Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Unidad Zacatenco
Departamento de Computación

Curvas elípticas en la criptografía clásica y post-cuántica

Tesis que presenta

Jesús Javier Chi Domínguez

Para obtener el grado de

Doctor en Ciencias en Computación

Director de tesis

Dr. Francisco José Rambó Rodríguez Henríquez

Ciudad de México Diciembre, 2019

Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Zacatenco Campus
Computer Science Department

Elliptic curves in classical and post-quantum cryptography

Thesis submitted by

Jesús Javier Chi Domínguez

as fulfillment of the requirement for

Doctor in Computer Science

Advisor:

Dr. Francisco José Rambó Rodríguez Henríquez

México City December, 2019

Acknowledgements

I would like to start by thanking the "Consejo Nacional de Ciencia y Tecnología" (CONACyT) for the scholarship they provided me during the period I was a Ph.D. candidate at Computer Sciences Department of the Cinvestav - IPN.

I would like to thank my advisor Francisco Rodríguez-Henríquez for his guidance and constant support.

I would like to thank Sofía Reza, Erika Ríos and Felipa Rosas for their administrative help.

Additionally, I would like to thank my synodals Guillermo Morales-Luna, Cuauhtemoc Mancillas, Juan-Carlos Ku-Cauich, Luis-Julian Domínguez-Perez, and Gina Gallegos-García for their comments about this thesis.

I would like to finish by thanking my wife Fabiola Hernández, son Javier Chi-Hernández, dad Javier Chi, mom Teresa Domínguez, and uncle Moisés Domínguez.

Resumen

En la actualidad, la criptografía de curvas elípticas está jugando un papel muy importante en lo que se refiere a diseños de esquemas criptográficos, cuya clave de cifrado es relativamente pequeña. No obstante, ante la inminente llegada de las computadoras cuánticas, la criptografía de curvas elípticas se ve impactada negativamente por la existencia de un algoritmo cuántico (el algoritmo de Shor), el cual tiene un tiempo de ejecución polinómico.

Debido a esta problemática, la comunidad criptográfica ha sugerido el uso de la criptografía basada en isogenias, la cual sigue basándose en el uso de curvas elípticas pero con la diferencia de que el algoritmo de Shor no peligra su seguridad. Una de las ventajas del uso de isogenias es que permite claves de cifrado pequeñas, aunque son costosos (computacionalmente hablando) con respecto a otros esquemas criptográficos.

Por su parte, mientras las computadoras cuánticas no cuenten con las características necesarias para quebrantar la criptografía de curvas elípticas, sigue siendo de interés determinar que tan segura es esta criptografía. Por ejemplo, el ataque gGHS es usado en la criptografía de curvas elípticas binarias para reducir instancias del Problema del Logaritmo Discreto en una curva elíptica hacia el *jacobiano* de una curva hiperelíptica; se dice que un criptosistema basado en curvas elípticas es vulnerable ante este ataque si es mucho más sencillo resolver esta nueva instancia del Problema del Logaritmo Discreto.

En esta tesis se presenta una implementación en Magma para resolver el problema del logaritmo discreto sobre una curva binaria GLS; construyendo así una curva vulnerable ante el ataque gGHS, junto con la aplicación de algoritmos basados en cálculo de índices, para la resolución del Problema del Logaritmo Discreto en el *jacobiano* de una curva hiperelíptica de género grande; en particular, fue posible demostrar que el endomorfismo asociado a la curva GLS permite acelerar el ataque GHS del descenso de Weil. A su vez se presentan implementaciones eficientes en lenguaje C de i) ataques clásicos al protocolo SIDH, y ii) el protocolo CSIDH para intercambio de claves; ambos protocolos basados en isogenias.

Abstract

Nowadays, the elliptic curve cryptography is playing an important role in the design of cryptographic schemes. Nevertheless, the imminent arrival of quantum computers risk the security of elliptic curves cryptography; that is because Shor's algorithm allows to break it with a polynomial running-time.

Due to this problematic, the cryptographic community has suggested the use of isogenies in cryptography, which is still using elliptic curves such that Shor's algorithm doesn't apply. Additionally, isogeny-based cryptography allows small keys but it is slow compared with other cryptographic schemes.

For its part, before quantum computers break elliptic curve cryptography, it is important to determine whether an elliptic curve is recommended for cryptographic usage. For example, one can use the gGHS Weil descent attack in order to reduce an instance of the Discrete Logarithm Problem on a binary elliptic curve into the Jacobian of a hyperelliptic curve of higher genus; in particular, a cryptosystem based on elliptic curves is called vulnerable against this attack if it is much easier to solve the new instance of the Discrete Logarithm Problem.

In this thesis, it is presented a Magma-code implementation for solving the DLP on a binary GLS curve. For this purpose, a vulnerable curve against the gGHS Weil descent attack was constructed, and then an index-calculus based algorithm was used in order to solve the Discrete Logarithm Problem on the Jacobian of a hyperelliptic curve of a higher genus; furthermore, it was possible to prove that the associated GLS endomorphism allows to speedup the GHS Weil descent attack. Additionally, it is presented efficient C-code implementations of i) classical attacks to SIDH protocol, and ii) CSIDH protocol for key-agreement algorithms; both protocols are based on isogenies.

Contents

List of Figures

List of Tables

List of algorithms and program codes

Chapter 1

Introduction

Pure mathematics is, in its way, the poetry of logical ideas.

Albert Einstein

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. In a basic communication scheme, party **A** (Alice) wants to send a message to party **B** (Bob), so that an adversary **E** (Eve) cannot understand the message.

The current cryptography can be divided into *symmetric-key* (same secret key for encryption and decryption), and *public-key* (a public and private keys for encryption and decryption, respectively [1]) cryptography. The main advantage of symmetric encryption over public-key encryption is that it is fast and efficient for large amounts of data; the disadvantage is the key management necessary to use it, the symmetric ecnryption. Nevertheless, the nature of public-key encryption allows to perform a key exchange in order to obtaining a shared secret-key. A key exchange algorithm can develop it in three different modes: i) static-static mode (Alice and Bob store their private and public keys), ii) static-dynamic mode (Alice and Bob only store their private keys), and iii) ephemeral-static mode (one party will generate a new private and public key every time). The resulting shared secret with static-static and static-dynamic modes will be the same every time but with ephemeral-static mode, new shared secret-key will be generated every time.

In 1976, Whitfield Diffie and Martin Hellman published the first key-exchange protocol [1] that provides the capability for Alice and Bob to agree upon a shared secret between them. Two years later, Adleman, Shamir, and Rivest bring the first public-key cryptosystem, whose security is based on the hardness of factoring large integer numbers [2]. In 1985, Miller [3] and Koblitz [4] proposed to use elliptic curves in public-key cryptosystems, which both base their security on the hardness of computing discrete logarithms over elliptic curves. The

[1] The keys are generated in such a way that it is impossible to derive the private key from the public key.

main advantage of using elliptic curve cryptography is that requires smaller keys compared to the aforementioned; for example, in order to achieve 128-bits of security, the RSA, Diffie-Hellman, and Elliptic Curve cryptosystems require a key of 3072, 3072, and 256 bits, respectively.

Quantum computing: In a classical computer the quintessential information particle, the bit, can only exist in two states, 0 or 1; but this is no the case in a quantum computer, which benefits from the ability of subatomic particles to exist in more than one state simultaneously. To be more precise, quantum bits (qubits) can store much more information because they make direct use of quantum mechanical properties, such as superposition and entanglement. Essentially, while bits can only be 0 or a 1, qubits can assume any superposition of these values. In other words, computational operations can be performed at a much higher speed and with much less power consumption. Certainly, quantum computing is today in its infancy but the global giants such as Intel [1], Google [2], IBM [3], Rigetti [4] and Microsoft [5], are investing heavily in the development of quantum computers (see Table 1.1).

Company	USTC China	Rigetti	Intel	IBM Q	Google
Number of qubits	10	16	49	50	72

Table 1.1: Quantum computers advances in industry.

In 1996, Grover gave a quantum algorithm that finds an input to a function that produces a particular output value, using just $O(\sqrt{N})$ evaluations of the function, where N is the size of the function's domain [5] (the analogous problem in classical computation cannot be solved in fewer than $O(N)$). In 1997, Shor provided a polynomial-time quantum algorithm [6] for factoring integer numbers and computing discrete logarithms. These results and the advances on quantum computer development risk the security of the current public-key cryptography. Therefore, assuming that with a few more years of evolution, quantum computers will reach the point where public-key cryptography can be easily broken. Due to this, the post-quantum cryptography emerges like a solution by proposing the use of classical cryptosystems that are secure against classical and quantum attacks. In 2017, the National Institute of Standards and Technology (NIST) of the U.S. initiated a process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms. The following year, the NIST released its list of first round algorithms as potential standards (69 candidates), and in 2019 this number was reduced for its 2nd-round into 26. However, these 26 candidates base their security on computationally hard problems in lattices, codes, hash function, multivariate polynomials, and isoge-

[1] https://www.intel.com/content/www/us/en/research/quantum-computing.html
[2] https://ai.google/research/teams/applied-science/quantum-ai/
[3] https://www.research.ibm.com/ibm-q/
[4] https://www.rigetti.com/qpu
[5] https://www.microsoft.com/en-us/quantum/

nies. One important fact, is that all of them are ephemeral-static key exchange. The last candidate, isogeny-based cryptography, is in some way, a generalization of the current elliptic curve but basing its security on the hardness of finding "special functions" between two elliptic curves. The main advantage of using isogenies is the smaller keys in comparison with its contenders but the disadvantage is that it is slower.

1.1 Elliptic curve in classical cryptography

Let $q = 2^n$. The set of affine points $(x, y) \in \mathbb{F}_{q^\ell} \times \mathbb{F}_{q^\ell}$ of a binary elliptic curve given by the Weierstrass equation

$$\mathcal{E}/\mathbb{F}_{q^\ell} : y^2 + xy = x^3 + ax^2 + b, \tag{1.1}$$

together with a point at infinity denoted by $\mathcal{O}$, form an abelian group denoted by $\mathcal{E}(\mathbb{F}_{q^\ell})$. The order of this group is written as $\#\mathcal{E}(\mathbb{F}_{q^\ell}) = c \cdot r$. By a careful selection of the constants a, b, the cofactor c is chosen to be a small integer number, whereas r is a large prime. The group law of $\mathcal{E}(\mathbb{F}_{q^\ell})$ is defined by the group point addition operation. Let $\langle \mathsf{P} \rangle$ be a prime order-r subgroup in $\mathcal{E}(\mathbb{F}_{q^\ell})$, and let k be a positive integer such that $k \in [1, r-1]$. The elliptic curve scalar multiplication operation computes the multiple $\mathsf{Q} = k\mathsf{P}$, which corresponds to the point resulting of adding P to itself $k-1$ times.

Galbraith-Lin-Scott (GLS) elliptic curves were introduced in [7]. GLS curves are cryptographically interesting because they come equipped with a two dimensional endomorphism Ψ. Constructively, one can take advantage of this feature by applying the Gallant-Lambert-Vanstone (GLV) approach presented in [25], where the elliptic curve scalar multiplication $\mathsf{Q} = k\mathsf{P}$ splits into two half-sized scalar multiplications such that,

$$\mathsf{Q} = k\mathsf{P} = k_1\mathsf{P} + k_2\psi(\mathsf{P}).$$

The authors of [8] reported a family of binary GLS curves defined over the quadratic field $\mathbb{F}_{q^2}$ that happened to have an almost-prime group order of the form $\#\mathcal{E}_{a,b}(\mathbb{F}_{q^2}) = c \cdot r$, with $c = 2$ and where r is a $(2n-1)$-bit prime number. The software and hardware implementations of constant-time variable-base-point elliptic curve scalar multiplication using binary GLS curves rank among the fastest at the 128-bit security level [9, 10, 11, 12].

Previous work: The Weil descent technique was introduced by Gerhard Frey in 1998 as a means of transferring a Discrete Logarithm Problem (DLP) on an elliptic curve $\mathcal{E}$ defined over the field $\mathbb{F}_{q^\ell}$, into another discrete logarithm problem, this time defined on the Jacobian of a genus-g curve $\mathcal{H}$ that lies in the field $\mathbb{F}_q$ [13]. This transfer becomes useful if the DLP on the Jacobian of the curve $\mathcal{H}/\mathbb{F}_q$ shows lesser computational cost than the DLP on $\mathcal{E}/\mathbb{F}_{q^\ell}$, a situation that usually happens if the genus g of $\mathcal{H}$ is not too large, nor too small (i.e., $g \geq \ell$ and $g \approx \ell$).

Frey's initial construction was further refined by Galbraith, and Smart in [14]. A few years later, Gaudry, Hess and Smart gave an efficient version of the Weil descent technique applied to curves defined over binary extension fields [26]. After that, Galbraith, Hess, and Smart extended the GHS Weil descent method by applying the GHS attack on elliptic curves isogenous to the one where the DLP was originally targeted. This was done in the hope that the GHS attack could possibly be more effective on any of those isogenous elliptic curves [27]. Furthermore, in 2003 Hess generalized the GHS Weil descent attack by allowing the possibility that $\mathcal{C}/\mathbb{F}_q$ could end up being non-hyperelliptic [28, 29]; additionally, Hess showed that the genus of the curve $\mathcal{C}/\mathbb{F}_q$ can be completely determined by finding the roots of polynomials of the form, $\hat{h}(x) = \sum_{i=0}^{d} h_i \cdot x^{q^i} \in \mathbb{F}_p[x]$.

In 2001, Menezes and Qu showed that the GHS Weil descent attack cannot be efficiently applied to characteristic-two fields with prime extensions n that lie in the cryptographically-interesting range $n \in \{160, \ldots, 600\}$ [15]. Moreover, Jacobson, Menezes, and Stein studied the GHS Weil descent attack on elliptic curves defined over $\mathbb{F}_{q^{31}}$ with $q = 2^5$ [33]. This was followed by an analysis of Maurer, Menezes, and Teske, where the authors investigated the feasibility of the GHS Weil descent attack on elliptic curves defined over binary extension fields with composite extensions n within the interval $n \in \{100, \ldots, 600\}$ [16]. In 2009, Hankerson, Karabina and Menezes showed that the Galbraith-Lin-Scott (GLS) binary elliptic curves defined over $\mathbb{F}_{2^{2\ell}}$ are secure against the (generalized)GHS Weil descent attack when $\ell \in \{80, \ldots, 256\} \setminus \{127\}$ is a prime number [8].

Chi and Oliveira presented an efficient algorithm to determine if a given GLS elliptic curve is vulnerable against the GHS Weil descent attack [31]. Finally, Velichka *et al.* presented an explicit computation of a discrete logarithm problem on a hyperelliptic genus-31 curve $\mathcal{H}/\mathbb{F}_{2^5}$ that corresponds with the image of the Weil descent attack of an elliptic curve $\mathcal{E}/\mathbb{F}_{2^{5\times 31}}$ [32].

1.2 Elliptic curve in post-quantum cryptography

Isogeny-based cryptography was introduced by Couveignes [17], 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 [18, 19], who were the first to recognize its potential as a post-quantum candidate.

1.2.1 Supersingular Isogeny Diffie-Hellman

The Supersingular Isogeny Diffie-Hellman (SIDH) key agreement scheme was proposed by Jao and De Feo [54] (see also [49]). It is one of 26 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 [53]. 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 [44] 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 [51, 70] and an undeniable signature scheme [55].

Let p be a prime, let ℓ be a small prime (e.g., $\ell \in \{2, 3\}$), and let $\mathcal{E}$ and $\mathcal{E}'$ be two supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ for which a (separable) degree-ℓ^e isogeny $\phi : \mathcal{E} \to \mathcal{E}'$ defined over $\mathbb{F}_{p^2}$ exists. The CSSI problem is that of constructing such an isogeny. In [49], the CSSI problem is assessed as having a complexity of $O(p^{1/4})$ and $O(p^{1/6})$ against classical and quantum attacks [66], respectively.

1.2.2 Commutative Supersingular Isogeny Diffie-Hellman

Recent efforts to make Isogeny-based cryptosystem practical have put it back at the forefront of research in post-quantum cryptography [96]. A major breakthrough was achieved by Castryck, Lange, Martindale, Panny, and Renes with CSIDH [73], a reinterpretation of Couveignes' system using supersingular curves defined over a prime field. On the positive side, CSIDH has smaller public keys, is based on a better understood security assumption, and supports an easy key validation procedure, making it better suited than SIDH for CCA-secure encryption, static-dynamic and static-static key exchange.

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 [77]. However, both [73] and [77] 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 [73] and [77] are thus vulnerable to simple timing attacks.

The first attempt at implementing CSIDH in constant-time was realized by Bernstein, Lange, Martindale, and Panny [81], 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 [79], and independently by Meyer, Campos, and Reith [78], whose work was improved by Onuki, Aikawa, Yamazaki, and Takagi [82].

The approach of Jalali *et al.* is similar to that of [81], 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 unpro-tected algorithm in [77]. 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 [77].

1.3 Organization of the thesis

The mathematical background required is given in chapter 2, which is composed by §2.1, §2.2, and §2.3 that present the mathematical definitions related with groups and rings, finite fields, and genus-g hyperelliptic curves, respectively.

Chapter 3 is centered on the implications of the gGHS Weil descent technique applied to binary GLS curves, and it is structured as follows. In §3.1, a brief description of the generalized GLS binary curves and their corresponding endomorphisms are given. A concrete formulation of the GLS endomorphism induced in the Weil restriction is presented in §3.2. This is followed in §3.3 by a concrete definition of the GLS endomorphism on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$, which is the main result of this chapter. §3.3 also includes a detailed discussion of the DLP computation on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ by means of a standard index-calculus procedure, and it is shown that the GLS endomorphism provides a factor n acceleration in this DLP computation. Additionally, comparisons of our results with related work are drawn in §3.3, and an efficient root-finding algorithm for linearized polynomials (required in the gGHS weil descent technique) is given in §3.4.

Chapter 4 is focused on the classical security of SIDH protocol, which is organized as follows. The CSSI problem and Vélu's formulas for constructing isogenies are presented in §4.1. This is followed in §4.2 by a simple description of SIDH protocol. In §4.3 and §4.4, 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 §4.5.

Chapter 5 is oriented in an efficient a more secure constant-time C-code implementation of CSIDH protocol. This chapter is ordered as follows. In §5.1.2 we briefly recall ideas, algorithms and parameters from CSIDH [73]. In §5.2 we highlight a shortcoming in [78] and [82] and propose a way to fix it. In §5.3 we introduce new optimizations compatible with all previous versions of CSIDH. In §5.4 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 §5.5 we discuss a more costly variant of CSIDH with stronger security guarantees.

Finally, the concluding remarks, Magma and C codes implementations, and publications are presented and summarized in Chapter 6.

Chapter 2

Mathematical background

Everyone knows what a curve is, until he has studied enough mathematics to become confused through the countless number of possible exceptions.

Felix Klein

Algebraic structures like *groups*, *rings*, *finite fields*, and *homomorphisms* are the bases required for understanding how elliptic curves plays in the cryptographic world. Therefore, let's focus by first in describing these algebraic structures.

2.1 Groups and rings

A *group* $(\mathsf{G}, \star)$, usually written just by G, is determined by a set $\mathsf{G} \neq \emptyset$ and a binary operation $\star\colon \mathsf{G} \times \mathsf{G} \to \mathsf{G}$ that satisfies the following properties:

1. **Closure.** $\forall a, b \in \mathsf{G}$, $a \star b \in \mathsf{G}$.
2. **Associativity.** $\forall a, b, c \in \mathsf{G}$, $(a \star b) \star c = a \star (b \star c)$.
3. **Neutral element.** $\exists! e \in \mathsf{G}\colon \ \forall a \in \mathsf{G}$, $e \star a = a \star e = a$.
4. **Inverse element.** $\forall a \in \mathsf{G}$, $\exists \overline{a} \in \mathsf{G}\colon \ a \star \overline{a} = \overline{a} \star a = e$.

Groups for which the commutativity equation $a \star b = b \star a$ always holds are called *abelian* groups.

Definition 2.1.1. *Let $\mathsf{G} \coloneqq (\mathsf{G}, \star)$ be a group. The order of the group G is defined as the size of the set G. The order of an element $g \in \mathsf{G}$ is defined as the smallest positive integer r (if it exists) such that*

$$\star^r(g) \coloneqq \underbrace{g \star \ldots \star g}_{r \text{ times}} = e.$$

Remark 2.1.1. *Let* G *be a group and* $r \in \mathbb{Z}$*. Now, let* $g \in \mathsf{G}$ *and* $\overline{g} \in \mathsf{G}$ *the inverse of* g*. Then, operation* $\star^r(g)$ *is extended as follows:*

$$\star^r(g) \coloneqq \begin{cases} \star^r(g) & \text{if } r > 0, \\ \star^{-r}(\overline{g}) & \text{if } r < 0, \\ e & \text{otherwise.} \end{cases}$$

A *subgroup* H of G, denoted by $\mathsf{H} \leq \mathsf{G}$, is a subset $\mathsf{H} \subseteq \mathsf{G}$ such that $(\mathsf{H}, \star)$ is also a group. In addition, given subset $S = \{s_1, \ldots, s_n\} \subseteq \mathsf{G}$ with $n \geq 1$ elements, the subgroup generated by S is defined as follows

$$\langle s_1, \ldots, s_n \rangle \coloneqq \left\{ \left(\star^{i_1}(s_1) \right) \star \left(\star^{i_2}(s_2) \right) \star \cdots \star \left(\star^{i_n}(s_n) \right) \colon s_i \in S, i_j \in \mathbb{Z} \right\}.$$

A group G is called cyclic if it is generated by only one element, i.e., $\mathsf{G} = \langle s \rangle$ for some $s \in \mathsf{G}$.

Theorem 2.1.1. *(Lagrange's theorem) Let* G *be a group of finite order, and* $\mathsf{H} \leq \mathsf{G}$*. Then, the order of* H *divides the order of* G*.*

Some special maps play an important role in group theory by allowing to connect different groups and "preserving" their binary operations. To be more precise, given two groups $(\mathsf{G}_1, \star)$ and $(\mathsf{G}_2, *)$, a map $f \colon \mathsf{G}_1 \to \mathsf{G}_2$ is a *group homomorphism* if

$$\forall g_1, g_2 \in \mathsf{G}_1 \colon f(g_1 \star g_2) = f(g_1) * f(g_2).$$

There are two important subgroups induced by f, the image and the kernel of f, which are defined as the $\operatorname{Im} f \coloneqq \{f(g_1) \in \mathsf{G}_2 \colon g_1 \in \mathsf{G}_1\} \leq \mathsf{G}_2$ and $\ker f \coloneqq \{g_1 \in \mathsf{G}_1 \colon f(g_1) = e_1\} \leq \mathsf{G}_1$, respectively. The homomorphisms are clasified by five different cases:

- **Monomorphisms:** injective maps,
- **Epimorphisms:** surjective maps,
- **Isomorphisms:** bijective maps,
- **Endomorphisms:** if G_1 is equal to G_2, and
- **Automorphism:** endomorphisms that are isomorphisms.

Remark 2.1.2. *Two groups* G_1 *and* G_2 *are called to be isomorphic, and denoted by* $\mathsf{G}_1 \cong \mathsf{G}_2$*, if there is an group isomorphism from* G_1 *to* G_2*.*

Likewise, a *ring* cosists of a non-empty set R with two binary operations $+$ (addition) and $\cdot$ (multiplication) such that

1. $(\mathsf{R}, +)$ is an abelian group.
2. The binary operation $\cdot \colon \mathsf{R} \times \mathsf{R} \to \mathsf{R}$ satisfies the following three properties
   (a) **Closure.** $\forall a, b \in \mathsf{R}$, $a \cdot b \in \mathsf{R}$.

(b) **Associativity.** $\forall a, b, c \in \mathsf{R}$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(c) **Neutral element.** $\exists! 1 \in \mathsf{R}\colon\ \forall a \in \mathsf{G}$, $1 \cdot a = a \cdot 1 = a$.

3. Multiplication is distributive concerning addition:

(a) $\forall a, b, c \in \mathsf{R}$, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

(b) $\forall a, b, c \in \mathsf{R}$, $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$.

Similarly as in groups, if the commutativity equation $a \cdot b = b \cdot a$ always holds then the ring R is called *commutative*, and a subset $\mathsf{S} \subseteq \mathsf{R}$ is a *subring* of R, denoted by $\mathsf{S} \leq \mathsf{R}$, if S is also a ring. On the other hand, R is called a *field* if $(\mathsf{R} \setminus \{0\}, \cdot)$ is an abelian group. Consequently, given a field $\mathbb{F}$, the set of polynomials in the variable x (i.e., finite linear combinations of monomials x^i) with coefficients in $\mathbb{F}$ is denoted by $\mathbb{F}[x]$ and it forms a commutative ring.

2.2 Finite fields

Given $a, b \in \mathbb{Z}$ with $a \neq 0$, there is a unique pair of integers $q, r \in \mathbb{Z}$ such that $b = q \cdot a + r$ with $r \in [\![0, n-1]\!]$; if $r = 0$ then a is called a factor of b (consequently, b is called a multiple of a), and we say that a divides b and it is denoted by $a \mid b$. The greatest common divisor and least common multiple of a and b are denoted by $\gcd(a, b)$ and $\operatorname{lcm}(a, b)$, respectively.

Definition 2.2.1. *Let $p \in \mathbb{Z}$. Then, p is a prime number (or irreducible in $\mathbb{Z}$) if the only factors of p are the 1 and itself. Otherwise, p is called a composite number.*

Definition 2.2.2. *Let $n \in \mathbb{N}$. Then, $a, b \in \mathbb{Z}$ are congruent modulo n, denoted by $a \equiv b \mod n$, if and only if $n|(a - b)$. In particular, for all $m \in \mathbb{Z}$ we have that $m \equiv r_m \mod n$ where $m = q_m \cdot n + r_m$ with $r_m \in [\![0, n-1]\!]$. Therefore, the modular addition and multiplication are defined by the next two congruences*

$$(a + b) \equiv (r_a + r_b) \mod n \equiv r_{a+b} \mod n, \text{ and}$$
$$(a \cdot b) \equiv (r_a \cdot r_b) \mod n \equiv r_{a \cdot b} \mod n.$$

Theorem 2.2.1. *Let $n \in \mathbb{N}$. Then $\mathbb{Z}_n \coloneqq \{0, 1, \ldots, n-1\}$, joined with the modular addition and multiplication, forms an commutative ring. Moreover, if $a \in \mathbb{Z}_n$ satisfies $\gcd(a, n) = 1$, then there is $a^{-1} \in \mathbb{Z}_n$ such that $a \cdot a^{-1} \mod n \equiv a^{-1} \cdot a \mod n \equiv 1 \mod n$. As a consequence, for any prime number p, $\mathbb{Z}_p$ is a field and it is denoted by $\mathbb{F}_p$.*

Clearly, all the definitions and properties in $\mathbb{Z}$ can be easily extended into any polynomial ring $\mathbb{F}_p[u]$, where p is a prime number. Consequently, given an irreducible degree-n polynomial $a(u) \in \mathbb{F}_p[u]$, the set

$$\mathbb{F}_{p^n} \coloneqq \mathbb{F}_p[u]/\langle a(u) \rangle \coloneqq \{c(u) \in \mathbb{F}_p[u]\colon \deg c(u) < n\},$$

joined with the modular addition and multiplication with modulus $a(u)$, forms a finite field of size p^n. Furthermore, let $q \coloneqq p^n$ and $b(v) \in \mathbb{F}_q[v]$ be an irreducible degree-ℓ polynomial, then

$$\mathbb{F}_{q^\ell} \coloneqq \mathbb{F}_q[v]/\langle b(v)\rangle \coloneqq \{c(v) \in \mathbb{F}_q[x] \colon \deg c(v) < \ell\}$$

is a field with q^ℓ elements. In addition, $\mathbb{F}_{q^\ell}$ is called a degree-ℓ extension field of $\mathbb{F}_q$, and it can be viewed as a $\mathbb{F}_q$-vector space of dimension ℓ.

Definition 2.2.3. *Given a finite field $\mathbb{F}_{q^\ell}$ with q^ℓ elements. The algebraic closure of $\mathbb{F}_{q^\ell}$, denoted by $\overline{\mathbb{F}}_{q^\ell}$, is defined as the field extension which satisfies:*

1. **Algrebraic extension:** $\forall r \in \overline{\mathbb{F}}_{q^\ell}$, $\exists h(x) \in \mathbb{F}_{q^\ell}[x] \colon h(r) = 0$.
2. **Algebraically closed:** *the only irreducible polynomials in $\overline{\mathbb{F}}_{q^\ell}[x]$ are those of degree one.*

In particular,

$$\overline{\mathbb{F}}_{q^\ell} = \bigcup_{i>1} \mathbb{F}_{q^i}.$$

2.3 Hyperelliptic curves over finite fields

Let p, ℓ and n be three positive integer numbers such that p is a prime number. Let $\mathbb{F}_q$ be the finite field with $q = p^n$ elements and $\mathbb{F}_{q^\ell}$ be a degree-ℓ extension field of $\mathbb{F}_q$. A genus-g hyperelliptic curve $\mathcal{H}/\mathbb{F}_{q^\ell}$ defined over $\mathbb{F}_{q^\ell}$ is given as

$$\mathcal{H}/\mathbb{F}_{q^\ell} \colon y^2 + h(x) \cdot y = f(x), \tag{2.1}$$

where $f, h \in \mathbb{F}_{q^\ell}[x]$ satisfy $\deg f = 2g+1$ and $\deg h \leq g$. The set of $(\mathbb{F}_{q^\ell})$-rational points of $\mathcal{H}/\mathbb{F}_{q^\ell}$ is defined as

$$\mathcal{H}\left(\mathbb{F}_{q^\ell}\right) = \left\{(x, y) \in \mathbb{F}_{q^\ell} \times \mathbb{F}_{q^\ell} \colon y^2 + h(x) \cdot y = f(x)\right\} \cup \{\mathcal{O}\}. \tag{2.2}$$

Here, $\mathcal{O}$ denotes the point at infinity. The negative of any point $\mathsf{P} = (x, y) \in \mathcal{H}(\mathbb{F}_{q^\ell}) \setminus \{\mathcal{O}\}$ is defined as $\overline{\mathsf{P}} = (x, -y - h(x))$. In particular, $\mathcal{H}/\mathbb{F}_{q^\ell}$ is called an elliptic curve if $g = 1$, and it has group law that is given by the secant and tangent rules. However, such rules are not longer well defined when $g > 1$.

For the case $g > 1$, one usually works with the jacobian of $\mathcal{H}/\mathbb{F}_{q^\ell}$, which is denoted by $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ and it is defined in terms of the group of divisors of $\mathcal{H}/\mathbb{F}_{q^\ell}$. Overall, a divisor D is a finite formal sum of points on the curve, i.e., $\mathsf{D} = \sum_{\mathsf{P}_i \in \mathcal{H}(\mathbb{F}_{q^\ell})} c_i(\mathsf{P}_i)$ where for almost all point $\mathsf{P}_i \in \mathcal{H}(\mathbb{F}_{q^\ell})$, its corresponding coefficient c_i is equal to zero. The degree of a divisor D is defined as $\deg \mathsf{D} = \sum c_i$. In addition, one can define a divisor to any rational function on $\mathcal{H}/\mathbb{F}_{q^\ell}$, which is called principal. Using the divisors terminology, the jacobian of a curve $\mathcal{C}$ on $\mathbb{F}_q$ can be defined as, $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell}) = \mathrm{Div}^0_{\mathcal{H}}(\mathbb{F}_{q^\ell})/\mathrm{Prin}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ where $\mathrm{Div}^0_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ and $\mathrm{Prin}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ denote the group of degree zero and principal divisors on $\mathcal{H}/\mathbb{F}_{q^\ell}$, respectively.

Often, it results more convenient to describe the jacobian $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ by means of the set of pair of polynomials $u, v \in \mathbb{F}_{q^\ell}[x]$ that satisfy the following three properties (known as Mumford's representation),

1) $\deg v < \deg u \leq g$,
2) u is a monic polynomial, and
3) $u|(v^2 + vh - f)$.

The divisor determined by the pair (u, v) is denote as $\mathsf{div}(u, v)$. The group law for operating two pairs $(u_i, v_i), (u_j, v_j)$ can be computed using Cantor's algorithm [21].

Remark 2.3.1. *The jacobian of any elliptic curve $\mathcal{E}/\mathbb{F}_{q^\ell}$ is isomorphic to its group of rational points, i.e., $\mathsf{Jac}_{\mathcal{E}}(\mathbb{F}_{q^\ell}) \cong \mathcal{E}(\mathbb{F}_{q^\ell})$.*

Notice that Mumford's representation allows to define the notions of divisor's irreducibility and smoothness: i) $\mathsf{div}(u, v)$ is irreducible if u is irreducible, and ii) $\mathsf{div}(u, v)$ is s-smooth if u is s-smooth. An important and useful fact is that if $u = \prod_i u_i$ then $\mathsf{div}(u, v) = \sum_i \mathsf{div}(u_i, v \mod u_i)$. Let us consider a point $\mathsf{P} = (x_\mathsf{P}, y_\mathsf{P}) \in \mathcal{H}(\overline{\mathbb{F}}_{q^\ell})$. Then, its corresponding divisor (P) is equal to $\mathsf{div}\big(x - x_\mathsf{P}, y_\mathsf{P}\big)$. Consequently, any divisor $\mathsf{div}(u, v) \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ can be written as $\sum_i c_i \cdot \mathsf{div}\big(x - x_{\mathsf{P}_i}, y_{\mathsf{P}_i}\big)$, where $(x_{\mathsf{P}_i}, y_{\mathsf{P}_i}) \in \mathcal{H}(\overline{\mathbb{F}}_{q^\ell})$, $u = \prod_i (x - x_{\mathsf{P}_i})^{c_i}$, and $v(x_{\mathsf{P}_i}) = y_{\mathsf{P}_i}$.

Theorem 2.3.1. *Let p, ℓ and n be three positive integer numbers such that p is a prime number. Let $\mathbb{F}_q$ be the finite field with $q = p^n$ elements and $\mathbb{F}_{q^\ell}$ be a degree-ℓ extension field of $\mathbb{F}_q$. And let $\mathcal{E}/\mathbb{F}_{q^\ell}$ be an elliptic curve. Then, $\#\mathcal{E}(\mathbb{F}_{q^\ell}) = q^\ell + 1 - t$ where $\mid t \mid \leq 2\sqrt{q^\ell}$. The value t is known as the Frobenius trace. Additionally, an elliptic curve is called supersingular if $p \mid t$; otherwise, is called ordinary.*

Definition 2.3.1. *Let p, ℓ and n be three positive integer numbers such that p is a prime number. Let $\mathbb{F}_q$ be the finite field with $q = p^n$ elements and $\mathbb{F}_{q^\ell}$ be a degree-ℓ extension field of $\mathbb{F}_q$. Let $\mathcal{E}/\mathbb{F}$ and $\mathcal{E}'/\mathbb{F}$ be two elliptic curves. An isogeny from $\mathcal{E}$ to $\mathcal{E}'$ is a non-constant rational map $\phi\colon \mathcal{E} \to \mathcal{E}'$ that it is also a group homomorphism over $\mathbb{F}_{q^\ell}$. In particular, $\phi(x, y) \mapsto (\phi_1(x, y), \phi_2(x, y))$ where $\phi_i(x, y)$ is a quotient of polynomials with coefficients over $\mathbb{F}_{q^\ell}$. Moreover, if such isogeny exists then, $\mathcal{E}$ and $\mathcal{E}'$ are called to be isogenous over $\mathbb{F}_{q^\ell}$.*

Theorem 2.3.2. *(Tate's theorem) Two elliptic curves $\mathcal{E}/\mathbb{F}_{q^\ell}$ and $\mathcal{E}'/\mathbb{F}_{q^\ell}$ are isogenous over $\mathbb{F}_{q^\ell}$ if and only if $\#\mathcal{E}(\mathbb{F}_{q^\ell}) = \#\mathcal{E}'(\mathbb{F}_{q^\ell})$. That is, supersingularity (and ordinarity) is preserved under isogeny maps.*

The Discrete Logarithm Problem (DLP) on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ is defined as follows: given two divisors $\mathsf{D} \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ and $\mathsf{D}' \in \langle \mathsf{D} \rangle$ of prime order r, find $\lambda \in \{0, \ldots, r-1\}$ such that $\mathsf{D}' = \lambda \mathsf{D}$. The most efficient method for solving the DLP on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ relies on an index-calculus approach that consists of the

following steps. First, let us fix s as the smoothness bound to be used in the attack. Second, let $\mathcal{F}_s$ be the set of irreducible divisors $\mathsf{div}(u,v) \in \mathsf{Jac}_H(\mathbb{F}_q)$ with $\deg u \leq s$, and ϵ a small integer. Then, one needs to generate $\#\mathcal{F}_s + \epsilon$ relations of the form, $\alpha_i \mathsf{D} + \beta_i \mathsf{D}' = \sum_{j=1}^{\#\mathcal{F}_s} m_{i,j} \mathsf{D}_j$, in order to construct three matrices $\alpha = (\alpha_i)^\mathsf{T}$, $\beta = (\beta_i)^\mathsf{T}$ and $M = (m_{i,j})$, whose coefficients belong to $\mathbb{Z}_r$. Once that this task is completed, one proceeds finding an element γ of the kernel of M^T so that $\gamma^\mathsf{T} M = 0$. Consequently, γ satisfies $\left(\gamma^\mathsf{T}\alpha\right)\mathsf{D} + \left(\gamma^\mathsf{T}\beta\right)\mathsf{D}' = 0$. In the case that $\gamma^\mathsf{T}\beta = 0$, one needs to repeat the whole procedure; otherwise, the discrete logarithm of D' with respect to D is given as, $\lambda = -\frac{\gamma^\mathsf{T}\alpha}{\gamma^\mathsf{T}\beta}$.

Clearly, the two main steps of the index-calculus approach are, i) The search for s-smooth divisors and; ii) The computation of a concrete kernel's element. This task is handled as a linear algebra problem. Regarding the first task, one can approximate the cost of the s-smooth divisors search as follows (for more details see [33]). Let $A_{s'}$ be the number of irreducible divisors $\mathsf{div}(u,v) \in \mathsf{Jac}_H(\mathbb{F}_q)$ with $\deg u = s'$, then

$$A_{s'} \approx \frac{1}{2}\left(\frac{1}{s'}\sum_{d|s'} \mu\left(\frac{s'}{d}\right) q^d\right), \tag{2.3}$$

where μ denotes the Möbius function, i.e., for every positive integer n we have

$$\mu(n) = \begin{cases} (-1)^k & \text{if } n \text{ is square free and has} \\ & k \text{ different prime factors,} \\ 0 & \text{if } n \text{ is not square free.} \end{cases} \tag{2.4}$$

Consequently, $\#\mathcal{F}_s \approx \sum_{i=1}^{s} A_i$. On the other hand, the number of s-smooth divisors $\mathsf{div}(u,v) \in \mathsf{Jac}_\mathcal{H}(\mathbb{F}_q)$ with $\deg u \leq g$ is given as

$$M(g,s) = \sum_{i=1}^{g}\left(\left[x^i\right]\prod_{s'=1}^{s}\left(\frac{1+x^{s'}}{1-x^{s'}}\right)^{A_{s'}}\right), \tag{2.5}$$

where $[.]$ denotes the coefficient operator. However, when $A_{s'}$ is known, $M(g,s)$ can be computed by finding the first $(g+1)$ terms of the Taylor expansion of $\prod_{s'=1}^{s}\left(\frac{1+x^{s'}}{1-x^{s'}}\right)^{A_{s'}}$ around $x = 0$, and adding the coefficients of x, x^2, $\ldots$, x^g. Thus, the expected number of random-walk iterations before a t-smooth divisor is encountered is

$$E(s) = \frac{\#\mathsf{Jac}_\mathcal{H}(\mathbb{F}_q)}{M(g,s)} \approx \frac{q^g}{M(g,s)}. \tag{2.6}$$

In addition, the expected number of random-walk iterations before $\left(\#\mathcal{F}_s + \epsilon\right)$ relations are generated is

$$T(s) = \left(\#\mathcal{F}_s + \epsilon\right) E(s). \tag{2.7}$$

With respect to the linear algebra task, the running time of the Lanczo's algorithm employed by magma can be approximated by, $L(s) \approx d \cdot \left(\#\mathcal{F}_s + \epsilon\right)^2$, where d denotes the per-row density of the matrix M. In fact, it can be shown that $d \leq g$.

Now, the most efficient choice for solving the DLP on a higher genus hyperelliptic curve is the Enge-Gaudry algorithm [23, 24], which has a *sub-exponential* running-time complexity of

$$L_{\left(q^\ell\right)^g}\left[\frac{1}{2}, \sqrt{2} + o(1)\right] = \exp\Big(\sqrt{2} + o(1)\Big)\sqrt{\log\left(q^\ell\right)^g}\sqrt{\log\log\left(q^\ell\right)^g}.$$

Remark 2.3.2. *For general small genus $g \geq 3$ curves, the algorithm of Gaudry, Thomé and Diem is the most efficient choice [22]. On the other hand, for genus-2 hyperelliptic curves, the most efficient approach is the Pollard's rho algorithm with an exponential running-time complexity of*

$$\sqrt{\frac{\pi \cdot \left(q^\ell\right)^g}{2}}.$$

Chapter 3

Extending the GLS endomorphism to speedup the GHS Weil descent

The definition of a good mathematical problem is the mathematics it generates rather than the problem itself.

Andrew Wiles

Let $q = 2^n$, and let $\mathcal{E}/\mathbb{F}_{q^\ell}$ be a generalized Galbraith-Lin-Scott (GLS) binary curve, with $\ell \geq 2$, $(\ell, n) = 1$. In this chapter it is shown that the GLS endomorphism on $\mathcal{E}/\mathbb{F}_{q^\ell}$ induces an efficient endomorphism on the jacobian $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ of a genus-g hyperelliptic curve $\mathcal{H}/\mathbb{F}_q\colon y^2 + h(x) \cdot y = f(x)$, where $\mathcal{H}/\mathbb{F}_q$ corresponds to the image of the GHS Weil descent attack applied on $\mathcal{E}/\mathbb{F}_{q^\ell}$. This endomorphism is defined as,

$$\mathsf{div}(u, v) \mapsto \mathsf{div}\left(\delta_1^{\deg u} \cdot ({}^{\sigma}u)\left(\tfrac{x}{\delta_1}\right), \delta_3\,({}^{\sigma}v)\left(\tfrac{x}{\delta_1}\right) + \delta_4\,({}^{\sigma}(h \mod u))\left(\tfrac{x}{\delta_1}\right)\right), \text{ for}$$

some elements δ_1, δ_3, and $\delta_4 \in \mathbb{F}_q$. Here the homomorphism ${}^{\sigma}(\cdot)$, operates over the coefficients of its input polynomial by applying the mapping ${}^{\sigma}\colon x \mapsto x^{2^\ell}$. We show that the above endomorphism yields a factor n speedup when using standard index-calculus procedures for solving the Discrete Logarithm Problem (DLP) on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Our analysis is backed up by the explicit computation of a discrete logarithm defined on a prime order subgroup of a GLS elliptic curve over the field $\mathbb{F}_{2^{5\cdot 31}}$. Indeed, a Magma implementation of a standard index-calculus procedure finds the aforementioned discrete logarithm in about $1,035$ CPU days. This computational cost is significantly less expensive than a comparable discrete logarithm computation reported in [32] (cf. Table 3.1). Remarkably, we managed to produce a faster discrete logarithm attack than the one reported in [32], in spite of the fact that we relied on a non-optimized implementation based on the computer algebra system Magma [30].

3.1 The GLS endomorphism

Let ℓ and n be two positive integer number such that $(\ell, n) = 1$. Let $\mathbb{F}_q$ be the finite field with $q = 2^n$ elements and $\mathbb{F}_{q^\ell}$ be a degree-ℓ extension field of $\mathbb{F}_q$. Let $\mathcal{E}/\mathbb{F}_{q^\ell}$ be an elliptic curve defined over $\mathbb{F}_{q^\ell}$ and given by Equation (3.1) such that $a \in \mathbb{F}_q \subset \mathbb{F}_{q^\ell}$ and $b \in \mathbb{F}_{2^\ell} \subset \mathbb{F}_{q^\ell}$.

$$\mathcal{E}/\mathbb{F}_{q^\ell} \colon y^2 + x \cdot y = x^3 + a \cdot x^2 + b, \quad a \in \mathbb{F}_{q^\ell}, \quad \text{and } b \in \left(\mathbb{F}_{q^\ell}\right)^\times. \tag{3.1}$$

The j-invariant of $\mathcal{E}/\mathbb{F}_{q^\ell}$ is $j(\mathcal{E}) = 1/b$. Now, for each integer i we let $\mathcal{E}_i/\mathbb{F}_{q^\ell}$ be the elliptic curve defined over $\mathbb{F}_{q^\ell}$ as,

$$\mathcal{E}_i/\mathbb{F}_{q^\ell} \colon y^2 + xy = x^3 + \left(a^{2^i}\right) x^2 + \left(b^{2^i}\right). \tag{3.2}$$

The curves $\mathcal{E} = \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_{n\cdot\ell-1}, \mathcal{E}_{n\cdot\ell} = \mathcal{E}$ are connected by a cycle of 2-power Frobenius maps $\pi \colon \mathcal{E}_i/\mathbb{F}_{q^\ell} \to \mathcal{E}_{i+1}/\mathbb{F}_{q^\ell}$ (that is, $\pi(x, y) = \left(x^2, y^2\right)$). Abusing notation, we will write π^k for the composition of any k successive Frobenius maps π. Since b is in $\mathbb{F}_{2^\ell}$, the curve $\mathcal{E}_\ell/\mathbb{F}_{q^\ell}$ is isomorphic to $\mathcal{E}/\mathbb{F}_{q^\ell}$; the isomorphism is

$$\begin{aligned} \phi \colon \mathcal{E}_\ell/\mathbb{F}_{q^\ell} &\longrightarrow \mathcal{E}/\mathbb{F}_{q^\ell} \\ (x, y) &\longmapsto (x, y + \delta x), \end{aligned} \tag{3.3}$$

where $\delta^2 + \delta = a + a^{2^\ell}$. Moreover, if $n \cdot \ell$ is odd then $\delta \in \mathbb{F}_{q^\ell} \setminus \mathbb{F}_{2^\ell}$, so the isomorphism ϕ is defined over $\mathbb{F}_{q^\ell}$, and in particular $\delta = \sum_{j=0}^{\frac{n\cdot\ell-1}{2}} \left(a + a^{2^\ell}\right)^{2^{2j}}$.

By composing the 2^ℓ-power Frobenius $\pi^\ell \colon \mathcal{E} \to \mathcal{E}_\ell$ with the isomorphism $\phi \colon \mathcal{E}_\ell \to \mathcal{E}$, we obtain a generalized Galbraith–Lin–Scott (GLS) endomorphism of $\mathcal{E}$, defined by

$$\psi \coloneqq \phi \circ \pi^\ell \colon (x, y) \longmapsto \left(x^{2^\ell}, y^{2^\ell} + \delta x^{2^\ell}\right) \in \mathrm{End}(\mathcal{E}).$$

The endomorphism ψ is defined over $\mathbb{F}_{q^\ell}$ and satisfies $\psi^n = 1$ or $\psi^n = -1$. Endomorphisms such as ψ are cryptographically interesting because they can be used to accelerate scalar multiplication on $\mathcal{E}$, by applying the technique of Gallant, Lambert, and Vanstone (for more details, see [25]).

In the sequel, we will show that these endomorphisms can also be used to improve the efficiency of the Gaudry–Hess–Smart Weil descent attack on weak curves of this kind.

3.2 Extending the GLS endomorphism

From now on, we fix an element w of $\mathbb{F}_{q^\ell}$ such that $w + w^2 + \cdots + w^{2^{\ell-1}} = 1$ and

$$\mathbb{F}_{q^\ell} = \mathbb{F}_q(w) = \left\langle w, w^2, w^4, \ldots, w^{2^{\ell-1}} \right\rangle_{\mathbb{F}_q};$$

that is, $\{w, w^2, w^4, \ldots, w^{2^{\ell-1}}\}$ is a normal basis for $\mathbb{F}_{q^\ell}$ over $\mathbb{F}_q$. Let

$$\mathcal{A}_i/\mathbb{F}_q := \mathcal{W}_{\mathbb{F}_q}^{\mathbb{F}_{q^\ell}}(\mathcal{E}_i)$$

be the Weil restriction of $\mathcal{E}_i/\mathbb{F}_{q^\ell}$ to $\mathbb{F}_q$. Recall that $\mathcal{A}_i$ is an ℓ-dimensional abelian variety over $\mathbb{F}_q$, and that there is an isomorphism between the groups $\mathcal{A}_i(\mathbb{F}_q)$ and $\mathcal{E}_i(\mathbb{F}_{q^\ell})$.[1] The different isogenies and endomorphisms of $\mathcal{E}_i$ correspond to isogenies and endomorphisms of $\mathcal{A}_i$.

We will use the following explicit affine model for $\mathcal{A}_i$. Consider the polynomial ring $R = \mathbb{F}_q[x_0, x_1, \ldots, x_{\ell-1}, y_0, \ldots, y_{\ell-1}]$, and set

$$X = \sum_{j=0}^{\ell-1} x_j w^{2^j} \qquad \text{and} \qquad Y = \sum_{j=0}^{\ell-1} y_j w^{2^j} \tag{3.4}$$

in $R \otimes \mathbb{F}_{q^\ell}$. By expanding the defining equation of $\mathcal{E}_i$ into the variables X and Y, there exist $W_0, \ldots, W_{\ell-1}$ in R such that $Y^2 + XY - (X^3 - (a^{2^i})X^2 - b) = \sum_{j=0} W_j w^{2^j}$ in $R \otimes \mathbb{F}_{q^\ell}$. The affine scheme $\mathrm{Spec}(R/(W_0, \ldots, W_{\ell-1}))$ is then $\mathbb{F}_q$-isomorphic to an open affine subset of $\mathcal{A}_i$. By construction, we have a bijection of sets

$$\begin{aligned} \iota\colon \mathcal{E}_i(\mathbb{F}_{q^\ell}) &\longrightarrow \mathcal{A}_i(\mathbb{F}_q) \\ (x, y) &\longmapsto (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}), \end{aligned} \tag{3.5}$$

where $x = \sum_{j=0}^{\ell-1} x_j w^{2^j}$ and $y = \sum_{j=0}^{\ell-1} y_j w^{2^j}$. In fact, ι is an isomorphism of groups.

We want now to make the isogenies and endomorphisms of $\mathcal{A}_i$ that correspond with π, ϕ, and ψ completely explicit with respect to this affine model of $\mathcal{A}_i$. To this end, observe first that if $X = \sum_{j=0}^{\ell-1} x_j w^{2^j}$, then $X^2 = \sum_{j=0}^{\ell-1} x_j^2 w^{2^{j+1}}$, so the 2-powering Frobenius isogeny $\pi : \mathcal{E}_i \to \mathcal{E}_{i+1}$ corresponds to an isogeny $\Pi : \mathcal{A}_i \to \mathcal{A}_{i+1}$ which squares and cyclically permutes the coordinates:

$$\Pi : (x_0, \ldots, x_{\ell-1}, y_0, \ldots, y_{\ell-1}) \longmapsto (x_{\ell-1}^2, x_0^2, \ldots, x_{\ell-2}^2, y_{\ell-1}^2, y_0^2, \ldots, y_{\ell-2}^2). \tag{3.6}$$

Let us recall that since a of Equation (3.1) is in $\mathbb{F}_q$, so is δ. Now, the isomorphism $\phi : \mathcal{E}_\ell \to \mathcal{E}$ maps (X, Y) to $(X, Y + \delta X)$, and so corresponds to an isomorphism $\Phi : \mathcal{A}_\ell \to \mathcal{A}$ defined on our affine model above by

$$\Phi : (x_0, \ldots, x_{\ell-1}, y_0, \ldots, y_{\ell-1}) \longmapsto (x_0, \ldots, x_{\ell-1}, y_0 + \delta x_0, \ldots, y_{\ell-1} + \delta x_{\ell-1}).$$

As with the elliptic curves $\mathcal{E}_i$, composing $\Pi^\ell : \mathcal{A} \to \mathcal{A}_\ell$ with $\Phi : \mathcal{A}_\ell \to \mathcal{A}$ yields an endomorphism Ψ of $\mathcal{A}$, defined (over $\mathbb{F}_q$) by

$$\Psi : (x_0, \ldots, x_{\ell-1}, y_0, \ldots, y_{\ell-1}) \longmapsto \left(x_0^{2^\ell}, \ldots, x_{\ell-1}^{2^\ell}, y_0^{2^\ell} + \delta x_0^{2^\ell}, \ldots, y_{\ell-1}^{2^\ell} + \delta x_{\ell-1}^{2^\ell}\right).$$

Clearly on groups of points we have $\Pi = \iota \circ \pi \circ \iota^{-1}$, $\Phi = \iota \circ \phi \circ \iota^{-1}$, and $\Psi = \iota \circ \psi \circ \iota^{-1}$. The relationships between all of these various maps are summarized in Figure 3.1.

[1] More generally, for any algebra K over $\mathbb{F}_q$, there is an isomorphism between $\mathcal{E}_i(\mathbb{F}_{q^\ell} \otimes_{\mathbb{F}_q} K)$ and $\mathcal{A}_i(K)$; in fact, $\mathcal{A}_i$ is the group scheme realizing the functor $K \mapsto \mathcal{E}_i(\mathbb{F}_{q^\ell} \otimes_{\mathbb{F}_q} K)$.

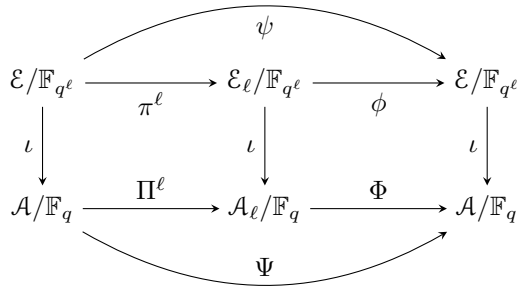

Figure 3.1: Endomorphism diagram

We note that if G is a cyclic subgroup of $\mathcal{E}(\mathbb{F}_{q^\ell})$ of order r, and ψ acts on G as multiplication by some eigenvalue $\lambda \pmod r$, then Ψ must act on $\iota(\mathsf{G})$ as multiplication by exactly the same eigenvalue λ.

3.3 Combining the GLS and GHS techniques

Firstly, the genus-g algebraic curve $\mathcal{C}/\mathbb{F}_q$ (not necessary hyperelliptic) that the generalized GHS (gGHS) Weil descent technique computes can be obtained by constructing the Weil restriction $\mathcal{A}/\mathbb{F}_q$ of $\mathcal{E}/\mathbb{F}_{q^\ell}$, intersecting $\mathcal{A}/\mathbb{F}_q$ with $(\ell-1)$-dimensional hyperplanes to obtain a subvariety $\mathcal{A}'/\mathbb{F}_q$ of $\mathcal{A}/\mathbb{F}_q$, and finding an irreducible component $\mathcal{C}/\mathbb{F}_q$ of $\mathcal{A}'/\mathbb{F}_q$ (for more details see [26, 27, 28, 29]). Therefore, let us intersect $\mathcal{A}/\mathbb{F}_q$ with the hyperplanes $x_0 = x_1 = \cdots = x_{\ell-1} = x \in \mathbb{F}_q$. However, assuming $a \in \mathbb{F}_q$ and $b \in \mathbb{F}_{2^\ell}$, and using the linear independence property of a normal basis $\{w, w^2, w^4, \ldots, w^{2^{\ell-1}}\}$ we can obtain a subvariety $\mathcal{A}'/\mathbb{F}_q$ of $\mathcal{A}/\mathbb{F}_q$, which is determined by equation (3.7).

$$\mathcal{A}'/\mathbb{F}_q\colon \begin{cases} x^3 + a\cdot x^2 + x\cdot y_0 + y_{\ell-1}^2 + b_0 &= 0 \\ x^3 + a\cdot x^2 + x\cdot y_1 + y_0^2 + b_1 &= 0 \\ & \vdots \\ x^3 + a\cdot x^2 + x\cdot y_{\ell-2} + y_{\ell-3}^2 + b_{\ell-2} &= 0 \\ x^3 + a\cdot x^2 + x\cdot y_{\ell-1} + y_{\ell-2}^2 + b_{\ell-1} &= 0 \end{cases} \tag{3.7}$$

where $b = \sum_{i=0}^{\ell-1} b_i w^{2^i}$ and each b_i belongs to $\mathbb{F}_2$. Thus, if $\mathcal{A}'_\ell/\mathbb{F}_q$ denotes the variety determined by equation (3.8), then the action of the endomorphism Φ on $\mathcal{A}'_\ell/\mathbb{F}_q$ sends points into the variety $\mathcal{A}'/\mathbb{F}_q$.

$$\mathcal{A}'_\ell/\mathbb{F}_q\colon \begin{cases} x^3 + a^{2^\ell}\cdot x^2 + x\cdot y_0 + y_{\ell-1}^2 + b_0 &= 0 \\ x^3 + a^{2^\ell}\cdot x^2 + x\cdot y_1 + y_0^2 + b_1 &= 0 \\ \vdots & \\ x^3 + a^{2^\ell}\cdot x^2 + x\cdot y_{\ell-2} + y_{\ell-3}^2 + b_{\ell-2} &= 0 \\ x^3 + a^{2^\ell}\cdot x^2 + x\cdot y_{\ell-1} + y_{\ell-2}^2 + b_{\ell-1} &= 0 \end{cases} \tag{3.8}$$

3.3.1 New endomorphism on the hyperelliptic curve

Let $\mathcal{H}/\mathbb{F}_q\colon y^2 + h(x)\cdot y = f(x)$ be the genus-g hyperelliptic curve that is an irreducible component of $\mathcal{A}'/\mathbb{F}_q$. Let us write

$$h(x) = \sum_{i=0}^{g} h_i\cdot x^i \text{ and } f(x) \;=\; \sum_{i=0}^{2g+1} f_i\cdot x^i.$$

Then, the corresponding hyperelliptic irreducible component $\mathcal{H}_\ell/\mathbb{F}_q$ of $\mathcal{A}'_\ell/\mathbb{F}_q$ is determined by equation (3.9).

$$\mathcal{H}_\ell/\mathbb{F}_q\colon y^2 + ({}^\sigma h)(x)\cdot y = ({}^\sigma f)(x) \tag{3.9}$$

where $({}^\sigma h)(x) = \sum_{i=0}^{g}\sigma(h_i)\cdot x^i$, $({}^\sigma f)(x) = \sum_{i=0}^{2g+1}\sigma(f_i)\cdot x^i$, and $\sigma(x) = x^{2^\ell}$ for all $x\in\mathbb{F}_q$. Therefore, $\Pi^\ell\colon \mathcal{H}/\mathbb{F}_q \to \mathcal{H}_\ell/\mathbb{F}_q$ and $\Phi\colon \mathcal{H}_\ell/\mathbb{F}_q \to \mathcal{H}/\mathbb{F}_q$ are determined as follows:

$$\Pi^\ell\colon (x,y)\longmapsto \left(x^{2^\ell}, y^{2^\ell}\right) \text{ and } \Phi\colon (x,y)\longmapsto \big(\delta_1\cdot x + \delta_2, \delta_3\cdot y + t(x)\big),$$

for some $\delta_1,\delta_2,\delta_3\in\mathbb{F}_q$ and $t(x)\in\mathbb{F}_q[x]$ with $\deg t(x)\le g$ and $\delta_1\neq 0$ [1]. Consequently, $\Psi = \Phi\circ\Pi^\ell$ induces the following endomorphism:

$$\begin{aligned}\Psi^\star\colon \mathsf{Jac}_{\mathcal{H}}\left(\overline{\mathbb{F}}_q\right) &\longrightarrow \mathsf{Jac}_{\mathcal{H}}\left(\overline{\mathbb{F}}_q\right)\\ \sum_j c_j\left(\mathsf{P}_j\right) &\longmapsto \sum_j c_j\left(\Psi(\mathsf{P}_j)\right).\end{aligned}$$

Therefore, one can check that the divisor $\mathsf{div}(u,v) = \sum_j c_j\cdot\mathsf{div}(x + x_{\mathsf{P}_j}, y_{\mathsf{P}_j})$ is mapped to $\sum_j c_j\cdot\mathsf{div}(x + x_{\Psi(\mathsf{P})_j}, y_{\Psi(\mathsf{P})_j})$, and therefore irreducible factors (over $\mathbb{F}_q[x]$) of u are mapped to irreducible factors of the same degree, i.e., $\Psi^\star$ sends smooth divisors to smooth divisors.

Now, because the curve $\mathcal{H}/\mathbb{F}_q$ has genus $g\ge\ell$, its Jacobian $\mathsf{Jac}_{\mathcal{H}}$ is g-dimensional. By the universal property of the Jacobian,[2] The ℓ-dimensional $\mathcal{A}$

[1] Any isomorphism between two hyperelliptic curves over finite fields is of the form of Φ (for more details see section 10.2 of [20]).

[2] Universal property: let $\kappa\colon\mathcal{H}\to\tilde{A}$ be a morphism, where $\tilde{A}$ is an abelian variety. Let $\mathsf{P}_0\in\mathcal{H}(\overline{\mathbb{F}}_q)$ be such that $\kappa(\mathsf{P}_0) = 0$, and consider the map $\tilde{\kappa}\colon\mathcal{H}\to\mathsf{Jac}_{\mathcal{H}}$ given by $\mathsf{P}\mapsto(\mathsf{P}) - (\mathsf{P}_0)$. Then there is a unique homomorphism $\psi\colon\mathsf{Jac}_{\mathcal{H}}\to\tilde{A}$ of abelian varieties such that $\kappa = \psi\circ\tilde{\kappa}$ (for more details see section 10.5 of [20]).

is a quotient (and so an isogeny factor) of $\mathsf{Jac}_{\mathcal{H}}$. Hence, we have $\mathsf{Jac}_{\mathcal{H}} \cong \mathcal{A} \times \mathcal{B}$ for some $(g-\ell)$-dimensional abelian variety $\mathcal{B}$. The situation is illustrated by the diagram in Figure 3.2.

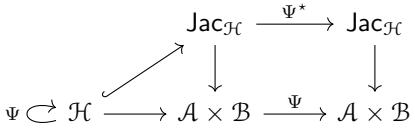

Figure 3.2: Endomorphism diagram for $\mathcal{H}/\mathbb{F}_q$

As we noted above, if G is a cyclic subgroup of $\mathcal{E}(\mathbb{F}_{2^{\ell \cdot n}})$ fixed by ψ and ψ acts on G as multiplication by an eigenvalue λ, then Ψ acts on $\iota(\mathsf{G})$ as multiplication by λ. As a consequence, we have $(\Psi^\star)^n = 1$ or $(\Psi^\star)^n = -1$, and (because of the nature of Φ on $\mathcal{A}'_\ell/\mathbb{F}_q$) we have that $t(x) = \delta_4(^\sigma h)(\delta_5 \cdot x)$ for some $\delta_4, \delta_5 \in \mathbb{F}_q$. Moreover, the morphism $\Psi^n \colon \mathcal{H}/\mathbb{F}_q \to \mathcal{H}/\mathbb{F}_q$ must fix its x-coordinate, and

$$x_{\Psi^n(\mathsf{P})} = \delta_1^{\left(\sum_{k=0}^{n-1} 2^{k\cdot\ell}\right)} \cdot x_{\mathsf{P}}^{\left(2^{n\cdot\ell}\right)} + \sum_{i=0}^{n-1} \delta_1^{\left(\sum_{k=0}^{i-1} 2^{k\cdot\ell}\right)} \cdot \delta_2^{\left(2^{i\cdot\ell}\right)}.$$

However, recall that $(\ell, n) = 1$ and $q = 2^n$. Then, $\delta_1^{\left(\sum_{k=0}^{n-1} 2^{k\cdot\ell}\right)} = (\delta_1)^{2^n-1} = (\delta_1)^{q-1} = 1$, $x_{\mathsf{P}}^{2^{n\cdot\ell}} = x_{\mathsf{P}}^{q^\ell} = x_{\mathsf{P}}$, and $x_{\Psi^n(\mathsf{P})} = x_{\mathsf{P}} + \sum_{i=0}^{n-1} \delta_1^{\left(\sum_{k=0}^{i-1} 2^{k\cdot\ell}\right)} \cdot \delta_2^{\left(2^{i\cdot\ell}\right)}$. Therefore, $\sum_{i=0}^{n-1} \delta_1^{\left(\sum_{k=0}^{i-1} 2^{k\cdot\ell}\right)} \cdot \delta_2^{\left(2^{i\cdot\ell}\right)} = 0$. In particular, considering the nature of Φ on $\mathcal{A}'_\ell/\mathbb{F}_q$, it must follow that $\delta_2 = 0$.

3.3.2 Explicit description of the new endomorphism

Recall that i) for any point $\mathsf{P} = (x_{\mathsf{P}}, y_{\mathsf{P}}) \in \mathcal{H}(\mathbb{F}_q)$ its corresponding divisor (P) is equal to $\mathsf{div}\big(x + x_{\mathsf{P}}, y_{\mathsf{P}}\big)$, and ii) any divisor $\mathsf{div}(u, v) \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{q^\ell})$ can be written as $\sum_i c_i \cdot \mathsf{div}\big(x + x_{\mathsf{P}_i}, y_{\mathsf{P}_i}\big)$, where $(x_{\mathsf{P}_i}, y_{\mathsf{P}_i}) \in \mathcal{H}(\overline{\mathbb{F}}_{q^\ell})$, $u = \prod_i (x + x_{\mathsf{P}_i})^{c_i}$, and $v(x_{\mathsf{P}_i}) = y_{\mathsf{P}_i}$.

Therefore, i) and ii) imply that the divisor $\Psi^\star\big((\mathsf{P})\big) = \big(\Psi(\mathsf{P})\big)$ is equal to $\mathsf{div}\Big(x + \big(\delta_1 x_{\mathsf{P}}^{2^\ell}\big), \delta_3 y_{\mathsf{P}}^{2^\ell} + \delta_4(^\sigma h)\big(\delta_5 \delta_1 x_{\mathsf{P}}^{2^\ell}\big)\Big)$, and any divisor $\mathsf{div}(u, v) \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ satisfies $\mathsf{div}(u, v) = \sum_{i=0}^{\deg u} \mathsf{div}(x + x_i, v(u_i)) = \sum_{i=0}^{\deg u} \Big(\big(x_i, v(x_i)\big)\Big)$, where $x_0, \ldots,$

$x_{\deg u} \in \overline{\mathbb{F}}_q$ are the roots of u. In addition,

$$\begin{aligned}\Psi^\star\big(\mathsf{div}(u,v)\big) &= \Psi^\star\left(\sum_{i=0}^{\deg u}\Big(\big(x_i, v(x_i)\big)\Big)\right) = \sum_{i=0}^{\deg u}\Psi^\star\Big(\big(x_i, v(x_i)\big)\Big)\\ &= \sum_{i=0}^{\deg u}\mathsf{div}\Big(x + \big(\delta_1 {x_i}^{2^\ell}\big), \delta_3\big(v(x_i)\big)^{2^\ell} + \delta_4({}^\sigma h)\big(\delta_5\delta_1 {x_i}^{2^\ell}\big)\Big).\\ &= \sum_{i=0}^{\deg u}\mathsf{div}\left(\delta_1\Big(\frac{x}{\delta_1} + {x_i}^{2^\ell}\Big), \delta_3({}^\sigma v)\big(x_i^{2^\ell}\big) + \delta_4({}^\sigma h)\big(\delta_5\delta_1 {x_i}^{2^\ell}\big)\right).\end{aligned}$$

Let us recall now that our goal is to find the two polynomials $u^*, v^* \in \mathbb{F}_q[x]$ such that $\Psi^\star\big(\mathsf{div}(u,v)\big) = \mathsf{div}(u^*, v^*)$, $u^*\big(\delta_1 {x_i}^{2^\ell}\big) = 0$, and $v^*\big(\delta_1 {x_i}^{2^\ell}\big) = \delta_3\big(v(x_i)\big)^{2^\ell} + \delta_4({}^\sigma h)\big(\delta_5\delta_1 {x_i}^{2^\ell}\big)\Big)$.

In particular, $u^*(x) \coloneqq \delta_1^{\deg u}\cdot({}^\sigma u)\Big(\frac{x}{\delta_1}\Big) = \prod_{i=0}^{\deg u}\Big(x + \delta_1\cdot {x_i}^{2^\ell}\Big)$, and $v^*(x) \coloneqq \delta_3({}^\sigma v)\Big(\frac{x}{\delta_1}\Big) + \delta_4\big({}^\sigma h\big)(\delta_5 x)$. Moreover, because u^* and v^* must satisfy $\deg v^* < \deg u^* \le g$ and $u^* \mid \big((v^*)^2 + (v^*\cdot h) + f\big)$, one can set $v^*(x) \coloneqq \delta_3({}^\sigma v)\Big(\frac{x}{\delta_1}\Big) + \Big(\delta_4\big({}^\sigma h\big)(\delta_5 x)\Big) \mod u^*(x)$.

Additionally, the endomorphism Ψ^* must be well-defined in the sense that v^* should be the same if we reduce h modulus u from the beginning. This observation implies that the following equation must be satisfied,

$$\Big(\delta_4\big({}^\sigma h\big)(\delta_5 x)\Big) \mod u^*(x) = \Big(\delta_4\big({}^\sigma(h \mod u)\big)(\delta_5 x)\Big) \mod u^*(x). \quad (3.10)$$

Now, let us write $u = \sum_i u_i x^i$, $v = \sum_i v_i x^i$, $h = \sum_i h_i x^i$, $(h \mod u) = \sum_i h'_i x^i$, and $\Big(\delta_4\big({}^\sigma h\big)(\delta_5 x)\Big) \mod u^*(x) = \sum_i h^*_i x^i$. Then, $u^* = \delta_1^{\deg u}\sum_i \frac{{u_i}^{2^\ell}}{\delta_1^i}x^i$, $\delta_3({}^\sigma v)\Big(\frac{x}{\delta_1}\Big) = \delta_3\sum_i \frac{{v_i}^{2^\ell}}{\delta_1^i}x^i$, and $\delta_4\big({}^\sigma(h \mod u)\big)(\delta_5 x) = \delta_4\sum_i \big(h'_i\big)^{2^\ell}\big(\delta_5\big)^i x^i$. In particular, equation (3.10) holds for any divisor $\mathsf{div}(u,v) \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Let us analyze the cases $\deg u = g$ and $\deg u \le g$ separately.

The case when $\deg u = g$

Assuming that $\deg u = g$, one can write,

$$\begin{aligned} h'_i &= c_{h_i} + h_g\cdot u_i, \text{ and}\\ h^*_i &= \delta_4\cdot\Big({h_i}^{2^\ell}\cdot\delta_5^i + \big({h_g}^{2^\ell}\cdot\delta_5^g\big)\cdot\delta_1^{g-i}\Big).\end{aligned} \quad (3.11)$$

Considering equations (3.10) and (3.11), for each $i = 0, \ldots, (g-1)$ we have

$$\delta_4\cdot(h_i + h_g\cdot u_i)^{2^\ell}\cdot\delta_5^i = \delta_4\cdot\Big({h_i}^{2^\ell}\cdot\delta_5^i + \big({h_g}^{2^\ell}\cdot\delta_5^g\big)\cdot\delta_1^{g-i}\Big). \quad (3.12)$$

It follows that equation (3.12) is satisfied iff, $\delta_5^i = \delta_5^g \cdot \delta_1^{g-i}$, for each $i = 0, \ldots, (g-1)$. Moreover, $\delta_5 = \frac{1}{\delta_1}$ and

$$\Psi^\star\big(\mathsf{div}(u,v)\big) = \mathsf{div}\left(\delta_1^{\deg u} \cdot ({}^\sigma u)\Big(\frac{x}{\delta_1}\Big), \delta_3({}^\sigma v)\Big(\frac{x}{\delta_1}\Big) + \delta_4\big({}^\sigma(h \mod u)\big)\left(\frac{x}{\delta_1}\right)\right). \tag{3.13}$$

General case when $\deg u \leq g$

Let us consider again the general case when $\deg u \leq g$. Let us assume $\mathsf{div}(u,v)$ is a divisor of maximal prime order r (i.e, r is the largest prime factor of $\#\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$). Therefore, we know that Ψ^* acts on $\langle \mathsf{div}(u,v)\rangle$ as multiplication by an eigenvalue λ, i.e., $\Psi^*(\mathsf{div}(u,v)) = \lambda \cdot \mathsf{div}(u,v)$. Thus, we can compute the divisor $\mathsf{div}(u',v') \coloneqq \lambda \cdot \mathsf{div}(u,v)$ by using Cantor's algorithm. As a consequence, we have that $\mathsf{div}(u^*,v^*) \coloneqq \Psi^*(\mathsf{div}(u,v))$ must be equal to $\mathsf{div}(u',v')$.

Now, let us write $u' = \sum_i u'_i x^i$ and $v' = \sum_i v'_i x^i$. Then $u^* = u'$ and $v^* = v'$ imply that $\tilde{\delta_1} = \frac{1}{\delta_1}, \delta_3, \delta_4 \in \mathbb{F}_q$ must belong to the varieties V_1 and $V_{3,4}$, which are given by equations (3.14) and (3.15), respectively.

$$V_1/\mathbb{F}_q\colon \begin{cases} \big(u_0\big)^{2^\ell} &= \big(\tilde{\delta}_1\big)^{\deg u} \cdot u'_0 \\ \big(u_1\big)^{2^\ell} \cdot \tilde{\delta}_1 &= \big(\tilde{\delta}_1\big)^{\deg u} \cdot u'_1 \\ \big(u_2\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^2 &= \big(\tilde{\delta}_1\big)^{\deg u} \cdot u'_2 \\ & \vdots \\ \big(u_{\deg u-1}\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^{\deg u-1} &= \big(\tilde{\delta}_1\big)^{\deg u} \cdot u'_{\deg u-1} \end{cases} \tag{3.14}$$

and

$$V_{3,4}/\mathbb{F}_q\colon \begin{cases} \delta_3 \cdot \big(v_0\big)^{2^\ell} + \delta_4 \cdot \big(h'_0\big)^{2^\ell} &= v'_0 \\ \delta_3 \cdot \big(v_1\big)^{2^\ell} \cdot \tilde{\delta}_1 + \delta_4 \cdot \big(h'_1\big)^{2^\ell} \cdot \tilde{\delta}_1 &= v'_1 \\ \delta_3 \cdot \big(v_2\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^2 + \delta_4 \cdot \big(h'_2\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^2 &= v'_2 \\ & \vdots \\ \delta_3 \cdot \big(v_{\deg v}\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^{\deg v} + \delta_4 \cdot \big(h'_{\deg v}\big)^{2^\ell} \cdot \big(\tilde{\delta}_1\big)^{\deg v} &= v'_{\deg v} \end{cases} \tag{3.15}$$

It is important to note that the variety $V_1/\mathbb{F}_q$ only depends on the parameter $\tilde{\delta}_1$, and it is determined by $(\deg u)$ polynomial equations of degree at most $\deg u$. In particular, the $(\deg u)$-th equation of V_1 implies $\tilde{\delta}_1 = \frac{(u_{\deg u-1})^{2^\ell}}{u'_{\deg_u - 1}}$, if $u_{\deg u-1} \neq 0$. Otherwise, using the i-th and j-th equations of V_1 with $j < i$, implies $\tilde{\delta_1}^{i-j} = \frac{(u_j)^{2^\ell} \cdot u'_i}{(u_i)^{2^\ell} \cdot u'_j}$, when $u_i \cdot u_j \neq 0$. On the other hand, the variety $V_{3,4}/\mathbb{F}_q$ only depends on the parameters δ_3 and δ_4, and it is determined by $(\deg v + 1)$

linear equations. Then, $V_{3,4}(\mathbb{F}_q)$ consists of a unique point $(\delta_3, \delta_4) \in \mathbb{F}_q \times \mathbb{F}_q$. Moreover, combining the i-th and j-th equations of $V_{3,4}$ with $i \neq j$,

$$\delta_3 = \frac{v'_i \cdot (\delta_1)^i + \delta_4 \cdot (h'_i)^{2^\ell}}{(v_i)^{2^\ell}}, \text{ and} \tag{3.16}$$

$$\delta_4 = \frac{v'_i \cdot (\delta_1)^i \cdot (v_j)^{2^\ell} + v'_j \cdot (\delta_1)^j \cdot (v_i)^{2^\ell}}{(h'_i)^{2^\ell} \cdot (v_j)^{2^\ell} + (h'_j)^{2^\ell} \cdot (v_i)^{2^\ell}} \tag{3.17}$$

where the denominators of equation (3.16) and (3.17) are different than zero.

3.3.3 Speeding-up the Index-Calculus algorithm in $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$

Given two divisors $\mathsf{D} \in \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ and $\mathsf{D}' \in \langle \mathsf{D} \rangle$ of order r, a positive integer s, and a small positive integer ϵ. Recall that the main steps of an index-calculus-based algorithm are:

1. To set s as the smoothness bound, and $\mathcal{F}_s$ as the set of irreducible divisors $\mathsf{div}(u, v) \in \mathsf{Jac}_H(\mathbb{F}_q)$ with $\deg u \leq s$.

2. To generate $\#\mathcal{F}_s + \epsilon$ relations of the form, $\alpha_i \mathsf{D} + \beta_i \mathsf{D}' = \sum_{j=1}^{\#\mathcal{F}_s} m_{i,j} \mathsf{D}_j$, in order to construct the three matrices $\alpha = (\alpha_i)^\mathsf{T}$, $\beta = (\beta_i)^\mathsf{T}$ and $M = (m_{i,j})$, whose coefficients belongs to $\mathbb{Z}_r$.

3. To find an element γ of the kernel of M^T so that $\gamma^\mathsf{T} M = 0$.

4. If $\beta \cdot \overrightarrow{\gamma} \neq 0$ then compute the discrete logarithm: $\gamma = -\frac{\alpha \cdot \overrightarrow{\gamma}}{\beta \cdot \overrightarrow{\gamma}}$. Otherwise, go back to step 2.

The Index-Calculus procedure just described requires to find a number $\#\mathcal{F}_s + \epsilon$ of s-smooth divisors. Nevertheless, once that an s-smooth divisor is found, the explicit endomorphism described in the previous section allows us to find at once, $n-1$ linearly independent extra divisors that are also s-smooth. Furthermore, we can do even better by exploiting the endomorphism $\Psi^\star$ to reduce the size of the factor base $\mathcal{F}_s$ from $\#\mathcal{F}_s$ to $\frac{\#\mathcal{F}_s}{n}$.

Remark 3.3.1. *Since $\lambda \neq \mp 1$ and $\lambda^n \pm 1 \equiv 0 \mod r$, it implies that the divisors $\mathsf{D}, \lambda \cdot \mathsf{D}, \ldots, [\lambda^{n-2}]\mathsf{D}$, and $[\lambda^{n-1}]\mathsf{D}$ are linearly dependent. Therefore, the vector γ of Step 4 of the above procedure, could possibly annihilate α and β, i.e., it may happen that $\gamma \cdot \alpha \equiv 0$ and $\gamma \cdot \beta \equiv 0$. Thus, it seems more prudent to use only $(n-1)$ related divisors, namely, $\mathsf{D}, \lambda \cdot \mathsf{D}, \ldots, [\lambda^{n-3}]\mathsf{D}$, and $[\lambda^{n-2}]\mathsf{D}$.*

3.3.3.1 Solving discrete logarithms on $\mathcal{E}/\mathbb{F}_{2^{5\times 31}}$.

In order to illustrate the implications of the previous analysis, we solved the DLP on an weak GLS binary curve. More precisely, let q and ℓ be given as, $q = 2^n$

with $n = 5$, and $\ell = 31$. Then, let us define the fields, $\mathbb{F}_q = \mathbb{F}_2[u]/\langle u^5 + u^2 + 1\rangle$ and $\mathbb{F}_{q^\ell} = \mathbb{F}_q[v]/\langle v^{31} + v^3 + 1\rangle$. Let $\mathcal{E}/\mathbb{F}_{q^\ell}$ be an elliptic curve given as,

$$\mathcal{E}/\mathbb{F}_{q^\ell}\colon y^2 + x \cdot y = x^3 + x^2 + \big(v^{18} + v^{17} + v^{12} + v^8 + v^5 + v^4 + 1\big).$$

Using Magma, it can be easily verified that, $\#\mathcal{E}\left(\mathbb{F}_{q^\ell}\right) = c \cdot r$, where $c =$ 0x12E7FB306F6 and $r =$ 0x6C530B0FAF0022649878E620CAE2D is a 115-bit prime. Next, we randomly selected an order-r point $\mathsf{P} = (X_\mathsf{P}, Y_\mathsf{P})$ by using the `Random()` function of Magma, and we set $\mathsf{P}' = [c]\,(\pi_x, \pi_y)$ where $\pi_x = \frac{v^{355}}{v^{133}} + (v + u + 1)$ and π_y is one of the roots of $\pi_x^3 + \pi_x^2 + \big(v^{18} + v^{17} + v^{12} + v^8 + v^5 + v^4 + 1\big)$. Our goal was to find $\lambda \in \{1, \ldots, r\}$ such that $\mathsf{P}' = \lambda \cdot \mathsf{P}$. The discrete logarithm problem on this weak elliptic curve is described by the Magma code given in Appendix A.1. Now, using the function `WeilDescent()` of Magma, we reduced the problem into a hyperelliptic genus-32 curve $\mathcal{H}/\mathbb{F}_q$ defined over $\mathbb{F}_q$ and given by the following equation

$$\begin{aligned}\mathcal{H}/\mathbb{F}_q\colon y^2 + \big(u^7x^{32} + u^{12}x^{16} + u^{30}x^8 + u^{28}x^2 + u^7x\big) \cdot y = \\ u^4x^{65} + u^{14}x^{64} + u^{14}x^{33} + u^19x^{17} + u^{16}x^8 + u^{15}x^5 + u^{25}x^4 + u^4x^3 + u^{24}x.\end{aligned}$$

Hence, the points P and P' are mapped to D and D', respectively. The new DLP instance this time defined on the jacobian $\mathcal{H}/\mathbb{F}_q$, is described by the Magma code given in Appendix A.2. Additionally, the endomorphism $\Psi^*\colon \mathsf{Jac}_\mathcal{H}(\mathbb{F}_q) \to \mathsf{Jac}_\mathcal{H}(\mathbb{F}_q)$, is given by the relation,

$$\Psi^\star\colon \mathsf{div}(u, v) \mapsto \mathsf{div}\left(\big(u^{21}\big)^{\deg u} \cdot (^\sigma u)\Big(\frac{x}{u^{21}}\Big), \big(u^{14}\big) \cdot (^\sigma v)\Big(\frac{x}{u^{21}}\Big)\right).$$

Using this setting, we implemented a parallel version of the Enge-Gaudry algorithm using the computer algebra system Magma [30]. Our magma-code implementetion was done as follows: the i-th thread had the task of building its own local factor base of size $\frac{\#\mathcal{F}_s}{n\cdot(\text{number of threads})} + \epsilon$,

$$\mathcal{F}_{i,s} = \left\{ \max\Big\{(\Psi^*)^i\big(\mathsf{div}(u, v)\big)\colon i = 0, \ldots, n-1\Big\} : \mathsf{div}(u, v) \text{ is an irreducible divisor with } \deg u \le s \right\};$$

that is, each thread was working without communication between the other threads. After all the threads have finished their tasks, our magma-code implementation proceed by merging and constructing the matrices to be used in the Linear algebra step. As a consequence, we successfully accelerated the relation step of the Enge-Gaudry algorithm by using the endomorphism Ψ^* as discussed in §3.3. Moreover, the factor base was dynamically built as in [31], and the smoothness bound used was $s = 4$. As a result, we solved the DLP in $\mathsf{Jac}_\mathcal{H}(\mathbb{F}_{2^5})$

in 1034.596 CPU days and we found γ = 0x618877C96DE350E8C7980393356E3 [1]. Our Magma-code implementation of our procedures is available at

https://github.com/JJChiDguez/combining_GLS_with_GHS.

3.3.3.2 Comparison with related work

In [32], Velichka *et al.* reported the solution of this same discrete logarithm instance using a factor base size of 136533. Velichka *et al.* gave timing estimates for computing the discrete logarithm problem based on the Enge-Gaudry algorithm using optimal parameters derived from [33]. In addition, they gave the timings obtained from their sieve-based version of Vollmer's algorithm [2]. Table 3.1 compare the timings obtained from our experiments with the ones reported in [32]. As expected from our analysis, in our work we were able to reduce the factor base to 27271 $\approx \frac{136533}{5}$. Due to the more advanced micro-architecture, the speedup achieved by our approach is higher than expected.

Notice that the endomorphism $\Psi^* \colon \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q) \to \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ can also be used in the sieve-based version of Vollmer's algorithm [32]. Extrapolating the timing costs given in Table 3.1 we would expect 344.164 and 0.569 CPU days for the computational costs of the divisor search and linear algebra steps, respectively.

	This work	Velichka *et al.* work [32]		
		JMS EG Estimate	Opt. EG Estimate	Sieving method
4-smooth divisors search (CPU days)	**1034.572**	8492.67	6338.01	1720.818
Linear algebra step (CPU days)	**0.024**	2.470	2.800	14.244
Total	**1034.597**	8495.650	6340.810	1735.063
Speedup:		**8.212**	**6.129**	**1.677**

Table 3.1: CPU days required in the Index-Calculus based algorithm with smoothness bound equals 4: solving the DLP on a hyperelliptic genus-32 curve $\mathcal{H}/\mathbb{F}_{2^5}$. The 2nd, 3rd, and 4th column show the timing estimations of using the Enge-Gaudry algorithm with i) the strategy and optimal parameters from [33], ii) an optimized version that incorporates large prime variations, and the sieve-based version of Vollmer's algorithm, respectively.

[1] The linear algebra step was solved with one core of an Intel Xeon E5 2.60GHz machine by using the function `ModularSolution()` that Magma has implemented. In addition, we used 96 cores of 16 Intel Core i7 machines (3.20GHz, 3.40GHz, and 3.47GHz) and 32 cores of two Intel Xeon E5 2.60GHz machines for finding 4-smooth divisors.

[2] The authors used 152 dual Intel P4 Xeon machines (2.4GHz, and 2.8GHz) with 512 kb cache and 2 GB of RAM. On the top of that, they used the GNU Multi-Precision C library version 4.2.2, Automatically Tuned Linear Algebra Software (ATLAS) version 3.7.31 ([2]), linbox version 1.1.3 to perform linear algebra, and the GCC version 3.4.4.

3.4 Root-finding problem related with the gGHS Weil descent attack

Recall, in the introduction we mentioned that the genus of the image curve $\mathcal{C}/\mathbb{F}_q$ of the gGHS Weil descent technique can be completely determined by finding the roots of polynomials of the form $\hat{h}(x) = \sum_{i=0}^{d} h_i \cdot x^{q^i} \in \mathbb{F}_p[x]$. Thus, let's procced by given an efficient root-finding algorithm for these kind of polynomials, and then ilustrate how our proposed algorithm can be combined with the gGHS Weil descent.

Let p be a prime integer number, and $n \geq 1$ be an integer number. Let $\mathbb{F}_q$ be the finite field with $q = p^n$ elements. Now, for any degree-d polynomial $h(x) = \sum_{i=0}^{d} h_i \cdot x^i \in \mathbb{F}_q[x]$, let us define $h^\sigma(x) = \sum_{i=0}^{d} h_i \cdot x^{(q^i)}$, which is a degree-$\left(q^d\right)$ polynomial with coefficients in $\mathbb{F}_q$. The polynomial $h^\sigma(x)$ is called linearized. Now, let us focus on the case when the univariate polynomial $h(x) = \prod_i (x - x_i) \prod_j \hat{f}_j(x)$ is the product of different linear and irreducible degree-t polynomials in $\mathbb{F}_q[x]$. In particular, let's assume $x \nmid h(x)$. Let $\mathbb{F}_{q^\ell}$ be a degree-ℓ extension field of $\mathbb{F}_q$ where ℓ is defined as,

$$\ell = \begin{cases} p^t - 1 & \text{if } (x - x_i), \hat{f}_j(x) \in \mathbb{F}_p[x], \\ q^t - 1 & \text{otherwise.} \end{cases} \tag{3.18}$$

It follows that $h(x) \mid (x^\ell - 1)$. Since $(x^\ell - 1)^\sigma(x) = x^{q^\ell} - x$, has all its roots in $\mathbb{F}_{q^\ell}$, $h^\sigma(x)$ splits into linear factors over $\mathbb{F}_{q^\ell}$.

3.4.1 Efficient root-finding algorithm for linearized polynomials

Almost all of the most popular factoring and root-finding algorithms over finite fields are based on Berlekamp [34, 35] and Cantor-Zassenhaus [36] procedures. In particular, those algorithms require $\tilde{O}\Big(\left(q^d\right)^3 + \log(q^\ell) \cdot \left(q^d\right)^2\Big)$ and $\tilde{O}\Big(\left(q^d\right)^2 \cdot \log(q^\ell)\Big)$ field operations for computing all the roots of $h^\sigma(x)$, respectively (for more details, see [37]).[1] In addition, the Successive Resultant Algorithm (SRA) computes all the roots of $h^\sigma(x)$ at a cost of $\tilde{O}\big(q \cdot \ell \cdot \left(q^d\right) + \ell^2 \cdot \left(q^d\right)\big)$ field operations [38, 40, 39]. Notice that computing the roots of $h^\sigma(x)$ becomes costly when $\deg\left(h^\sigma(x)\right) = q^d$ is a large integer number.

Let us denote by V_{h^σ} the set of roots of $h^\sigma(x)$ over $\mathbb{F}_{q^\ell}$, and let $f_\ell(x) \in \mathbb{F}_q[x]$ be an irreducible degree-ℓ polynomial. Let us write $\mathbb{F}_{q^\ell} \cong \mathbb{F}_q[\eta]/\langle f_\ell(\eta)\rangle$; then, Theorem 3.4.1, and Lemma 14 hold.

Theorem 3.4.1. *V_{h^σ} forms a $\mathbb{F}_q$-vector space of dimension d. In addition, there is an efficient way of computing a basis β of V_{h^σ} over $\mathbb{F}_{q^\ell}$.*

[1] $\tilde{O}(g(n))$ means $O\Big(g(n) \cdot \left(\log g(n)\right)^k\Big)$ for some integer k.

Proof. Let $\gamma_1, \gamma_2 \in \mathrm{V}_{h^\sigma}$ and $c \in \mathbb{F}_q$. Then, $h^\sigma(c \cdot \gamma_1 + \gamma_2) = c \cdot h^\sigma(\gamma_1) + h^\sigma(\gamma_2) = c \cdot 0 + 0 = 0$. That is, V_{h^σ} forms a $\mathbb{F}_q$-vector space. Clearly, $h^\sigma(x)$ has exactly q^d zeroes. Therefore $\mid \mathrm{V}_{h^\sigma} \mid = q^d$, and any basis β of V_{h^σ} (over $\mathbb{F}_{q^\ell}$) must have d elements, i.e., V_{h^σ} has dimension d. Furthermore, any $\gamma \in \mathbb{F}_{q^\ell}$ can be writing as $\gamma = \sum_{i=0}^{\ell-1} \gamma_i \cdot \eta^i$, where each $\gamma_i \in \mathbb{F}_q$ and,

$$\begin{aligned} h^\sigma(\gamma) = \sum_{i=0}^{d} h_i \cdot \gamma^{q^i} &= \sum_{i=0}^{d} h_i \cdot \left(\sum_{j=0}^{\ell-1} \gamma_j \cdot \eta^j \right)^{q^i} = \sum_{i=0}^{d} h_i \cdot \left(\sum_{j=0}^{\ell-1} \gamma_j \cdot \left(\eta^j\right)^{q^i} \right) \\ &= \sum_{j=0}^{\ell-1} \gamma_j \cdot \left(\sum_{i=0}^{d} h_i \cdot \left(\eta^j\right)^{q^i} \right) = \sum_{j=0}^{\ell-1} \gamma_j \cdot h^\sigma\left(\eta^j\right). \end{aligned} \tag{3.19}$$

Notice $h^\sigma\left(\eta^j\right) = \sum_{k=0}^{\ell-1} \left(c_{k,j} \cdot \eta^k\right)$ with each $c_{k,j} \in \mathbb{F}_q$. Moreover, $h^\sigma(\gamma) = 0$ iff $\overrightarrow{\gamma} \cdot \mathrm{C}_{h^\sigma} = \overrightarrow{0}$, where $\overrightarrow{\gamma} = (\gamma_0, \ldots, \gamma_{\ell-1}) \in (\mathbb{F}_q)^\ell$, and $\mathrm{C}_{h^\sigma} = (c_{k,j}) \in (\mathbb{F}_q)^{\ell \times \ell}$ is the matrix with coefficients $c_{k,j} \in \mathbb{F}_q$. That is, V_{h^σ} is isomorphic to the kernel (null-space) of the matrix C_{h^σ} seen as an $\mathbb{F}_q$-vector space. In addition, the task of finding a basis $\beta \in \left(\mathbb{F}_{q^\ell}\right)^d$ of V_{h^σ} can be reduced to the problem of computing a basis of the kernel of C_{h^σ}. However, notice that the rows of C_{h^σ} are determined by the evaluations $h^\sigma(\eta^j)$, and each evaluation can be done by means of Horner's rule with $d \cdot n$ raised to the p-th power and at most $(d-1)$ additions in $\mathbb{F}_{q^\ell}$. Thus, C_{h^σ} can be computed with $d \cdot (\ell-1) \cdot n$ raised to the p-th power, $(\ell-1)$ multiplications, and at most $(\ell-1)(d-1)$ additions in $\mathbb{F}_{q^\ell}$. Gaussian elimination can be used in order to compute a basis of the kernel of C_{h^σ} by transforming the matrix $\left[\begin{array}{c|c} \mathrm{C}_{h^\sigma} & \mathrm{Id}_{\ell \times \ell} \end{array}\right]$ into its row echelon form,

$$\left[\begin{array}{c|c} \mathrm{A} & \mathrm{B} \\ \hline 0_{d \times \ell} & \mathrm{K} \end{array}\right].$$

Now, the rows of the matrix K, form a basis of $\ker \mathrm{C}_{h^\sigma}$. Moreover, the row echelon form computation of $\left[\begin{array}{c|c} \mathrm{C}_{h^\sigma} & \mathrm{Id}_{\ell \times \ell} \end{array}\right]$ requires at most $2\ell^3$ multiplications in $\mathbb{F}_q$. Once the basis K has been obtained, the next step is writing each row $\overrightarrow{k} \coloneqq (k_0, \ldots, k_{d-1})$ of K as $b \coloneqq \sum_{i=0}^{\ell-1} k_i \cdot \eta^i$. Thus, the cost of computing a basis $\beta \in \left(\mathbb{F}_{q^\ell}\right)^d$ of V_{h^σ} is at most $d \cdot (\ell-1) \cdot n$ raised to the p-th power, [1] $(3\ell-1)$ multiplications, and $(\ell-1)(2d-1)$ additions in $\mathbb{F}_{q^\ell}$. Algorithm 3.1 implements the above procedure. □

Lemma 3.4.1. *All the roots in $\mathbb{F}_{q^\ell}$ of $h^\sigma(x)$ can be computed at a cost of at most $\left(3\ell - 1 + \frac{d \cdot (\ell-1) \cdot n \cdot \log_2(p)}{2}\right)$ multiplications, $\left(d \cdot (\ell-1) \cdot n \cdot \log_2(p)\right)$ squarings and $\left[(\ell-1)(2d-1) + q^d\right]$ additions in $\mathbb{F}_{q^\ell}$* [2].

[1] One single exponentiation to the p-th power requires $\log_2(p)$ squaring and $\frac{\log_2(p)}{2}$ multiplication operations.

[2] The computation of all the linear combinations of $\langle \bar{b}_1, \ldots, \bar{b}_d \rangle$ requires q^d additions in $\mathbb{F}_{q^\ell}$.

Algorithm 3.1: Root-finding algorithm for linearized polynomials.

Input: $q = p^n$, ℓ, a degree-d $h(x) \in \mathbb{F}_q[x]$, and $\mathbb{F}_{q^\ell} = \mathbb{F}_q[\eta]/\langle f_\ell(\eta)\rangle$
Output: $\mathrm{V}_{h^\sigma} = \langle b_0, \ldots, b_{d-1}\rangle_{\mathbb{F}_q}$

$C_{h^\sigma} \leftarrow 0_{\ell\times\ell} \in (\mathbb{F}_q)^{\ell\times\ell}$;
for $j \coloneqq 0$ *to* $\ell - 1$ **do**
  $(C_{j,0}, \ldots, C_{j,\ell-1}) \leftarrow$ the coefficients of $h^\sigma(\eta^j) \coloneqq \sum_{i=0}^{\ell-1} C_{j,i} \cdot \eta^j$
$\left[\begin{array}{c|c} \mathrm{A} & \mathrm{B} \\ \hline 0_{s\times\ell} & \mathrm{K} \end{array}\right] \leftarrow$ the echelon form of $\left[\begin{array}{c|c} \mathrm{C}_{h^\sigma} & \mathrm{Id}_{\ell\times\ell} \end{array}\right]$;
for $j \coloneqq 0$ *to* $d - 1$ **do**
  $b_j \leftarrow \sum_{i=0}^{\ell-1} \mathrm{K}_{j,i} \cdot \eta^i$;
return $\mathrm{V}_{h^\sigma} \leftarrow \langle b_0, \ldots, b_{d-1}\rangle_{\mathbb{F}_q}$

3.4.2 Comparisons and experiments

In this Section we present a Magma-code implementation of algorithm 3.1, and compare it with the Schonhage, Berlekamp, and von zur Gathen-Kaltofen-Shoup (GKS) root-finding algorithms already available in Magma, along with the SRA technique as presented in [38] (cf. table 3.2). Our code is available at

https://github.com/JJChiDguez/root-finding

Our experiments were done on an Intel Core i7-8550U CPU 1.80GHz machine with 16GB of RAM. We noticed that Magma was not able to compute a degree-q^d polynomial with coefficients in $\mathbb{F}_q$ and $q^d \geq 2^{30}$. For this instance of the problem only our method could complete the computation. Table 3.3 shows the measured timings for computing all the roots of $h^\sigma(x)$ when $q^\ell \approx 2^{2048}$.

p	n	ℓ	d	`Roots()`	`Factorization()`		basic SRA [38]	**This work**
					Berlekamp	GKS		
Case $\ell = p^d - 1$								
2	3	7	3	0.040	0.030	0.020	1.040	**0.010**
3	2	26	3	0.110	0.120	0.110	58.880	**0.010**
5	2	24	2	0.050	0.060	0.070	10.500	**0.020**
7	2	48	2	1.240	1.270	1.280	1702.880	**0.020**
Case $\ell = q^d - 1$								
2	3	63	2	0.110	0.050	0.040	2.390	**0.010**
3	2	80	2	7.420	1.630	1.500	20.020	**0.010**

Table 3.2: CPU seconds required for finding all the q^d roots of $h^\sigma(x)$ in $\mathbb{F}_{q^\ell}$. The Magma algorithms used were the Schonhage, Berlekamp, and von zur Gathen-Kaltofen-Shoup algorithms; which were invoked as `Roots`$(h^\sigma(x)$: Al:="Schonhage", IsSquarefree:=true), `Factorization`$(h^\sigma(x))$, and `Factorization`$(h^\sigma(x)$: Al := "GKS"), respectively.

p	2	3	5	7	11	13
n	2	5	7	5	5	3
l	1023	242	124	342	120	168
d	10	5	3	3	2	2
$\log_2\left(q^d\right)$	20.000	39.624	48.760	42.110	34.594	22.203
CPU time	0.690	0.080	0.200	0.480	0.280	0.910

Table 3.3: CPU seconds required for finding all the q^d roots $h^\sigma(x)$ in $\mathbb{F}_{q^\ell}$.

3.4.3 Finding instances of elliptic curves for which the gGHS Weil descent becomes effective

Finally, when the gGHS Weil descent technique is applied to the elliptic curve $\mathcal{E}/\mathbb{F}_{q^\ell}$ with $q = 2^n$, the genus of the resulting curve $\mathcal{H}/\mathbb{F}_q$ is given as [29], $g = 2^m - 2^{m-\deg\left(\mathrm{Ord}_{\gamma_1}(x)\right)} - 2^{m-\deg\left(\mathrm{Ord}_{\gamma_2}(x)\right)} + 1$, where $\mathrm{Ord}_\gamma(x)$ denotes the irreducible polynomial of minimal degree d that satisfies $\left(\mathrm{Ord}_\gamma\right)^\sigma(\gamma) = 0$, and m is the degree of the following polynomial

$$\mathrm{Ord}_{\gamma_1,\gamma_2,\gamma_3}(x) = \begin{cases} \mathrm{lcm}\left(\mathrm{Ord}_{\gamma_1}(x), \mathrm{Ord}_{\gamma_2}(x)\right) & \text{if } \mathrm{Tr}_{\mathbb{F}_{q^\ell}/\mathbb{F}_2}(\gamma_3) = 0, \\ \mathrm{lcm}\left(\mathrm{Ord}_{\gamma_1}(x), \mathrm{Ord}_{\gamma_2}(x), x+1\right) & \text{otherwise.} \end{cases}$$

The curve $\mathcal{E}/\mathbb{F}_{q^\ell}$ is defined by following equation,

$$\mathcal{E}/\mathbb{F}_{q^\ell} : y^2 + x \cdot y = x^3 + \gamma_3 \cdot x^2 + \left(\gamma_1 \cdot \gamma_2\right)^2, \quad \gamma_3 \in \mathbb{F}_{q^\ell}, \quad \text{and } \gamma_1, \gamma_2 \in \left(\mathbb{F}_{q^\ell}\right)^\times.$$

As an implication of the Florian Hess work, theorem 3.4.2 gives a novel characterization of elliptic curves $\mathcal{E}/\mathbb{F}_{q^\ell}$ for which the gGHS Weil descent is effective in the sense that the resulting (non)-hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ has genus $g \approx \ell$.

Theorem 3.4.2. *Let $q = 2^n$ and $\ell \coloneqq 2^t - 1$ be a Mersenne's prime. Then, there are elliptic curves $\mathcal{E}/\mathbb{F}_{q^\ell}$ such that the gGHS Weil descent permits to construct a (non-)hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ with genus $g \approx \ell$.*

Proof. First, the polynomial $(x^\ell - 1) = \frac{x^{2^t} - x}{x}$ gives all the degree-t irreducible polynomials $f_k(x)$ in $\mathbb{F}_2[x]$. Now, let $\gamma_1 \in \mathbb{F}_{q^\ell}$, $\gamma_2, \gamma_3 \in \left(\mathbb{F}_{q^\ell}\right)^\times$, and $\mathrm{Ord}_{\gamma_i}(x) = (x-1)^{j_i} \cdot \left(f_k(x)\right)^{m_i}$ with $m_i, j_i \in \{0, 1\}$ such that $(m_i + j_i) \neq 0$ and $(m_1 + m_2) \neq 0$. Moreover, $\deg\left(\mathrm{Ord}_{\gamma_i}(x)\right) = (m_i \cdot t + j_i) \in \{1, t, t+1\}$. In additon,

$$\mathrm{Ord}_{\gamma_1,\gamma_2,\gamma_3}(x) = \begin{cases} (x-1)^{\max\{j_1,j_2\}} \cdot f_k(x) & \text{if } \mathrm{Tr}_{\mathbb{F}_{q^\ell}/\mathbb{F}_2}(\gamma_3) = 0, \\ (x-1) \cdot f_k(x) & \text{otherwise.} \end{cases}.$$

Then, $m \coloneqq \deg\left(\mathrm{Ord}_{\gamma_1,\gamma_2,\gamma_3}(x)\right) \in \left\{\max\{j_1, j_2\} + t, 1 + t\right\}$, and the integer $g = 2^m - 2^{m-\deg\left(\mathrm{Ord}_{\gamma_1}(x)\right)} - 2^{m-\deg\left(\mathrm{Ord}_{\gamma_2}(x)\right)} + 1$ can take the values ℓ, $\ell+1$, $2\ell-1$,

2ℓ, or $2\ell+1$; that is, $g \approx \ell$. Consequently, the elliptic curve $\mathcal{E}/\mathbb{F}_{q^\ell}: y^2 + x \cdot y = x^3 + \gamma_3 \cdot x^2 + \left(\gamma_1 \cdot \gamma_2\right)^2$ is such that the gGHS Weil descent permits to construct a (non-)hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ of genus $g \approx \ell$. □

Finally, algorithm 3.1 permits to efficiently compute all the roots of the polynomials $\mathrm{Ord}_{\gamma_1}(x)$ and $\mathrm{Ord}_{\gamma_2}(x)$. Thus, it is possible to efficiently find elliptic curves $\mathcal{E}/\mathbb{F}_{q^\ell}$ such that the gGHS Weil descent technique permits to construct a hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ of genus $g \approx \ell$.

Example: let $q = 2^2$, $\ell = 2^7 - 1$, $\mathbb{F}_q = \mathbb{F}_2[\nu]\langle\nu^2+\nu+1\rangle$ and $\mathbb{F}_{q^\ell} = \mathbb{F}_q[\eta]/\langle\eta^{127}+\eta+1\rangle$. Then $\gamma_3 = 0$ and $\gamma_1, \gamma_2 \coloneqq \gamma = \nu \cdot$ `0x1170004172B01161470037B2A0A` . [1] Define the elliptic curve $\mathcal{E}/\mathbb{F}_{q^\ell}: y^2 + x \cdot y = x^3 + \gamma^4$ for which the gGHS Weil descent constructs an hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ of genus $g = 127$. In particular, $\mathrm{Ord}_\gamma(x) = (x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1) \in \mathbb{F}_2[x]$ and, $\#\mathcal{E}(\mathbb{F}_{q^\ell}) = (2^8 \cdot 3) \cdot r$, with $r =$ `0x155555555555555555555555555555555A7B52674132A255076D8A961E886A1` is 245-bit prime integer number.

[1] The j-th bit of γ corresponds with j-th coefficient in $\mathbb{F}_2[\eta]/\langle\eta^{127}+\eta+1\rangle$.

Chapter 4

On the Cost of Computing Isogenies Between Supersingular Elliptic Curves

The beauty of mathematics only shows itself to more patient followers.

Maryam Mirzakhani

The security of the Jao-De Feo Supersingular Isogeny Diffie-Hellman (SIDH) key agreement scheme is based on the intractability of the Computational Supersingular Isogeny (CSSI) problem — computing $\mathbb{F}_{p^2}$-rational isogenies of degrees 2^e and 3^e between certain supersingular elliptic curves defined over $\mathbb{F}_{p^2}$. The classical meet-in-the-middle attack on CSSI has an expected running time of $O(p^{1/4})$, but also has $O(p^{1/4})$ storage requirements. In this chapter, we demonstrate that the van Oorschot-Wiener golden collision finding algorithm has a lower cost (but higher running time) for solving CSSI, and thus should be used instead of the meet-in-the-middle attack to assess the security of SIDH against classical attacks. The smaller parameter p brings significantly improved performance for SIDH.

4.1 Supersingular elliptic curves and isogenies

Let $p = \ell_A^{e_A}\ell_B^{e_B} - 1$ be a prime[1], where ℓ_A and ℓ_B are distinct small primes and $\ell_A^{e_A} \approx \ell_B^{e_B} \approx p^{1/2}$. Let $\mathcal{E}$ be a (supersingular) elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#\mathcal{E}(\mathbb{F}_{p^2}) = (p+1)^2$. Then $\mathcal{E}(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$, whence the torsion groups

[1] More generally, one can take $p = \ell_A^{e_A}\ell_B^{e_B} d \pm 1$ where d is a small cofactor.

$\mathcal{E}[\ell_A^{e_A}]$ and $\mathcal{E}[\ell_B^{e_B}]$ are contained in $\mathcal{E}(\mathbb{F}_{p^2})$ [1].

In the following, we write (ℓ, e) to mean either (ℓ_A, e_A) or (ℓ_B, e_B). All isogenies ϕ considered in this chapter are separable, whereby $\deg \phi = \#\mathrm{Ker}(\phi)$.

Let S be an order-ℓ^e subgroup of $\mathcal{E}[\ell^e]$. Then there exists an isogeny $\phi : \mathcal{E} \to \mathcal{E}'$ (with both ϕ and $\mathcal{E}'$ defined over $\mathbb{F}_{p^2}$) with kernel S. The isogeny ϕ is unique up to isomorphism in the sense that if $\tilde{\phi} : \mathcal{E} \to \tilde{\mathcal{E}}$ is another isogeny defined over $\mathbb{F}_{p^2}$ with kernel S, then there exists an $\mathbb{F}_{p^2}$-isomorphism $\psi : \mathcal{E}' \to \tilde{\mathcal{E}}$ with $\tilde{\phi} = \psi \circ \phi$.

Given $\mathcal{E}$ and S, an isogeny ϕ with kernel S and the equation of $\mathcal{E}'$ can be computed using Vélu's formulas [67]. The running time of Vélu's formulas is polynomial in $\#S$ and $\log p$. Since $\#S \approx p^{1/2}$, a direct application of Vélu's formulas does not yield a polynomial-time algorithm for computing ϕ and $\mathcal{E}'$. However, since $\#S$ is a power of a small prime, one can compute ϕ and $\mathcal{E}'$ in time that is polynomial in $\log p$ by using Vélu's formulas to compute a sequence of e degree-ℓ isogenies.

We will denote the elliptic curve that Vélu's formulas yields by $\mathcal{E}/S$ and the (Vélu) isogeny by $\phi_S : \mathcal{E} \to \mathcal{E}/S$. As noted above, ϕ_S is unique up to isomorphism. Thus, for any fixed $\mathcal{E}$, there is a one-to-one correspondence between order-ℓ^e subgroups of $\mathcal{E}[\ell^e]$ and degree-ℓ^e isogenies $\phi : \mathcal{E} \to \mathcal{E}'$ defined over $\mathbb{F}_{p^2}$. It follows that the number of degree-ℓ^e isogenies $\phi : \mathcal{E} \to \mathcal{E}'$ is $\ell^e + \ell^{e-1} = (\ell+1)\ell^{e-1}$.

Vélu's formulas (see [44]) can be used to compute degree-ℓ isogenies. We present Vélu's formulas for $\ell = 2$ and $\ell = 3$.

Consider the elliptic curve $\mathcal{E}/\mathbb{F}_{p^2} : y^2 = x^3 + ax + b$, and let $\mathsf{P} = (x_\mathsf{P}, y_\mathsf{P}) \in \mathcal{E}(\mathbb{F}_{p^2})$ be a point of order two. Let $v = 3x_\mathsf{P}^2 + a$, $a' = a - 5v$, $b' = b - 7vx_\mathsf{P}$, and define the elliptic curve $\mathcal{E}'/\mathbb{F}_{p^2} : y^2 = x^3 + a'x + b'$. Then the map

$$(x, y) \mapsto \left(x + \frac{v}{x - x_\mathsf{P}},\ y - \frac{vy}{(x - x_\mathsf{P})^2} \right)$$

is a degree-2 isogeny from $\mathcal{E}$ to $\mathcal{E}'$ with kernel $\langle \mathsf{P} \rangle$.

Let $\mathsf{P} = (x_\mathsf{P}, y_\mathsf{P}) \in \mathcal{E}(\mathbb{F}_{p^2})$ be a point of order three. Let $v = 6x_\mathsf{P}^2 + 2a$, $u = 4y_\mathsf{P}^2$, $a' = a - 5v$, $b' = b - 7(u + vx_\mathsf{P})$, and define the elliptic curve $\mathcal{E}'/\mathbb{F}_{p^2} : y^2 = x^3 + a'x + b'$. Then the map

$$(x, y) \mapsto \left(x + \frac{v}{x - x_\mathsf{P}} + \frac{u}{(x - x_\mathsf{P})^2},\ y\left(1 - \frac{v}{(x - x_\mathsf{P})^2} - \frac{2u}{(x - x_\mathsf{P})^3}\right) \right)$$

is a degree-3 isogeny from $\mathcal{E}$ to $\mathcal{E}'$ with kernel $\langle \mathsf{P} \rangle$.

Suppose now that $\mathsf{R} \in \mathcal{E}(\mathbb{F}_{p^2})$ has order ℓ^e where $\ell \in \{2, 3\}$ and $e \geq 1$. Then the isogeny $\phi : \mathcal{E} \to \mathcal{E}/\langle \mathsf{R} \rangle$ can be efficiently computed as follows. Define $\mathcal{E}_0 = \mathcal{E}$ and $\mathsf{R}_0 = \mathsf{R}$. For $i = 0, 1, \ldots, e-1$, let $\phi_i : \mathcal{E}_i \to \mathcal{E}_{i+1}$ be the degree-ℓ isogeny obtained using Vélu's formulas with kernel $\langle \ell^{e-1-i}\mathsf{R}_i \rangle$, and let $\mathsf{R}_{i+1} = \phi_i(\mathsf{R}_i)$. Then $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$.

[1] The torsion-m subgroup is defined as $\mathcal{E}[m] \coloneqq \{\mathsf{P} \in \mathcal{E}(\mathbb{F}_{p^2}) \colon [m]\mathsf{P} = \mathcal{O}\}$.

Remark 4.1.1. (*cost of computing an ℓ^e-isogeny*) As shown in [49], a 'balanced strategy' for computing a degree-ℓ^e isogeny requires approximately $\frac{e}{2}\log_2 e$ point multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves. Also presented in [49] is a slightly faster 'optimal strategy' that accounts for the relative costs of a point multiplication and a degree-ℓ isogeny evaluation.

4.2 SIDH protocol

In SIDH, the parameters ℓ_A, ℓ_B, e_A, e_B, p and $\mathcal{E}$ are fixed and public, as are bases $\{\mathsf{P}_A, \mathsf{Q}_A\}$ and $\{\mathsf{P}_B, \mathsf{Q}_B\}$ for the torsion groups $\mathcal{E}[\ell_A^{e_A}]$ and $\mathcal{E}[\ell_B^{e_B}]$.

In (unauthenticated) SIDH, Alice selects $m_A, n_A \in_R [0, \ell_A^{e_A} - 1]$, not both divisible by ℓ_A, and sets $\mathsf{R}_A = m_A\mathsf{P}_A + n_A\mathsf{Q}_A$ and $A = \langle \mathsf{R}_A \rangle$; note that A is an order-$\ell_A^{e_A}$ subgroup of $\mathcal{E}[\ell_A^{e_A}]$. Alice then computes the isogeny $\phi_A : \mathcal{E} \to \mathcal{E}/A$ while keeping A and ϕ_A secret. She transmits

$$\mathcal{E}/A, \quad \phi_A(\mathsf{P}_B), \quad \phi_A(\mathsf{Q}_B)$$

to Bob. Similarly, Bob selects $m_B, n_B \in_R [0, \ell_B^{e_B} - 1]$, not both divisible by ℓ_B, and sets $\mathsf{R}_B = m_B\mathsf{P}_B + n_B\mathsf{Q}_B$ and $B = \langle \mathsf{R}_B \rangle$. Bob then computes the isogeny $\phi_B : \mathcal{E} \to \mathcal{E}/B$. He keeps B and ϕ_B secret and transmits

$$\mathcal{E}/B, \quad \phi_B(\mathsf{P}_A), \quad \phi_B(\mathsf{Q}_A)$$

to Alice. Thereafter, Alice computes $\phi_B(\mathsf{R}_A) = m_A\phi_B(\mathsf{P}_A) + n_A\phi_B(\mathsf{Q}_A)$ and

$$(\mathcal{E}/B)/\langle \phi_B(\mathsf{R}_A) \rangle,$$

whereas Bob computes $\phi_A(\mathsf{R}_B) = m_B\phi_A(\mathsf{P}_B) + n_B\phi_A(\mathsf{Q}_B)$ and

$$(\mathcal{E}/A)/\langle \phi_A(\mathsf{R}_B) \rangle.$$

The compositions of isogenies

$$\mathcal{E} \to \mathcal{E}/A \to (\mathcal{E}/A)/\langle \phi_A(\mathsf{R}_B) \rangle$$

and

$$\mathcal{E} \to \mathcal{E}/B \to (\mathcal{E}/B)/\langle \phi_B(\mathsf{R}_A) \rangle$$

both have kernel $\langle \mathsf{R}_A, \mathsf{R}_B \rangle$. Hence the elliptic curves computed by Alice and Bob are isomorphic over $\mathbb{F}_{p^2}$, and their shared secret k is the j-invariant of these curves.

Remark 4.2.1. (*SIDH vs. SIKE*) SIDH is an unauthenticated key agreement protocol. The NIST submission [53] specifies a variant of SIDH that is a key encapsulation mechanism (KEM) called SIKE (Supersingular Isogeny Key Encapsulation). In SIKE, Alice's long-term public key is $(\mathcal{E}/A, \phi_A(\mathsf{P}_B), \phi_A(\mathsf{Q}_B))$. Bob sends Alice an ephemeral public key $(\mathcal{E}/B, \phi_B(\mathsf{P}_A), \phi_B(\mathsf{Q}_A))$ where B is derived from Alice's public key and a random string, and then computes a session

key from the j-invariant of the elliptic curve $(\mathcal{E}/A)/\langle\phi_A(\mathsf{R}_B)\rangle$, the aforementioned random string, and the ephemeral public key. One technical difference between the original SIDH specification in [54, 49] and the SIKE specification in [53] (and also the SIDH implementation in [45]) is that in the latter the secret R_A is of the form $\mathsf{P}_A + n_A\mathsf{Q}_A$ where n_A is selected (almost) uniformly at random from the interval $[0, \ell_A^{e_A} - 1]$ (and similarly for R_B). Thus, R_A is selected uniformly at random from a subset of size approximately ℓ^{e_A} of the set of all order-$\ell_A^{e_A}$ subgroups (which has cardinality $\ell_A^{e_A} + \ell_A^{e_A-1}$).

The challenge faced by a passive adversary is to compute k given the public parameters, $\mathcal{E}/A$, $\mathcal{E}/B$, $\phi_A(\mathsf{P}_B)$, $\phi_A(\mathsf{Q}_B)$, $\phi_B(\mathsf{P}_A)$ and $\phi_B(\mathsf{Q}_A)$. 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, $\mathcal{E}$, P_A, Q_A, P_B, Q_B, the elliptic curve $\mathcal{E}/A$, and the auxiliary points $\phi_A(\mathsf{P}_B)$ and $\phi_A(\mathsf{Q}_B)$, compute the Vélu isogeny $\phi_A : \mathcal{E} \to \mathcal{E}/A$ (or, equivalently, determine a generator of A).

An assumption one makes (e.g., see [49]) is that the auxiliary points $\phi_A(\mathsf{P}_B)$ and $\phi_A(\mathsf{Q}_B)$ are of no use in solving CSSI. Thus, we can simplify the statement of the CSSI problem to the following:

Poblem 4.2.1 (CSSI)**.** Given the public parameters ℓ_A, ℓ_B, e_A, e_B, p, $\mathcal{E}$, P_A, Q_A, and the elliptic curve $\mathcal{E}/A$, compute a degree-$\ell_A^{e_A}$ isogeny $\phi_A : \mathcal{E} \to \mathcal{E}/A$.

4.3 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 $\mathcal{E}[\ell^e]$ by $N = (\ell+1)\ell^{e/2-1} \approx p^{1/4}$.

Let $\mathcal{E}_1 = \mathcal{E}$ and $\mathcal{E}_2 = \mathcal{E}/A$. Let R denote the set of all j-invariants of elliptic curves that are isogenous to $\mathcal{E}_1$; then $\#R \approx p/12$ [63]. 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 $\mathcal{E}_1$. Since $\#R \gg N$, one expects that the number of pairs of distinct order-$\ell^{e/2}$ subgroups (A_1, A_2) of $\mathcal{E}_1[\ell^e]$ with $j(\mathcal{E}_1/A_1) = j(\mathcal{E}_1/A_2)$ 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 $\mathcal{E}_2$, and assume that $\#R_2 = N$. Since $\mathcal{E}_1$ is ℓ^e-isogenous to $\mathcal{E}_2$, 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 $\phi : \mathcal{E}_1 \to \mathcal{E}_2$.

4.3.1 Basic method

The meet-in-the-middle attack on CSSI [49], which we denote by MITM-basic, proceeds by building a (sorted) table with entries $(j(\mathcal{E}_1/A_1), A_1)$, where A_1 ranges over all order-$\ell^{e/2}$ subgroups of $\mathcal{E}_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $\mathcal{E}_2[\ell^e]$, one computes $\mathcal{E}_2/A_2$ and searches for $j(\mathcal{E}_2/A_2)$ in the table (see

Figure 4.1). If $j(\mathcal{E}_2/A_2) = j(\mathcal{E}_1/A_1)$, then the composition of isogenies

$$\phi_{A_1} : \mathcal{E}_1 \to \mathcal{E}_1/A_1, \quad \psi : \mathcal{E}_1/A_1 \to \mathcal{E}_2/A_2, \quad \hat{\phi}_{A_2} : \mathcal{E}_2/A_2 \to \mathcal{E}_2,$$

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 $\mathcal{E}_1$ to $\mathcal{E}_2$. 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 4.1.1). The average-case time complexity is $1.5N$. The attack has space complexity N.

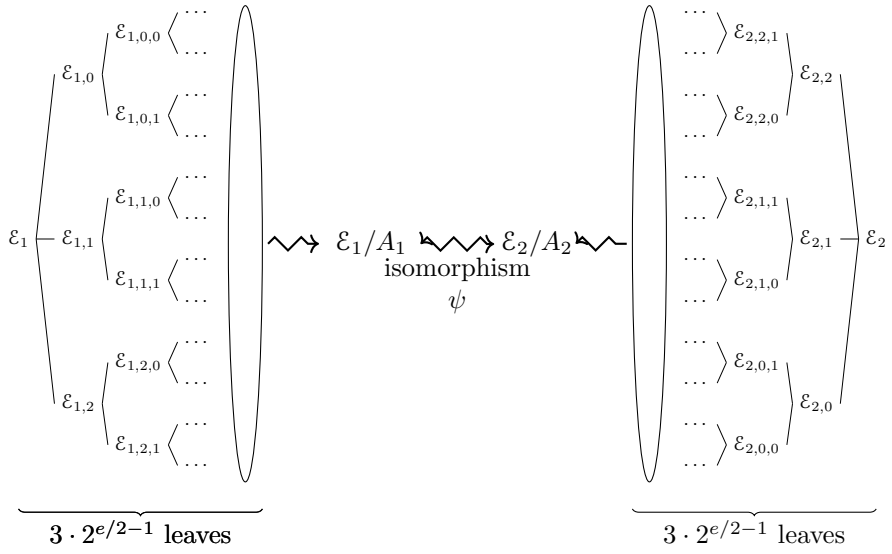

Figure 4.1: Meet-in-the-middle attack for degree-2 isogeny trees.

4.3.2 Depth-first search

The set of pairs $(j(\mathcal{E}/A), A)$, with A ranging over all order-$\ell^{e/2}$ subgroups of $\mathcal{E}[\ell^e]$, can also be generated by using a depth-first search (DFS) to traverse the tree in the left of Figure 4.1 (and also the tree in the right of Figure 4.1). 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 $\{\mathsf{P}, \mathsf{Q}\}$ be a basis for $\mathcal{E}[2^{e/2}]$. Let $\mathsf{R}_0 = 2^{e/2-1}\mathsf{P}$, $\mathsf{R}_1 = 2^{e/2-1}\mathsf{Q}$, $\mathsf{R}_2 = \mathsf{R}_0 + \mathsf{R}_1$ be the order-2 points on $\mathcal{E}$. For $i = 0, 1, 2$, the degree-2 isogenies $\phi_i : \mathcal{E} \to \mathcal{E}_i = \mathcal{E}/\langle \mathsf{R}_i \rangle$ are computed, as are bases $\{\mathsf{P}_0 = \phi_0(\mathsf{P}), \mathsf{Q}_0 = \phi_0(2\mathsf{Q})\}$, $\{\mathsf{P}_1 = \phi_1(\mathsf{Q}), \mathsf{Q}_1 = \phi_1(2\mathsf{P})\}$, $\{\mathsf{P}_2 = \phi_2(\mathsf{P} + \mathsf{Q}), \mathsf{Q}_2 = \phi_2(2\mathsf{P})\}$ for $\mathcal{E}_0[2^{e/2-1}]$, $\mathcal{E}_1[2^{e/2-1}]$, $\mathcal{E}_2[2^{e/2-1}]$, respectively. A memory stack is initialized with the tuples

[1] For the sake of concreteness, all implementation reports of CSSI attacks in this section 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.

$(\mathcal{E}_0, 0, \mathsf{P}_0, \mathsf{Q}_0)$, $(\mathcal{E}_1, 1, \mathsf{P}_1, \mathsf{Q}_1)$, $(\mathcal{E}_2, 2, \mathsf{P}_2, \mathsf{Q}_2)$, and the tuple on the top of the stack is processed recursively as described next.

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

(i) If $n < e/2 - 1$, then let $\mathsf{P}_{x0} = \phi_{x0}(\mathsf{P}_x)$, $\mathsf{Q}_{x0} = \phi_{x0}(2(\mathsf{P}_x + \mathsf{Q}_x))$, $\mathsf{P}_{x1} = \phi_{x1}(\mathsf{P}_x + \mathsf{Q}_x)$, $\mathsf{Q}_{x1} = \phi_{x1}(2\mathsf{P}_x)$; one can check that $\{\mathsf{P}_{xi}, \mathsf{Q}_{xi}\}$ is a basis for $\mathcal{E}_{xi}[2^{e/2-n-1}]$ for $i = 0, 1$. Then, $(\mathcal{E}_{x1}, x1, \mathsf{P}_{x1}, \mathsf{Q}_{x1})$ is added to the stack and $(\mathcal{E}_{x0}, x0, \mathsf{P}_{x0}, \mathsf{Q}_{x0})$ is processed next.

(ii) If $n = e/2 - 1$, the leaves $(j(\mathcal{E}_{x0}), x0)$ and $(j(\mathcal{E}_{x1}), x1)$ 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 4.1.1). 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 $\frac{e}{12.4}\log_2 e$.

4.3.3 Implementation report

The MITM-basic and MITM-DFS attacks (for $\ell = 2$) were implemented in C, compiled using gcc version 4.7.2, and executed on an Intel Xeon processor E5-2658 v2 server of 2.40GHz equipped with 20 physical cores and 256 GB of shared RAM memory; we used fopenmp for the parallelization, and the turbo boost was enabled. Our code for the MITM-basic, MITM-DFS and VW golden collision search CSSI attacks is available at

https://github.com/JJChiDguez/CSSI

For $p = 2^{e_A}3^{e_B}d - 1$, the elliptic curve $\mathcal{E}/\mathbb{F}_p : y^2 = x^3 + x$ has $\#\mathcal{E}(\mathbb{F}_p) = p+1$ and $\#\mathcal{E}(\mathbb{F}_{p^2}) = (p+1)^2$. A point $\mathsf{P} \in \mathcal{E}(\mathbb{F}_{p^2})$ of order $2 \cdot 3 \cdot d$ was randomly selected, and the isogenous elliptic curve $\mathcal{E}_1 = \mathcal{E}/\langle \mathsf{P} \rangle$ was computed. Then, a random order-2^{e_A} subgroup A of $\mathcal{E}_1(\mathbb{F}_{p^2})$ was selected, and the isogenous elliptic curve $\mathcal{E}_2 = \mathcal{E}_1/A$ was computed. Our CSSI challenge was to find a generator of A given $\mathcal{E}_1$ and $\mathcal{E}_2$.

We used Jacobian coordinates for elliptic curve arithmetic, isogeny computations, and isogeny evaluations. For MITM-basic, the leaves of the $\mathcal{E}_1$-rooted tree shown in Figure 4.1 were generated as follows. Let $\{\mathsf{P}, \mathsf{Q}\}$ be a basis for $\mathcal{E}_1[2^{e/2}]$. Then for each pair $(b, k) \in \{0, 1, 2\} \times \{0, 1, \ldots, 2^{e/2-1} - 1\}$, triples

$$\begin{aligned} \left(j(\mathcal{E}_1/\langle \mathsf{P} + (b2^{e/2-1} + k)\mathsf{Q}\rangle),\ \ b,\ \ b2^{e/2-1} + k\right), \qquad & \text{for } b = 0, 1, \\ \left(j(\mathcal{E}_1/\langle (2k)\mathsf{P} + \mathsf{Q}\rangle),\ \ b,\ \ k\right), \qquad & \text{for } b = 2, \end{aligned}$$

were computed and stored in 20 tables sorted by j-invariant (each of the 20 cores was responsible for generating a portion of the leaves). The 20 tables were stored in shared RAM memory.

MITM-DFS was executed using 12 cores. Each core was responsible for generating a portion of the leaves, and the 12 sets of leaves were stored in shared RAM memory. Table 4.1 shows the time expended for finding 2^e-isogenies for $e \in \{32, 34, 36, 38, 40, 42, 44\}$ with the MITM-basic and MITM-DFS attacks. These experimental results confirm the accuracy of the attacks' heuristic analysis.

			MITM-basic				MITM-DFS
			expected		measured	clock	clock
e_A	e_B	d	time	space	time	cycles	cycles
32	20	23	$2^{17.17}$	$2^{20.72}$	$2^{17.26}$	$2^{34.50}$	$2^{31.73}$
34	21	109	$2^{18.17}$	$2^{21.83}$	$2^{18.24}$	$2^{35.49}$	$2^{32.71}$
36	22	31	$2^{19.17}$	$2^{22.87}$	$2^{19.14}$	$2^{36.43}$	$2^{33.67}$
38	23	271	$2^{20.17}$	$2^{23.99}$	$2^{20.20}$	$2^{37.59}$	$2^{34.60}$
40	25	71	$2^{21.17}$	$2^{25.04}$	$2^{21.15}$	$2^{38.63}$	$2^{35.71}$
42	26	37	$2^{22.17}$	$2^{26.09}$	$2^{22.11}$	$2^{39.83}$	$2^{36.78}$
44	27	37	$2^{23.17}$	$2^{27.14}$	$2^{23.25}$	$2^{41.07}$	$2^{37.87}$

Table 4.1: Meet-in-the-middle attacks 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, 25 randomly generated CSSI instances were solved and the average of the results are reported. The 'expected time' and 'measured time' columns give the expected number and the actual number of degree-$2^{e_A/2}$ isogeny computations for MITM-basic. The space is measured in bytes.

4.4 Golden collision search

4.4.1 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 [60] 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 4.2). 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 4.3).

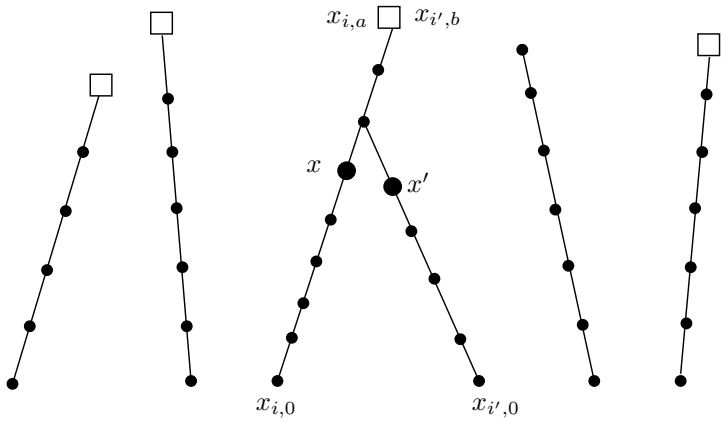

Figure 4.2: VW method: detecting a collision (x, x').

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

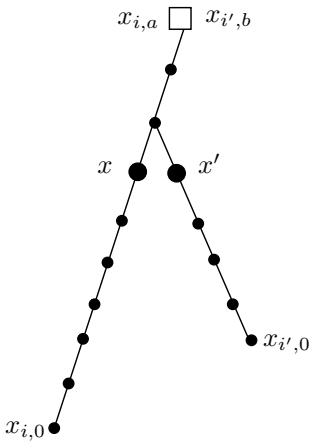

Figure 4.3: VW method: finding a collision (x, x').

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 $\theta\sqrt{\pi M/2}$.

4.4.2 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 [59]), 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

$$\theta = \alpha\sqrt{w/M} \tag{4.1}$$

and using each version of f to produce βw distinguished points. Experimental data presented in [60] suggested that the total running time to find the golden collision is minimized by setting $\alpha = 2.25$ and $\beta = 10$. Then, for $2^{10} \le w \le M/2^{10}$, the expected running time to find the golden collisions

when m processors are employed is slightly overestimated as

$$\frac{1}{m}(2.5\sqrt{M^3/w}). \tag{4.2}$$

Remark 4.4.1. (*verifying the VW heuristic analysis*) The running time estimate (4.2) 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 [60]. Then the probability that a particular version of f yields the golden collision is approximately $1.1w/(M/2)$, whence the ex-pected 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 4.2 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 [60].

4.4.3 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 $\mathcal{E}_i[\ell^e]$, define $f_i : \mathcal{A}_i \to \mathsf{R}_i$ by $f_i(A_i) = j(\mathcal{E}_i/A_i)$, 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 $\mathcal{E}_1[\ell^e]$, $\mathcal{E}_2[\ell^e]$, respectively, with $j(\mathcal{E}_1/A_1) = j(\mathcal{E}_2/A_2)$. 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

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

Remark 4.4.2. (*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 [60]).

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 4.2: Observed number $c_1 w$ of collisions and number $c_2 w$ 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 $w \geq 2^9$.

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

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

4.4.4 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 §4.3.3.

Let $\{\mathsf{P}_1, \mathsf{Q}_1\}$, $\{\mathsf{P}_2, \mathsf{Q}_2\}$ be bases for $\mathcal{E}_1[2^{e/2}]$, $\mathcal{E}_2[2^{e/2}]$, 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

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

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 4.1. We then define the version $f_n : S \to S$ of f by $(i, x) \mapsto g_n(f_i(h_i(x)))$.

Algorithm 4.1: The "random" function g_n

Input: $n \in \{0, 1\}^{64}$ and $j \in \mathbb{F}_{p^2}$.
Output: $c \in \{1, 2\}$, $b \in I_1$, $k \in I_2$.

```
counter := 0;
repeat
    h ← 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 ≠ 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 $\mathrm{MD5}(2, X)$ 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 $\mathrm{MD5}(3, X)$. If a distinguished point is not encountered after $10/\theta$ iterations, then that trail is abandoned and a new trail is formed.

Table 4.3 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 4.4, confirm the $1.3w$ and $1.1w$ estimates in [60].

4.5 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.

						median			average		
e_A	e_B	d	w	expected time	number of runs	# f_n's	measured time	clock cycles	# f_n's	measured time	clock 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 4.3: 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.

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*[1].

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 [42].

5. *Custom-designed devices*: the possible speedups that can be achieved by executing the algorithm on custom-designed hardware. Examples of such devices are TWINKLE [64] and TWIRL [65] 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.

[1] 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 4.4: Observed number $c_1 w$ of collisions and number $c_2 w$ 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).

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 4.5.1. (*feasible amount of storage and number of processors*) The Sun-way TaihuLight supercomputer, the most powerful in the world as of March 2018, has $2^{23.3}$ CPU cores [69]. In 2013, it was estimated that Google's data centres have a total storage capacity of about a dozen exabytes[1] [69]. Thus it is reasonable to argue that acquiring 2^{64} processors and a storage capacity (with low access times) of several dozen yottabytes[2] for the purpose of solving a CSSI problem will be prohibitively costly for the foreseeable future.

4.5.1 Meet-in-the-middle

As stated in §4.3, 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 $(j(\mathcal{E}_1/A_1), A_1)$, where A_1 ranges over a w-subset of order-$\ell^{e/2}$ subgroups of $\mathcal{E}_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $\mathcal{E}_2[\ell^e]$, $\mathcal{E}_2/A_2$ is computed and $j(\mathcal{E}_2/A_2)$ 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 $\mathcal{E}_1[\ell^e]$, and so on. The running time of this time-memory tradeoff is approximately

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

[1] An exabyte is 2^{60} bytes.
[2] A yottabyte is 2^{80} bytes.

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. §4.3.2). One can see that this time-memory-tradeoff can be parallelized naturally.

Another possible time-memory tradeoff is to store $(j(\mathcal{E}_1/A_1), A_1)$, where A_1 ranges over all order-ℓ^c subgroups of $\mathcal{E}_1[\ell^e]$ and $c \approx \log_\ell w$. Let $d = e - c$. Then, for each order-ℓ^d subgroup A_2 of $\mathcal{E}_2[\ell^e]$, $\mathcal{E}_2/A_2$ is computed and $j(\mathcal{E}_2/A_2)$ 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 naturally. 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.

4.5.2 Golden collision search

As stated in §4.4.3, the running time of van Oorschot-Wiener golden collision search is approximately

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

The algorithm parallelizes naturally.

4.5.3 Mesh sorting

The mesh sorting attack is analogous to the one described by Bernstein [42] 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 $(j(\mathcal{E}_1/A_1), A_1)$, where A_1 is an order-$\ell^{e/2}$ subgroup of $\mathcal{E}_1[\ell^e]$. Next, these stored pairs are sorted in time $\approx m^{1/2}$ (e.g., see [62]). In the next stage, a second two-dimensional grid of m processors computes and stores pairs $(j(\mathcal{E}_2/A_2, A_2)$, where A_2 is an order-$\ell^{e/2}$ subgroup of $\mathcal{E}_2[\ell^e]$, 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 $\mathcal{E}_2[\ell^e]$ have been tested. Then, the process is repeated for a disjoint subset of order-$\ell^{e/2}$ subgroups A_1 of $\mathcal{E}_1[\ell^e]$ until a match is found. One can check that the calendar running time[1] is approximately

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

4.5.4 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.

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

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 4.5 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.

			$p \approx 2^{512}$		$p \approx 2^{448}$	
	# processors m	space w	calendar time	total time	calendar time	total 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 4.5: 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.

Since the total times for $p \approx 2^{512}$ in Table 4.5 are all significantly greater than 2^{128}, one can consider using smaller primes p while still achieving the 128-bit security level. Table 4.5 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 [61].

Remark 4.5.2. (*communication costs*) Consider the case $p \approx 2^{448}$, $e = 224$, $m = 2^{64}$, $w = 2^{80}$. From (4.1) and (4.2) 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 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.

4.5.5 Targetting the 160-bit security level

Using similar arguments as in §4.5.4, one surmises that SIDH parameters with $p \approx 2^{536}$ offer at least 160 bits of CSSI security against known classical (see Table 4.6). For example, one could select the 546-bit prime

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

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

			$p \approx 2^{536}$		$p \approx 2^{614}$	
	# processors m	space w	calendar time	total time	calendar time	total time
Meet-in-the-middle (DFS)	48	64	150	198	188	236
time-memory tradeoff	48	80	134	182	172	220
	64	80	118	182	156	220
Van Oorschot-Wiener	48	64	121	169	149	197
golden collision search	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 4.6: 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.

4.5.6 Targetting the 192-bit security level

Using similar arguments as in §4.5.4, one surmises that SIDH parameters with $p \approx 2^{614}$ offer at least 192 bits of CSSI security against known classical (see Table 4.6). For example, one could select the 610-bit prime

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

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

4.5.7 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 [66]. 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 $\mathcal{E}_i$, g_i to be the function that maps a degree-$\ell^{e/2}$ isogeny originating at $\mathcal{E}_i$ 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 [52]. 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 $\mathcal{E}_1$, $\mathcal{E}_2$, 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 [54] 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 [48] for the 128-bit security level, and this prime has been used in many subsequent works, e.g., [57, 46, 47, 53, 72]. Also, the 964-bit prime $p = 2^{486}3^{301} - 1$ was proposed in [53] 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 [71]. Some recent work [41, 56] 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 [58] (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 [58] (classical and quantum security comparable to that of SHA-384).

4.5.8 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 §4.1 (see Remark 4.1.1) is faster.

Table 4.7 lists timings for SIDH operations for p_{434}, p_{546} and p_{751} using the SIDH library of Costello et al. [45]. The timings show that SIDH operations are about 4.8 times faster when p_{434} is used instead of p_{751}.

Protocol phase		CLN library [48]			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 4.7: 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 [46] and Montgomery ladders from [50] into the CLN library.

Chapter 5

Stronger and Faster Side-Channel Protections for CSIDH

Programming is one of the most difficult branches of applied mathematics; the poorer mathematicians had better remain pure mathematicians.

Edsger Dijkstra

CSIDH is a recent quantum-resistant primitive based on the difficulty of finding isogeny paths between supersingular curves. Recently, two constant-time versions of CSIDH have been proposed: first by Meyer, Campos and Reith, and then by Onuki, Aikawa, Yamazaki and Takagi. While both offer protection against timing attacks and simple power consumption analysis, they are vulnerable to more powerful attacks such as fault injections. In this chapter, we identify and repair two oversights in these algorithms that compromised their constant-time character. By exploiting Edwards arithmetic and optimal addition chains, we produce the fastest constant-time version of CSIDH to date. We then consider the stronger attack scenario of fault injection, which is relevant for the security of CSIDH static keys in embedded hardware. We propose and evaluate a dummy-free CSIDH algorithm. While these CSIDH variants are slower, their performance is still within a small constant factor of less- protected variants. Finally, we discuss derandomized CSIDH algorithms.

5.1 CSIDH protocol

CSIDH is an isogeny based primitive, similar to Diffie–Hellman, that can be used for key exchange and encapsulation [73], signatures [74, 75, 76], and other more advanced protocols. Compared to the other main isogeny-based primitive

SIDH [54, 49], CSIDH is slower. On the positive side, CSIDH has smaller public keys, is based on a better understood security assumption, and supports an easy key validation procedure, making it better suited than SIDH for CCA-secure encryption, static-dynamic and static-static key exchange. In this work we will use the jargon of key exchange when we refer to cryptographic concepts.

CSIDH works over a finite field $\mathbb{F}_\mathsf{P}$, where p is a prime of the special form

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

with $\ell_1, \ldots, \ell_n$ a set of small odd primes. Concretely, the original CSIDH article [73] defined a 511-bit p with $\ell_1, \ldots, \ell_{n-1}$ the first 73 odd primes, and $\ell_n = 587$.

The set of public keys in CSIDH is a subset of all supersingular elliptic curves defined over $\mathbb{F}_\mathsf{P}$, in *Montgomery form* $y^2 = x^3 + Ax^2 + x$, where $A \in \mathbb{F}_\mathsf{P}$ is called the *A-coefficient* of the curve.[1] The endomorphism rings of these curves are isomorphic to orders in the imaginary quadratic field $\mathbb{Q}(\sqrt{-4p})$. Castryck *et al.* [73] choose to restrict the public keys to the *horizontal isogeny class* of the curve with $A = 0$, so that all endomorphism rings are isomorphic to $\mathbb{Z}[\sqrt{-p}]$.

5.1.1 The class group action

Let $\mathcal{E}/\mathbb{F}_\mathsf{P}$ be an elliptic curve with $\mathrm{End}(\mathcal{E}) \cong \mathbb{Z}[\sqrt{-p}]$. If $\mathfrak{a}$ is a nonzero ideal in $\mathbb{Z}[\sqrt{-p}]$, then it defines a finite subgroup $\mathcal{E}[\mathfrak{a}] = \bigcap_{\alpha\in\mathfrak{a}} \ker(\alpha)$, where we identify each α with its image in $\mathrm{End}(\mathcal{E})$. We then have a quotient isogeny $\phi : \mathcal{E} \to \mathcal{E}' = \mathcal{E}/\mathcal{E}[\mathfrak{a}]$ 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 $\mathcal{E}/\mathcal{E}[\mathfrak{a}] \cong \mathcal{E}$. 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 $\mathcal{E}$ over $\mathbb{F}_\mathsf{P}$ with $\mathrm{End}(\mathcal{E}) \cong \mathbb{Z}[\sqrt{-p}]$; this action is faithful and transitive. We write $\mathfrak{a} * \mathcal{E}$ for the image of (the class of) $\mathcal{E}$ under the action of $\mathfrak{a}$, which is (the class of) $\mathcal{E}/\mathcal{E}[\mathfrak{a}]$ 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

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

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

5.1.2 The CSIDH algorithm

At the heart of CSIDH is an algorithm that evaluates the class group action described above on any supersingular curve over $\mathbb{F}_\mathsf{P}$. Cryptographically, this plays the same role as modular exponentiation in classic Diffie–Hellman.

[1] Following [48], we represent $A = A'/C'$ as a projective point $(A' : C')$; see §5.3.1.1.

The input to the algorithm is an elliptic curve $\mathcal{E} : y^2 = x^3 + Ax^2 + x$, represented by its A-coefficient, and an ideal class $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$, represented by its list of exponents $(e_i, \ldots, e_n) \in \mathbb{Z}^n$. The output is the ($A$-coefficient of the) elliptic curve $\mathfrak{a} * \mathcal{E} = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * \mathcal{E}$.

The isogenies corresponding to $\mathfrak{l}_i = (\ell_i, \pi - 1)$ can be efficiently computed using Vélu's formulas and their generalizations: exploiting the fact that $\#\mathcal{E}(\mathbb{F}_\mathsf{P}) = p + 1 = 4\prod \ell_i$, one looks for a point R of order ℓ_i in $\mathcal{E}(\mathbb{F}_\mathsf{P})$ (i.e., a point that is in the kernels of both the multiplication-by-ℓ_i map and $\pi - 1$), computes the isogeny $\phi : \mathcal{E} \to \mathcal{E}/\langle \mathsf{R} \rangle$ with kernel $\langle \mathsf{R} \rangle$, and sets $\mathfrak{l}_i * \mathcal{E} = \mathcal{E}/\langle \mathsf{R} \rangle$. Iterating this procedure lets us compute $\mathfrak{l}_i^e * \mathcal{E}$ for any exponent $e \geq 0$.

The isogenies corresponding to $\mathfrak{l}_i^{-1}$ are computed in a similar fashion: this time one looks for a point R of order ℓ_i in the kernel of $\pi + 1$, i.e., a point in $\mathcal{E}(\mathbb{F}_{p^2})$ of the form (x, iy) where both x and y are in $\mathbb{F}_\mathsf{P}$ (since $i = \sqrt{-1}$ is in $\mathbb{F}_{p^2} \setminus \mathbb{F}_\mathsf{P}$ and satisfies $i^p = -i$). Then one proceeds as before, setting $\mathfrak{l}_i^{-1} * \mathcal{E} = \mathcal{E}/\langle R \rangle$.

In the sequel we assume that we are given an algorithm `QuotientIsogeny` which, given a curve $\mathcal{E}/\mathbb{F}_\mathsf{P}$ and a point R in $\mathcal{E}(\mathbb{F}_{p^2})$, returns the pair $(\phi, \mathcal{E}')$ where $\phi : \mathcal{E} \to \mathcal{E}' \cong \mathcal{E}/\langle \mathsf{R} \rangle$ is an isogeny with kernel $\langle \mathsf{R} \rangle$. We refer to this operation as *isogeny computation*. Algorithm 5.1, taken from the original CSIDH article [73], 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 [73] 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. With this choice, the implementation of [73] computes one class group action in 40 ms on average. Meyer and Reith [77] further improved this to 36 ms on average. Neither implementation is constant-time.

5.1.3 The Meyer–Campos–Reith constant-time algorithm

As Meyer, Campos and Reith observe in [78], Algorithm 5.1 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 5.1 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 [78] a constant-time CSIDH algorithm whose running time does not depend on the private key (though, unlike [79], it still varies due to randomness). The essential differences between the algorithm of [78] 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.

Algorithm 5.1: The original CSIDH class group action algorithm for supersingular curves over $\mathbb{F}_\mathsf{P}$ 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 $\mathfrak{l}_i$.

Input: $A \in \mathbb{F}_\mathsf{P}$ such that $\mathcal{E}_A \colon y^2 = x^3 + Ax^2 + x$ is supersingular, and an integer exponent vector $(e_1, \ldots, e_n)$
Output: B such that $\mathcal{E}_B : y^2 = x^3 + Bx^2 + x$ is $\mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * \mathcal{E}_A$,

$B \leftarrow A$;
while *some* $e_i \neq 0$ **do**
  Sample a random $x \in \mathbb{F}_\mathsf{P}$;
  $s \leftarrow +1$ if $x^3 + Bx^2 + x$ is square in $\mathbb{F}_\mathsf{P}$, else $s \leftarrow -1$;
  $S \leftarrow \{i \mid e_i \neq 0, \mathrm{sign}(e_i) = s\}$;
  if $S \neq \emptyset$ **then**
    $k \leftarrow \prod_{i \in S} \ell_i$;
    $\mathsf{Q} \leftarrow [(p+1)/k]\mathsf{P}$, where P is the projective point with x-coordinate x. ;
    for $i \in S$ **do**
      $\mathsf{R} \leftarrow [k/\ell_i]\mathsf{Q}$; `// Point to be used as kernel generator`
      if $\mathsf{R} \neq \infty$ **then**
        $(\mathcal{E}_B, \phi) \leftarrow \mathtt{QuotientIsogeny}(\mathcal{E}_B, \mathsf{R})$;
        $\mathsf{Q} \leftarrow \phi(\mathsf{Q})$;
        $(k, e_i) \leftarrow (k/\ell_i, e_i - s)$
return B

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 [77]. These include the Elligator 2 map of [80] and [81], multiple batches for isogeny computation (SIMBA), and sample the exponents e_i from intervals of different sizes depending on ℓ_i.

5.1.4 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 [82].

The key idea is to use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (i.e., in $\mathcal{E}(\mathbb{F}_\mathsf{P})$) and one in $\ker(\pi + 1)$ (i.e., in $\mathcal{E}(\mathbb{F}_{p^2})$ with x-coordinate in $\mathbb{F}_\mathsf{P}$). 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 [78]. With these improvements, they achieve a speed-up of 27.35% compared to [78].

We include pseudo-code for the algorithm of [82] in Algorithm 5.2, to serve both as a reference for a discussion of some subtle leaks in §5.2 and also as a departure point for our dummy-free algorithm in §5.4.

Algorithm 5.2: The Onuki–Aikawa–Yamazaki–Takagi CSIDH algorithm for supersingular curves over $\mathbb{F}_\mathsf{P}$, 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 $(m_1, \ldots, m_n)$, are system parameters. This algorithm computes exactly m_i isogenies for each ℓ_i.

Input: A supersingular curve $\mathcal{E}_A\colon y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_\mathsf{P}$, and an integer exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [-m_i, m_i]$.

Output: $\mathcal{E}_B\colon y^2 = x^3 + Bx^2 + x$ such that $\mathcal{E}_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * \mathcal{E}_A$.

$(e'_1, \ldots, e'_n) \leftarrow (m_i - |e_1|, \ldots, m_i - |e_n|)$; `// Number of dummy computations`

$\mathcal{E}_B \leftarrow \mathcal{E}_A$;

while *some* $e_i \neq 0$ *or* $e'_i \neq 0$ **do**

  $S \leftarrow \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$;

  $k \leftarrow \prod_{i \in S} \ell_i$;

  $(\mathsf{T}_+, \mathsf{T}_-) \leftarrow \texttt{Elligator}(\mathcal{E}_B, u)$; `//` $\mathsf{T}_- \in \mathcal{E}_B[\pi + 1]$ `and` $\mathsf{T}_+ \in \mathcal{E}_B[\pi - 1]$

  $(\mathsf{P}_0, \mathsf{P}_1) \leftarrow \big([(p+1)/k]\mathsf{T}_+, [(p+1)/k]\mathsf{T}_-\big)$;

  for $i \in S$ **do**

    $s \leftarrow \operatorname{sign}(e_i)$; `// Ideal` $\mathfrak{l}_i^s$ `to be used`

    $\mathsf{Q} \leftarrow [k/\ell_i]\mathsf{P}_{\frac{1-s}{2}}$; `// Secret kernel point generator`

    $\mathsf{P}_{\frac{1+s}{2}} \leftarrow [\ell_i]\mathsf{P}_{\frac{1+s}{2}}$; `// Secret point to be multiplied`

    if $\mathsf{Q} \neq \infty$ **then**

      if $e_i \neq 0$ **then**

        $(\mathcal{E}_B, \varphi) \leftarrow \texttt{QuotientIsogeny}(\mathcal{E}_B, \mathsf{Q})$;

        $(\mathsf{P}_0, \mathsf{P}_1) \leftarrow \big(\varphi(\mathsf{P}_0), \varphi(\mathsf{P}_1)\big)$;

        $e_i \leftarrow e_i - s$.

      else

        $\mathcal{E}_B \leftarrow \mathcal{E}_B$; $\mathsf{P}_{\frac{1-s}{2}} \leftarrow [\ell_i]\mathsf{P}_{\frac{1-s}{2}}$; $e'_i \leftarrow e'_i - 1$; `// Dummies`

    $k \leftarrow k/\ell_i$

return B

5.2 Repairing constant-time versions

5.2.1 Projective Elligator

Both [78] and [82] use the Elligator 2 map to sample a random point on the current curve $\mathcal{E}_A$ in step 6 of Algorithm 5.2. 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 $\mathcal{E}_A[\pi - 1]$ and $\mathcal{E}_A[\pi + 1]$ respectively.

To avoid a costly inversion of $u^2 - 1$, instead of sampling u randomly, Meyer, Campos and Reith[1] follow [81] 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 [81], where *all* precomputed values of u are tried *for each isogeny computation*, and the algorithm succeeds if at least one passes the test. And indeed the implementation of [78] 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 $\mathcal{E}_A$. Plugging V into the homogeneous equation

$$\mathcal{E}_A : Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3$$

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

$$\mathsf{T}_+ = (A : u^2 - 1), \qquad \mathsf{T}_- = (-Au^2 : u^2 - 1)$$

are in $\mathcal{E}_A[\pi - 1]$ and $\mathcal{E}_A[\pi + 1]$ respectively, otherwise their roles are swapped.

We are left with the case $A = 0$. Following [81], Meyer, Campos and Reith precompute once and for all a pair of generators $\mathsf{T}_+, \mathsf{T}_-$ 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,

[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.

[2] The Elligator optimization is described in §5.3 of [78]. The unoptimized constant-time version described in Algorithm 2 therein is not affected by this problem.

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 [81]: with $u \neq 0$, only on E_0, we define the output of Elligator as $\mathsf{T}_+ = (u : 1), \mathsf{T}_- = (-u : 1)$ 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 [81], 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 5.3, generalizing it to the case of Montgomery curves represented by projective coefficients (see also Section 5.3.1.1).

Algorithm 5.3: Constant-time projective Elligator

Input: A supersingular curve $\mathcal{E}_{(A':C')} : C'y^2 = C'x^3 + A'x^2 + C'x$ over $\mathbb{F}_\mathsf{P}$, and an element $u \in \{2, \ldots, \frac{p-1}{2}\}$.

Output: A pair of points $\mathsf{T}_+ \in \mathcal{E}_{(A':C')}[\pi - 1]$ and $\mathsf{T}_- \in \mathcal{E}_{(A':C')}[\pi + 1]$.

1 $t \leftarrow A'\big((u^2-1)u^2{A'}^2C' + ((u^2-1)C')^3\big)$;
2 $a \leftarrow \texttt{isequal}(t, 0)$; // $t = 0$ iff $A' = 0$
3 $\alpha, \beta \leftarrow 0, u$;
4 $\texttt{cswap}(\alpha, \beta, a)$; // $\alpha = 0$ iff $A' \neq 0$
5 $t' \leftarrow t + \alpha(u^2 + 1)$; // $t' \neq 0$
6 $\mathsf{T}_+ \leftarrow (A' + \alpha C'(u^2 - 1) : C'(u^2 - 1))$;
7 $\mathsf{T}_- \leftarrow (-A'u^2 - \alpha C'(u^2 - 1) : C'(u^2 - 1))$;
8 $b \leftarrow \texttt{Legendre\_symbol}(t', p)$; // $b = \pm 1$
9 $c \leftarrow \texttt{isequal}(b, -1)$;
10 $\texttt{cswap}(\mathsf{T}_+, \mathsf{T}_-, c)$;
11 **return** $(\mathsf{T}_+, \mathsf{T}_-)$;

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

The algorithm from [82], essentially reproduced in Algorithm 5.2, 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 $\mathsf{Q}_0 \leftarrow [k/\ell_i]\mathsf{P}_0$ and $\mathsf{Q}_1 \leftarrow [k/\ell_i]\mathsf{P}_1$ at Line 10, and replacing the condition in Line 12 with a test for $(\mathsf{Q}_0 = \infty)$ **or** $(\mathsf{Q}_1 = \infty)$ (and using constant-time conditional swaps throughout).[1] This fix is visible in

[1] We also found a branch on secret data in the code provided with [78] at https://zenon.cs.hs-rm.de/pqcrypto/faster-csidh, during the 3-isogeny computation, when computing $[\ell]\mathsf{P} = [(\ell-1)/2]\mathsf{P} + [(\ell+1)/2]\mathsf{P}$. This can be easily fixed by a conditional swap, without any significant impact on running time.

Line 13 of Algorithm 5.5 (see page 64).

5.3 Optimizing constant-time implementations

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

5.3.1 Isogeny and point arithmetic on twisted Edwards curves

In this subsection, we present efficient formulas in twisted-Edwards coordinates for four fundamental operations: point addition, point doubling, isogeny computation (as presented in [83]; cf. §5.1.2), and isogeny evaluation (*i.e.* computing the image of a point under an isogeny). Our approach obtains a modest but still noticeable improvement with respect to previous proposals based on Montgomery representation, or hybrid strategies that propound combinations of Montgomery and twisted-Edwards representations [84, 85, 86, 87, 88].

Castryck, Galbraith, and Farashahi [84] proposed using a hybrid representation to reduce the cost of point doubling on certain Montgomery curves, by exploiting the fact that converting between Montgomery and twisted Edwards models can be done at almost no cost. In [88], Meyer, Reith and Campos considered using twisted Edwards formulas for computing isogeny and elliptic curve arithmetic, but concluded that a pure twisted-Edwards-only approach would not be advantageous in the context of SIDH. Bernstein, Lange, Martindale, and Panny observed in [81] that the conversion from Montgomery XZ coordinates to twisted Edwards YZ coordinates occurs naturally during the Montgomery ladder. Kim, Yoon, Kwon, Park, and Hong presented a hybrid model in [86] using Edwards and Montgomery models for isogeny computations and point arithmetic, respectively; in [85] and [87], they suggested computing isogenies using a modified twisted Edwards representation that introduces a fourth coordinate w.

To the best of our knowledge, the quest for more efficient elliptic curve and isogeny arithmetic than that offered by pure Montgomery and twisted-Edwards-Montgomery representations remains an open problem. As a step forward in this direction, Moody and Shumow [83] showed that when dealing with isogenies of odd degree $d = 2\ell - 1$ with $\ell \geq 2$, twisted Edwards representation offers a cheaper formulation for isogeny computation than the corresponding one using Montgomery curves; nevertheless, they did not address the problem of getting a cheaper twisted Edwards formulation for the isogeny evaluation operation.

5.3.1.1 Montgomery curves

A Montgomery curve [89] is defined by the equation $\mathcal{E}_{A,B} : By^2 = x^3 + Ax^2 + x$, such that $B \neq 0$ and $A^2 \neq 4$ (we often write $\mathcal{E}_A$ for $\mathcal{E}_{A,1}$). We refer to [90] for a survey on Montgomery curves. When performing isogeny computations

and evaluations, it is often more convenient to represent the constant A in the projective space $\mathbb{P}^1$ as $(A' : C')$, such that $A = A'/C'$. Montgomery curves are attractive because they are exceptionally well-suited to performing the differential point addition operation which computes $x(\mathsf{P}+\mathsf{Q})$ from $x(\mathsf{P})$, $x(\mathsf{Q})$, and $x(\mathsf{P}-\mathsf{Q})$. Equations (5.1) and (5.2) describe the differential point doubling and addition operations proposed by Montgomery in [89]:

$$\begin{aligned} X_{[2]\mathsf{P}} &= C_{24}(X_\mathsf{P}+Z_\mathsf{P})^2(X_\mathsf{P}-Z_\mathsf{P})^2, \\ Z_{[2]\mathsf{P}} &= ((X_\mathsf{P}+Z_\mathsf{P})^2-(X_\mathsf{P}-Z_\mathsf{P})^2)\cdot \\ &\qquad (C_{24}(X_\mathsf{P}-Z_\mathsf{P})^2 + A_{24p}((X_\mathsf{P}+Z_\mathsf{P})^2-(X_\mathsf{P}-Z_\mathsf{P})^2)) \end{aligned} \tag{5.1}$$

where $A_{24p} = A + 2C$ and $C_{24} = 4C$, and

$$\begin{aligned} X_{\mathsf{P}+\mathsf{Q}} &= Z_{\mathsf{P}-\mathsf{Q}}\left[(X_\mathsf{P}-Z_\mathsf{P})(X_\mathsf{Q}+Z_\mathsf{Q})+(Z_\mathsf{P}+Z_\mathsf{P})(X_\mathsf{Q}-Z_\mathsf{Q})\right]^2, \\ Z_{\mathsf{P}+\mathsf{Q}} &= X_{\mathsf{P}-\mathsf{Q}}\left[(X_\mathsf{P}-Z_\mathsf{P})(X_\mathsf{Q}+Z_\mathsf{Q})-(Z_\mathsf{P}+Z_\mathsf{P})(X_\mathsf{Q}-Z_\mathsf{Q})\right]^2. \end{aligned} \tag{5.2}$$

Montgomery curves can be used to efficiently compute isogenies using Vélu's formulas [91]. Suppose we want the image of a point Q under an ℓ-isogeny ϕ, where $\ell = 2k+1$. For each $1 \le i \le k$ we let $(X_i : Z_i) = x([i]\mathsf{P})$, where $\langle P\rangle = \ker\phi$. Equation (5.3) computes $(X' : Z') = x(\phi(\mathsf{Q}))$ from $(X_\mathsf{Q} : Z_\mathsf{Q}) = x(\mathsf{Q})$.

$$\begin{aligned} X' &= X_\mathsf{P}\Big(\prod_{i=1}^{k}\left[(X_\mathsf{Q}-Z_\mathsf{Q})(X_i+Z_i)+(Z_\mathsf{Q}+Z_\mathsf{Q})(X_i-Z_i)\right]\Big)^2, \\ Z' &= Z_\mathsf{P}\Big(\prod_{i=1}^{k}\left[(X_\mathsf{Q}-Z_\mathsf{Q})(X_i+Z_i)-(Z_\mathsf{Q}+Z_\mathsf{Q})(X_i-Z_i)\right]\Big)^2. \end{aligned} \tag{5.3}$$

5.3.1.2 Twisted Edwards curves

In [92] we see that every Montgomery curve $\mathcal{E}_{A,B} : By^2 = x^3 + Ax^2 + x$ is birationally equivalent to a twisted Edwards curve $\mathcal{E}_{A,d} : ax^2 + y^2 = 1 + dx^2y^2$; the curve constants are related by

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

and the rational maps $\phi : \mathcal{E}_{A,d} \to \mathcal{E}_{A,B}$ and $\psi : \mathcal{E}_{A,B} \to \mathcal{E}_{A,d}$ are defined by

$$\begin{aligned} \phi &: (x,y) \longmapsto ((1+y)/(1-y), (1+y)/(1-yx)), \\ \psi &: (x,y) \longmapsto (x/y, (x-1)/(x+1)). \end{aligned} \tag{5.4}$$

Rewriting this relationship for Montgomery curves with projective constants, $\mathcal{E}_{A,d}$ is equivalent to the Montgomery curve $\mathcal{E}_{(A:C)} = \mathcal{E}_{A/C,1}$ with constants

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

To avoid notational ambiguities, we write $(Y_\mathsf{P} : T_\mathsf{P})$ for the $\mathbb{P}^1$ projection of the y-coordinate of the point $P \in \mathcal{E}_{A,d}$. Let $P \in \mathcal{E}_{(A:C)}$. In projective coordinates, the map ψ of (5.4) becomes

$$\psi : (X_\mathsf{P} : Z_\mathsf{P}) \longmapsto (Y_\mathsf{P} : T_\mathsf{P}) = (X_\mathsf{P} - Z_\mathsf{P} : X_\mathsf{P} + Z_\mathsf{P}). \tag{5.5}$$

Comparing (5.5) with (5.1) reveals that Y_P and T_P appear in the doubling formula, so we can substitute them at no cost. Replacing A_{24p} and C_{24} with their twisted Edwards equivalents a and $e = a - d$, respectively, we obtain a doubling formula for twisted Edwards YT coordinates:

$$\begin{aligned} Y_{[2]\mathsf{P}} &= e \cdot Y_\mathsf{P}^2 \cdot T_\mathsf{P}^2 - (T_\mathsf{P}^2 - Y_\mathsf{P}^2) \cdot (eY_\mathsf{P}^2 + a(T_\mathsf{P}^2 - Y_\mathsf{P}^2)), \\ T_{[2]\mathsf{P}} &= e \cdot Y_\mathsf{P}^2 \cdot T_\mathsf{P}^2 + (T_\mathsf{P}^2 - Y_\mathsf{P}^2) \cdot (eY_\mathsf{P}^2 + a(T_\mathsf{P}^2 - Y_\mathsf{P}^2)). \end{aligned}$$

Similarly, the coordinates $Y_\mathsf{P}, T_\mathsf{P}, Y_\mathsf{Q}, T_\mathsf{Q}, Y_{\mathsf{P}-\mathsf{Q}}$ and $T_{\mathsf{P}-\mathsf{Q}}$ appear in (5.2), and thus we derive differential addition formulas for twisted Edwards coordinates:

$$\begin{aligned} Y_{\mathsf{P}+\mathsf{Q}} &= (T_{\mathsf{P}-\mathsf{Q}} - Y_{\mathsf{P}-\mathsf{Q}})(Y_\mathsf{P}T_\mathsf{Q} + Y_\mathsf{Q}Z_\mathsf{P})^2 - (T_{\mathsf{P}-\mathsf{Q}} + Y_{\mathsf{P}-\mathsf{Q}})(Y_\mathsf{P}T_\mathsf{Q} - Y_\mathsf{Q}Z_\mathsf{P})^2, \\ T_{\mathsf{P}+\mathsf{Q}} &= (T_{\mathsf{P}-\mathsf{Q}} - Y_{\mathsf{P}-\mathsf{Q}})(Y_\mathsf{P}T_\mathsf{Q} + Y_\mathsf{Q}Z_\mathsf{P})^2 + (T_{\mathsf{P}-\mathsf{Q}} + Y_{\mathsf{P}-\mathsf{Q}})(Y_\mathsf{P}T_\mathsf{Q} - Y_\mathsf{Q}Z_\mathsf{P})^2. \end{aligned}$$

The computational costs of doubling and differential addition are $4\mathbf{M}+2\mathbf{S}+4\mathbf{A}$ (the same as evaluating (5.1)) and $4\mathbf{M}+2\mathbf{S}+6\mathbf{A}$ (the same as (5.2)), respectively.

The Moody–Shumow formulas for isogeny computation [83] are given in terms of twisted Edwards YT-coordinates. It remains to derive a twisted Edwards YT-coordinate isogeny-evaluation formula for ℓ-isogenies where $\ell = 2k + 1$. We do this by applying the map in (5.5) to (5.3), which yields

$$\begin{aligned} Y' = (T_{\mathsf{P}-\mathsf{Q}} + Y_{\mathsf{P}-\mathsf{Q}}) \cdot \Big(\prod_{i=1}^{k} \big[T_\mathsf{Q} Y_{[i]\mathsf{P}} + Y_\mathsf{Q} T_{[i]\mathsf{P}}\big] \Big)^2 \\ - (T_{\mathsf{P}-\mathsf{Q}} - Y_{\mathsf{P}-\mathsf{Q}}) \cdot \Big(\prod_{i=1}^{k} \big[T_\mathsf{Q} Y_{[i]\mathsf{P}} - Y_\mathsf{Q} T_{[i]\mathsf{P}}\big] \Big)^2, \\ T' = (T_{\mathsf{P}-\mathsf{Q}} + Y_{\mathsf{P}-\mathsf{Q}}) \cdot \Big(\prod_{i=1}^{k} \big[T_\mathsf{Q} Y_{[i]\mathsf{P}} + Y_\mathsf{Q} T_{[i]\mathsf{P}}\big] \Big)^2 \\ + (T_{\mathsf{P}-\mathsf{Q}} - Y_{\mathsf{P}-\mathsf{Q}}) \cdot \Big(\prod_{i=1}^{k} \big[T_\mathsf{Q} Y_{[i]\mathsf{P}} - Y_\mathsf{Q} T_{[i]\mathsf{P}}\big] \Big)^2. \end{aligned}$$

The main advantage of the approach outlined here is that by only using points given in YT coordinates, we can compute point doubling, point addition and isogeny construction and evaluation at a lower computational cost. Indeed, isogeny evaluation in XZ costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas the above YT coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k + 4)\mathbf{A}$, thus saving $4k - 4$ field additions.

5.3.2 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 use shorter differential addition chains.[1]

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 $[k]\mathsf{P}$ 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 $[k]\mathsf{P}$ 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.

5.4 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 18 of Algorithm 5.2 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 ([93], [94]), 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 [82], then we must also compute $k - |e_i|$ dummy ℓ_i-isogenies to ensure a constant-time behaviour.

[1] 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.

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

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

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 $(e_1, \ldots, e_n)$ drawn from $\mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$. Algorithm 5.4 achieves this in constant-time and without using dummy operations. The outer loop at line 3 is repeated exactly $\max(m_i)$ times, but the inner "if" block at line 5 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 $\mathfrak{l}_i^{\mathtt{sign}(e_i)}$ 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 $\mathtt{sign} : \mathbb{Z} \to \{\pm 1\}$ operation is implemented in constant time, and that $\mathtt{sign}(0) = 1$. 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 5.4: An idealized dummy-free constant-time evaluation of the CSIDH group action.

Input: Secret vector $(e_1, \ldots, e_n) \in \mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$

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

However, Algorithm 5.4 is only an idealized version of the CSIDH group action algorithm. Indeed, like in [78, 82], 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[1], and thus the fact that an iteration is skipped does not leak information on the secret. The resulting algorithm is summarized in Algorithm 5.5.

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 CSIDH protocol implementation [73] 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.

5.5 Derandomized CSIDH algorithms

As we stressed in Section 5.2, 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

[1] Assuming the usual heuristic assumptions on the distribution of the output of Elligator, see [78].

Algorithm 5.5: Dummy-free randomized constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_{\mathsf{P}}$, 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 $\mathfrak{l}_i$.

Input: A supersingular curve $\mathcal{E}_A$ over $\mathbb{F}_{\mathsf{P}}$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [-m_i, m_i]$ and $e_i \equiv m_i \pmod 2$.

Output: $\mathcal{E}_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * \mathcal{E}_A$.

$(t_1, \ldots, t_n) \leftarrow \left(\frac{\mathtt{sign}(e_1)+1}{2}, \ldots, \frac{\mathtt{sign}(e_n)+1}{2}\right)$; `// Secret`

$(z_1, \ldots, z_n) \leftarrow (m_1, \ldots, m_n)$; `// Not secret`

$\mathcal{E}_B \leftarrow \mathcal{E}_A$;

while *some* $z_i \neq 0$ **do**

  $u \leftarrow \mathtt{Random}\left(\left\{2, \ldots, \frac{p-1}{2}\right\}\right)$;

  $(\mathsf{T}_1, \mathsf{T}_0) \leftarrow \mathtt{Elligator}(\mathcal{E}_B, u)$; `//` $\mathsf{T}_1 \in \mathcal{E}_B[\pi - 1]$ `and` $\mathsf{T}_0 \in \mathcal{E}_B[\pi + 1]$

  $(\mathsf{T}_0, \mathsf{T}_1) \leftarrow ([4]\mathsf{T}_0, [4]\mathsf{T}_1)$; `//` `Now` $\mathsf{T}_0, \mathsf{T}_1 \in \mathcal{E}_B\left[\prod_i \ell_i\right]$

  for $i \in \{1, \ldots, n\}$ **do**

    if $z_i \neq 0$ **then**

      $(\mathsf{G}_0, \mathsf{G}_1) \leftarrow (\mathsf{T}_0, \mathsf{T}_1)$;

      for $j \in \{i + 1, \ldots, n\}$ **do**

        $(\mathsf{G}_0, \mathsf{G}_1) \leftarrow ([\ell_j]\mathsf{G}_0, [\ell_j]\mathsf{G}_1)$

      if $\mathsf{G}_0 \neq \infty$ *and* $\mathsf{G}_1 \neq \infty$ **then**

        $\mathtt{cswap}(\mathsf{G}_0, \mathsf{G}_1, t_i)$; `// Secret kernel point generator:` G_0

        $\mathtt{cswap}(\mathsf{T}_0, \mathsf{T}_1, t_i)$; `// Secret point to be multiplied:` T_1

        $(\mathcal{E}_B, \phi) \leftarrow \mathtt{QuotientIsogeny}(\mathcal{E}_B, \mathsf{G}_0)$;

        $(\mathsf{T}_0, \mathsf{T}_1) \leftarrow \big(\phi(\mathsf{T}_0), \phi(\mathsf{T}_1)\big)$;

        $\mathsf{T}_1 \leftarrow [\ell_i]\mathsf{T}_1$;

        $\mathtt{cswap}(\mathsf{T}_0, \mathsf{T}_1, t_i)$;

        $b \leftarrow \mathtt{isequal}(e_i, 0)$;

        $e_i \leftarrow e_i + (-1)^{t_i}$;

        $t_i \leftarrow t_i \oplus b$;

        $z_i \leftarrow z_i - 1$

      else if $\mathsf{G}_0 \neq \infty$ **then**

        $\mathsf{T}_0 \leftarrow [\ell_i]\mathsf{T}_0$;

      else if $\mathsf{G}_1 \neq \infty$ **then**

        $\mathsf{T}_1 \leftarrow [\ell_i]\mathsf{T}_1$;

return B

be computed anyway for key validation purposes, and thus this computation only slows down key validation by a factor of two.

Since we have restricted exponents to $\{-1, 0, 1\}$, every ℓ_i-isogeny in Algorithm 5.2 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 5.5. 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 [95, 96, 73]). 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 [74], 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.

5.6 Experimental results

Tables 5.1 and 5.2 summarize our experimental results, and compare our algorithms with those of [73], [78], and [82]. Table 5.1 compares algorithms in terms of elementary field operations, while Table 5.2 compares cycle counts of C implementations. All of our experiments were ran on a Intel(R) Core(TM) i7-6700 CPU Skylake 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 [73]), the group action is evaluated using the SIMBA method (Splitting Isogeny computations into Multiple BAtches) proposed by Meyer, Campos, and Reith in [78]. 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 [73]. None of our countermeasures or algorithmic improvements were applied.

Meyer–Campos–Reith. We used the software library freely available from the authors of [78]. This software batches isogenies using SIMBA-5-11. The improvements we describe in §5.2 and §5.3 were *not* applied.

Onuki *et. al.* Unfortunately, the source code for the implementation in [82] was not freely available, so direct comparison with our implementation was impossible. Table 5.1 includes their field operation counts for their unmodified algorithm (which, as noted in §5.2, is insecure) using SIMBA-3-8, and our estimates for a repaired version applying our fix in §5.2. We did not apply the optimizations of §5.3 here. (We do not replicate the cycle counts from [82] in Table 5.2, 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 [82, §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 §5.2 and §5.3.

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 §5.2 and §5.3.

No-dummy This is Algorithm 5.5 (with two torsion points and no dummy operations), batching isogenies using SIMBA-5-11.

In each case, the improvements and optimizations of §5.2-5.3 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 [78]; since the underlying arithmetic is essentially identical, the performance comparisons below reflect differences in the CSIDH algorithms.

Results. It is shown in Table 5.2 that the techniques we introduced in §5.2 and §5.3 produce substantial savings compared with the implementation of [78]. In particular, our OAYT-style implementation yields a 39% improvement over [78]. Since the implementations use the same underlying field arithmetic library, these improvements are entirely due to the techniques introduced in this work. 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 22% slower than [78].

Implementation	CSIDH Algorithm	M	S	A	Ratio
Castryck et al. [73]	unprotected, unmodified	0.252	0.130	0.348	0.26
Meyer–Campos–Reith [78]	unmodified	1.054	0.410	1.053	1.00
Onuki et al. [82]	unmodified	0.733	0.244	0.681	0.67
This work	MCR-style	0.901	0.309	0.965	0.83
	OAYT-style	0.657	0.210	0.691	0.59
	No-dummy	1.319	0.423	1.389	1.19

Table 5.1: 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 [82] are estimates, and hence displayed in italics. The performance ratio uses [78] as a baseline, considers only multiplication and squaring operations, and assumes $M = S$.

Implementation	CSIDH algorithm	Mcycles	Ratio
Castryck et al. [73]	unprotected, unmodified	155	0.39
Meyer–Campos–Reith [78]	unmodified	395	1.00
This work	MCR-style	337	0.85
	OAYT-style	239	0.61
	No-dummy	481	1.22

Table 5.2: Clock cycle counts for constant-time CSIDH implementations, averaged over 1024 experiments. The ratio is computed using [78] as baseline implementation.

Chapter 6

Concluding remarks

Mathematics is a game played according to certain simple rules with meaningless marks on paper.

David Hilbert

We have first shown that extending the GLS endomorphism allows to speedup the GHS Weil descent attack. In particular, we have proven that the GLS endomorphism on $\mathcal{E}/\mathbb{F}_{2^{n\cdot\ell}}$ induces an efficient endomorphism $\Psi^*\colon \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q) \to \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ on the jacobian of the image of the GHS Weil descent applied on $\mathcal{E}/\mathbb{F}_{2^{n\cdot\ell}}$. We show that the above endomorphism yields a factor n speedup when using standard index-calculus procedures for solving the Discrete Logarithm Problem (DLP) on $\mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Our analysis is backed up by the explicit computation of a DLP defined on a prime order subgroup of a GLS elliptic curve over the field $\mathbb{F}_{2^{5\cdot 31}}$. A Magma implementation of a standard index-calculus procedure boosted with the GLS endomorphism is able to find this discrete logarithm in about $1,035$ CPU days.

Additionally, we have presented an efficient root-finding algorithm for linearized polynomials. Such polynomials are of interest for determining the genus of the image of the gGHS Weil descent technique. Our experiments show that our method succeeds finding the roots of $h^{\sigma}(x)$ for relatively large instances of this problem, instances where the generic state-of-the-art root-finding algorithms tend to fail because of memory exhaustion. Further, we took advantage of our algorithm to show that Mersene's primes $\ell = 2^t - 1$, allows to discover a family of elliptic curves $\mathcal{E}/\mathbb{F}_{q^\ell}$ for which the gGHS Weil descent successfully and deterministically constructs a (non-)hyperelliptic curve $\mathcal{H}/\mathbb{F}_q$ of genus $g \approx \ell$.

We have then analyzed the cost of computing isogenies elliptic curves (the computationally hard problem for which the security of isogeny-based cryptography is based). Our implementations of the Meet-in-The-Middle (MITM) and van Oorschot & Wiener (VW) golden collision search CSSI attacks are, to the best of our knowledge, the first ones reported in the literature. The implemen-

tations confirm that the performance of these attacks is accurately predicted by their heuristic analysis.

Our concrete cost analysis of the attacks leads to the conclusion that VW golden collision search is more effective that the MITM 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. As a consequence, our security analysis has strongly impacted on the post-quantum cryptography community, to the point of being endorsed by the SIKE protocol designers and pushing them to use our proposed smaller prime integer numbers.

Finally, we have studied side-channel protected implementations of the isogeny based primitive CSIDH. Previous implementations failed at being fully constant time on the inputs 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 have 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 conservative", with much stronger security guarantees, however at the moment this version seems too costly for the security benefits; further work is required to make it comparable with the original definition of CSIDH in terms of efficiency.

6.1 List of implemented codes

1. A parallel version of the Enge-Gaudry algorithm was implemented using the computer algebra system Magma [30]. Our Magma-codes implementation of our procedures is available at

   https://github.com/JJChiDguez/combining_GLS_with_GHS.

2. A Magma-code implementation of an efficient root-finding algorithm for linearized polynomials (see Algorithm 3.1). Our code is available at

   https://github.com/JJChiDguez/root-finding.

3. The MITM-basic and MITM-DFS attacks (for $\ell = 2$) were implemented in C. This C-code implementation requires the library fopenmp for the parallelization. Our code for the MITM-basic, MITM-DFS and VW golden collision search CSSI attacks is available at

   https://github.com/JJChiDguez/CSSI.

4. A C-code implementation of the following three different constant-time CSIDH algorithms was performed:

   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 §5.2 and §5.3.

   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 §5.2 and §5.3.

   No-dummy This is Algorithm 5.5 (with two torsion points and no dummy operations), batching isogenies using SIMBA-5-11.

   In each case, the improvements and optimizations of §5.2-5.3 were 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 .

6.2 List of publications

As a result of this thesis research, we obtain three international conferences articles accepted and published:

[31] **Jesús-Javier Chi** and Thomaz Oliveira, "Attacking a Binary GLS Elliptic Curve with Magma", *Progress in Cryptology - LATINCRYPT 2015*, LNCS 9230 (2015), 308–326.

[99] Gora Adj, Daniel Cervantes-Vázquez, **Jesús-Javier Chi-Domínguez**, Alfred Menezes, Francisco Rodríguez-Henríquez, "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves", *Selected Areas in Cryptography — SAC 2019*. LNCS 11349 (2018), 322–343.

[100] 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", *Progress in Cryptology - LATINCRYPT 2019*. LNCS 11774 (2019), 173-193

Additionally, the following work has been submitted to the indexed journal *Finite Fields and Their Applications*, which is still under revision.

- **Jesús-Javier Chi-Domínguez**, Francisco Rodríguez-Henríquez, and Benjamin Smith, "Extending the GLS endomorphism to speedup the GHS Weil descent using Magma".

6.3 Forthcoming research

First of all, the main result of chapter 3 implies that the GLS endomorphism of a GLS curve $\mathcal{E}/\mathbb{F}_{2^{2n}}$ induces an efficient endomorphism $\Psi^*\colon \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{2^n}) \to \mathsf{Jac}_{\mathcal{H}}(\mathbb{F}_{2^n})$ on the jacobian of the image of the GHS Weil descent applied on $\mathcal{E}/\mathbb{F}_{2^{2n}}$; that is, there is an efficient endomorphism $\Psi\colon \mathcal{H}(\mathbb{F}_{2^n}) \to \mathcal{H}(\mathbb{F}_{2^n})$ on the genus two hyperelliptic curve $\mathcal{H}/\mathbb{F}_{2^n}$. This observation is useful because it allows to combine a GLV method [25] with the induced endomorphism Ψ^* in order to perform an efficient scalar multiplication on $\mathcal{H}/\mathbb{F}_{2^n}$, and clearly this observation requires to be analyzed for constructive aspects of genus-2 curve based cryptography.

Subsequently, Martins-Banegas-Custódio show in [98] analyzed a code-based scheme which was submitted to the NIST competition. In particular, they present four root-finding algorithms that are protected against remote timing exploitation. However, we think that our efficient root-finding algorithm for linearized polynomials can be used to accelerate their computations.

Next, the VW golden collision search CSSI attack is based on a parallel paradigm of programming, and GPU's are designed for being used to solve hard problems which allow parallelization. Thus, it is of interest to know how the VW golden collision search CSSI attack behaves in a cuda-code implementation on GPU's. On the other hand, VW golden collision search method is backed on the fact of using hash tables per each different random function. However, Dequen-Ionica-Trimoska showed that half of the hash table cells are never reached [97] and therefore, they suggest to use a structure called radix-tree to accelerate the timings of the method. Consequently, it is of interest to know the practical implications of using radix-tree in VW golden collision search CSSI attack.

As mentioned in chapter 4, the only two applications of the VW golden collision search method are for attacking 2DES, 3DES, and (as a result of this thesis) SIDH schemes. Thus, it is important to know if CSIDH is falling in this small and particular set of cryptographic schemes. To be more precise, an analysis of the VW golden collision search CSSI attack but applied to CSIDH protocol must be done.

Finally, as mentioned in chapter 5, the quantum security of CSIDH protocol is still not well understanding because of different algorithms, and debatable assumptions about the required resources; anyhow, the best option would be to propose a paranoic constant-time c-code implementation of CSIDH protocol, which requires a finite field with at least 2^{1500} elements. Clearly, this paranoic CSIDH protocol will be very slow (perhaps, with stronger security) and needs a further research study.

Appendix

Magma codes: Discrete Logarithm Problem on $\mathcal{E}/\mathbb{F}_{2^{5\times 31}}$

A.1 Elliptic curve instances

Listing 1: EC_instance.mag

```
clear;

n := 5; l := 31; q := 2^n; N := 2^l;

F_2 := GF(2);
P_2<t> := PolynomialRing(F_2);

F_q<u> := ext<F_2| t^5 + t^2 + 1>;
F_qn<v>:= ext<F_q| t^31 + t^3 + 1>;

a_qn := F_qn!1; b_qn := v^18 + v^17 + v^12 + v^8 + v^5 + v^4 + 1;
E_qn := EllipticCurve([F_qn| 1, a_qn, 0, 0, b_qn]);
c := 0x12E7FB306F6; r := 0x6C530B0FAF0022649878E620CAE2D;

Pt_x := F_qn![ u^10, u^30, u^24, u^17, u^26, u^23, u^22, u^8,
u^4, u^25, u^24, u^19, 0, u^30, u^2, u^8, u^24, u^16, u^21,
u^19, u^3, u^2, u^21, u^7, u^11, u^4, u^23, u^13, u^3, u^23,
u^23 ];
Pt_y := F_qn![ u^25, u^29, u^16, u^20, 0, 1, u^10, u^6, u^13,
u^30, u^8, u^30, u^9, u^9, 0, u^9, u^8, u^28, u^21, u^23, u^23,
u^16, u^27, u^22, u^8, u^4, u^8, u^12, u^17, u^7, u^9 ];
Pt := E_qn![Pt_x, Pt_y];

Pt_prime_x := v^355/v^133 + (v+u+1);
Pt_prime_y := F_qn![ u^15, u^12, u^12, 1, u^15, u^22, u^16, 0,
u^17, u^3, u^19, u^10, u^9, u^25, u^18, u^23, u^13, u^9, u^12,
u^22, u^30, u^17, u^15, u^22, u^2, u^22, u^21, u^16, u^13, u^7,
u^20 ];

Pt_prime := c*E_qn![Pt_prime_x, Pt_prime_y];
```

A.2 Hyperelliptic curve instances

Listing 2: HEC_instance.mag

```
P_q<w> := PolynomialRing(F_q);

h_q := u^7*w^32 + u^12*w^16 + u^30*w^8 + u^28*w^2 + u^7*w;
f_q := u^4*w^65 + u^14*w^64 + u^14*w^33 + u^19*w^17 + u^16*w^8
+ u^15*w^5 + u^25*w^4 + u^4*w^3 + u^24*w;

H_q := HyperellipticCurve(f_q, h_q);
J_q := Jacobian(H_q);

D_x := P_q![ u^9, u^18, u^28, u^3, u^29, u^21, u^17, u^19, u^26,
u^16, u^8, u^25, u^11, u^8, u^5, u^18, 0, u^2, u^21, u^3, u^28,
u^19, u^22, u^14, u^24, u^6, u^28, u^19, u^16, u^21, u^20, u^18,
1 ];
D_y := P_q![ u^4, u^24, 0, u^2, u^20, u^18, u^30, u, u^6, u^6,
u^27, u^29, u^14, u^29, u^17, u^10, u^12, u^23, u^11, u^3, u^12,
u^11, u^9, u^14, u^30, u^25, u^6, 0, u^5, u^2, u^29, u^25 ];
D := J_q![D_x, D_y];

D_prime_x := P_q![ u^19, u^8, u^23, u^7, u^26, 0, u^2, u^4, u^21,
u^12, u^17, u^20, u^22, u^2, u^5, u^17, u, u^27, u^28, u^16, u^6,
u^18, u^5, u^27, u^19, u^15, u^11, u^14, u^8, u^6, u^26, u^11, 1 ];
D_prime_y := P_q![ u^2, u^24, u^21, u^13, u^10, u^17, 1, u^15,
u^29, u^3, u^16, u^4, u, u^17, u^13, u^22, u^26, u^18, u^8, u^16,
u^21, u^26, u, u^16, u^16, u^3, u^5, u^24, u^26, u^26, u^14, u^14];

D_prime := J_q![D_prime_x, D_prime_y];
```

A.3 Testing the solution

Listing 3: checking_dlog.mag

```
load "EC_instance.mag";
load "HEC_instance.mag";

dLog := 0x618877C96DE350E8C7980393356E3;
(Pt * dLog) eq Pt_prime;
(D * dLog) eq D_prime;
```

Bibliography

[1] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, 22 (1976), 644–654.

[2] L. Adleman, R. Rivest and A. Shamir, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM*, ACM 21 (1978), 120–126.

[3] S. Miller, "Use of Elliptic Curves in Cryptography", *Advances in Cryptology - CRYPTO '85*, LNCS 218 (1986), 417–426.

[4] N. Koblitz, "Elliptic curve cryptosystems', *Mathematics of Computation*, AMS 48 (1987), 203–209.

[5] L. Grover, "A Fast Quantum Mechanical Algorithm for Database Search", *Symposium on the Theory of Computing*, ACM (1996), 212–219.

[6] P. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *Journal on Computing*, SIAM 26 (1997), 1484–1509.

[7] S. Galbraith, X. Lin, and M. Scott, "Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves". *Journal of Cryptology*, 24 (2011), 446–469".

[8] D. Hankerson, K. Karabina, and A. Menezes, "Analyzing the Galbraith-Lin-Scott Point Multiplication Method for Elliptic Curves over Binary Fields", *IEEE Transactions on Computers*, 58 (2009), 1411–1420.

[9] A. Ay, C. Mancillas-López, E. Öztürk, Francisco Rodríguez-Henríquez, and Erkay Savas, "Constant-time hardware computation of elliptic curve scalar multiplication around the 128 bit security level", *Microprocessors and Microsystems — Embedded Hardware Design*, 62 (2018): 79–90.

[10] T. Oliveira, D. Aranha, J. López, and F. Rodríguez-Henríquez, "Improving the performance of the GLS254", *Presentation at CHES 2016 rump session*, (2016).

[11] T. Oliveira, J. López, D. Aranha, and F. Rodríguez-Henríquez, "Two is the fastest prime: lambda coordinates for binary elliptic curves", *Journal of Cryptographic Engineering* 4 (2014), 3–17.

[12] T. Oliveira, J. López, D. Aranha, and F. Rodríguez-Henríquez, "Lambda Coordinates for Binary Elliptic Curves", *Cryptographic Hardware and Embedded Systems – CHES 2013*. LNCS 8086 (2013), 311-330.

[13] G. Frey, "How to disguise an elliptic curve". *Talk at ECC'98, Waterloo*. Public version available at https://cr.yp.to/bib/1998/frey-disguise.ps.

[14] S. Galbraith and N. Smart, "A Cryptographic Application of Weil Descent", *Cryptography and Coding — IMA 1999*, LNCS 1746 (1999), 191–200.

[15] A. Menezes and M. Qu, "Analysis of the Weil Descent Attack of Gaudry, Hess and Smart", *Topics in Cryptology — CT-RSA 2001*, LNCS 2020 (2001), 308–318.

[16] M. Maurer, A. Menezes, and E. Teske, "Analysis of the GHS Weil Descent Attack on the ECDLP over Characteristic Two Finite Fields of Composite Degree", *Progress in Cryptology — INDOCRYPT 2001*, LNCS 2247 (2001), 195–213.

[17] J. Couveignes, "Hard Homogeneous Spaces", Cryptology ePrint Archive: Report 2006/291. Available http://eprint.iacr.org/2006/291.

[18] A. Rostovtsev and A. Stolbunov, "Public-Key Cryptosystem Based on Isogenies", Cryptology ePrint Archive: Report 2006/145. Available http://eprint.iacr.org/2006/145.

[19] A. Stolbunov, "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves", *Advances in Mathematics of Communications*, AMC 4 (2010), 215–235.

[20] S. Galbraith, "Mathematics of Public Key Cryptography", Cambridge University Press, New York, NY, USA, 1st edition, 2012. Public version 2.0 available at https://www.math.auckland.ac.nz/~sgal018/crypto-book/main.pdf.

[21] L. Washington, "Elliptic Curves: Number Theory and Cryptography", Second Edition, Chapman & Hall/CRC, 2008.

[22] P. Gaudry, E. Thomé, N. Thériault, and C. Diem, "A double large prime variation for small genus hyperelliptic index calculus", *Mathematics of Computation*. AMS 76 (2007), 475–492.

[23] P. Gaudry, "An algorithm for solving the discrete log problem on hyperelliptic curves", *Advances in Cryptology - EUROCRYPT 2000*, 1807 (2000), 19–34.

[24] A. Enge and P. Gaudry, "A general framework for subexponential discrete logarithm algorithms", *Acta Arithmetica*, 102 (2002), 83–103.

[25] R. Gallant, R. Lambert, and S. Vanstone, "Faster point multiplication on elliptic curves with efficient endomorphisms", *Advances in Cryptology — CRYPTO 2001*, LNCS 2139 (2001), 190–200.

[26] P. Gaudry, F. Hess, and N. Smart, "Constructive and destructive facets of weil descent on elliptic curves", *Journal of Cryptology*, 15 (2002), 19–46.

[27] S. Galbraith, F. Hess, and N. Smart, "Extending the GHS weil descent attack", *Advances in Cryptology - EUROCRYPT 2002*, LNCS 2332 (2002), 29-44.

[28] F. Hess, "The GHS Attack Revisited", *Advances in Cryptology - EUROCRYPT 2003*, LNCS 2656 (2003), 374–387.

[29] F. Hess, "Generalising the GHS Attack on the Elliptic Curve Discrete Logarithm Problem", *LMS Journal of Computation and Mathematics*, 7 (2004), 167–192.

[30] Magma Computational Algebra System version 2.19-7, Online public calculator available at http://magma.maths.usyd.edu.au/magma/.

[31] J.-J. Chi and T. Oliveira, "Attacking a binary GLS elliptic curve with magma", *Progress in Cryptology - LATINCRYPT, 2015*, LNCS 9230 (2015), 308–326.

[32] M. Velichka, M. Jacobson Jr, and A. Stein, "Computing discrete logarithms in the jacobian of high-genus hyperelliptic curves over even characteristic finite fields", *Mathematics of Computation*, AMS 83 (2014), 935–963.

[33] M. Jacobson, A. Menezes, and A. Stein, "Solving elliptic curve discrete logarithm problems using Weil descent", *Journal of the Ramanujan Mathematical Socciety*, 16 (2001), 231–260.

[34] E.-R. Berlekamp, "Factoring polynomials over finite fields", *The Bell System Technical Journal*, 46 (1967), 1853–1859.

[35] E.-R. Berlekamp, "Factoring Polynomials Over Large Finite Fields", *Mathematics of Computation*, 24 (1970), 713–735

[36] D.-G. Cantor and H. Zassenhaus, "A New Algorithm for Factoring Polynomials Over Finite Fields", *Mathematics of Computation*, 36 (1981), 587–592.

[37] J. von zur Gathen and D. Panario, "Factoring Polynomials Over Finite Fields: A Survey" *Journal of Symbolic Computation*, 31 (2001), 3–7.

[38] "Finding Roots in GF(p^n) with the Successive Resultant Algorithm" Cryptology ePrint Archive: Report 2014/506. Available http://eprint.iacr.org/2014/506.

[39] J.-H. Davenport, C. Petit, and B. Pring, "A Generalised Successive Resultants Algorithm", *Arithmetic of Finite Fields - WAIFI 2016*, LNCS 10064 (2016), 105–124.

[40] L. De Feo, C. Petit, and M. Quisquater, "Deterministic root finding in finite fields" *ACM Comm. Computer Algebra*, 49 (2015), 87-89.

[41] G. Adj, D. Cervantes-Vázquez, J. Chi-Domínguez, A. Menezes and F. Rodríguez-Henríquez, "On the cost or computing isogenies between supersingular elliptic curves", Cryptology ePrint Archive: Report 2018/313. Available http://eprint.iacr.org/2018/313.

[42] D. 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.

[43] 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.

[44] D. Charles, E. Goren and K. Lauter, "Cryptographic hash functions from expander graphs", *Journal of Cryptology*, 22 (2009), 93–113.

[45] C. Costello et al., SIDH Library, https://www.microsoft.com/en-us/research/project/sidh-library/.

[46] C. Costello and H. Hisil, "A simple and compact algorithm for SIDH with arbitrary degree isogenies", *Advances in Cryptology — ASIACRYPT 2017*, LNCS 10624 (2017), 303–329.

[47] C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes and D. Urbanik, "Efficient compression of SIDH public keys", *Advances in Cryptology — EUROCRYPT 2017*, LNCS 10210 (2017), 679–706.

[48] C. Costello, P. Longa and M. Naehrig, "Efficient algorithms for supersingular isogeny Diffie-Hellman", *Advances in Cryptology — CRYPTO 2016*, LNCS 9814 (2016), 572–601.

[49] L. De Feo, D. Jao and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies", *Journal of Mathematical Cryptology*, 8 (2014), 209–247.

[50] A. Faz-Hernández, J. López, E. Ochoa-Jiménez and F. Rodríguez-Henríquez, "A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol", *IEEE Transactions on Computers*, 67 (2018), 1622–1636.

[51] S. 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.

[52] L. 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.

[53] D. Jao et al., "Supersingular isogeny key encapsulation", Round 1 submission, NIST Post-Quantum Cryptography Standardization, November 30, 2017.

[54] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies", *Post-Quantum Cryptography — PQCrypto 2011*, LNCS 7071 (2011), 19–34.

[55] D. Jao and V. Soukharev, "Isogeny-based quantum-resistant undeniable signatures", *Post-Quantum Cryptography — PQCrypto 2014*, LNCS 8772 (2014), 160–179.

[56] S. Jaques and J. Schanck: "Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE", Cryptology ePrint Archive: Report 2019/103. Available http://eprint.iacr.org/2019/103.

[57] B. Koziel, R. Azarderakhsh and M. Mozaffari-Kermani, "Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA", *Progress in Cryptology — INDOCRYPT 2016*, LNCS 10095 (2016), 191–206.

[58] National Institute of Standards and Technology, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process", December 2016. Available from https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[59] 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.

[60] P. van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, 12 (1999), 1–28.

[61] C. Petit, "Faster algorithms for isogeny problems using torsion point images", *Advances in Cryptology — ASIACRYPT 2017*, LNCS 10625 (2017), 330–353.

[62] 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.

[63] R. Schoof, "Nonsingular plane cubic curves over finite fields", *Journal of Combinatorial Theory, Series A*, 46 (1987), 183–211.

[64] A. Shamir, "Factoring large numbers with the TWINKLE device", *Cryptographic Hardware and Embedded Systems — CHES 1999*, LNCS 1717 (1999), 2–12.

[65] A. Shamir and E. Tromer, "Factoring large numbers with the TWIRL device", *Advances in Cryptology — CRYPTO 2003*, LNCS 2729 (2003), 1–26.

[66] S. Tani, "Claw finding algorithms using quantum walk", *Theoretical Computer Science*, 410 (2009), 5285–5297.

[67] J. Vélu, "Isogénies entre courbes elliptiques", *C. R. Acad. Sc. Paris*, 273 (1971), 238–241.

[68] Wikipedia, "Sunway TaihuLight", https://en.wikipedia.org/wiki/Sunway_TaihuLight.

[69] Wikipedia, "Exabyte", https://en.wikipedia.org/wiki/Exabyte#Google.

[70] 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.

[71] C. Zalka, "Grover's quantum searching algorithm is optimal", *Physical Review A*, 60 (1999), 2746–2751.

[72] 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.

[73] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: An Efficient Post-Quantum Commutative Group Action", *Advances in Cryptology — ASIACRYPT 2018*, LNCS 11274 (2018), 395–427.

[74] L. De Feo and S. Galbraith, "SeaSign: Compact Isogeny Signatures from Class Group Actions", *Advances in Cryptology — EUROCRYPT 2019*, LNCS 11478 (2019), 759–789.

[75] T. Decru, L. Panny, and F. Vercauteren, "Faster SeaSign Signatures Through Improved Rejection Sampling", *Post-Quantum Cryptography — PQCrypto 2019*, LNCS 11505 (2019), 271–285.

[76] W. Beullens, T. Kleinjung, F. Vercauteren, "CSI-FiSh: Efficient Isogeny based Signatures through Class Group Computations", Cryptology ePrint Archive: Report 2019/498. Available http://eprint.iacr.org/2019/498.

[77] M. Meyer and S. Reith, "A Faster Way to the CSIDH", *Progress in Cryptology — INDOCRYPT 2018*, LNCS 11356 (2018), 137–152.

[78] M. Meyer, F. Campos, and S. Reith, "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH", *Post-Quantum Cryptography — PQCrypto 2019*, LNCS 11505 (2019), 307–325.

[79] A. Jalali, R. Azarderakhsh, M. Kermani, and D. Jao, "Towards Optimized and Constant-Time CSIDH on Embedded Devices", *Constructive Side-Channel Analysis and Secure Design — COSADE 2019*, LNCS 11421 (2019), 215–231.

[80] D. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: elliptic-curve points indistinguishable from uniform random strings", *Conference on Computer and Communications Security — CCS 2013*, ACM (2013), 967–980.

[81] D. Bernstein, T. Lange, C. Martindale, and L. Panny, "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies", *Advances in Cryptology — EUROCRYPT 2019*, LNCS 11477 (2019), 409–441.

[82] H. Onuki, Y. Aikawa, T. Yamazaki, and T. Takagi, "A Faster Constant-time Algorithm of CSIDH keeping Two Torsion Points", Cryptology ePrint Archive: Report 2019/353. Available http://eprint.iacr.org/2019/353.

[83] D. Moody and D. Shumow, "Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves", *Mathematics of Computation*, AMS 85 (2016), 1929–1951.

[84] W. Castryck, S. Galbraith, and R. Farashahi, Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. Cryptology ePrint Archive: Report 2008/218. Available http://eprint.iacr.org/2008/218.

[85] S. Kim, K. Yoon, J. Kwon, S. Hong, and Young-Ho Park, "Efficient Isogeny Computations on Twisted Edwards Curves", *Security and Communication Networks* (2018).

[86] S. Kim, K. Yoon, J. Kwon, Y. Park, and S. Hong, "New Hybrid Method for Isogeny-based Cryptosystems using Edwards Curves", Cryptology ePrint Archive: Report 2018/1215, Available http://eprint.iacr.org/2018/1215.

[87] S. Kim, K. Yoon, Y. Park, S. Hong, "Optimized Method for Computing Odd-Degree Isogenies on Edwards Curves", Cryptology ePrint Archive: Report 2019/110, Available http://eprint.iacr.org/2019/110.

[88] M. Meyer, S. Reith, and F. Campos, "On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic", Cryptology ePrint Archive: Report 2017/1213, Available http://eprint.iacr.org/2017/1213.

[89] P. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization", *Mathematics of Computation*, 48 (1987), 243–234.

[90] C. Costello and B. Smith, "Montgomery curves and their arithmetic - The case of large characteristic fields", *Journal of Cryptographic Engineering* 8 (2018), 227–240.

[91] J. Vélu, "Isogénies entre courbes elliptiques", *Comptes Rendus de l'Académie des Sciences des Paris*, 273 (1971), 238–241.

[92] D. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves", *Progress in Cryptology — AFRICACRYPT 2008*, LNCS 5023 (2008), 389–405.

[93] A. Gélin and B. Wesolowski, "Loop-Abort Faults on Supersingular Isogeny Cryptosystems", *Post-Quantum Cryptography — PQCrypto 2017*, LNCS 10346 (2017), 93–106.

[94] Y. Ti, "Fault Attack on Supersingular Isogeny Cryptosystems", *Post-Quantum Cryptography — PQCrypto 2017*, LNCS 10346 (2017), 107–122.

[95] A. Childs, D. Jao, and V. Soukharev, "Constructing elliptic curve isogenies in quantum subexponential time", *Journal of Mathematical Cryptology*, 8 (2014), 1–29.

[96] L. De Feo, J. Kieffer, and B. Smith, "Towards Practical Key Exchange from Ordinary Isogeny Graphs", *Advances in Cryptology — ASIACRYPT 2018*, LNCS 11274 (2018), 365–394.

[97] G. Dequen, S. Ionica, and M. Trimoska, "Time-Memory Trade-offs for Parallel Collision Search Algorithms", Cryptology ePrint Archive: Report 2017/581. Available http://eprint.iacr.org/2017/581.

[98] D. Martins, G. Banegas, and R. Custódio, "Don't Forget Your Roots: Constant-Time Root Finding over $\mathbb{F}_{2^m}$", *Progress in Cryptology - LATINCRYPT 2019*. LNCS 11774 (2019), 109–129.

[99] G. Adj, D. Cervantes-Vázquez, J. Chi-Domínguez, A. Menezes and F. Rodríguez-Henríquez, "On the cost or computing isogenies between supersingular elliptic curves", *Selected Areas in Cryptography — SAC 2018*. LNCS 11349 (2019), 322–343.

[100] D. Cervantes-Vázquez, M. Chenu, J.-J. Chi-Domínguez, L. De Feo, F. Rodríguez-Henríquez, and Benjamin Smith, "Stronger and Faster Side-Channel Protections for CSIDH", *Progress in Cryptology - LATINCRYPT 2019*. LNCS 11774 (2019), 173-193