly_3,
    g, g_tilde, R1, middle_ord_3);
  elif skip_mid_3 eq 1 or ((e-1)-(e1-1))-e0 le 2 then
    k, H, J, Rk := ideal_J_append(F_p2, mod_poly_9,
    g, g_tilde, R1, middle_ord_3);
  elif skip_mid_3 eq 2 or ((e-1)-(e1-1))-e0 le 3 then
    k, H, J, Rk := ideal_J_append(F_p2, mod_poly_27,
    g, g_tilde, R1, middle_ord_3);
  else
    k, H, J, Rk := ideal_J_append(F_p2, mod_poly_81, g, g_tilde, R1, middle_ord_3);
  end if;
  print "J: ", J;
/* Step 4-2: compute the minimal polynomial h, \tilde{h} */
  print "compute a GB ...";  Subtimer := Cputime();
  GB := append_GB(k, H, J, Rk, R1);
```

```
  /* Step 4: output a j-invariant */
    if Type(gf_3) eq Type(0) then J_half := VarietySequence(ideal<Rk|GB>);
    else                          J_half := VarietySequence(ideal<Rk|GB>);
    end if;
    if J_half eq [] then error("Fail to compute J as a root of GB"); end if;
    print "a", ((e-1) div 2)+2, "-th j-invariant: ", J_half;
    print"";  print "total time:",Cputime(Timer);
    return J_half[1][1],((e-1) div 2)+2;
end function;

/*** 3-SECTION (Symmetric Version) ***/
// mini_pol_3_s: "GB", skip_3_s/skip_mid_3_s: 0, gf_3_s: 0 or GF(p^2), middle_ord_3_s:
a term order
function j_invariant_3_section_symmetric(e,jE,jE_tilde:
         mini_pol_3_s:="GB",skip_3_s:=0,skip_mid_3_s:=0,gf_3_s:=0,
         middle_ord_3_s:="grevlex",Al_3_s:=0)
  /* Step 0: test */
    if e lt 2 then error("Require e >= 2"); end if;
    if e mod 2 ne 0 then error("Require e is even"); end if;
    if e eq 2 then
      print "use j_invariant_2_section since e = 2";
      return j_invariant_2_section(e,jE,jE_tilde:mini_pol_2:=mini_pol_3_s,
      skip_2:=skip_3_s,Al_2:=Al_3_s);
    end if;
  /* Step 1: generate the modular polynomials */
    Timer := Cputime();
    if Type(gf_3_s) eq Type(0) then
      p := 2^250*3^159-1;
      F_p := ResidueClassRing(p);
      R1<t> := PolynomialRing(F_p);
      P := ideal<R1 | t^2+1>;
      F_p2<t> := quo<R1 | P>;
    else
      F_p2 := gf_3_s;
    end if;
    R1<Z> := PolynomialRing(F_p2);  R2<X,Y> := PolynomialRing(F_p2,2);
    mod_poly_3 := R2!ClassicalModularPolynomial(3);
    mod_poly_9 := R2!ClassicalModularPolynomial(9);
    mod_poly_27:= R2!ClassicalModularPolynomial(27);
    mod_poly_81:= R2!list_mod_poly(81,F_p2);
  /* Step 2: generate the ideal I, \tilde{I} */
    e0 := e div 2;
    if   skip_mid_3_s eq 0 or e0 le 1 then e1 := e0;     e2 := e1-1;
    elif skip_mid_3_s eq 1 or e0 le 2 then e1 := e0 - 1; e2 := e1;
    elif skip_mid_3_s eq 2 or e0 le 3 then e1 := e0 - 1; e2 := e1-1;
    else                                   e1 := e0 - 2; e2 := e1;   end if;
    if skip_3_s eq 0 or e1 le 1 then
      j, G, I, Rj := ideal_I_0(e1, F_p2, mod_poly_3, jE);
    end if;
    if skip_3_s eq 0 or e2 le 1 then
      j_tilde, G_tilde, I_tilde, Rj_tilde := ideal_I_0(e2, F_p2, mod_poly_3, jE_tilde);
    end if;
    print "I: ", I;  print "tilde{I}: ", I_tilde;
  /* Step 3: compute the minimal polynomials of I, \tilde{I} w.r.t. j_{e_0} */
```

```
    if mini_pol_3_s eq "GB" then
      g := mini_pol_GB(e1, F_p2, R1, j, I, Rj);
      g_tilde := mini_pol_GB(e2, F_p2, R1, j_tilde, I_tilde, Rj_tilde);
    end if;
  /* Step 4: compute the minimal polynomial w.r.t. e_0 */
  /* Step 4-1: generate the ideal J */
    if skip_mid_3_s eq 0 or (e-e2)-e1 le 1 then
      k, H, J, Rk := ideal_J_append(F_p2, mod_poly_3, g, g_tilde, R1, middle_ord_3_s);
    elif skip_mid_3_s eq 1 or (e-e2)-e1 le 2 then
      k, H, J, Rk := ideal_J_append(F_p2, mod_poly_9, g, g_tilde, R1, middle_ord_3_s);
    elif skip_mid_3_s eq 2 or (e-e2)-e1 le 3 then
      k, H, J, Rk := ideal_J_append(F_p2, mod_poly_27, g, g_tilde, R1, middle_ord_3_s);
    else
      k, H, J, Rk := ideal_J_append(F_p2, mod_poly_81, g, g_tilde, R1, middle_ord_3_s);
    end if;
    print "J: ", J;
  /* Step 4-2: compute the minimal polynomial h, \tilde{h} w.r.t. e_0 */
    print "compute a GB ...";  Subtimer := Cputime();
    // mini_pol w.r.t. Rk.1, where Rk = F_p2[j_1,j_2] and j_1 = j_{e_0}
    GB := append_GB(k, H, J, Rk, R1);
  /* Step 4: output a j-invariant */
    if Type(gf_3_s) eq Type(0) then J_half := VarietySequence(ideal<Rk|GB>);
    else                            J_half := VarietySequence(ideal<Rk|GB>);
    end if;
    if J_half eq [] then
      error("Fail to compute J as a root of GB");
    end if;
    print [e1+1,e-e2], "-th j-invariants: ", J_half;  print"";
    print "total time:",Cputime(Timer);  return J_half, [e1+1,e-e2];
end function;
```