International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 10, Number 2, pages 227–241, July 2020

A Performance Analysis and Evaluation of SIDH Applied Several Implementation-Friendly Quadratic Extension Fields

Yuki Nanjo[†], Masaaki Shirase[‡], Takuya Kusaka[†] and Yasuyuki Nogami[†]

[†]Okayama University, Tsushima–naka 3–1–1, Kita–ku, Okayama 700–8530, Japan.
[‡]Future University Hakodate, Kamedanakano-cho 116–2, Hakodate, Hokkaido 041–8655, Japan.

Received: February 13, 2020
Revised: April 24, 2020
Accepted: June 2, 2020
Communicated by Toru Nakanishi

Abstract

It is well-known that quadratic extension fields (QEFs) based on optimal extension fields (OEFs) are typically used for supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol. On the other hand, there is a possibility of the performance improvement of SIDH by employing other attractive choices of QEFs with efficient performing arithmetics which are based on all-one polynomial extension fields (AOPFs) and extension fields with normal basis representation (EFNs). Thus, the authors confirm that the applicability of the new candidates of QEFs for SIDH and evaluate SIDH applied the possible choices of QEFs. As a result of the experiment, the authors found that the performances of SIDH applied the QEFs based on AOPF and EFN are comparable to that of the previous QEF. Moreover, one of the QEFs based on EFN result in a new efficient implementation of the SIDH with SIDH-friendly prime given as $p = 2^{e_A}3^{e_B}f + 1$ where e_A, e_B and f are positive integers.

Keywords: Post-quantum cryptography, SIDH, Quadratic extension fields

1 Introduction

Background and motivation. Since a quantum computer that is capable of executing Shor's algorithm is one of the major threats for the cryptosystems based on the classical algorithms, post-quantum cryptography occupies a major place in the current research of security. In 2011, Jao and De Feo proposed a Diffie-Hellman key exchange protocol based on the difficulty of computing a kernel of isogenies between supersingular elliptic curves, which is called supersingular isogeny Diffie-Hellman (SIDH) [1]. Since the best-known algorithms against the SIDH have an exponential time complexity for both classical and quantum attackers at this time, SIDH is expected as one of the candidates of the post-quantum cryptosystems. Therefore, researchers actively work on the SIDH, however, there remain problems where high computational complexity and limitations of the supersingular elliptic curves.

To overcome the problems, the authors focus on quadratic extension fields (QEFs) by the following reasons: 1) Since SIDH requires arithmetic operations in the QEFs, the performance of the arithmetic operations in the QEFs might affect the performance of SIDH. 2) Moreover, since the range of the supersingular elliptic curves depends on the QEFs which restrict conditions of field characteristics, there is a possibility that the range can expand by changing the QEFs. In the context, the authors focus on the QEFs with efficient performing arithmetics as shown in the below.

(i) The QEFs based on optimal extension fields (OEFs) proposed by Bailey et al. [2] in which polynomial multiplication is implemented by using Karatsuba multiplication [3].
(ii) The QEFs based on all-one polynomial extension fields (AOPFs) proposed by Nogami et al. with an efficient multiplication algorithm named as cyclic vector multiplication algorithm (CVMA) in [4].
(iii) The QEFs based on extension fields with normal basis representation (EFNs) of which multiplication is efficiently implemented by the NTT method [5].

In the following, these implementation-friendly QEFs which are based on OEFs, AOPFs, and EFNs are denoted as F_{OEF}, F_{AOPF}, and F_{EFN}, respectively.

Since F_{OEF} is the most well-known QEFs with efficient performing arithmetics, the previous SIDH implementations such that [6, 7] employed F_{OEF} of which modular polynomial is given by an irreducible binomial $f(x) = x^2 + 1$, which results in the best performing arithmetics among F_{OEF}. However, the number of prime field additions required for the multiplication in F_{AOPF} and F_{EFN} is smaller than that of F_{OEF}. Thus, there is a possibility of the performance improvement of SIDH by exploiting F_{AOPF} and F_{EFN} instead of F_{OEF}. From the above reason, the authors try to investigate the computational complexity and execution time of SIDH applied these QEFs and provide a performance evaluation.

However, it is not clear whether the implementation-friendly QEFs can be applied for SIDH or not since the applicability of the QEFs typically depends on the condition of field characteristics p. The characteristic used for SIDH has a specific form given as $p = l_A^{e_A} l_B^{e_B} f \pm 1$, which is called a SIDH-friendly prime, where l_A and l_B are typically chosen as $l_A = 2$ and $l_B = 3$. Indeed, the previous choice of QEFs, i.e., F_{OEF} with the modular polynomial $f(x) = x^2 + 1$, restricts the field characteristic as $p \equiv 3 \pmod 4$ and is only available for $p = 2^{e_A} 3^{e_B} f - 1$. In contrast, a practical implementation of the SIDH with $p = 2^{e_A} 3^{e_B} f + 1$ which is comparable for that of $p = 2^{e_A} 3^{e_B} f - 1$ are not found with the view of the choice of QEFs. Thus, the authors also try to find the QEFs which enable the practical implementations for the SIDH with $p = 2^{e_A} 3^{e_B} f + 1$.

Our contributions. The major contributions of this research are given as follows:

1. The authors find that not only F_{OEF} and but also F_{AOPF} and F_{EFN} can be applied for the SIDH with $p = 2^{e_A} 3^{e_B} f - 1$. As for F_{EFN}, it is also available for the SIDH with $p = 2^{e_A} 3^{e_B} f + 1$.
2. The authors confirm the performance of the SIDH with $p = 2^{e_A} 3^{e_B} f \pm 1$ applied the QEFs by an implementation. The results of the experiment show that the performance of the SIDH with $p = 2^{e_A} 3^{e_B} f - 1$ is competitive between F_{AOPF}, F_{EFN}, and F_{OEF}. The authors also find that the SIDH with $p = 2^{e_A} 3^{e_B} f + 1$ applied F_{EFN} are almost competitive to the SIDH with $p = 2^{e_A} 3^{e_B} f - 1$ applied F_{OEF}.
3. As an additional contribution, the authors slightly improve the complexity to determine a curve coefficient of 3-isogenies by modifying a form of the coefficient compared with the previous result [6].

The differences from the original version. This work is extended from the authors' previous work [8] submitted in CANDAR'19. Although [8] present the experimental result of the SIDH with $p_{758-} = 2^{378} 3^{237} 17 - 1$ and $p_{759+} = 2^{378} 3^{236} 89 + 1$ which are not commonly used, this work provide new results of the SIDH with $p_{434-} = 2^{216} 3^{137} - 1$ and $p_{441+} = 2^{216} 3^{137} 139 + 1$ where p_{434-} is especially suggested for the specification of supersingular isogeny key encapsulation (SIKE) [9]. Moreover, this work analyzes the performance of SIDH not only form the performance of isogeny computations but also that of point multiplications required for the subgroup generation for a kernel of isogeny.

Organization. In the following, Sect. 2 overviews the necessary fundamentals of SIDH. Then, Sect. 3 describes the implementation-friendly QEFs with the applicability for SIDH. The performance analyses of SIDH are given in Sect. 4. Finally, Sect. 5 draws the conclusion.

2 Preliminaries

This section describes the fundamentals of Montgomery curves and isogenies and the details of SIDH protocol.

Notation. For a prime $p > 3$ and a positive integer m, let $\mathbb{F}_p$ and $\mathbb{F}_{p^m}$ denote a finite field with a characteristic p and its extension field of degree m, respectively. A polynomial ring in x defined over $\mathbb{F}_p$ is denoted as $\mathbb{F}_p[x]$. The calculation costs of a single multiplication, squaring, addition and shift operation in $\mathbb{F}_{p^m}$ are written as $\mathbf{M_m}$, $\mathbf{S_m}$, $\mathbf{a_m}$ and $\mathbf{h_m}$, respectively.

2.1 Montgomery curves

For a field K, a *Montgomery curve* E defined over K is an elliptic curve given as $E/K : by^2 = x^3 + ax^2 + x$ where a and b are coefficients in K satisfying $b \neq 0$ and $a^2 \neq 4$. The j-invariant of E is given as $j(E) = 256(a^2 - 3)^3/(a^2 - 4)$. A solution $(x, y) \in K^2$ of E is called a *rational point.* A set of the rational points including a point at infinity $\mathcal{O}_E$ is called a *group of K-rational points on E*, which is denoted as $E(K)$. For an arbitrary non-negative integer s and rational point $P \in E(K)$, a point multiplication endomorphism can be defined as follows:

$$[s] : P \mapsto \underbrace{P + P + \cdots + P}_{s \text{ times}}. \tag{1}$$

All the rational points can be represented in projective coordinates, i.e., $(X : Y : Z)$ assuming $x = X/Z, y = Y/Z$ with $Z \neq 0$, which a point at infinity becomes $\mathcal{O}_E = (0 : 1 : 0)$. In [10], Montgomery gave efficient formulas to compute the group low in projective coordinates without Y-coordinate by using a 2-to-1 mapping as shown in the below.

$$x : E \to E/\langle - \rangle, (X : Y : Z) \mapsto \begin{cases} (X : Z) & \text{if } Z \neq 0 \\ (1 : 0) & \text{if } Z = 0 \end{cases}, \tag{2}$$

where $-$ is a negation automorphism given as $- : (x, y) \mapsto (x, -y)$. A set of the projective rational points of $E/\langle - \rangle$ is denoted as $\mathbb{P}^1$. Since $-$ is commutative with $[s]$, a point multiplication $x(P) \mapsto x([s]P)$ can also be available in $\mathbb{P}^1$. The above operation only requires a coefficient a, which is typically taking as $(a \pm 2)/4$ for efficient formulas.

2.2 Isogenies

Let E and $\tilde{E}$ be elliptic curves over a finite field K. An isogeny $\phi : E \to \tilde{E}$ defined over K is a surjective morphism such that $\mathcal{O}_E \mapsto \mathcal{O}_{\tilde{E}}$, which induces a group homomorphism $E(\overline{K}) \to \tilde{E}(\overline{K})$. For all $(x, y) \in E(\overline{K})$, the image of (x, y) under ϕ is given as $\phi(x, y) = (r_1(x), r_2(x) \cdot y)$, where $r_1(x)$ and $r_2(x)$ are quotients of polynomials with coefficient in K. Note that every isogeny appears in this paper is separable. If a cyclic subgroup $G \subset E(K)$ is given, there is a unique isogeny $\phi : E(\overline{K}) \to \tilde{E}(\overline{K}) \cong E(\overline{K})/G$ with $\ker(\phi) = G$, which is called a *$\#G$-isogeny.* The isogeny ϕ and $\tilde{E}$ can be made explicit by using Vélu's formulas [11] once E and G are known.

A large-degree isogeny can be computed efficiently as a composition of low degree isogenies in Sect. 4.2.2 of [12]. Let R be a rational point on E with the order l^e, where l and e are positive integer with small l. Assuming $G = \langle R \rangle$, there exists a l^e-isogeny $\phi_R : E \to E/\langle R \rangle$. Then ϕ_R can be computed efficiently as a composition of e isogenies of degree l. Set $E_0 = E$ and $R_0 = R$ initially, iterate for $0 \leq i < e$ with an integer i, the operations are given as follows:

$$E_{i+1} = E_i/\langle [l^{e-i-1}]R_i \rangle, \phi_i : E_i \to E_{i+1}, R_{i+1} = \phi_i(R), \tag{3}$$

which results in $\phi_R = \phi_{e-1} \circ \cdots \circ \phi_1 \circ \phi_0$. An isogeny ϕ_i and curve E_{i+1} are computed by Vélu's formulas from the knowledge of E_i and $\langle [l^{e-i-1}]R_i \rangle$. The large-degree isogenies can be accelerated by finding an optimal path of a directed acyclic graph as described in FIGURE 2 of [12]. The path can be determined by the relative costs of point multiplication by l and l-isogeny evaluation.

2.3 Supersinglar isogeny Diffie-Hellman key exchange protocol

In the following, steps for SIDH between the two-person, Alice and Bob, are described.

Setup. Let p be a prime given as follows:

$$p = l_A^{e_A} l_B^{e_B} f \pm 1, \tag{4}$$

where l_A and l_B are two small distinct primes, e_A and e_B are two positive integers, and f is a cofactor. A prime of the above form is called *SIDH-friendly prime.* Let E be a supersingular elliptic curve defined over a QEF $\mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p \mp 1)^2$. And let P_A, Q_A, P_B, Q_B are rational points in $E(\mathbb{F}_{p^2})$ such that $\langle P_A, Q_A \rangle \cong \mathbb{Z}/l_A^{e_A}\mathbb{Z} \times \mathbb{Z}/l_A^{e_A}\mathbb{Z}$ and $\langle P_B, Q_B \rangle \cong \mathbb{Z}/l_B^{e_B}\mathbb{Z} \times \mathbb{Z}/l_B^{e_B}\mathbb{Z}$. A public parameter set of SIDH is given as $\{p, l_A, l_B, e_A, e_B, E, P_A, Q_A, P_B, Q_B\}$.

Key generation. Alice chooses a secret key as $s_A \in \mathbb{Z}/l_A^{e_A}\mathbb{Z}$ and computes a secret subgroup $G_A = \langle P_A + [s_A]Q_A \rangle$. Alice also computes a $l_A^{e_A}$-isogeny $\phi_A : E \to E_A \cong E/G_A$ and images $\phi_A(P_B)$ and $\phi_A(Q_B)$, and sets her public key $pk_A = \{E_A, \phi_A(P_B), \phi_A(Q_B)\}$. Similarly, Bob chooses a secret key $s_B \in \mathbb{Z}/l_B^{e_B}\mathbb{Z}$ and obtains his public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$ by computing a $l_B^{e_B}$-isogeny $\phi_B : E \to E_B \cong E/G_B$ with $G_B = \langle P_B + [s_B]Q_B \rangle$ and images $\phi_B(P_A)$ and $\phi_B(Q_A)$. Finally, they send their public key to each other. Note that the authors refers to the definition of the secret subgroup from [6] for an efficient implementation.

Shared secret. Alice computes a subgroup $G'_A = \langle \phi_B(P_A) + [s_A]\phi_B(Q_A) \rangle$ from the received Bob's public key. Then Alice computes a $l_A^{e_A}$-isogeny $\phi'_A : E_B \to E_{BA} \cong E_B/G'_A$ and obtains a shared key as a j-invariant $j(E_{BA})$. Bob also computes a $l_B^{e_B}$-isogeny $\phi'_B : E_A \to E_{AB} \cong E_A/G'_B$ with $G'_B = \langle \phi_A(P_B) + [s_B]\phi_A(Q_B) \rangle$ and obtains a shared key as $j(E_{AB})$. They can share the same j-invariant since $E_{BA} \cong E/\langle P_A + [s_A]Q_A, P_B + [s_B]Q_B \rangle \cong E_{AB}$, which means that E_{BA} and E_{AB} are isomorphic.

In the following, operations to compute a l^e-isogeny with images of some points in the key generation phase and l^e-isogeny in the shared secret phase are denoted as `keygen_iso` and `keyshare_iso`, respectively. An operation to compute a generator point of kernel subgroups of order l^e, i.e., $R = P + [k]Q$ with $P, Q \in E[l^e]$ and $k \in \mathbb{Z}/l^e\mathbb{Z}$ is denoted as `kernel_gen`. Note that these operations occupy almost all computational complexity of SIDH.

2.4 Projective operations for SIDH

Following the previous works [6, 12], this paper also works in the projective coordinates of the Montgomery curve. The projective point operations and isogeny formulas required for the typical SIDH with $p = l_A^{e_A} l_B^{e_B} f \pm 1$ where $l_A = 2$, $l_B = 3$, and $2 \mid e_A$ are summarized in the below.

Projective point operations. This paper uses the projective coordinates not only the points of the curve but also the curve coefficients since they are not fixed but moved in isogeny graphs. Thus, the constant term $(a-2)/4$ in the projective coordinates is denoted as $(A_{24} : C_{24})$. Assuming $x(P) = (X_P : Z_P)$, $x(Q) = (X_Q : Z_Q)$, and $x(Q-P) = (X_{Q-P} : Z_{Q-P})$, a point doubling operation `xDBL` : $(x(P), (a-2)/4) \mapsto x([2]P)$, a tripling operation `xTPL` : $(x(P), (a-2)/4) \mapsto x([3]P)$, and a point addition `xADD` : $(x(P), x(Q), x(Q-P)) \mapsto x(Q+P)$ are given as follows:

- Doubling operation (`xDBL`)

$$[2](X_P : Z_P) = (C_{24}(X_P + Z_P)^2(X_P - Z_P)^2 : 4X_P Z_P(C_{24}(X_P + Z_P)^2 + 4A_{24}X_P Z_P). \tag{5}$$

- Tripling operation (`xTPL`)

$$\begin{aligned}[3](X_P : Z_P) = (&X_P(16A_{24}X_P Z_P^3 - C_{24}(X_P - 3Z_P)(X_P + Z_P)^3)^2 : \\ &Z_P(16A_{24}X_P^3 Z_P + C_{24}(3X_P - Z_P)(X_P + Z_P)^3)^2).\end{aligned} \tag{6}$$

- Addition operation (`xADD`)

$$\begin{aligned}(X_Q : Z_Q) + (X_P : Z_P) = (&Z_{Q-P}((X_Q - Z_Q)(X_P + Z_P) + (X_Q + Z_Q)(X_P - Z_P))^2 : \\ &X_{Q-P}((X_Q - Z_Q)(X_P + Z_P) - (X_Q + Z_Q)(X_P - Z_P))^2).\end{aligned} \tag{7}$$

Table 1: The calculation costs of the projective operations for SIDH.

Operation/ from	Input(s) type(s)	Output(s) type(s)	Operations $\mathbf{M_2}$	$\mathbf{S_2}$	$\mathbf{a_2}$	$\mathbf{h_2}$
`xDBL` [10]	$x(P), A_{24}, C_{24}$ $\mathbb{P}^1 \times \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$	$x([2]P)$ $\mathbb{P}^1$	4	2	4	-
`xTPL` App. A in [13]	$x(P), A_{24}, K_{24}$ $\mathbb{P}^1 \times \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$	$x([3]P)$ $\mathbb{P}^1$	7	5	10	-
`xDBLADD` [6]	$x(P), x(Q), x(Q-P), \frac{a+2}{4}$ $\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{F}_{p^2}$	$x([2]P), x(Q-P)$ $\mathbb{P}^1$	7	4	8	-
`3_iso_curve` App. A in [13]	$x(P_3)$ $\mathbb{P}^1$	$\boldsymbol{c}_2, A'_{24}, C'_{24}$ $(\mathbb{F}_{p^2})^2 \times \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$	2	3	14	-
`3_iso_curve*` This work	$x(P_3)$ $\mathbb{P}^1$	$\boldsymbol{c}_2, A'_{24}, K'_{24}$ $(\mathbb{F}_{p^2})^2 \times \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$	2	3	13	-
`3_iso_point` App. A in [13]	$(\boldsymbol{c}_2, x(P))$ $(\mathbb{F}_{p^2})^2 \times \mathbb{P}^1$	$x(\phi(P))$ $\mathbb{P}^1$	4	2	4	-
`4_iso_curve` App. A in [13]	$x(P_4)$ $\mathbb{P}^1$	$\boldsymbol{c}_3, A'_{24}, C'_{24}$ $(\mathbb{F}_{p^2})^3 \times \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$	-	4	3	1
`4_iso_point` App. A in [13]	$(\boldsymbol{c}_3, x(P))$ $(\mathbb{F}_{p^2})^3 \times \mathbb{P}^1$	$x(\phi(P))$ $\mathbb{P}^1$	6	2	6	-

According to [13], `xTPL` can be computed efficiently by taking a coefficient as $(A_{24}, K_{24} = A_{24} + C_{24})$. The operations `xDBL` and `xTPL` are used for the computations of the points of order 2 and 3 required for 2- or 4-isogeny and 3-isogeny computations, respectively. Although `xADD` is typically does not exploited for SIDH, an operation to compute `xDBL` and `xADD` simultaneously, i.e., `xDBLADD` : $(x(P), x(Q), x(Q-P), (a+2)/4) \mapsto (x([2]P), x(Q-P))$ is used for the SIDH operation `kernel_gen` as described in [12, 14].

Projective isogenies computation. An isogeny on Montgomery curves can be computed in $\mathbb{P}^1$, i.e. $x(P) \mapsto x(\phi(P))$, since a x-coordinate of $\phi(P)$ is determined without y-coordinate of a point P. As for the computation of the 2^{e_A}-isogeny with $2 \mid e_A$, the authors employ not 2-isogenies but 4-isogenies, since a performance using 4-isogenies to be significantly faster than that of 2-isogenies. Let $(X'_P : Z'_P)$ and $(A'_{24} : C'_{24})$ be an image of $(X_P : Z_P)$ and coefficient of an elliptic curve given by ϕ, respectively. Assuming $(X_3 : Z_3)$ and $(X_4 : Z_4)$ denote rational points of order 3 and 4, the isogenies of degree 3 and 4 are computed as follows:

- 3-isogeny operations (`3_iso_curve`, `3_iso_point`)

$$(A'_{24} : C'_{24}) = ((X_3 + Z_3)(Z_3 - 3X_3)^3 : 16X_3Z_3^3), \tag{8}$$
$$(X'_P : Z'_P) = (X_P(X_3X_P - Z_3Z_P)^2 : Z_P(Z_3X_P - X_3Z_P)^2). \tag{9}$$

- 4-isogeny operations (`4_iso_curve`, `4_iso_point`)

$$(A'_{24} : C'_{24}) = (X_4^4 - Z_4^4 : Z_4^4), \tag{10}$$
$$\begin{aligned}(X'_P : Z'_P) = (&X_P(2X_4Z_4Z_P - X_P(X_4^2 + Z_4^2))(X_4X_P - Z_4Z_P)^2 : \\ &Z_P(2X_4Z_4X_P - Z_P(X_4^2 + Z_4^2))(Z_4X_P - X_4Z_P)).\end{aligned} \tag{11}$$

The authors modify `3_iso_curve` as `3_iso_curve*` in App. A by using $K_{24} = A_{24} + C_{24}$ which results in a reduction of single $\mathbb{F}_{p^2}$-addition. However, the curve determination operations occupy low computational complexity of the isogenies computation comparing with the other operations such that point multiplications and image computations.

The calculation costs and I/O specifications of `xDBL`, `xTPL`, `xDBLADD`, `3_iso_curve`, `3_iso_point`, `3_iso_curve*`, `4_iso_curve`, and `4_iso_point` are summarized in Table 1. The notations $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ are common variables for the curve determination and point evaluation.

3 Implementation-Friendly QEFs Applicable for SIDH

In this section, the several attractive QEFs with efficient performing arithmetics are described. This section also presents the applicability of these QEFs for the SIDH with SIDH-friendly primes given as $p = 2^{e_A}3^{e_B}f \pm 1$.

3.1 Implementation-friendly QEFs

A QEF applied for SIDH has to be particularly efficient since the efficiency of SIDH strongly depends on the efficiency of arithmetics in the QEF. Thus, the authors construct implementation-friendly QEFs by exploiting the existing construction methods of extension fields with efficient performing arithmetics as follows:

(i) QEF based on OEFs. Bailey and Paar proposed OEFs [2] which are defined by using irreducible binomials. An OEF of degree m of $\mathbb{F}_p$ is defined as $K_{\text{OEF}} = \mathbb{F}_p[\omega]/(g_{\text{OEF}}(\omega) = \omega^m - c_0) \cong \mathbb{F}_{p^m}$, where $g_{\text{OEF}}(x)$ is an irreducible binomial of degree m defined over $\mathbb{F}_p$ of which a root is ω. An arbitrary element $a \in K_{\text{OEF}}$ is represented as $a = a_0 + a_1\omega + \cdots + a_{m-1}\omega^{m-1}$ where $a_i \in \mathbb{F}_p$ with $i \in \{0, 1, \ldots, m-1\}$ and $\{1, \omega, \ldots, \omega^{m-1}\}$ is a basis that is classified into a *polynomial basis*. For an arithmetic operation in K_{OEF}, several efficient multiplication algorithms such that Karatsuba multiplication [3] and Toom-Cook multiplication [15,16] are available. Although the field characteristics of original OEF are pseudo-Mersenne primes, it is possible to extend for the general characteristics including the SIDH-friendly primes. Thus, a QEF with $p = l_A^{e_A} l_B^{e_B} f \pm 1$ can be defined as follows:

$$F_{\text{OEF}} = \mathbb{F}_p[\alpha]/(f_{\text{OEF}}(\alpha) = \alpha^2 - c_0) \cong \mathbb{F}_{p^2}, \tag{12}$$

where $f_{\text{OEF}}(x)$ is an irreducible polynomial defined over $\mathbb{F}_p$ with a SIDH-friendly characteristic of which a root is α. The small c_0 typically results in an efficient performing arithmetics. Note that the choice of $c_0 = -1$ results in the best performing arithmetics among F_{OEF}.

(ii) QEF based on AOPFs. Nogami et al. [4] proposed other attractive extension fields which is AOPFs. An AOPF of degree m of $\mathbb{F}_p$ is defined as $K_{\text{AOPF}} = \mathbb{F}_p[\upsilon]/(g_{\text{AOPF}}(\upsilon) = (\upsilon^r - 1)/(\upsilon - 1)) \cong \mathbb{F}_{p^m}$, where $r = m + 1$ is a prime and $g_{\text{AOPF}}(x)$ is an irreducible all-one polynomial of degree m defined over $\mathbb{F}_p$ with a root υ. An arbitrary element $a \in K_{\text{AOPF}}$ is represented as $a = a_0\upsilon + a_1\upsilon^2 + \cdots + a_{m-1}\upsilon^m$ where $a_i \in \mathbb{F}_p$ with $i \in \{0, 1, \ldots, m-1\}$ and $\{\upsilon, \upsilon^2, \ldots, \upsilon^m\}$ is a basis which is classified into a *optimal normal basis* [17]. For the arithmetic operation in K_{AOPF}, an efficient multiplication algorithm named as CVMA [4] is available. Since AOPFs can also be extended for extension fields with SIDH-friendly characteristics, the authors consider to construct QEFs with $p = l_A^{e_A} l_B^{e_B} f \pm 1$ as follows:

$$F_{\text{AOPF}} = \mathbb{F}_p[\beta]/(f_{\text{AOPF}}(\beta) = \beta^2 + \beta + 1) \cong \mathbb{F}_{p^2}, \tag{13}$$

where $f_{\text{AOPF}}(x)$ is an irreducible polynomial defined over $\mathbb{F}_p$ with a SIDH-friendly characteristic of which a root is β.

(iii) QEF based on EFNs. There exist extension fields of which arbitrary elements are represented by using a basis classified into a *normal basis*, which are called as EFNs in this paper. An EFN of degree m of $\mathbb{F}_p$ is defined as $K_{\text{EFN}} = \mathbb{F}_p[\nu]/(g_{\text{EFN}}(\nu) = \nu^m + c_{m-1}\nu^{m-1} + \cdots + c_0)) \cong \mathbb{F}_{p^m}$ where $g_{\text{EFN}}(x)$ is an irreducible polynomial with non-zero trace of which a root is ν. An arbitrary element $a \in K_{\text{EFN}}$ is represented as $a = a_0\nu + a_1\nu^2 + \cdots + a_{m-1}\nu^m$ where $a_i \in \mathbb{F}_p$ with $i \in \{0, 1, \ldots, m-1\}$ and $\{\nu, \nu^p, \ldots, \nu^{p^{m-1}}\}$ is a normal basis. The EFNs are efficiently implemented by using the NTT method [5]. From the above, a QEF with $p = l_A^{e_A} l_B^{e_B} f \pm 1$ can also be defined as

$$F_{\text{EFN}} = \mathbb{F}_p[\gamma]/(f_{\text{EFN}}(\gamma) = \gamma^2 + c_1\gamma + c_0) \cong \mathbb{F}_{p^2}, \tag{14}$$

where $f_{\text{EFN}}(x)$ is an irreducible polynomial with $c_1 \neq 0$ defined over $\mathbb{F}_p$ with a SIDH-friendly characteristic and γ is a root of $f_{\text{EFN}}(x)$.

Table 2: The calculation costs of arithmetic operations in the implementation-friendly QEFs.

QEFs	Multiplication				Squaring			
	$\mathbf{M_1}$	$\mathbf{S_1}$	$\mathbf{a_1}$	$\mathbf{h_1}$	$\mathbf{M_1}$	$\mathbf{S_1}$	$\mathbf{a_1}$	$\mathbf{h_1}$
`OEF_x2+1`	3	-	5	-	2	-	3	-
`OEF_x2+2`	3	-	6	-	2	-	5	-
`OEF_x2-2`	3	-	5	-	2	-	5	-
`OEF_x2+3`	3	-	5	1	2	-	3	2
`OEF_x2-3`	3	-	6	-	2	-	5	-
`OEF_x2+4`	3	-	5	1	2	-	5	1
`OEF_x2+5`	3	-	6	1	2	-	4	2
`OEF_x2-5`	3	-	5	1	2	-	4	2
`AOPF_x2+x+1`	3	-	4	-	2	-	4	-
`EFN_x2-x+1`	3	-	4	-	2	-	4	-
`EFN_x2-x-1`	3	-	4	-	-	3	3	-

3.2 The calculation costs of arithmetic operations in implementation-friendly QEFs

In the context, the authors consider the following QEFs which are classified into the implementation-friendly QEFs described in Sect 3.1.

`OEF_x2+1` : $F_{\mathrm{OEF1}} = \mathbb{F}_p[\alpha_1]/(\alpha_1^2+1)$, `OEF_x2+2` : $F_{\mathrm{OEF2}} = \mathbb{F}_p[\alpha_2]/(\alpha_2^2+2)$,
`OEF_x2-2` : $F_{\mathrm{OEF3}} = \mathbb{F}_p[\alpha_3]/(\alpha_3^2-2)$, `OEF_x2+3` : $F_{\mathrm{OEF4}} = \mathbb{F}_p[\alpha_4]/(\alpha_4^2+3)$,
`OEF_x2-3` : $F_{\mathrm{OEF5}} = \mathbb{F}_p[\alpha_5]/(\alpha_5^2-3)$, `OEF_x2+4` : $F_{\mathrm{OEF6}} = \mathbb{F}_p[\alpha_6]/(\alpha_6^2+4)$,
`OEF_x2+5` : $F_{\mathrm{OEF7}} = \mathbb{F}_p[\alpha_7]/(\alpha_7^2+5)$, `OEF_x2-5` : $F_{\mathrm{OEF8}} = \mathbb{F}_p[\alpha_8]/(\alpha_8^2-5)$,
`AOPF_x2+x+1` : $F_{\mathrm{AOPF1}} = \mathbb{F}_p[\beta_1]/(\beta_1^2+\beta_1+1)$,
`EFN_x2-x+1` : $F_{\mathrm{EFN1}} = \mathbb{F}_p[\gamma_1]/(\gamma_1^2-\gamma_1+1)$, `EFN_x2-x-1` : $F_{\mathrm{EFN2}} = \mathbb{F}_p[\gamma_2]/(\gamma_2^2-\gamma_2-1)$,

where α_i, β_j, and γ_k with $i \in \{1, 2, \ldots, 8\}$, $j \in \{1\}$, and $k \in \{1, 2\}$ are roots of modular polynomials. In the following, the QEFs based on some extension fields are called as `[field_name]_[polynomial]`, e.g., a QEF based on OEFs given by a modular polynomial $f(x) = x^2 + 1$ is denoted as `OEF_x2+1`. Note that `OEF_x2+1` is employed for the previous SIDH implementations. The details of the operation algorithms for `OEF_x2+1`, `OEF_x2-5`, `AOPF_x2+x+1`, `EFN_x2-x+1`, and `EFN_x2-x-1` are especially presented in App. B.

The calculation costs of multiplication and squaring in the implementation-friendly QEFs are given in Table 2. From the table, it is found that `OEF_x2+1` is the best performing arithmetic among the QEFs based on OEFs. In contrast, one $\mathbb{F}_p$-addition required for the multiplications in `AOPF_x2+x+1`, `EFN_x2-x+1`, and `EFN_x2-x-1` is reduced compared with that of `OEF_x2+1`. However, one $\mathbb{F}_p$-addition required for squarings in `AOPF_x2+x+1` and `EFN_x2-x+1` is increased than that of `OEF_x2+1`, which is a degradation. As for the squaring in `EFN_x2-x-1`, two $\mathbb{F}_p$-multiplications are replaced with three $\mathbb{F}_p$-squarings from that of `OEF_x2+1`. According to Table 1, since the $\mathbb{F}_{p^2}$-multiplication is more often required for the SIDH operations than $\mathbb{F}_{p^2}$-squarings, if it is possible to apply `AOPF_x2+x+1`, `EFN_x2-x+1`, and `EFN_x2-x-1` for SIDH, the performance of SIDH might be competitive to or rather better than that of `OEF_x2+1`.

3.3 Applicability of implementation-friendly QEFs for SIDH

Since there exist restrictions of field characteristics from the irreducibility of modular polynomials of the QEFs, not all SIDH-friendly characteristic results in the implementation-friendly QEFs. In the following, the authors consider the applicability of these QEFs for the SIDH with the typical

Table 3: Applicability of QEFs for the typical SIDH.

QEFs	Applicability	
	$p = 2^{e_A}3^{e_B}f - 1$	$p = 2^{e_A}3^{e_B}f + 1$
`OEF_x2+1`	✓	X
`OEF_x2+2`	✓	X
`OEF_x2-2`	X	X
`OEF_x2+3`	X	X
`OEF_x2-3`	✓	X
`OEF_x2+4`	✓	X
`OEF_x2+5`	✓**	✓*
`OEF_x2-5`	✓*	✓*
`AOPF_x2+x+1`	✓	X
`EFN_x2-x+1`	✓	X
`EFN_x2-x-1`	✓*	✓*

*If only a SIDH-friendly prime satisfies $p \equiv 2, 3 \pmod 5$

**If only a SIDH-friendly prime satisfies $p \equiv 1, 4 \pmod 5$

SIDH-friendly prime given as $p = 2^{e_A}3^{e_B}f \pm 1$. Firstly the authors provide a lemma associated with the conditions of the irreducibility of modular polynomials of QEFs.

Lemma 1. The field characteristic p has to satisfy the following conditions to exploit the certain implementation-friendly QEFs.

`OEF_x2+1` : $p \equiv 3 \pmod 4$, `OEF_x2+2` : $p \equiv 5, 7 \pmod 8$,
`OEF_x2-2` : $p \equiv 3, 5 \pmod 8$, `OEF_x2+3` : $p \equiv 2 \pmod 3$,
`OEF_x2-3` : $p \equiv 5, 7 \pmod{12}$, `OEF_x2+4` : $p \equiv 3 \pmod 4$,
`OEF_x2+5` : $p \equiv 11, 13, 17, 19 \pmod{20}$, `OEF_x2-5` : $p \equiv 2, 3 \pmod 5$,
`AOPF_x2+x+1` : $p \equiv 2 \pmod 3$,
`EFN_x2-x+1` : $p \equiv 2 \pmod 3$, `EFN_x2-x-1` : $p \equiv 2, 3 \pmod 5$.

Proof. According to [18], to construct the QEFs, a modular polynomial of QEFs, i.e., $f(x) = x^2 + c_1x + c_0$ with $c_0, c_1 \in \mathbb{F}_p$, has to be irreducible over $\mathbb{F}_p$. The irreducibility of $f(x)$ depends on the quadratic residue properties of the discriminant $D = c_1^2 - 4c_0$ since a root of $f(x)$ is given as $(-c_1 \pm \sqrt{D})/2$. If D is a quadratic non-residue in $\mathbb{F}_p$, the polynomial becomes irreducible. For `OEF_x2+1`, `OEF_x2+2`, `OEF_x2-2`, `OEF_x2+3`, `OEF_x2-3`, `OEF_x2+4`, `OEF_x2+5`, `OEF_x2-5`, `AOPF_x2+x+1`, `EFN_x2-x+1`, and `EFN_x2-x-1`, the discriminants are given as $D = -4$, -8, 8, -12, 12, -16, -20, 20, -3, -3, and 5, respectively. Applying the properties of the Legendre symbol described in [19], the restriction of the characteristic for the certain discriminant can be uniquely obtained as follows: $(\frac{-4}{p}) = (\frac{-16}{p}) = -1 \Leftrightarrow p \equiv 3 \pmod 4$, $(\frac{-8}{p}) = -1 \Leftrightarrow p \equiv 5, 7 \pmod{12}$, $(\frac{8}{p}) = -1 \Leftrightarrow p \equiv 3, 5 \pmod 8$, $(\frac{-12}{p}) = (\frac{-3}{p}) = -1 \Leftrightarrow p \equiv 2 \pmod 3$, $(\frac{12}{p}) = -1 \Leftrightarrow p \equiv 5, 7 \pmod{12}$, $(\frac{-20}{p}) = -1 \Leftrightarrow p \equiv 11, 13, 17, 19 \pmod{20}$, and $(\frac{20}{p}) = (\frac{5}{p}) = -1 \Leftrightarrow p \equiv 2, 3 \pmod 5$. Thus, the restrictions to apply the QEFs are obtained as shown in the lemma. □

The SIDH-friendly prime given as $p = 2^{e_A}3^{e_B}f \pm 1$ are clearly satisfy the condition $p \equiv \pm 1 \pmod{2^{e_A}}$ and $p \equiv \pm 1 \pmod{3^{e_B}}$, respectively. When comparing to these restrictions to exploit the QEFs given in Lemma 1, the applicability of QEFs for the SIDH with $p = 2^{e_A}3^{e_B}f \pm 1$ is obtained as shown in Table 3 where ✓ and X denote applicable and inapplicable, respectively.

From Table 3, the new candidates of QEFs such that `AOPF_x2+x+1` and `EFN_x2-x+1` can be available for the SIDH with $p = 2^{e_A}3^{e_B}f - 1$. Moreover, if the primes satisfy $p \equiv 2, 3 \pmod 5$, `EFN_x2-x-1` can also be applied not only for $p = 2^{e_A}3^{e_B}f - 1$ but also for $p = 2^{e_A}3^{e_B}f + 1$ which have not so much choices of QEFs based on OEFs. Although the previous SIDH implementation does not focus on $p = 2^{e_A}3^{e_B}f + 1$, there is a possibility that the SIDH with such the prime also results in an efficient implementation. Note that the authors consider that the sign of constant term of the SIDH-friendly prime might be not affect to the performance of the modular reduction described in Sect. 5 of [6] which is based on Montgomery reduction formula [20]: assuming $p = 2^{e_A}3^{e_B}f \pm 1$ and R is slightly larger than the size of p given as $R = 2^m$ with an integer m, one can compute the Montgomery residue $c = aR^{-1} (\mathrm{mod}\ p)$ for an input $a < pR$ as $c = (a + (aM'(\mathrm{mod}\ R))p)/R = (a \pm aM'(\mathrm{mod}R))/R + ((p \mp 1)(aM'(\mathrm{mod}R))) = (a \pm aM'(\mathrm{mod}R))/R + (2^{e_A}3^{e_B}f(aM'(\mathrm{mod}R)))$ where $M' = -p^{-1}(\mathrm{mod}R)$.

4 Performance Comparison of SIDH between Implementation-friendly QEFs

The authors pick up the four implementation-friendly QEFs, i.e., `OEF_x2+1`, `OEF_x2-5`, `AOPF_x2+x+1`, and `EFN_x2-x-1`, and compare the performance of the operations `keygen_iso`, `keyshare_iso` and `ker_gen` which occupy almost all computational complexity of SIDH. The authors also confirm the performance of the SIDH with the both SIDH-friendly prime $p = 2^{e_A}3^{e_B}f - 1$ and $p = 2^{e_A}3^{e_B}f + 1$.

4.1 Assumptions

In the following, the authors provide the parameter setting and environment of the experiments.

Parameters setup. The authors choose the SIDH-friendly primes satisfying $p \equiv 2, 3 \pmod 5$, which can construct various implementation-friendly QEFs. The primes which can ensure the quantum security at the 128-bit levels are given as follows:

$$p_{434-} = 2^{216}3^{137} - 1, \tag{16}$$

$$p_{441+} = 2^{216}3^{137}139 + 1. \tag{17}$$

where the size of the prime is one of 434-bit and 441-bit. The prime p_{434-} is suggested in Chap. 1.6.1 of the specification of SIKE [9] and p_{441+} is found by this work. Note that the authors consider that the proposed parameter p_{441+} can also ensure the 128-bit security level since e_A and e_B which are parameterized the size of the kernel of isogenies are the same size as p_{434-} ones.

The authors use supersingular elliptic curves of Montgomery form defined over $\mathbb{F}_{p^2}$ of which orders are $(p + 1)^2$ and $(p - 1)^2$ for the prime p_{434-} and p_{441+}, respectively. For p_{434-}, the supersingular elliptic curve is given as $E/\mathbb{F}_p : y^2 = x^3 + 6x^2 + x$. For p_{441+}, the curve can be found by using a quadratic twist as described in App. C.

Experimental environment. To evaluate the performance of the SIDH applied the implementation-friendly QEFs, the authors implemented the protocol by C language. In the implementation, the big integer arithmetics are implemented by using `mpz_t` data type of GMP library [21]. The software is compiled with GCC 8.3.0 with the option `-O2 -march=native`, and is executed on 3.50GHz Intel(R) Core(TM) i7-7567U CPU running macOS High Sierra version 10.13.6.

The four categories of arithmetic functions of GMP which are `mpz_mul`, `mpz_add`/ `mpz_sub`, `mpz_mul_2exp`/ `mpz_tdiv_q_2exp`, `mpz_invert`, `mpz_mod`, and `mpz_set` are employed in the software. The categories are referred as `mul`, `add`, `shift`, `mod`, and `set` respectively. If `mpz_mul` has the same operands, it is denoted as the sixth category `sqr`. To minimize the number of function calls of `mod` which has one of the highest computational complexity among the categories, the authors allow the operands with twice size of characteristic for `add`. The size of the operand(s) is denoted as a subscript of the category's name, e.g., `mpz_mul` with s-bit operands is denoted as $\mathtt{mul}_s$.

The weight of these operation categories with the specific size of operand(s) used for the implementation is given in Table. 4. The weight are derived from one hundred million trials of execution time excluding the overhead on this environment. Unlike `mul`, `sqr`, and `mod`, the differences of the operation weight of `add`, `shift`, and `set` between p_{434-} and p_{441+} are invisible since these operations are low computational complexity.

Table 4: Weight of the operation categories employed in the implementation of SIDH.

$\log_2\lfloor s \rfloor$	$\mathtt{mul}_s$	$\mathtt{sqr}_s$	$\mathtt{add}_s$	$\mathtt{add}_{2s}$	$\mathtt{shift}_s$	$\mathtt{mod}_{2s}$	$\mathtt{set}_s$
434	5.12	3.46	1.00	1.14	1.04	16.4	0.63
441	5.13	3.47	1.00	1.14	1.04	16.8	0.63

Optimization. All arithmetics are performed on Montgomery curves and applied the optimization proposed in [6, 12] as described in Sect. 2. The authors refer Sect. 4.2.2 in [12] and find the optimal paths of computing 4^{108}- and 3^{137}-isogenies from the ratio of a single point multiplication and isogeny evaluation. The ratio is derived from the computational complexities of these operations which are calculated by the sum of the number of operation categories multiplied by the weight given in Table 4. From the optimal paths, the authors find that the numbers of operations `xDBL` and `4_iso_point` for computing 4^{108}-isogeny are specifically given as 666 and 405 for all the QEFs with a certain characteristic used in this implementation, respectively. Similarly, the numbers of operations `xTPL` and `3_iso_point` required for computing 4^{108}-isogeny are also specifically given as 407 and 597, respectively. Note that this implementation does not adopt the Montgomery reduction described in Sect. 3.3 since the performance of that of p_{434-} and p_{441-} are might be competitive and the purpose of the experiment is not providing very efficient implementation but a performance comparison of SIDH between the QEFs.

Evaluation. The authors measure the number of function calls required for the SIDH operations, i.e., `keygen_iso`, `keyshare_iso`, and `kernel_gen`, which occupy almost all portion of complexity of SIDH. Since the number of function calls of `kernel_gen` typically depends on the secret key, the authors calculate the average of the result of 1,000 random secret keys. The authors also calculate the computational complexity of the SIDH operations by the sum of the numbers of the function calls multiplied by the weight of the operation categories. Besides, average execution times of 100,000 trials of the operations are measured. Note that the measurement is performed by repeating the operations for 1,000 random secret keys 100 times.

4.2 Results and analyses

Tables 5 and 6 show that the numbers of the function calls of the operations (a) Alice's `keygen_iso`, (b) Bob's `keygen_iso`, (c) Alice's `keyshare_iso`, (d) Bob's `keyshare_iso`. (e) Alice's `kernel_gen`, and (f) Bob's `kernel_gen` for the primes p_{434-} and p_{441+}, respectively. The tables also involve computational complexity and average execution time. Figs. 1 and 2 also provide the results of the computational complexity and execution time for p_{434-} and p_{441+}. The details of the results and their analyses are described in the below.

From Table 5 and Fig. 1, the performance of the SIDH operations with p_{434-} applied `AOPF_x2+x+1` and `EFN_x2-x-1` are competitive to that of `OEF_x2+1` which is exploited for the previous implementations. The results are caused by the complexities of the multiplication and squaring in the QEFs as described in Sect. 3.2. Moreover, `EFN_x2-x-1` can achieve more 1% improvement than that of `OEF_x2+1` since the computational complexity of three $\mathbb{F}_p$-squarings is lower than that of two $\mathbb{F}_p$-multiplications which results in more efficient performing squaring in `EFN_x2-x-1` than that of `OEF_x2+1`. Therefore, the performance improvement for the entire SIDH can be expected by using `AOPF_x2+x+1` or `EFN_x2-x-1` as a replacement for `OEF_x2+1`. Since the calculation costs of arithmetic operations in `EFN_x2+x-1` are exactly the same as `AOPF_x2+x+1`, `EFN_x2+x-1` is yet another candidate of the replacement. However, the results of the execution time with `OEF_x2+1` is slightly better than that of `AOPF_x2+x+1` in spite of the reduction of the complexity. The authors confirm the software by GNU profiler and find that the number of function calls of the operations applied `OEF_x2+1` and `AOPF_x2+x+1` is exactly correct, however, the execution time of single $\mathtt{add}_{434}$ of `AOPF_x2+1` is strangely slower than that of `OEF_x2+1`. At this time, the authors consider that it might come from the effects of cache and parallel processing.

The results Table 5 and Fig. 1 also show that the performance of the SIDH operations applied `OEF_x2-5` compares unfavorably to `OEF_x2+1`. Thus, such the QEF should be kept away from the

Table 5: The number of function calls, computational complexity and execution time of the SIDH operations (a) Alice's `keygen_iso`, (b) Bob's `keygen_iso`, (c) Alice's `keyshare_iso`, (d) Bob's `keyshare_iso`. (e) Alice's `kernel_gen`, and (f) Bob's `kernel_gen` with p_{434-}.

QEFs	Operation	Function calls							Complexity	Time [ms]
		$\texttt{mul}_{434}$	$\texttt{sqr}_{434}$	$\texttt{add}_{434}$	$\texttt{add}_{868}$	$\texttt{shift}_{434}$	$\texttt{mod}_{868}$	$\texttt{set}_{434}$		
OEF_x2+1	(a)	27,557	0	35,759	23,820	216	20,520	1,284	541,567.20	5.04
	(b)	30,389	0	43,473	25,453	0	23,234	1,632	610,146.86	5.66
	(c)	20,429	0	27,186	16,841	216	15,336	1,284	403,525.18	3.70
	(d)	23,813	0	35,146	19,806	0	18,302	1,632	480,828.36	4.47
	(e)	6,231	2	8,291	5,251	0	4,729	16	123,752.46	1.15
	(f)	6,271	0	8,331	5,292	0	4,757	16	124,496.28	1.15
OEF_x2-5	(a)	27,557	0	35,709	27,092	13,698	20,520	1,284	559,268.56	5.24
	(b)	30,389	0	43,403	29,985	16,079	23,234	1,632	631,965.50	5.88
	(c)	20,429	0	27,145	19,456	10,458	15,336	1,284	417,116.96	3.86
	(d)	23,813	0	35,091	23,501	12,791	18,302	1,632	498,288.30	4.64
	(e)	6,231	2	8,278	6,124	3,224	4,729	16	128,087.64	1.19
	(f)	6,271	0	8,318	6,170	3,243	4,757	16	128,856.92	1.20
AOPF_x2+x+1	(a)	27,557	0	42,711	13,052	216	20,520	1,284	536,243.68	5.08
	(b)	30,389	0	53,085	13,148	0	23,234	1,632	605,731.16	5.73
	(c)	20,429	0	32,462	9,045	216	15,336	1,284	399,913.74	3.78
	(d)	23,813	0	42,925	10,156	0	18,302	1,632	477,606.36	4.52
	(e)	6,231	2	10,142	2,755	0	4,729	16	122,758.02	1.16
	(f)	6,271	0	10,199	2,776	0	4,757	16	123,496.04	1.17
EFN_x2-x-1	(a)	21,113	9,666	33,771	18,770	216	20,520	1,284	534,273.28	4.97
	(b)	21,465	13,386	40,582	21,189	0	23,234	1,632	603,019.58	5.59
	(c)	15,281	7,722	25,329	13,604	216	15,336	1,284	398,338.36	3.64
	(d)	16,533	10,920	32,723	16,718	0	18,302	1,632	475,394.64	4.41
	(e)	4,512	2,581	7,749	4,289	0	4,729	16	122,235.84	1.14
	(f)	4,541	2,595	7,787	4,322	0	4,757	16	122,967.58	1.15

Table 6: The number of function calls, computational complexity and execution time of the SIDH operations (a) Alice's `keygen_iso`, (b) Bob's `keygen_iso`, (c) Alice's `keyshare_iso`, (d) Bob's `keyshare_iso`. (e) Alice's `kernel_gen`, and (f) Bob's `kernel_gen` with p_{441+}.

QEFs	Operation	Function calls							Complexity	Time [ms]
		$\texttt{mul}_{441}$	$\texttt{sqr}_{441}$	$\texttt{add}_{441}$	$\texttt{add}_{882}$	$\texttt{shift}_{441}$	$\texttt{mod}_{882}$	$\texttt{set}_{441}$		
OEF_x2-5	(a)	27,557	0	35,244	27,557	13,698	20,520	1,284	567,817.23	5.23
	(b)	30,389	0	43,000	30,388	16,079	23,234	1,632	641,619.41	5.88
	(c)	20,429	0	26,814	19,787	10,458	15,336	1,284	423,501.99	3.84
	(d)	23,813	0	34,780	23,812	12,791	18,302	1,632	505,890.77	4.64
	(e)	6,232	1	8,181	6,221	3,224	4,729	16	130,056.81	1.20
	(f)	6,271	0	8,223	6,265	3,243	4,757	16	130,835.73	1.21
EFN_x2-x-1	(a)	21,113	9,666	32,028	20,513	216	20,520	1,284	543,033.09	5.01
	(b)	21,465	13,386	38,544	23,227	0	23,234	1,632	612,947.01	5.66
	(c)	15,281	7,722	24,030	14,903	216	15,336	1,284	404,884.65	3.68
	(d)	16,533	10,920	31,145	18,296	0	18,302	1,632	483,210.89	4.46
	(e)	4,512	2,581	7,321	4,716	0	4,729	16	124,257.15	1.16
	(f)	4,541	2,595	7,357	4,752	0	4,757	16	125,001.94	1.16

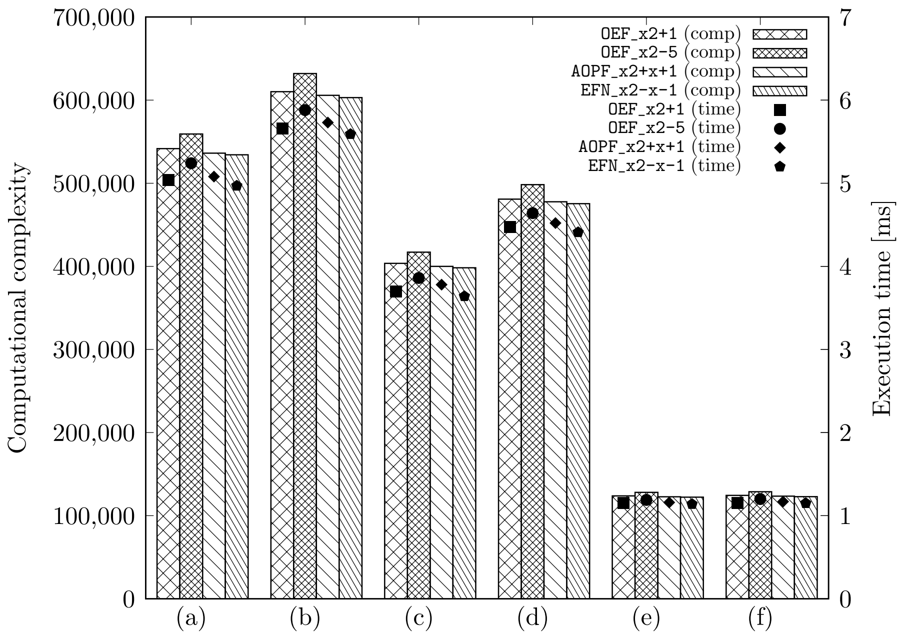

Figure 1: Computational complexity and execution time of the SIDH operations (a) Alice's `keygen_iso`, (b) Bob's `keygen_iso`, (c) Alice's `keyshare_iso`, (d) Bob's `keyshare_iso`. (e) Alice's `kernel_gen`, and (f) Bob's `kernel_gen` with p_{434-}.

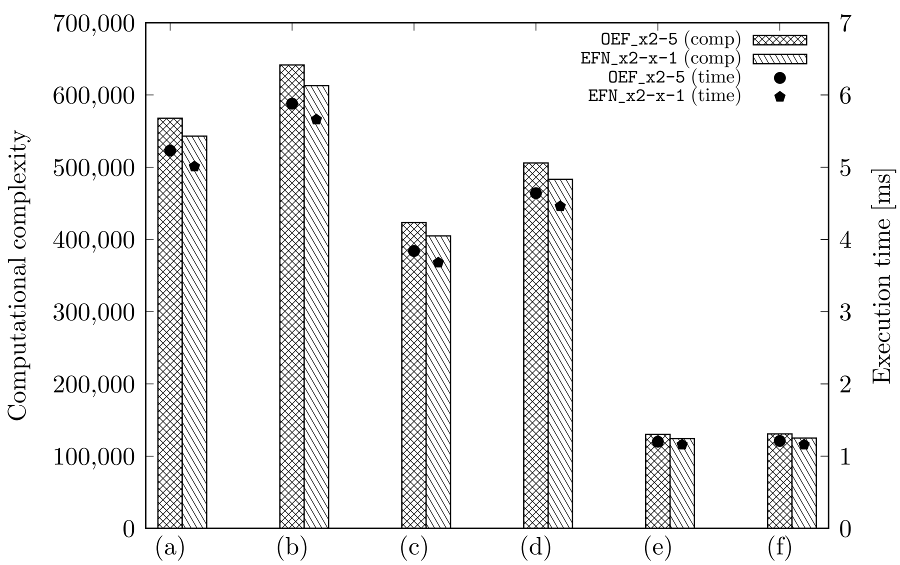

Figure 2: Computational complexity and execution time of the SIDH operations (a) Alice's `keygen_iso`, (b) Bob's `keygen_iso`, (c) Alice's `keyshare_iso`, (d) Bob's `keyshare_iso`. (e) Alice's `kernel_gen`, and (f) Bob's `kernel_gen` with p_{441+}.

practical implementations. However, as described in Sect. 3.3, there does not exist good choices of QEFs based on OEFs for the SIDH with $p = 2^{e_A}3^{e_B}f + 1$. In contrast, the authors found the new candidate of QEFs, i.e., `EFN_x2-x-1`, for such the SIDH. According to Table 6 and Fig. 2, `EFN_x2-x-1` contributes to improve the performance of the SIDH operations around 4% comparing with the previous best choice of QEFs based on OEFs, i.e., `OEF_x2-5`. Moreover, the performance of the SIDH with p_{441+} applied `EFN_x2-x-1` is competitive to that of p_{434-} applied `OEF_x2+1`. Thus, the authors conclude that the efficient implementation of the SIDH with $p = 2^{e_A}3^{e_B}f + 1$ can exist.

5 Conclusion

The authors apply the implementation-friendly QEFs which are based on OEFs, AOPFs, and EFNs such that `OEF_x2+1`, `AOPF_x2+x+1`, and `EFN_x2-x-1`, for SIDH and compare the performance of SIDH between these QEFs. As a result of the experiment, the performance of the SIDH with $p = 2^{e_A}3^{e_B}f - 1$ applied `AOPF_x2+x+1` and `EFN_x2-x-1` are competitive to that of `OEF_x2+1` which is employed for the previous implementation as one of the best performing arithmetics. The authors also confirmed that `EFN_x2-x-1` can be applied not only the SIDH with $p = 2^{e_A}3^{e_B}f - 1$ but also $p = 2^{e_A}3^{e_B}f + 1$ and the performance of the later SIDH has competitive to that of the previous implementation. As one of the future works, the authors would like to investigate the reason why the execution time of the SIDH operations applied `AOPF_x2+x+1` is slightly worse than that of `OEF_x2+1`. The authors also try to find the SIDH-friendly prime given as $p = 2^{e_A}3^{e_B}f + 1$ with $f = 1$.

Acknowledgment

This research was supported by JSPS KAKENHI Grant Numbers 19J2108612 and 19K11966.

References

[1] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," International Workshop on Post-Quantum Cryptography, pp.19–34, Springer, 2011.

[2] D.V. Bailey and C. Paar, "Efficient arithmetic in finite field extensions with application in elliptic curve cryptography," Journal of cryptology, vol.14, no.3, pp.153–176, 2001.

[3] A.A. Karatsuba and Y.P. Ofman, "Multiplication of many-digital numbers by automatic computers," Doklady Akademii Nauk, pp.293–294, Russian Academy of Sciences, 1962.

[4] Y. Nogami, A. Saito, and Y. Morikawa, "Finite extension field with modulus of all-one polynomial and representation of its elements for fast arithmetic operations," IEICE transactions on fundamentals of electronics, communications and computer sciences, vol.86, no.9, pp.2376–2387, 2003.

[5] T. Kobayashi, K. Aoki, and F. Hoshino, "Oef using a successive extension," The 2000 Symposium on Cryptography and Information Security, no. B02, 2000.

[6] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," Annual International Cryptology Conference, pp.572–601, Springer, 2016.

[7] R. Azarderakhsh, B. Koziel, A. Jalali, M.M. Kermani, and D. Jao, "Neon-sidh: Efficient implementation of supersingular isogeny diffi e-hellman key-exchange protocol on arm.," IACR Cryptology ePrint Archive, vol.2016, p.669, 2016.

[8] Y. Nanjo, M. Shirase, T. Kusaka, and Y. Nogami, "A performance analysis and evaluation of sidh with implementation-friendly classes of quadratic extension fields," 2019 Seventh International Symposium on Computing and Networking (CANDAR), pp.178–184, IEEE, 2019.

[9] M. Campagna, C. Costello, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, D. Urbanik, *et al.*, "Supersingular isogeny key encapsulation," 2019.

[10] P.L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," Mathematics of computation, vol.48, no.177, pp.243–264, 1987.

[11] J. Vélu, "Isogénies entre courbes elliptiques," CR Acad. Sci. Paris, Séries A, vol.273, pp.305–347, 1971.

[12] L. De Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," Journal of Mathematical Cryptology, vol.8, no.3, pp.209–247, 2014.

[13] C. Costello and H. Hisil, "A simple and compact algorithm for sidh with arbitrary degree isogenies," International Conference on the Theory and Application of Cryptology and Information Security, pp.303–329, Springer, 2017.

[14] A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez, "A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol," IEEE Transactions on Computers, vol.67, no.11, pp.1622–1636, 2017.

[15] A.L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," Soviet Mathematics Doklady, pp.714–716, 1963.

[16] S.A. Cook and S.O. Aanderaa, "On the minimum computation time of functions," Transactions of the American Mathematical Society, vol.142, pp.291–314, 1969.

[17] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson, "Optimal normal bases in gf (pn)," Discrete Applied Mathematics, vol.22, no.2, pp.149–161, 1988.

[18] R. Lidl and H. Niederreiter, Finite fields, Cambridge university press, 1997.

[19] N. Koblitz, A course in number theory and cryptography, Springer Science & Business Media, 1994.

[20] P.L. Montgomery, "Modular multiplication without trial division," Mathematics of computation, vol.44, no.170, pp.519–521, 1985.

[21] T. Granlund and the GMP development team, "Gnu mp: the gnu multiple precision arithmetic library, 6.1.2," 2015. `https://gmplib.org`.

[22] W.C. Waterhouse, "Abelian varieties over finite fields," Annales scientifiques de l'École Normale Supérieure, pp.521–560, 1969.

[23] J. Jančár, "Ecgen: tool for generating elliptic curve domain parameters," 2018. `https://github.com/J08nY/ecgen`.

A Improvement of Curve Determination of 3-Isogeny

According to App. A in [13], the operation `3_iso_curve` is computed in $2\mathbf{M_2} + 3\mathbf{S_2} + 14\mathbf{a_2}$.

$$\begin{gathered} K_1 = X_3 - Z_3, R_1 = K_1^2, K_2 = X_3 - Z_3, R_2 = K_2^2, R_3 = R_2 + R_1, R_4 = K_1 - K_2, R_4 = R_4^2, \\ R_4 = R_4 - R_3, R_3 = R_4 + R_2, R_4 = R_4 + R_1, R_5 = R_1 + R_4, R_5 = R_5 + R_5, R_5 = R_5 + R_2, \\ A_{24} = R_5 \cdot R_3, R_5 = R_2 + R_3, R_5 = R_5 + R_5, R_5 = R_5 + R_1, \\ R_5 = R_5 \cdot R_4, C_{24} = R_5 - A_{24}. \end{gathered} \tag{18}$$

The authors propose to compute not (A_{24}, C_{24}) but $(A_{24}, K_{24} = A_{24} + C_{24})$ by replacing Eq. (18) as $K_{24} = R_5 \cdot R_4$. The proposed operation is named as `3_iso_curve*`. The complexity of `3_iso_curve*` is $2\mathbf{M_2} + 3\mathbf{S_2} + 13\mathbf{a_2}$. This optimization can contribute to reduce e-times $\mathbf{a_2}$ for 3^e-isogeny.

B Arithmetic Operations in QEFs

Let $A = (a_0, a_1)$ and $B = (b_0, b_1)$ be arbitrary elements in $\mathbb{F}_{p^2}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_p$. Then multiply of A and B and square of A, i.e., $A \cdot B = (u_0, u_1)$ and $A^2 = (v_0, v_1)$ with $u_0, u_1, v_0, v_1 \in \mathbb{F}_p$, can be computed by using variable elements $t_1, t_2, t_3, t_4 \in \mathbb{F}_p$ as follows:

- `OEF_x2+1`

  Mul.; $t_1 = a_0b_0, t_2 = a_1b_1, t_3 = a_0 + a_1, t_4 = b_0 + b_1, u_0 = t_1 - t_2, u_1 = t_3t_4, u_1 = u_1 - t_1, u_1 = u_1 - t_2$.

  Sqr.; $t_1 = a_0 + a_1, t_2 = a_0 - a_1, v_1 = a_0a_1, v_1 = v_1 + v_1, v_0 = t_1t_2$.

- `OEF_x2-5`

  Mul.: $t_1 = a_0 + a_1, t_2 = b_0 + b_1, t_1 = t_1t_2, t_2 = a_0b_0, t_3 = a_1b_1, t_2 = t_2 + t_3, u_1 = t_1 - t_2, t_3 = 4t_3, u_0 = t_2 + t_3$.

  Sqr.: $t_1 = a_0 + a_1, t_2 = 4a_1, t_2 = t_2 + a_1, t_2 = a_0 + t_2, t_1 = t_1t_2, t_2 = a_0a_1, v_1 = t_2 + t_2, t_3 = 4v_1, t_3 = t_3 - v_1, v_0 = t_1 - t_3$.

- `AOPF_x2+x+1`:

  Mul.: $t_1 = a_0 - a_1, t_2 = b_0 - b_1, t_1 = t_1t_2, t_2 = a_0b_0, t_3 = a_1b_1, u_0 = t_1 - t_2, u_1 = t_1 - t_3$.

  Sqr.: $t_1 = a_0 + a_0, t_1 = a_1 - t_1, t_2 = a_1 + a_1, t_2 = a_0 - t_2, v_0 = t_1a_1, v_1 = t_2a_0$.

- `EFN_x2-x+1`

  Mul.: $t_1 = a_0 - a_1, t_2 = b_0 - b_1, t_1 = t_1t_2, t_2 = a_0b_0, t_3 = a_1b_1, u_0 = t_2 - t_1, u_1 = t_3 - t_1$.

  Sqr.: $t_1 = a_0 + a_0, t_1 = t_1 - a_1, t_2 = a_1 + a_1, t_2 = t_2 - a_0, v_0 = t_1a_1, v_1 = t_2a_0$,

- `EFN_x2-x-1`

  Mul.: $t_1 = a_0 - a_1, t_2 = b_0 - b_1, t_1 = t_1t_2, t_2 = a_0b_0, t_3 = a_1b_1, u_0 = t_1 + t_2, u_1 = t_1 + t_3$.

  Sqr.: $t_1 = a_0 - a_1, t_1 = t_1^2, t_2 = a_0^2, t_3 = a_1^2, v_0 = t_1 + t_2, v_1 = t_1 + t_3, v_1 = t_1 + t_3$.

C Supersingular Elliptic Curves of Order $(p-1)^2$

Let E be a supersingular elliptic curve of which order is $\#E(\mathbb{F}_{p^2}) = (p-1)^2$. According to [22], a twist of E, which is denoted as $E'/\mathbb{F}_p : y^2 = x^3 + ax + b$, has an order $\#E'(\mathbb{F}_{p^2}) = (p+1)^2$. For $p = p_{441+}$, the coefficients of E' are easily found by using ecgen library [23].

$a =$ `00627426 b720ddfa 4e7970c2 25f07717 f583111e 9cba318c 9bba7fcd`
`d4e49249 24924924 92492492 49249249 24924924 92492492 49249245`,

$b =$ `00cfd8c3 829ab82c de8e98b6 501817dd 3f312424 2e6ca17e 2c50d4eb`
`6c1b6db6 db6db6db 6db6db6d b6db6db6 db6db6db 6db6db6d b6db6dbc`.

Since E is a quadratic twist of E' defined over $\mathbb{F}_{p^2}$, the curve E is obtained as $E/\mathbb{F}_{p^2} : y^2 = x^3 + \delta^{2/3}ax + \delta b$ where δ is quadratic non-residue and cubic residue in $\mathbb{F}_{p^2}$. Note that the elliptic curve of the Weierstrass form can be easily converted to that of the Montgomery form.