Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Isogenias en criptografía de llave pública

Tesis que presenta

Daniel Idelfonso Cervantes Vázquez

para obtener el Grado de

Doctor en Ciencias en Computación

Director de tesis:

Dr. Francisco José Rambó Rodríguez Henríquez

Ciudad de México **Febrero 2021**

Center for Research and Advanced Studies of National Polytechnic Institute

Zacatenco Campus

Computer Science department

Isogenies on public-key cryptography

A dissertation presented by

Daniel Idelfonso Cervantes Vázquez

as the fulfillment of the requirement for the degree of

Ph. D in Computer Science

Advisor

Dr. Francisco José Rambó Rodríguez Henríquez

Mexico City **February 2021**

Resumen

La criptografía es la práctica y estudio de técnicas para asegurar las comunicaciones en presencia de terceras personas llamadas adversarios. La criptografía toma como base "problemas computacionalmente difíciles" para proveer servicios como autenticación, confidencialidad, integridad, no repudio, entre otros. Se dice que un problema es computacionalmente difćil si no existe un algoritmo polinomial (en complejidad y espacio) capaz de resolverlo en un escenario factible.

Algunos problemas computacionales que son considerados díciles en computadoras clásicas podrían no ser difíciles en computadoras cuánticas por ejemplo, la factorización entera. El algoritmo de Shor puede resolver la factorización entera en tiempo polinomial cuántico, más aún, también puede resolver el problema del logaritmo discreto. Hoy en día muchos protocolos de seguridad están basados en esos dos problemas por lo que surge la necesidad de encontrar problemas difćiles para computadoras cuánticas.

Recientemente los procotolos basados en isogenias han llamado la atención de los criptógrafos por su capacidad de resistir las computadoras cuánticas. Las isogenias entre curvas elípticas con homomorphismos de grupo y para propoósitos criptográficos, es posible definir un par de problemas computacionales difíciles aún ante la amenaza de computadoras cuánticas.

Si bien estudio de las isogenias en matemáticas no es nuevo, el uso de estas como una primitiva criptográfica viable lo es. En éste trabajo presentamos un estudio de cómo las isogenias son utilizadas en criptografía y de manera particular en la criptografía de llave pública. Presentamos las cuatro clases de isogenia que existen en las curvas de Koblitz definidas sobre $\mathbb{F}_4$ y como una de éstas (hasta donde sabemos no estudiada antes) contiene un endomorfismo que permite una aceleración considerable en la multiplicación escalar en dichas curvas. Se presenta un análisis de seguridad del protocolo SIDH tomando como base el problema de encontrar colisiones en dos conjuntos, y con base en éste análisis se proponen nuevos parámetros para ser utilizados en el protocolo SIDH. Además presentamos algunas remediaciones ante ataques de fallo a la implementación del protocolo CSIDH y se presenta la forma de obtener una implementación de tiempo constante. Se introduce el uso de algoritmos aritméticos para mejorar el cómputo de isogenias de grado impar de la forma

$d = 8k + r$ con $r \in \{1, 3, 5, 7\}$. Finalmente se estudia la inclusión del cómputo paralelo en el protocolo SIDH la cuál nos permite desarrollar una variante del mismo llamada eSIDH ("extended SIDH"). El cómputo paralelo en el contexto del cómputo de isogenias ha sido estudiado antes pero nuestra variante introduce el uso de 3 primos en la configuración de SIDH lo cuál permite tomar ventaja del cómputo paralelo en la generación de llaves y el cómputo de isogenias. Más aún, tomamos ventaja de alguans propiedades de la escalera de Montgomery para computar eficientemente múltiplos de la llave privada en paralelo y en conjunto, todas estas mejoras derivan en una aceleración teórica cercana a computar el protocolo eSIDH hasta 3 veces más rápido.

Abstract

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries [99]. Cryptography uses "hard computational problems" to provide security services such as authentication, confidentiality, integrity, non-repudiation, among others. We say that a computational problem is hard if there is no polynomial (time and space) algorithm able to solve it in a feasible scenario.

Some computational problems that are considered hard on non-quantum (classical) computers could be not hard on quantum computers, for example, the integer factorization. Shor's algorithm can solve integer factorization in a quantum computer in polynomial time; moreover, it can also break the discrete log problem. Nowadays, most security protocols are based on those problems, and then there is a necessity to find hard problems for quantum computing.

Recently, isogeny-based protocols took the attention of cryptographers due to its resistance to quantum computers. Isogenies between elliptic curves are group homomorphisms, and for cryptography purposes, it is possible to define a couple of hard computational problems even considering the quantum menace. The study of isogenies in mathematics is not new, but the use as a viable cryptographic primitive is, in this work, we review how isogenies are used in cryptography in particular, in public key cryptography. We present the 4 isogeny classes on Koblitz curves over $\mathbb{F}_4$ and how a no-studied-before (as far as we know) class contains an efficient endomorphim which allows a considerable speed-up for scalar multiplication computation on such curves. We present a security analysis of the SIDH protocol based on the problem of found collitions on two sets and exhibe new parameters to be used in SIDH protocol. We present some remediations to CSIDH implementation against fault attacks an how we can achieve a constant-time implementation. We introduce the use of some arithmetic algorithms to improve the computation of odd-degree isogenies of the form $d = 8k + r$ with $r \in \{1, 3, 5, 7\}$. Finally we introduce the use of parallel computing in SIDH and we develop a variant of SIDH that we dubed as eSIDH (extended SIDH). Parallel computing has been studies before in the isogeny computation context but our novel variant of eSIDH introduces the use of 3 primes into the SIDH configuration, allowing us to take more advantage of parallel computing in key generation and isogeny

computations. Furthermore we take advantage of some properties of Montgomery ladder to compute multiples of the key in parallel and this all together derives in a theoretically speed up of about 3 times faster than traditional SIDH.

Este trabajo es dedicado a...

mis padres, hermana, sobrina y a mi futura esposa.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el sustento económico ofrecido por el programa de becas nacionales, sin el cual no sería posible lograr este objetivo.

A mi asesor Francisco Rodríguez Henríquez por todo su apoyo durante mi trayecto, por su dirección y sus consejos.

Al personal administrativo, en especial Sofía Reza, Erika Ríos y Felipa Rosas, de quienes siempre he recibido apoyo total.

A mis sinodales, la doctora Sandra Diaz y los doctores Juan Carlos Ku Cahuich, Cuauhtemoc Mancillas y Guillermo Morales Luna, por sus valiosos comentarios.

A mis padres que sin ellos no podría haber llegado hasta aquí, a mi hermana y mi sobrina por su apoyo emocional y por último y no menos importante a mi compañera de vida.

A mis compañeros del laboratorio de criptografía por sus valiosos comentarios y aportes durante mi estancia en el departamento, de manera especial a José Bernal y Eduardo Ochoa por su amistad.

Al Dr. Alfred Menezes por su amabilidad y apoyo durante mi estadía en Canadá que sin su apoyo no hubiera sido posible.

A los doctores Julio López y Ricardo Dahab y a Vitor Satoru Matsumine por sus valiosos comentarios.

Finalmente a los amigos que me acompañaron durante ésta aventura, de manera presencial en el departamento del Cinvestav y de manera virtual a mis amigos de *DMX plus*

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

> -How long do you want these messages to remain secret?[...]
> +I want them to remain secret for as long as men are capable of evil.
>
> Neal Stephenson, Cryptonomicon

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries [99]. Cryptography uses ”hard computational problems” to provide security services such as authentication, confidentiality, integrity, non-repudiation, among others. We say that a computational problem is hard if there is no polynomial (time and space) algorithm able to solve it in a feasible scenario. Algorithms used in cryptography are commonly known as cryptographic algorithms, and the combination of cryptographic algorithms to provide a security service is usually called a cryptographic protocol.

Cryptographic protocols are divided into two main kinds, private key and, public key protocols. A natural way to explain both concepts is using two fictional characters, namely Alice and Bob. Private key protocols are based on the assumption of Alice and Bob have a common secret-key then if Alice wants to send a message to Bob she uses a cryptographic algorithm using the common secret-key and then she sends the ciphertext to Bob. Once Bob receives Alice's message, he can decrypt the ciphertext using the shared secret key. Notice that as long as only Alice and Bob know the secret key cryptography guarantees that anyone can decrypt the message. The main issue of private key cryptography is that in a real-world scenario, it could be challenging that Alice and Bob agree on a secret key. Fortunately, the public key cryptography provides a solution to this problem. Public key protocols scenario consider the use of two keys, one secret-key as in private key protocols, and one public key. The public key is a key related to the secret one in the sense that commonly the secret key is used to generate the public key but, anyone can derive the secret key knowing the public key. If Alice wants to send a message to Bob, she encrypts the message using Bob's public key, and then she sends the message to Bob. One Bob receives the message he uses his secret key to decrypt the message.

Chronologically speaking, public-key cryptography was proposed in 1976 by Whitfield Diffie and Martin Hellman [37], who devised a novel scheme that allows two entities to agree to a secret key using an insecure communication channel, without previously agreeing on a secret. This scheme bases its security on the difficulty of discrete logarithm problem (DLP) in finite fields. Time after that, in 1977 Ron Rivest, Adi Shamir, and Leonard Adleman proposed the cryptographic scheme known as RSA [84], which can be used as both encryption scheme and digital signature scheme. RSA security is based on the difficulty of factoring large integers. Until 1985 the public key schemes were based on number theory, particularly they used the multiplicative group of integers modulo a large integer (in RSA) or a prime number (in the Diffie-Hellman scheme). In that year Neal Koblitz [53] and Victor Miller [67] proposed the use of elliptic curves independently for cryptographic purposes, giving in this way birth to the field of elliptic curve cryptography (ECC). They observed that when an elliptic curve is defined over a finite field, the points on the elliptic curve form an Abelian group, whose associated DLP results difficult to solve. Such DLP is even much more difficult to solve than its analog in finite fields using the same group and field order. Thus, it is possible to offer the same security provided by the other existing public-key schemes, but using much smaller fields (rings for RSA).

As we mention, cryptographic protocols security relies on hard computational problems, and there are several alternatives to the discrete log and integer factorization. For example, hash functions security is based on the assumption that it is difficult to find collisions between two sets. This problem is considered hard even on the presence of quantum computers because the only algorithm able to dramatically weakens hash functions security is Grover's. Quantum computing is the use of the quantum-phenomena to perform computations. Some computational problems that are considered hard on non-quantum (classical) computers could be not hard on quantum computers, for example, the integer factorization. Shor's algorithm can solve integer factorization in a quantum computer in polynomial time; moreover, it can also break the discrete log problem. Nowadays, most security protocols are based on those problems, and then there is a necessity to find hard problems for quantum computing.

Recently, isogeny-based protocols took the attention of cryptographers due to its resistance to quantum computers. Isogenies between elliptic curves are group homomorphisms, and for cryptography purposes, it is possible to define a couple of hard computational problems even considering the quantum menace. The study of isogenies in mathematics is not new, but the use as a viable cryptographic primitive is, in this work, we review how isogenies are used in cryptography even when we did not call them isogenies[1].

1.1 Motivation

Quantum computers seem to be a real treat to the current state of security. Among quantum-resistant protocols, isogeny-based ones, are a promising candidate to be established as a quantum-resistant standard. The main advantage of such protocols is the use of short keys in comparison with the other competitors (Lattice-based, Codes-based, among others). The main disadvantage is that isogeny computations are slower than others like Lattice-based protocols. Another disadvantage is that isogenies, as a cryptographic primitive, is a new area and there is no sufficient evidence about its security assumptions.On the other hand, it is not clear when quantum computers could break current cryptography, and then it is also necessary to improve classical cryptographic primitives.

1.2 Outline

The organization of the document is a follows. In Chapter Two, we introduce some mathematical concepts related to isogenies between elliptic curves. In Chapter three, we study a family of curves called Koblitz curves and how to isogenies helps to improve scalar multiplication, which is the most important primitive in elliptic curve cryptography. Chapter four introduce two isogeny-based protocols and present a brief recapitulation of state of the art related to isogeny-based protocols. Chapter five review the NAF algorithm to be used into isogeny computations. Chapter six shows a classical attack against SIDH. The security of SIDH relies on the computational supersingular isogeny problem which is modeled as a collision search in two sets. In Chapter seven, we analyze and propose a constant time csidh secured against fault attacks. In chapter eight we present a variant of the SIDH protocol that we dubbed as eSIDH(extended SIDH) which takes advantage of parallel computing and makes use of a different structure of primes than SIDH primes. In the end, in Chapter nine, we summarize all the work and present our conclusions.

[1] Isogenies are isogenies by definition but, cryptographic community maked use of a special kind of isogenies called endomorphisms.

Chapter 2

Mathematical Background

2.1 Algebraic Varieties

We define some concepts which arise in the study of algebraic geometry. This concepts lead us to the next section when we define elliptic curves. The intention of this section is to introduce the notion of *affine* and *projective* varieties over finite fields, for further reading we refer the reader to [89],[45] and [39]. We set the following notation which will be used throughout this document.

$\mathbb{F}_q$ Denotes a finite field of characteristic p, with $q = p^n$ elements.

$\bar{\mathbb{F}}_q$ Denotes the algebraic closure of $\mathbb{F}_q$.

Along this chapter, m and n denote positive integers.

Definition 2.1.1. Affine n-space over $\mathbb{F}_q$ is the set of n-tuples

$$\mathbb{A}^n = \mathbb{A}^n(\bar{\mathbb{F}}_q) = \{P = (x_1, x_2, \dots, x_n) \mid x_i \in \bar{\mathbb{F}}_q\}.$$

Similarly, the set of $\mathbb{F}_q$-rational points in $\mathbb{A}^n$ is the set

$$\mathbb{A}^n(\mathbb{F}_q) = \{P = (x_1, x_2, \dots, x_n) \in \mathbb{A}^n \mid x_i \in \mathbb{F}_q\}.$$

Let $\bar{\mathbb{F}}_q[X] = \bar{\mathbb{F}}_q[X_1, \ldots, X_n]$ be a polynomial ring in n variables, and let $I \subset \bar{\mathbb{F}}_q[X]$ be an ideal. To each such I we associate a subset of $\mathbb{A}^n$,

$$V_I = \{P \in \mathbb{A}^n \mid f(P) = 0 \text{ for all } f \in I\}.$$

Definition 2.1.2. An affine algebraic set is any set of the form V_I. If V is an algebraic set the ideal of V is given by

$$I(V) = \{f \in \bar{\mathbb{F}}_q[X] \mid f(P) = 0 \text{ for all } P \in V\}.$$

Al algebraic set is defined over $\mathbb{F}_q$ if its ideal $I(V)$ can be generated by polynomials in $\mathbb{F}_q[X]$. We denote this by $V/\mathbb{F}_q$. If V is defines over $\mathbb{F}_q$, the set of $\mathbb{F}_q$-rational points of V is the set

$$V(\mathbb{F}_q) = V \cap \mathbb{A}^n(\mathbb{F}_q).$$

Definition 2.1.3. An affine algebraic set V is called an affine variety if $I(V)$ is a prime ideal in $\bar{\mathbb{F}}_q[X]$.

Lemma 2.1.1. The dimension of $\mathbb{A}^n$ is n, since $\bar{\mathbb{F}}_q(\mathbb{A}^n) = \bar{\mathbb{F}}_q(\mathbb{A}^n) = \bar{\mathbb{F}}_q(x_1, \ldots, x_n)$. Similarly, if $V \subset \mathbb{A}^n$ is given by a non-constant polynomial equation

$$f(X_1, \ldots, X_n) = 0,$$

then $\dim(V) = n - 1$.

Definition 2.1.4. Projective n-space over $\mathbb{F}_q$, denoted $\mathbb{P}^n$ or $\mathbb{P}^n(\bar{\mathbb{F}}_q)$, is the set of all $(n+1)$-tuples

$$(x_0, \ldots, x_n) \in \mathbb{A}^{n+1}$$

such that at least one x_i is non-zero, modulo the equivalence relation given by

$$(x_0, \ldots, x_n) = (y_0, \ldots, y_n)$$

if there exists a $\lambda \in \bar{\mathbb{F}}_q^{\,*}$ with $x_i = \lambda y_i$ for all i. An equivalence class

Definition 2.1.5. A polynomial $f \in \bar{\mathbb{F}}_q[X] = \bar{\mathbb{F}}_q[X_0, \ldots, X_n]$ is homogeneous of degree d if

$$f(\lambda X_0, \ldots, \lambda X_n) = \lambda^d f(X_0, \ldots, X_n)$$

for all $\lambda \in \bar{\mathbb{F}}_q$. An ideal $I \subset \bar{\mathbb{F}}_q[X]$ is homogeneous if it is generated by homogeneous polynomials.

For each homogeneous ideal I we associate a subset of $\mathbb{P}^n$,

$$V_I = \{P \in \mathbb{P}^n \mid f(P) = 0 \text{ for all homogeneous } f \in I\}.$$

Definition 2.1.6. A projective algebraic set is any set of the form V_I. If V is a projective algebraic set, the homogeneous ideal of V, denoted $I(V)$, is the ideal in $\bar{\mathbb{F}_q}[X]$ generated by

$$\{f \in \bar{\mathbb{F}_q}[X] \mid f \text{ is homogeneous and } f(P) = 0 \text{ for all } P \in V\}$$

Such a V is defined over $\mathbb{F}_q$, denoted by $V/\mathbb{F}_q$, if its ideal $I(V)$ can be generated by homogeneous polynomials in $K[X]$. If V is defined over $\mathbb{F}_q$, the set of $\mathbb{F}_q$-rational points of V is the set

$$V(K) = V \cap \mathbb{P}^n(K).$$

Definition 2.1.7. A projective algebraic set is called a projective variety if its homogeneous ideal $I(V)$ is a prime ideal in $\bar{\mathbb{F}_q}[X]$.

2.2 Elliptic Curves

In this section we overview the mathematical concepts related to elliptic curves and maps between them. For further reading and more details we suggest to revise [103, 92]. Most of the content of this section comes from the recommended lectures.

Definition 2.2.1. An *elliptic curve* over a field $\mathbb{F}_q$ is defined by an equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and $\Delta \neq 0$, where Δ is the discriminant of E and is defined as follows:

$$\begin{aligned}
\Delta =& - d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6\\
d_2 =& a_1^2 + 4a_2\\
d_4 =& 2a_4 + a_1a_3\\
d_6 =& a_3^2 + 4a_6\\
d_8 =& a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3 - a_4^2.
\end{aligned}$$

Above definition is the general Weierstrass equation, or model. There are different models (equations) defining elliptic curves with different properties, in the rest of the chapter, we discuss two short Weierstrass models, the Montgomery model and the Edwards one.

From the point of view of the Algebraic geometry, an elliptic curve is a pair (E, ∞), where E is a projective variety of genus 1 and $\infty \in E$. Since elliptic curves are projective varieties, definition, notation and lemmas of previous section applies on it.

Definition 2.2.2 (Short Weierstrass model ($p > 3$))**.** An elliptic curve in Weierstrass short model over a finite field $\mathbb{F}_q$ where $q = p^m$ for some prime $p > 3$, is given by the equation

$$E/\mathbb{F}_q : Y^2 = X^3 + AX + B$$

where $A, B \in \mathbb{F}_q$.

Definition 2.2.3 (Short Weierstrass model ($p = 2$))**.** An elliptic curve in Weierstrass short model over a finite field $\mathbb{F}_q$ where $q = 2^m$, is given by the equation

$$E/\mathbb{F}_q : Y^2 + XY = X^3 + AX^2 + B$$

where $A, B \in \mathbb{F}_q$. We say that this curve is a binary elliptic curve and we call binary field to a field of characteristic two.

If we consider the set containing the set of $\mathbb{F}_q$-rational points of an elliptic curve $E/\mathbb{F}_q$ and the set containing a *point at infinity* $\{\infty\}$, then we have an additive Abelian group $(E(\mathbb{F}_q) \cup \{\infty\}, +)$. This means that we can add and negate (additive inverse) points, the case when we add a point with itself, we say that we doubling the point.

Definition 2.2.4 (Scalar multiplication)**.** Let k be an positive integer and P be a point in an elliptic curve E. We define the scalar multiplication on elliptic curves as

$$[k]P = \underbrace{P + P + \cdots + P}_{k-1 \text{ times}}.$$

If $k < 0$ then $[k]P = [-k](-P)$. For $m = 0$ then $[0]P = \infty$.

Algorithm 1 computes $[k]P$ using the binary representation of the scalar $k = (k_{t-1}, \ldots, k_1, k_0)_2$, on average half of the bits of k are 1, then the cost of computing $[k]P$ using this algorithm is t doublings and $t/2$ additions. In the literature, the scalar multiplication is also called multiplication-by-k map.

Definition 2.2.5. Let P be a point in an elliptic curve $E/\mathbb{F}_q$. The minimum integer $m > 0$ shuch that $[m]P = \infty$ is called the *order* of P.

Definition 2.2.6. The *subgroup generated* by P is the set $\{P, [2]P, [3]P, \ldots, [m-1]P, \infty\}$ and is denoted by $\langle P \rangle$.

Definition 2.2.7. The *m-torsion subgroup* of an elliptic curve $E/\mathbb{F}_q$ is defined as

$$E[m] = \{P \in E(\mathbb{F}_q) \mid [m]P = \infty\}.$$

Definition 2.2.8. Let E to be an elliptic curve in short Weierstrass model. The *j-invariant* of E is

$$j(E) = 1728 \frac{4A^3}{4A^3 + 27B^2}.$$

Algorithm 1 Left-to-Right Binary scalar multiplication

```
Require: Integer $k = (k_{t-1}, \dots, k_1, k_0)_2$, Point $P \in E(\mathbb{F}_q)$.
Ensure: Point $Q = [k]P$.
$Q \leftarrow P$;
for i=t-1 down to 0 do
    if $k_i = 1$ then
        $Q \leftarrow Q + P$;
    end if;
    $Q \leftarrow [2]Q$;
end for;
return $Q$
```

Theorem 2.2.1 (Hasse)**.** Let E be an elliptic curve over the finite field $\mathbb{F}_q$. Then the order of $E(\mathbb{F}_q)$ satisfies

$$\mid q + 1 - \#E(\mathbb{F}_q) \mid \leq 2\sqrt{q}.$$

There are several ways to classify elliptic curves, the following definition exhibit one of them.

Definition 2.2.9. An elliptic curve $E/\mathbb{F}_q$ is *supersingular* if

$$\#E(\mathbb{F}_q) = q + 1 + k \cdot p.$$

Otherwise we say that $E/\mathbb{F}_q$ is *ordinary*.

2.3 Isogenies between elliptic curves

Isogenies are the main core of our work, in this section we introduce the definition and some properties about isogenies between elliptic curves.

Definition 2.3.1. Let E_0 and E_1 be elliptic curves. An isogeny between E_0 and E_1 is a morphism

$$\varphi : E_0 \to E_1$$

satisfying $\varphi(\infty) = \infty$. E_0 and E_1 are *isogenous* if there is an isogeny between them with $\varphi(E_0) \neq \{\infty\}$.

Remark 1. An isogeny is a rational map but since elliptic curves are smooth curves then this rational map turns into a morphism *i.e.* if $\varphi : E_0 \to E_1$ is an isogeny then

$$\varphi(P + Q) = \varphi(P) + \varphi(Q)$$

holds for all points $P, Q \in E(\mathbb{F}_q)$.

Remark 2. As an isogeny is a morphism between elliptic curves, then we can define it explicitly as

$$\varphi(x, y) = (r_0(x), yr_1(x)),$$

where r_0 and r_1 are rational functions. If the coefficients of r_0 and r_1 lie in $\mathbb{F}_q$ then we say that φ is defined over $\mathbb{F}_q$. As r_0 is a rational function then we can write it as

$$r_0(x) = \frac{p(x)}{q(x)}$$

such that $\gcd(p(x), q(x)) = 1$.

Lemma 2.3.1 (Tate's theorem [95])**.** Let E_0 and E_1 be two isogenous elliptic curves over a field $\mathbb{F}_q$. The number of $\mathbb{F}_q$-rational points of both curves is the same. The converse is also true.

Definition 2.3.2. Let $\varphi : E_0 \to E_1$ be an isogeny. We define the *degree* of φ to be

$$\deg(\varphi) = \max\{\deg(p(x)), \deg(q(x))\},$$

where $p(x)$ and $q(x)$ are like in Remark 2

Definition 2.3.3. Let $\varphi : E_0 \to E_1$ be an isogeny. If the derivative $r_0'(x)$ is not identically zero, we say that φ is *separable*, otherwise, we say that φ is *inseparable*.

Definition 2.3.4. Let $\varphi : E_0 \to E_1$ be an isogeny. The *Kernel* of φ is

$$\mathrm{Ker}(\varphi) = \varphi^{-1}(\infty).$$

Proposition 2.3.2. Let $\varphi : E_0 \to E_1$ be an isogeny. If φ is separable, then

$$\deg(\varphi) = \#\,\mathrm{Ker}(\varphi).$$

If φ is inseparable, then

$$\deg(\varphi) > \#\,\mathrm{Ker}(\varphi).$$

In particular, the kernel of an isogeny is a finite subgroup of $E_0(\bar{\mathbb{F}}_q)$.

Since elliptic curves are groups maps between them forms groups.

Definition 2.3.5. Let $\mathrm{Hom}(E_0, E_1)$ to be the set $\{\varphi \mid \varphi : E_0 \to E_1 \text{is an isogeny}\}$. Then $\mathrm{Hom}(E_0, E_1)$ is a group under the addition law *i.e.* , let $\phi, \psi \in \mathrm{Hom}(E_0, E_1)$ and $P \in E(K)$ then

$$(\phi + \psi)(P) = \phi(P) + \psi(P).$$

$$\begin{tikzcd}
E_0 \arrow[r, "\varphi_1"] \arrow[dr, "\varphi_2"'] & E_1 \arrow[d, leftrightarrow, "\alpha"] \\
& E_2
\end{tikzcd}$$

Figure 2.1: Commutative diagram of Theorem 2.3.3. If $\mathrm{Ker}(\varphi_1) = \mathrm{Ker}(\varphi_2)$ then exists an isomorphism $\alpha : E_1 \to E_2$.

Moreover, if $E_0 = E_1$ then we can also compose isogenies. Let E_0 be an elliptic curve, we define $\mathrm{End}(E_0)$ to be $\mathrm{Hom}(E_0, E_0)$ then $End(E_0)$ is a ring with addition as above and multiplication given by composition,

$$(\phi\psi)(P) = \phi(\psi(P)).$$

This ring is called the *endomorphism ring* of E_0. The invertible elements of $\mathrm{End}(E_0)$ form the *automorphism group* of E_0, which is denoted $\mathrm{Aut}(E_0)$.

Remark 3. The elements of $\mathrm{End}(E_0)$ as in Definition 2.3.5 are called *endomorphisms*.

Theorem 2.3.3. Let E_0, E_1, E_2 be elliptic curves over $\mathbb{F}_q$ and suppose that there exist separable isogenies $\varphi_1 : E_0 \to E_1$ and $\varphi_2 : E_0 \to E_2$ defined over $\bar{\mathbb{F}}_q$. If $\mathrm{Ker}(\varphi_1) = \mathrm{Ker}(\varphi_2)$, then $E_1 \equiv E_2$ over $\mathbb{F}_q$. In fact, there is an isomorphism $\alpha : E_1 \to E_2$ such that $\alpha \circ \varphi_1 = \varphi_2$. This is that, the diagram of Figure 2.1 commutes.

For now on, we only write E to refer us to an elliptic curve instead of $E/\mathbb{F}_q$ unless there will be ambiguities.

2.4 Other models of curves

2.4.1 Koblitz curves

Content of this section is based on [44, §3.4].

Definition 2.4.1. Anomalous binary curves, generally referred to as Koblitz curves, are elliptic curves satisfying the Weierstrass equation,

$$E_a : y^2 + xy = x^3 + ax^2 + 1,$$

with $a \in \mathbb{F}_2$.

Curve Model	Doubling	Addition
Lambda Coordinates	3**M** + 4**S** + 1**C**	11**M** + 2**S**
López-Dahab ($a_2 = 0$)	2**M** + 5**S** + 1**C**	14**M** + 3**S**
López-Dahab ($a_2 = 1$)	2**M** + 4**S** + 2**C**	13**M** + 3**S**
Twisted μ_4-normal form	2**M** + 5**S** + 2**C**	9**M** + 2**S**
Semisplit μ_4-normal form	2**M** + 5**S** + 2**C**	7**M** + 2**S**

Table 2.1: Cost to compute Doubling and Addition in different binary curve models. Considering binary field arithmetic, **M** refers to multiplication, **S** for squaring and **C** for multiplication by constants.

For cryptographic purposes, one uses a subgroup of the group $E_a(\mathbb{F}_{2^m})$ of $\mathbb{F}_{2^m}$-rational points for some extension field 2^m with m prime.

Definition 2.4.2. A Koblitz curve E_a has almost-prime group order over $\mathbb{F}_{2^m}$ if $\#E(\mathbb{F}_{2^m}) = hn$ where n is prime and

$$h = \begin{cases} 4 & ifa = 0 \\ 2 & ifa = 1. \end{cases}$$

h is called the cofactor.

Definition 2.4.3. Let E_a be a Koblitz curve. The Frobenius map $\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$ is defined by

$$\tau(\infty) = \infty, \qquad \tau(x, y) = x^2, y^2.$$

It is known that

$$(\tau^2 + 2)P = \mu\tau(P) \text{ for all } P \in E_a(\mathbb{F}_{2^m}),$$

where $\mu = (-1)^{1-a}$ and τ^l denotes the l-fold application of τ to P.

Remark 4. On Koblitz curves, Frobenius map is an endomorphism.

Scalar multiplication on Koblitz curves

From Definition 2.4.3 the Frobenius endomorphism can be viewed as a complex number satisfying $\tau^2 + 2 = \mu\tau$; we choose $\tau = (\mu + \sqrt{-7})/2$. Let $\mathbb{Z}[\tau]$ be the ring of polynomials in τ with integer coefficients. Now, it makes sense multiply elements in $E(\mathbb{F}_{2^m})$ by elements in $\mathbb{Z}[\tau]$ as follows

$$(\sum_{i=0}^{l-1} u_i \tau^i)P = \sum_{i=0}^{l-1} u_i \tau^i(P).$$

We can make use of the Frobenius endomorphism replacing $[k]P$ by $[\bar{k}]P$, where $\bar{k}$ is an element of $\mathbb{Z}[\tau]$. There are different methods to transform an integer k into an element of $\mathbb{Z}[\tau]$ for example τ-NAF recoding. We refer the reader to [2, 11, 96, 98] for different τ-NAF recodings. The main advantage of using the τ-NAF recoding in Algorithm 2 is that we replace point doublings by τ evaluations which costs 2**S** saving on average 2**M** + 3**S** in comparison with Algorithm 1 according to Table 2.1.

Algorithm 2 Left-to-Right τ-NAF scalar multiplication

```
Require: $\tau$-NAF representation of integer $k = (u_{t-1}, \dots, u_1, u_0)$, Point $P \in E(\mathbb{F}_q)$.
Ensure: Point $Q = [k]P$.
Q ← P;
for i=t-1 down to 0 do
    if $u_i$ = 1 then
        $Q \leftarrow Q + P$;
    end if;
    if $u_i$ = -1 then
        $Q \leftarrow Q - P$;
    end if;
    $Q \leftarrow \tau(Q)$;
end for;
return Q
```

2.4.2 Normal Form

In 2012 David Kohel [56] presented a new models of Binary Elliptic Curves called $\mathbb{Z}/4\mathbb{Z}$-normal form and split μ_4-normal form-normal based on the symmetries in curves presented in [56]. This curve models are of particular interest because allows a faster addition and doubling formulas[1] as can be seen in 2.1. A Binary curve in $\mathbb{Z}/4\mathbb{Z}$-normal form is a curve $E/\mathbb{F}_{2^m}$ in $\mathbb{P}^3$ given by the equations

$$(X_0 + X_1 + X_2 + X3)^2 = eX_0X_2 = eX_1X_3,$$

with $e \in \mathbb{F}_{2^m}$ and identity $\mathcal{O} = (1 : 0 : 0 : 1)$.

Definition 2.4.4. [57, Definition 1] An elliptic curve in μ_4-normal form is a genus one curve in the family

$$X_0^2 + rX_2^2 = X_1X_3, \quad (X_1 + X_3)^2 = X_0X_2,$$

with base point $\mathcal{O} = (1 : 1 : 0 : 1)$. An elliptic curve in semisplit μ_4-normal form is a genus one curve in the family

$$(X_0 + X_2)^2 = X_1X_3, \quad (X_1 + X_3)^2 = sX_0X_2,$$

[1] in fact, there are a set of "rules" which derives in a four different addition formulas and four different doubling formulas.

with identity $\mathcal{O} = (1 : 1 : 0 : 1)$, and an elliptic curve is in split μ_4-normal form if it takes the form

$$(X_0 + X_2)^2 = c^2 X_1 X_3, \quad (X_1 + X_3)^2 = c^2 X_0 X_2,$$

with identity $\mathcal{O} = (c : 1 : 0 : 1)$. There exists a transformation from the split μ_4-normal form to semisplit μ_4-normal form setting $s = c^4$ and it is given by

$$(X_0, X_1, X_2, X_3) \mapsto (X_0, cX_1, cX_2, X_3),$$

and if we setting $r = 1/s^2$ the transformation

$$(X_0, X_1, X_2, X_3) \mapsto (X_0, X_1, sX_2, X_3)$$

maps the semisplit μ_4-normal form to μ_4-normal form.

Lemma 2.4.1. [56, Lemma 11] An elliptic curve in split μ_4-normal form is isomorphic to the curve

$$Y(Y + X)Z = X(X + c^{-2}Z)^2$$

in Weierstrass form. The linear map $(X : Y : Z) = (c(X_1 + X_3) : X_0 + cX_1 + X2 : c^4 X_2)$ defines the isomorphism except at $\mathcal{O}$.

Remark 5. The semisplit μ_4-normal form is called μ_4-normal form in [56].

Remark 6. One important fact about this normal forms is that the curve must have a rational 4-torsion point T ([56, axiom 2]).

In [57] Kohel introduce a new model called Twisted-μ_4-normal form-Normal form.

Lemma 2.4.2. [57, Lemma 11] Let C^t be a binary elliptic curve in twisted μ_4-normal form

$$X_0^2 + bX_2^2 = X_1 X_3 + aX_0 X_1, \quad (X_1 + X_3)^2 = X_0 X_2.$$

Then C^t is isomorphic to the elliptic curve

$$y^2 + xy = x^3 + ax^2 + b,$$

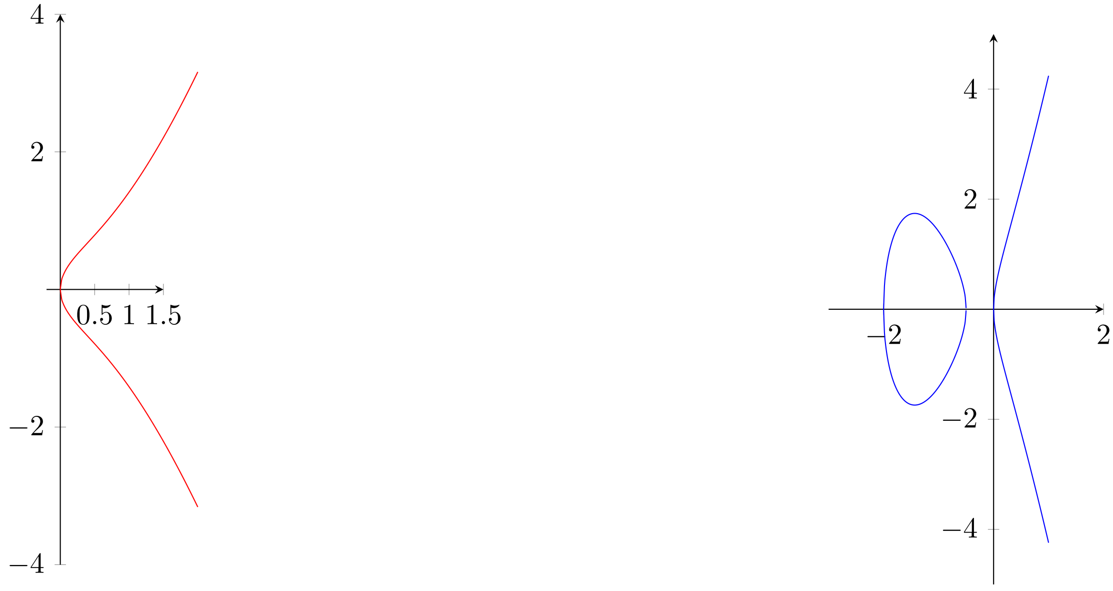
in Weierstrass form via the map $(X_0 : X_1 : X_2 : X_3) \mapsto (X_1 + X_3 : X_0 + X_1 : X_2)$. On affine points (x, y) the inverse is $(x, y) \mapsto (x^2 : x^2 + y : 1 : x^2 + x + y)$.

2.4.3 Montgomery Curves

Content of this section is based on [27, 23].

An elliptic curve E over a field K written in Montgomery form is a curve such that

$$E_{A,B}/K : By^2 = x(x^2 + Ax + 1)$$



(a) $y^2 = x(x^2+1)$

(b) $\frac{1}{4}y^2 = x(x^2+2.5x+1)$

Figure 2.2: Two Examples of Montgomery curves over rational numbers

where $A, B \in K$ are such that $B \neq 0$ and $A^2 \neq 4$. The negation map $\ominus$ is an automorphism

$$\begin{aligned} \ominus : \quad & E \to E \\ \ominus(P) & \mapsto -P \end{aligned}$$

The quotient $E/\langle\ominus\rangle$ is the 2-to-1 mapping $\mathbf{x} : E \to \mathbb{P}^1 \cong E/\langle\ominus\rangle$ such that $\mathbf{x}(P) = \mathbf{x}(Q)$ if and only if $P = Q$ or $P = \ominus Q$. It is common to call $\mathbb{P}^1$ the x-line of E. We can consider projective coordinates $(X : Y : Z)$, with $x = X/Z$ and $y = Y/Z$, and the projective model $E : BY^2Z = X(X^2 + AXZ + Z^2)$. Quotient map $\mathbf{x}$ in projective coordinates is $\mathbf{x}(P) = (x_P : 1)$ for $P = (x_P, y_P, 1)$ and $\mathbf{x}(P) = (1 : 0)$ if P is the point at infinity ∞. In fact, there is no group structure in $\mathbb{P}$ in the sense that there is no map such that $(\mathbf{x}(P), \mathbf{x}(Q)) \mapsto \mathbf{x}(P+Q)$. But as Montgomery observed, we can define an x*-only* addition on $\mathbb{P}^1$ if we know $\mathbf{x}(P), \mathbf{x}(Q)$ and $\mathbf{x}(P-Q)$, we write this by $\texttt{xADD}$ $(\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P-Q)) = \mathbf{x}(P+Q)$. Also we can consider the degenerate case when $P = Q$ and then we have $\texttt{xDBL}$ $(\mathbf{x}(P)) \mapsto \mathbf{x}([2]P)$. Since $\ominus$ commute with $[k]$ and by the x-only arithmetic we can compute $[k]\mathbf{x}(P) = \mathbf{x}([k]P)$ using the mythical Montgomery ladder. There are several works explaining this in depth like [27, 38, 23]. The j-invariant of a Montgomery Curve is the $\mathbb{F}_q$ element $j(E) = 256(A^2-3)^3/(A^2-4)$, which is totally dependent only of A.

Isogenies

Lemma 2.4.3. Theorem 1 of [23] stablish that: Let $P \in E(\bar{K})$ be a point of order $d = 2\ell+1$ on the Montgomery curve $E_{A,B}/K : By^2 = x(x^2 + Ax + 1)$ and write

$$\sigma = \sum_{i=1}^{\ell} \frac{1}{x_{[i]P}} - x_{[i]P} \text{ and } \pi = \prod_{i=1}^{\ell} x_{[i]P}.$$

The Montgomery curve

$$E'_{A',B'}/K : B'y^2 = x(x^2 + A'x + 1)$$

with $A' = (6\sigma + A) \cdot \pi$ and $B' = B \cdot \pi^2$ is the codomain of the normalized d-isogeny $\phi : E \to E'$ with $\mathrm{Ker}(\phi) = \langle P \rangle$.
Moreover, also we can evaluate a point $Q = (x : z)$ not in $\langle P \rangle$ via

$$x' = x_Q \cdot \left(\prod_{i=1}^{\ell} \left[(x_Q - z_Q)(x_{[i]P} + z_{[i]P}) + (x_Q + z_Q)(x_{[i]P} - z_{[i]P}) \right] \right)^2 \tag{2.1}$$

$$z' = z_Q \cdot \left(\prod_{i=1}^{\ell} \left[(x_Q - z_Q)(x_{[i]P} + z_{[i]P}) - (x_Q + z_Q)(x_{[i]P} - z_{[i]P}) \right] \right)^2 \tag{2.2}$$

which has a cost of $4\ell\mathbf{M} + 2\mathbf{S} + (4\ell + 2)\mathbf{A}$.

As we observe from Lemma 2.4.3, the Montgomery coefficient A' depends of σ which involves a field inversion which from the point of view of computational arithmeticians is costly (several multiplications) then, in order to avoid this field inversion is convenient to work with a projective coefficient $A' = (A : C)$ (abusing of the notation) where $A' = A/C$.

Efficient Arithmetic

At the beginning of this section we introduce the x-only arithmetic but we did not give explicit formulas for `xADD` and `xDBL` because the conventional ones, make use of the Montgomery affine constant A and for this work purposes, we require formulas using projective versions of the point and the Montgomery constant. The point doubling (`xDBL`) and differential addition (`xADD`) operations proposed by Montgomery in [68], can be computed as shown in Equations (2.3) and (2.4), where $A_{24p} = A + 2C$ and $C_{24} = 4C$.

$$\begin{aligned} X_{[2]P} &= C_{24}(X_P + Z_P)^2(X_P - Z_P)^2, \\ Z_{[2]P} &= ((X_P + Z_P)^2 - (X_P - Z_P)^2)\cdot \\ &\qquad (C_{24}(X_P - Z_P)^2 + A_{24p}((X_P + Z_P)^2 - (X_P - Z_P)^2)). \end{aligned} \tag{2.3}$$

$$\begin{aligned} X_{P+Q} &= Z_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) + (Z_P + Z_P)(X_Q - Z_Q)\right]^2 \\ Z_{P+Q} &= X_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) - (Z_P + Z_P)(X_Q - Z_Q)\right]^2 \end{aligned} \tag{2.4}$$

The computational cost of `xDBL` (Equation 2.3) is $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$ and for the `xADD` (Equation 2.4) we get $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$. The current state-state-of the art indicates that this formulas are the cheaper ones at least, for isogeny-based cryptography purposes.

2.4.4 Edwards Curves

Edwards curves were introduced by H. Edwards in 2007 [64]. Edwards curves are Elliptic curves that can be written in the form

$$E_d/\mathbb{F}_q : x^2 + y^2 = 1 + dx^2y^2,$$

with $d \neq 1$. Later in 2008 Bernstein-Lange [5] proposed a generalized model called Twisted Edwards curves given by the equation

$$E_{a,d}/\mathbb{F}_q : ax^2 + y^2 = 1 + dx^2y^2,$$

where a and $d \neq 1$ are distinct, non-zero elements of $\mathbb{F}_q$.

Twisted Edwards curves and Montgomery curves are strongly related in the sense that every twisted Edwards curve is birationally equivalent to a Montgomery curve over $\mathbb{F}_q$ [5, Theorem 3.2], this equivalence is given by

$$\begin{aligned} \phi : E_{a,d} &\to E_{A,B} \\ (x, y) &\mapsto \left(\frac{1+y}{1-y}, \frac{1+y}{(1-yx)}\right), \end{aligned}$$

where $A := \frac{2(a+d)}{(a-d)}$ and $B := \frac{4}{a-d}$. Conversely,

$$\begin{aligned} \psi : E_{A,B} &\to E_{a,d} \\ (x, y) &\mapsto \left(\frac{x}{y}, \frac{x-1}{x+1}\right), \end{aligned} \tag{2.5}$$

where $a := \frac{A+2}{B}$ and $d := \frac{A-2}{B}$. Let $E_{a,d}$ be a Twisted Edwards curve then, there exists the following correspondence with its equivalent Montgomery Curve $E_{(A:C)}$

$$A_{24p} := A + 2C = a, \qquad A_{24m} := A - 2C = d \quad \text{and} \quad C_{24} := 4C = a - d,$$

As we can see in map ϕ the x-coordinate of the image only depend of the y-coordinate of the input and the opposite for the map ψ, then is natural to think in a y-only arithmetic for Edwards curves. In order to avoid notation ambiguities, we will write the $\mathbb{P}^1$ projection of the y-coordinate of the point $P \in E_{a,d}$ as $(Y_p : T_p)$. Let $P \in E_{(A:C)}$, from equation 2.5 we have the map,

$$(X_P : Z_P) \mapsto (X_P - Z_P : X_P + Z_P) = (Y_P : T_P) \tag{2.6}$$

A work of Castrick, Galbraith and Farashahi[14] put this in practice and propose a YT-doubling for Edwards curves given by

$$\begin{aligned} Y_{[2]P} :=& -(T_P^4 + dY_P^4) + 2Y_P^2T_P^2, \\ T_{[2]P} :=&(T_P^4 + dY_P^4) - 2dY_P^2T_P^2, \end{aligned}$$

Which can be computed at cost of 1**M** + 3**S** + 3**C**[2] if d is a square and 5**S** + 2**C** if d is non-square. For Twisted Edwards curves they said that the cost is 1**M** + 3**S** +6**C** if $a \cdot d$ is square and 5**S** + 4**C** in general.

Isogenies

Dustin Moody presented formulas for isogenies between Edwards curves [69] and more precisely, twisted Edwards curves which are of our particular interest.

Lemma 2.4.4. ([69, Corollary 1]) Suppose F is a subgroup of the twisted Edwards curve $E_{a,d}$ with odd order $s = 2\ell + 1$, where F is the set of points

$$F = \{(\infty), (\pm\alpha_1, \beta_1), \ldots, (\pm\alpha_\ell, \beta_\ell)\}$$

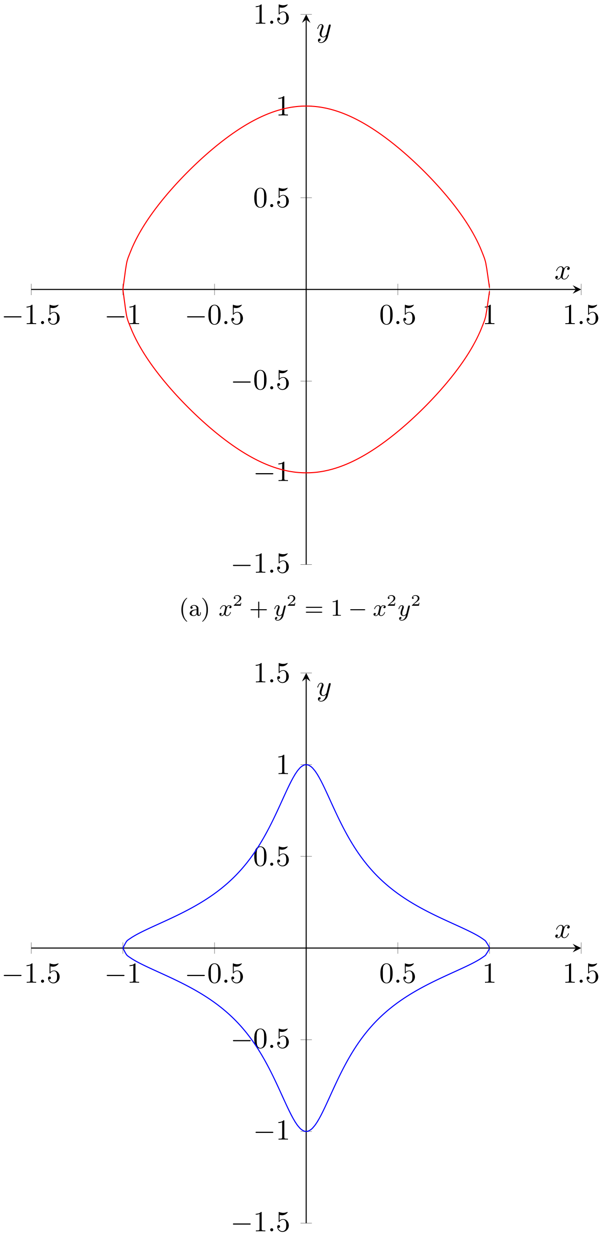
Define

$$\psi(Q) = \left(\prod_{P \in F \setminus \{\infty\}} \frac{x_{Q+P}}{x_P}, \prod_{P \in F \setminus \{\infty\}} \frac{y_{Q+P}}{y_P} \right).$$

Then ψ is an s-isogeny, with kernel F, from the curve $E_{a,d}$ to the curve $E_{a',d'}$ where $a' := a^s, d' = B^8d^s$ and $B = \prod_{i=1}^{\ell} \beta_i$.

We can observe, as was pointed by Meyer and Reith [66] ,that parameters a' and d' depend only of a, d and the y-coordinates of points in F, so we can also consider Edwards YT-coordinates *i.e.* , consider $\mathbf{y}(P) := \beta_i = (Y_{P_i} : T_{P_i})$ for $P \in F \setminus \infty$ and obtain that $a' := B_T^8a^s$ and $d' := B_Y^8d^s$ where $B_Y := \prod_{i=1}^{\ell} Y_{P_i}$ and $B_T := \prod_{i=1}^{\ell} T_{P_i}$. The cost of computing this point-projective version of a' and d' is about $(2\ell + 2 + \log(s)/2)$**M** + $(6 + \log(s))$**S** .

[2] **C** is the cost of compute multiplication by a constant, in this case, one multiplication by d



(a) $x^2 + y^2 = 1 - x^2y^2$

(b) $x^2 + y^2 = 1 - 30x^2y^2$

Figure 2.3: Two examples of Edwards curves over rational numbers

Chapter 3

Koblitz curves over quadratic fields

3.1 Introduction

In 1985, Victor Miller[67] and Neal Koblitz [53] independently show that th e group of points in an elliptic curve defined over a finite field could be used for designing a public key cryptosystem, having the DLP in that group as underlying hard computational problem. This was the birth of Elliptic Curve Cryptography (ECC), which during which across the years has become one of the most intensively analyzed public key schemes in our discipline[1].

Since their introduction in 1991 by Koblitz [54], these curves have been extensively studied for their additional structure that allows, in principle, a performance speedup in the computation of the elliptic curve point multiplication operation. Koblitz curves defined over an extension of $\mathbb{F}_2$ were also proposed in [54]. Nevertheless, until now the research works related to binary elliptic curves such as the binary curves standarized by NIST [70, 73, 71] or the suite of elliptic curves supported by the TLS protocol [36, 10], have exclusively analyzed the security and performance of curves defined over binary extension fields $\mathbb{F}_{2^m}$, with m a prime number (for recent examples see [2, 11, 96, 104]).

[1] see [55] for a historical recount of the first three decades of elliptic curve cryptography

Koblitz curves over quadratic fields

We find interesting to explore the cryptographic usage of Koblitz curves defined over $\mathbb{F}_4$ due to their inherent usage of quadratic field arithmetic. Indeed, it has been recently shown [63, 76, 78] that quadratic field arithmetic is extraordinarily efficient when implemented in software. This is because one can take full advantage of the Single Instruction Multiple Data (SIMD) paradigm, where a vector instruction performs simultaneously the same operation on a set of input data items.

Quadratic extensions of a binary finite field $\mathbb{F}_{q^2}$ can be defined by means of a monic polynomial of degree two $h(u) \in \mathbb{F}_2[u]$ irreducible over $\mathbb{F}_q$. The field $\mathbb{F}_{q^2}$ is isomorphic to $\mathbb{F}_q[u]/(h(u))$ and its elements can be represented as $a_0 + a_1 u$, with $a_0, a_1 \in \mathbb{F}_q$. The addition of two elements $a, b \in \mathbb{F}_{q^2}$, can be performed as $c = (a_0 + b_0) + (a_1 + b_1)u$. Using $h(u) = u^2 + u + 1$, the multiplication of a, b can be computed as, $d = a_0 b_0 + a_1 b_1 + ((a_0 + a_1) \cdot (b_0 + b_1) + a_0 b_0)u$. By carefully organizing the code associated to these arithmetic operations, one can greatly exploit the instruction-level parallelism of the pipelines that are available in contemporary high-end processors.

3.2 Koblitz curves over $\mathbb{F}_4$

Koblitz curves over $\mathbb{F}_4$ are defined by the following equation

$$E_a : y^2 + xy = x^3 + a\gamma x^2 + \gamma, \tag{3.1}$$

where $\gamma \in \mathbb{F}_4$ satisfies $\gamma^2 = \gamma + 1$ and $a \in \{0, 1\}$. The number of points in the curves $E_0/\mathbb{F}_4$ and $E_1/\mathbb{F}_4$ are 4 and 6, respectively. For cryptographic purposes, one uses Eq. (3.1) operating over binary extension fields of the form $\mathbb{F}_q$, with $q = 4^m$, and m a prime number.

The Frobenius map $\tau : E_a(\mathbb{F}_q) \rightarrow E_a(\mathbb{F}_q)$ defined by $\tau(\infty) = \infty$, $\tau(x, y) = (x^4, y^4)$, is a curve automorphism satisfying $(\tau^2 + 4)P = \mu\tau(P)$ for $\mu = (-1)^a$ and all $P \in E_a(\mathbb{F}_q)$. By solving the equation $\tau^2 + 4 = \mu\tau$, the Frobenius map can be seen as the complex number $\tau = (\mu \pm \sqrt{-15})/2$.

Remark 7. Notice that the Frobenius map of Definition 2.4.3 is not an endomorphism on this curves and we need to adapt it to the $\mathbb{F}_4$ case.

3.3 Extended Koblitz curves over $\mathbb{F}_4$

There exist several ordinary elliptic curves over $\mathbb{F}_4$ that strictly speaking cannot be considered Koblitz curves in the way that they were defined in § 3.2. Since some of these additional curves come out equipped with additional endomorphisms, they are also of cryptographic interest. This *extended* set of Koblitz curves can be better described using isogeny classes as discussed next. It is known that two curves E_0 and E_1 are isogenous over $\mathbb{K}$ if and only if they have the same number of points [95, Theorem 1]. This fact helps to classify elliptic curves by isogeny classes, i.e, by their point cardinality.

3.3.1 The twelve ordinary elliptic curves over $\mathbb{F}_4$

The description of Koblitz curves given in § 3.2 define four ordinary elliptic curves. Nevertheless, there exist a total of twelve ordinary elliptic curves over the field $\mathbb{F}_4$. As shown in Table 3.1, these twelve ordinary elliptic curves define four different isogeny classes.

	$E_{(a,b)}/\mathbb{F}_4 : y^2 + xy = x^3 + ax^2 + b$			
Isogeny class	0	1	2	3
$\#E_{(a,b)}$	8	6	4	2
Parameters (a, b)	$(1,1)$, $(0,1)$.	(γ,γ), (γ^2,γ^2), (γ,γ^2), (γ^2,γ).	$(1,\gamma)$, $(1,\gamma^2)$, $(0,\gamma)$, $(0,\gamma^2)$.	$(\gamma,1)$, $(\gamma^2,1)$.

Table 3.1: The twelve ordinary elliptic curves $E_{(a,b)}/\mathbb{F}_4 : y^2 + xy = x^3 + ax^2 + b$, define four isogeny classes. The curve parameters $a, b \in \mathbb{F}_4$, can take the values $[0, 1, \gamma, \gamma^2]$, with $\gamma \in \mathbb{F}_4 \setminus \mathbb{F}_2$.

Notice that the Koblitz curves described in § 3.2 corresponds to the isogeny classes 1 and 2 of Table 3.1, with curve parameters (a, b) given as, (γ, γ), (γ^2, γ^2), $(0, \gamma)$, and $(0, \gamma^2)$. Since these two classes were already studied, in the following we will focus our attention to the isogeny classes 0 and 3.

The curves $E_{(1,1)}$ and $E_{(0,1)}$ are the only two members of the isogeny class 0. Notice that the curve parameters a, b of these curves lie in $\mathbb{F}_2$. Furthermore, it can be shown that $E_{(1,1)}$ and $E_{(0,1)}$ become isomorphic over $\mathbb{F}_4$ and that $(\#E_{(0,1)}(\mathbb{F}_2) \cdot \#E_{(1,1)}(\mathbb{F}_2)) \mid \#E_0(\mathbb{F}_4)$, where E_0 is the Koblitz curve defined in Eq. (3.1) with $a = 0$. This observation can be generalized to prove that $(\#E_{(0,1)}(\mathbb{F}_{2^m}) \cdot \#E_{(1,1)}(\mathbb{F}_{2^m})) \mid \#E_0(\mathbb{F}_{4^m})$.We sketch the proof as follows. Figure 3.1 shows the relation between the curves E_0 and E_1, we know that points lying in $E_0(\mathbb{F}_{2^m})$ belongs to the rational points of E_0 over $(\mathbb{F}_{4^m})$, moreover is a subgroup of $E_0(\mathbb{F}_{4^m})$. Same argues are valid for the curve E_1 when defined over $\mathbb{F}_2, \mathbb{F}_{2^m}$ and $\mathbb{F}_{4^m}$. Now, as $E_0/\mathbb{F}_{4^m}$ and $E_1/\mathbb{F}_{4^m}$ are isomorphic, then $\#E_0(\mathbb{F}_{4^m}) = \#E_1(\mathbb{F}_{4^m})$. As we state before, $E_0(\mathbb{F}_{2^m})$ is a subgroup of $E_0(\mathbb{F}_{4^m})$ then, $\#E_0(\mathbb{F}_{2^m}) \mid \#E_0(\mathbb{F}_{4^m})$ moreover, $\#E_0(\mathbb{F}_{2^m}) \mid \#E_1(\mathbb{F}_{4^m})$ because $E_0 \equiv E_1$ when defined over $\mathbb{F}_{4^m}$. Therefore $(\#E_{(0,1)}(\mathbb{F}_{2^m}) \cdot \#E_{(1,1)}(\mathbb{F}_{2^m})) \mid \#E_0(\mathbb{F}_{4^m})$. A direct consequence of this relation is that the largest prime factor of $\#E_0(\mathbb{F}_4)$ must be smaller than $\#E_{(0,1)}(\mathbb{F}_{2^m}) \approx \#E_{(1,1)}(\mathbb{F}_{2^m}) \approx 2^m$. Thus, when the two curves in the isogeny class 0 are defined over the field $\mathbb{F}_{4^m}$ one can only hope to achieve at most an $\frac{m}{2}$-bit security level. We conclude that the isogeny class 0 is of little or no cryptographic value.

$$\begin{tikzcd}
E_0/\mathbb{F}_{4^m} \arrow[r, leftrightarrow] \arrow[d] & E_1/\mathbb{F}_{4^m} \arrow[d] \\
E_0/\mathbb{F}_{2^m} \arrow[d] & E_1\mathbb{F}_{2^m} \arrow[d] \\
E_0/\mathbb{F}_2 & E_1/\mathbb{F}_2
\end{tikzcd}$$

Figure 3.1: Diagram showing the behavior of the curves E_0 and E_1 when defined over the field extensions of our interest

3.3.2 A Novel endomorphism for the isogeny Class 3

The curves $E_{(\gamma,1)}$ and $E_{(\gamma^2,1)}$ are the only two members of the isogeny class 3 shown in Table 3.1. Both of these two curves are equipped with an efficient endomorphism as discussed next.

It can be seen that the Frobenius mapping $\tau_2 : E_{(a,b)}(\mathbb{F}_q) \to E_{(a',b')}(\mathbb{F}_q)$ defined by $\tau(\infty) = \infty$, $\tau(x, y) = (x^2, y^2)$, is a two-degree isogeny such that, $\tau_2(E_{(\gamma,1)}(\mathbb{F}_q)) = E_{(\gamma^2,1)}(\mathbb{F}_q)$, and $\tau_2(E_{(\gamma^2,1)}(\mathbb{F}_q)) = E_{(\gamma,1)}(\mathbb{F}_q)$.

Moreover, the curves $E_{(\gamma,1)}$ and $E_{(\gamma^2,1)}$ are also isomorphic, since one can define the isogenies, $\phi_0 : E_{(\gamma,1)} \to E_{(\gamma^2,1)}$ and $\phi_1 : E_{(\gamma^2,1)} \to E_{(\gamma,1)}$ such that, $\phi_0(x, y) = (x, y + \gamma \cdot x)$ and $\phi_1(x, y) = (x, y + \gamma^2 \cdot x)$. As illustrated in Figure 3.2, for each one of the curves in the class 3 one can therefore build two novel endomorphisms $\bar{\tau}_0, \bar{\tau}_1$ as follows,

$$\bar{\tau}_0(x, y) = (\phi_0 \circ \tau_2)(x, y) = (x^2, y^2 + \gamma \cdot x^2)$$
$$\bar{\tau}_1(x, y) = (\phi_1 \circ \tau_2)(x, y) = (x^2, y^2 + \gamma^2 \cdot x^2).$$

Using affine λ-coordinates as defined in [78], the endomorphisms $\bar{\tau}_0$ and $\bar{\tau}_1$ can be written as, $\bar{\tau}_0(x, \lambda) = (x^2, \lambda^2 + \gamma)$ and $\bar{\tau}_0(x, \lambda) = (x^2, \lambda^2 + \gamma^2)$, respectively. Using projective λ-coordinates they become $\bar{\tau}_0(x, \lambda, z) = (x^2, \lambda^2 + \gamma \cdot z^2, z^2)$ and $\bar{\tau}_0(x, \lambda) = (x^2, \lambda^2 + \gamma^2 \cdot z^2, z^2)$, respectively.

Since $\bar{\tau}_0$ and $\bar{\tau}_1$ satisfy the same properties we will use in the following $\bar{\tau}$ to refer both of them. We stress that $\bar{\tau}$ is computationally cheaper than the endomorphism τ of the Koblitz curves discussed in the previous §. Indeed, the computational cost of $\bar{\tau}$ is of two squaring operations instead of the four squaring operations associated to τ. As it will be further discussed in §3.5, this computational saving induces an important reduction in the number of pre-computed points for the point multiplication $Q = [k]P$ that uses a width-w τNAF scalar representation.

$$\begin{tikzcd}
E_{(\gamma,1)} \arrow[d, "\tau_2"'] \\
E_{(\gamma^2,1)} \arrow[u, bend right=90, "\phi_0"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
E_{(\gamma^2,1)} \arrow[d, "\tau_2"'] \\
E_{(\gamma,1)} \arrow[u, bend right=90, "\phi_1"']
\end{tikzcd}$$

Figure 3.2: Construction of the endomorphisms $\bar{\tau}_0$ and $\bar{\tau}_1$ for the isogeny class 3 elliptic curves $E_{(\gamma,1)}$ and $E_{(\gamma^2,1)}$.

Another interesting property of the $\bar{\tau}$ endomorphism is that, $\bar{\tau}^2(x, y) = (x^4, y^4 + x^4) = -\tau(x, y)$. Moreover, for the elliptic curves in the isogeny class 3, τ satisfies the equation $\tau^2 + 4 = 3\tau$, which implies that, $\bar{\tau}^2 + \bar{\tau} = -2$. It also follows that, $\bar{\tau} = (1 + \sqrt{(- 7)})/2$. Since the ring $\mathbb{Z}[(-1+\sqrt{-7})/2]$ has been extensively studied in the literature, we can adopt the same existing methods reported in [2, 11, 96, 98] for performing the τw-NAF scalar recoding.

We computed the cardinality of the elliptic curves belonging to the isogeny class 3 defined over the field $\mathbb{F}_{4^m}$ with m a prime extension in the range $[127, 191]$. From this experiment we found out that the only extension of cryptographic interest is $m = 163$. Indeed, for this extension field $\mathbb{F}_{4^{163}}$, the cardinality of the elliptic curves in the class 3 has the following integer factorization,

$$0x2 \cdot 0x28D \cdot 0xC8B90A95C20EE5BBC91D671B0CEFED2E\backslash$$
$$A701F5CE9AAA522F37A4E0D020A19EBBDC1D0437C458139.$$

The largest prime factor above has a size of 316 bits. Hence, its associated security level is of around 158 bits. This curve is comfortably above the 128-bit security level (even considering the criterion that for a given field extension m, a binary curve offers 10 bits less of security than the number $\lfloor \frac{m}{2} \rfloor$).

3.4 Semisplit μ_4-normal form-Normal form

In this section we study how to send our curve $E_{(0,u)}$ to a μ_4-normal form in particular to a semisplit μ_4-normal form because as Kohel mentioned in [56, Appendix] there are addition formulas free of multiplication by constants. Our landscape is as follows, we know how to send a Koblitz curve to a Twisted μ_4-normal form (Lemma 2.4.2), how to send a μ_4-normal form to a Koblitz curve (Lemma 2.4.1), how to map a split μ_4-normal form into a semisplit μ_4-normal form and how to map a semisplit μ_4-normal form into a μ_4-normal form.

If we apply Lemma 2.4.2 to our curve $E_{0,u}$ we get the curve

$$X_0^2 + uX_2^2 = X_1X_3, \quad X_1^2 + X_3^2 = X_0X_2,$$

which is in μ_4-normal form with parameter $r = u$. If we setting $s = 1/r = u$ we claim that the transformation

$$(X_0, X_1, X_2, X_3) \mapsto (X_0, X_1, cX_2, X_3),$$

maps the split μ_4-normal form to the semisplit μ_4-normal form.

Lemma 3.4.1. Let C be a binary elliptic curve in semisplit μ_4-normal form

$$(X_0 + X_2)^2 = X_1X_3, \quad (X_1 + X_3)^2 = bX_0X_2.$$

Then C is isomorphic to the elliptic curve

$$y^2 + xy = x^3 + b,$$

in Weierstrass form via the map $(X_0 : X_1 : X_2 : X_3) \mapsto (X_1 + X_3 : X_0 + X_1 : bX_2)$. On affine points (x, y) the inverse is $(x, y) \mapsto (x^2 : x^2 + y : b^2 : x^2 + x + y)$.

Proof. This map is the result of the composition of the map from Lemma 2.4.2 and the inverse of the map on Definition 2.4.4. □

Then we can make use of the faster doubling and addition formulas proposed in [56].

3.5 Results and discussion

Our software library was designed for 64-bit high-end desktops, provided with SSE 4.1-equivalent vector instructions and a 64-bit carry-less multiplier. The timings were measured in an Intel Core i7 4770k 3.50 GHz machine (Haswell architecture) with the Turbo Boost and Hyper-Threading technologies disabled. The implementation was coded in the GNU11 C and Assembly languages. We compiled our code with the GCC (Gnu Compiler Collection) version 5.4 with the optimization flags `--march=haswell -fomit-frame-pointer -O3`.

We present in Table 3.2 our results of scalar multiplication for curve $E_{(u,1)}/\mathbb{F}_{4^{163}}$ and different width w for window τ-NAF using left-to-right and right-to-left scalar multiplication. We observe that the Right-to-left method using a window size of $w = 4$ give us the best result. Table 3.3 show the comparison against the current state-of-the-art of 128-bit secure timing-resistant scalar multiplication on binary and prime curves.

Our $E_{(u,1)}/\mathbb{F}_{4^{163}}$ is only 6,000 clock cycles more expensive than the K-283 implementation in [76], but offers about 15 extra bits of security. Both curves works with the same Frobenius endomorphism τ and we can see that the quadratic field arithmetic plays an important role in the efficiency of the field arithmetic.

Left-to-right τ-NAF	cost (clock cycles)	ratio cycles/bit
$w = 2$	234,880	752.56
$w = 3$	144,988	464.70
$w = 4$	116,560	373.58
$w = 5$	115,420	369.93
Right-to-left τ-NAF	cost (clock cycles)	ratio cycles/bit
$w = 2$	221,132	708.75
$w = 3$	128,980	413.39
$w = 4$	105,952	339.58
$w = 5$	116,888	374.64

Table 3.2: Timings (in clock cycles) for the τ-NAF scalar multiplication on $E_{(0,u)}/\mathbb{F}_{4^{163}}$

Curve/Method	Timings
Koblitz over $\mathbb{F}_{2^{283}}$ (τ-and-add, 5-τ-NAF) [76]	99,000
GLS over $\mathbb{F}_{4^{127}}$ (2-GLV double-and-add, 5-NAF) [77]	48,300
Twisted Edwards over $\mathbb{F}_{(2^{127}-1)^2}$ (double-and-add) [25]	56,000
Kummer genus-2 over $\mathbb{F}_{2^{127}-1}$ (Kummer ladder) [6]	60,556
Koblitz over $\mathbb{F}_{4^{149}}$ (τ-and-add, 3-τNAF) [79]	82,872
Koblitz over $\mathbb{F}_{4^{163}}$ (τ-and-add, 4-τNAF) (**this work**)	105, 952

Table 3.3: A comparative between state-of-the-art software implementations of 128-bit secure timing-resistant scalar multiplication. The timings were measured in the Haswell platform and are given in clock cycles

3.6 Conclusion and future work

In this section we present a new kind of Koblitz curves defined over $\mathbb{F}_4$ and a taxonomy of all ordinary curves defined over the same field. This taxonomy is based on the isogeny class of such curves. We present a novel endomorphism which allows some Koblitz curves over $\mathbb{F}_4$ use the well-known τ-NAF and it variants for Koblitz curves defined over $\mathbb{F}_2$. This endomorphism is more efficient than the Frobenius map for the classical Koblitz curves over $\mathbb{F}_4$ (class 1 and 2 of) allowing class 3 to be a good candidate for cryptographic purposes. Our curve $E_{(u,1)}/\mathbb{F}_{4^{163}}$ offers more security than the current state-of-the-art allowing a conservative and faster alternative to the standard curves. Regarding to semisplit μ_4-normal form-normal form, we did not implemented because, even when those curves have a faster addition formulas, they require to operate in four coordinates instead of the three used in proyective lambda coordinates. This increase the size of the tables and the cost of the countermeasures for cache attacks. Nevertheless, as a future work we need to compare the faster ladder proposed by Kohel in [56] against the one we used. The class 1 of Table 3.1 has been studied in the state-of-the-art, and in this document we analyze the class 3, but there is necessary a study of class 2 or even to make a taxonomy for curves defined over other extensions like $\mathbb{F}_{2^3}$ or $\mathbb{F}_{2^4}$.

Chapter 4

An introduction to Isogeny-based cryptography

“Stranger”, replied the Man Who Counted, “I do not disapprove of this curiosity that disturbs the peace of my thoughts and calculations. And now that you have spoken to me with such courtesy and graciousness, I am going to accede to your wishes. But first I must tell you”[...] (about isogenies)[1]
Malba Tahan - *The Man Who counted*

4.1 Introduction

Isogeny-based cryptography was proposed by Couveignes in 1997. Complete details of his proposal were eventually reported in [28] . In 2006, Couveignes’ protocol was independently

[1] Original quote: “Stranger,” replied the Man Who Counted, “I do not disapprove of this curiosity that disturbs the peace of my thoughts and calculations. And now that you have spoken to me with such

rediscovered by Rostovtsev and Stolbunov in [85, 93]. Also in 2006, Charles-Lauter-Goren introduced in [19] the hardness of path-finding in supersingular isogeny graphs and its application to the design of hash functions. Later in 2011, the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was proposed by Jao and de Feo in [49]. Later in 2017, the Supersingular Isogeny Key Encapsulation (SIKE) protocol, which can be seen as a descendant of SIDH, was submitted to the NIST post-quantum cryptography standardization project [3]. The isogeny-based protocol SIKE is one of the seventeen key-exchange schemes accepted for the second round of the NIST contest.

The two most costly computational tasks of SIDH are, (i) the computation of large smooth-degree isogenies of supersingular elliptic curves along with the evaluation of the image of elliptic curve points in those isogenies and; (ii) elliptic curve scalar multiplication computations via three-point Montgomery ladder procedures. The optimal computation of large degree isogenies for single-core processors was presented and solved in [32]. Also, efficient algorithms for computing the SIDH three-point scalar multiplications can be found in [49, 38]. Several general ideas for a sensible improving of the SIDH performance were introduced in [26].

Let E be a supersingular elliptic curve defined over the quadratic extension field $\mathbb{F}_{p^2}$. Let $S = \langle R_0 \rangle$ be an order-ℓ^e subgroup of $E[\ell^e]$, where $R_0 \in E(\mathbb{F}_{p^2})$, is a point of order ℓ^e, e is a positive number, and ℓ is a (power) of a small prime. Then there exists a degree-ℓ^e isogeny $\phi : E \rightarrow E'$ having kernel S. The image curve E' is also a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Moreover, $\#E = \#E'$ [95, Theorem 1]. In this thesis, the computational task of finding E' will be referred as isogeny construction. Furthermore, given a point $P \in E(\mathbb{F}_{p^2})$ such that $P \notin \mathrm{Ker}(\phi)$, a closely related problem is that of finding $\phi(Q)$, *i.e.*, the image of the point Q over E'. We will refer to this computation as isogeny evaluation.

In [32], optimal strategy techniques were introduced to efficiently compute degree-ℓ^e isogenies at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.[1] The strategies described in [32] are provable optimal for those architectures equipped with a single unit of processing, i.e., single-core platforms. Virtually all SI(DH/KE) implementations published as of today, compute degree-ℓ^e isogenies using optimal strategies.

Optimal strategies can be depicted as a weighted directed graph whose vertices are elliptic curve points and whose vertical and horizontal edges have as associated weight the cost of performing one scalar multiplication by ℓ and one degree-ℓ isogeny, respectively. That weighted directed graph can be drawn as a right triangular lattice Δ_e having $\frac{e(e+1)}{2}$ points distributed in e columns and rows. A leaf is defined as the most bottom point of a given column in that lattice. All vertical edges must be computed sequentially, whereas all the horizontal edges can be computed in parallel. At the beginning of the isogeny

courtesy and graciousness, I am going to accede to your wishes. But first I must tell you the story of my life. ”

[1] An analysis of the computational cost of small degree isogeny construction and evaluation can be found in [26, 23, 18].

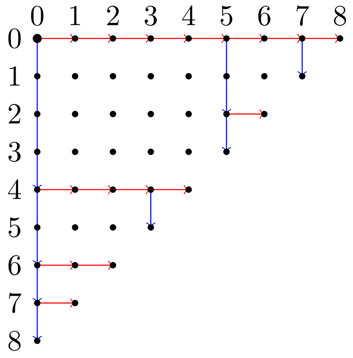


Figure 4.1: Strategy to compute a degree-$\ell^e = 4^9$ isogeny. The root point at row and column zero, represents the elliptic curve point R_0 of order 4^9. Each one of the nine leaves at the bottom of the columns represent elliptic curve points of order 4. The sequential cost of this strategy is of thirteen scalar multiplications by four plus sixteen degree-4 isogeny evaluations.

computation, only the point R_0 of order ℓ^e is known. The isogeny computation is carried out by obtaining from left to right, each one of the leaves in Δ_e until the farthest right one, R_{e-1}, is computed. Then, $\phi : E \to E'$ can be found by calculating a degree-ℓ isogeny with kernel R_{e-1}.

An interesting consequence of the weighted directed graph representation is that one can abstract oneself from the cryptographic nature of the isogeny computation problem, and solely focus in the combinatorial structure associated to the graph.

As an illustrative example, consider the toy example depicted in Figure 4.1 using the parameters $\ell^e = 4^9$. In the event that that strategy is executed on a single-core platform, it would have an associated timing cost of thirteen scalar multiplications by 4 (corresponding to the thirteen vertical blue edges shown in the graph), plus sixteen degree-4 isogeny evaluations (corresponding to the sixteen horizontal red edges shown in the graph).[2] However, if one happens to have four cores available for performing this task, then the timing cost can be reduced to thirteen scalar multiplications by 4, plus just eight degree-4 isogeny evaluations.

4.2 SIDH

The security of the SIDH key agreement protocol is based on the intractability of the Computational Supersingular Isogeny (CSSI) problem, which consists of computing $\mathbb{F}_{p^2}$-rational isogenies of degrees 2^e and 3^e between pairs of supersingular elliptic curves defined over

[2] The cost of computing the strategy shown in Figure 4.1 also includes ten degree-ℓ isogeny constructions not relevant for the discussion here.

$\mathbb{F}_{p^2}$. It is believed [1, 50] that the van Oorschot-Wiener golden collision finding algorithm is the best classical or quantum attack on CSSI, having an expected running time of $O(p^{1/4})$.

Because of its credible security arguments, the supersingular-isogeny Diffie-Hellman protocol (SIDH) has become a strong candidate for post-quantum cryptography. In addition, thanks to the high complexity of its underlying hard problem, SIDH provides key sizes comparable to those of the public-key cryptosystems currently in use.

The main computational tasks associated to the implementation of the SIDH protocol include the computation of large degree isogenies and the evaluation of elliptic curve points in those isogenies. Another important task of this scheme is to compute several elliptic curve scalar multiplications over the quadratic field $\mathbb{F}_{p^2}$. It is noticed that in a typical software or hardware implementation of SIDH, the isogeny computations and evaluations can take 70-80% of the whole computation, whereas the elliptic curve scalar multiplications may require up to 30% of the total computational cost of the SIDH protocol.

Since the publication of [49], researchers have focused their efforts on trying to come out with algorithmic improvements for the SIDH protocol. Efficient algorithms for computing the SIDH three-point scalar multiplications can be found in [49, 38]. Likewise, the computation of large degree isogenies for single-core processors was thoroughly addressed in [49, 32].

SIDH protocol description

Let $p = 4^{e_A}3^{e_B} - 1$ be a prime, so that $4^{e_A} \approx 3^{e_B} \approx p^{1/2}$. Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. In addition, let $P_A, Q_A \in E[4^{e_A}]$ be two points of order 4^{e_A}, and $P_B, Q_B \in E[3^{e_B}]$ be two points of order 3^{e_B} such that $E[4^{e_A}] = \langle P_A, Q_A \rangle$ and $E[3^{e_B}] = \langle P_B, Q_B \rangle$. In SIDH, e_A, e_B, p and E and the bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$, are all considered public domain parameters.

Alice begins the key generation phase by selecting her secret $m_A \in_R [0, 4^{e_A} - 1]$. Then she computes $R_A = P_A + [m_A]Q_A$. Thereafter, Alice constructs the isogeny $\phi_A : E \to E/A$ and while computing E/A, *simultaneously* evaluates Bob's public points P_B, Q_B. Alice keeps secret, m_A and R_A. Then she transmits to Bob, $E/A, \phi_A(P_B)$ and $\phi_A(Q_B)$.

Analogously, Bob selects $m_B \in_R [0, 3^{e_B} - 1]$ to compute $R_B = P_B + [m_B]Q_B$. Bob then constructs the isogeny $\phi_B : E \to E/B$, and while computing E/B, *simultaneously* evaluates Alice's public points P_A, Q_A. He keeps secret, m_B and R_B. Then he transmits to Alice $E/B, \phi_B(P_A)$ and $\phi_B(Q_A)$. This action ends the SIDH key generation phase.

Starting the SIDH shared secret phase, Alice computes $\phi_B(R_A) = \phi_B(P_A) + [m_A]\phi_B(Q_A)$ and uses this point to construct $(E/B)/\langle \phi_B(R_A) \rangle$. Meanwhile, Bob computes $\phi_A(R_B) = \phi_A(P_B) + [m_B]\phi_A(Q_B)$ and uses this point to construct $(E/A)/\langle \phi_A(R_B) \rangle$.

These two actions complete the secret shared phase. As a result, both of the compositions of isogenies $E \to E/A \to (E/A)/\langle \phi_A(R_B) \rangle$ and $E \to E/B \to (E/B)/\langle \phi_B(R_A) \rangle$, have kernel $\langle R_A, R_B \rangle$. Hence, the elliptic curves computed by Alice and Bob are isomorphic over $\mathbb{F}_{p^2}$, and their shared secret is the j-invariant of these curves.

Remark 8. In the SIDH key exchange protocol, the key generation phase is always more expensive than the shared secret one. This is because in the former phase Alice and Bob must compute not only the isogenies ϕ_A, ϕ_B, but they also have to evaluate the other party's public points namely, $(\phi_B(P_A), \phi_B(Q_A))$ and $(\phi_A(P_B), \phi_A(Q_B))$, respectively

Remark 9. In order to compute the points $R_A, \phi_B(R_A), R_B$ and $\phi_A(R_B)$, Alice and Bob must perform each, two three-point scalar multiplication procedures, which can be efficiently computed using a right-to-left Montgomery ladder procedure [38]. This Montgomery ladder has a per-step cost of one point addition and one point doubling. Due to the fact that these two operations are usually performed in the projective space $\mathbb{P}^1$, we will refer to them as the xADD and the xDBL operations, respectively.

Remark 10. Since for current SIDH state-of-the-art implementations it is observed that the costs of xDBL and xADD are about the same, one can assume that the per-step computational cost of the three-point Montgomery ladder is essentially of two xDBL operations. It follows that the cost of computing R_A is of $4e_A$ xDBL operations.

4.3 Sequential strategies for large smooth-degree isogenies

As mentioned in the introduction, any strategy that successfully constructs/evaluates a degree-ℓ^e isogeny can be associated with a subgraph St_e of a weighted directed graph Δ_e. In this thesis, Δ_e is depicted as a right triangular lattice with e rows and columns. The triangular lattice Δ_e has exactly $\frac{e(e+1)}{2}$ points and e *leaves*, which are defined as the most bottom points in each one of the e columns of the lattice. For the sake of convenience, we will often refer to the directed graph Δ_e as a triangle of size e. The points of Δ_e represent elliptic curve points. The vertical lines indicate scalar multiplications by ℓ, whereas the horizontal lines represent degree-ℓ isogeny evaluations that could in principle be computed in parallel.

Several basic definitions and other useful properties of the lattice Δ_e are summarized in the following subsection.

4.3.1 Walking across Δ_e

Aiming to find a strategy St_e able to compute degree-ℓ^e isogenies, the following navigation rules to walk across the triangular grid Δ_e must be observed.

1. All the vertices of Δ_e are represented as nodes (i, j), with $0 \leq i, j \leq e-1$. The root of St_e is the vertex $(0, 0)$ and represents a point R_0 of order ℓ^e.

2. All the nodes in a row i with $i = 0, 1, \ldots, e-1$, represent points belonging to different elliptic curves. Likewise, all the nodes in a column j with $j = 0, 1, \ldots, e-1$, represent points that belong to the same elliptic curve. All nodes having the same *Manhattan distance* to the vertex $(0, 0)$, represent points having the same order.

3. A *Vertical edge* corresponds to a scalar multiplication by ℓ. For example in Figure 4.1, the edge $[(2,0),(3,0)]$ indicates that starting from the node $[\ell^2]R_0$, one arrives to the node $[\ell^3]R_0$. Every vertical edge has the same weight p_ℓ, which is the computational cost associated to one scalar multiplication by ℓ.

4. A *Horizontal edge* corresponds to a degree-ℓ isogeny evaluation. For example in Figure 4.1, the edge $[(3,0),(3,1)]$ indicates that starting from the node $[\ell^3]R_0$ one arrives to the node $\phi_0([\ell^3]R_0)$. Every horizontal edge has the same weight q_ℓ, which is the computational cost associated to one degree-ℓ isogeny evaluation.

5. The *depth* at column j for $j \in [0, e-1]$, defined as its number of vertices, is of $e-1-j$ vertices.

6. One can only compute a horizontal edge $[(i,j)(i,j+1)]$ with $i \in [0, e-2]$ and $j \in [0, e-i-2]$, provided that one has previously reached the leave of the column j, represented by the vertex $(e-j-1, j)$.

7. All horizontal edges $[(i,j),(i,j+1)]$ are independent of each other and therefore can be computed in parallel.

8. One can only compute the vertical edge $[(i,j),(i+1,j)]$ for $i \in [0, e-2]$ and $j \in [0, e-i-2]$, if either $i = 0$ or the predecessor edge $[(i-1,j),(i,j)]$ has already been visited. Thus, vertical edges in the same column have computational dependencies among them and in general must be computed sequentially.

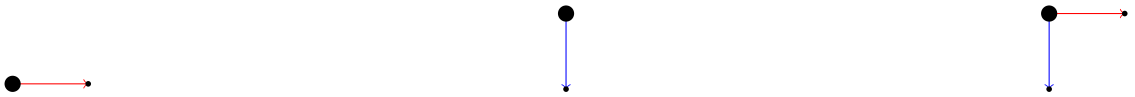
9. A *split* node is a node that has both a vertical edge and a horizontal edge leaving from it. The weight of a split node is the number of nodes between it and either the next split node in the column or the leave in the column. There are always $e-1$ split nodes in any valid strategy.

10. By definition, there are two possible triangles Δ_1 of size one, but only one triangle Δ_2 of size two (cf. Figure 4.2).

4.3.2 Sequential strategies for computing large smooth-degree isogenies

Let Δ_e be the upper-left right triangle of an $e \times e$ grid, so that Δ_e has $\frac{e(e+1)}{2}$ points distributed in e rows and columns. The optimal strategy problem consists of finding a directed-rooted-weighted subtree St_e, such that $\sum_{E \in \text{Edges}(St_e)} w(E)$ is minimum. Here $w(E) \in \{p_\ell, q_\ell\}$ represents the weight of the edge $E \in \text{Edges}(St_e)$.

In the remaining of this subsection, we start by describing first two naive strategies, followed by an approach that finds optimal strategies as presented in [32]

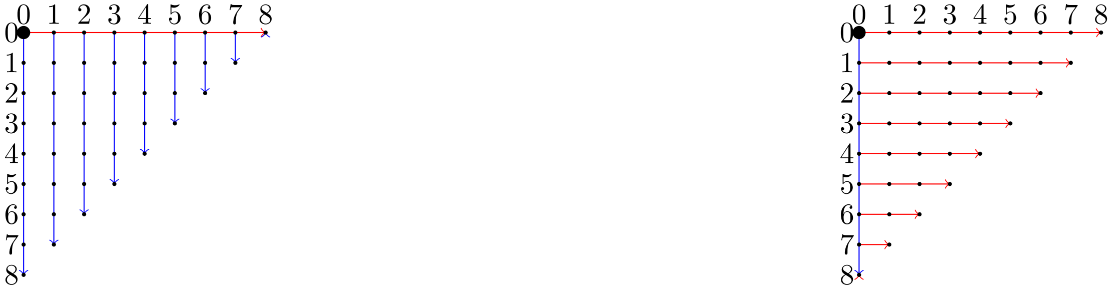


(a) Δ_1

(b) Δ_1

(c) Δ_2

Figure 4.2: The three smallest triangles, Subfigure 4.2a shows a size-1 triangle consisting of its root and one horizontal edge. Subfigure 4.2b shows a size-1 triangle consisting of its root and one vertical edge. Subfigure 4.2c shows the only size-2 triangle having exactly two leaves.



(a) Δ_9

(b)

Figure 4.3: Two basic strategies for computing a degree-ℓ^9 isogeny. Subfigures 4.3a-4.3b illustrate a multiplicative-oriented approach and an isogeny-oriented approach, respectively. Vertical blue lines indicate scalar multiplications by ℓ, whereas horizontal red lines indicate degree-ℓ isogeny evaluations.

Two naive strategies

Two natural albeit naive strategies for computing a degree-ℓ^e isogeny can be summarized as follows.

Suppose that $R \in E(\mathbb{F}_{p^2})$ has order $\ell^e, e \geq 1$. Then the isogeny $\phi : E \to E/\langle R\rangle$ can be computed as follows. Define $E_0 = E$ and $R_0 = R$. For $j = 0, 1, \ldots, e-1$, let $\phi_j : E_j \to E_{j+1}$ be the degree-ℓ isogeny with kernel $\langle \ell^{e-1-j} R_i\rangle$, and let $R_{j+1} = \phi_i(R_j)$. Then $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$. The computational cost associated to the multiplicative-oriented procedure described above is of $\frac{e(e-1)}{2}$ scalar multiplications by ℓ, $e-1$ isogeny evaluations and e isogeny constructions.

A second naive approach to compute a degree-ℓ^e isogeny can be formulated as follows. Define $E_0 = E$. For $i = 0, 1, \ldots, e-1$, compute and store all the e points $R_i^0 = [\ell^i]R$. Compute $\phi_0 : E_0 \to E_1$ such that $\mathrm{Ker}(\phi_0) = \left\langle R_{e-1}^0 \right\rangle$. For $j = 1, \ldots, e-1$ and for $i = 0, \ldots, e-1-j$, compute $R_i^j = \phi_{j-1}(R_i^{j-1})$; followed by $\phi_j : E_j \to E_{j+1}$ such that $\mathrm{Ker}(\phi_j) = \left\langle R_{e-1-j}^j \right\rangle$. The computational cost associated to the isogeny-oriented procedure described above is of $\frac{e(e-1)}{2}$ isogeny evaluations, $e-1$ scalar multiplications by ℓ and e isogeny constructions.

Instantiated for the computation of a degree-ℓ^9 isogeny, Figure 4.3 illustrates the computations that one must perform for the two basic methods previously outlined.

However, we can do much better as discussed next.

Optimal strategies for SIDH

Optimal strategies as defined in [32] exploit the fact that a triangle Δ_e can be optimally and recursively decomposed into two sub-triangles Δ_h and Δ_{e-h} as shown in Figure 4.4. Let us denote as Δ_e^h the design decision of splitting a triangle Δ_e at row h. Then, the sequential cost of walking across the strategy St_e, which is a direct subgraph of Δ_e, is given as

$$C(St_e^h) = C(St_h) + C(St_{e-h}) + (e-h) \cdot q_\ell + h \cdot p_\ell.$$

We say that $\mathsf{S}_e^{\hat{h}}$ is optimal if $C(St_e^{\hat{h}})$ is minimal among all St_e^h for $h \in [1, e-1]$.

Applying this strategy recursively leads to a procedure that computes a degree-ℓ^e isogeny at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.

As an illustrative example, consider the strategy shown in Figure 4.5. Assuming that all the vertical and horizontal edges costs 1 unit, then Subfigure 4.5a shows an optimal partition of Δ_9 into two subtriangles Δ_6 and Δ_3, which can in turn be subdivided into two subtriangles Δ_4 and Δ_2; and Δ_2 and Δ_1, respectively. The strategy shown in Subfigure 4.5b is optimal to traverse Δ_9 for single-core processor platforms.

Remark 11. (*cost of computing an ℓ^e-isogeny*) As shown in [32], a 'balanced strategy' for computing a degree-ℓ^e isogeny requires approximately $\frac{e}{2}\log_2 e$ point multiplications by ℓ,

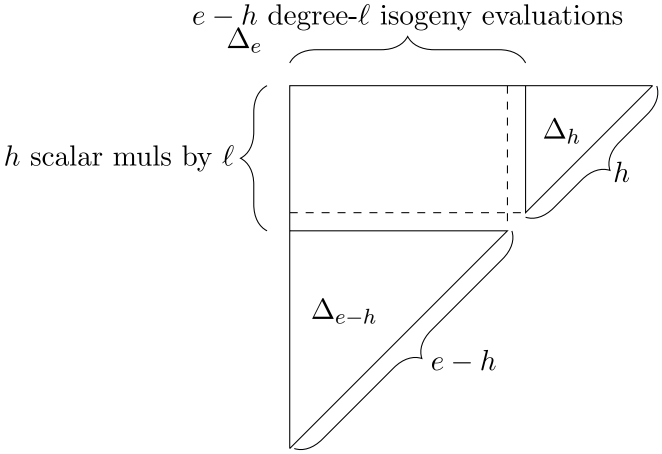

$e - h$ degree-ℓ isogeny evaluations
Δ_e
h scalar muls by ℓ
Δ_h
h
Δ_{e-h}
$e - h$

Figure 4.4: Using an optimal SIDH strategy as in [32], a triangular lattice Δ_e is processed by splitting it into two sub-triangles. After applying this splitting strategy recursively, the cost of computing ϕ drops to approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.

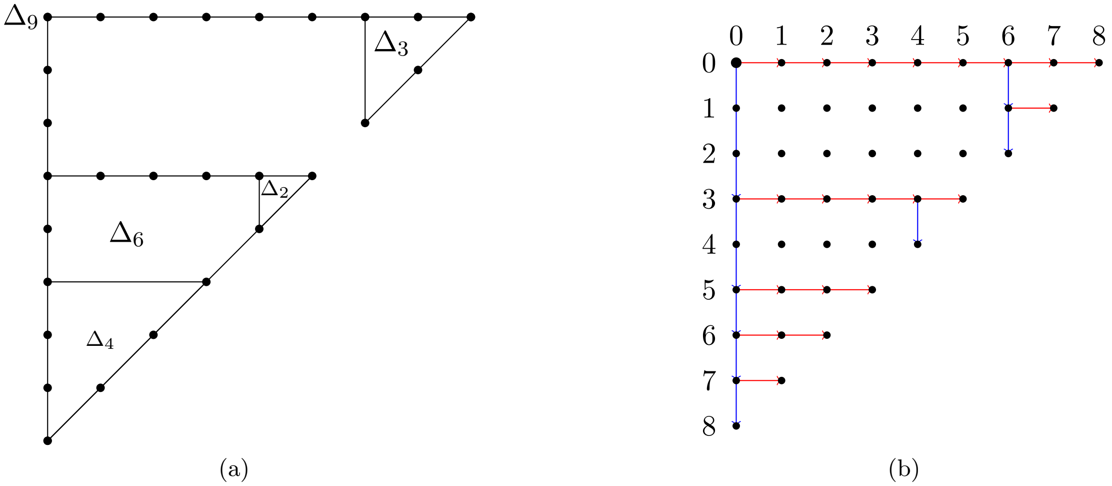


Figure 4.5: Assuming that all the vertical and horizontal edges costs 1 unit, this figure shows an optimal strategy for traversing Δ_9 on single-core processor architectures.

$\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves. Also presented in [32] is a slightly faster 'optimal strategy' that accounts for the relative costs of a point multiplication and a degree-ℓ isogeny evaluation.

4.3.3 Linearizing strategies

In SIKE specification [3, §1.3.7], computational strategies St_e for constructing/evaluating isogenies are described by means of a full binary tree on $e-1$ nodes. The authors of [3] represent such a tree using a so-called *linear representation*, which can be obtained by walking through the tree according to a depth-first left-first ordering and outputting the bifurcations as they are encountered. It is straightforward to apply the same linearization process to the right triangular lattices adopted in this thesis. To this end, one just needs to record the weight of all the split nodes (see Rule 9 in §4.3.1) as they are encountered when combing the triangular lattice by columns from $j = 0$ to $e - 2$. This process is illustrated in the following examples.

Example 1. Referring to the strategies depicted in Figure 4.1 its linear representation is given by $(4, 2, 1, 1, 1, 2, 1, 1)$. The first column has four split nodes of weight $4, 2, 1, 1$, respectively. The other four split nodes are located in the columns three (one), five (two) and seven (one). all of these four split node have weight one, except the first split node of column 5, whose first split node has weight two.

Example 2. Referring to the strategies depicted in Figure 4.3, their linear representation is given as follows,

- Subfigure 4.3a: $(8, 7, 6, 5, 4, 3, 2, 1)$. Each one of the first eight columns of this strategy has only one split node of weight equal to $8 - j$, for $j = 0, \dots, 7$.
- Subfigure 4.3b: $(1, 1, 1, 1, 1, 1, 1)$. All the eight split nodes of this strategy have weight one and are located in the column zero.

Example 3. Referring to the strategies depicted in Figure 4.5 its linear representation is given by $(3, 2, 1, 1, 1, 1, 1, 1)$. The first column has five split nodes of weight $3, 2, 1, 1, 1$, respectively. The other three split nodes are located in the columns four (one) and six (two) and all three of them have weight one.

Executing linearized strategies

A strategy specified as a vector of $e - 1$ split nodes, can be processed as described in Algorithm 3. Algorithm 3 performs a non-recursive walk across the parallel strategy St_e for computing a degree-ℓ^e isogeny.[3] A general procedure for computing linearized strategies is described next.

[3] The interested reader is also referred to [3, Algorithm 19].

1. Initialize the three counters $i, j, k = 0$. Also, initialize a stack *Points*, with the point R_0.

2. Process the element $St_e[i]$ as follows

   (a) Get the top element in *Points*, namely R_t and compute $R'_t = [\ell^{St_e[i]}]R_t$. Then store R'_t in *Points*.

   (b) Assign $j = j + St_e[i]$ and $i = i + 1$.

3. Repeat Step 2 until $j = e - 1 - k$.
   // The leaf node is reached when $j = e - 1 - k$.//

4. Construct a degree-ℓ isogeny ϕ using the top element in *Points*, then remove that element.

5. Find the image of all the points stored in *Points* under the isogeny ϕ.
   //This computes all the horizontal edges from column k to column $k+1$ that belong to the strategy St_e.//

6. Assign $k = k + 1$ and $j = j - St_e[i-1]$.
   // This indicates the algorithm that a new column will start being processed. Now j indicates the position in the grid of the top element of *Points* corresponding to the vertex (j, k).

7. If $k \leq e - 2$, then repeat Step 2. If $k = e - 1$ go to step 3 and then finish the procedure.

4.4 CSIDH

Commutative Supersingular Isogeny Diffie Hellman (CSIDH) is an isogeny-based protocol *a la* Diffie-Hellman that can be used for key exchange and encapsulation [15], signatures [31, 35, 9], and other more advanced protocols. When we review current cryptographic protocols[49, 32], CSIDH seem to be the slowest among them, but it is compensate by the fact that is the one with smaller keys. CSIDH also support an easy key validation procedure and allows a static-static key exchange.

CSIDH as SIDH does, use supersingular curves but in contrast to SIDH, it works on $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$. As in SIDH, we make use of an special prime, this time defined by

$$p := 4\prod_{i=1}^{n} \ell_i - 1$$

Algorithm 3 Non-recursive walking across the Strategy _Stn

Require: A strategy St_e obtained from algorithm 11, Elliptic Curve E_0, Point $R \in E_0$ of order ℓ^e.

Ensure: Elliptic Curve E_e such that there is a degree-ℓ^e-isogeny between E_0 and E_e.

```
idx := 0;
i := 1;
points := [[$R$, 0]];
for row := 0 to $e-1$ do
    while idx< $n$ - row do
        $R_t := [d^S t[i]]R_t$;
        idx +:= St[i];
        Append(points, [$R_t$, idx]);
        i +:=1;
    end while
    Compute $\phi_{row}$ and $E_{row+1}$ using $E_{row}$ and $R_t$.
    Prune(points);
    for $j$ := 1 to #points do                                //PARALLEL FOR
        points[$j$, 1] := $\phi_{row}$(points[$j$, 1]);
    end for
    $[R_t, idx]$ :=Pop(points);
end for
Compute $\phi_{e-1}$ and $E_e$ using $E_{e-1}$ and $R_t$.
return $E_e$;
```

with $\ell_1, \ldots, \ell_n$ a set of small odd primes. The original proposal of CSIDH [15] works with a prime of 511 bits using 74 odd primes, where $\ell_1, \ell_2, \ldots, \ell_{73}$ are the first 73 odd primes and $\ell_74 = 587$.

CSIDH works on Montgomery curves over $\mathbb{F}_p$, *i.e.* , the A coefficient is on $\mathbb{F}_p$. The endomorphism ring of these curves are isomorphic to order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-4p})$. Castryck *et al.* [15] choose to restrict the public keys to the *horizontal isogeny class*[4] of the curve with $A = 0$, so that all endomorphism rings are isomorphic to $\mathbb{Z}[\sqrt{-p}]$.

4.4.1 The class group action

Let $E/\mathbb{F}_p$ be an elliptic curve with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$. If $\mathfrak{a}$ is a nonzero ideal in $\mathbb{Z}[\sqrt{-p}]$, then it defines a finite subgroup $E[\mathfrak{a}] = \bigcap_{\alpha\in\mathfrak{a}} \ker(\alpha)$, where we identify each α with its image in $\mathrm{End}(E)$. We then have a quotient isogeny $\phi : E \to E' = E/E[\mathfrak{a}]$ with kernel $\mathfrak{a}$; this isogeny and its codomain is well-defined up to isomorphism. If $\mathfrak{a} = (\alpha)$ is principal, then $\phi \cong \alpha$ and $E/E[\mathfrak{a}] \cong E$. Hence, we get an action of the ideal class group $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$ on the set of isomorphism classes of elliptic curves E over $\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$; this action is faithful and transitive. We write $\mathfrak{a} * E$ for the image of (the class of) E under the action of $\mathfrak{a}$, which is (the class of) $E/E[\mathfrak{a}]$ above.

For CSIDH, we are interested in computing the action of small prime ideals. Consider one of the primes ℓ_i dividing $p+1$; the principal ideal $(\ell_i) \subset \mathbb{Z}[\sqrt{-p}]$ splits into two primes, namely $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$, where π is the element of $\mathbb{Z}[\sqrt{-p}]$ mapping to the Frobenius endomorphism of the curves. Since $\bar{\mathfrak{l}}_i\mathfrak{l}_i = (\ell_i)$ is principal, we have $\bar{\mathfrak{l}}_i = \mathfrak{l}_i^{-1}$ in $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$, and hence

$$\bar{\mathfrak{l}}_i * (\mathfrak{l}_i * E) = \mathfrak{l}_i * (\bar{\mathfrak{l}}_i * E) = E$$

for all $E/\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$.

4.4.2 Class group action algorithm

The main core of the CSIDH protocol is the evaluation of an action of the class group on any supersingular curve over $\mathbb{F}_p$. To fix ideas, this is the equivalent to exponentiation on the classical Diffie-Hellman. The input of the algorithm is the A coefficient of a Montgomery curve over $\mathbb{F}_p$ and a list of exponents $(e_i, \ldots, e_n) \in \mathbb{Z}^n$. The output is the A-coefficient of the elliptic curve $\mathfrak{a} * E = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E$. Isogenies can be computed using Velu's formulae on Montgomery curves or Edwards curves(§2.4.3,2.4.4). As we mention before, in order to compute an isogeny using Velu's formulae, we need the curve an a subgroup that will be the kernel of the isogeny, then we look for points R_i for $i = 1, \ldots, n$ such that

$$R_i \begin{cases} \text{Point of order } \ell_i \text{ of the form } (x, i\cdot y) \text{ in the kernel of } \pi + 1 & \text{if } e_i < 0 \\ \text{Point of order } \ell_i \text{ in } E(\mathbb{F}_p) \text{ in the kernel of } [\ell_i] & \text{if } e_i > 0 \end{cases}$$

[4] Isogenies of non-even degree

and use $< R_i >$ as kernel for the isogenies. In the sequel we assume that we are given an algorithm `QuotientIsogeny` which, given a curve $E/\mathbb{F}_p$ $\phi : E \to E' \cong E/\langle R \rangle$, and returns the pair (ϕ, E'). We refer to this operation as *isogeny computation.* Algorithm 4, taken from the original CSIDH article [15], computes the class group action. For cryptographic purposes, the exponent vectors $(e_1, \ldots, e_n)$ must be taken from a space of size at least $2^{2\lambda}$, where λ is the (classical) security parameter. The CSIDH-512 parameters in [15] take $n = 74$, and all e_i in the interval $[-5, 5]$, so that $74 \log_2(2 \cdot 5 + 1) \simeq 255.99$, consistent with the NIST-1 security level.

Algorithm 4 The original CSIDH class group action algorithm for supersingular curves over $\mathbb{F}_p$ where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The choice of ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π is the element of $\mathbb{Q}(\sqrt{-p})$ is mapped to the p-th power Frobenius endomorphism on each curve in the isogeny class, is a system parameter. This algorithm constructs exactly $|e_i|$ isogenies for each ideal $\mathfrak{l}_i$.

```
Require: $A \in \mathbb{F}_p$ such that $E_A\colon y^2 = x^3 + Ax^2 + x$ is supersingular, and an integer
  exponent vector $(e_1, \ldots, e_n)$
Ensure: $B$ such that $E_B : y^2 = x^3 + Bx^2 + x$ is $\mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$,
  $B \leftarrow A$
  while some $e_i \neq 0$ do
    Sample a random $x \in \mathbb{F}_p$
    $s \leftarrow +1$ if $x^3 + Bx^2 + x$ is square in $\mathbb{F}_p$, else $s \leftarrow -1$
    $S \leftarrow \{i \mid e_i \neq 0, \mathrm{sign}(e_i) = s\}$
    if $S \neq \emptyset$ then
      $k \leftarrow \prod_{i \in S} \ell_i$
      $Q \leftarrow [(p+1)/k]P$, where $P$ is the projective point with $x$-coordinate $x$.
      for $i \in S$ do
        $R \leftarrow [k/\ell_i]Q$                    //Point to be used as kernel generator
        if $R \neq \infty$ then
          $(E_B, \phi) \leftarrow$ QuotientIsogeny$(E_B, R)$
          $Q \leftarrow \phi(Q)$
          $(k, e_i) \leftarrow (k/\ell_i, e_i - s)$
        end if
      end for
    end if
  end while
  return $B$
```

4.4.3 Non-Interactive Key Exchange

The setup of CSIDH is a prime of the form $p := 4\ell_1 \cdots \ell_n - 1$ where ℓ_i are small disctint odd primes and the supersingular curve $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$. For the public key generation step Alice chooses a randomly sampled n-tuple $(e_1, \ldots, e_n)$ from a range $(-m, m)$ representing the ideal class $[\mathfrak{a}] = [\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_i^{e_i}] \in \mathrm{Cl}(\mathcal{O})$. Alice compute her public key using Algorithm 4 with inputs $A = 0$ and $(e_1, \ldots, e_n)$ and gets the public key $A_{\mathfrak{a}}$ associated to $[\mathfrak{a}]E_0$. Bob do the same, in order to get his public key $A_{\mathfrak{b}}$ associated to the curve $[\mathfrak{b}]E_0$. For the key exchange step, Alice use Algorithm 4 with inputs $A = A_{\mathfrak{b}}$ and her n-tuple generated in previous step in order to compute the Montgomery coefficient of the curve $[\mathfrak{a} \cdot \mathfrak{b}]E_0$. Bob imitates Alice and get the Montgomery coefficient of the curve $[\mathfrak{b} \cdot \mathfrak{a}]E_0$. As the ideals of $\mathrm{Cl}(\mathcal{O})$ commutes then both curves are the same and the shared key is the Montgomery coefficient $A_{\mathfrak{ab}} = A_{\mathfrak{ba}}$.

Chapter 5

How to compute odd-degree isogenies

In the last few years there has been an intense interest in finding efficient formulas for computing odd degree isogenies using different models of elliptic curves. Several authors have found efficient formulas for computing isogenies using Weierstrass curves [102], Edwards, Twisted Edwards and Huff curves [69], Montgomery curves [23], and more recently, Hessian and twisted Hessian curves [30]. Nonetheless, designers of isogeny-based protocols such as SIDH[49], CSIDH[15, 66] and BSIDH[21], regularly prefer to adopt Montgomery and twisted Edwards curve models for their schemes. This is because it is widely believed that for isogeny-based protocols these two elliptic curve models provide a much more efficient curve arithmetic.

Let $q = p^n$, where p is a prime number and n a positive integer; and let ℓ be an odd number $\ell = 2s + 1$, with $s > 1$. Let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_q$ for which there exists a separable degree-ℓ isogeny $\phi : E \to E'$ defined over $\mathbb{F}_q$. This implies that there must exist an ℓ-order point $P \in E(\mathbb{F}_q)$ such that $\mathrm{Ker}(\phi) = \{\infty, \pm P, \pm[2]P, \ldots, \pm[s]P\}$. Given the domain elliptic curve E and an ℓ-order point $P \in E(\mathbb{F}_q)$, in this note we are interested in the problem of computing the co-domain elliptic curve E'. Furthermore, given a point $Q \in E(\mathbb{F}_q)$ such that $Q \notin \mathrm{Ker}(\phi)$, a closely related problem is that of finding $\phi(Q)$, *i.e.*, the image of the point Q over E'.[1]

[1] We will sometimes refer to these two problems as the isogeny construction and the isogeny evaluation computations, respectively.

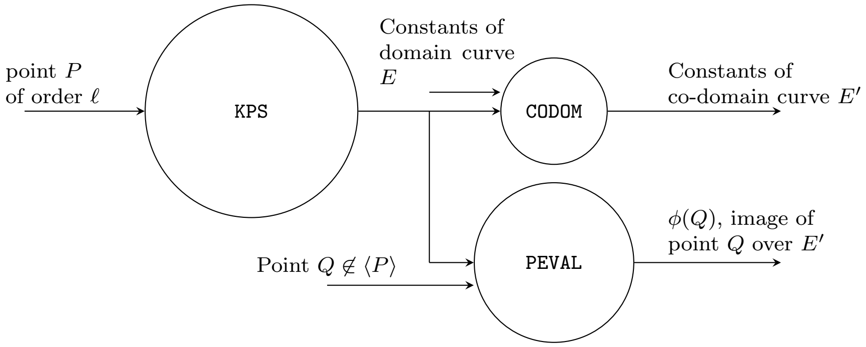


Figure 5.1: Given a supersingular elliptic curve E and an order-ℓ point $P \in E(\mathbb{F}_q)$ this diagram shows the main modules for computing a degree-ℓ isogeneous curve E' and the image of a point $Q \in E(\mathbb{F}_q)$, subject to the condition that Q is not in the kernel subgroup $\notin \langle P \rangle$. The circles are drawn to scale the relative computational costs of the modules.

In order to find efficient formulations for the above two problems and inspired in the notation used in [47, Table 1], we define `KPS` as the task of computing the first s multiples of the point P, namely, the set $\{P, [2]P, \ldots, [s]P\}$. Using `KPS` as a building block, the module `CODOM` computes the per-field constants that determine the co-domain curve E' defined over $\mathbb{F}_q$. Also, using `KPS` as a building block, `PEVAL` computes the image point $\phi(Q)$.

Figure 5.1 shows the dependencies among the `KPS` , `CODOM` and `PEVAL` primitives, where the circles are drawn to scale the relative costs of these three tasks.[2]. Both `CODOM` and `PEVAL` require the points in $\mathrm{Ker}(\phi)$ as input parameters, which are computed by the `KPS` primitive. Notice that since `CODOM` and `PEVAL` show no dependencies between them, once that the kernel points have been computed, it is possible to compute `CODOM` and `PEVAL` in parallel. Furthermore, when evaluating an arbitrary number of points in E that do not belong to the $\mathrm{Ker}(\phi)$ subgroup, `KPS` must be computed only once. Hence, the computational cost associated to `KPS` gets amortized when computing the image of two or more points.

A Montgomery curve [68] is defined by the equation $E_{A,B} : By^2 = x^3 + Ax^2 + x$, such that $B \neq 0$ and $A^2 \neq 4$. For the sake of simplicity, we will write E_A for $E_{A,1}$. Moreover, it is customary to represent the constant A in the projective space $\mathbb{P}^1$ as $(A' : C')$, such that $A = A'/C'$ (see [26]). In [5] it was shown that every Montgomery curve $E_{A,B} : By^2 = x^3+Ax^2+x$ is birationally equivalent to a twisted Edwards curve $E_{a,d} : ax^2+y^2 = 1+dx^2y^2$. The curve constants are related by

$$(A, B) = \left(\frac{2(a+d)}{a-d}, \frac{4}{a-d}\right) \quad \text{and} \quad (a, d) = \left(\frac{A+2}{B}, \frac{A-2}{B}\right).$$

[2] In fact, `KPS` becomes more expensive than `PEVAL` starting from $\ell \geq 11$ for ℓ a prime number. When $\ell \leq 7$, the block `KPS` is considerably cheaper or even free of cost for the case $\ell = 3$.

Primitive	**M**	**S**	**A**	
			Montgomery[23]	Edwards[16]
KPS	$4(s-1)$	$2(s-1)$	$6s-2$	$6s-2$
PEVAL	$4s$	2	$6s$	$2s+4$

Table 5.1: Current State-of-the-art costs for KPS and PEVAL . Field multiplication (**M**) and squaring (**S**) costs are taken from [23] and [16]. We are using the fact that KPS can be computed by performing one point doubling and $s-2$ point additions. The computational costs associated to the point addition and point doubling operations is of $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ and $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, respectively.

Table 5.1 summarizes the field arithmetic costs associated to the KPS and PEVAL operations. Note that KPS is a straightforward computation that can be performed at the cost of one point doubling and $k-2$ point additions. Efficient formulas for computing PEVAL can be found in [23] and [16] for Montgomery and twisted Edwards curves, respectively.

In the remainder of this note, different strategies for the efficient computation of the CODOM operation will be discussed. A Magma implementation of all the procedures described here along with the KPS , CODOM and PEVAL primitives, are available at, https://github.com/dcervantesv/Odd_Degree_Isogenies.

5.1 Twisted Edwards curves

In [69], Moody and Shumov presented *à la* Vélu formulas for computing isogenies on Edwards, twisted Edwards and Huff curves. Later, Meyer and Reith in [66] utilized a projective version of those formulas working on twisted Edwards YZ-coordinates. Arguably, these formulas are more efficient than the corresponding to Montgomery curves [23]. In the following, the projective version of Corollary 1 of [69] using Edwards YZ-coordinates will be assumed.

Proposition 5.1.1. Let us suppose that F is a subgroup of the twisted Edwards curve $E_{a,d}$ with odd order $\ell = 2s+1$, $s > 1$. Let the points in F be given in twisted Edwards YZ-coordinates as the set,

$$\{(Y_1 : Z_1), \dots, (Y_s : Z_s)\}.$$

Then, there exists a degree-ℓ isogeny ψ with kernel F that takes us from the curve $E_{a,d}$ to the curve $E_{a',d'}$. The constants a', d' can be computed as,

$$By = \prod_{i=1}^{s} Y_i; \qquad Bz = \prod_{i=1}^{s} Z_i; \qquad a' = a^{\ell} B_z^8; \qquad d' = B_y^8 d^{\ell}. \tag{5.1}$$

Proof. From Corollary 1 of [69], if we consider $F' = \{\infty, (\pm\alpha_1, \beta_1), \dots (\pm\alpha_s, \beta_s)\}$, then from the curve $E_{a,d}$ to the curve $E_{\bar{a},\bar{d}}$ there exists a degree-ℓ isogeny ψ' with kernel F' that can be computed as,

$$B = \prod_{i=1}^{s} \beta_i; \quad \bar{a} = a^{\ell}; \qquad \bar{d} = B^8 d^{\ell}. \tag{5.2}$$

Using $\beta_i = Y_i/Z_i$ and plugging in into Eq. (5.2) yields,

$$B = \prod_{i=1}^{s} \frac{Y_i}{Z_i} = \frac{\prod_{i=1}^{s} Y_i}{\prod_{i=1}^{s} Z_i} = \frac{B_y}{B_z}; \qquad \bar{a} = a^{\ell}; \qquad \bar{d} = (\frac{B_y}{B_z})^8 d^{\ell}.$$

It is known from [5] that $E_{a,d} \cong E_{1,d/a}$. This implies that

$$E_{\bar{a},\bar{d}} \cong E_{1,\frac{(\frac{B_y}{B_z})^8 d^{\ell}}{a^{\ell}}} \cong E_{1,\frac{B_y^8 d^{\ell}}{B_z^8 a^{\ell}}} \cong E_{a',d'}.$$

□

The computational costs associated to Eq. (5.1) can be upper bounded by assuming that the exponentiations to the power ℓ are performed independently and by means of the binary method. Let $\mathrm{H}(\ell)$ and $\lambda(\ell)$ denote the Hamming weight and the bit-length of the integer ℓ, respectively. Hence, the cost of computing the co-domain curve is given as follows. The values B_y^8 and B_z^8 can be computed at a cost of $2(s-1)\mathbf{M} + 6\mathbf{S}$. When using the binary method the overall cost of the exponentiations a^{ℓ} and d^{ℓ} is of $2(\mathrm{H}(\ell) - 1)\mathbf{M} + 2(\lambda(\ell)-1)\mathbf{S}$. Two more multiplications are required to obtain a' and d'. Therefore, the cost of computing the constants a' and d' of Eq. (5.1), which define the co-domain isogenous curve is upper bounded by

$$2(s + \mathrm{H}(\ell) - 1)\mathbf{M} \; + 2(\lambda(\ell) + 2)\mathbf{S} \; .$$

In the remaining of these note, several methods for improving the above upper bound will be discussed.

5.1.1 Using NAF for reducing the computational cost of odd degree isogenies

By exploiting once again the property $E_{a,d} \cong E_{1,d/a}$, one can in general reduce the computation of the exponentiations of Eq. (5.1) using any signed representation of the exponents such as the well-known Non-adjacent Form (NAF). Let us recall that the NAF representation of a positive integer ℓ is an expression $\ell = \sum_{i=0}^{n-1} \ell_i 2^i$, where $\ell_i \in \{0, \pm 1\}$, $\ell_{n-1} \neq 0$ and no two consecutive digits ℓ_i are nonzero [44]. Let $L = NAF(\ell)$. In the case of Eq. (5.5), one can notice that the positive and negative values of the NAF representation of ℓ can be split as,

$$L = L_p - L_m, \quad \text{where} \quad L_p = \sum_{i=0|k_i>0}^{n-1} \ell_i 2^i, \quad \text{and} \quad L_m = -\sum_{i=0|k_i<0}^{n-1} \ell_i 2^i. \tag{5.3}$$

As an illustrative example, let us consider the representations of the integer 353,

$$\begin{aligned} (353)_2 &= (1,0,1,1,0,0,0,0,1); \\ L = NAF(353) &= (1,0,\bar{1},0,\bar{1},0,0,0,0,1); \\ (L_p)_2 &= (1,0,0,0,0,0,0,0,0,1); \\ (L_m)_2 &= (0,0,1,0,1,0,0,0,0,0). \end{aligned}$$

Using the above split NAF representation, one can compute n^{ℓ} as $n^{L_p} n^{-L_m}$. Moreover, exploiting $E_{a,d} \cong E_{1,d/a}$, one can compute $a' = a^{\ell} B_z^8$ and $d' = B_y^8 d^{\ell}$ as,

$$a' = B_z^8 a^{L_p} d^{L_m}, \qquad d' = B_y^8 a^{L_m} d^{L_p}. \tag{5.4}$$

Notice that the two exponentiations $a^{L_p} d^{L_m}$ and $a^{L_m} d^{L_p}$ of Eq. 5.4 can be computed *simultaneously* (cf. [44, §3.3.3.]), by using a right-to-left exponentiation approach that allows us to share the squaring operations. Moreover, since the *positive bit* exponentiations a^{L_p} and d^{L_p} of Eq. 5.4 share the same exponent, they are naturally synchronized. The same can be said about the *negative bit* exponentiations d^{L_m} and a^{L_m}.

We carefully exploit the dependencies on these four exponentiations as shown in Algorithm 5. The accumulators for the exponentiations (a^{L_p}, a^{L_m}) and (d^{L_p}, d^{L_m}), are stored into the two-entry arrays T_a and T_d, respectively. Notice that in lines 6-7, these arrays are initialized to one. As we are dealing with odd ℓ, this implies that the least significant bit of $NAF(\ell)$ is always non-zero. This observation is used to save two multiplications as shown in line 9. Depending on the sign of $L[0]$, in line 9 only the entries that will store the positive (or the negative) bit exponentiations get initialized with a and d, respectively. The other entries are only initialized when a sign change is detected. To simplify the algorithm, this change of sign can be pre-computed off-line, by recording the exact position ω where this change occurs. The two input parameters a, d are rewritten to accumulate the squaring operations by updating them at each iteration of the two main loops in lines 10 and 21. They are also updated in line 18 when a sign change has been detected. It can be shown that the cost of computing a' and d' using Algorithm 5 is given as,

$$2(s + \mathrm{H}(L))\mathbf{M} \ + 2(\#L + 2)\mathbf{S} \ ,$$

where $L = NAF(\ell)$. The above cost is often cheaper than the one presented in the previous section for a binary representation. This is due to the fact that the average non-zero density 1/3 of the NAF representation, is cheaper than the average non-zero density 1/2 of the binary representation.

If the NAF expansion of ℓ contains no $\bar{1}$, then $NAF(\ell) = (\ell)_2$ and a simple binary exponentiation suffices. In this case the computational expense of Algorithm 5 becomes $2(s+\mathrm{H}(\ell)-1)\mathbf{M} + 2(\lambda(\ell)+2)\mathbf{S}$. Therefore, it can be concluded that computing an isogeny using Algorithm 5 adds no extra costs, but savings due to the fact that the NAF recoding guarantees a smaller non-zero density in comparison to the binary method.[3]

As a relevant practical example, consider all the prime factors in the factorization of $(p_{512}+1)/4$ where p_{512} is the prime used in the CSIDH-512 protocol [15]. The NAF trick discussed here will have the same computational cost as the one associated to the binary representation for those primes in the set $\{5, 17, 37, 41, 73, 137, 149, 257, 277, 293, 337\}$. For all the other 63 prime factors of $(p_{512}+1)/4$, the NAF representation produces savings compared with the cost of the binary one. The Magma code of Algorithm 5 is available at, https://github.com/dcervantesv/Odd_Degree_Isogenies.

Remark 12. When Algorithm 5 deals with the case $\ell_2 = NAF(\ell)$, the variable w_L must be set to any position in the vector L with value zero. This way, the assignment on lines 19-20 does not interfere with the positive accumulator as the sign computed in Line 17 is the opposite of the first sign computed (that must be positive as $\ell_2 = NAF(L)$). On Line 29 if $L[\omega_L] = 0$, then a' and d' are computed as in Equation 5.1. Otherwise, a' and d' are computed as in Equation 5.4.

Remark 13. We are not aware of any isogeny-based protocol where the input parameter ℓ of Algorithm 5 must remain secret. Hence, no efforts to protect this procedure against timing-attacks were attempted.

5.1.2 Using modular arithmetic for reducing the computational cost of odd degree isogenies

As previously discussed, the computation of Eq. (5.1) requires two exponentiations to the power ℓ. These computational expenses can be reduced as follows.

Corollary 5.1.1.1. Let $E_{a,d}$, F and ℓ be given as in Corollary 5.1.1. Let us define $k = \lfloor \ell/8 \rfloor$, and $r = \ell \mod 8$. Then, from the curve $E_{a,d}$ to the curve $E_{a',d'}$ there exists a degree-ℓ isogeny ψ with kernel F, where the constants a', d' can be computed as

$$a' = (a^k B_z)^8 a^r; \qquad d' = (d^k B_y)^8 d^r; \qquad , By = \prod_{i=1}^{s} Y_i; \quad \text{and} \quad Bz = \prod_{i=1}^{s} Z_i. \tag{5.5}$$

Proof. It follows immediately from Proposition 5.1.1 and the property $E_{a,d} \cong E_{1,d/a}$. □

At first glance it may appear that compared with Eq. 5.1, the computational cost of Eq. (5.5) has been increased by the addition of two extra multiplications and two exponentiations by r. Nevertheless, we will argue in the following that the new formulation of Eq. (5.5) can save up to 6**S** compared with the costs associated to Eq. 5.1. To this end, let us first analyze the computation of a^r and d^r. Since we are dealing with odd degree isogenies, there are only four possible remainders modulo 8, namely, $r = 1, 3, 5, 7$. The

[3] It may incur though, to one extra squaring due to the extra bit associated to NAF [44]

Algorithm 5 Odd degree isogenous codomain curve computation using NAF recoding.

Require: Integer $\ell = 2s + 1$, Integer vector $L = NAF(\ell)$, Integer $\omega_L \in [0..(\#L - 1)]$ in which the first change of sign occurs. Edwards curve constants a and d, Subgroup $F = (Y_1 : Z_1), \ldots, (Y_s : Z_s)$.

Ensure: a' and d' defining the Twisted Edwards curve $E_{a',d'}$ isogenous to $E_{a,d}$ as in Corollary 5.1.1.

```
$B_y \leftarrow Y_1$;
$B_z \leftarrow Z_1$;
for i=2 to s do
    $B_y \leftarrow B_y Y_i$; $B_z \leftarrow B_z Z_i$;
end for
$T_a \leftarrow [1, 1]$                    //Initialization of $T_a$ which will hold $a_p$ and $a_m$
$T_d \leftarrow [1, 1]$                    //Initialization of $T_d$ which will hold $d_p$ and $d_m$
$Sign \leftarrow \frac{L[0]+1}{2}$;
$T_a[Sign] = a$ ; $T_d[Sign] = d$;
for $i := 1$ to $(\omega_L - 1)$ do
    $a \leftarrow a^2$; $d \leftarrow d^2$;
    if $L[i] \neq 0$ then
        $T_a[Sign] \leftarrow T_a[Sign] \cdot a$;
        $T_d[Sign] \leftarrow T_d[Sign] \cdot d$;
    end if
end for
$Sign \leftarrow (Sign + 1) \mod 2$;      //The opposite of the Sign of $L[0]$
$a \leftarrow a^2$; $d \leftarrow d^2$;
$T_a[Sign] \leftarrow a$;
$T_d[Sign] \leftarrow d$;
for $i := (\omega_L + 1)$ to $(\#(L) - 1)$ do
    $a \leftarrow a^2$; $d \leftarrow d^2$;
    if $L[i] \neq 0$ then
        $Sign \leftarrow \frac{L[i]+1}{2}$
        $T_a[Sign] \leftarrow T_a[Sign] \cdot a$;
        $T_d[Sign] \leftarrow T_d[Sign] \cdot d$;
    end if
end for
if $L[w_L] = 0$ then
    $a' \leftarrow B_z^8 \cdot T_a[1]$.
    $d' \leftarrow B_y^8 \cdot T_d[1]$.
else
    $a' \leftarrow B_z^8 \cdot T_a[1] \cdot T_d[0]$.
    $d' \leftarrow B_y^8 \cdot T_a[0] \cdot T_d[1]$.
end if
return $a', d'$.
```

cheapest case occurs when $r = 1$, because we do not incur in any additional multiplication and we trade two exponentiations to the power ℓ in Eq. (5.1), by two exponentiations to the power $k = \lfloor \frac{l}{8} \rfloor$ in Eq. (5.5). Further, one can compute the other three possible reminders at a cost of only two extra multiplications as shown in Eq. 5.6.

$$[a', d'] = \begin{cases} [((a^k B_z)^4 a)^2 a, & ((d^k B_y)^4 d)^2 d] & r = 3 \\ [((a^k B_z)^2 a)^4 a, & ((d^k B_y)^2 d)^4 d] & r = 5 \\ [(a^k B_z a)^8 d, & (d^k B_y d)^8 a] & r = 7 \end{cases} \tag{5.6}$$

Then, the cost of computing a' and d' as in Corollary 5.1.1.1 and assuming $k > 0$ is given as

$$\begin{cases} 2(\mathrm{H}(k) + s)\mathbf{M} \ + 2(\lambda(k) + 2)\mathbf{S} & \text{if } r = 1 \\ 2(\mathrm{H}(k) + s + 1)\mathbf{M} \ + 2(\lambda(k) + 2)\mathbf{S} & \text{otherwise} \end{cases}. \tag{5.7}$$

We dub this method as the `div8` approach.

Remark 14. Equation 5.7 holds for $k > 0$. When $k = 0$ one can save one extra multiplication. Therefore, the specialized cost for a degree-3, -5 and -7 isogeny computation using the `div8` approach is of $4\mathbf{M} \ + 6\mathbf{S}$, $6\mathbf{M} \ + 6\mathbf{S}$ and $8\mathbf{M} \ + 6\mathbf{S}$, respectively.

5.1.3 Combining NAF and modular reduction

In §5.1.1, it was observed that the NAF representation applied to the exponent ℓ is advantageous for constructing odd degree isogenies. In §5.1.2 an efficient isogeny construction by considering the exponent ℓ divided by eight and its respective remainder was described. The next natural step is to apply the NAF representation to the exponent k along the lines of the `div8` approach.

Corollary 5.1.1.2. Let $E_{a,d}$, F and $\ell = 2d + 1$ as in Corollary 5.1.1. Compute $K = NAF(\lfloor l/8 \rfloor)$ and consider

$$K_p = \sum_{i=0|K_i>0}^{n-1} k_i 2^i, \quad \text{and} \quad K_m = -\sum_{i=0|k_i<0}^{n-1} k_i 2^i. \tag{5.8}$$

Then, from the curve $E_{a,d}$ to the curve $E_{a',d'}$ there exists a degree-ℓ isogeny ψ with kernel F where

$$a' = (a^{K_p} d^{K_m} B_z)^8 a^r; \quad d' = (a^{K_m} d^{K_p} B_y)^8 d^r;$$

$$By = \prod_{i=1}^{s} Y_i; \quad \text{and} \quad Bz = \prod_{i=1}^{s} Z_i.$$

The proof is the same as in Corollary 5.1.1.1 along with a straightforward computation. The cost of computing this new rearrange is given by

$$\begin{cases} 2(\mathrm{H}(K) + s)\mathbf{M} \ + 2(\#K + 2)\mathbf{S} & \text{if } r = 1 \\ 2(\mathrm{H}(K) + s + 1)\mathbf{M} \ + 2(\#K + 2)\mathbf{S} & \text{otherwise} \end{cases} \tag{5.9}$$

The computational cost of the Corollary 5.1.1.2 given in Eq. (5.9) can be justified as follows,

- Again, in order to compute B_y and B_z one requires to perform $2(s-1)\mathbf{M}$.
- In this case, it is unknown whether the first bit of K is zero or not, but one can pre-compute off-line, how many zeros happen to be before the first non-zero element. Then after initializing the accumulator, one can follow a similar strategy as in Algorithm 5 to compute $a^{K_p}, a^{K_m}, d^{K_p}$ and d^{K_m}. This will help us to save two multiplications. The computational cost of this step is $2(\mathrm{H}(K)-2)\mathbf{M} + 2(\#K-1)\mathbf{S}$.
- Once we have $B_z, B_y, a^r, d^r, a^{K_p}, a^{K_m}, d^{K_p}$ and d^{K_m}, the cost of computing a' and d' is $6\mathbf{M} + 6\mathbf{S}$.
- Using the result of Eq.(5.6), it can be seen that two extra multiplications are required to compute a^r and d^r for $r \in \{3,5,7\}$, and no extra cost for the case $r = 1$.

The addition of the above computational expenses gives the result presented in Eq. (5.9).

5.2 Comparison

For the sake of fairness, in this section the isogeny construction methods that have been proposed by different authors using their cheapest version for as many odd values ℓ as possible, are reported. Table 5.2 shows the operation counts for several state-of-the-art isogeny construction algorithms using different elliptic curve models. Note that the `a_from_alpha` approach proposed in [23], computes an isogeny construction using the image of a two-torsion point in a Montgomery curve different than the point $(0,0)$. This approach can only be used when considering rational points over an extension of the base field $\mathbb{F}_p$. Algorithm `3_point_recover`[23] requires to know in advance the x-coordinates of the points $x(P), x(P)$ and $x(P-Q)$ for some P and Q belonging to the co-domain curve. It appears that this approach cannot be easily extended to a general scenario other than the key generation phase of the SIDH protocol, where primes of the form $p = \ell_a^{e_a}\ell_b^{e_b}f - 1$ with $\ell_i < 5$ are used. The rest of the algorithms reported in Table 5.2 can easily be applied to more generic settings.

We did not consider hybrid cases where curve constants must be translated into another model of curve or to a better representation of the constants. For example, Onuki and Takagi algorithm [82] returns the constant $(A : C)$. The cost of computing the constants A_{24} and C_{24} required for fast Montgomery curve-arithmetic is not taken into account here. Also for the Edwards curves, the computation of the constant $e = a - d$ used in [16] to attain faster curve arithmetic is disregarded. Table 5.3 summarizes the operation counts of optimized formulas for certain specific isogeny degrees. Among all state-of-the-art algorithms, only [23] reports one specialized algorithm for a specific isogeny degree. In Remark 14 of this note, concrete isogeny computation formulas for three specific degrees are given.

Work	Model	Cost		
		M	**S**	**A**
`a_from_alpha` [23]	Mont	$4s$	4	$4s+3$
`3_point_recover` [23]	Mont	8	5	11
CSIDH [15]	Mont	$6s-2$	3	4
Meyer-Reith [66]	Ed	$2(s+\mathrm{H}(\ell)$	$2(\lambda(\ell)+2)$	0
Onuki-Takagi [82]	Mont	$5s-1$	2	$s+5$
`NAF_exp` (Algorithm 5)	Ed	$2(s+\mathrm{H}(L)-1)$	$2(\#L+2)$	0
`div8` $r=1$ (§5.1.2)	Ed	$2(\mathrm{H}(k)+s)$	$2(\lambda(k)+2)$	0
`div8` $r=3,5,7$ (§5.1.2)	Ed	$2(\mathrm{H}(k)+s+1)$	$2(\lambda(k)+2)$	0
`div8NAF` $r=1$ (§5.1.3)	Ed	$2(\mathrm{H}(K)+s)$	$2(\#K+2)$	0
`div8NAF` $r=3,5,7$ (§5.1.3)	Ed	$2(\mathrm{H}(K)+s+1)$	$2(\#K+2)$	0

Table 5.2: General costs for different state-of-the-art isogeny construction algorithms (dubbed `CODOM` in Fig 5.1). Notice that the algorithms from [23] require extra input data to compute the co-domain curve, and might not be useful on the CSIDH framework. Here $\ell = 2s+1$, $k = \lfloor \ell/8 \rfloor$, $r = \ell \mod 8$, $L = NAF(\ell)$ and $K = NAF(k)$.

Work	Degree	Cost		
		M	**S**	**A**
Costello-Hisil [23]	3	2	3	14
This work (Remark 14)	3	4	6	0
This work (Remark 14)	5	6	6	0
This work (Remark 14)	7	8	6	0

Table 5.3: Specialized formulas for certain odd degree isogenies.

degree	Costello-Hisil [23]	Onuki-Takagi [82]	Meyer-Reith [66]	`div8`
3	5.1(*)	5.9	10.4	8.8(*)
5	11.75	10.95	14	10.8(*)
7	15.95	16	18	12.8(*)
9	20.15	21.05	19.6	14.8
11	24.35	26.1	23.6	18.8

Table 5.4: Costs assuming $\mathbf{S} = 0.8\mathbf{M}$ and $\mathbf{A} = 0.05\mathbf{M}$. All costs are given Using **M** as unit of measure. Costs with(*) indicate that we are using the operation counts reported in Table 5.3. The shaded cells indicate the minimum cost for each degree.

degree	Costello-Hisil[23]	Onuki-Takagi[82]	Meyer-Reith [66]	`div8`
3	5.7(*)	6.300	12	10(*)
5	12.55	11.35	16	12(*)
7	16.75	16.40	20	14(*)
9	20.95	21.45	22	16
11	25.15	26.50	26	20

Table 5.5: Costs assuming $\mathbf{S} = \mathbf{M}$ and $\mathbf{A} = 0.05\mathbf{M}$. All costs are given Using **M** as unit of measure. Costs with (*) indicate that we are using the operation counts reported in Table 5.3. The shaded cells indicate the minimum cost for each degree.

Tables 5.4 and 5.5 report the costs of computing the co-domain curve using **M** as cost metric. In Table 5.4 it is assumed that **S** =0.8**M** as in [82]. For this setting, it is observed that the `div8` approach introduced in §5.1.2 outperforms the other strategies for all the degrees except when computing isogenies of degree 3, where the specialized formula of Costello and Hisil [23] is optimal. Arguably, the assumption of the ratio **S** =0.8**M** for the field arithmetic is a bit unrealistic.[4] Hence in Table 5.5 a more realistic scenario for $\mathbb{F}_p$ arithmetic is considered by assuming **S** =**M** . In this setting the algorithm by Onuki and Takagi [82] emerges as the optimal method when constructing degree-5 isogenies. In the case of degree-3 isogeny constructions, the approach by Costello and Hisil [23] remains unbeatable.

We executed our Magma scripts for determining the associated computational costs for `div8` , `NAFexp` and `div8NAF` . We also report the computational costs for Onuki and Takagi [82] and Meyer-Reith [65] analyzed trough all the prime numbers in the interval $[11, 2^{20}]$. The corresponding computational costs are summarized in Tables 5.6 and 5.7.

The large interval considered in Tables 5.6 and 5.7 can be of interest for the search of more conservative parameters for the CSIDH protocol. Moreover, BSIDH[21] uses ℓ-isogenies where the biggest bit-length of ℓ is of about 22 bits. Furthermore, recently the SÉTA protocol, a new isogeny-based protocol, was introduced in [34]. The SÉTA protocol seems to make use of ℓ-isogenies where the largest ℓ is about 2^{14}.

Remark 15. We only consider odd prime degree isogenies due to the following observation.

[4] We still consider $\mathbf{S} = 0.8\mathbf{M}$ because this might be about the correct ratio for the $\mathbb{F}_{p^2}$ quadratic field arithmetic.

Vs.	Meyer-Reith [66]	Onuki-Takagi [82]	NAFexp	div8	div8NAF
Meyer-Reith [66]	-	100	6.163	0	0
Onuki-Takagi [82]	0	-	0.001	0	0
NAFexp	77.649	99.998	-	24.004	0
div8	100	100	62.254	-	9.408
div8NAF	100	100	100	67.239	-

Table 5.6: Comparison of different algorithms to compute CODOM , assuming $\mathbf{S} = \mathbf{M}$ and $\ell_2 \neq NAF(\ell)$ for a prime $\ell \in [11, 2^{20}]$. All numbers report the winning percentage when comparing the algorithm in row i versus the algorithm in column j. Since for some degrees there are ties, it may occur that the addition of the cells $(i,j)+(j,i)$ could be smaller than 100.

Vs.	Meyer-Reith [66]	Onuki-Takagi [82]	NAFexp	div8	div8NAF
Meyer-Reith [66]	-	100	6.163	0	0
Onuki-Takagi [82]	0	-	0	0	0
NAFexp	87.379	100	-	37.745	0
div8	100	100	62.254	-	9.408
div8NAF	100	100	100	79.884	-

Table 5.7: Comparison of different algorithms to compute CODOM assuming $\mathbf{S} = 0.8\mathbf{M}$ and $\ell_2 \neq NAF(\ell)$ for a prime $\ell \in [11, 2^{20}]$. All numbers report the winning percentage when compare the algorithm in row i versus the algorithm in column j. Since for some degrees there are ties, it may occur that the addition of the cells $(i,j)+(j,i)$ could be smaller than 100.

Let us assume that one wants to compute a composite odd degree-ℓ isogeny with $\ell = \prod_{i=1}^{r} \ell_i$, ℓ_i being distinct prime numbers and $r > 1$ a positive integer. Such isogeny can be computed as the composition of the r degree-ℓ_i isogenies. It is not hard to see that using that composition approach, the associated computational complexity is linear[5] with respect to $\sum_{i=1}^{r} \ell_i$. If one would try to compute the ℓ-degree isogeny by directly applying the formulas presented in this document, the computational complexity is also linear but this time with respect to $\prod_{i=1}^{r} \ell_i$. The latter is a much larger number than the one associated to the composition approach.

5.3 Conclusion

The cost of constructing odd-degree isogenies on Edwards curves is closely related to the problem of the simultaneous computation of several exponentiations over $\mathbb{F}_q$. In this note we review three strategies to compute such exponentiations. Our results show that for most

[5] And possibly linear-logarithmic with respect to $\prod_{i=1}^{r} \ell_i$, if we employ a multiplicative strategy to compute such composition (as in the CSIDH protocol).

prime numbers in the interval $[11, 2^{20}]$ the `div8NAF` approach described in this note appears to be the best option for computing odd degree isogenies on Edwards curves. On the other hand, if we focus on the prime numbers that appear in the factorization of the prime $p_{512}+1$ used in [15], it seems that the `div8` approach is the best (See Appendix A). Moreover, for the relevant case of isogenies of degree 3, the best algorithm is the one provided by Costello and Hissil in [23]. The best approach for constructing degree-5 isogenies is contested between our `div8` approach and the Onuki and Takagi [82] algorithm. The former strategy is better when it is assumed that $\mathbf{S}$ =0.8$\mathbf{M}$, but the latter is cheaper when $\mathbf{S}$ =$\mathbf{M}$.

The main contribution of this note is the introduction of the NAF exponentiation into the Edwards co-domain curve computation (§5.1.1) and the `div8` (§5.1.2) and `div8NAF` (§5.1.3) approaches. As a future work we will explore the use of wNAF recodings into the construction of large odd degree isogenies and the use of parallel computation of the three building blocks depicted in Fig. 5.1.

Chapter 6

On the Cost of Computing Isogenies Between Supersingular Elliptic Curves

> The Road goes ever on and on Down from the door where it began. Now far ahead the Road has gone, And I must follow, if I can, Pursuing it with eager feet, Until it joins some larger way Where many paths and errands meet. And whither then? I cannot say
>
> J. R. R. Tolkien, *The Fellowship of the Ring bk. 1, ch. 1 (1954)*

6.1 Introduction

The Supersingular Isogeny Diffie-Hellman (SIDH) key agreement scheme was proposed by Jao and De Feo [49] (see also [32]). It is one of 69 candidates being considered by the U.S. government's National Institute of Standards and Technology (NIST) for inclusion in a forthcoming standard for quantum-safe cryptography [3]. The security of SIDH is based on the difficulty of the Computational Supersingular Isogeny (CSSI) problem, which was first defined by Charles, Goren and Lauter [19] in their paper that introduced an isogeny-based hash function. The CSSI problem is also the basis for the security of isogeny-based signature schemes [40, 105] and an undeniable signature scheme [51].

Let p be a prime, let ℓ be a small prime (e.g., $\ell \in \{2, 3\}$), and let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ for which a (separable) degree-ℓ^e isogeny $\phi : E \to E'$ defined over $\mathbb{F}_{p^2}$ exists. The CSSI problem is that of constructing such an isogeny. In [32], the CSSI problem is assessed as having a complexity of $O(p^{1/4})$ and $O(p^{1/6})$ against classical and quantum attacks [94], respectively. The classical attack is a meet-in-the-middle attack (MITM) that has time complexity $O(p^{1/4})$ and space complexity $O(p^{1/4})$. We observe that the (classical) van Oorschot-Wiener golden collision finding algorithm [101, 100] can be employed to construct ϕ. Whereas the time complexity of the van Oorschot-Wiener algorithm is higher than that of the meet-in-the-middle attack, its space requirements are smaller. Our cost analysis of these two CSSI attacks leads to the conclusion that, despite its higher running time, the golden collision finding CSSI attack has a lower cost than the meet-in-the-middle attack, and thus should be used to assess the security of SIDH against (known) classical attacks.

The remainder of this Chapter is organized as follows. The CSSI problem and relevant mathematics background are presented in §6.2. In §6.3 and §6.4, we report on our implementation of the meet-in-the-middle and golden collision search methods for solving CSSI. Our implementations confirm that the heuristic analysis of these CSSI attacks accurately predicts their performance in practice. Our cost models and cost comparisons are presented in §6.5. Finally, in §6.6 we make some concluding remarks.

6.2 Computational Supersingular Isogeny Problem

6.2.1 CSSI

The challenge faced by a passive adversary is to compute the kernel of Alice (or Bob) secret isogeny given the public parameters, E/A, E/B, $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$ and $\phi_B(Q_A)$. A necessary condition for hardness of this problem is the intractability of the Computational Supersingular Isogeny (CSSI) problem: Given the public parameters ℓ_A, ℓ_B, e_A, e_B, p, E, P_A, Q_A, P_B, Q_B, the elliptic curve E/A, and the auxiliary points $\phi_A(P_B)$ and $\phi_A(Q_B)$, compute the Vélu isogeny $\phi_A : E \to E/A$ (or, equivalently, determine a generator of A).

An assumption one makes (e.g., see [32]) is that the auxiliary points $\phi_A(P_B)$ and $\phi_A(Q_B)$ are of no use in solving CSSI. Thus, we can simplify the statement of the CSSI problem to the following:

Problem 1 (CSSI)**.** Given the public parameters ℓ_A, ℓ_B, e_A, e_B, p, E, P_A, Q_A, and the elliptic curve E/A, compute a degree-$\ell_A^{e_A}$ isogeny $\phi_A : E \to E/A$.

6.3 Meet-in-the-Middle

For the sake of simplicity, we will suppose that e is even. We denote the number of order-$\ell^{e/2}$ subgroups of $E[\ell^e]$ by $N = (\ell+1)\ell^{e/2-1} \approx p^{1/4}$.

Let $E_1 = E$ and $E_2 = E/A$. Let R denote the set of all j-invariants of elliptic curves that are isogenous to E_1; then $\#R \approx p/12$ [87]. Let R_1 denote the set of all j-invariants of elliptic curves over $\mathbb{F}_{p^2}$ that are $\ell^{e/2}$-isogenous to E_1. Since $\#R \gg N$, one expects that the number of pairs of distinct order-$\ell^{e/2}$ subgroups (A_1, A_2) of $E_1[\ell^e]$ with $j(E_1/A_1) = j(E_1/A_2)$ is very small. Thus, we shall assume for the sake of simplicity that $\#R_1 = N$. Similarly, we let R_2 denote the set of all j-invariants of elliptic curves that are $\ell^{e/2}$-isogenous to E_2, and assume that $\#R_2 = N$. Since E_1 is ℓ^e-isogenous to E_2, we know that $R_1 \cap R_2 \neq \emptyset$. Moreover, since $\#R_1 \ll \#R$ and $\#R_2 \ll \#R$, it is reasonable to assume that $\#(R_1 \cap R_2) = 1$; in other words, we can assume that there is a unique degree-ℓ^e isogeny $\phi : E_1 \to E_2$.

6.3.1 Basic method

The meet-in-the-middle attack on CSSI [32], which we denote by MITM-basic, proceeds by building a (sorted) table with entries $(j(E_1/A_1), A_1)$, where A_1 ranges over all order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $E_2[\ell^e]$, one computes E_2/A_2 and searches for $j(E_2/A_2)$ in the table (see Figure 6.1). If $j(E_2/A_2) = j(E_1/A_1)$, then the composition of isogenies

$$\phi_{A_1} : E_1 \to E_1/A_1, \quad \psi : E_1/A_1 \to E_2/A_2, \quad \hat{\phi}_{A_2} : E_2/A_2 \to E_2,$$

where ψ is an $\mathbb{F}_{p^2}$-isomorphism and $\hat{\phi}_{A_2}$ denotes the dual of ϕ_{A_2}, is the desired degree-ℓ^e isogeny from E_1 to E_2. The worst-case time complexity of MITM-basic is $T_1 = 2N$, where a unit of time is a degree-$\ell^{e/2}$ Vélu isogeny computation (cf. Remark 11). The average-case time complexity is $1.5N$. The attack has space complexity N.

6.3.2 Depth-first search

The set of pairs $(j(E/A), A)$, with A ranging over all order-$\ell^{e/2}$ subgroups of $E[\ell^e]$, can also be generated by using a depth-first search (DFS) to traverse the tree in the left of Figure 6.1 (and also the tree in the right of Figure 6.1). We denote this variant of the meet-in-the-middle attack by MITM-DFS. We describe the depth-first search for $\ell = 2$.[1]

Let $\{P, Q\}$ be a basis for $E[2^{e/2}]$. Let $R_0 = 2^{e/2-1}P$, $R_1 = 2^{e/2-1}Q$, $R_2 = R_0 + R_1$ be the order-2 points on E. For $i = 0, 1, 2$, the degree-2 isogenies $\phi_i : E \to E_i = E/\langle R_i \rangle$ are computed, as are bases $\{P_0 = \phi_0(P), Q_0 = \phi_0(2Q)\}$, $\{P_1 = \phi_1(Q), Q_1 = \phi_1(2P)\}$, $\{P_2 = \phi_2(P+Q), Q_2 = \phi_2(2P)\}$ for $E_0[2^{e/2-1}]$, $E_1[2^{e/2-1}]$, $E_2[2^{e/2-1}]$, respectively. A

[1] For the sake of concreteness, all implementation reports of CSSI attacks in this document are for the case $\ell = 2$. However, all conclusions about the relative efficiencies of classical and quantum CSSI attacks for $\ell = 2$ are also valid for the $\ell = 3$ case.

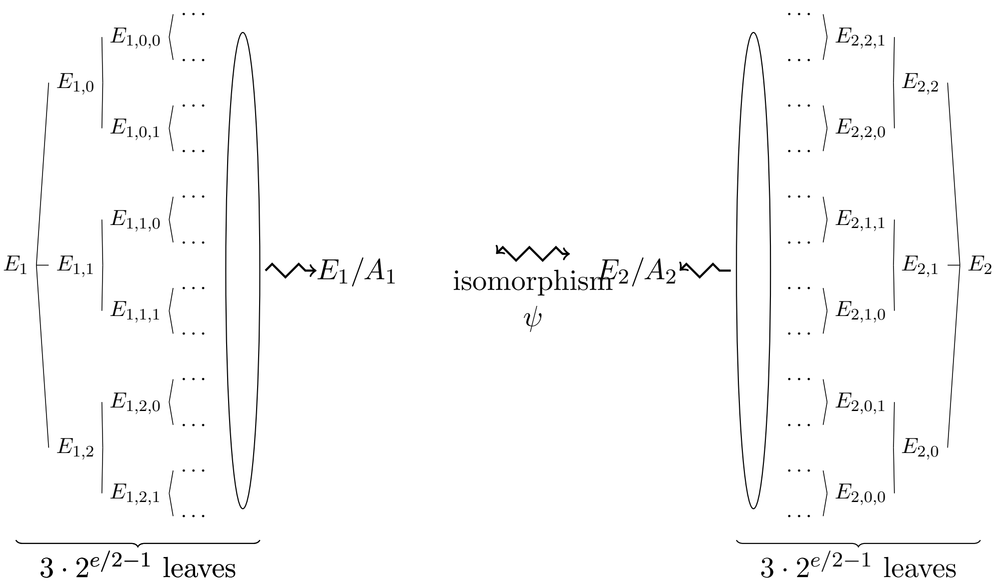


Figure 6.1: Meet-in-the-middle attack for degree-2 isogeny trees.

memory stack is initialized with the tuples $(E_0, 0, P_0, Q_0)$, $(E_1, 1, P_1, Q_1)$, $(E_2, 2, P_2, Q_2)$, and the tuple on the top of the stack is processed recursively as described next.

Suppose that we have to process (E_x, x, P_x, Q_x), where $x \in \{0,1,2\} \times \{0,1\}^{n-1}$ and $1 \leq n \leq e/2-1$. Let $B_0 = 2^{e/2-n-1}P_x$, $B_1 = 2^{e/2-n-1}Q_x$ and $B_2 = B_0 + B_1$ be the order-2 points on E_x. Let $R_{x0} = B_0$ and $R_{x1} = B_2$ (B_1 is the backtracking point), and compute the degree-2 isogenies $\phi_{xi} : E_x \to E_{xi} = E_x/\langle R_{xi} \rangle$ for $i = 0, 1$. Then two cases arise:

(i) If $n < e/2 - 1$, then let $P_{x0} = \phi_{x0}(P_x)$, $Q_{x0} = \phi_{x0}(2(P_x + Q_x))$, $P_{x1} = \phi_{x1}(P_x + Q_x)$, $Q_{x1} = \phi_{x1}(2P_x)$; one can check that $\{P_{xi}, Q_{xi}\}$ is a basis for $E_{xi}[2^{e/2-n-1}]$ for $i = 0, 1$. Then, $(E_{x1}, x1, P_{x1}, Q_{x1})$ is added to the stack and $(E_{x0}, x0, P_{x0}, Q_{x0})$ is processed next.

(ii) If $n = e/2 - 1$, the leaves $(j(E_{x0}), x0)$ and $(j(E_{x1}), x1)$ of the tree are stored in the table. If the stack is non-empty, then its topmost entry is processed next; otherwise the computation terminates.

The cost of building each of the two depth-first search trees is approximately $2N$ degree-2 isogeny computations, $2N$ degree-2 isogeny evaluations, $N/2$ point additions, and $2N$ point doublings (where $N = 3 \cdot 2^{e/2-1}$).

In contrast, the cost of building the table in MITM-basic (with $\ell = 2$) is approximately $\frac{Ne}{2}$ 2-isogeny computations, $\frac{Ne}{4} \log_2 \frac{e}{2}$ 2-isogeny evaluations, and $\frac{Ne}{4} \log_2 \frac{e}{2}$ point doublings (cf. Remark 11). A count of $\mathbb{F}_{p^2}$ multiplications and squarings yields the following costs for the core operations when Jacobian coordinates are used for elliptic curve arithmetic, isogeny computations, and isogeny evaluations: 8 (2-isogeny computation), 12 (2-isogeny evaluation), 14 (point addition), 9 (point doubling). This gives a per-table cost of approximately $5.25Ne\log_2 e$ for MITM-basic, and a cost of $65N$ for MITM-DFS. Thus, the depth-first search approach yields a speedup by a factor of approximately $\frac{e}{12.4}\log_2 e$.

6.4 Golden collision search

6.4.1 Van Oorschot-Wiener parallel collision search

Let S be a finite set of cardinality M, and let $f : S \to S$ be an efficiently-computable function which we shall heuristically assume is a random function. The van Oorschot-Wiener (VW) method [100] finds a collision for f, i.e., a pair $x, x' \in S$ with $f(x) = f(x')$ and $x \neq x'$.

Define an element x of S to be *distinguished* if it has some easily-testable distinguishing property. Suppose that the proportion of elements of S that are distinguished is θ. For $i = 1, 2, \ldots$, the VW method repeatedly selects $x_{i,0} \in_R S$, and iteratively computes a sequence $x_{i,j} = f(x_{i,j-1})$ for $j = 1, 2, 3, \ldots$ until a distinguished element $x_{i,a}$ is encountered. In that event, the triple $(x_{i,a}, a, x_{i,0})$ is stored in a table sorted by first entry. If $x_{i,a}$ was already in the table, say $x_{i,a} = x_{i',b}$ with $i \neq i'$, then a collision has been *detected* (see Figure 6.2). The two colliding table entries $(x_{i,a}, a, x_{i,0})$, $(x_{i',b}, b, x_{i',0})$ can then be used to find a collision for f by iterating the longer sequence (say the ith sequence) beginning at $x_{i,0}$ until it is the same distance from $x_{i,a}$ as $x_{i',0}$ is from $x_{i',b}$, and then stepping both sequences in unison until they collide (see Figure 6.3).

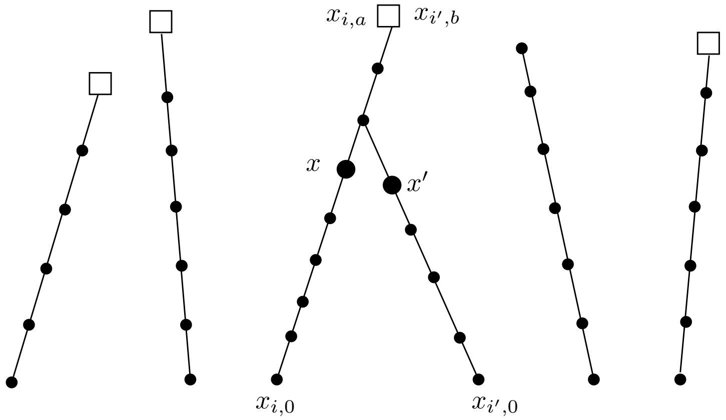


Figure 6.2: VW method: detecting a collision (x, x').

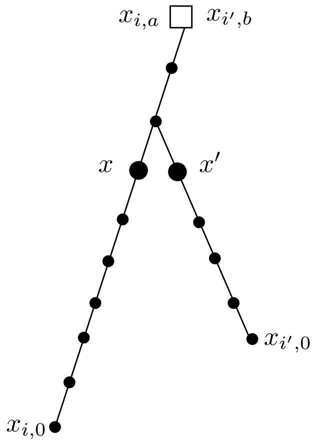


Figure 6.3: VW method: finding a collision (x, x').

By the birthday paradox, the expected time before a collision occurs is $\sqrt{\pi M/2}$, where a unit of time is an f evaluation. After a collision has occurred, the expected time before it is detected is $1/\theta$, and thereafter the expected time to find the collision is approximately $3/\theta$. Thus, the expected time complexity of the VW method is approximately $\sqrt{\pi M/2} + 4/\theta$. The expected storage complexity is $\theta\sqrt{\pi M/2}$. The parameter θ can be selected to control the storage requirements.

The collision detecting stage of the VW method can be effectively parallelized. Each of the available m processors computes its own sequences, and the distinguished elements are stored in shared memory. The expected time complexity of parallelized VW is then $\frac{1}{m}\sqrt{\pi M/2} + \frac{2.5}{\theta}$. The space complexity is $\theta\sqrt{\pi M/2}$.

6.4.2 Finding a golden collision

A random function $f : S \to S$ is expected to have $(M-1)/2$ unordered collisions. Suppose that we seek a particular one of these collisions, called a *golden collision*; we assume that the golden collision can be efficiently recognized. Thus one continues generating distinguished points and collisions until the golden collision is encountered. The expected time to find q collisions is only about $\sqrt{q}$ times as much as that to find one collision. However, since not all collisions are equally likely and the golden collision might have a very low probability of detection (see [101]), it is necessary to change the version of f periodically.

Suppose that the available memory can store w triples $(x_{i,a}, a, x_{i,0})$. When a distinguished point $x_{i,a}$ is encountered, the triple $(x_{i,a}, a, x_{i,0})$ is stored in a memory cell determined by hashing $x_{i,a}$. If that memory cell was already occupied with a triple holding a distinguished point $x_{i',b} = x_{i,a}$, then the two triples are used to locate a collision.

Van Oorschot and Wiener proposed setting

$$\theta = \alpha\sqrt{w/M} \tag{6.1}$$

and using each version of f to produce βw distinguished points. Experimental data presented in [100] suggested that the total running time to find the golden collision is minimized by setting $\alpha = 2.25$ and $\beta = 10$. Then, for $2^{10} \leq w \leq M/2^{10}$, the expected running time to find the golden collisions when m processors are employed is slightly overestimated as

$$\frac{1}{m}(2.5\sqrt{M^3/w}). \tag{6.2}$$

Remark 16. (*verifying the VW heuristic analysis*) The running time estimate (6.2) relies on several heuristics, the most significant of which is that when $2^{10} \leq w \leq M/2^{10}$ then each version of f generates approximately $1.3w$ collisions, of which approximately $1.1w$ are distinct. The numbers $1.3w$ and $1.1w$ were determined experimentally in [100]. Then the probability that a particular version of f yields the golden collision is approximately $1.1w/(M/2)$, whence the expected number of function versions needed to locate the golden collision is approximately $0.45M/w$, and the expected total time is

$$0.45\frac{M}{w} \times 10w \times \frac{1}{2.25}\sqrt{M/w} \;\approx\; 2\sqrt{M^3/w}.$$

To verify these numbers, we ran some experiments using a "random" function $f_{n,v} : \{0,1\}^n \to \{0,1\}^n$ (so $M = 2^n$), where v is a string identifying the function version, and $f_{n,v}(X)$ is defined to be the n most significant bits of $\mathrm{MD5}(v, X)$. Table 6.1 lists the numbers of collisions and distinct collisions that were found for different values of (n, w), confirming the $1.3w$ and $1.1w$ numbers reported in [100].

6.4.3 The attack

Let $I = \{1, 2, \ldots, N\}$ and $S = \{1, 2\} \times I$. For $i = 1, 2$, let $\mathcal{A}_i$ denote the set of all order-$\ell^{e/2}$ subgroups of $E_i[\ell^e]$, define $f_i : \mathcal{A}_i \to R_i$ by $f_i(A_i) = j(E_i/A_i)$, and let $h_i : I \to \mathcal{A}_i$ be bijections. Let $g : R \to S$ be a random function. Finally, define $f : S \to S$ by

$$f : (i, x) \mapsto g(f_i(h_i(x))).$$

Then one can view f as a "random" function from S to S.

Recall that one expects there are unique order-$\ell^{e/2}$ subgroups A_1, A_2 of $E_1[\ell^e]$, $E_2[\ell^e]$, respectively, with $j(E_1/A_1) = j(E_2/A_2)$. Let $y_1 = h_1^{-1}(A_1)$ and $y_2 = h_2^{-1}(A_2)$. Then the collision for f that we seek is the golden collision $(1, y_1)$, $(2, y_2)$. Using m processors and w cells of memory, the VW method can be used to find this golden collision in expected time

$$\frac{1}{m}(2.5\sqrt{8N^3/w}) \approx 7.1p^{3/8}/(w^{1/2}m).$$

w	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
$M = 2^{20}$																	
c_1	1.66	1.30	1.48	1.30	1.48	1.38	1.28	1.27	1.29	1.27	1.28	1.27	1.24	1.18	1.08	—	—
c_2	1.31	1.14	1.26	1.11	1.22	1.15	1.08	1.05	1.03	1.02	1.03	1.00	0.94	0.83	0.61	—	—
$M = 2^{24}$																	
c_1	1.38	1.36	1.38	1.37	1.33	1.31	1.31	1.36	1.32	1.33	1.31	1.30	1.30	1.29	1.29	1.27	1.24
c_2	1.21	1.14	1.16	1.16	1.12	1.10	1.11	1.13	1.11	1.11	1.09	1.06	1.06	1.05	1.04	1.00	0.95
$M = 2^{28}$																	
c_1	1.09	1.21	1.33	1.35	1.36	1.35	1.30	1.34	1.32	1.34	1.33	1.34	1.33	1.32	1.31	1.31	1.30
c_2	0.98	1.06	1.10	1.15	1.15	1.12	1.09	1.12	1.12	1.13	1.12	1.13	1.12	1.10	1.08	1.07	1.07
$M = 2^{32}$																	
c_1	1.21	1.44	1.35	1.35	1.35	1.31	1.30	1.32	1.33	1.35	1.33	1.34	1.33	1.34	1.33	1.33	1.32
c_2	1.00	1.18	1.17	1.12	1.16	1.10	1.10	1.11	1.13	1.13	1.13	1.13	1.12	1.13	1.12	1.12	1.11
$M = 2^{36}$																	
c_1	1.34	1.31	1.29	1.32	1.38	1.34	1.31	1.32	1.35	1.32	1.33	1.34	1.33	1.33	1.33	1.33	1.33
c_2	1.10	1.10	1.08	1.13	1.16	1.13	1.11	1.10	1.13	1.12	1.12	1.13	1.13	1.13	1.13	1.13	1.13

Table 6.1: Observed number c_1w of collisions and number c_2w of distinct collisions per version v of the MD5-based random function $f_{n,v} : \{0,1\}^n \to \{0,1\}^n$. The numbers are averages for 20 function versions when $w \leq 2^8$ and 10 function versions when $w \geq 2^9$.

Remark 17. (*finding any collision vs. finding a golden collision*) The problem of finding a collision for a hash function $H : \{0,1\}^* \to \{0,1\}^n$ and the problem of computing discrete logarithms in a cyclic group $\mathcal{G}$ can be formulated as problems of finding a collision for a random function $f : S \to S$, where $\#S = 2^n$ for the first problem and $\#S = \#\mathcal{G}$ for the second problem (see [100]). For both formulations, *any* collision for f yields a solution to the original problem. Thus, letting $N = 2^n$ or $N = \#\mathcal{G}$, the problems can be solved using van Oorschot-Wiener collision search in time approximately

$$\frac{1}{m}N^{1/2}.$$

In contrast, the only formulation of CSSI as a collision search problem for $f : S \to S$ that we know requires one to find a *golden* collision for f. For this problem, the van Oorschot-Wiener algorithm has running time approximately

$$N^{3/2}/(w^{1/2}m).$$

6.4.4 Implementation report

The VW attack (for $\ell = 2$) was implemented in C, compiled using gcc version 4.7.2, and executed on an Intel Xeon processor E5-2658 v2 server equipped with 20 physical cores and 256 GB of shared RAM memory. We used fopenmp for the parallelization and openssl's MD5 implementation. The CSSI challenges were the same as the ones in §3.3.

Let $\{P_1, Q_1\}$, $\{P_2, Q_2\}$ be bases for $E_1[2^{e/2}]$, $E_2[2^{e/2}]$, respectively. Noting that $N = 3 \cdot 2^{e/2-1}$, we identify the elements of $I = \{1, 2, \ldots, N\}$ with elements of $I_1 \times I_2$ where $I_1 = \{0, 1, 2\}$ and $I_2 = \{0, 1, \ldots, 2^{e/2-1} - 1\}$. The bijections $h_i : I_1 \times I_2 \to \mathcal{A}_i$ for $i = 1, 2$ are defined by

$$h_i : (b, k) \mapsto \begin{cases} P_i + (b2^{e/2-1} + k)Q_i, & \text{if } b = 0, 1, \\ (2k)P_i + Q_i, & \text{if } b = 2. \end{cases}$$

Let $S = \{1, 2\} \times I_1 \times I_2$. For $n \in \{0, 1\}^{64}$, we let $g_n : R \to S$ be the function computed using Algorithm 6. We then define the version $f_n : S \to S$ of f by $(i, x) \mapsto g_n(f_i(h_i(x)))$.

Algorithm 6 The "random" function g_n

```
**Require:** $n \in \{0,1\}^{64}$ and $j \in \mathbb{F}_{p^2}$.
**Ensure:** Output $c \in \{1,2\}$, $b \in I_1$, $k \in I_2$.
$counter := 0$.
**repeat**
    $h := \text{MD5}(1, j, n, counter)$.
    Let $h'$ be the $e/2 + 2$ least significant bits of $h$, and parse $h'$ as $(k, c, b)$, where $k, c, b$ have
    bitlengths $e/2 - 1$, 1, and 2, respectively.
    $counter := counter + 1$.
**until** $b \neq 11$
**return** $(c + 1, b, k)$.
```

We set $\theta = 2.25\sqrt{w/2N}$, where $w = 2^t$, and declare an element $X \in S$ to be distinguished if the integer formed from the 32 least significant bits of $\text{MD5}(2, X)$ is $\leq 2^{32}\theta$. If X is distinguished, then it is placed in memory cell s, where s is the integer determined by the t least significant bits of $\text{MD5}(3, X)$. If a distinguished point is not encountered after $10/\theta$ iterations, then that trail is abandoned and a new trail is formed.

Table 6.2 shows the time expended for finding 2^e-isogenies for $e \in \{32, 34, 36, 38, 40, 42, 44\}$ with the VW attack. These experimental results confirm the accuracy of the VW attack's heuristic analysis.

To gain further confidence that the VW attack's heuristic analysis is accurate for cryptographically-interesting CSSI parameters (e.g., $e = 256$), we ran some experiments to estimate the number of collisions and distinct collisions for functions f_n when $e = 50, 60, 70, 80$. The results, listed in Table 6.3, confirm the $1.3w$ and $1.1w$ estimates in [100].

							median			average	
				expected	number		measured	clock		measured	clock
e_A	e_B	d	w	time	of runs	# f_n's	time	cycles	# f_n's	time	cycles
32	20	23	2^9	$2^{23.20}$	25	180	$2^{23.55}$	$2^{40.79}$	319	$2^{24.38}$	$2^{41.62}$
34	21	109	2^9	$2^{24.70}$	25	256	$2^{24.54}$	$2^{41.89}$	714	$2^{26.02}$	$2^{43.37}$
36	22	31	2^{10}	$2^{25.70}$	25	369	$2^{26.06}$	$2^{43.51}$	838	$2^{27.25}$	$2^{44.70}$
38	23	271	2^{11}	$2^{26.70}$	25	196	$2^{26.15}$	$2^{43.70}$	567	$2^{27.69}$	$2^{45.23}$
40	25	71	2^{11}	$2^{28.20}$	25	162	$2^{26.36}$	$2^{43.99}$	1015	$2^{29.01}$	$2^{46.64}$
42	26	37	2^{12}	$2^{29.20}$	25	477	$2^{28.92}$	$2^{46.52}$	1940	$2^{30.95}$	$2^{48.55}$
44	27	37	2^{13}	$2^{30.20}$	25	431	$2^{29.78}$	$2^{47.46}$	942	$2^{30.91}$	$2^{48.58}$

Table 6.2: Van Oorschot-Wiener golden collision search for finding a 2^{e_A}-isogeny between two supersingular elliptic curves over $\mathbb{F}_{p^2}$ with $p = 2^{e_A} \cdot 3^{e_B} \cdot d - 1$. For each p, the listed number of CSSI instances were solved and the median and average of the results are reported. The $\#f_n$'s column indicates the number of random functions f_n that were tested before the golden collision was found. The expected and measured times list the number of degree-$2^{e_A/2}$ isogeny computations.

6.5 Comparisons

There are many factors that can affect the efficacy of an algorithm.

1. *Time*: the worst-case or average-case number of basic arithmetic operations performed by the algorithm.

2. *Space*: the amount of storage (RAM, hard disk, etc.) required.

3. *Parallelizability*: the speedup achievable when running the algorithm on multiple processors. Ideally, the speedup is by a factor equal to the number of processors, and the processors do not need to communicate with each other; if this is the case then the parallelization is said to be *perfect*[2].

4. *Communication costs*: the time taken for communication between processors, and the memory access time for retrieving data from large storage devices. Memory access time can be a dominant cost factor when using extremely large storage devices [4].

5. *Custom-designed devices*: the possible speedups that can be achieved by executing the algorithm on custom-designed hardware. Examples of such devices are TWIN-

[2] If the processors share the same storage space, then frequent storage accesses might decrease the parallelizability of the algorithm.

e	p	w	2^8	2^{10}	2^{12}	2^{14}	2^{16}
50	$2^{50}3^{31}179-1$	c_1	1.37	1.36	1.37	1.41	1.49
		c_2	1.14	1.12	1.12	1.11	1.09
60	$2^{60}3^{37}31-1$	c_1	1.37	1.34	1.34	1.35	1.36
		c_2	1.15	1.13	1.13	1.12	1.12
70	$2^{70}3^{32}127-1$	c_1	1.33	1.34	1.34	1.34	1.34
		c_2	1.13	1.14	1.13	1.13	1.13
80	$2^{80}3^{25}71-1$	c_1	1.35	1.32	1.33	1.34	1.33
		c_2	1.14	1.12	1.13	1.13	1.13

Table 6.3: Observed number c_1w of collisions and number c_2w of distinct collisions per CSSI-based random function f_n. The numbers are averages for 25 function versions (except for $(e,w) \in \{(80,2^{12}),(80,2^{14}),(80,2^{16})\}$ for which 5 function versions were used).

KLE [90] and TWIRL [91] that were designed for the number field sieve integer factorization algorithm.

In this section we analyze and compare the efficacy of the meet-in-the-middle algorithm, VW golden collision search, and a mesh sorting algorithm for solving CSSI. We make two assumptions:

1. The number m of processors available is at most 2^{64}.

2. The total amount of storage w available is at most 2^{80} units.

Our analysis will ignore communication costs, and thus our running time estimates can be considered to be lower bounds on the "actual" running time.

Remark 18. (*feasible amount of storage and number of processors*) The Sunway TaihuLight supercomputer, the most powerful in the world as of March 2018, has $2^{23.3}$ CPU cores [3]. In 2013, it was estimated that Google's data centres have a total storage capacity of about a dozen exabytes[4] [5]. Thus it is reasonable to argue that acquiring 2^{64} processors and a storage capacity (with low access times) of several dozen yottabytes[6] for the purpose of solving a CSSI problem will be prohibitively costly for the foreseeable future.

[3] Wikipedia, *Sunway TaihuLight*, `https://en.wikipedia.org/wiki/Sunway_TaihuLight`.

[4] An exabyte is 2^{60} bytes.

[5] Wikipedia, *Exabyte*, `https://en.wikipedia.org/wiki/Exabyte#Google`.

[6] A yottabyte is 2^{80} bytes.

6.5.1 Meet-in-the-middle

As stated in §6.3, the running time of MITM-basic and MITM-DFS is approximately $2N$ and the storage requirements are N, where $N \approx p^{1/4}$. Since for $N \geq 2^{80}$ the storage requirements are infeasible, we deem the meet-in-the-middle attacks to be prohibitively expensive when $N \gg 2^{80}$.

Of course, one can trade space for time. One possible time-memory tradeoff is to store a table with entries $(j(E_1/A_1), A_1)$, where A_1 ranges over a w-subset of order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $E_2[\ell^e]$, E_2/A_2 is computed and $j(E_2/A_2)$ is searched in the table. If no match is found, then the algorithm is repeated for a disjoint w-subset of order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$, and so on. The running time of this time-memory tradeoff is approximately

$$(w + N)\frac{N}{w} \approx N^2/w.$$

For MITM-basic, the unit of time is an $\ell^{e/2}$-isogeny computation. For MITM-DFS, the running time (for $\ell = 2$) can be scaled to $\ell^{e/2}$-isogeny computations by dividing by $\frac{e}{12.4}\log_2 e$ (cf. §3.2). One can see that this time-memory-tradeoff can be parallelized perfectly.

Another possible time-memory tradeoff is to store $(j(E_1/A_1), A_1)$, where A_1 ranges over all order-ℓ^c subgroups of $E_1[\ell^e]$ and $c \approx \log_\ell w$. Let $d = e - c$. Then, for each order-ℓ^d subgroup A_2 of $E_2[\ell^e]$, E_2/A_2 is computed and $j(E_2/A_2)$ is searched in the table. One can check that the running time of this time-memory tradeoff is approximately N^2/w, and that it can be parallelized perfectly. Note that the unit of time here is an ℓ^d-isogeny computation instead of an $\ell^{e/2}$-isogeny computation. The larger tree of ℓ^d-isogenies can be traversed using a depth-first search; the running time is then the same as that of the MITM-DFS variant described in the previous paragraph.

6.5.2 Golden collision search

As stated in §4.3, the running time of van Oorschot-Wiener golden collision search is approximately

$$N^{3/2}/w^{1/2}.$$

The algorithm parallelizes perfectly.

6.5.3 Mesh sorting

The mesh sorting attack is analogous to the one described by Bernstein [4] for finding hash collisions. Suppose that one has m processors arranged in a two-dimensional grid. Each processor only communicates with its neighbours in the grid. In one unit of time, each processor computes and stores pairs $(j(E_1/A_1), A_1)$, where A_1 is an order-$\ell^{e/2}$ subgroup of $E_1[\ell^e]$. Next, these stored pairs are sorted in time $\approx m^{1/2}$ (e.g., see [86]). In the next stage, a second two-dimensional grid of m processors computes and stores pairs $(j(E_2/A_2, A_2)$, where A_2 is an order-$\ell^{e/2}$ subgroup of $E_2[\ell^e]$, and the two sorted lists are compared for a match. This is repeated for a disjoint m-subset of order-$\ell^{e/2}$ subgroups A_2 until all order-$\ell^{e/2}$ subgroups of $E_2[\ell^e]$ have been tested. Then, the process is repeated for a disjoint subset of order-$\ell^{e/2}$ subgroups A_1 of $E_1[\ell^e]$ until a match is found. One can check that the calendar running time[7] is approximately

$$\left(m^{1/2} + m^{1/2}\frac{N}{m}\right)\frac{N}{m} \approx N^2/m^{3/2}.$$

6.5.4 Targetting the 128-bit security level

The CSSI problem is said to have a 128-bit security level if the fastest known attack has total time complexity at least 2^{128} and feasible space and hardware costs.

Suppose that $p \approx 2^{512}$, whereby $N \approx 2^{128}$; this would be a reasonable choice for the bitlength of p if the meet-in-the-middle attacks were assessed to be the fastest (classical) algorithm for solving CSSI. However, as noted above, the storage costs for the attacks are prohibitive. Instead, one should consider the time complexity of the time-memory tradeoffs, VW golden collision search, and mesh sorting under realistic constraints on the storage space w and the number m of processors. Table 6.4 lists the calendar time and the total time of these CSSI attacks for $(m, w) \in \{(2^{48}, 2^{64}), (2^{48}, 2^{80}), (2^{64}, 2^{80})\}$. One sees that in all cases the total time complexity is significantly greater than 2^{128}, even though we have ignored communication costs.

Since the total times for $p \approx 2^{512}$ in Table 6.4 are all significantly greater than 2^{128}, one can consider using smaller primes p while still achieving the 128-bit security level. Table 6.4 also lists the calendar time and the total time of these CSSI attacks for $(m, w) \in \{(2^{48}, 2^{64}), (2^{48}, 2^{80}), (2^{64}, 2^{80})\}$ when $p \approx 2^{448}$ and $N \approx 2^{112}$. One sees that all attacks have total time complexity at least 2^{128}, even though we have ignored communication costs. We can conclude that selecting SIDH parameters with $p \approx 2^{448}$ provides 128 bits of security against known classical attacks. For example, one could select the 434-bit prime

$$p_{434} = 2^{216}3^{137} - 1;$$

this prime is balanced in the sense that $3^{137} \approx 2^{217}$, thus providing maximal resistance to Petit's SIDH attack [83].

Remark 19. (*communication costs*) Consider the case $p \approx 2^{448}$, $e = 224$, $m = 2^{64}$, $w = 2^{80}$. From (6.1) and (6.2) we obtain $\theta \approx 1/2^{15.62}$ and an expected running time of $2^{131.7}$. For each function version, the 2^{64} processors will generate approximately $2^{48.4}$ distinguished points per unit of time (i.e., a 2^{112}-isogeny computation). So, on average, the 2^{80} storage

[7] *Calendar time* is the elapsed time taken for a computation, whereas *total time* is the sum of the time expended by all m processors.

			$p \approx 2^{512}$		$p \approx 2^{448}$	
	# processors	space	calendar	total	calendar	total
	m	w	time	time	time	time
Meet-in-the-middle (DFS)	48	64	138	186	106	154
time-memory tradeoff	48	80	122	170	90	138
	64	80	106	170	74	138
Van Oorschot-Wiener	48	64	112	160	88	136
golden collision search	48	80	104	152	80	128
	64	80	88	152	64	128
Mesh sorting	48	—	184	232	152	200
	64	—	160	224	128	192

Table 6.4: Time complexity estimates of CSSI attacks for $p \approx 2^{512}$ and $p \approx 2^{448}$, and $\ell = 2$. All numbers are expressed in their base-2 logarithms. The unit of time is a $2^{e/2}$-isogeny computation.

device will be accessed $2^{48.4}$ times during each unit of time. The cost of these accesses will certainly dominate the computational costs. Thus our security estimates, which ignore communication costs, should be regarded as being conservative.

6.5.5 Targetting the 160-bit security level

Using similar arguments as in §5.4, one surmises that SIDH parameters with $p \approx 2^{536}$ offer at least 160 bits of CSSI security against known classical (see Table 6.5). For example, one could select the 546-bit prime

$$p_{546} = 2^{273}3^{172} - 1;$$

this prime is nicely balanced since $3^{172} \approx 2^{273}$.

6.5.6 Targetting the 192-bit security level

Using similar arguments as in §5.4, one surmises that SIDH parameters with $p \approx 2^{614}$ offer at least 192 bits of CSSI security against known classical (see Table 6.5). For example, one could select the 610-bit prime

$$p_{610} = 2^{305}3^{192} - 1;$$

this prime is nicely balanced since $3^{192} \approx 2^{304}$.

	# processors	space	$p \approx 2^{536}$		$p \approx 2^{614}$	
	m	w	calendar time	total time	calendar time	total time
Meet-in-the-middle (DFS) time-memory tradeoff	48	64	150	198	188	236
	48	80	134	182	172	220
	64	80	118	182	156	220
Van Oorschot-Wiener golden collision search	48	64	121	169	149	197
	48	80	113	161	141	189
	64	80	97	161	125	189
Mesh sorting	48	—	196	244	234	282
	64	—	172	236	210	274

Table 6.5: Time complexity estimates of CSSI attacks for $p \approx 2^{536}$ and $p \approx 2^{614}$, and $\ell = 2$. All numbers are expressed in their base-2 logarithms. The unit of time is a $2^{e/2}$-isogeny computation.

6.5.7 Resistance to quantum attacks

The appeal of SIDH is its apparent resistance to attacks by quantum computers. What remains to be determined then is the security of CSSI against quantum attacks.

The fastest known quantum attack on CSSI is Tani's algorithm [94]. Given two generic functions $g_1 : X_1 \to Y$ and $g_2 : X_2 \to Y$, where $\#X_1 \approx \#X_2 \approx N$ and $\#Y \gg N$, Tani's quantum algorithm finds a claw, i.e., a pair $(x_1, x_2) \in X_1 \times X_2$ such that $g_1(x_1) = g_2(x_2)$ in time $O(N^{2/3})$. The CSSI problem can be recast as a claw-finding problem by defining X_i to be the set of all degree-$\ell^{e/2}$ isogenies originating at E_i, g_i to be the function that maps a degree-$\ell^{e/2}$ isogeny originating at E_i to the j-invariant of its image curve, and $Y = R$. Since $\#X_1 = \#X_2 = N \approx p^{1/4}$, this yields an $O(p^{1/6})$-time CSSI attack.

CSSI can also be solved by an application of Grover's quantum search [42]. Recall that if $g : X \to \{0, 1\}$ is a generic function such that $g(x) = 1$ for exactly one $x \in X$, then Grover's algorithm can determine the x with $g(x) = 1$ in quantum time $O(\sqrt{\#X})$. The CSSI problem can be recast as a Grover search problem by defining X to be the set of all ordered pairs (ϕ_1, ϕ_2) of degree-$\ell^{e/2}$ isogenies originating at E_1, E_2, respectively, and defining $g(\phi_1, \phi_2)$ to be equal to 1 if and only if the j-invariants of the image curves of ϕ_1 and ϕ_2 are equal. Since $\#X = N^2 \approx p^{1/2}$, this yields an $O(p^{1/4})$-time quantum attack on CSSI.

The Jao-De Feo paper [49] that introduced SIDH identified Tani's claw-finding algorithm as the fastest known attack, whether classical or quantum, on CSSI. The subsequent literature on SIDH used the simplified running time $p^{1/6}$ of Tani's algorithm (i.e., ignoring the implied constant in its $O(p^{1/6})$ running time expression) to select SIDH primes p

for a desired level of security. In other words, in order to achieve a b-bit security level against known classical and quantum attacks, one selects an SIDH prime p of bitlength approximately $6b$. For example, the 751-bit prime $p = 2^{372}3^{239} - 1$ was proposed in [26] for the 128-bit security level, and this prime has been used in many subsequent works, e.g., [60, 23, 24, 3, 107]. Also, the 964-bit prime $p = 2^{486}3^{301} - 1$ was proposed in [3] for the 160-bit security level.

However, this assessment of SIDH security does not account for the cost of the $O(p^{1/6})$ quantum space requirements of Tani's algorithm, nor for the fact that Grover's search does not parallelize well — using m quantum circuits only yields a speedup by a factor of $\sqrt{m}$ and this speedup has been proven to be optimal [106]. Some recent work [1, 50] suggests that Tani's and Grover's attacks on CSSI are costlier than the van Oorschot-Wiener golden collision search algorithm. If this is indeed the case, then one can be justified in selecting SIDH primes p_{434} (instead of p_{751}), p_{546} (instead of p_{964}) and p_{610} in order to achieve the 128-, 160- and 192-bit security levels, respectively, against both classical and quantum attacks. Furthermore, SIDH parameters with p_{434} could be deemed to meet the security requirements in NIST's Category 2 [72] (classical and quantum security comparable or greater than that of SHA-256 with respect to collision resistance), and p_{610} could be deemed to meet the security requirements in NIST's Category 4 [72] (classical and quantum security comparable to that of SHA-384).

6.5.8 SIDH performance

A significant benefit of using smaller SIDH primes is increased performance. The reasons for the boost in SIDH performance are twofold. First, since the computation of the ground field $\mathbb{F}_p$ multiplication operation has a quadratic complexity, any reduction in the size of p will result in significant savings. Since high-end processors have a word size of 64 bits, the primes p_{751}, p_{546} and p_{434} can be accommodated using twelve, nine and seven 64-bit words, respectively. Hence, if $\mathbb{F}_p$ multiplication using p_{751} can be computed in T clock cycles, then a rough estimation of the computational costs for $\mathbb{F}_p$ multiplication using p_{434} and p_{546} is as low as $0.34T$ and $0.56T$, respectively. Second, since the exponents of the primes 2 and 3 in p_{434} and p_{546} are smaller than the ones in p_{751}, the computation of the isogeny chain described in §2.2 (see Remark 11) is faster.

Table 6.6 lists timings for SIDH operations for p_{434}, p_{546} and p_{751} using the SIDH library of Costello et al. [22]. The timings show that SIDH operations are about 4.8 times faster when p_{434} is used instead of p_{751}.

6.6 Concluding remarks

Our implementations of the MITM and golden collision search CSSI attacks are, to the best of our knowledge, the first ones reported in the literature. The implementations confirm that the performance of these attacks is accurately predicted by their heuristic analysis.

Protocol phase		CLN library [26]			CLN + enhancements		
		p_{751}	p_{434}	p_{546}	p_{751}	p_{434}	p_{546}
Key Gen.	Alice	35.7	7.51	13.20	26.9	5.3	10.5
	Bob	39.9	8.32	14.84	30.5	6.0	11.7
Shared Secret	Alice	33.6	7.01	12.56	24.9	5.0	10.0
	Bob	38.4	7.94	14.35	28.6	5.8	11.5

Table 6.6: Performance of the SIDH protocol. All timings are reported in 10^6 clock cycles, measured on an Intel Core i7-6700 supporting a Skylake micro-architecture. The "CLN + enhancements" columns are for our implementation that incorporates improved formulas for degree-2 and degree-3 isogenies from [23] and Montgomery ladders from [38] into the CLN library.

Our concrete cost analysis of the attacks leads to the conclusion that golden collision search is more effective that the meet-in-the-middle attack. Thus one can use 448-bit primes and 536-bit primes p in SIDH to achieve the 128-bit and 160-bit security levels against *known* classical attacks on the CSSI problem. We emphasize that these conclusions are based on our understanding of how to best implement these algorithms, and on assumptions on the amount of storage and the number of processors that an adversary might possess. On the other hand, our conclusions are somewhat conservative in that the analysis does not account for communication costs. Moreover, whereas it is generally accepted that the AES-128 and AES-256 block ciphers attain the 128-bit security level in the classical and quantum settings, the time it takes to compute a degree-2^{112} isogeny (which is the unit of time for the golden collision search CSSI attack with balanced 448-bit prime p) is considerably greater than the time for one application of AES-128 or AES-256.

Chapter 7

Constant time CSIDH

7.1 Introduction

Isogeny-based cryptography was introduced by Couveignes [29], who defined a key exchange protocol similar to Diffie–Hellman based on the action of an ideal class group on a set of ordinary elliptic curves. Couveignes' protocol was independently rediscovered by Rostovtsev and Stolbunov [85, 93], who were the first to recognize its potential as a post-quantum candidate. Recent efforts to make this system practical have put it back at the forefront of research in post-quantum cryptography [33]. A major breakthrough was achieved by Castryck, Lange, Martindale, Panny, and Renes with CSIDH [15], a reinterpretation of Couveignes' system using supersingular curves defined over a prime field.

The first implementation of CSIDH completed a key exchange in less than 0.1 seconds, and its performance has been further improved by Meyer and Reith [66]. However, both [15] and [66] recognized the difficulty of implementing CSIDH with constant-time algorithms, that is, algorithms whose running time, sequence of operations, and memory access patterns do not depend on secret data. The implementations of [15] and [66] are thus vulnerable to simple timing attacks.

The first attempt at implementing CSIDH in constant-time was realized by Bernstein, Lange, Martindale, and Panny [8], but their goal was to obtain a fully deterministic reversible circuit implementing the class group action, to be used in quantum cryptanalyses.

The distinct problem of efficient CSIDH implementation with side-channel protection was first tackled by Jalali, Azarderakhsh, Mozaffari Kermani, and Jao [48], and independently by Meyer, Campos, and Reith [65], whose work was improved by Onuki, Aikawa, Yamazaki, and Takagi [81].

The approach of Jalali *et al.* is similar to that of [8], in that they achieve a stronger notion of constant time (running time independent from *all* inputs), at the cost of allowing the algorithm to fail with a small probability. In order to make the failure probability sufficiently low, they introduce a large number of useless operations, which make the performance significantly worse than the original CSIDH algorithm. This poor performance and possibility of failure reduces the interest of this implementation; we will not analyze it further here.

Meyer *et al.* take a different path: the running time of their algorithm is independent of the secret key, but not of the output of an internal random number generator. They claim a speed only 3.10 times slower than the unprotected algorithm in [66]. Onuki *et al.* introduced new improvements, claiming a speed-up of 27.35% over Meyer *et al.*, i.e., a net slow-down factor of 2.25 compared to [66].

Our contribution. In this Chapter we take a new look at side-channel protected implementations of CSIDH. We start by reviewing the implementations in [65] and [81]. We highlight some flaws that make their constant-time claims disputable, and propose fixes for them. Since these fixes introduce some minor slow-downs, we report on the performance of the revised algorithms.

Then, we introduce new optimizations to make both [65] and [81] faster: we improve isogeny formulas for the model, and we introduce the use of optimal addition chains in the scalar multiplications. With these improvements, we obtain a version of CSIDH protected against timing and some simple power analysis (SPA) attacks that is 25% more efficient than [65] and 15% more efficient than a repaired version of [81].

Then, we shift our focus to stronger security models. All constant-time versions of CSIDH presented so far use so-called “dummy operations”, i.e., computations whose result is not used, but whose role is to hide the conditional structure of the algorithm from timing and SPA attacks that read the sequence of operations performed from a single power trace. However, this countermeasure is easily defeated by fault-injection attacks, where the adversary may modify values during the computation. We propose a new constant-time variant of CSIDH without dummy operations as a first-line defence. The new version is only twice as slow as the simple constant-time version.

We conclude with a discussion of derandomized variants of CSIDH. The versions discussed previously are “constant-time” in the sense that their running time is uncorrelated to the secret key, however it depends on some (necessarily secret) seed to a PRNG. While this notion of “constant-time” is usually considered good enough for side-channel protection, one may object that a compromise of the PRNG or the seed generation would put the security of the implementation at risk, even if the secret was securely generated before-hand (with an uncomprised PRNG) as part of a long-term or static keypair. We observe that this dependence on additional randomness is not necessary: a simple modification of CSIDH, already considered in isogeny-based signature schemes [31, 35], can easily be made constant-time and free of randomness. Unfortunately this modification requires increasing substantially the size of the base field, and is thus considerably slower and not compatible with the original version. On the positive side, the increased field size makes it much more resistant to quantum attacks, a non-negligible asset in a context where the quantum security of CSIDH is still unclear; it can thus be seen as CSIDH variant for the paranoid.

Organization. In §7.2 we briefly recall ideas, algorithms and parameters from CSIDH [15]. In §7.3 we highlight shortcomings in [65] and [81] and propose ways to fix them. In §7.4 we introduce new optimizations compatible with all previous versions of CSIDH. In §7.5 we introduce a new algorithm for evaluating the CSIDH group action that is resistant against timing and some simple power analysis attacks, while providing protection against some fault injections. Finally, in §7.6 we discuss a more costly variant of CSIDH with stronger security guarantees.

Notation. **M**, **S**, and **A** denote the cost of computing a single multiplication, squaring, and addition (or subtraction) in $\mathbb{F}_p$, respectively. We assume that a constant-time equality test `isequal`(X, Y) is defined, returning 1 if $X = Y$ and 0 otherwise. We also assume that a constant-time conditional swap `cswap`(X, Y, b) is defined, exchanging (X, Y) if $b = 1$ (and not if $b = 0$).

7.2 CSIDH

We refer the reader to section 4.4 for CSIDH information, now we depict some improvements to the original protocol.

7.2.1 The Meyer–Campos–Reith constant-time algorithm

As Meyer, Campos and Reith observe in [65], Algorithm 4 performs fewer scalar multiplications when the key has the same number of positive and negative exponents than it does in the unbalanced case where these numbers differ. Algorithm 4 thus leaks information about the distribution of positive and negative exponents under timing attacks. Besides this, analysis of power traces would reveal the cost of each isogeny computation, and the number of such isogenies computed, which would leak the exact exponents of the private key.

In view of this vulnerability, Meyer, Campos and Reith proposed in [65] a constant-time CSIDH algorithm whose running time does not depend on the private key (though, unlike [48], it still varies due to randomness). The essential differences between the algorithm of [65] and classic CSIDH are as follows. First, to address the vulnerability to timing attacks, they choose to use only positive exponents in $[0, 10]$ for each ℓ_i, instead of $[-5, 5]$ in the original version, while keeping the same prime $p = \prod_{i=1}^{74} \ell_i - 1$. To mitigate power consumption analysis attacks, their algorithm always computes the maximal amount of isogenies allowed by the exponent, using dummy isogeny computations if needed.

Since these modifications generally produce more costly group action computations, the authors also provide several optimizations that limit the slow-down in their algorithm to a factor of 3.10 compared to [66]. These include the Elligator 2 map of [7] and [8], multiple batches for isogeny computation (SIMBA), and sample the exponents e_i from intervals of different sizes depending on ℓ_i.

7.2.2 The Onuki–Aikawa–Yamazaki–Takagi constant-time algorithm

Still assuming that the attacker can perform only power consumption analysis and timing attacks, Onuki, Aikawa, Yamazaki and Takagi proposed a faster constant-time version of CSIDH in [81].

The key idea is to use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (i.e., in $E(\mathbb{F}_p)$) and one in $\ker(\pi + 1)$ (i.e., in $E(\mathbb{F}_{p^2})$ with x-coordinate in $\mathbb{F}_p$). This allows them to avoid timing attacks, while keeping the same primes and exponent range $[-5, 5]$ as in the original CSIDH algorithm. Their algorithm also employs dummy isogenies to mitigate some power analysis attacks, as in [65]. With these improvements, they achieve a speed-up of 27.35% compared to [65].

We include pseudo-code for the algorithm of [81] in Algorithm 7, to serve both as a reference for a discussion of some subtle leaks in §7.3 and also as a departure point for our dummy-free algorithm in §7.5.

7.3 Repairing constant-time versions

Projective Elligator

Both [65] and [81] use the Elligator 2 map to sample a random point on the current curve E_A in step 6 of Algorithm 7. Elligator takes as input a random field element $u \in \{2, \ldots, \frac{p-1}{2}\}$ and the Montgomery A-coefficient from the current curve and returns a pair of points in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively.

To avoid a costly inversion of $u^2 - 1$, instead of sampling u randomly, Meyer, Campos and Reith[1] follow [8] and precompute a set of ten pairs $(u, (u^2 - 1)^{-1})$; they try them in order until one that produces a point Q passing the test in Step 12 is found. When this happens, the algorithm moves to the next curve, and Elligator can keep on using the next precomputed value of u, going back to the first value when the tenth has been reached. This is a major departure from [8], where *all* precomputed values of u are tried *for*

[1] Presumably, Onuki *et al.* do the same, however their exposition is not clear on this point, and we do not have access to their code.

Algorithm 7 The Onuki–Aikawa–Yamazaki–Takagi CSIDH algorithm for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve, and the exponent bound vector $(m_1, \dots, m_n)$, are system parameters. This algorithm computes exactly m_i isogenies for each ℓ_i.

```
Require: A supersingular curve $E_A\colon y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$, and an integer exponent
    vector $(e_1, \dots, e_n)$ with each $e_i \in [-m_i, m_i]$.
Ensure: $E_B\colon y^2 = x^3 + Bx^2 + x$ such that $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.
$(e'_1, \dots, e'_n) \leftarrow (m_i - |e_1|, \dots, m_i - |e_n|)$        //Number of dummy computations
$E_B \leftarrow E_A$
while some $e_i \neq 0$ or $e'_i \neq 0$ do
    $S \leftarrow \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$
    $k \leftarrow \prod_{i \in S} \ell_i$
    $(T_-, T_+) \leftarrow$ Elligator$(E_B, u)$        //$T_- \in E_B[\pi - 1]$ and $T_+ \in E_B[\pi + 1]$
    $(P_0, P_1) \leftarrow \big([(p+1)/k]T_+, [(p+1)/k]T_-\big)$
    for $i \in S$ do
        $s \leftarrow \mathrm{sign}(e_i)$        //Ideal $\mathfrak{l}_i^s$ to be used
        $Q \leftarrow [k/\ell_i]P_{\frac{1-s}{2}}$        //Secret kernel point generator
        $P_{\frac{1+s}{2}} \leftarrow [\ell_i]P_{\frac{1+s}{2}}$        //Secret point to be multiplied
        if $Q \neq \infty$ then
            if $e_i \neq 0$ then
                $(E_B, \varphi) \leftarrow$ QuotientIsogeny$(E_B, Q)$
                $(P_0, P_1) \leftarrow \big(\varphi(P_0), \varphi(P_1)\big)$
                $e_i \leftarrow e_i - s$.
            else        //Dummy operations
                $E_B \leftarrow E_B$
                $P_{\frac{1-s}{2}} \leftarrow [\ell_i]P_{\frac{1-s}{2}}$
                $e'_i \leftarrow e'_i - 1$
            end if
        end if
        $k \leftarrow k/\ell_i$
    end for
end while
return $B$
```

each isogeny computation, and the algorithm succeeds if at least one passes the test. And indeed the implementation of [65] leaks information on the secret via the timing channel:[2] since Elligator uses no randomness for u, its output only depends on the A-coefficient of the current curve, which itself depends on the secret key; but the running time of the algorithm varies and, not being correlated to u, it is necessarily correlated to A and thus to the secret.

Fortunately this can be easily fixed by (re)introducing randomness in the input to Elligator. To avoid field inversions, we use a projective variant: given $u \neq 0, 1$ and assuming $A \neq 0$, we write $V = (A : u^2 - 1)$, and we want to determine whether V is the abscissa of a projective point on E_A. Plugging V into the homogeneous equation

$$E_A : Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3$$

gives

$$Y^2(u^2-1)^2 = \big((A^2u^2 + (u^2-1)^2\big)A(u^2-1).$$

We can test the existence of a solution for Y by computing the Legendre symbol of the right hand side: if it is a square, the points with projective XZ-coordinates

$$T_+ = (A : u^2 - 1), \qquad\qquad T_- = (-Au^2 : u^2 - 1)$$

are in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively, otherwise their roles are swapped.

We are left with the case $A = 0$. Following [8], Meyer, Campos and Reith precompute once and for all a pair of generators T_+, T_- of $E_0[\pi - 1]$ and $E_0[\pi + 1]$, and output those instead of random points. This choice suffers from a similar issue to the previous one: because the points are output in a deterministic way, the running time of the whole algorithm will be correlated to the number of times the curve E_0 is encountered during the isogeny walk.

In practice, E_0 is unlikely to ever be encountered in a random isogeny walk, except as the starting curve in the first phase of a key exchange, thus this flaw seems hard to exploit. Nevertheless, we find it not significantly more expensive to use a different approach, also suggested in [8]: with $u \neq 0$, only on E_0, we define the output of Elligator as $T_+ = (u : 1), T_- = (-u : 1)$ when $u^3 + u$ is a square, and we swap the points when $u^3 + u$ is not a square.

With these choices, under reasonable heuristics experimentally verified in [8], the running time of the whole algorithm is uncorrelated to the secret key as long as the values of u are unknown to an adversary. We summarize our implementation of Elligator in Algorithm 8, generalizing it to the case of Montgomery curves represented by projective coefficients (see also Section 2.4.3).

[2] The Elligator optimization is described in §5.3 of [65]. The unoptimized constant-time version described in Algorithm 2 therein is not affected by this problem.

Algorithm 8 Constant-time projective Elligator

```
Require: A supersingular curve $E_{(A':C')} : C'y^2 = C'x^3 + A'x^2 + C'x$ over $\mathbb{F}_p$, and an
    element $u \in \{2, \dots, \frac{p-1}{2}\}$.
Ensure: A pair of points $T_+ \in E_{(A':C')}[\pi - 1]$ and $T_- \in E_{(A':C')}[\pi + 1]$.
$t \leftarrow A'\big((u^2-1)u^2A'^2C' + ((u^2-1)C')^3\big)$
$a \leftarrow$ isequal$(t, 0)$                                  //$t = 0$ iff $A' = 0$
$\alpha, \beta \leftarrow 0, u$
cswap$(\alpha, \beta, a)$                                       //$\alpha = 0$ iff $A' \neq 0$
$t' \leftarrow t + \alpha(u^2 + 1)$                            //$t' \neq 0$
$T_+ \leftarrow (A' + \alpha C'(u^2-1) : C'(u^2-1))$
$T_- \leftarrow (-A'u^2 - \alpha C'(u^2-1) : C'(u^2-1))$
$b \leftarrow$ Legendre_symbol$(t', p)$                        //$b = \pm 1$
$c \leftarrow$ isequal$(b, -1)$
cswap$(T_+, T_-, c)$
return $(T_+, T_-)$
```

7.3.1 Fixing a leaking branch in Onuki–Aikawa–Yamazaki–Takagi

The algorithm from [81], essentially reproduced in Algorithm 7, includes a conditional statement at Line 12 which branches on the value of the point Q computed at Line 10. But this value depends on the sign s of the secret exponent e_i, so the branch leaks information about the secret. We propose repairing this by always computing both $Q_0 \leftarrow [k/\ell_i]P_0$ and $Q_1 \leftarrow [k/\ell_i]P_1$ at Line 10, and replacing the condition in Line 12 with a test for $(Q_0 = \infty)$ **or** $(Q_1 = \infty)$ (and using constant-time conditional swaps throughout).[3] This fix is visible in Line 12 of Algorithm 10.

7.4 Optimizing constant-time implementations

In this subsection we propose several optimizations that are compatible with both non-constant-time and constant-time implementations of CSIDH.

7.4.1 Addition chains for a faster scalar multiplication

Since the coefficients in CSIDH scalar multiplications are always known in advance (they are essentially system parameters), there is no need to hide them by using constant-time scalar multiplication algorithms such as the classical Montgomery ladder. Instead, we can

[3] We also found a branch on secret data in the code provided with [65] at https://zenon.cs.hs-rm.de/pqcrypto/faster-csidh, during the 3-isogeny computation, when computing $[\ell]P = [(\ell-1)/2]P + [(\ell+1)/2]P$. This can be easily fixed by a conditional swap, without any significant impact on running time.

use shorter differential addition chains.[4]

In the CSIDH group action computation, any given scalar k is the product of a subset of the collection of the 74 small primes ℓ_i dividing $\frac{p+1}{4}$. We can take advantage of this structure to use shorter differential addition chains than those we might derive for general scalars of a comparable size. First, we pre-computed the shortest differential addition chains for each one of the small primes ℓ_i. One then computes the scalar multiplication operation $[k]P$ as the composition of the differential addition chains for each prime ℓ dividing k.

Power analysis on the coefficient computation might reveal the degree of the isogeny that is currently being computed, but, since we compute exactly one ℓ_i-isogeny for each ℓ_i per loop, this does not leak any secret information.

This simple trick allows us to compute scalar multiplications $[k]P$ using differential addition chains of length roughly $1.5\lceil \log_2(k) \rceil$. This yields a saving of about 25% compared with the cost of the classical Montgomery ladder.

7.5 Removing dummy operations for fault-attack resistance

The use of dummy operations in the previous constant-time algorithms implies that the attacker can obtain information on the secret key by injecting faults into variables during the computation. If the final result is correct, then she knows that the fault was injected in a dummy operation; if it is incorrect, then the operation was real. For example, if one of the values in in Line 20 of Algorithm 7 is modified without affecting the final result, then the adversary learns whether the corresponding exponent e_i was zero at that point.

Fault injection attacks have been considered in the context of SIDH ([41], [97]), but to the best of our knowledge, they have not been studied yet on dummy operations in the context of CSIDH. Below we propose an approach to constant-time CSIDH without dummy computations, making every computation essential for a correct final result. This gives us some natural resistance to fault, at the cost of approximately a twofold slowdown.

Our approach to avoiding fault-injection attacks is to change the format of secret exponent vectors $(e_1, \ldots, e_n)$. In both the original CSIDH and the Onuki *et al.* variants, the exponents e_i are sampled from an integer interval $[-m_i, m_i]$ centered in 0. For naive CSIDH, evaluating the action of $\mathfrak{l}_i^{e_i}$ requires evaluating between 0 and m isogenies, corresponding to either the ideal $\mathfrak{l}_i$ (for positive e_i) or $\mathfrak{l}_i^{-1}$ (for negative e_i). If we follow the approach of [81], then we must also compute $k - |e_i|$ dummy ℓ_i-isogenies to ensure a constant-time behaviour.

For our new algorithm, the exponents e_i are uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \bmod 2 \text{ and } |e| \leq m_i\},$$

[4] A differential addition chain is an addition chain such that for every chain element c computed as $a+b$, the difference $a-b$ is already present in the chain.

i.e., centered intervals containing only even or only odd integers. The interesting property of these sets is that a vector drawn from $\mathcal{S}(m)^n$ can always be rewritten (in a non-unique way) as a sum of m vectors with entries $\{-1, +1\}$ (i.e., vectors in $\mathcal{S}(1)^n$). But the action of a vector drawn from $\mathcal{S}(1)^n$ can clearly be implemented in constant-time without dummy operations: for each coefficient e_i, we compute and evaluate the isogeny associated to $\mathfrak{l}_i$ if $e_i = 1$, or the one associated to $\mathfrak{l}_i^{-1}$ if $e_i = -1$. Thus, we can compute the action of vectors drawn from $\mathcal{S}(m)^n$ by repeating m times this step.

More generally, we want to evaluate the action of vectors $(e_1, \dots, e_n)$ drawn from $\mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$. Algorithm 9 achieves this in constant-time and without using dummy operations. The outer loop at line 1 is repeated exactly $\max(m_i)$ times, but the inner "if" block at line 3 is only executed m_i times for each i; it is clear that this flow does not depend on secrets. Inside the "if" block, the coefficients e_i are implicitly interpreted as

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{(1-1) - (1-1) + (1-1) - \cdots}_{m_i - e_i \text{ times}},$$

i.e., the algorithm starts by acting by $\mathfrak{l}_i^{\texttt{sign}(e_i)}$ for e_i iterations, then alternates between $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ for $m_i - e_i$ iterations. We assume that the $\texttt{sign} : \mathbb{Z} \to \{\pm 1\}$ operation is implemented in constant time, and that $\texttt{sign}(0) = 1$. If one is careful to implement the isogeny evaluations in constant-time, then it is clear that the full algorithm is also constant-time.

Algorithm 9 An idealized dummy-free constant-time evaluation of the CSIDH group action.

```
Require: Secret vector $(e_1, \dots, e_n) \in \mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$ $(t_1, \dots, t_n) \leftarrow$
    $(\texttt{sign}(e_1), \dots, \texttt{sign}(e_n))$ (Secret) $(z_1, \dots, z_n) \leftarrow (m_1, \dots, m_n)$ (Not secret)
 1: while some $z_i \neq 0$ do
 2:   for $i \in \{1, \dots, n\}$ do
 3:     if $z_i > 0$ then
 4:       Act by $\mathfrak{l}_i^{t_i}$
 5:       $b = \texttt{isequal}(e_i, 0)$
 6:       $e_i \leftarrow e_i - t_i$
 7:       $t_i \leftarrow (-1)^b \cdot t_i$                //Swap sign when $e_i$ has gone past 0
 8:       $z_i \leftarrow z_i - 1$
 9:     end if
10:   end for
11: end while
```

However, Algorithm 9 is only an idealized version of the CSIDH group action algorithm. Indeed, like in [65, 81], it may happen in some iterations that Elligator outputs points of order not divisible by ℓ_i, and thus the action of $\mathfrak{l}_i$ or $\mathfrak{l}_i^{-1}$ cannot be computed in that iteration. In this case, we simply skip the loop and retry later: this translates into the variable z_i not being decremented, so the total number of iterations may end up being larger than $\max(m_i)$. Fortunately, if the input value u fed to Elligator is random, its output is uncorrelated to secret values[5], and thus the fact that an iteration is skipped does not leak information on the secret. The resulting algorithm is summarized in Algorithm 10.

To maintain the security of standard CSIDH, the bounds m_i must be chosen so that the key space is at least as large. For example, the original implementation [15] samples secrets in $[-5, 5]^{74}$, which gives a key space of size 11^{74}; hence, to get the same security we would need to sample secrets in $\mathcal{S}(10)^{74}$. But a constant-time version of CSIDH *à la* Onuki *et al.* only needs to evaluate five isogeny steps per prime ℓ_i, whereas the present variant would need to evaluate ten isogeny steps. We thus expect an approximately twofold slowdown for this variant compared to Onuki *et al.*, which is confirmed by our experiments.

7.6 Derandomized CSIDH algorithms

As we stressed in Section 7.3, all of the algorithms presented here depend on the availability of high-quality randomness for their security. Indeed, the input to Elligator must be randomly chosen to ensure that the total running time is uncorrelated to the secret key. Typically, this would imply the use of a PRNG seeded with high quality true randomness that must be kept secret. An attack scenario where the attacker may know the output of the PRNG, or where the quality of PRNG output is less than ideal, therefore degrades the security of all algorithms. This is true even when the secret was generated with a high-quality PRNG if the keypair is static, and the secret key is then used by an algorithm with low-quality randomness.

We can avoid this issue completely if points of order $\prod \ell_i^{|m_i|}$, where $|m_i|$ is the maximum possible exponent (in absolute value) for ℓ_i, are available from the start. Unfortunately this is not possible with standard CSIDH, because such points are defined over field extensions of exponential degree.

Instead, we suggest modifying CSIDH as follows. First, we take a prime $p = 4\prod_{i=1}^{n} \ell_i - 1$ such that $\lceil n \log_2(3) \rceil = 2\lambda$, where λ is a security parameter, and we restrict to exponents of the private key sampled from $\{-1, 0, 1\}$. Then, we compute two points of order $(p+1)/4$ on the starting public curve, one in $\ker(\pi - 1)$ and the other in $\ker(\pi + 1)$, where π is the Frobenius endomorphism. This computation involves no secret information and can be implemented in variable-time; furthermore, if the starting curve is the initial curve with $A = 0$, or a public curve corresponding to a long term secret key, these points can be precomputed offline and attached to the system parameters or the public key. We also remark that even for ephemeral public keys, a point of order $p+1$ must be computed anyway for key validation purposes, and thus this computation only slows down key validation by a factor of two.

[5] Assuming the usual heuristic assumptions on the distribution of the output of Elligator, see [65].

Algorithm 10 Dummy-free randomized constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n}\ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve, and the vector $(m_1, \ldots, m_n)$ of exponent bounds, are system parameters. This algorithm computes exactly m_i isogenies for each ideal $\mathfrak{l}_i$.

```
Require: A supersingular curve $E_A$ over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with
    each $e_i \in [-m_i, m_i]$ and $e_i \equiv m_i \pmod 2$.
Ensure: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.  $(t_1, \ldots, t_n) \leftarrow \left(\frac{\texttt{sign}(e_1)+1}{2}, \ldots, \frac{\texttt{sign}(e_n)+1}{2}\right)$ (Secret)
    $(z_1, \ldots, z_n) \leftarrow (m_1, \ldots, m_n)$ (Not secret)
$E_B \leftarrow E_A$
while some $z_i \neq 0$ do
  $u \leftarrow \texttt{Random}\big(\{2, \ldots, \frac{p-1}{2}\}\big)$
  $(T_1, T_0) \leftarrow \texttt{Elligator}(E_B, u)$        //$T_1 \in E_B[\pi - 1]$ and $T_0 \in E_B[\pi + 1]$
  $(T_0, T_1) \leftarrow ([4]T_0, [4]T_1)$                  //Now $T_0, T_1 \in E_B\left[\prod_i \ell_i\right]$
  for $i \in \{1, \ldots, n\}$ do
    if $z_i \neq 0$ then
      $(G_0, G_1) \leftarrow (T_0, T_1)$
      for $j \in \{i+1, \ldots, n\}$ do
        $(G_0, G_1) \leftarrow ([\ell_j]G_0, [\ell_j]G_1)$
      end for
      if $G_0 \neq \infty$ and $G_1 \neq \infty$ then
        $\texttt{cswap}(G_0, G_1, t_i)$              //Secret kernel point generator: $G_0$
        $\texttt{cswap}(T_0, T_1, t_i)$              //Secret point to be multiplied: $T_1$
        $(E_B, \phi) \leftarrow \texttt{QuotientIsogeny}(E_B, G_0)$
        $(T_0, T_1) \leftarrow \big(\phi(T_0), \phi(T_1)\big)$
        $T_1 \leftarrow [\ell_i]T_1$
        $\texttt{cswap}(T_0, T_1, t_i)$
        $b \leftarrow \texttt{isequal}(e_i, 0)$
        $e_i \leftarrow e_i + (-1)^{t_i}$
        $t_i \leftarrow t_i \oplus b$
        $z_i \leftarrow z_i - 1$
      else if $G_0 \neq \infty$ then
        $T_0 \leftarrow [\ell_i]T_0$
      else if $G_1 \neq \infty$ then
        $T_1 \leftarrow [\ell_i]T_1$
      end if
    end if
  end for
end while
return $B$
```

Since we have restricted exponents to $\{-1, 0, 1\}$, every ℓ_i-isogeny in Algorithm 7 can be computed using only (the images of) the two precomputed points. There is no possibility of failure in the test of Line 12, and no need to sample any other point.

We note that this algorithm still uses dummy operations. If fault-injection attacks are a concern, the exponents can be further restricted to $\{-1, 1\}$, and the group action evaluated as in (a stripped down form of) Algorithm 10. However this further increases the size of p, as n must now be equal to 2λ.

This protection comes at a steep price: at the 128 bits security level, the prime p goes from 511 bits to almost 1500. The resulting field arithmetic would be considerably slower, although the global running time would be slightly offset by the smaller number of isogenies to evaluate.

On the positive side, the resulting system would have much stronger quantum security. Indeed, the best known quantum attacks are exponential in the size of the key space ($\approx 2^{2\lambda}$ here), but only subexponential in p (see [20, 33, 15]). Since our modification more than doubles the size of p without changing the size of the key space, quantum security is automatically increased. For this same reason, for security levels beyond NIST-1 (64 quantum bits of security), the size of p increases more than linearly in λ, and the variant proposed here becomes natural. Finally, parameter sets with a similar imbalance between the size of p and the security parameter λ have already been considered in the context of isogeny based signatures [31], where they provide tight security proofs in the QROM.

Hence, while at the moment this costly modification of CSIDH may seem overkill, we believe further research is necessary to try and bridge the efficiency gap between it and the other side-channel protected implementations of CSIDH.

7.7 Experimental results

Tables 7.1 and 7.2 summarize our experimental results, and compare our algorithms with those of [15], [65], and [81]. Table 7.1 compares algorithms in terms of elementary field operations, while Table 7.2 compares cycle counts of C implementations. All of our experiments were ran on a Intel(R) Core(TM) i7-6700K CPU 4.00GHz machine with 16GB of RAM. Turbo boost was disabled. The software environment was the Ubuntu 16.04 operating system and `gcc` version 5.5.

In all of the algorithms considered here (except the original [15]), the group action is evaluated using the SIMBA method (Splitting Isogeny computations into Multiple BAtches) proposed by Meyer, Campos, and Reith in [65]. Roughly speaking, SIMBA-m-k partitions the set of primes ℓ_i into m disjoint subsets S_i (batches) of approximately the same size. SIMBA-m-k proceeds by computing isogenies for each batch S_i; after k steps, the unreached primes ℓ_i from each batch are merged.

Castryck et al. We used the reference CSIDH implementation made available for download by the authors of [15]. None of our countermeasures or algorithmic improvements were applied.

Meyer–Campos–Reith. We used the software library freely available from the authors of [65]. This software batches isogenies using SIMBA-5-11. The improvements we describe in §7.3 and §7.4 were *not* applied.

Onuki *et. al.* Unfortunately, the source code for the implementation in [81] was not freely available, so direct comparison with our implementation was impossible. Table 7.1 includes their field operation counts for their unmodified algorithm (which, as noted in §7.3, is insecure) using SIMBA-3-8, and our estimates for a repaired version applying our fix in §7.3. We did not apply the optimizations of §7.4 here. (We do not replicate the cycle counts from [81] in Table 7.2, since they may have been obtained using turbo boost, thus rendering any comparison invalid.)

Our implementations. We implemented three constant-time CSIDH algorithms, using the standard primes with the exponent bounds m_i from [81, §5.2].

MCR-style This is essentially our version of Meyer–Campos–Reith (with one torsion point and dummy operations, batching isogenies with SIMBA-5-11), but applying the techniques of §7.3 and §7.4.

OAYT-style This is essentially our version of Onuki *et. al.* (using two torsion points and dummy operations, batching isogenies with SIMBA-3-8), but applying the techniques of §7.3 and §7.4.

No-dummy This is Algorithm 10 (with two torsion points and no dummy operations), batching isogenies using SIMBA-5-11.

In each case, the improvements and optimizations of §7.3-7.4 are applied, including projective Elligator, short differential addition chains, and twisted Edwards arithmetic and isogenies. Our software library is freely available from

`https://github.com/JJChiDguez/csidh`.

The field arithmetic is based on the Meyer–Campos–Reith software library [65]; since the underlying arithmetic is essentially identical, the performance comparisons below reflect differences in the CSIDH algorithms.

Results. We see in Table 7.2 that the techniques we introduced in §7.3 and §7.4 produce substantial savings compared with the implementation of [65]. In particular, our OAYT-style implementation yields a 25% improvement over [65]. Since the implementations use the same underlying field arithmetic library, these improvements are entirely due to the techniques introduced in this thesis. While our no-dummy variant is (unsurprisingly) slower, we see that the performance penalty is not prohibitive: it is less than twice as slow as our fastest dummy-operation algorithm, and only 44% slower than [65].

Table 7.1: Field operation counts for constant-time CSIDH. Counts are given in millions of operations, averaged over 1024 random experiments. The counts for a possible repaired version of [81] are estimates, and hence displayed in italics. The performance ratio uses [65] as a baseline, considers only multiplication and squaring operations, and assumes $M = S$.

Implementation	**CSIDH Algorithm**	**M**	**S**	**A**	**Ratio**
Castryck et al. [15]	unprotected, unmodified	0.252	0.130	0.348	0.26
Meyer–Campos–Reith [65]	unmodified	1.054	0.410	1.053	1.00
Onuki et al. [81]	unmodified	0.733	0.244	0.681	0.67
	repaired as in §7.3	*0.920*	*0.338*	*0.867*	*0.86*
This work	MCR-style	0.901	0.309	0.965	0.83
	OAYT-style	0.802	0.282	0.900	0.74
	No-dummy	1.525	0.526	1.686	1.40

Table 7.2: Clock cycle counts for constant-time CSIDH implementations, averaged over 1024 experiments. The ratio is computed using [65] as baseline implementation.

Implementation	**CSIDH algorithm**	**Mcycles**	**Ratio**
Castryck et al. [15]	unprotected, unmodified	155	0.39
Meyer–Campos–Reith [65]	unmodified	395	1.00
This work	MCR-style	337	0.85
	OAYT-style	300	0.76
	No-dummy	569	1.44

7.8 Conclusion and perspectives

We studied side-channel protected implementations of the isogeny based primitive CSIDH. Previous implementations failed at being constant time because of some subtle mistakes. We fixed those problems, and proposed new improvements, to achieve the most efficient version of CSIDH protected against timing and simple power analysis attacks to date. All of our algorithms were implemented in C, and the source made publicly available online.

We also studied the security of CSIDH in stronger attack scenarios. We proposed a protection against some fault-injection and timing attacks that only comes at a cost of a twofold slowdown. We also sketched an alternative version of CSIDH “for the paranoid”, with much stronger security guarantees, however at the moment this version seems too costly for the security benefits; more work is required to make it competitive with the original definition of CSIDH.

Chapter 8

Extended SIDH

> The two that are one, the one that is all!
>
> Michael Scott, *The Alchemyst*

8.1 Introduction

In this Chapter, a variant of the SIDH protocol that allows us to accelerate Bob's computations on single and multi-core platforms without modifying the formats and lengths of its private/public keys is presented. The SIDH variant proposed in this Chapter is dubbed Extended-SIDH (eSIDH),[1] because of the pair of primes assigned to Bob for performing his isogeny computations. The eSIDH domain parameters are a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, where p is a prime of the form,

[1] Pronounced it spelling out all the letters.

$$p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1. \tag{8.1}$$

Here ℓ_B, ℓ_C are two small prime numbers;[2] f is a cofactor that for efficiency reasons is usually selected as a power of two. Finally, e_A, e_B and e_C are positive integers such that $4^{e_A} \approx \ell_B^{e_B}\ell_C^{e_C}$.

Just as it would happen in SIKE, in eSIDH Alice limits herself to compute degree-4^{e_A} isogenies. This naturally implies that Alice can still take advantage of the cheap cost associated to the fast degree-4 isogeny arithmetic. On the other hand, Bob is now responsible of computing degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies. At first glance it would appear that Bob's task in eSIDH has just become more expensive than what used to be his computational role on a traditional SIDH scheme. Nonetheless, we will show in this Chapter that Bob's eSIDH tasks offer several advantages such as a faster underlying field arithmetic, and novel opportunities for exploiting the parallelism associated to his new computational responsibilities.

Indeed, the rich abundance of the family of primes given in Eq. 8.1, produces for certain instantiations of eSIDH a faster field arithmetic by taking advantage of friendlier Montgomery-friendly primes [12, 38]. Our experimental results show that the computational advantages of eSIDH more than well compensate the extra calculations demanded by this variant. For example, using a single-core SIKE prime p_{751} implementation as a baseline, a comparable eSIDH prime p_{765} instantiation yields an acceleration factor of $1.05, 1.30$ and 1.41, when implemented on $k = \{1, 2, 3\}$-core processors.

As of today, relatively few works have attempted to exploit the rich opportunities that SIDH main computations can offer for parallel computations. In this direction, we are only aware of the works reported in [60, 46], where explicit efforts for parallelizing the computations of the SIDH protocol were attempted and/or exploited. Using a similar approach as the one followed in [60, 46], in this Chapter we report that with respect to a sequential implementation, a two-core and a three-core parallel implementation of the SIDH p_{751} instantiation yields a speedup factor of 1.118 and 1.216, respectively. To our knowledge this work reports the first multi-core implementation of SIDH. In addition when both protocols are implemented on $k = \{1, 2, 3\}$-core processors, eSIDH p_{765} yields an acceleration factor of $1.050, 1.160$ and 1.162 over SIDH.

The remainder of this Chapter is organized as follows. In §8.2 a summary of the SIDH protocol and associated implementations aspects is presented. In §8.3 three different approaches for implementing the eSIDH protocol are presented. In §8.4 several relevant eSIDH implementations aspects on single-core and multi-core processors are discussed. We draw our concluding remarks in §8.5.

[2] In the eSIDH instantiations described in this Chapter we always choose $\ell_B = 3, \ell_C = 5$.

8.2 Preliminaries

In this section, a brief summary of the SIDH protocol and its optimal strategies is given. For more in-deep details see [32, 3] and Chapter 4.

8.2.1 The SIDH protocol

The most popular key exchange SIDH protocol instantiation operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A}3^{e_B} - 1$. The exponents e_A and e_B are typically chosen such that $4^{e_A} \approx 3^{e_B}$. Let us define the constants $r_A = 4^{e_A}$ and $r_B = 3^{e_B}$. The public parameters of SIDH are given by a supersingular base curve E_0, and the basis points $P_A, Q_A, P_B, Q_B \in E_0$, such that $\langle P_A, Q_A\rangle = E_0[r_A]$ and $\langle P_B, Q_B\rangle = E_0[r_B]$.

During the initial *Key Generation* phase, Alice chooses a random integer $m_A \in [1, r_A - 1]$, which acts as her secret key. Thereafter, Alice computes a secret key $R_A = P_A + [m_A]Q_A$ and a degree-4^{e_A} isogeny public curve E_A such that $\phi_A : E_0 \to E_A$ with $\mathrm{Ker}(\phi_A) = \langle R_A\rangle$. Likewise, Bob chooses a secret random integer $m_B \in [1, r_B - 1]$. Then, Bob computes a secret key $R_B = P_B + [m_B]Q_B$ and a degree-3^{e_B} isogeny public curve E_B such that $\phi_B : E_0 \to E_B$ with $\mathrm{Ker}(\phi_B) = \langle R_B\rangle$. These computations complete the *Key Generation* phase.

During the SIDH second phase, known as the *Key Agrement* phase, Alice sends Bob the tuple $[E_A, \phi_A(P_B), \phi_A(Q_B)]$, whereas Bob sends Alice the tuple $[E_B, \phi_B(P_A), \phi_B(Q_A)]$.[3] Alice uses Bob's information to recover the image of her secret key under Bob's curve E_B, as $\phi_B(R_A) = \phi_B(P_A) + [m_A]\phi_B(Q_A)$. Then Alice computes the curve E_{BA} such that there is a degree-4^{e_A} isogeny $\phi_{BA} : E_B \to E_{BA}$ with $\mathrm{Ker}(\phi_{BA}) = \langle\phi_B(R_A)\rangle$. Similarly, Bob's recovers the image of his secret key under Alice's curve E_A by computing $\phi_A(R_B) = \phi_A(P_B) + [m_B]\phi_A(Q_B)$. Bob then computes the isogenous curve E_{AB} such that there is a degree-3^{e_B} isogeny $\phi_{AB} : E_A \to E_{AB}$ with $\mathrm{Ker}(\phi_{AB}) = \langle\phi_A(R_B)\rangle$. This ends the SIDH protocol. Alice and Bob can now create a shared secret by computing the j-invariant of their respective curves, using the fact that $E_{BA} \cong E_{AB}$ implies $j(E_{BA}) = j(E_{AB})$.

Remark 20. The most prominent SIDH computational tasks include the computation of large degree isogenies and the evaluation of elliptic curve points in those isogenies. Another large operation of this scheme is the computation of four three-point scalar multiplications. For a typical software or hardware implementation of SIDH, the isogeny computations and associated point evaluations on one hand, along with the three-point scalar multiplications on the other hand, may take 70-80% and 20-30% of the overall protocol's computational cost, respectively.

Remark 21. In order to compute the points $R_A, \phi_B(R_A)$ (resp. $R_B, \phi_A(R_B)$), Alice (resp. Bob) must perform two three-point scalar multiplication procedures using a right-to-left

[3] State-of-the-art SIDH implementations use differential point arithmetic on Montgomery curves. Consequently, Alice and Bob evaluate and transmit three points each, namely, $x(P_A), x(Q_A), x(P_A - Q_A)$; and $x(P_B), x(Q_B)$, and $x(P_B - Q_B)$, respectively [26].

Montgomery ladder algorithm [49, 38]. This kind of Montgomery ladder has a per-step cost of one point addition (xADD) and one point doubling (xDBL), which are usually performed in the projective space $\mathbb{P}^1$. Noticing that for current state-of-the-art SIDH implementations the costs of xDBL and xADD are about the same, one can assume that the per-step computational cost of the three-point Montgomery ladder is essentially that of two xDBL operations. It follows that the cost of computing R_A or $\phi_B(R_A)$ (resp. R_B or $\phi_A(R_B)$) is of $4e_A$ (resp. $2\log_2(3)e_B$) xDBL operations.

8.3 The extended SIDH (eSIDH) Protocol

The extended SIDH (eSIDH) Protocol operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C} - 1$. The exponents e_A, e_B and e_C are chosen so that $4^{e_A} \approx \ell_B^{e_B}\ell_B^{e_C}$. The eSIDH protocol flow is quite similar to the one of a traditional SIDH as described in §8.2.1. Alice must still compute degree-4^{e_A} isogenies, but now Bob is responsible for computing degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies.

In this section, three different approaches for computing the eSIDH protocol are presented. We start in §8.3.1 with the description of a simple naive eSIDH approach that is relatively expensive and offers little opportunities for exploiting parallelism. In §8.3.2, an eSIDH approach especially designed for exploiting parallelism opportunities is presented.

Table 8.1 shows the estimated scalar multiplication expenses incurred by SIDH and the two eSIDH instantiations discussed in this section. All the costs are given in number of xDBL operations.[4]

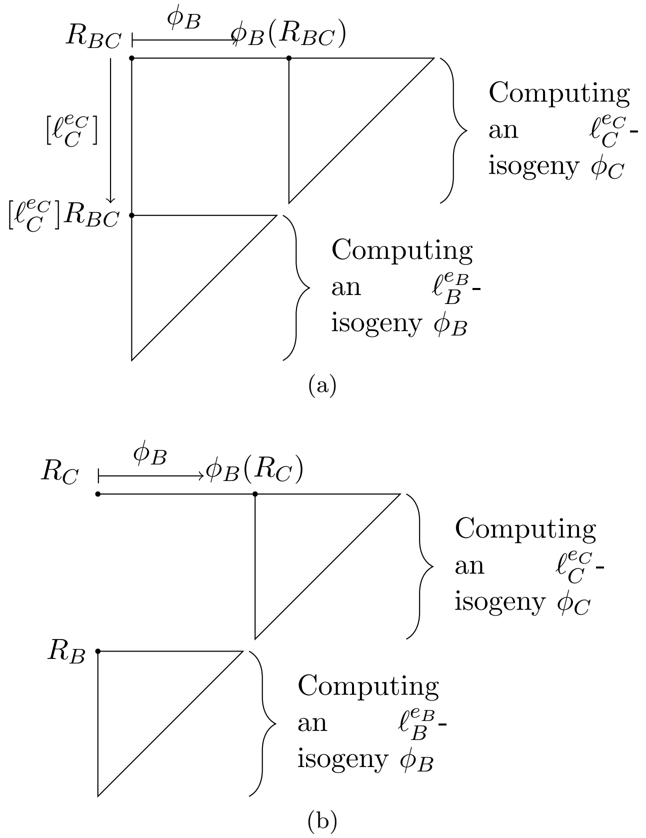
In the case of two-core implementations, the parallel eSIDH described in §8.3.2, is significantly faster than the SIDH implementation of [3] and any other eSIDH instantiation discussed here.

8.3.1 A naive approach for computing eSIDH

Mimicking his role in SIDH, in a naive eSIDH instantiation Bob can first choose a basis for $\langle P_{BC}, Q_{BC}\rangle = E[\ell_B^{e_B} \cdot \ell_C^{e_C}]$. Thereafter, Bob computes his secret point as $R_{BC} = P_{BC} + [m_{BC}]Q_{BC}$ followed by the computation of a degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogeny using an optimal strategy *à la* SIDH as shown in Figure 8.1a.

Alice's eSIDH computational expenses are exactly the same as in SIDH. In the case of Bob, we stress that the computational expense of computing his eSIDH secret point R_{BC} as defined above, is about the same of computing Bob's SIDH secret point R_B as given in §8.2.1.

[4] We do not account for isogeny computations, because the computational cost associated to this task is about the same for all the two variants of eSIDH and the SIDH protocol.


R_{BC}
ϕ_B
$\phi_B(R_{BC})$
$[\ell_C^{e_C}]$
$[\ell_C^{e_C}]R_{BC}$
Computing an $\ell_C^{e_C}$-isogeny ϕ_C
Computing an $\ell_B^{e_B}$-isogeny ϕ_B
R_C
ϕ_B
$\phi_B(R_C)$
R_B
Computing an $\ell_C^{e_C}$-isogeny ϕ_C
Computing an $\ell_B^{e_B}$-isogeny ϕ_B

(a)

(b)

Figure 8.1: Overview of an strategy for computing a degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogeny. Each isogeny ϕ_B and ϕ_C can be computed using a traditional SIDH strategy as in [32]. The kernel of ϕ_B is the subgroup $\langle[\ell_C^{e_C}]R_{BC}\rangle$, and the kernel of ϕ_C is the subgroup $\langle\phi_B(R_{BC})\rangle$. Figure 8.1a shows a naive way for computing the $\ell_B^{e_B}\ell_C^{e_C}$-isogeny $\phi_{BC} = \phi_C \circ \phi_B$. Figure 8.1b shows a parallel-oriented approach for computing such strategy.

Protocol	**Single Core processor required number of xDBL operations**	**Two-Core processor required number of xDBL operations**
SIDH [32]	$\frac{16\lambda}{4}$	$\frac{16\lambda}{4}$
Naive §8.3.1	$\frac{16\lambda}{4}$	$\frac{16\lambda}{4}$
Parallel §8.3.2	$\frac{16\lambda}{4}$	$\frac{11\lambda}{4}$
CRT-based §8.3.3	$\frac{15\lambda}{4}$	$\frac{13\lambda}{4}$

Table 8.1: Let $\lambda = \lceil \log_2(p) \rceil$ be the bit-length of the eSIDH prime p. This table reports the approximate number of xDBL operations processed by the SIDH protocol of [32] compared against the three eSIDH variants discussed in this section (for the experimental clock cycle cost of xDBL see Table 8.3).

Figure 8.1a depicts an optimal strategy procedure for computing Bob's degree-$\ell_B^{e_B}\ell_C^{e_C}$. The computational cost of this isogeny is of about $\frac{e_B}{2}\log_2 e_B$, $\frac{e_C}{2}\log_2 e_C$ scalar multiplications by ℓ_B and ℓ_C, $\frac{e_B}{2}\log_2 e_B$ degree-ℓ_B and $\frac{e_C}{2}\log_2 e_C$ degree-ℓ_C isogeny evaluations, and e_B and e_C constructions of degree-ℓ_B and degree-ℓ_C isogenous curves, respectively. This computational expense is nearly the same as the one required by Alice for computing a degree-4^{e_A} isogeny, using the optimal strategies described in §4.3.2 and Figure 4.4.

There seems to be no obvious way of parallelizing the main computation of this naive eSIDH instantiation. In the following two subsections, two eSIDH instantiations more amenable for parallelization are described.

8.3.2 A parallel approach for computing eSIDH

As mentioned before, eSIDH offers rich opportunities for exploiting its inherent parallelism. In this subsection an eSIDH instantiation specifically designed for the concurrent computation of this protocol's scalar multiplication operations will be presented.

As before, let $\lambda = \lceil \log_2(p) \rceil$ be the bit-length of the eSIDH prime $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C} - 1$. For the sake of compactness let us define $r_B = \ell_B^{e_B}$ and $r_B = \ell_C^{e_C}$. Rather than defining Bob's secret point R_{BC} as in the previous subsection, Bob has now two secret points that he can calculate by choosing two pairs of bases such that $\langle P_B, Q_B \rangle = E[r_B]$ and $\langle P_C, Q_C \rangle = E[r_C]$. Afterwards, Bob randomly chooses two integers $m_B \in [1, r_B - 1]$ and $m_C \in [1, r_C - 1]$ to compute his secret points as,

$$R_B = P_B + [m_B]Q_B; \qquad\qquad R_C = P_C + [m_C]Q_C. \tag{8.2}$$

Now, by picking ℓ_B, ℓ_C, e_B and e_C such that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C$, it follows that the cost of computing R_B is of about $\frac{2\lambda}{4}$ xDBL operations (cf. remark 21), which is nearly the same cost of computing R_C, and about half of the cost of computing Alice's secret point R_A. Furthermore, the calculations of Bob's secret points R_B and R_C are fully independent. Therefore, one can compute them in parallel on multi-core platforms. Moreover, the isogeny $\phi_{BC} = \phi_C \circ \phi_B$ can now be determined without performing the multiplication by r_C depicted in Figure 8.1a. This computational saving comes from the facts that $gcd(r_B, r_C) = 1$ and that R_B, R_C are points of order r_B and r_C, respectively. Hence as shown in Figure 8.1b, R_B and $\phi_B(R_C)$ can serve to generate the kernels of the isogenies ϕ_B and ϕ_C, respectively. This observation yields a significant saving of about $\frac{\lambda}{4}$ xDBL operations.

Reducing the public-key size of the parallel instantiation of eSIDH

Seemingly, an important drawback of using two secret points for Bob is that in the *Key Agrement* phase, this design decision forces Bob to know the images of his public points P_B, Q_B, P_C and Q_C, all of them evaluated under Alice's degree-4^{e_A} isogeny ϕ_A. Sending these four points implies an increment on the data to be transfered from Alice to Bob. This in turn implies an increment on Alice's computational load since now, she would need to find the isogeny images of four points (instead of two as in the original SIDH).[5]

Alternatively, one can reduce the eSIDH public-key size at the same time that Alice's extra work is prevented. This can be done by defining two auxiliary public points that while codifying Bob's public points P_B, Q_B, P_C and Q_C, provide an efficient way to recover them. Let us re-define Bob's public points as $S = P_B + P_C$ and $T = Q_B + Q_C$. This implies that,

$$\begin{aligned} [r_B]S = [r_B]P_C, &\quad [r_C]S = [r_C]P_B, \\ [r_B]T = [r_B]Q_C, &\quad \text{and} \quad [r_C]T = [r_C]Q_B. \end{aligned} \tag{8.3}$$

Hence, given the points S, T, one can recover multiples of Bob's original four public points by performing four scalar multiplications. Notice that all four of these scalar multiplications are fully independent. Nonetheless, we can do better as discussed below.

Remark 22. From the multiples $[r_C]P_B$ and $[r_C]Q_B$, one can recover the points P_B, Q_B, by multiplying them by the scalars $r_C^{-1} \bmod r_B$ and $r_B^{-1} \bmod r_C$, respectively. However, it is easier to directly use $[r_C]P_B$ and $[r_C]Q_B$ to generate the point $R'_B = [r_C]P_B + [m_B]([r_C]Q_B)$. Provided that $gcd(r_C, r_B) = 1$, it follows that $R'_B = [r_C]R_B$. Thus, $\langle R'_B \rangle = \langle R_B \rangle$, which implies that the degree-r_C isogenies with kernels $\langle R'_B \rangle$ and $\langle R_B \rangle$, are one and the same. Similarly, the point $R'_C = [r_B]P_C + [m_C]([r_B]Q_C)$, is sufficient to generate the degree-r_B isogeny with kernel $\langle R_C \rangle$.

The observation stated in Remark 22 along with the relations given in Eq. (8.3.2) suggest an approach where Bob can efficiently recover the points R'_B, R'_C, by the direct

[5] In practice one uses differential point arithmetic on Montgomery curves. Hence, Alice would need to evaluate and transmit six points, namely, $x(P_B), x(Q_B), x(P_B - Q_B), x(P_C), x(Q_C)$, and $x(P_B - Q_B)$.

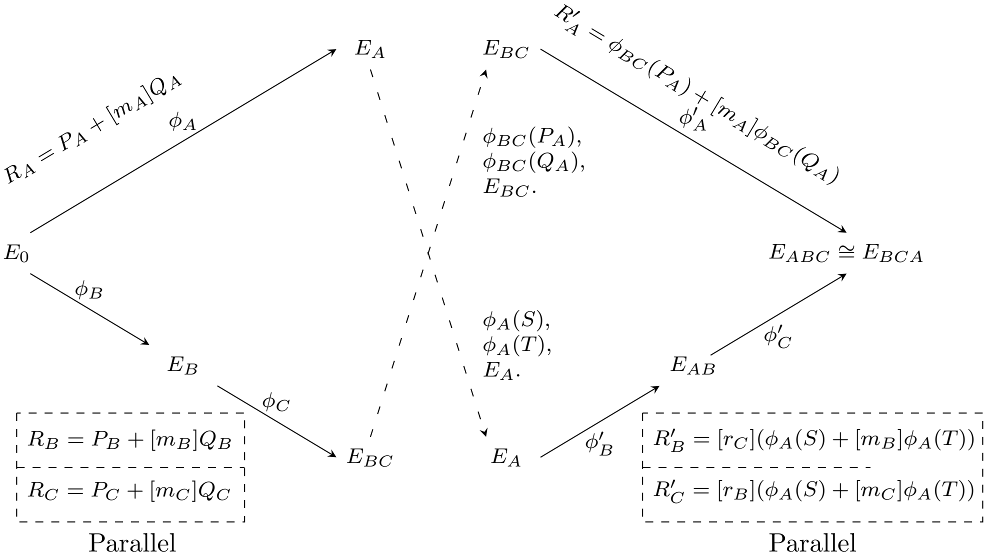


Figure 8.2: Overview of an eSIDH parallel instantiation with Bob's secret points computed in parallel. In the *Key Generation* phase $\text{Ker}(\phi_B) = \langle R_B \rangle$ and $\text{Ker}(\phi_C) = \langle \phi_B(R_C) \rangle$. In the *Key Agrement* phase $\text{Ker}(\phi'_B) = \langle R'_B \rangle$ and $\text{Ker}(\phi'_C) = \langle \phi'_B(R'_C) \rangle$

computation of,

$$R'_B = [r_C](S + [m_B]T) \qquad \text{and} \qquad R'_C = [r_B](S + [m_C]T). \tag{8.4}$$

Remark 23. Eq. (8.4) is useful during the eSIDH *Key Agrement* phase. For the eSIDH *Key Generation* phase, it results more efficient to compute the points R_B and R_C as discussed at the beginning of Subsection 8.3.2.

Figure 8.2 shows a general overview of the eSIDH parallel instantiation described in this subsection. Assuming that a multi-core platform is available for the execution of this eSIDH instantiation, most Bob's scalar multiplications can be computed in parallel.

Remark 24. **eSIDH security**: Recall that $gcd(r_B, r_C) = 1$ and $r_A \approx \log_2(\ell_B)r_B \cdot \log_2(\ell_C)r_C$. Given the points S and T, computing a degree-r_Br_C isogeny between E_0 and E_{BC} should have the same computational complexity as the problem of, given the points P_A and Q_A, finding a degree-r_A isogeny between E_0 and E_A. Furthermore, provided that $4^{e_A} \approx \ell_B^{e_B} \cdot \ell_C^{e_C}$, the heuristic polynomial time key recovery attacks presented in [83] do not appear to apply against eSIDH.

Computational cost of the eSIDH parallel instantiation

As in Table 8.1, the eSIDH required number of xDBL operations will be used as cost metric. We further assume that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C \approx \frac{r_A}{2} \approx \frac{\lambda}{4}$.

Note that the private/public key sizes of eSIDH are the same as the traditional SIDH protocol of [32]. Moreover, Alice's isogeny computations are exactly the same for both protocols. Nevertheless, Bob can compute his two degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies using the computational trick shown in Figure 8.1b. This approach yields a saving of about $\frac{2\lambda}{4}$ xDBL operations compared against the performance cost required by the SIDH strategy shown in Figure 4.4, without incurring in any extra computational overhead.

The scalar multiplications computational expenses of the parallel eSIDH variant are dispensed as discussed next. As in the traditional SIDH, Alice must perform two $\frac{2\lambda}{4}$-bit scalar multiplications that involve the computation of about $\frac{8\lambda}{4}$ xDBL operations (cf. Remark 21). Moreover,

during the *Key Generation* phase, Bob computes the points R_B and R_C, by performing $\frac{4\lambda}{4}$ and $\frac{2\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively. During the *Key Agrement* phase, Bob computes the points R'_B, R'_C, by performing $\frac{6\lambda}{4}$ and $\frac{3\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively.

Thus, the eSIDH combined scalar multiplication effort of Alice and Bob for a a single-core and two-core implementation is of $\frac{16\lambda}{4}$ and $\frac{11\lambda}{4}$, respectively (see Table 8.1).

8.3.3 A CRT-based approach for computing eSIDH

Another instantiation of eSIDH can be constructed by taking advantage of the Chinese Remainder Theorem (CRT). As in §8.3.2, let $\lambda = \lceil \log_2(p) \rceil$ be the bit-length of the eSIDH prime $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C} - 1$. For the sake of compactness let us define $r_B = \ell_B^{e_B}$ and $r_B = \ell_C^{e_C}$. A CRT-based approach for eSIDH can be computed as explained in the remainder of this subsection.

First choose a pair of random integers under the following restrictions. Pick randomly $m_B \in [1, r_B]$ and $m_C \in [1, r_C]$ such that, $gcd(m_B, r_C) = gcd(m_C, r_B) = 1$. Then compute the following integers,

$$\begin{aligned} \hat{m}_B &= m_B^{-1} \bmod r_C; & \hat{m}_C &= m_C^{-1} \bmod r_B; \\ \bar{m}_B &= m_B \cdot \hat{m}_B \bmod r_B; & \bar{m}_C &= m_C \cdot \hat{m}_C \bmod r_C; \\ & m_{BC} = m_B \cdot \hat{m}_B \cdot m_C \cdot \hat{m}_C \bmod (r_B \cdot r_C). \end{aligned} \tag{8.5}$$

From Eq. (8.5) it follows that $m_{BC} \equiv \bar{m}_B \mod r_B$ and $m_{BC} \equiv \bar{m}_C \mod r_C$.

For the execution of the eSIDH *Key Generation* phase the following two points are computed, $R_B = P_B + [\bar{m}_B]Q_B$ and $R_C = P_C + [\bar{m}_C]Q_C$. Thereafter, one can compute ϕ_{BC} as shown in Figure 8.1b, such that the kernel of ϕ_B is generated by R_B and the kernel of ϕ_C is generated by $\phi_B(R_C)$. Since $|m_B| \approx |m_C| \approx \frac{|m_A|}{2} = \frac{\lambda}{4}$,[6] the combined cost of computing R_B and R_C is about the same as the cost of computing R_A. As a side effect, note that these computations imply a saving of $r_C \approx \frac{\lambda}{4}$ xDBL operations corresponding to the left most vertical edge between the points R_C and R_B shown in Figure 8.1b.

[6] The operator $|\cdot|$ evaluates the bit-length of its operand.

For the computation of the eSIDH *Key Agrement* phase as in §8.3.2, let us define the auxiliary public points $S = P_B + P_C$ and $T = Q_B + Q_C$. It turns out that the generators of the subgroups $\langle R_B\rangle$ and $\langle R_C\rangle$ can be recovered by invoking the CRT and Remark 22 applied on the integers given in Eq. (8.5).

Proposition 8.3.1. Let P_B, Q_B, P_C, Q_C, $\bar{m}_B$, $\bar{m}_C$, m_{BC}, R_B as R_C be defined as before, and fix $S = P_B + P_C$ and $T = Q_B + Q_C$. Then $[r_C]R_B = [r_C](S + [m_{BC}]T)$ and $[r_B]R_C = [r_B](S + [m_{BC}]T)$.

Proof. By straightforward substitution we get,

$$\begin{aligned}
[r_C](S + [m_{BC}]T) =&[r_C](P_B + P_C + [m_{BC}]Q_B + [m_{BC}]Q_C))\\
=&[r_C](P_B + [m_{BC}]Q_B)\\
=&[r_C](P_B + [m_{BC} \mod r_B]Q_B)\\
=&[r_C](P_B + [\bar{m}_B]Q_B)\\
=&[r_C]R_B.
\end{aligned}$$

Using an analogous procedure one can show that $[r_B]R_C = [r_B](S + [m_{BC}]T)$. □

Using Proposition 8.3.1, one can recover the generator R'_B of the subgroup $\mathrm{Ker}(\phi'_B)$ and $\phi'_B(R'_C)$, the generator of the subgroup $\mathrm{Ker}(\phi'_C)$. To this end, one can compute,

$$\begin{aligned}
R'_B &= [r_C](\phi_A(S) + [m_{BC}]\phi_A(T)) = \phi_A([r_C]R_B);\\
R'_C &= [r_B](\phi_A(S) + [m_{BC}]\phi(T)) \quad = \phi_A([r_B]R_C).
\end{aligned}$$

Nevertheless, these computations have a steep cost of $\frac{10\lambda}{4}$ xDBL operations. Fortunately, there is an efficient way to reduce this expense.

Proposition 8.3.2. Fix $R'_B = [r_C](\phi_A(S) + [m_{BC}]\phi_A(T)) = [r_C]R'$. The point $\phi'_B(R')$ has order r_C and $\phi'_B(R') = \phi'_B((\phi_A(R_C))$.

Proof. By virtue of Proposition 8.3.1, the order-r_B point R_B generates the kernel of the degree-r_B isogeny ϕ'_B, that is, $\mathrm{Ker}(\phi'_B) = \langle R'_B\rangle$. By straightforward substitution we get,

$$\begin{aligned}
R' =&\phi_A(S + [m_{BC}]T)\\
=&\phi_A(P_B + [m_{BC}]Q_B + P_C + [m_{BC}]Q_C)\\
=&\phi_A(R_B + R_C).
\end{aligned}$$

It follows that

$$\begin{aligned}
\phi'_B(R') =&\phi'_B((\phi_A(R_B + R_C))\\
=&\phi'_B((\phi_A(R_B) + \phi_A(R_C))\\
=&\phi'_B((\phi_A(R_C)),
\end{aligned}$$

which yields an order-r_C point. □

Note that the points R'_B and $\phi'_B(R')$ can serve as the kernel generators of Bob's key-agreement phase isogenies ϕ'_B and ϕ'_C, respectively. Moreover, the cost of computing those two points is of about $\frac{5\lambda}{4}$ xDBL operations. There seems to be no obvious way to parallelize these two calculations.

Computational cost of the CRT-based eSIDH instantiation

As in §8.3.2, the eSIDH required number of xDBL operations will be used as cost metric, and we will assume that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C \approx \frac{r_A}{2} \approx \frac{\lambda}{4}$. Also, as argued in §8.3.2, the private/public key sizes of eSIDH and Alice's isogeny computations are exactly the same as in SIDH. Bob can compute his two degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies using the computational trick shown in Figure 8.1b, obtaining a saving of about $\frac{2\lambda}{4}$ xDBL operations compared against SIDH.

The scalar multiplications computational expenses of the CRT-based eSIDH variant are dispensed as discussed next. Let us consider the eSIDH instantiation depicted in Figure 8.2. Then, as in the traditional SIDH, Alice must perform two $\frac{2\lambda}{4}$-bit scalar multiplications that involve the computation of about $\frac{8\lambda}{4}$ xDBL operations (cf. Remark 21).

During the *Key Generation* phase, Bob computes the points R_B, R_C, by performing $\frac{4\lambda}{4}$ and $\frac{2\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively. During the *Key Agrement* phase, Bob computes the points R', R'_B, by performing $\frac{5\lambda}{4}$ xDBL operations for either a single-core or a two-core implementation.

Thus, the eSIDH combined scalar multiplication effort of Alice and Bob for a single-core and a two-core implementation is of $\frac{15\lambda}{4}$ and $\frac{13\lambda}{4}$, respectively (see Table 8.1).

8.4 Parameter selection and implementation aspects

8.4.1 The hunting for efficient eSIDH Primes

Let $N = \lceil \lceil \log_2(p) \rceil / w \rceil$ be the minimum number of 64-bit words needed to represent an eSIDH prime p. In this Chapter it is assumed $w = 64$. We say that a modulus p is γ-Montgomery-friendly if $p \equiv \pm 1 \mod 2^{\gamma \cdot w}$ for a positive integer γ [43, 52]. This property implies that $-p^{-1} \equiv \mp 1 \mod 2^{\gamma \cdot w}$, which is conveniently exploited to produce savings in the Montgomery's REDC reduction algorithm [12].

SIKE uses primes of the form $p := 4^{e_A}3^{e_B} - 1$. There are at least two computer arithmetic reasons for this choice. One of them, is that this family of primes are Montgomery-friendly, which implies that they admit fast Montgomery Reduction [38, 12]. The second advantage is that there exist highly efficient formulas for computing degree-3 and degree-4 isogenies [23, 18]. The eSIDH primes proposed in this Chapter are of the form $p := 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1$, which are much more flexible and abundant than the SIKE primes. Then, given some fixed values for N and the primes ℓ_B and ℓ_C, one searches for $\frac{N}{2}$-Montgomery-friendly primes (if they exist) by varying e_B, e_C and f. These friendlier

eSIDH primes proposed here	N	γ	**SIKE primes as in [3]**	N	γ
$p_{434} = 2^{218}3^{70}5^{45} - 1$	7	3	$p_{434} = 2^{216}3^{137} - 1$	7	3
$p_{507} = 2^{256}3^{79}5^{54} - 1$	8	4	$p_{503} = 2^{250}3^{159} - 1$	8	3
$p_{632} = 2^{321}3^{96}5^{67}7 - 1$	10	5	$p_{610} = 2^{305}3^{192} - 1$	10	4
$p_{765} = 2^{391}3^{119}5^{81} - 1$	12	6	$p_{751} = 2^{372}3^{239} - 1$	12	5

Table 8.2: Our selection of eSIDH primes matching the four security levels offered by the SIKE primes included in [3], where $N = \lceil \lceil \log_2(p) \rceil / 64 \rceil$, and γ is the largest integer for that N such that $p \equiv -1 \mod 2^{\gamma \cdot 64}$ holds.

Montgomery-friendly primes achieve a faster Montgomery reduction (see [38, Algorithm 6]) than the ones that could possibly be obtained from comparable SIKE primes.

Another important design aspect to be considered is that on Bob's side, there exists a trade-off between the size of the base-primes ℓ_B and ℓ_C and their corresponding exponents e_B and e_C, respectively. The base-primes define the *size of the step*, whereas their exponents determine how many steps one must perform for isogeny evaluations and constructions. Depending on the exact choice of these parameters, one can make a few big steps or many small steps. Furthermore as discussed in §8.3.2, in order to take full advantage of parallel computing and also for security reasons (cf. Remark 24), it is important to choose $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C$.

For all the eSIDH instances considered in this Chapter, we use primes of the form $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1$, such that $2e_A \approx \log_2(\ell_B^{e_B}\ell_C^{e_C})$, and where e_A is chosen so that the security level offered by the SIKE primes as specified in [3] is matched (see also [1]). The cofactor $f = 2^k c$ is carefully selected so that p qualifies as an $\frac{N}{2}$-Montgomery-friendly prime (if at all possible). Table 8.2 shows our selection of four eSIDH primes matching the four security levels specified in [3]. When searching for eSIDH primes with comparable security as the one offered by the p_{434} SIKE prime, the best choice that we were able to find is eSIDH-p_{434} as specified in Table 8.2. Both of them, SIKE-p_{434} and eSIDHp_{434}, fit in seven 64-bit words and they are 3-Montgomery-friendly primes. This implies that the field arithmetic costs associated to SIKE-p_{434} and eSIDH-p_{434} are fairly similar (cf. Table 8.3). Luckily, for the other three security levels we managed to find eSIDH $\frac{N}{2}$-Montgomery-friendly primes sharing the same security level as their SIKE prime counterparts.

8.4.2 Results and discussion

In this subsection, a full implementation of the eSIDH protocol proposed in this work is presented. We mainly focus ourselves on the eSIDH parallel instantiation discussed in §8.3.2, and we use the SIDH implementation of [3] as a baseline to compare the acceleration factor achieved by the eSIDH scheme. Building on the techniques proposed in [46], we also report a multi-core implementation of the SIDH protocol. To the best of our knowledge this is the

Operation	p_{434}	p_{751}	p_{765}	AF
Reduction $\mathbb{F}_p$	78	154	137	1.12
Mult $\mathbb{F}_{p^2}$	466	1,029	977	1.05
Sqr $\mathbb{F}_{p^2}$	349	780	716	1.08
Inv $\mathbb{F}_{p^2}$	77,764/77025 (*)	317,655	251,366	1.26
Doubling	2,961	6,186	5,845	1.12
4-IsoGen	1,793	3,691	3,442	1.07
4-IsoEval	3,955	8,407	7,972	1.05
Tripling	5,595	11,999	11,292	1.06
3-IsoGen	2,850	5,720	5,418	1.05
3-IsoEval	2,717	5,944	5,612	1.05
Quintupling	7,995	-	16,285	-
5-IsoGen	7,951	-	16,179	-
5-IsoEval	4,703	-	9,682	-

Table 8.3: Timing performance of selected quadratic field arithmetic operations and isogeny evaluations and constructions. Timings are reported in clock cycles measured on a Skylake processor at 4.0GHz. (*) In this cell the left number indicates the cost of quadratic-field inversion for SIKE-p_{434} whereas the right number indicates the cost of quadratic-field inversion for P434eSIDH-p_{434}. Right most column shows the Acceleration factor when comparing p_{751} versus p_{765}

Phase	SIKE-p_{434}			eSIDH-p_{434}		
	Cores number			Cores number		
	1	2	3	1	2	3
Alice Key Generation	5.93	5.62	5.36	5.88	5.60	5.36
Bob Key Generation	6.54	6.20	5.88	6.56	4.89	4.43
Alice Key Agrement	4.80	4.49	4.22	4.75	4.48	4.20
Bob Key Agrement	5.50	5.14	4.82	6.16	4.73	4.47
Total	22.77	21.45	20.28	23.35	19.70	18.46

Table 8.4: Performance comparison of the SIKE prime p_{434} against the eSIDH prime p_{434}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

Phase	SIKE-p_{434}			eSIDH-p_{434}		
	Cores number			Cores number		
	1	2	3	1	2	3
Key Generation	6.54	6.16	5.87	6.63	4.92	4.42
Encapsulation	10.75	10.12	9.54	10.77	10.09	9.56
Decapsulation	11.44	10.75	10.14	12.19	10.34	9.83
Total	28.73	27.03	25.55	29.59	25.35	23.81

Table 8.5: Performance comparison of SIKE protocol comparing the SIKE prime p_{434} against the eSIDH prime p_{434}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

first reported software implementation of SIDH.[7] Our two case studies targeted p_{434} and p_{751}, the smallest and largest SIKE primes that are included in the SIKE specification [3].

All the timings were measured using an Intel core i7-6700K processor with micro-architecture Skylake at 4.0 GHz. Using the Clang-3.9 compiler and the flags `-Ofast -fwrapv -fomit-frame-pointer -march=native -madx -mbmi2`.

Our software library is freely available from `https://github.com/dcervantesv/eSIDH`

Quadratic field arithmetic and isogeny computations

Table 8.3 presents a comparison of the field arithmetic costs associated to the SIKE primes p_{434} and p_{751} against the ones exhibit by the eSIDH primes p_{434} and p_{765}, respectively. Note that our eSIDH prime p_{765} field arithmetic gets noticeable timing speedups compared against the SIKE p_{751} field arithmetic. This acceleration is justified from the fact that since p_{765} is a friendlier Montgomery-friendly prime, it has a faster modular reduction than p_{751}.

[7] A reconfigurable hardware parallel version of SIDH was previously reported in [60].

Phase	p_{751}			p_{765}		
	Cores number			Cores number		
	1	2	3	1	2	3
Alice Key Generation	23.59	21.74	19.88	22.27	20.19	18.89
Bob Key Generation	26.74	23.71	22.24	24.34	17.76	15.79
Alice Key Agrement	19.37	17.49	15.64	18.21	16.12	14.83
Bob Key Agrement	22.76	19.74	18.25	23.24	17.16	15.94
Total	92.46	82.67	76.01	88.05	71.23	65.42

Table 8.6: Performance comparison of the SIKE prime p_{751} against the eSIDH prime p_{765}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

Phase	SIKE-p_{751}			eSIDH-p_{765}		
	Cores number			Cores number		
	1	2	3	1	2	3
Key Generation	26.66	23.52	22.17	24.20	17.83	15.77
Encapsulation	42.98	37.99	36.25	40.29	36.23	33.65
Decapsulation	46.35	40.96	38.05	44.92	37.16	34.70
Total	115.99	105.61	96.47	109.35	91.22	84.12

Table 8.7: Performance comparison of SIKE protocol comparing the SIKE prime p_{751} against the eSIDH prime p_{765}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

Parallelizing the SIDH protocol

Using a similar approach as the one followed in [60, 46], in this work we parallelize the SIDH implementation of [3] as follows. Alice and Bob isogeny evaluations and constructions were computed using the optimal strategy of [32]. Optimal strategies typically produce an average of four points per curve whose isogeny images can be processed concurrently [60]. Hence, our two- and three-core implementations actively strove for concurrently performing as many isogeny evaluations as possible.[8]

Table 8.4 shows that with respect to a sequential implementation, a two-core and a three-core parallel implementation of the SIDH p_{434} instantiation yields a speedup factor of 1.062 and 1.123, respectively. Likewise, Table 8.6 reports that with respect to a sequential implementation, a two-core and a three-core parallel implementation of the SIDH p_{751} instantiation yields a speedup factor of 1.118 and 1.216, respectively.

[8] Parallel canonical strategies for SIDH are studied and proposed in [46].

Performance evaluation of the eSIDH parallel instantiation

Table 8.4 reports the performance timing achieved by the eSIDH p_{434} parallel instantiation. Using a single-core SIDH p_{434} implementation as a baseline, it can be seen from Table 8.4 that a parallel eSIDH p_{434} implementation yields an acceleration factor of $0.97, 1.15$ and 1.23, when executed on $k = \{1, 2, 3\}$-core processors. On the other hand, eSIDH p_{434} yields an acceleration factor of $0.971, 1.073$ and 1.086, when both protocols are implemented on $k = \{1, 2, 3\}$-core processors. Hence for a single-core implementation, eSIDH p_{434} is slower than its SIDH p_{434} counterpart.

Using a single-core SIDH p_{434} implementation as a baseline, it can be seen from Table 8.5 that a parallel eSIDH p_{434} implementation yields an acceleration factor of $0.97, 1.13$ and 1.20, when executed on $k = \{1, 2, 3\}$-core processors. On the other hand, eSIDH p_{434} yields an acceleration factor of $0.97, 1.06$ and 1.07, when both protocols are implemented on $k = \{1, 2, 3\}$-core processors.

Table 8.6 reports the performance timing achieved by the eSIDH p_{765} parallel instantiation. Using a single-core SIDH p_{751} implementation as a baseline, It can be seen that a parallel implementation of eSIDH p_{765} yields an acceleration factor of $1.05, 1.29$ and 1.41, when executed on $k = \{1, 2, 3\}$-core processors. Furthermore, eSIDH p_{765} yields an acceleration factor of $1.050, 1.160$ and 1.161. when both protocols are implemented on $k = \{1, 2, 3\}$-core processors. Table 8.7 reports the performance timing achieved by the eSIDH p_{765} parallel instantiation. Using a single-core SIKE p_{751} implementation as a baseline, it can be seen that a parallel implementation of SIKE using eSIDH p_{765} prime, yields an acceleration factor of $1.06, 1.27$ and 1.37, when executed on $k = \{1, 2, 3\}$-core processors. Furthermore, eSIDH p_{765} yields an acceleration factor of $1.06, 1.15$ and 1.14. when both protocols are implemented on $k = \{1, 2, 3\}$-core processors. We stress that even for a single-core implementation of this case study, our eSIDH variant produces a modest but noticeable speedup of about 5% in SIKE protocol and %6 in the SIDH protocol.

As a general summary we note that for single-core implementations, Bob's $3^{e_3}5^{e_5}$ isogeny computation has no overhead impact on the *Key Generation* phase. However, the public key recovery mechanism (cf. §8.3.2) proves to be relatively expensive on the *Key Agrement* phase. On the other hand, for two- and three-core implementations, our eSIDH instantiation clearly outperforms SIDH on all Bob's computations. In Table 8.6, the comparison of SIDH p_{751} against eSIDH p_{765} reveals the superiority of the latter over the former in all the phases of the protocol, even for a sequential implementation. The one exception being Bob's *Key Agrement* phase. For the two- and three- core implementations, Bob's *Key Agrement* for eSIDH p_{765} is even faster than Alice's *Key Agrement* for SIDH p_{751}.

Performance evaluation of the CRT-based eSIDH instantiation

A shown in Table 8.1, the CRT-based eSIDH instantiation presented in §8.3.3 offers less parallelism opportunities than the ones enjoyed by the eSIDH parallel instantiation discussed in 8.3.2. However, according to the estimates given in Table 8.1, the CRT-based eSIDH instantiation is a promising economical scheme for a sequential single-core processor. As before, let $\lambda = \lceil \log_2(p) \rceil$. Referring to Table 8.1, the computational cost of the CRT-based eSIDH instantiation saves $\approx \frac{\lambda}{4}$ xDBL operations.

Case study p_{443}
Based on the timing computational costs reported in Table 8.3, the expected computational saving for a single-core implementation of SIDH p_{434} with respect to eSIDH p_{443} is given as,

$$\begin{aligned} e_C \cdot \text{Quintupling} &= 45 \cdot 7995 \\ &= 359,775 \text{ clock cycles.} \end{aligned}$$

This implies that compared against a single-core SIDH p_{434} implementation, a single-core CRT-based eSIDH p_{443} implementation is expected to produce a 1.02 speedup factor.

Case study p_{765}
Based on the prime specifications given in 8.2 and the timing computational costs reported in Table 8.3, the expected computational saving for a single-core implementation of eSIDH p_{765} with respect to SIDH p_{751} is given as,

$$\begin{aligned} e_C \cdot \text{Quintupling} &= 81 \cdot 16285 \\ &= 1,319,085 \text{ clock cycles.} \end{aligned}$$

This saving combined with the experimental results reported in Table 8.6 implies that compared against a single-core SIDH p_{751} implementation, a single-core CRT-based eSIDH p_{765} implementation is expected to produce a 1.07 speedup factor.

8.5 Conclusions

In this Chapter, the extended SIDH scheme, a variant of the SIDH protocol in [32], was presented. Our experimental results show that an eSIDH parallel implementation is faster than a corresponding parallel version of SIDH.

Our future work includes to expand the search of more efficient eSIDH primes for all the four security levels considered in [3]. Building on the work presented in [46], we would also like to explore more aggressive approaches for parallelizing the SIDH isogeny computations and evaluations. The algorithmic ideas discussed here might be useful for the B-SIDH construction [21], where given the large size of the prime factors involved in the factorization of $p \pm 1$, parallel implementations of SIDH become mandatory. We also would like to explore applications of eSIDH to the client-server scenarios discussed in [21].

Chapter 9

Parallel Strategies for SIDH

9.1 Introduction

Isogeny-based cryptography was proposed by Couveignes in 1997. Complete details of his proposal were eventually reported in [28]. In 2006, Couveignes' protocol was independently rediscovered by Rostovtsev and Stolbunov in [85, 93]. Also in 2006, Charles-Lauter-Goren introduced in [19] the hardness of path-finding in supersingular isogeny graphs and its application to the design of hash functions. In 2011, the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was proposed by Jao and de Feo in [49]. Later in 2017, the Supersingular Isogeny Key Encapsulation (SIKE) protocol was submitted to the NIST post-quantum cryptography standardization project [3]. On July 2020, NIST announced the results of its third round evaluation, naming SIKE as an alternative candidate. SIKE allows public key encryption and comprises a key encapsulation mechanism that is protected against chosen-ciphertext attacks.

The two most costly computational tasks of SIDH are, (i) the computation of large smooth-degree isogenies of supersingular elliptic curves along with the evaluation of the image of elliptic curve points in those isogenies and; (ii) elliptic curve scalar multiplication computations via three-point Montgomery ladder procedures. Optimal computation of large degree isogenies for single-core processors was presented and solved in [32]. Also, efficient algorithms for computing the SIDH three-point scalar multiplications can be found in [49, 38]. Several general ideas for a sensible improving of the SIDH performance were introduced in [26].

Let A be a Montgomery coefficient of an elliptic curve $E : y^2 = x^3 + Ax^2 + x$, defined over the quadratic extension field $\mathbb{F}_{p^2}$. Let $S = \langle R_0 \rangle$ be an order-ℓ^e subgroup of $E[\ell^e]$, where $R_0 \in E(\mathbb{F}_{p^2})$, is a point of order ℓ^e, e is a positive number, and ℓ is a small prime. Then there exists a degree-ℓ^e isogeny $\phi : E \to E'$ having kernel S. The image curve E' is also a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Moreover, $\text{Order}(E(\mathbb{F}_{p^2})) = \text{Order}(E'(\mathbb{F}_{p^2}))$ [95, Theorem 1]. Let $Q \in E(\mathbb{F}_{p^2}) \backslash \langle P \rangle$. Computing the Montgomery coefficient $A' \in \mathbb{F}_{p^2}$ of the codomain curve $E' : y^2 = x^3 + A'x^2 + x$, and the image point $\phi(Q)$, is referred in this chapter as the isogeny construction and the isogeny evaluation computational tasks, respectively.

In [32], optimal strategy techniques were introduced to efficiently compute degree-ℓ^e isogenies at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.[1] Virtually all SIDH and SIKE implementations published as of today, compute degree-ℓ^e isogenies using optimal strategies, which are provable optimal for those architectures equipped with a single unit of processing, i.e., single-core platforms.

In [32], optimal strategies were depicted as a weighted directed graph whose vertices are elliptic curve points and whose left and right edges have as associated weight the cost of performing one scalar multiplication by ℓ and one degree-ℓ isogeny, respectively. That weighted directed graph was drawn in Chapter 4, as a right triangular lattice Δ_e having $\frac{e(e+1)}{2}$ points distributed in e columns and rows. A leaf is defined as the most bottom point of a given column in that lattice. All vertical edges must be computed sequentially, whereas all the horizontal edges can be computed in parallel. At the beginning of the isogeny computation, only the point R_0 of order ℓ^e is known. The isogeny computation is carried out by obtaining from left to right, each one of the leaves in Δ_e until the farthest right one, R_{e-1}, is computed. Then, $\phi : E \to E'$ can be found by calculating a degree-ℓ isogeny with kernel R_{e-1}.

An interesting consequence of the weighted directed graph representation is that one can abstract oneself from the cryptographic nature of the isogeny computation problem, and solely focus on the combinatorial structure associated to the graph.

As an illustrative example, consider the toy example depicted in Figure 9.1 using the parameters $\ell^e = 4^9$. In the event that that strategy is executed on a single-core platform, it would have an associated timing cost of thirteen scalar multiplications by 4 (corresponding to the thirteen vertical blue edges shown in the graph), plus sixteen degree-4 isogeny evaluations (corresponding to the sixteen horizontal red edges shown in the graph).[2] However, if one happens to have four cores available for performing this task, then the timing cost

[1] An analysis of the computational cost of small degree isogeny construction and evaluation can be found in [26, 23, 18].

[2] The cost of computing the strategy shown in Figure 9.1 also includes ten degree-ℓ isogeny constructions not relevant for the discussion here.

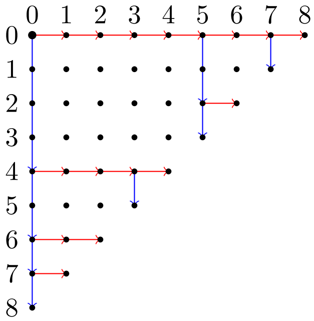


Figure 9.1: Strategy to compute a degree-$\ell^e = 4^9$ isogeny. The root point at row and column zero, represents the elliptic curve point R_0 of order 4^9. Each one of the nine leaves at the bottom of the columns represent elliptic curve points of order 4.

can be reduced to thirteen scalar multiplications by 4, plus just eight degree-4 isogeny evaluations.

Parallel computations of SIDH

The chief criticism made against SIDH, is that its latency is much higher than the ones associated to several other candidates of the NIST standardization project [74]. Motivated by this, numerous efforts to speed up the performance of SIDH both in software [26, 61, 38, 88], and in hardware [60, 59], have been reported. Nonetheless, to our knowledge only the works presented in [60, 46, 58], have attempted to exploit the rich opportunities for parallelism that SIDH has to offer.

In [60], a hardware implementation of SIDH on an FPGA device was presented. The authors' architecture was able to concurrently process an average of four degree-4 isogeny evaluations. Using the 751-bit prime SIKE$p_{751} = 2^{372} \cdot 3^{239} - 1$,[3] these parallel calculations accounted for a saving of 36.5% in the number of clock cycles required for computing degree-4^{186} isogenies over $\mathbb{F}_{p^2}$.

The noticeable speedup reported in [60] can however be considered sub-optimal, in the sense that for a degree-4^{186} isogeny computation, the authors adapted a strategy originally conceived for a sequential execution (as opposed to explicitly designing a parallel strategy for that purpose). Pointing out this limitation, the authors of [46] proposed the usage of provable optimal strategies specifically conceived for the parallel computation of large degree isogenies in SIDH. From a careful theoretical analysis, the authors concluded that an eight-core parallel implementation of the SIDH isogeny computation using their approach, should achieve a performance speedup of up to 55% compared against a sequential version of SIDH. However, this acceleration factor was achieved by assuming isogeny

[3] A SIKE instantiation using the prime SIKEp_{751}, achieves the NIST's category 5 security level [3].

and scalar multiplication costs that not necessarily correspond to the ones observed in the most efficient software and hardware implementations reported for SIDH. Moreover, the authors of [46] focused all their attention to the efficient parallelization of the SIDH isogeny computations, leaving out attempts for concurrently executing other SIDH computational tasks, such as three-point Montgomery ladders.

A variant of SIDH named eSIDH in Chapter 8, permits to accelerate Bob's computations on single and multi-core platforms. Remarkably, eSIDH uses Montgomery-friendly primes of the form $p = 4^{e_A}3^{e_B}5^{e_C}f - 1$, which offer a faster field arithmetic than the one associated to comparable SIKE primes. Comparing against a SIKEp_{751} sequential instantiation of SIDH, the authors reported an acceleration factor of $1.05, 1.30$ and 1.41, when eSIDH was implemented on $k = \{1, 2, 3\}$-core processors, respectively. However, the approach proposed in Chapter 8 does not *per se* provide speedups for Alice SIDH computations. Moreover, as in [46], eSIDH does not attempt to compute concurrently three-point Montgomery ladders with large degree isogeny computations.

Contributions and organization of this chapter

The main contribution of this chapter is the proposal of a concrete, efficient and practical strategy for a parallelized computation of the SIDH and SIKE protocols. The strategies presented in this work strive for concurrently computing the two most prominent SIDH primitives, namely, the evaluation/construction of large degree isogenies and the computation of right-to-left three-point Montgomery ladders. We propose efficient core load distributions for evaluating large degree isogenies using k cores. Furthermore, we report experimental results showing that a three-core implementation of our parallel approach achieves an acceleration factor of 1.45 compared against a sequential implementation of SIKE (cf. Table 9.8).

The remainder of this chapter is organized as follows. A general description of sequential and parallel strategies for computing large smooth-degree isogenies are presented in §9.2 (an interested reader must also see §4.3). In §9.3, it is observed that the multiples of Alice and Bob secret points, which are always required in any valid strategy for computing isogenies, can be calculated independently and concurrently. The performance implications of this trick are further discussed in §9.3. Estimates and experimental results are presented in §9.5. Finally, some concluding remarks are drawn in §9.6. We recall that an interested reader must see the Chapter 4 for further reading about SIDH and strategies.

9.2 Parallel strategies for large smooth-degree isogenies

In this section, the problem of designing parallel strategies and associated criteria to decide when a parallel strategy is optimal are presented. We say that an strategy St_e is better than another strategy St'_e, if the cost of computing St_e is lesser than the cost of computing St'_e when executed on a k-core platform.

In a nutshell, our approach to parallelize isogeny computations exploits two main tricks: (i) As per Rule 7 of §4.3.1, one can make use of all of the k available cores to concurrently compute the horizontal edges associated to any given column; (ii) As it was done in the sequential setting in [32], one can use dynamic programming to translate the problem of optimizing a strategy Δ_e to the simpler problem of optimizing the sub-triangles T_{e-h} and T_h, for $h \in \{1, 2, \ldots, e-1\}$. This process must carefully consider the parallel computational cost of the strategy.

In the remaining of this Section, we describe in detail both of these two options.

9.2.1 Exploiting the parallelism of the horizontal edges

In order to measure strategy costs the following proposition becomes useful.

Proposition 9.2.1. Let q_ℓ be the timing cost associated to the computation of a degree-ℓ isogeny. Let us define a set of horizontal edges for a fixed index $j \in \{0, 1, \ldots, e-2\}$ by $Col_j(St) = \{[(i,j),(i,j+1)] \in Edges(St_e) \mid i \in [0, e-j-2]\}$. The timing cost of computing all horizontal edges in $Col_j(St_e)$ using k cores is of

$$\left\lceil \frac{\#Col_j(St_e)}{k} \right\rceil \cdot q_\ell$$

Proof. Let us say that $\#Col_j(St_e) = m$. If $k \geq m$ then one can compute all m edges at once, at an equivalent cost of one isogeny evaluation q_ℓ. Otherwise, the equivalent of $a = \lceil \frac{m}{k} \rceil$ isogeny evaluations q_ℓ suffice for computing all the horizontal edges in column j of strategy St_e. □

Using the previous Proposition one can compute the cost of all the *horizontal edges* of St_t using k cores, denoted by $C^k(St_e)$, by applying the following Lemma.

Lemma 9.2.2. Let us define the set of horizontal edges from the column j to the column $j+1$ as in Proposition 9.2.1. The cost of all evaluations defined by St_e using k cores is given by

$$\sum_{j=0}^{e-2} \left\lceil \frac{\#Col_j(St_e)}{k} \right\rceil \cdot q_\ell$$

Now the cost of evaluating St_e using k cores is given as

$$C^k(St_e) = \sum_{j=0}^{e-1} \left\lceil \frac{\#Col_j(St_e)}{k} \right\rceil \cdot q_\ell + \#V(St_e) \cdot p_\ell,$$

where $V(St_e)$ is the set of all vertical edges in St_e, and as before p_ℓ and q_ℓ represent the costs of computing one scalar multiplication by ℓ and evaluating one degree-ℓ isogeny.

Lemma 9.2.3. An $e-1$-core platform can compute an ℓ^e isogeny at a cost of $e-1$ scalar multiplications by ℓ, e degree-ℓ isogeny evaluations and e degree-ℓ isogeny constructions.

Proof. Using of the isogeny-oriented strategy described in § 4.3.2 (cf. Subfigure 4.3b), this cost can be justified as follows. Compute and store all vertices $(0,i)$ for $i=0$ to $e-1$. This operation costs $e-1$ scalar multiplications by ℓ. Now, for $i=0$ to $e-2$ using $e-1-i$ cores one can perform the isogeny evaluation of the per-column $e-1-i$ points in parallel. Moreover, at each one of the e columns, one isogeny construction must be performed. The last degree-ℓ isogeny with kernel given by the point in the vertex $(0,e-1)$ is computed using only one core. □

9.2.2 Using Dynamic programming for finding parallel strategies

Lemma 9.2.4. [32] Given a triangle Δ_e and its decomposition into Δ_h and Δ_{e-h}, the sequential cost of traversing St_e using this particular decomposition is given as,

$$C^1(St_e^h) = C^1(St_h) + C^1(St_{e-h}) + (e-h)\cdot q_\ell + h\cdot p_\ell.$$

We say that St_e is an optimal strategy if $C^1(St_e^h)$ is minimal among all St_e^h for $h\in[1,e-1]$.

This lemma is illustrated in Figure 4.4. Lemma 9.2.4 can be generalized in a natural way to k cores as follows.

Lemma 9.2.5. The cost of traversing St_e^h using k cores is given as,

$$C^k(St_e^h) = C^k(St_{e-h}) + C^k(St_h) + \frac{(e-h)\cdot q_\ell}{k} + h\cdot p_\ell, \tag{9.1}$$

We say that St_e^h is an optimal parallel strategy if $C^k(St_e^h)$ is minimum among all St_e^h for $h\in[1,e-1]$.

The cost above can be justified by the fact that one can include the $(e-h)$ extra degree-ℓ isogeny evaluations into the computation of the horizontal edges of St_{e-h}. Since the cost $C^k(St_h)$ depends on two sets, namely, the set of all columns of St_h and the set of all vertical edges $V(St_h)$, a precise way to keep track of both sets must be put in place as discussed next.

9.2.3 Constructing and Traversing parallel strategies

Let us assume that the parameters e,k,p_ℓ,q_ℓ, corresponding to the size of the tree Δ_e, the number of available cores, the cost of performing one scalar multiplication by ℓ and the cost of performing one degree-ℓ isogeny, respectively, are all given. Then Algorithms 12, 13 and Algorithm 11, find an optimal parallel strategy for St_e by using a bottom-up approach.

Algorithm 11 is essentially the same as Algorithm 46 in [3, Appendix C]. Algorithm 11 produces as an output, a linear vector of the split nodes included in the optimal parallel strategy St_e. To this end, Algorithm 11 invokes Algorithm 13 at Line 6.

Algorithm 13 iteratively finds the row $h \in [1, e-1]$ that produces a minimum cost strategy St_e composed of the two sub-strategies St_{e-h} and St_h (cf. Figure 4.4). For this purpose, Algorithm 13 invokes Algorithm 12, which uses Eq. (9.1) to calculate the computational expenses associated to the strategies St_h and St_{e-h}.

We stress that Algorithm 13 follows a bottom-up approach by constructing optimal parallel strategies of size $1, 2, \ldots, e-1$, in that order. To illustrate the process outlined above consider the following toy example.

Algorithm 11 `get_Parallel_Strategy: Obtains the optimal parallel strategy for` Δ_e `using` k`-cores`

```
Require: $e$: the number of leaves, $p_\ell$: Cost of the scalar multiplications by $\ell$, $q_\ell$: the cost
    of a degree-$\ell$ isogeny evaluation, $K$: number of available cores)
Ensure: S: Strategy to traverse $\Delta_e$.
 1: $E := [[], [1]]$;                   //Set of set of horizontal edges
 2: $M := [0, 1]$;                      //Set of multiplication counts
 3: $S := [[], [1]]$;                   //Set to keep the partial strategies
 4: 4C := $[0, p_\ell + q_\ell]$4;      //Set to keep the cost of partial strategies
 5: for $i \in [3..(e+1)]$ do
 6:     $cost, h, Ei, muls := GMPS(C, E, M, i, p_\ell, q_\ell, K)$;
                                        //Algorithm 13
 7:     Append($\sim C, cost$);         //getting cost;
 8:     Append($\sim E, Ei$);           //updating the set of sets of horizontal edges
 9:     Append($\sim M, muls$);         //updating the set of Muls.
10:     Append($\sim S$, $[b]$ cat $S[i-b]$ cat $S[b]$);//building the new strategy
11: end for
12: return $S[e+1]$;
```

Example 4. Let us assume $e = 6, p_\ell = 1, q_\ell = 1$ and $k = 2$. Algorithm 11 uses the following construction to discover an efficient parallel-strategy St_6.

1. Figure 9.2a shows the initial setting of size-1 triangles.

2. Figure 9.2b shows the two size-3 strategies considered by algorithm 13 for $e = 3$. The parallel cost of the left and right strategies is 4 and 5, respectively. Hence, the left one is chosen. The output vector S is set to $S = [[], [1], [1, 1]]$.

3. Figure 9.2c shows the three size-4 strategies considered by algorithm 13 for $e = 4$. The parallel cost of the first and second strategies is of 7 units. Algorithm 13 chooses

Algorithm 12 Ev_counts: Computes Eq.(9.1) for a given partition St_e^h

Require: H_0 and H_1 are sets of horizontal edges counts, k is the number of cores available

Ensure: H_e: The new set of horizontal edges generated by H_0 and H_1. evs: The number of evaluations required to compute the set of edges H_e when k cores are available.

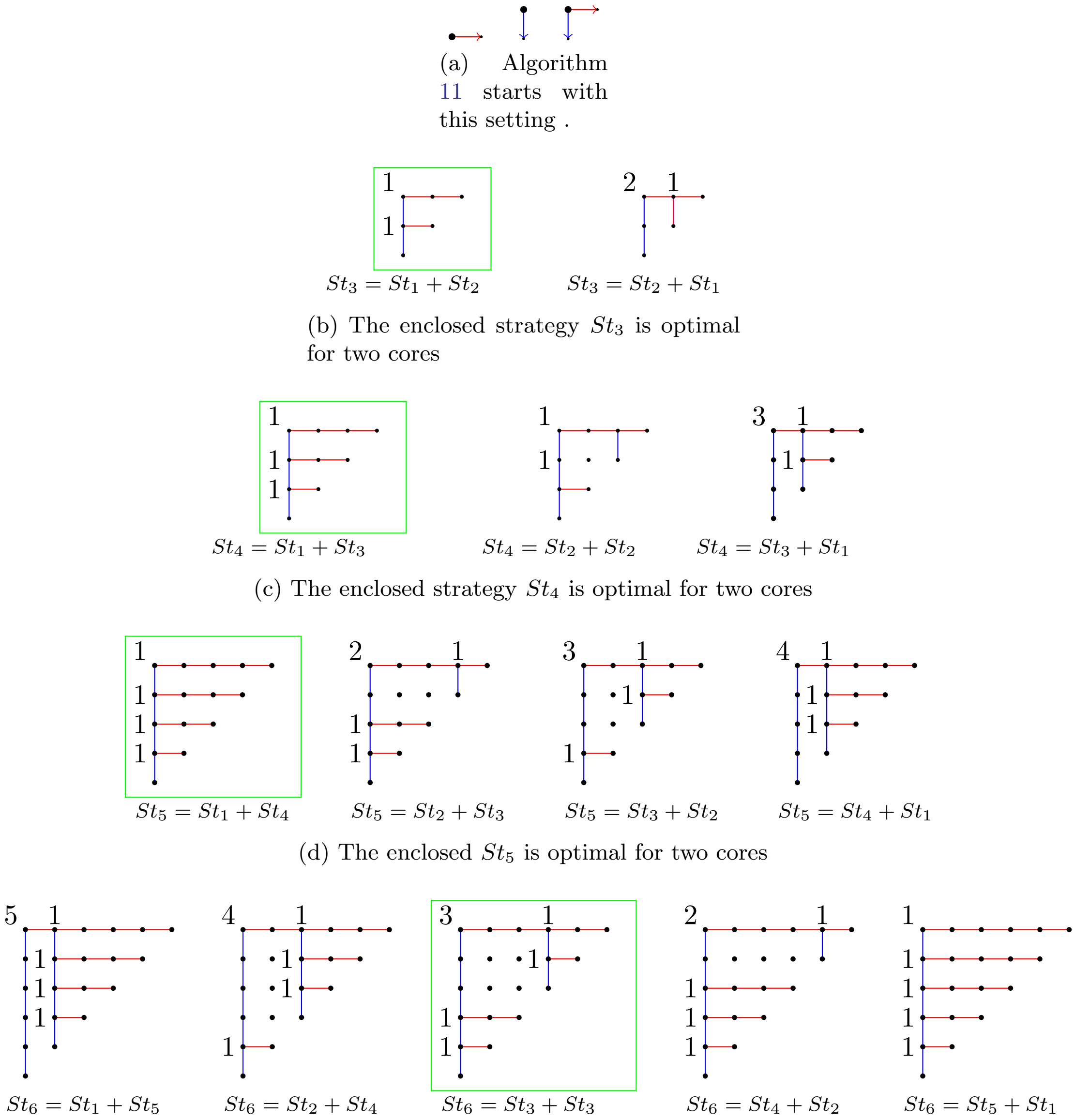
```
if $H_0$ is empty then                    //Merging both strategies
    $H_e := [1]$ cat $[i : i \in H_1]$;
else
    $H_e := [i + 1 : i \in H_0]$ cat $[1]$ cat $H_1$;
end if
$evs := 0$;
if $k = 1$ then                           //getting the cost of $H_e$ when $k$ cores used
    $evs := \sum_{e \in He} e$;
else
    for $i \in He$ do
        if $i \leq k$ then                //Using Lemma 9.2.5
            $evs +:= 1$;
        else
            $evs +:= \lceil i/k \rceil$;
        end if
    end for
end if
return $evs, He$;
```

Algorithm 13 `get_Min_Parallel_Strat: Finds an optimal` St_e^h `splitting`

Require: C: Set of costs, E: Set of sets of horizontal edges, V: Set of Multiplication operations, e: number of leaves , p_ℓ: Cost of one scalar multiplication by ℓ, q_ℓ: Cost of one evaluation by a degree-ℓ isogeny, k: number of available cores.

Ensure: minC: The cost of the minimum strategy for e leaves using k cores. minEn: The new set of edges for the minimum strategy for e leaves using k cores. minM : The multiplication counts for the minimum strategy for e leaves using k cores.

```
$minC := e^{2*(p_\ell+q_\ell)}$;                  //just an upper bound
for $b := 1$ to $e-1$ do
    $ev\_counts\_temp, St\_temp := Ev\_Counts(E[e-b], E[b], k)$;
    //Algorithm 12
    $muls\_temp := (V[e-b] + V[b] + b)$;  //number of multiplications on $St\_temp$
    $cost\_temp := (ev\_counts\_temp * q_\ell) + (muls\_temp * p_\ell)$;
    if $cost\_temp < minC$ then
        $split := b$;
        $minC := cost\_temp$;
        $minEn := E\_temp$;
        $minV := muls\_temp$;
    end if
end for;
return $minC, split, minEn, minV$;
```



(a) Algorithm 11 starts with this setting .

(b) The enclosed strategy St_3 is optimal for two cores

(c) The enclosed strategy St_4 is optimal for two cores

(d) The enclosed St_5 is optimal for two cores

(e) The enclosed St_6 is optimal for two cores

Figure 9.2: A toy example of a parallel optimal-strategy search using dynamic programming and the parameter set $e = 5, p_\ell = 1, q_\ell = 1$ and $k = 2$. 11

Algorithm 14 Non-recursive walking across a Strategy St_e

```
Require: A strategy $St_e$ obtained from algorithm 11, Elliptic Curve $E_0$, Point $R \in E_0$ of
    order $\ell^e$.
Ensure: Elliptic Curve $E_e$ such that there is a degree-$\ell^e$-isogeny between $E_0$ and $E_e$.
$idx := 0$;
$i := 1$;
$points := [[R, 0]]$;
for $row := 0$ to $e - 1$ do
    while $idx < n - row$ do
        $R_t := [\ell^{St[i]}]R_t$;
        $idx +:= St[i]$;
        Push($points$, $[R_t, idx]$);
        $i +:= 1$;
    end while
    Compute $\phi_{row}$ and $E_{row+1}$ using $E_{row}$ and $R_t$.
    Prune($points$);
    for $j := 1$ to $\#points$ do                                  //PARALLEL FOR
        $points[j, 1] := \phi_{row}(points[j, 1])$;
    end for
    $[R_t, idx] :=$ Pop($points$);
end for
Compute $\phi_{e-1}$ and $E_e$ using $E_{e-1}$ and $R_t$.
return $E_e$;
```

the first one because in Line 6 of this procedure there is a strict less condition. If one relaxes this condition to a strict less or equal comparison, then the second strategy would be used. Now the output vector is set to: $S = [[], [1], [1,1], [1,1,1]]$.

4. Figure 9.2d shows the four different size-5 strategies for $n = 5$. In this case, the first three strategies cost 10 units. Again, Algorithm 13 chooses the first one. Now, the output vector is set to $S = [[], [1], [1,1], [1,1,1], [1,1,1,1]]$ and the optimal parallel strategy output by Algorithm 13 is completely defined by the linearized vector, $St_5 \;=\; [1,1,1,1]$.

5. Figure 9.2e shows the 5 different size-6 strategies for $n = 6$. In this case, the third and four strategies cost 13 units. Again, Algorithm 13 chooses the first one of both. Now, the output vector is set to $S = [[], [1], [1,1], [1,1,1], [1,1,1,1], [3,1,1,1,1]]$ and the optimal parallel strategy output by Algorithm 13 is completely defined by the linearized vector, $St_5 \;=\; [1,1,1,1]$.

6. The vector $St_6 = [3,1,1,1,1]$ as well as the base curve E and the order-ℓ^e point $R \in E(\mathbb{F}_q)$, are the input parameter required by Algorithm 14 for computing a degree-ℓ^e isogeny.

9.3 Parallelizing the computation of the multiples of the SIDH secret points

In this section, an interesting property of the Montgomery ladders is exploited. This property allows us to extract more parallelism opportunities from the SIDH main computations. For the sake of simplicity, optimization opportunities for computing Alice's degree-4^{e_A} isogenies are mostly discussed. Details of Bob's degree-3^{e_B} isogeny computations are given in Appendix 9.4.

Let us recall that in order to compute a scalar multiplication of the form $P + [m]Q$, the three-point Montgomery ladder used in SIDH has a per-step cost of 1 xADD and 1 xDBL [38]. The cost of this ladder (cf. Remark 21), is essentially of two xDBL operations per step, which implies that the computation of Alice's secret point R_A costs about $4e_A$ xDBL operations.

In §4.3, it was discussed that starting from the root point R_A of order-4^{e_A}, any strategy St_e must compute the multiples $[4^i]R_A$ belonging to its first column, for $i = 1, \ldots, e-1$. Hence, a naive iterative approach for computing the point multiple $[4^i]R_A$, would compute first the point R_A. Thereafter, from R_A the desired multiple can be obtained by performing $2i$ doubling operations. The computational cost of such approach is of about,

$$4e_A \text{ xDBL} + 2i \text{ xDBL} = (4e_A + 2i) \text{ xDBL operations.}$$

Note that this approach also finds as by-products, the multiples $[4^j]R_A$ for $j = 1, \ldots, i-1$. However using the approach discussed in [80] (see also [38]), there exists a more efficient strategy for computing any multiple of R_A.

Proposition 9.3.1. Let P_A, Q_A, m_A, R_A be the public and private keys of Alice where $R_A = P_A + [m_A]Q_A$, and Order(P_A)= Order(Q_A)= Order(R_A) $= 4^{e_A}$. Then, for $i = 1, \ldots, e_A - 1$, the computation of the point $[4^i]R_A$ costs $2(e_A - i)$ xDBL operations.

Proof. Since P_A and Q_A are public parameters, one can pre-compute all the multiples $[4^i]P_A$ and $[4^i]Q_A$ for $i = 1, \ldots, e_A - 1$. From a direct manipulation one can write,

$$[4^i]R_A = [4^i]P_A + [m_A]([4^i]Q_A).$$

Observing that the point $[4^i]Q_A$ has order $4^{e_A - i}$, then m_A can be replaced by $\bar{m_A}$ where $\bar{m_A} = m_A \mod 4^{e_A-i}$, which is a $2(e_A - i)$-bit long integer and compute

$$[4^i]R_A = [4^i]P_A + [\bar{m_A}][4^i]Q_A,$$

using the fixed-point three-point Montgomery ladder of [80, 38], at a cost of about 1 xADD (≈ 1xDBL) per bit.[4] □

Proposition 9.3.2. Let P_A, Q_A be the public keys of Alice with Order(P_A)= Order(Q_A) $= 4^{e_A}$. Let $\phi_B(P_A)$ and $\phi_B(Q_A)$ be the public points that Alice receives from Bob, and let m_A be Alice's secret scalar. Then, for $i = 1, \ldots, e_A - 1$, the computation of the point $[4^i]\phi_B(R_A)$ costs $(4e_A - 2i)$ xDBL operations.

Proof. From a direct manipulation one can write,

$$\begin{aligned}[4^i]\phi_B(R_A) =& [4^i](\phi_B(P_A) + [m_A]\phi_B(Q_A)) \\ =& [4^i]\phi_B(P_A) + [m_A]([4^i]\phi_B(Q_A)).\end{aligned}$$

Similar to Proposition 9.3.1, the multiple $[4^i]\phi_B(Q_A)$ has order $4^{e_A - i}$. Then, one can replace m_A by $\bar{m_A}$, where $\bar{m_A} = m_A \mod 4^{e_A - i}$ which has $2(e_A - i)$ bits. One can compute $(P_A + [\bar{m_A}]Q_A)$ using a three-point Montgomery ladder at a cost of $4(e_A - i)$ xDBL operations. As $\phi_B(P_A)$ and $\phi_B(Q_A)$ both depend on Bob's secret key, it is not possible to pre-compute off-line anything relevant. Thus, one needs to compute $[4^i](\phi_B(P_A) + [m_A]\phi_B(Q_A))$, which can be done by repeatedly doubling $\phi_B(P_A) + [\bar{m_A}]\phi_B(Q_A)$. This has a computational cost of $2i$ doublings. Adding this to the cost of the three-point ladder gives us the desired result of $(4(e_A - i) + 2i) = (4e_A - 2i)$ xDBL operations. □

Propositions 9.3.1 and 9.3.2, state that the multiples $[4^i]R_A$ and $[4^i]\phi_B(R_A)$ for $i \in \{1, \ldots, e_A - 1\}$, can be computed at a cheaper cost than the one associated to the calculation

[4] A detailed low level description of this computation for both, Alice and Bob, is given in §9.4.1.

of the points R_A and $\phi_B(R_A)$, respectively. As a way of illustration, in Figure 9.3, one core can be devoted to compute R_A. A second core can compute concurrently the multiple $[4^{e-b}]R_A$, requiring $2b$ less xDBL operations than the calculation performed by the first core.

Remark 25. Using an e-core architecture, one can concurrently compute all multiples $[4^i]R_A$ or $[4^i]\phi_B(R_A)$, for $i = 0, 1, \ldots, e-1$, at an equivalent computational cost of $2e_A$ and $4e_A$ xDBL operations, respectively.

Let us assume that one disposes of a k-core processing unit. Since the computation of any of the multiples of the point R_A is more economical than the computation of R_A, it just makes sense to devote one core to the relatively expensive task of computing R_A, while the other $k-1$ cores can team up to concurrently compute the smaller lower triangle shown in Figure 9.3, which is done by means of a parallel strategy St_b. Note that to be able to compute such triangle, the multiple $[4^{e-b}]R_A$ must be computed first. For efficiency reasons all the costs associated to these tasks, must be carefully balanced as formalized in the following proposition.

Proposition 9.3.3. Let P_A, Q_A, m, R_A be the public and private keys of Alice where $R_A = P_A + [m]Q_A$, and let k be the number of cores available to compute a 4^{e_A}-isogeny. Let $p_4 \ = \ 2$ xDBL be the cost of computing a point multiplication-by-4, and r_4 be the cost of a degree-4 isogeny construction. Then, one can compute a 4^b-isogeny by means of a parallel strategy St_b that uses $k-1$ cores, at the same time that one core is devoted to compute R_A (resp. $\phi_B(R_A)$), where b is given as,

$$b = \max_{i:=1,2,\ldots,\frac{e_A-1}{2}} \{i \mid C^{k-1}(St_i) + i \cdot p_4 + i \cdot r_4 \leq e_A p_4\} \tag{9.2}$$

For the key generation and the key agreement phases.

Proof. Let $C^{k-1}(St_i)$ denote the cost of a parallel strategy St_i using $k-1$ cores (cf. §9.2). In the case of the key generation phase, the result of Proposition 9.3.1 states that the computation of R_A and $[4^{e-i}]R_A$ require $e_A \cdot p_4$ and $i \cdot p_4$ operations, respectively. Then, the cost of computing a 4^b-isogeny using $k-1$ cores is approximately the same of computing R_A with a single core, for the maximum value i such that the inequality of Eq. (9.2) still holds.

In the case of the key agreement phase, the result of Proposition 9.3.2 states that computing $\phi_B(R_A)$ and $[4^{e-i}]\phi_B(R_A)$ require $2e_A \cdot p_4$ and $(e_a+i) \cdot p_4$ operations, respectively. Hence, once again Eq. (9.2) gives the approximately crossover point where computing R_A costs about the same of calculating a 4^b-isogeny.

□

Figure 9.3 illustrates how the result of Proposition 9.3.3 can be used to compute a degree-4^{e_A} isogeny using a core-k processing unit. One core is exclusively dedicated to the

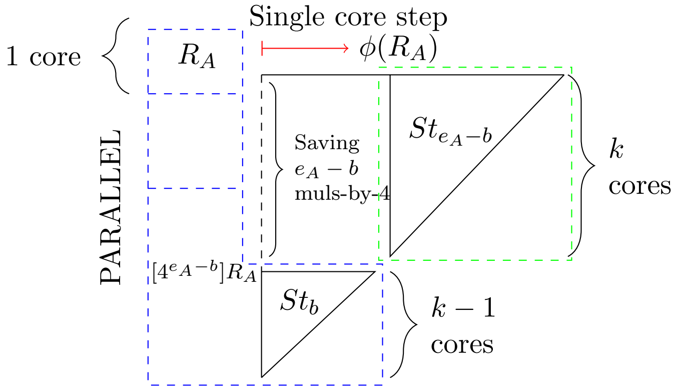


Figure 9.3: Representation of a k-core load distribution for the parallel computation of the strategy St_{e_A} as stated in Proposition 9.3.3. The left-most blue dash-rectangle computations are performed in parallel during the first phase of this isogeny evaluation.

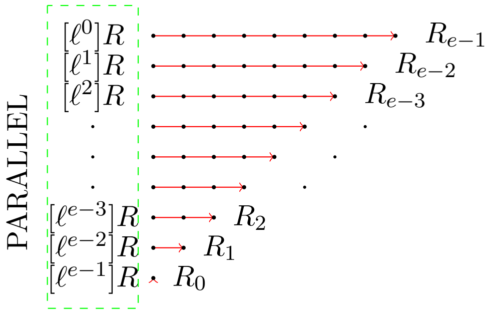


Figure 9.4: Parallel evaluation of an isogeny if the hardware resources are plentiful enough.

computation of the point R_A. At the same time, the other $k-1$ cores compute the lower subtriangle shown in Figure 9.3, using a parallel strategy St_b. The size of this subtriangle has been chosen according to Proposition 9.3.3 so that both of these two computations are completed at roughly the same time. After that, $\phi(R_A)$, a degree-4^b isogeny evaluation of the point R_A is sequentially computed as the single-core step shown in Figure 9.3. Thereafter, the upper subtriangle of Figure 9.3 can be computed using a k-core parallel strategy St_{e-b}. This completes the parallel evaluation of a strategy St_{e_A} using k cores. It is worth mentioning that the multiples of the point $[4^i]R_A$, for $i = 1, \ldots, e-b-1$, are completely skipped in this computation. See §9.4.1 for a discussion of practical low-level aspects associated to the k-core parallel implementation of Figure 9.3.

From Proposition 25, if e cores are available for the computation of SIDH, then one can compute $[4^i]R$ for $i = 0$ to $e-1$ in parallel at the same cost of computing one three-point-ladder of $e\log_2(\ell)$-bits. Then all vertices $(0, i)$ for $i = 0$ to $e-1$ of a given strategy can be computed at once. Let p_ℓ, q_ℓ and r_ℓ be the cost of a multiplication-by-ℓ, and the cost

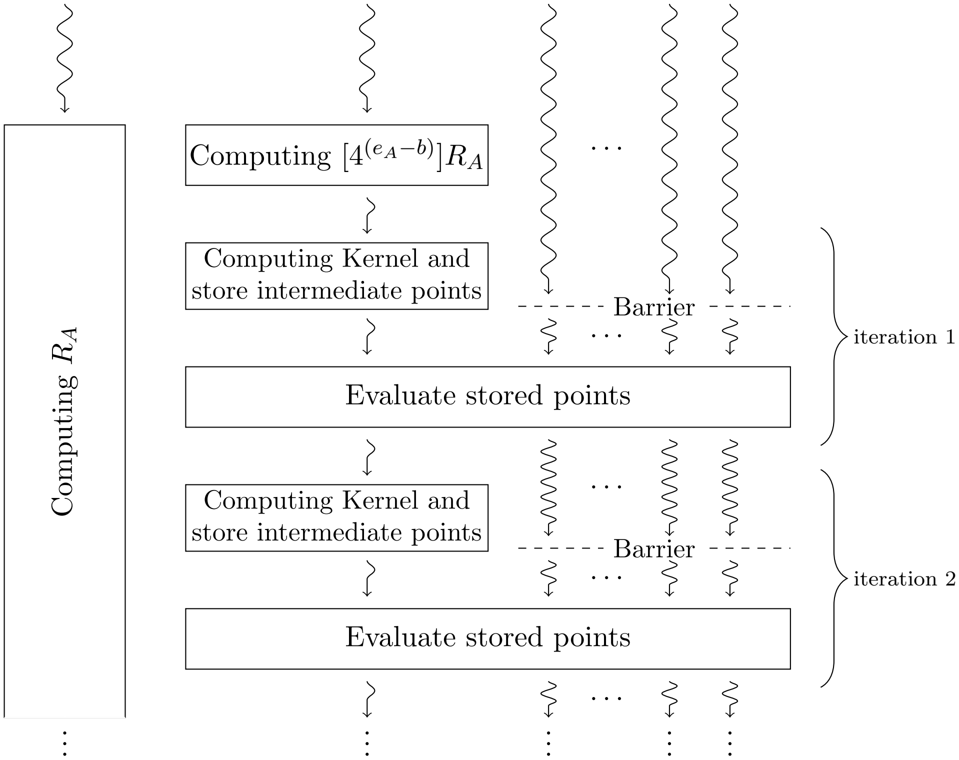

Computing R_A
Computing $[4^{(e_A-b)}]R_A$
Computing Kernel and store intermediate points
Barrier
Evaluate stored points
iteration 1
Computing Kernel and store intermediate points
Barrier
Evaluate stored points
iteration 2

Figure 9.5: Diagram showing the flow of the threads in our parallel eSIDH proposal for Alice. This flow is only for the first b iterations, afterwards the flow is similar but the thread computing R_A is then joined to the threads computing point evaluations.

of a degree-ℓ isogeny evaluation and construction, respectively. Now if $r_\ell < p_\ell$ then the core that computes $[\ell^{e-1}]R$ can also compute the first isogeny construction and from then on, this core can be dedicated to compute all remaining codomain curves. Since $r_\ell < q_\ell$, then, r_ℓ is dominated by q_ℓ and its associated cost is dominated by the computation of the codomain curve and evaluations in parallel. The assumptions $r_\ell < q_\ell$ and $r_\ell < p_\ell$ are valid for $\ell = 3, 4$[5]

In summary, if e cores are available for the computation of SIDH, then the computation of an ℓ^e isogeny costs $e-1$ ℓ-isogeny evaluations plus one three-point ladder as illustrated in Figure 9.4.

[5] This assumption is in general true, but for $\ell \geq 5$ there is an extra cost associated to the kernel points generation because the corresponding kernel subgroup has more than one point.

9.4 Parallelizing the computation of the multiples of the point R_0 for Bob

In the following, Propositions 9.3.1 and 9.3.2 are generalized to consider multiplications by ℓ different than 4.

Proposition 9.4.1. Let P, Q, R be points on an elliptic curve E, m, ℓ and e be integers such that $\text{Order}(P) = \text{Order}(Q) = \text{Order}(R) = \ell^e$, $R = P + [m]Q$, and $m < \ell^e$. Then computing $[\ell^i]R$ for $i = 0$ to $e - 1$, costs about $(2e - i)\log_2(\ell)$ xDBL. This implies that computing the multiple $[\ell^i]R$ for $i = 1$ to $e-1$, costs less than the computation of the point R. If points are known in advance, computing $R = P + [m]Q$ costs about $(2e - i)\log_2(\ell)$ xDBL

Proof. The cost of computing R is of about $2e\log_2(\ell)$ xDBL because m is at most an $(e\log_2(\ell))$-bit long integer. As in Proposition 9.3.2, one has

$$[\ell^i]R = [\ell^i](P + [m \mod \ell^{e-i}]Q).$$

Here, $m \mod \ell^{e-i}$ has at most $\log_2(\ell^{e-i})$ bits. Then computing $P + [m \mod \ell^{e-i}]Q$ costs $2(e - i)\log_2(\ell)$ xDBL. By adding i scalar multiplications by ℓ at a cost of $i\log_2(\ell)$, the claimed result is obtained. Now if P and Q are known in advance, one can mimic portions of the proof of Proposition 9.3.1. More precisely, the three-point-ladder costs one xDBL per bit. Now changing this cost to the previous steps, we end with $\log_2(\ell^{e-i})$xDBL for the three-point-ladder computing $([\ell^i]P) + [m \mod \ell^{e-i}]([\ell^i]Q))$. Notice that as P and Q are know in advance, in this case the points $[\ell^i]P$ and $[\ell^i]Q$ can be precomputed off-line giving us the desired cost. □

Remark 26. In fact the above result is an upper bound because for scalar multiplications by $\ell = 3, 4, 5$, there exist formulas with a cost less than $1.5\log_2(\ell)$ xDBL. Moreover, depending on the specific setting, one can pre-compute off-line point multiples that may lead to a further reduction of the computational cost given in Proposition 9.3.1.

Now the Proposition 9.3.3 is extended to include isogenies of degree different than 4.

Proposition 9.4.2. Let P_B, Q_B, m_B, R_B be the public and private keys of Bob where $R_B = P_B + [m_B]Q_B$, and let k be the number of cores available to compute a d^{e_B}-isogeny where d is a prime number different than 2. Let p_d be the cost of computing a point multiplication-by-d, q_d be the cost of a degree-d isogeny evaluation and p_2 the cost of computing a multiplication-by-2. Then, one can compute a d^b-isogeny by means of a parallel strategy St_b that uses $k - 1$ cores, at the same time that one core is devoted to compute R_B (or $\phi_A(R_B)$), where b is given as,

$$b = \max_{i:=1,2,\ldots,\frac{e_B-1}{3}} \{i \mid C^{k-1}(St_i) + \log_2(d^i) \cdot p_2 + i \cdot q_d \leq \log_2(d^{n_B})p_2\}.$$

For the Key Generation phase; and

$$b = \max_{i:=1,2,\ldots,\frac{e_B-1}{2}} \{i \mid C^{k-1}(St_i) + 2\log_2(d^i)\cdot p_2 + (e-i)\cdot p_d + i\cdot q_d \leq 2\cdot \log_2(d^{n_B})p_2\}.$$

For the Key Agreement phase.

9.4.1 Low-level implementation notes

Several low-level design aspects and implementation notes are described next.

Synchronization effort for computing Figure 9.3 in parallel

In practice, the implementation of Figure 9.3 requires a delicate synchronization effort due to the different tasks that each core must execute.

Figure 9.5 depicts the computational flow of k threads concurrently evaluating the isogeny St_{e_A} shown in Figure 9.3. One core is assigned to the computation of the point R_A. Meanwhile, the rest of the $k-1$ cores evaluate the lower subtriangle St_b of Figure 9.3. At each iteration, one column of the subtriangle St_b of Figure 9.3 is processed. After performing b iterations, the image point $\phi(R_A)$ is sequentially computed. Then, a similar flow is performed but this time using all the k available cores. Barriers mechanisms are used abundantly to ensure the correct synchronization of the k threads.

Precomputation and memory requirements

In the following, low-level details of the Montgomery ladders computations associated to the key generation points R_A and $[4^{e_A-b}]R_A$ using pre-computed tables are given.

Introduced in [80], the coefficients μ_i, for $i = 1,\ldots,n$ are the most efficient pre-computation technique reported in the literature for n-step Montgomery ladders. If one wants to compute the scalar multiplication $R = kP$, these coefficients are defined as,

$$\mu_i = \frac{x_i+1}{x_i-1}, \text{ where } x_i = x([2^i]P) \tag{9.3}$$

In the context of SIDH, since the x-coordinate of the SIDH points are defined over $\mathbb{F}_p$, so are the coefficients μ_i. Hence, a total of $2e_A$ and $\log_2 3e_B$ field elements for Alice and Bob, must be pre-computed and stored.

Following the approach reported in [80], a Montgomery Ladder step can be compute at cost of $(3\mathbf{M} + 2\mathbf{S})$ field operations.[6] Hence, the computational cost of calculating the secret points R_A and R_B is of $2e_A(3\mathbf{M}+2\mathbf{S})$ and $\log_2 3 \cdot e_A(3\mathbf{M}+2\mathbf{S})$, respectively.

[6] Here **M** and **S** stand for a field multiplication and a field squaring, respectively.

	Alice	Bob
Memory	376 field elements (69KB)	$383 + \log_2(3^b)$ field elements (about 82KB)
Savings	$372(5\mathbf{M} + 2\mathbf{S})$	$379(5\mathbf{M} + 2\mathbf{S})$

Table 9.1: Precomputation memory costs (in kilo bytes) and field operation savings when using the SIKE prime p_{751}. **M** stands for a field multiplication and **S** for a field squaring.

The point $[4^{e_A-b}]R_A$ (cf. propositions 9.3.2-9.3.3) can be computed as

$$[4^{e_A-b}]R_A = [4^{e_A-b}](P_A + [m_A \mod 4^b]Q_A)$$

Since the μ_i values have been already stored for the computation of R_A, one can compute $P_A + [m_A \mod 4^b]Q_A$ re-using those coefficients. Moreover, since P_A and Q_A are public parameters, one can also compute $[4^{e_A-b}]R_A$ as

$$[4^{e_A-b}]R_A = [4^{e_A-b}]P_A + [m_A \mod 4^b]([4^{e_A-b}]Q_A) \tag{9.4}$$

Thus, for an efficient computation of the Montgomery ladder associated to Equation 9.4, one re-uses the coefficients μ_{i+e_A-b}, which have been already stored, the pre-computed points $[4^{e_A-b}]P_A$ and $[4^{e_A-b}](P_A - Q_A)$ and the scalar $m_A \mod 4^b$. This approach only adds two XZ-Proyective points to the pre-computed table (equivalent to four field elements).

The derivation described above, is completely analogous for computing R_B, Bob's secret point in the key generation phase. However, Bob's coefficients μ_i for the computation of the multiple $[3^{e_B-b}]R_B$ cannot be reused, due to the fact that

$$\{[2^i]([3^{b_3}]Q_3)|i{\in}[1..\lceil b\log_2(3)\rceil]\}{\not\subset}\{[2^i]Q_3|i{\in}[1..\lceil e_3\log_2(3)\rceil]\}$$

Hence, one needs to store $\lceil b\log_2(3)\rceil$ extra μ-coefficients to compute a three-point-ladder as in Eq. (9.4).

Table 9.1 reports the memory expenses associated to the pre-computation efforts for Alice and Bob discussed here, along with the computational savings in term of field operations associated to them.

9.5 Cost estimates and experimental results

In this section we present concrete cost estimates and experimental results associated to the execution of SIDH and SIKE when they are instantiated with the SIKE prime p_{751}. We also include in our experiments the Extended-SIDH protocol presented in Chapter 8 instantiated with the eSIDH prime p_{765}.

We begin by giving cost estimates for performing the key agreement phase of SIDH using the parallel tricks discussed in §§ 9.2-9.3. Then, we present experimental results for performing the key agreement phase of SIDH and the three main phases of SIKE, namely, Key generation, Encapsulation and Decapsulation.

We benchmarked our software on an Intel(R) Core(TM) i7-6700K processor at 4.00GHz supporting the Skylake micro-architecture. To guarantee the reproducibility of our measurements, the Intel Hyper-Threading and Intel Turbo Boost technologies were disabled. We used the OpenMP v4.5 API for parallel tasks and POSIX threads. Our source code was compiled using Clang v6.0 with the `-O3` optimization flag and using the options `-mbmi2 -madx -fwrapv -fomit-frame-pointer -fopenmp -pthread`. Our software library is freely available from,

https://github.com/dcervantesv/eSIDH

9.5.1 Cost estimates

The cost estimates and experimental results presented in this section focus on two case studies,

- SIKE Prime $p_{751} = 4^{186}3^{239} - 1$.

- $e_4 = 186$,
- $p_4 = 6186 \cdot 2$,
- $q_4 = 8407$,
- $r_4 = 3691$,
- $e_3 := 239$
- $p_3 := 11999$
- $q_3 := 5944$
- $r_3 := 5720$

- $\mathbb{F}_{p_{751}^2}$ inversion $= 317655$

Where r_4 is the cost of constructing a degree-4 isogenous curve. All the costs above are given in Skylake clock cycles.

Tables 9.2, 9.3, 9.4 and 9.5 show our cost estimates for Alice's and Bob's SIDH key Generation, and Alice's and Bob's SIDH key agreement phases using the SIKE prime p_{751}, respectively. The estimates reported in those tables were organized as follows. The first column gives the number of cores used by the parallel strategy. The second and third columns show the equivalent timing cost associated to the computation of degree-4 isogeny evaluations when using a single and k cores, respectively. The unit of measure for these costs are given in terms of equivalent isogeny evaluations. The fourth column indicates the number of scalar multiplications performed by both, the single-core and the multi-core processors. The fifth column reports the value b as defined in Propositions 9.3.3 and 9.4.2. This parameter indicates the height of the lower subtriangle in Figure 9.3. The sixth column reports the expected computational cost of Alice's (resp. Bob's) key generation

	Evaluations					
Cores	Serial	Parallel	Muls	b	Cost	AF
1	784	—-	636	0	22.45	1
2	929	531	490	32	15.13	1.48
3	1128	447	423	38	12.39	1.81
4	1266	390	394	41	11.27	1.99
5	1395	356	373	43	10.75	2.08
8	1855	313	325	47	9.84	2.28
10	1988	283	317	48	9.51	2.35
61	9519	247	184	62	7.69	2.91
122	9519	184	184	62	7.18	3.12
185	9519	184	184	62	7.18	3.12

Table 9.2: Estimate costs (in millions of clock cycles) of Alice's Key Generation SIDH phase for the prime p_{751}. The AF column is the quotient of the Single core cost and the parallel cost using k cores. The parameter b is given as defined in Proposition 9.3.3

(resp. key agreement) phase (given in millions of Skylake clock cycles), including the expenses associated to walking across St_{186} (resp. St_{239}) in both phases, plus the cost of computing the x-only-coordinate public points (one quadratic field inversion) in the key generation phase, the expenses of obtaining in the Key Agreement phase the Montgomery constant for the curve E_A (E_B) and the j-Invariant of the curve $E_{AB}(E_{BA})$ (essentially two field inversions). The cost of evaluating $\phi_A(R_B)(\phi_B(R_A))$ in Alice's (Bob's) Key Generation (by computing $3e_A(e_B)$ 4-(3-)isogeny evaluations). The expenses of computing 186 degree-4 (resp. 239 degree-3) isogenies in both Alice's phases (resp. Bob's phases). Finally, the seventh column reports the Acceleration Factor (AF) achieved by the parallel strategy compared against a sequential single-core implementation.

The estimates given in Tables 9.2, 9.3, 9.4 and 9.5 theoretically predict that an acceleration factor of 2.02 is achievable provided that a 10-core processing unit is available to compute a SIKE prime p_{751} instantiation of SIDH. Also, it is observed that the maximum AF for Alice in both phases is achieved when 122 cores are available. For Bob's key generation and key agreement phases, the maximum parallelism is achieved when 167 cores and 179 cores are available, respectively. By considering those core numbers we can observe that if there are 179 cores availables to compute SIDH protocol, the maximum AF that one can achieve using the approach discussed here is 2.64.

9.5.2 Experimental results

Table 9.6 presents a comparison of estimated versus experimental costs for the computation of the key agreement phase of SIDH, instantiated with the prime p_{751}. The data in this table was organized as follows. The first column reports the number k of cores. The second

	Evaluations					
Cores	Serial	Parallel	Muls	b	Cost	AF
1	1226	—-	755	0	24.63	1
2	1529	863	581	37	17.03	1.44
3	1812	701	521	43	14.24	1.73
4	2085	621	484	46	13.10	1.87
6	2638	542	434	50	12.07	2.04
8	2944	474	413	52	11.44	2.15
10	3187	427	403	54	11.04	2.22
57	16881	417	237	67	9.12	2.69
71	16584	358	237	70	8.77	2.80
167	16584	237	237	70	8.05	3.05
238	16584	237	237	70	8.05	3.05

Table 9.3: Estimate costs (in millions of clock cycles) of Bob's Key Generation SIDH phase for the prime p_{751}. The AF column is the quotient of the Single core cost and the parallel cost using k cores. The parameter b is given as defined in Proposition 9.4.2

	Evaluations					
Cores	Serial	Parallel	Muls	b	Cost	AF
1	784	—-	636	0	20.06	1
2	929	531	490	32	14.88	1.34
3	1128	447	423	38	13.43	1.49
4	1266	390	394	41	12.67	1.58
6	1565	339	353	45	11.80	1.69
8	1855	313	325	47	11.29	1.77
10	1988	283	317	48	10.97	1.82
28	4243	244	256	55	10.02	2.00
61	9519	247	184	62	9.27	2.16
122	9519	184	184	62	8.76	2.29
185	9519	184	184	62	8.76	2.29

Table 9.4: Estimate costs (in millions of clock cycles) of Alice's Key Agreement SIDH phase for the prime p_{751}. The AF column is the quotient of the Single core cost and the parallel cost using k cores. The parameter b is given as defined in Proposition 9.3.3

	Evaluations					
Cores	Serial	Parallel	Muls	b	Cost	AF
1	1226	—-	755	0	22.72	1
2	1518	847	590	31	17.42	1.30
3	1864	714	519	36	15.85	1.43
4	2062	612	492	39	14.95	1.51
6	2675	548	436	42	13.95	1.62
8	3003	481	414	44	13.31	1.70
10	3259	434	404	45	12.94	1.75
44	11568	385	285	56	11.33	2.00
56	16032	408	248	58	11.05	2.05
60	17880	415	237	58	10.98	2.06
179	17880	237	237	58	9.92	2.28
238	17880	237	237	58	9.92	2.28

Table 9.5: Estimate costs (in millions of clock cycles) of Bob's Key Agreement SIDH phase for the prime p_{751}. The AF column is the quotient of the Single core cost and the parallel cost using k cores. The parameter b is given as defined in Proposition 9.4.2

# of cores	Parallel Strategy	including R	Serial Strategy
1	19.55	19.55	19.55
2	17.02	15.16	17.25
3	15.7	13.86	16.55

Table 9.6: Estimated Vs. experimental costs for the computation of the key agreement phase of SIDH instantiated with the prime p_{751}. All estimates and experimental results are given in 10^6 clock cycles.

and third columns report the estimated and experimental SIDH key agreement costs using parallel strategies, with and without the computation of multiples of the secret point R in parallel, respectively. The last column reports the estimated costs of a parallelized version of SIDH using the sequential optimal strategies of [32].

The relatively complex synchronization of the core loads, has so far prevented us to experimentally achieve the expected theoretical speedups for $k = 2, 3$. Nonetheless, our experiments show that the parallel strategies reported in Appendix A.2 yields an acceleration factor of 1.14, and 1.24 when using two and three cores respectively when compared with a single core implementation. Including the trick of computing the multiples of R in parallel, provides an acceleration factor of 1.28 and 1.41 when using two and three cores respectively, again when compared with a single core implementation. Our method achieves an acceleration factor of 1.13 and 1.19 when compared with a parallel version of SIDH without any of the improvements included in this thesis using two and three cores respectively.

Using the parallelization techniques discussed in §§9.2-9.3, we also implemented all three phases of the SIKE protocol instantiated with the SIKE prime p_{751} [3] and the eSIDH prime p_{765} Chapter 8. In all of our comparisons, we use a sequential SIDH implementation instantiated with the prime p_{751} as a baseline. As reported in Table 9.7, we implemented all four phases of SIDH using the parallelization techniques discussed in §§9.2-9.3. For a three-core implementation of eSIDH (Chapter 8), an acceleration factor of about 1.56 was observed. Also in Table 9.7, one can observe that our three-core implementation of SIDH for the prime p_{751}, which included the parallel computation of the points $R_A, R_B, \phi_B(R_A)$ and $\phi_A(R_B)$ achieved an acceleration factor of 1.46.

In all of our comparisons, we use a sequential SIKE implementation instantiated with the prime p_{751} as a baseline. Table 9.8 reports that a three-core implementation of SIKE instantiated with the prime p_{765} achieves an acceleration factor of 1.56. Similarly, an acceleration factor of 1.45 is achieved when our approach for a three-core SIKE implementation using the prime p_{751} is chosen.

9.6 Conclusion

In this work, we presented a framework that permits an acceleration of the execution of the SIDH and SIKE protocols when they are executed on multi-core platforms. Our approach combines the concurrent computation of degree-ℓ^e isogenies and three-point Montgomery ladders. Our experiments shows that compared against their sequential counterparts, our proposed SIDH and SIKE parallel variants achieve important acceleration factors.

It appears that there exist several other parallelization opportunities that were not considered in this work. For example, we did not consider the design decision of reserving $k - 1$ cores for the computation of the vertices $(0, i)$ of the subtriangle Δ_b in Figure 9.3. One can then compute in parallel the intermediate points of the strategy associated to Δ_b, which would produce a reduction in its computational cost. Since this approach appears

	p_{751}			p_{765}		
Phase	**Number of cores**			**Number of cores**		
	1	2	3	1	2	3
Alice KeyGen	23.59	16.73	15.26	22.27	15.93	14.80
Bob KeyGen	26.74	18.97	17.55	24.34	17.76	15.79
Alice KeyAg	19.37	15.04	13.91	18.21	14.30	13.07
Bob KeyAg	22.76	18.15	16.62	23.24	17.16	15.94
Total	92.46	68.89	63.34	88.05	65.15	59.06

Table 9.7: SIDH experimental timings for a SIKE prime p_{751} and an eSIDH prime p_{765} instantiation. All timings are given in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

	p_{751}			p_{765}		
Phase	**Number of cores**			**Number of cores**		
	1	2	3	1	2	3
KeyGen	26.71	19.10	17.62	24.78	17.71	15.93
Encaps	43.01	31.96	29.86	40.43	29.95	27.64
Decaps	46.34	35.05	32.34	45.58	32.92	30.79
Total	116.06	86.11	79.82	110.79	80.58	74.36

Table 9.8: SIKE experimental timings for a SIKE prime p_{751} and an eSIDH prime p_{765} instantiation. All timings are given in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

to be a non-trivial design task, we leave this option as a future work.

Chapter 10

Summary and conclusions

> “We are nothing more than the sum of our memories and experiences”
>
> Michael Scott, *The Sorceress*

This document was intended to show how isogenies can be used in public-key cryptography. There are other uses of isogenies in cryptography like point counting [62], but our principal goal was to improve the performance of Elliptic curves protocols and isogeny based ones. We also study the hard problem associated with the SIDH protocol and provide a concrete analysis of the used parameters.

We presented a complete taxonomy of Binary elliptic curves over $\mathbb{F}_4$ classifying it into four classes, providing class three with an efficiently computable endomorphism. This endomorphism allows our curve to compete with the current state-of-the-art curves in performance but offering more security. Also, this curve can use the well know τ-NAF scalar multiplication of Koblitz curves over $\mathbb{F}_2$.

We present an analysis on how to attack SIDH protocol using the Van Oorschot-Wiener golden collision search. Such analysis derives a reduction of the primes used, that is, the assumed security was overestimated. This reduction in parameters implies a direct speed-up to the protocol because isogeny computation and private key generation depend on the prime bit-size. We study how to improve the isogeny computation of isogenies of degree $d > 3$ using twisted Edwards curves. We specialize the case for $d = 5, 7$ and in general for integers of the form $d = 8k + q$. Furthermore, we also study how to take advantage of parallel computing to reduce SIDH latency. We present an algorithm able to compute optimal parallel strategies for SIDH taking the benefit of parallel computing. This allows us to present another parallelism choice for SIDH, our eSIDH, which use isogenies of degree three and five for Bob's side. Our eSIDH can be seen as a particular case of a strategy, but the inclusion of two primes for Bob allows a new way to parallelize the private key computation giving Bob's computations a speed up. As a novel inclusion, we discovered that we could also compute multiples of the private key, used in isogeny computation, in parallel. Both private key computation and strategies can be mixed deriving into a powerful strategy.

As conclusión of this thesis we have the following statements

- Scalar multiplication can be improved on a particular set of Koblitz curves over $\mathbb{F}_4$ making those curves competitive when compared with state-of-the-art curves.
- At the moment of this thesis, the SIDH protocol classical security is well defined using the Van-Oorschot -Wiener attack to solve the CSSI problem by considering the storage limits to make an attack feasible.
- SIDH protocol can be benefit from other programming paradigms such as parallel computing to improve the performance of the protocol by taking advantage of the architecture and parallel algorithms (such as the ones using in parallel strategy constructions).
- SIDH protocol can be modified to take advantage of the hardware and parallel algorithms to improve the performance. As evidence, we present in this thesis the eSIDH protocol.
- There are other isogeny-based protocols that can be secured in terms of implementation like our proposal for CSIDH implementation.

10.1 Future work

Despite we try to cover different aspects of isogeny-based protocols, as there is a new field in cryptography there are new isogeny-based protocols that can be improved using our algorithms and other problems that for limitations on the scope of this thesis can not be covered. In the following, we list some points that can be cover in the future as an extension of this thesis.

- To search for efficient isogenies for Koblitz curves over other extensions like $\mathbb{F}_8$ or $\mathbb{F}_{16}$.
- To implement and compare the curves in normal form. As a recall, those curves have the benefit of faster addition formulas but represent the points using 4 coordinates which could derive in a slow performance in implementation.
- In this thesis, we present two tricks to improve both (but separate) Alice and Bob each one taking benefit of parallel computing in different ways, but the trick associated with Alice is independent to the degree of Alice´s isogenies thus it could be applied in both Bob phases in eSIDH. We left as a future work the study of the performance of merging both algorithms and the possible implementation of it.

10.2 Publications

As a part of this thesis we produce the following papers:

- Thomaz Oliveira, Julio López, Daniel Cervantes-Vázquez, Francisco Rodríguez-Henríquez,*Koblitz Curves over Quadratic Fields.* J. Cryptology 32(3), p867-894 (2019).[79]
- Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, Francisco Rodríguez-Henríquez, *On the Cost of Computing Isogenies Between Supersingular Elliptic Curves.* SAC 2018, p322-343. [1]
- Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, Benjamin Smith:,*Stronger and Faster Side-Channel Protections for CSIDH.* Progress in Cryptology – LATINCRYPT 2019, pages 173–193, Cham, 2019. Springer International Publishing. [16]
- Daniel Cervantes-Vázquez, Eduardo Ochoa-Jiménez, Francisco Rodríguez-Henríquez. *Parallel strategies for SIDH: Towards computing SIDH twice as fast.* IEEE Transactions on Computers (To be published)
- Daniel Cervantes-Vázquez, Eduardo Ochoa-Jiménez, Francisco Rodríguez-Henríquez.*eSIDH: the revenge of the SIDH.* IET Information Security (To be published)

BIBLIOGRAPHY

[1] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. Cryptology ePrint Archive, Report 2018/313, 2018. https://eprint.iacr.org/2018/313.

[2] Diego F. Aranha, Armando Faz-Hernández, Julio López, and Francisco Rodríguez-Henríquez. Faster Implementation of Scalar Multiplication on Koblitz Curves. In *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 177–193. Springer, 2012.

[3] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation. second round candidate of the nist's post-quantum cryptography standardization process, 2017. Available at: https://sike.org/.

[4] Daniel J. Bernstein, Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?. In: Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems, 2009. Available from https://cr.yp.to/papers.html#collisioncost.

[5] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In Serge Vaudenay, editor, *Progress in Cryptology – AFRICACRYPT 2008*, pages 389–405, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Peter Schwabe. Kummer strikes back: New DH speed records. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 317–337. Springer, 2014.

[7] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM*

SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pages 967–980, 2013.

[8] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 409–441, 2019.

[9] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. *IACR Cryptology ePrint Archive*, 2019:498, 2019.

[10] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492. Internet Engineering Task Force (IETF), 2006. https://tools.ietf.org/html/rfc4492.

[11] Manuel Bluhm and Shay Gueron. Fast software implementation of binary elliptic curve cryptography. *J. Cryptographic Engineering*, 5(3):215–226, 2015.

[12] Joppe W. Bos and Simon Friedberger. Arithmetic considerations for isogeny-based cryptography. *IEEE Trans. Computers*, 68(7):979–990, 2019.

[13] G. Brassard, P. Høyer and A. Tapp, Quantum cryptanalysis of hash and claw-free functions. *Latin American Symposium on Theoretical Informatics — LATIN'98*, LNCS 1380 (1998), 163–169.

[14] Wouter Castryck, Steven D. Galbraith, and Reza Rezaeian Farashahi. Efficient arithmetic on elliptic curves using a mixed edwards-montgomery representation. Cryptology ePrint Archive, Report 2008/218, 2008. https://eprint.iacr.org/2008/218.

[15] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 395–427, 2018.

[16] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for csidh. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, pages 173–193, Cham, 2019. Springer International Publishing.

[17] Daniel Cervantes-Vázquez, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A [magical] parallel variant of SIDH. CHES 2018 Rump session, 2018. https://ches.iacr.org/2018/slides/ches2018-rump-talk19-slides.pdf.

[18] Daniel Cervantes-Vázquez and Francisco Rodríguez-Henríquez. A note on the cost of computing odd degree isogenies. Cryptology ePrint Archive, Report 2019/1373, 2019. https://eprint.iacr.org/2019/1373.

[19] Denis Charles, Eyal Goren, and Kristin Lauter. Cryptographic hash functions from expander graphs. Cryptology ePrint Archive, Report 2006/021, 2006. http://eprint.iacr.org/2006/021.

[20] Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014.

[21] Craig Costello. B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. Cryptology ePrint Archive, Report 2019/1145, 2019. https://eprint.iacr.org/2019/1145.

[22] C. Costello et al., SIDH Library, https://www.microsoft.com/en-us/research/project/sidh-library/.

[23] Craig Costello and Huseyin Hisil. A simple and compact algorithm for sidh with arbitrary degree isogenies. Cryptology ePrint Archive, Report 2017/504, 2017. https://eprint.iacr.org/2017/504.

[24] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 679–706, 2017.

[25] Craig Costello and Patrick Longa. Four($\mathbb{Q}$): Four-Dimensional Decompositions on a ($\mathbb{Q}$)-curve over the Mersenne Prime. In *ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 214–235. Springer, 2015.

[26] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 572–601. Springer, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[27] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *J. Cryptographic Engineering*, 8(3):227–240, 2018.

[28] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. http://eprint.iacr.org/2006/291.

[29] Jean Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006.

[30] Thinh Dang and Dustin Moody. Twisted hessian isogenies. Cryptology ePrint Archive, Report 2019/1003, 2019. https://eprint.iacr.org/2019/1003.

[31] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. Cryptology ePrint Archive, Report 2018/824, 2018.

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

[33] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 365–394, 2018.

[34] Cyprien Delpech de Saint Guilhem, Péter Kutas, Christophe Petit, and Javier Silva. SÉta: Supersingular encryption from torsion attacks. Cryptology ePrint Archive, Report 2019/1291, 2019. https://eprint.iacr.org/2019/1291.

[35] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. Faster SeaSign signatures through improved rejection sampling. to appear at PQCrypto 2019, 2019.

[36] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. Internet Engineering Task Force (IETF), 2008. https://tools.ietf.org/html/rfc5246.

[37] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[38] Armando Faz-Hernández, Julio López Hernandez, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol. *IEEE Trans. Computers*, 67(11):1622–1636, 2018.

[39] William Fulton. *Algebraic curves.* Addison-Wesley, 1969.

[40] Steven D. Galbraith, C. Petit and J. Silva, Identification protocols and signature schemes based on supersingular isogeny problems. *Advances in Cryptology — ASIACRYPT 2017*, LNCS 10624 (2017), 3–33.

[41] Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, pages 93–106, 2017.

[42] Lov Grover, A fast quantum mechanical algorithm for database search. *Proceedings of the Twenty-Eighth Annual Symposium on Theory of Computing — STOC '96*, ACM Press (1996), 212–219.

[43] Shay Gueron and Vlad Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, pages 1–11, 2014.

[44] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Secaucus, NJ, USA, 2003.

[45] Robin Hartshorne. *Algebraic curves*. Springer Verlag, 1977.

[46] Aaron Hutchinson and Koray Karabina. Constructing canonical strategies for parallel implementation of isogeny based cryptography. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018*, volume 11356 of *Lecture Notes in Computer Science*, pages 169–189. Springer, 2018.

[47] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of csidh: A systematic approach to efficient strategies, permutations, and bound vectors. Cryptology ePrint Archive, Report 2019/1121, 2019. https://eprint.iacr.org/2019/1121.

[48] Amir Jalali, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. Towards optimized and constant-time CSIDH on embedded devices. In *Constructive Side-Channel Analysis and Secure Design*, pages 215–231. Springer International Publishing, 2019.

[49] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, pages 19–34, 2011.

[50] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. *Advances in Cryptology – CRYPTO 2019*, Springer International Publishing, pages 32–61, 2019.

[51] D. Jao and V. Soukharev, Isogeny-based quantum-resistant undeniable signatures. *Post-Quantum Cryptography — PQCrypto 2014*, LNCS 8772 (2014), 160–179.

[52] Miroslav Knezevic, Frederik Vercauteren, and Ingrid Verbauwhede. Speeding up bipartite modular multiplication. In M. Anwar Hasan and Tor Helleseth, editors, *Arithmetic of Finite Fields, Third International Workshop, WAIFI 2010, Istanbul, Turkey, June 27-30, 2010. Proceedings*, volume 6087 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2010.

[53] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of computation*, 48:203–9, 1987.

[54] N. Koblitz. CM-Curves with Good Cryptographic Properties. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 279–287. Springer, 1991.

[55] Neal Koblitz and Alfred Menezes. A Riddle Wrapped in an Enigma. Cryptology ePrint Archive, Report 2015/1018, 2015. http://eprint.iacr.org/2015/1018.

[56] David Kohel. Efficient arithmetic on elliptic curves in characteristic 2. In *Progress in Cryptology - INDOCRYPT 2012*, volume 7668 of *LNCS*. Springer, Berlin, Heidelberg, 2012.

[57] David Kohel. Twisted μ_4-normal form for elliptic curves. Cryptology ePrint Archive, Report 2017/121, 2017. https://eprint.iacr.org/2017/121.

[58] Brian Koziel, A.-Bon Ackie, Rami El Khatib, Reza Azarderakhsh, and Mehran Mozaffari Kermani. Sike'd up: Fast and secure hardware architectures for supersingular isogeny key encapsulation. *IACR Cryptol. ePrint Arch.*, 2019:711, 2019.

[59] Brian Koziel, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. Post-quantum cryptography on FPGA based on isogenies on elliptic curves. *IEEE Trans. on Circuits and Systems*, 64-I(1):86–99, 2017.

[60] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, pages 191–206. Springer International Publishing, 2016.

[61] Brian Koziel, Amir Jalali, Reza Azarderakhsh, David Jao, and Mehran Mozaffari Kermani. NEON-SIDH: efficient implementation of supersingular isogeny diffie-hellman key exchange protocol on ARM. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016*, volume 10052 of *Lecture Notes in Computer Science*, pages 88–103, 2016.

[62] H. W. Lenstra, Factoring Integers with Elliptic Curves, *Annals of Mathematics*, Annals of Mathematics 126 (1987), 649-673.

[63] Patrick Longa and Francesco Sica. Four-Dimensional Gallant-Lambert-Vanstone Scalar Multiplication. *J. Cryptology*, 27(2):248–283, 2014.

[64] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of The American Mathematical Society - BULL AMER MATH SOC*, 44:393–423, 07 2007.

[65] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In *Post-Quantum Cryptography - 10th International Workshop, PQCrypto 2019*, 2019.

[66] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, pages 137–152, 2018.

[67] V. Miller. Uses of Elliptic Curves in Cryptography. In *CRYPTO 85*, volume 218 of *LNCS*, pages 417–426. Springer, 1985.

[68] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–234, 1987.

[69] Dustin Moody and Daniel Shumow. Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. *Mathematics of Computation*, 85(300):1929–1951, 2016.

[70] National Institute of Standards and Technology. Recommended Elliptic Curves for Federal Government Use. NIST Special Publication, 1999. http://csrc.nist.gov/csrc/fedstandards.html.

[71] National Institute of Standards and Technology. FIPS PUB 186-4: Digital Signature Standard (DSS). Federal Information Processing Standards, 2013. nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[72] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

[73] National Security Agency. The case for elliptic curve cryptography, October 2005. tinyurl.com/NSAandECC.

[74] NIST. NIST Post-Quantum Cryptography Standardization Process. Second Round Candidates, 2017. Available at: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[75] José Eduardo Ochoa-Jiménez. *Analysis and efficient implementation of public key cryptographic protocols*. PhD thesis, CINVESTAV-IPN, February 2019. Available at: http://delta.cs.cinvestav.mx/~francisco/je.pdf.

[76] Thomaz Oliveira, Diego F. Aranha, Julio López, and Francisco Rodríguez-Henríquez. Fast Point Multiplication Algorithms for Binary Elliptic Curves with and without Precomputation. In Antoine Joux and Amr Youssef, editors, *Selected Areas in Cryptography – SAC 2014: 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, pages 324–344, Cham, 2014. Springer International Publishing.

[77] Thomaz Oliveira, Diego F. Aranha, Julio López, and Francisco Rodríguez-Henríquez. Improving the performance of the GLS254. Presentation at CHES 2016 rump session, 2016.

[78] Thomaz Oliveira, Julio López, Diego F. Aranha, and Francisco Rodríguez-Henríquez. Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014.

[79] Thomaz Oliveira, Julio López, Daniel Cervantes-Vázquez, and Francisco Rodríguez-Henríquez. Koblitz curves over quadratic fields. *Journal of Cryptology*, 32(3):867–894, Jul 2019.

[80] Thomaz Oliveira, Julio César López-Hernández, Hüseyin Hisil, Armando Faz-Hernández, and Francisco Rodríguez-Henríquez. How to (pre-)compute a ladder - improving the performance of X25519 and X448. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017*, volume 10719 of *Lecture Notes in Computer Science*, pages 172–191. Springer, 2017.

[81] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. A faster constant-time algorithm of CSIDH keeping two torsion points. To appear in IWSEC 2019 – The 14th International Workshop on Security, 2019.

[82] Hiroshi Onuki and Tsuyoshi Takagi. On collisions related to an ideal class of order 3 in csidh. Cryptology ePrint Archive, Report 2019/1209, 2019. https://eprint.iacr.org/2019/1209.

[83] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2017.

[84] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[85] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006.

[86] C. Schnorr and A. Shamir, An optimal sorting algorithm for mesh connected computers. *Proceedings of the Eighteenth Annual Symposium on Theory of Computing — STOC '86*, ACM Press (1986), 255–263.

[87] R. Schoof, Nonsingular plane cubic curves over finite fields, *Journal of Combinatorial Theory, Series A*, 46 (1987), 183–211.

[88] Hwajeong Seo, Zhe Liu, Patrick Longa, and Zhi Hu. SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):1–20, 2018.

[89] Igor R. Shafarevich. *Basic Algebraic Geometry.* Springer-Verlag, 1977.

[90] A. Shamir, Factoring large numbers with the TWINKLE device. *Cryptographic Hardware and Embedded Systems — CHES 1999*, LNCS 1717 (1999), 2–12.

[91] A. Shamir and E. Tromer, Factoring large numbers with the TWIRL device. *Advances in Cryptology — CRYPTO 2003*, LNCS 2729 (2003), 1–26.

[92] Joseph H Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag New York, 2 edition, 2009.

[93] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. in Math. of Comm.*, 4(2):215–235, 2010.

[94] S. Tani, Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410 (2009), 5285–5297.

[95] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones Mathematicae*, 22:134—-144, 1966.

[96] J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López. Software Implementation of Binary Elliptic Curves: Impact of the Carry-less Multiplier on Scalar Multiplication. In *CHES 2011*, volume 6917 of *LNCS*, pages 108–123. Springer, 2011.

[97] Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, pages 107–122, 2017.

[98] William R. Trost and Guangwu Xu. On the Optimal Pre-Computation of Window τ-NAF for Koblitz Curves. Cryptology ePrint Archive, Report 2014/664, 2014. http://eprint.iacr.org/.

[99] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity.* Elsevier and MIT Press, 1990.

[100] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, Jan 1999.

[101] P. van Oorschot and M. Wiener, Improving implementable meet-in-the-middle attacks by orders of magnitude. *Advances in Cryptology — CRYPTO '96*, LNCS 1109 (1996), 229–236.

[102] Jacques Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris Sér. A-B*, 273:A238–A241, 1971.

[103] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition.* Chapman & Hall/CRC, 2 edition, 2008.

[104] Erich Wenger and Paul Wolfger. Solving the Discrete Logarithm of a 113-Bit Koblitz Curve with an FPGA Cluster. In *SAC 2014*, volume 8781 of *LNCS*, pages 363–379. Springer, 2014.

[105] Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao and V. Soukharev, A post-quantum digital signature scheme based on supersingular isogenies. *Financial Cryptography and Data Security — FC 2017*, LNCS 10322 (2018), 163–181.

[106] C. Zalka, Grover's quantum searching algorithm is optimal. *Physical Review A*, 60 (1999), 2746–2751.

[107] G. Zanon, M. Simplicio Jr., G. Pereira, J. Doliskani and P. Barreto, Faster isogeny-based compressed key agreement. *Post-Quantum Cryptography — PQCrypto 2018*, LNCS 10786 (2018), 248–268.

Appendix A

Node split vectors for the parallel strategies

We report several parallel strategies generated using the algorithms presented in § 9.2.3.

A.1 Strategies for the SIKE prime p_{434}

- 2 cores [39, 21, 18, 10, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 8, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 13, 8, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1]
- 9 cores [10, 10, 10, 10, 10, 10, 10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
- 55 cores [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

The following strategies include the parallel computation of the multiples of R_A:

- 2 cores [86, 10, 5, 3, 2, 1, 1, 1, 1, 2, 1, 1, 1, 5, 2, 1, 1, 1, 2, 1, 1, 1, 28, 19, 13, 8, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 8, 4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 8, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1]

- 20 cores [74, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 21, 14, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

A.2 Strategies for the SIKE prime p_{751}

- 2 cores [57, 42, 28, 21, 12, 8, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 8, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 8, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 14, 8, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 6, 3, 2, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 18, 13, 8, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1]

- 9 cores [20, 18, 18, 18, 14, 10, 10, 10, 10, 10, 10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

- 94 cores [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

The following strategies include the parallel computation of the multiples of R_A:

- 2 cores [152, 16, 9, 4, 2, 1, 1, 1, 2, 1, 1, 4, 2, 1, 1, 1, 2, 1, 1, 7, 4, 2, 1, 1, 1, 2, 1, 1, 3, 2, 1, 1, 1, 1, 44, 39, 21, 18, 10, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 8, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 13, 8, 8, 3, 3, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 5, 3, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 16, 8, 8, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 8, 3, 2, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1]

- 22 cores [131, 14, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 23, 23, 23, 19, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
- 62 cores [123, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]