Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Estimados de Seguridad Cuántica para Criptografía Basada en Isogenias y Nuevas Primitivas Criptográficas Basadas en Isogenias

Tesis que presenta

Jorge Emmanuel Chávez Saab

para obtener el Grado de

Doctor en Ciencias

en Computación

Directores de Tesis

Dr. Francisco Rodríguez Henríquez

Dr. Cuauhtemoc Mancillas López

Ciudad de México Julio 2023

Center for Research and Advanced Studies of
the National Polytechnic Institute

Zacatenco Campus

Computer Science Department

Quantum Security Estimates for Isogeny-based Cryptography and New Isogeny-based Cryptographic Primitives

A dissertation submitted by

Jorge Emmanuel Chávez Saab

for the degree of

Doctor in

Computer Sciences

Thesis advisors:

Dr. Francisco Rodríguez Henríquez

Dr. Cuauhtemoc Mancillas López

Mexico City July 2023

A mi familia que siempre me apoyó.

Acknowledgements

The author acknowledges and thanks Mexico's Consejo Nacional de Ciencia y Tecnología (CONACYT) for the maintainance grant provided throughout the duration of this doctoral proyect.

Spanish/Español

El autor agradece al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca de manutención aportada por la duración de este proyecto doctoral.

Summary

This doctoral project is focused in contributing to the development of post-quantum cryptography, specifically in the area of cryptography based on isogenies of elliptic curves. We have studied the security of various schemes by simulating the best known quantum attacks against them, as well as by implementing large-scale classical attacks to understand their efficiency and practicality. We have used our results to sustain the claim that many of the protocols proposed in the literature are not secure when using their original parameters, specially the isogeny-based Diffie-Hellman protocol known as CSIDH. We also extrapolated our results to larger instances in order propose new parameter sets that we deem secure. Moreover, we have provided highly efficient and constant-time implementations of the CSIDH protocol, both with the original and the newly proposed parameters. Finally, we have also proposed novel protocols that implement other cryptographic primitives based on isogenies and elliptic curves, specifically a verifiable delay function and an admissible encoding function for points in ordinary elliptic curves that is currently the most efficient one in the literature.

Spanish/Español

Este proyecto doctoral se ha dedicado a contribuir al desarrollo de la criptografía postcuántica, específicamente en el área de criptografía basada en isogenias de curvas elípticas. Se ha estudiado la seguridad de diversos esquemas haciendo simulaciones de ataques cuánticos y también implementando ataques clásicos a grandes escalas para medir su eficiencia y practicidad. Con base en estos estudios, hemos podido sustentar la afirmación de que muchos de los protocolos propuestos en la literatura no son lo suficientemente seguros usando sus parámetros originales, especialmente el protocolo de Diffie-Hellman basado en isogenias conocido como CSIDH. A su vez, hemos extrapolado nuestro estudio a instancias con parámetros de mayor tamaño, para proponer nuevos conjuntos de parámetros que sí son seguros. Hemos también realizado implementaciones altamente optimizadas y en tiempo constante de CSIDH, tanto con los parámetros originales como con los aquí propuestos. Finalmente, se han propuesto protocolos novedosos que implementan diferentes primitivas criptográficas basadas en isogenias y curvas elípticas, específicamente una función de retraso verificable y una función de codificado admisible para puntos en curvas elípticas ordinarias que es, a la fecha, la más eficiente de su tipo en la literatura.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Context and Motivation

Since its start in the 1950s, the digital revolution has continued to periodically bring about new technologies that drastically change our everyday lives. One of the areas that has seen great advances is that of digital communications, which include everything from cellphones and the internet to credit card payments and bank transfers. With the increasing ubiquitousness of digital communications in our lives, there has been a growing concern for studying how to keep these communications safe: for example, how to ensure a message will only be read by the intended recipient or how to verify that a message does indeed come from the source it indicates. Cryptography is the science that studies these questions and comes up with solutions in the form of cryptographic schemes such as public key encryption schemes or digital signature schemes.

Although a great variety of schemes are known and have been studied for either encrypting or signing a message, only a few of them are used in practice. This is mainly due to resource constraints since large volumes of traffic, the need for fast transactions, and the desire to enable safe communications on lightweight devices all require algorithms that are highly efficient both in terms of computational power and key storage size. Most of the schemes used today are based on RSA and Elliptic Curve Cryptography, which have levels of security based on the hardness of factoring a large integer and of computing logarithms over large finite groups, respectively.

The next major advance in the digital revolution is expected to be the emergence of quantum computation. First theorized by Richard Feynman in 1981, a quantum computer is a machine that exploits the laws of quantum physics to perform calculations with a computational power much greater than any classical computer. While a classical bit always takes one of two values, a quantum bit, or qubit, can be in a superposition of the two states. This leads to an unbounded increase in the amount of information that can be stored in one bit, and allows for superposed computations with different input bits at once similar to the model of a non-deterministic state machine.

While quantum computers could positively affect our lives by enabling new computational capabilities, the facet of quantum computation that has become more prominent is as a threat to the security of our communications. This has been evident since 1994, when MIT mathematician Peter Shor showed the existence of algorithms that can efficiently solve both the factorization and discrete logarithm problems, thus violating the security of all cryptographic schemes used today. The imminent arrival of quantum computation has thus created an urgent need to revise our current schemes and optimize new ones that are based on different problems which are not known to be solvable even for a quantum computer. This has given raise to the area known as post-quantum cryptography.

It may still be decades for the first useful quantum computers to be available, but that does not mean that the study of post-quantum cryptography is irrelevant in the meantime. For one, it is clear that current post-quantum schemes are much more inefficient than RSA or elliptic curve schemes and so could not be immediately applied in all areas once quantum computers arrive. It is thus imperative to dedicate research now to the optimization of post-quantum schemes so that we can be ready when quantum computation arrives. This urge is even more pressing when considering the fact that a malicious agent could be storing encrypted information now to decrypt it once a quantum computer is available, so if there is any information that we consider may still be sensitive a few years down the road, we should already be using post-quantum encryption on it now.

Finally, the other reason why its important to dedicate work to post-quantum cryptography now is to start building up confidence on it. The security of cryptographic schemes is very rarely demonstrable, but is rather conjectured based on our lack of success in trying to break them. Therefore, it is important to spend a large amount of time putting these systems to test in order to convince ourselves of their security, a discipline known as cryptanalysis.

1.2 Problem Exposition

This project aims to advance the development of post-quantum cryptography, which is achieved by analyzing existing protocols and programming efficient implementations of them, as well as by proposing and implementing novel post-quantum constructions. Specifically, we focus on isogeny-based cryptography which is one of the principal candidates being considered for post-quantum cryptography in the form of the SIDH [1, 2] and CSIDH [3] protocols.

We analyze the existing protocols by considering the known quantum attacks and simulating the size of the circuit that one would need to implement them. This results in an assessment of the quantum security provided by the currently proposed parameters, as well as a more informed proposal for new sets of parameters that conform more suitably with the intended security goals. Additionally, we develop programming optimizations for these protocols and provide efficient and secure implementations of them.

We also propose brand new constructions based on elliptic curve isogenies that lead to post-quantum versions of useful cryptographic primitives other than encryption. This is a topic that has been largely overlooked, with public key encryption and signatures gathering most of the attention in post-quantum cryptography. For instance, we propose the use of isogenies to obtain a post-quantum version of a verifiable delay function, or constructing hash functions into the set of points of an elliptic curve.

As a secondary aim, we also advance the development of classical elliptic curve cryptography, specifically by proposing a novel construction for efficiently encoding and hashing to points in an ordinary curve. While this task is of lesser interest for postquantum cryptography, which usually focuses on supersingular curves, it holds immense interest for several classical algorithms that are widely in use today and will continue to be so for years. Moreover, despite not being applicable to postquantum protocol, the construction we propose is not isolated from the context of the rest of this project, as it exploits much of the same elliptic curve and isogeny frameworks that are used in our other constructions.

1.3 Objectives and contributions

The specific objectives of this project were the following:

1. Assess the quantum security of existing isogeny-based protocols by obtaining a numeric estimation of the resources needed to carry out a quantum attack on them
2. Based on these results, suggest new parameter sizes for existing protocols that conform better to the security goals
3. Assess also the classical security of these protocols by launching classical attacks on small-parameter instances and comparing the performances of different attacks
4. Optimize existing isogeny-based protocols and provide efficient secure implementations of them
5. Build upon the existing isogeny-based framework to obtain cryptographic primitives for purposes other than encryption, which are also quantum-secure
6. Exploit the acquired dominance of the elliptic curve and isogeny frameworks to also propose new primitives for classical elliptic curve cryptography.

By the end of the project, we are able to claim the following specific contributions:

1. Provided the first constant-time C implementation of the CSIDH protocol using the sublinear Vélu formulas [4]

2. Published quantum circuit estimates that deem the security of original CSIDH instances insufficient [4]

3. Proposed and implemented larger CSIDH instances along with new quantum circuit estimates to conform to the original security goal [4]

4. Provided parallel implementations of the meet-in-the-middle and van Oorschot-Wiener attacks for isogeny path finding [5]

5. Presented precise estimates for the cost of the previous attacks for a given setup and used them to derive classically-secure parameters [5]

6. Proposed the first instance of a verifiable delay function with post-quantum security using the isogeny framework [6]

7. Proposed and implemented the most efficient hash function into a large set of ordinary elliptic curves known to date [7]

1.4 Structure of the manuscript

The remainder of this manuscript is structured as follows. In Chapter 2, we go over all the background that is relevant to understanding the contributions of this project, including topics on quantum computation, elliptic curves and isogenies. In Chapter 3 we describe the state-of-the-art for isogeny-based cryptography focusing on two protocols, CSIDH and SIDH, and the various recent efforts in cryptanalysis that have advanced our understanding of them. For the reader interested only in the general idea of this project's contributions, Chapter 4 gives a brief overview of each of the four works that were developed. These are then presented in more depth in the remaining chapters: first, a major analysis of the quantum security of the CSIDH protocol along with a proposal of new parameters that is covered in more detail in Chapter 5. Second, a parallel implementation and a comprehensive analysis of the classical attacks that are applicable to isogeny-based cryptography in general, presented in Chapter 6. Third, a proposal of the first post-quantum verifiable delay function based on isogenies, which is contained in Chapter 7. And fourth, a new construction for encoding points in ordinary elliptic curves and hashing onto them that is covered in Chapter 8. Finally, Chapter 9 contains the concluding remarks and future work ensuing from this project.

Chapter 2

Background

In this section we present all the background that is relevant to better understand the contributions of this project. We begin by presenting an overview of quantum computation in Section 2.1 and of post-quantum cryptography candidates in Section 2.2. The mathematical background of elliptic curves and isogenies is then covered in detail in Section 2.3.

2.1 Quantum Computation

Quantum computation provides a more powerful model of computation which, despite currently remaining mainly theoretical due to engineering constraints, holds great promise in pushing the frontier for problems that are considered feasibly solvable. Most importantly, the factoring problem and the discrete logarithm problem, on which virtually all cryptographic systems that are used today are based on, are both known to be solvable in polynomial time by a quantum computer.

In this section we briefly describe the model by breaking it into its two main components: qubits and operators. This is followed by a brief discussion on reversibility and cost metrics, and lastly we present a short overview of key quantum algorithms.

2.1.1 Qubits

Unlike classical computers, which use high or low voltages to represent the discrete binary values of bits, quantum computers operate on the *spin states* of spin-1/2 particles like electrons or nuclei. Each particle constitutes a *qubit*, and its spin state is a vector in a complex Hilbert space of dimension 2, whose orthonormal basis vectors are denoted $|0\rangle$ and $|1\rangle$. Each qubit can be in either of these states, but also in a *superposition* (linear combination over the complex numbers) of them.

Multiple qubits can be described by a single Hilbert space by taking the tensor product of each qubit's space, so the two-qubit vector $|1\rangle \otimes |0\rangle$ (or $|10\rangle$ for short) represents the

first qubit in state $|1\rangle$ and the second in $|0\rangle$. A general state of n qubits can be written as

$$|\Psi\rangle = \sum_b \alpha_b |b_1 \dots b_n\rangle ,$$

where the sum is over all n-bit numbers and the amplitudes α_b are complex constants. The state itself is not accessible to a computer, but only the result of a randomized measurement which returns state $|b\rangle$ with probability proportional to $|\alpha_b|^2$. After the measurement, the state collapses to just $|\Psi\rangle = |b\rangle$, so all other information about the state is lost.

2.1.2 Operators

We use linear operators to modify qubits and realize computations, analogous to the use of gates for classical bits. As per the laws of quantum physics, the time evolution of any state can always be described by a unitary linear operator. If we write the basis vectors as column vectors,

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} ,$$

then any operation on a single qubit can be expressed as a 2×2 unitary matrix. Likewise, operations involving two qubits can be expressed as 4×4 unitary matrices. We typically consider only these cases, with operations on several qubits obtained by composing simpler two-qubit operations.

A set of quantum operators is said to be *universal* if any unitary operator can be approximated by a series of operators in the set, up to an arbitrarily small loss in precision. A usual universal set of operators consists [8] of the one-qubit operators

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} , \; H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} , \; T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

as well as the two-qubit *controlled-NOT* operator,

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} ,$$

which flips the second qubit if and only if the first qubit is in the $|1\rangle$ state.

2.1.3 Reversible Computation and Gate Costs

An important consequence of operators being necessarily unitary is that of *reversibility*: unlike with classical computers, where actions like clearing or overwriting a bit are considered trivial, quantum operations on qubits must always be reversible. Consider for

example the task of computing the AND of two bits b_1, b_2. In a classical computer with only two bits of memory one could compute the result and overwrite it in b_1 while leaving b_2 the same, which leads to the following truth table:

b_1^{in}	b_2^{in}	b_1^{out}	b_2^{out}
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	1

This function is not realizable by a quantum computer since output (0,0) is produced by both inputs (0,0) and (1,0), making reversibility impossible. The workaround for a quantum computer is to use a third qubit b_3, called an *ancilla bit*, and the three-qubit *Toffoli gate*

$$F = \begin{pmatrix} I_{6\times 6} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

which has the effect of flipping the state of b_3 if and only if b_1 and b_2 are both in state $|1\rangle$, leaving the other qubits intact. That is, it maps $|b_1, b_2, b_3\rangle \mapsto |b_1, b_2, b_3 \oplus b_1 b_2\rangle$ and as long as the ancilla qubit b_3 starts in state $|0\rangle$, it will end up holding the correct result. Note that an ancilla qubit with a different initial state may produce undesired results, so convention dictates that before deallocating the ancilla we must perform the inverse calculation (in this case, the Toffoli gate is its own inverse) to return it to the $|0\rangle$ state (this process is known as *uncomputing*).

Because the AND gate cannot be computed using only the two qubits involved, we say that it is *non-linear*. The process described in the previous paragraph is an example of *Bennett's conversion*, which translates any classical program into a quantum one following a simple rule:

- For each non-linear gate acting on bits x, y to produce $f(x, y)$, allocate an ancilla qubit z (assumed to be in state $|0\rangle$) and perform the reversible calculation $|x, y, z\rangle \mapsto |x, y, z \oplus f(x, y)\rangle$ (e.j. by using a Toffoli gate)

- Use the value of $f(x, y)$ on other calculations as needed, but never changing the state of the initial three qubits

- Once $f(x, y)$ is not needed anymore, perform the inverse calculation on the three qubits to revert $|x, y, z \oplus f(x, y)\rangle \mapsto |x, y, z\rangle$

Just like the AND gate, the OR and NAND gates are also non-linear and must follow this procedure with Toffoli gate implementations for step one, whereas the NOT and XOR gates are both linear and can be implemented in a more straightforward way (the

NOT gate, for example, can be computed using only the single qubit involved by simply applying the N operator).

On top of the additional steps and ancilla qubits required for non-linear gates, their implementations all make use of Toffoli gates. Each Toffoli gate, in turn, is implemented with seven T operators, which are widely thought to be considerably more expensive than the other operators (N, H, CN). Because of this, some authors opt for a cost metric that counts only the number of non-linear gates, disregarding the number of linear ones [9, 10, 11].

2.1.4 Quantum Algorithms

We are now ready to define a quantum algorithm as a fixed number of qubits along with a circuit of operators to be applied to them, called the *quantum circuit*, whose size should scale efficiently with the size of the input (for general purposes, efficiency means that something grows only as a polynomial function of the input size). Since we have discussed how to emulate the AND and NOT gates, it should be clear that any classical algorithm for computing a function can be emulated by some quantum circuit given a large enough number of ancilla qubits. The required number of qubits and the size of the quantum circuit are the so-called *quantum resources* that we are interested in estimating.

Given that emulating a function through a quantum circuit always requires additional ancilla qubits and more gates, the utility of this construction remains unclear so far. However, the real power of quantum computation starts to show once we exploit the properties of superposition. Given a function $f : \{0,1\}^n \mapsto \{0,1\}^m$, consider starting with $n+m$ qubits in the $|0\rangle$ state and applying an H gate to each of the first n qubits to obtain

$$(H|0\rangle)^{\otimes n} \otimes |0\rangle^{\otimes m} = \frac{1}{2^{n/2}}(|0\rangle + |1\rangle)^{\otimes n} \otimes |0\rangle^{\otimes m} = \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |0\rangle^{\otimes m},$$

which has a superposition of all possible n-bit inputs in the first n qubits. If we are able to construct a quantum circuit that emulates f storing the result in the last m qubits (using additional ancilla bits not shown here), then we could apply this circuit to our superposition and thanks to the linearity of operators we will obtain the new superposition

$$\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |f(x)\rangle$$

which has, in a way, computed the function f for all possible values of x at once, somewhat resembling the concept of a non-deterministic machine. Recall, however, that what is accessible is not the state itself but the result of a randomized measurement. In 1996, Grover published his algorithm [12] which, given a value of y, modifies the

amplitudes of this superposition iteratively so that after just $\mathcal{O}(\sqrt{N})$ iterations, where $N = 2^n$, a measurement will have probability above 25% of producing a state $|x\rangle \otimes |f(x)\rangle$ where $f(x) = y$, yielding a noticeable improvement over the $\mathcal{O}(N)$ evaluations needed for a classical exhaustive search.

An even greater proof of the power of quantum computation is **Shor's Algorithm**, published in 1994 [13], which factorizes an integer N using only $\mathcal{O}((\log N)^2(\log\log N)(\log\log\log N))$ quantum operators, providing an exponential improvement over the best known classical algorithm and showing that the security of RSA cryptography can be easily broken provided that a large enough stable quantum computer can be built. Shor also presented an algorithm for solving the *abelian hidden subgroup problem*, which solves the discrete logarithm problem for any finite abelian group in subexponential time, breaking also the security of all Diffie-Hellman-based protocols.

2.2 Post-Quantum Cryptography

In view of the threat posed by quantum computation, the goal of post-quantum cryptography is therefore to find protocols that are based on problems different than factorization and discrete logarithms, which are not known to be easily solvable by a quantum computer. Even if a problem is not easily solvable by a quantum algorithm, quantum computation can still facilitate the attack via generic methods such as the Grover search. Therefore, assessing a post-quantum protocol's security always requires the consideration of quantum attacks and the resources needed to carry them out, usually in the form of three metrics: the total number of qubits, the circuit size (total number of gates applied), and the circuit depth (maximum number of gates applied to the same qubit). The later is specially important since qubit decoherence due to noise limits the number of times we can operate on a qubit realistically before its information degrades.

Although several algorithms exist that are not known to be breakable by a quantum computer, these are plagued by the recurring problem of excessively long key sizes or encryption/decryption times, leaving much work to be done in the research for new quantum-resistant algorithms and efficient implementations. Since 2016, the United States' *National Institute for Standards and Technology* (NIST) had started a process [14] to solicit and evaluate quantum-resistant cryptographic algorithms both for digital signatures and public key encapsulation, which finalized in 2022. Symmetric key schemes are not considered since their constructions are more heuristic and, as far as we know, there are no quantum-based threats beyond the usual Grover search which reduces the effective key length for an exhaustive search by one half.

The NIST candidates fell into five main areas: hash-based, code-based, lattice-based, multivariate, and elliptic curve isogeny-based, which tend to vary greatly in terms of key or signature lengths. Table 2.1 shows the typical lengths by referring to a representative scheme from each area in the NIST security level 1. When it comes to key encapsulation

versus digital signatures, we note that some areas have a distinct strength in one category over the other. Isogeny-based cryprtography, which is the focus of this work, does not have any particularily good signature candidates. However, it has a niche application for PKE since it has the smallest key sizes despite not having the best timing performance.

Area	Public key	Private key	Signature
Isogeny	751[15]	48[15]	141,312[16]
Hash	64[17]	128[17]	29,792[17]
Code	1MB[18]	11.5 KB[18]	-
Lattice	6,130[19]	6,743[19]	1,024[19]
Multivariate	15 KB[20]	10 KB[20]	-

Table 2.1: Typical key and signature lengths (in bytes) for representative schemes in each area of post quantum cryptography. The schemes being referred to are shown within citation.

The NIST process finalized with a selection of mostly lattice-based protocols along with a single hash-based signature scheme, while SIDH [1, 2], the only isogeny-based candidate, had been designated an "alternative candidate" probably due to its poor performance but niche characteristics. Later, SIDH was completely ruled out due to the discovery of a polynomial-time attack by Castryck and Decru [21] which completely broke its security by exploiting additional information leaked in the protocol. However, this attack does not generalize to other isogeny-based protocol that don't have this leakage, and the niche characteristics of isogeny-based cryptography mean that it is still a topic of ongoing interest despite having missed the NIST standardization process. For instance, a newer isogeny-based protocol called CSIDH [3] is the only post-quantum candidate to provide optimally small keys (in the sense that the key size is just enough to guarantee the desired number of possible keys), and is also the only non-interactive postquantum key exchange. Additionally, isogeny-based cryptography has the advantage of relying on elliptic curves which have been in use classically for a long time. This means that they have been studied with mathematical robustness for much longer than other areas, and also that they can exploit many parts of the elliptic curve libraries that have already been widely rolled out today.

2.3 Elliptic Curves and Isogenies

Elliptic curve cryptography is the basis for several cryptographic schemes that are used today both for key exchanges and digital signatures. However, its security is based on the difficulty of the discrete logarithm problem and so elliptic curve cyptography by itself is not post-quantum secure. To work around this, new cryptographic schemes have been developed that exploit the idea of isogenies over elliptic curves. These schemes are thought

to be resistant to quantum attacks while maintaining the framework of elliptic curves and the short key sizes that characterize them. To fully understand elliptic curve isogenies, we must first go over the general concepts of classical elliptic curve cryptography.

2.3.1 Elliptic Curve Basics

An **elliptic curve** is a an equation of the form

$$E: \ y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_6,$$

where x, y are variables and a_i are constants, all over a given field $\mathbb{F}$ and such that the curve is non-singular. Whenever the field characteristic is not 2 or 3 (which we will assume throughout), this equation can be brought to the *short Weiestrass form*

$$E: \ y^2 = x^3 + ax + b$$

via a simple change of variables. In this form, the condition that the curve be non-singular is equivalent to the **discriminant** $\Delta_E = -16(4a^3 + 27b^2)$ being nonzero.

We use $E(\mathbb{F})$ to refer to the set of points $(x, y) \in \mathbb{F}^2$ that satisfy this equation, whose size can be estimated by Hasse's Theorem [22]:

Theorem 2.3.1 (Hasse)**.** *The number of points in an elliptic curve over a finite field of size $q = p^n$ is always $\#E(\mathbb{F}) = q - t + 1$ for some integer t called the Frobenius trace, which takes values in the interval $-2\sqrt{p} \leq t < 2\sqrt{p}$.*

This set can be expanded into an additive group by appending an identity element 0 and defining the group operation

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3),$$

where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

if $(x_1, y_1) \neq (x_2, y_2)$, or

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

if $(x_1, y_1) = (x_2, y_2)$, and with additive inverses defined by

$$-(x, y) = (x, -y).$$

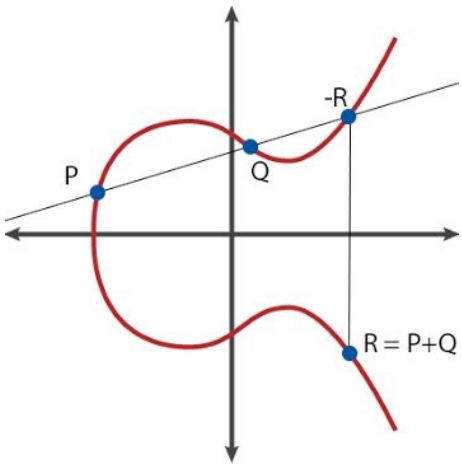


Figure 2.1: Group operation $P + Q$ over $\mathbb{R}$ for the elliptic curve shown in red.

The geometric motivation for this operation is simple: when $\mathbb{F} = \mathbb{R}$, we have $P + Q = R$ where R is the x-axis reflection of the third point that lies on the line going through P and Q, as shown in Fig. 2.1. A result of interest for many applications is that the points of order 2 in this group are of the form $(x, 0)$ where x is a root of $x^3 + ax + b$, so there either zero or three such points if the discriminant Δ_E is a square, and exactly one if it is not.

In practical applications, the field is taken to be $\mathbb{F}_{p^m}$ for some large prime p, so $E(\mathbb{F}_{p^m})$ is a finite group. One can use the Diffie-Hellman protocol over this group (known as the ECDH protocol) by defining an initial point P of prime order r and choosing a private key $d \in \mathbb{F}_r^*$, then publishing $Q = [d]P$ (which denotes P added to itself d times) as the public key. Two parties with public keys $Q_A = [d_A]P$ and $Q_B = [d_B]P$, can easily compute the common point $[d_A d_B]P$ by multiplying each other's public key by their own private key, and the x coordinate of this point is taken as the shared key. The security of ECDH therefore relies on the following problem:

Problem 2.3.1 (Elliptic Curve Diffie-Hellman Problem)**.** *Given a generator P of a large group in an elliptic curve as well as the public keys $[d_A]P$ and $[d_B]P$, find the shared secret $[d_A d_B]P$.*

As previously mentioned, this problem is not quantum-hard since it can be solved as a discrete logarithm problem using Shor's algorithm [13].

2.3.2 Isogenies

The study of isogenies has emerged as a way to pose an elliptic curve-related problem that is not easily solvable by a quantum computer. The next sections offer a broad overview

of the most important definitions and results for elliptic curve isogenies. For proofs and a more detailed survey, the reader is referred to Washington's book [23].

An **isogeny** between elliptic curves E_1, E_2 is a rational map $\phi : E_1 \to E_2$ that preserves the group identity and is a group homomorphism. A general isogeny can be written as [22]

$$\phi : (x, y) \mapsto \left(\frac{p_1(x)}{p_2(x)}, \frac{p_3(x, y)}{p_2(x)} \right),$$

where p_i are polynomials, and the **degree** of ϕ is defined as $\max\{\deg p_1,\ \deg p_2\}$. The degree is multiplicative under composition, in the sense that $\deg(\phi_1 \circ \phi_2) = \deg(\phi_1)\deg(\phi_2)$. We use the term d-isogeny as a shorthand for an isogeny of degree d.

An important result regarding the existence of an isogeny between two elliptic curves is due to Tate [24]:

Theorem 2.3.2 (Tate)**.** *Two elliptic curves are isogenous over a field $\mathbb{F}$ if and only if they have the same number of points over that same field.*

An isogeny is said to be an **endomorphism** when the domain and codomain curves are equal. For any isogeny $\phi : E_1 \to E_2$ of degree d, there must also exist a **dual isogeny** $\hat{\phi} : E_2 \to E_2$ of the same degree such that the composition $\hat{\phi} \circ \phi$ is an endomorphism of E_1 that equals the multiplication-by-d map. The dual may be defined over a finite field extension, and so may not exist if restricting isogenies to a non-algebraically closed field.

An isogeny of degree 1 is called an **isomorphism** since it is always a bijection that preserves the group law. The **j-invariant** of a curve, defined as

$$j_E = 1728 \frac{4a^3}{4a^4 + 27b^2},$$

is important for discerning isomorphism classes since it can be shown [23] that two curves are isomorphic over an algebraically closed field if and only if their j-invariants are equal.

An isogeny is said to be **separable** if its degree equals $|\text{Ker } \phi|$, the number of elements mapped to the identity in E_2, and any isogeny can be written as the composition $\phi = \sigma \circ \pi^m$ where m is a nonnegative integer, σ is a a separable isogeny, and π is the **Frobenius endomorphism** $\pi : (x, y) \mapsto (x^p, y^p)$ with p the field characteristic. Moreover, the separable isogeny can always be written as a composition of prime-degree separable isogenies.

An important mathematical resultis that, up to composition with an isomorphism, a separable isogeny is uniquely determined by its kernel. We often exploit this fact to define isogenies before even knowing their codomain: given a curve E, one can choose a finite set of points $P_1, \ldots, P_k$ and denote by $E/ < P_1, \ldots, P_k >$ the unique curve (up to isomorphism) that is the image of E under an isogeny whose kernel is the group generated by $P_1, \ldots, P_k$. In practice, one can easily compute $E/ < P_1, \ldots, P_k >$ given the list of points $P_1, \ldots, P_k$ using Vélu's formulas. The fastest implementation of these formulas is due to Bernstein *et al.* [25], and computes an isogeny of degree d in time $\mathcal{O}(\sqrt{d})$.

A second method for constructing isogenies that is alternative to Vélu's formulas are the **modular polynomials:** for any prime ℓ, there exists a polynomial $\Phi_\ell \in \mathbb{Z}[X, Y]$ of degree $\ell + 1$, called the ℓ^{th} modular polynomial, such that two curves E and E' are $\ell-$isogenous if and only if $\Phi_\ell(j_E, j_{E'}) = 0$. Given a starting curve E, we can find the $\ell-$isogenous curves (characterized by their j-invariant) by finding the roots of $\Phi_\ell(j_E, X)$ rather than explicitly computing an isogeny. However, this method is not used in practice for large ℓ since solving a degree-ℓ equation results in a much worse complexity than Vélu's formulas.

2.3.3 Supersingular curves

Another concept frequently used in isogeny-based cryptography is that of supersingular curves. For an elliptic curve E over a finite field $\mathbb{F}_q$ of characteristic p, recall by Theorem 2.3.1 that $\#E(\mathbb{F}_q) = q + 1 - t$ for some $|t| \leq 2\sqrt{q}$. The curve is said to be **supersingular** if p divides t, and **ordinary** otherwise. As per Theorem 2.3.2, supersingular curves can only ever be isogenous to other supersingular curves, allowing us to never leave the supersingular sector while working with isogenies.

Moreover, supersingular curves over any extension of a prime field $L \supset \mathbb{F}_p$ have the important property that their $j-$invariant is always contained in $\mathbb{F}_{p^2}$, even if the curve coefficients are defined over a larger extension. In particular, every supersingular elliptic curve defined over a finite field is isomorphic to a curve defined over a quadratic field, and so we can always assume without loss of generality that the field of definition of a supersingular curve is either $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$.

Supersingular curves are useful in cryptography for two reasons:

First, they allow us to work with curves of a known group structure. For instance, given a prime of the form $p = \prod_{i=0}^{n} \ell_i - 1$ where ℓ_i are distinct small primes, we know that all supersingular curves over $\mathbb{F}_p$ for large enough p will have $t = 0$ and $\#E(\mathbb{F}_p) = p+1 = \prod_{i=0}^{n} \ell_i$ must contain exactly one subgroup of order ℓ_i for each i. We can use these small-order points to define efficiently-computable isogenies to other curves using Vélu's formulas.

Second, supersingular curves provide us with the following hard problem:

Problem 2.3.2 (Supersingular Isogeny Problem)**.** *Given two supersingular curves E_1 and E_2 that are known to be related through a large-degree isogeny, find such isogeny.*

The supersingular isogeny problem is used as the basis for the security of all isogeny-based protocols, and is thought to be hard to solve even for a quantum computer. On the other hand, the analogous problem using ordinary curves is not quantum-hard since it is known to be reducible to the abelian hidden group problem [26].

For any prime ℓ, the **supersingular isogeny graph** $G_{\mathbb{F}_q}(\ell)$ is the directed graph having $\mathbb{F}_q$-isomorphism classes of ℓ-isogenous supersingular elliptic curves (represented by their j-invariants) as vertices, and ℓ-degree isogenies as edges. The graph $G_{\mathbb{F}_q}(\ell)$ is a Ramanujan

graph [27], meaning that the diameter of the graph grows at an asymptotically optimal rate with the size of q. This makes it so that random walks in it tend to "mix" and become unpredictable at a quick pase, making them ideal for cryptographic applications.

Due to the existence of dual isogenies, the graph $G_{\mathbb{F}_q}(\ell)$ is symmetric, and it is also $(\ell+1)$-regular whenever $\left(\frac{-p}{\ell}\right) = 1$. There are two special vertices, $j = 0$ and $j = 1728$, which represent the only curves with non-constant automorphisms. Outside of these two vertices the graph contains no self-loops. For a more comprehensive study of supersingular isogeny graphs, the reader is referred to Kohel's work [28].

2.3.4 Endomorphism rings and the class group

Let $\mathrm{End}(E)$ denote the set of endomorphisms of the curve E over an algebraically closed field, together with the identity map. This set has the structure of a ring, with composition as the multiplicative operation. The structure of this ring is that of an order $\mathcal{O}$ in a $\mathbb{Q}$-algebra K. For ordinary curves, K is an imaginary quadratic field (making it commutative), but for supersingular curves it is a quaternion algebra (making it non-commutative). We use $\mathrm{Ell}(\mathcal{O})$ to refer to the set of all the curves that share the same endomorphism ring, $\mathrm{Ell}(\mathcal{O}) = \{E : \mathrm{End}(E) = \mathcal{O}\}$, which are all isogenous under the algebraic closure of the field.

We say that two left-ideals $\mathfrak{a}, \mathfrak{b} \subset \mathcal{O}$ are equivalent if there exist $\lambda \in K$ such that $\mathfrak{a} = \lambda\mathfrak{b}$. We use $\mathrm{Cl}(\mathcal{O})$ to refer to set of equivalence classes of left-$\mathcal{O}$ ideals under this relation. When K is a an imaginary quadratic field, $\mathrm{Cl}(\mathcal{O})$ takes the structure of an abelian group known as the **ideal class group**.

Regardless of whether K is an imaginary quadratic field or a quaternion algebra, the Deuring correspondence [29] states that there is a one-to-one relationship between elements of $\mathrm{Cl}(\mathcal{O})$ and isomorphism classes in $\mathrm{Ell}(\mathcal{O})$, and we use $E_\mathfrak{a}$ to refer to a curve (unique up to isomorphism) that corresponds to the class of the ideal $\mathfrak{a}$. This allows us to define an action of $\mathrm{Cl}(\mathcal{O})$ on $\mathrm{Ell}(\mathcal{O})$ through $\mathfrak{a} * E_\mathfrak{b} = E_{\mathfrak{a}\cdot\mathfrak{b}}$, which is well-defined up to isomorphisms. This action is a faithful and transitive, but it is not generally commutative in the case of supersingular curves.

Since the curves $E_\mathfrak{b}$ and $E_{\mathfrak{a}\cdot\mathfrak{b}}$ are isogenous, the action of $\mathrm{Cl}(\mathcal{O})$ can be understood as an isogeny evaluation, with the ideal class $\mathfrak{a}$ representing the isogeny $\phi_\mathfrak{a} : E_\mathfrak{b} \to E_{\mathfrak{a}\cdot\mathfrak{b}}$. While the typical description of an isogeny has well-defined domain and image curves, the representation of an isogeny as an ideal is agnostic to the curves at the endpoints. For instance, the same ideal $\mathfrak{a}$ could be used to describe an isogeny between a different pair of curves $\phi'_\mathfrak{a} : E_{\mathfrak{b}'} \to E_{\mathfrak{a}\cdot\mathfrak{b}'}$. This idea is exploited by various protocols where two isogenies can be seen as "the same isogeny but acting on a different curve", also commonly called **parallel isogenies**.

Chapter 3

State of the Art

In this section we will cover the state of the art for isogeny-based cryptography. We split our discussion between isogenies over quadratic fields $\mathbb{F}_{p^2}$ and isogenies over prime fields $\mathbb{F}_p$. As previously mentioned, no other cases need to be considered since any supersingular curve defined over a larger extension field is isomorphic to a curve over $\mathbb{F}_{p^2}$.

3.1 Isogenies over quadratic fields

In the most general case of isogenies over quadratic fields, it is common to chose a prime of the shape

$$p = \ell_A^{e_A} \ell_B^{e_B} \pm 1$$

where ℓ_A, ℓ_B are distinct small primes and $\ell_A^{e_A} \approx \ell_B^{e_B}$. We also work with isogenies of supersingular curves of Frobenius trace $t = \pm 2p$ so that the curves' cardinality is

$$\#E(\mathbb{F}_{p^2}) = p^2 - t + 1 = (p \mp 1)^2 = (2^{e_A} \cdot 3^{e_B})^2.$$

In this case the group structure is always $\mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}} \oplus \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}}$, which allows us to find a couple of generators P_A, Q_A for $E[\ell_A^{e_A}]$ (the subgroup of points whose order divides $\ell_A^{e_A}$) as well as generators P_B, Q_B for $E[\ell_B^{e_B}]$. We can use the basis to parametrize all the points order $\ell_A^{e_A}$ and $\ell_B^{e_B}$, which are used as kernel points and passed to Vélu's formulas to efficiently compute isogenies of the corresponding degrees. This setting is the most common since it allows two parties to work with two classes of isogenies which are always of coprime degree, but note that group cardinalities with a larger number of $\ell_i^{e_i}$ factors are also possible and work in the same way.

Despite being now broken, the most historically important protocol based on supersingular isogenies over quadratic fields was the **Supersingular Isogeny Diffie-Hellman** protocol (SIDH), proposed by De Feo and Jao [1, 2]. It was a key exchange scheme where Alice starts from a predefined curve E_0 and obtains an order-$\ell_A^{e_A}$ point by picking a random integer $k_A \in \mathbb{Z}_{\ell_A^{e_A}}$ and computing $R_A = P_A + k_A Q_A$ which she uses as kernel to

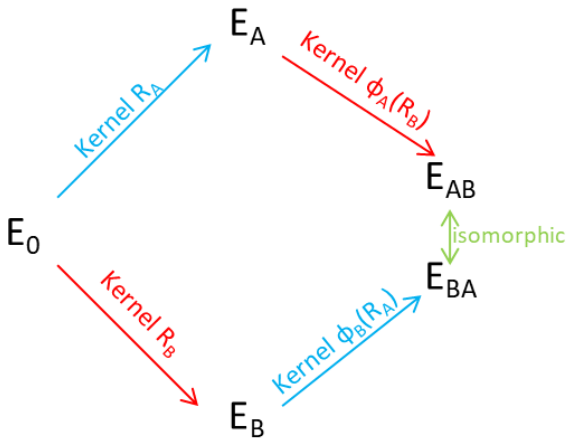


Figure 3.1: Diagram of the SIDH protocol. Arrows in blue represent isogenies computed by Alice, and arrows in red represent isogenies computed by Bob.

construct the isogeny ϕ_A and obtain the curve $E_A = E_0/ < R_A >$. On the other hand, Bob does the same using a point R_B of order $\ell_B^{e_B}$, leading to an isogeny ϕ_B and a curve $E_B = E_0/ < R_B >$. The scalars k_A, k_B act as Alice's and Bob's private keys, whereas the curves E_A and E_B act as their public keys. Additionally, Bob must publish the images $\phi_B(P_A)$ and $\phi_B(Q_A)$ of Alice's generators under his isogeny, which allows Alice to compute the image $\phi_B(R_A) = \phi_B(P_A) + k_a\phi_B(Q_A)$ of her secret point under Bob's isogeny. She uses this to compute a new isogeny starting from Bob's curve and obtain the curve $E_{BA} = E_B/ < \phi_B(R_A) >$. Analogously, Alice publishes the image of Bob's generators so that he can compute $\phi_A(R_B)$ and construct an isogeny starting from Alice's curve to obtain the curve $E_{AB} = E_A/ < \phi_A(R_B) >$. This procedure is illustrated by Figure 3.1.

Note that the composition of the isogenies from E_0 to E_A and from E_A to E_{AB} is an isogeny whose kernel is the union of $< R_A >$ and $< R_B >$, and the same is true for the composition of the isogenies from E_0 to E_B and from E_B to E_{BA}. It follows that that the curves E_{AB} and E_{BA} are isomorphic, and their common j-invariant is what is used as the protocol's shared secret.

The key encapsulation version of SIDH with concrete sets of parameters was named SIKE [15], which was one of the finalists and the only isogeny-based candidate for the NIST postquantum standardization process. It was later improved by Costello in an extension called B-SIDH [30] which increased efficiency by considering both curves of cardinality $(p+1)^2$ and $(p-1)^2$ at once. Unfortunately, it was then found by Castryck and Decru [21] that the necessary publication of the image of Alice's basis points under Bob's isogeny and vice-versa leaked important information that allows one to recover the secret isogeny kernel points. This led to a polynomial time attack that completely broke the security of SIDH, SIKE and B-SIDH.

This is not to say that the general supersingular isogeny problem has been broken, however, since the attack relies on the leakage of additional information. Many schemes that rely on a problem closer to the "pure" supersingular isogeny problem still remain secure, which include hash functions [31], signature schemes [32] and proofs of knowledge [33, 34], as well as the verifiable delay function presented in Chapter 7.

Cryptanalysis. Regarding the SIDH setting, following the devastating attack of Castryck and Decru [21] there were several countermeasures proposed including hiding the degree of the isogeny [35], the endomorphism ring of the base curve [36], or the images of the torsion points [37]. However, there have been follow-up attacks that are fast to adapt, such as that of Maino and Martindale [38] and yet another one by Robert [39] which do not rely on knowledge of the endomorphism ring anymore. It is now generally accepted that leaking torsion point images is a severe security risk, and focus has shifted to other types of schemes whose security is more closely based on Problem 2.3.2.

In the general case of the supersingular isogeny problem, the most efficient quantum algorithm for finding secret isogenies is that from Biasse, Jao and Sankar [40] which runs in time $\mathcal{O}(p^{1/4})$, where p is the field's characteristic. This algorithm exploits the algebraic structure of supersingular elliptic curves to achieve an asymptotic improvement over a bare Grover search which would run in $\mathcal{O}(p^{1/2})$. In most cases, this is in fact no better than a classical attack. For instance, finding an isogeny of degree ℓ^{e_A} would have a complexity of $\mathcal{O}(\ell^{e_A/2})$ with a classical meet-in-the middle attack, which is just as good in the common scenario that $\ell^{e_A} \approx \sqrt{p}$. Because of this, it is often argued that the overall security of the supersingular isogeny problem is bounded by its classical security alone, making it a strong candidate for post-quantum cryptography.

When choosing concrete parameters, most schemes (including SIKE) have historically based their parameter sizes based on the complexity of the meet-in-the-middle attack. However, this choice has proven to be controversial and it is argued in [41] that even the classical MITM attack is unrealistic to implement due to its memory requirements, and parameters should be chosen according to a more conservative attack that compromises between performance and memory such as the van Oorschot-Wiener golden collision finding algorithm [42]. If this line of thought is followed, then the required parameter sizes can be relaxed while maintaining security. In the case of SIKE, this had already led to a more in-depth analyses of the van Oorschot-Wiener attack and revision of the parameter sizes by the authors of the original proposal [43].

The record for a real-world implementation is currently held by Udovenko and Vitto [44] who brute-forced a degree-2^{88} isogeny using the classical MITM, but already requiring 70 TB of memory for this relatively small instance. However, no large-scale implementations of the van Oorschot-Wiener attack have been attempted to date there are still several unanswered questions regarding the concrete performance and parallelism scalability of these attacks that would help shed light on the appropriate choice of parameters for other still relevant schemes. These open problems are tackled in the work presented in

Chapter 6.

3.2 Isogenies over prime fields

The special case of restricting all elliptic curves and isogenies to be defined over a prime field $\mathbb{F}_p$ has also proven to be of great cryptographic interest. As explained in Chapter 2, the ring of endomorphisms of a curve is generally an order in a quaternion algebra for supersingular curves or an order in an imaginary quadratic field for ordinary curves. However, it was shown by Delfs and Galbraith [45] that if endomorphisms are restricted to $\mathbb{F}_p$ only then the endomorphism ring is always contained in an imaginary quadratic field, even for supersingular curves. This allows us to obtain a commutative ideal class group $\mathrm{Cl}(\mathcal{O})$, and to combine the mathematical usefulness of a commutative class group action with the cryptographical security and efficiency of supersingular curves.

The common scenario is to work over a field $\mathbb{F}_p$ with a prime of the form

$$p = f \prod_i \ell_i - 1,$$

where f is a co-factor and ℓ_i are distinct odd primes co-prime to f. Supersingular curves over the prime field will always have a Frobenius trace $t = 0$, and a cardinality of $\#E(\mathbb{F}_p) = p + 1 - t = f \prod_i \ell_i$, so there will be exactly one subgroup of order ℓ_i for each i. We use ϕ_i to denote the isogeny whose kernel is the subgroup of order ℓ_i, and $\mathfrak{l}_i$ to denote an ideal from the ideal class that corresponds to this isogeny under the Deuring correspondence. The $\mathfrak{l}_i$ are used as a spanning set for $\mathrm{Cl}(\mathcal{O})$, and an arbitrary ideal class $\mathfrak{a} = \mathfrak{l}_0^{a_0}\mathfrak{l}_1^{a_1}\cdots\mathfrak{l}_n^{a_n}$ can be represented by the integer vector $\vec{a} = (a_0, \ldots, a_n)$. Note that it is not clear that the $\mathfrak{l}_i$ necessarily generate all of $\mathrm{Cl}(\mathcal{O})$, but empirically it has been shown that even a single $\mathfrak{l}_i$ will usually span the entirety of it [46]. Additionally, we can also use negative exponents to denote powers of the inverse ideal $\mathfrak{l}^{-1}$, which is well-defined since $\mathrm{Cl}(\mathcal{O})$ is a group and corresponds to the isogeny that is the dual of ϕ_i. Since there is only one subgroup of order ℓ_i in $E(\mathbb{F}_p)$, the dual must necessarily be defined outside of $\mathbb{F}_p$, but it has been shown in [3] that it can still be computed efficiently using $\mathbb{F}_p$ arithmetic only.

One of the most important isogeny-based schemes still of relevance today is the **Commutative Supersingular Isogeny Diffie-Hellman** scheme (CSIDH) proposed by Castryk *et al.*in 2018 [3]. The scheme has Alice and Bob pick random integer vectors $\vec{a}, \vec{b}$ which act as their private keys and encode ideals $\mathfrak{a}, \mathfrak{b}$. They then compute the action of their ideals on a base curve E_0 to obtain $E_A = \mathfrak{a} * E_0$ and $E_B = \mathfrak{b} * E_0$, which act as their public keys. Alice then computes the action of her secret ideal $\mathfrak{a}$ on Bob's public curve E_B to obtain $\mathfrak{a} * E_B = \mathfrak{a} \cdot \mathfrak{b} * E_0$, while Bob does the analogue and arrives to the same curve thanks to the commutativity of the group action. This commutativity allows the group action to act as a drop-in substitute for the operation in the original Diffie-Hellman protocol, but the fact that the public information is the action of a group element on a

set rather than a group element itself makes it resistant to Shor's algorithm. Thanks to this, CSIDH is currently the only known non-interactive key exchange protocol[1] that is post-quantum.

Commutative class group actions have also been used in signature schemes such as CSI-FiSh [46] and SeaSign seasign, but they suffer from poor performance relative to other alternatives.

Cryptanalysis. Despite the usefulness of commutative group actions, restricting only to curves over $\mathbb{F}_p$ means that we are not dealing with the most general case of the supersingular isogeny problem, and the commutative group structure can be just as useful in providing new means for attacks.

In general, the problem of finding an isogeny in the setting of commutative group actions can be reduced to a problem known as the *Abelian Hidden Shift Problem*, for which Kuperberg has presented a quantum attack that takes sub-exponential time in 2005 [47], with a small improvement to the memory-time trade-off presented later in 2013 [48]. While not prohibitively devastating, a sub-exponential attack means that a quantum attacker can achieve a much better complexity than a classical one and so, unlike the general case of isogenies over quadratic fields, the quantum security of these protocols needs to be scrutinized by analyzing the quantum circuit for Kuperberg's attack closely.

For instance, the smallest instance of the original CSIDH proposal used a 512-bit prime where the number of ℓ_i factors is $n = 74$ and the private keys had the form $\vec{a} = (a_1, a_2, \ldots, a_{74})$ where each exponent a_i is chosen from the interval $[-5, 5]$, hence yielding a keyspace of $11^{74} \approx 2^{256}$ and a classical security of 128 bits. However, whether or not this instance is also big enough to be considered secure against a quantum attack has proven to be a topic of great controversy. In 2018, Bernstein *et al.*[9] estimated that a quantum circuit that evaluates the CSIDH-512 action would require around 2^{40} T-gates, and argued that the advantage of Kuperberg's algorithm would be outweighed by this costly evaluation cost. However, in 2020 Peikert [49] estimated that a full execution of Kuperberg's algorithm only requires about 2^{16} queries to the evaluation oracle, which combined with Bernstein's result means that CSIDH-512 could have as little as 56 bits of quantum security. Also in 2020, Bonnetain and Schrottenloher [50] found that a full simulation of the attack against CSIDH-512 took 2^{19} queries and at $2^{52.6}$ T-gates per query, for a total security of 71.6 bits. While this result is more optimistic, it is still insufficient since a similar study by the same team found that AES-128 (to which CSIDH-512 and other postquantum candidates are supposed to be comparable to in security) has at least 113 bits of quantum security [51].

Still, this may not tell the whole story since parameters such as the circuit depth are not being restricted, and only the operations in the calls to the CSIDH evaluation oracle are

[1] That is, Alice can compute the shared secret after learning Bob's public key without previously exchanging any auxiliary information with him, and vice-versa.

being counted (there is another large part of Kuperberg's algorithm known as the *sieving phase* which progressively collapses the superposed state into a basic state containing the answer, that has not been taken into consideration in these works). All of these were open issues that were tackled in the first contribution of this project, which is detailed in Chapter 5.

Chapter 4

Overview of Results

These contributions of this project were obtained though four main works, which will be summarized in this chapter. The four works are developed in full in the next four chapters.

4.1 The SQALE of CSIDH: scaling up CSIDH to quantum-secure parameters

The acronym SQALE is used for "***S**ublinear Vélu **Q**uantum-resistant isogeny **A**ction with **L**ow **E**xponents*", and is also an allegory to the fact that this work's main goal is to scale the parameters of CSIDH up to a quantum-secure state.

One of the main theoretical contributions of this work is the argument that bloating the size of the prime field alone should be enough to increase the quantum security, even if the key space of usable isogenies is not increased at all. This is because the main quantum threat to CSIDH, Kuperberg's attack [47], requires working over a set with group structure and even if only a small subset of $\mathrm{Cl}(\mathcal{O})$ is used as keys, the group that they generate will almost surely be all of $\mathrm{Cl}(\mathcal{O})$.

Based on this, we proposed 5 new instances of CSIDH using primes of size 4096 or 5120 targeting NIST security level 1 (compared to the original 512 bits), 6144 to target NIST security level 2 (compared to the original 1024 bits), and 8192 or 9216 to target NIST security level 3 (compared to the original 1792 bits). We also provided quantum circuit estimations for the sieving phase of Kuperberg's attack, in order to back up the the quantum resistance claim of our new instances. We argue that this sieving phase in fact becomes asymptotically more costly than the CSIDH query calls, even though the later had been the only part of the attack that previous authors [49, 50] considered.

One of the main disadvantage of CSIDH relative to other post-quantum candidates was its slow performance even with its current parameters, and so the adopting of larger parameters should be taken with great reluctance. To better understand how negative the

effects can be, we presented full constant-time C implementations for each of our proposed new instances. We benchmarked all of our implementations, which used two important optimizations to ameliorate the damage of using larger primes:

- **Low Exponents** Recall that the private keys in CSIDH are vectors of exponents a_i which indicate how many times the isogeny of each degree ℓ_i is repeated. Having a larger prime allows us to increase the number of ℓ_i factors in the factorization of $p+1$ and the pool of isogenies, thus allowing us to reach the same key space using only exponents $a_i \in \{-1, 0, +1\}$. This ensures that each isogeny degree will be used at most once, and avoids having to constantly look for new points of order ℓ_i.
- **Sublinear Vélu formulas** In 2020, Bernstein *et al.* [25] presented a new algorithm that evaluates the Vélu formulas for an isogeny of degree ℓ in time $\mathcal{O}(\sqrt{\ell})$ as opposed to the straightforward implementation which took $\mathcal{O}(\ell)$ and was the previous state-of-the-art. We provided the first constant-time C implementation of Bernstein's algorithm, and integrated it to our implementations. The quadratic speedup proved specially beneficial to our scaled-up instances, since they make use of much larger primes ℓ_i.

Despite these improvements, it was concluded that there is still a long way to go before CSIDH can be considered practical, as even the fastest version of CSIDH-4096 was found to run in over 23 billion clock cycles.

4.2 Parallel Isogeny Path Finding: classical cryptanalysis of isogeny-based cryptography

In this work we analyze the security of isogeny-based cryptography in the more general scenario of isogenies over quadratic fields. We work on the special but common case that the isogeny degree is known to the attacker, but do not assume any additional information such as the one leaked in SIDH. The cryptanalytic efforts for this scenario usually focus on classical attacks only, since there are no known quantum attacks that are more asymptotically efficient than the classical ones. Specifically, the fastest known attack is an adaptation of the meet-in-the-middle attack (MitM), but it has intensive memory requirements and the golden collision search (GCS) of van Oorschot and Wienner [42] is probably more viable as a time-memory trade-off. A proper choice of parameters for cryptographic protocols requires determining which attack is optimal under given conditions and predicting the concrete performance of said attack, but so far these questions had not been widely studied in practice and the analysis remained theoretical.

We have aimed to provide concrete cryptanalytic results by producing large-scale parallel implementations of both the MitM and GCS attacks which were made freely available. The MitM implementation is CPU-based and is similar to the one of Udovenko and

Vitto [44], while the GCS implementation is the first ever large-scale GPU-based implementation of this attack. Having both implementations readily available, we were able to measure the performances for varying isogeny degrees and determine the crossing point where the GCS starts being more efficient than the MitM under a limited-memory setting. This was found to be at a degree of around 2^{96} for reasonable memory bounds, which is well bellow common cryptographic sizes, and so we concluded that the GCS should be considered the attack of interest for all cryptanalytic applications.

Additionally, we developed a detailed theoretical analysis for the cost of either attack in order to derived explicit formulas for their concrete cost. These formulas were verified by comparing their prediction with the actual cost of our implementations, which allows us to confidently extrapolate the model to larger instances which were unfeasible to solve. The extrapolation predicts that an isogeny of degree 2^{190} should already provide 128−bit security. This bound is considerably more aggressive compared to the choices of previous protocols, such as SIKE [15] which used a degree of 2^{216} aiming for the same security level.

4.3 Verifiable Isogeny Walks: Towards a Post-Quantum Isogeny-based VDF

This work focused on using the isogeny framework to obtain a post-quantum instance of a verifiable delay function (VDF), which is a relatively new cryptographic primitive that has gathered great interest in areas such as blockchain, proofs of work, and verifiable computation.

The goal of a VDF is to provide a function that can only be evaluated via a very slow algorithm, and which cannot be accelerated substantially even by using parallelization. However, once the output has been obtained and published by someone, it should be easily verifiable by anyone in a time that is exponentially faster than having to redo the computation. Before the publication of this work, no VDFs had ever been proposed that were quantum-secure in the sense of explicitly relying on one of the commonly studied post-quantum problems.

The construction proposed by this work gives an explicit algorithm for performing a random isogeny walk which can be easily plugged into a SNARG framework. SNARGs, or *Succinct Non-interactive Arguments*, are constructions of great interest which allow for the verification of any computation in a time exponentially faster than the computation itself. Despite having the necessary asymptotic complexities, SNARGs are still considered of mainly theoretical interest since they come with immense overheads that make them inviable for most practical applications. Nonetheless, our construction is still of great importance since it is the first post-quantum construction with the right asymptotic complexities.

Additionally, we also we show that the SNARG can be adapted to verify computations

at the field arithmetic level, which should save significantly on overhead compared to universal SNARGs which are used to verify arbitrary algorithms at the bit-operation level. However, we have not provided any implementation nor estimate as to how much this explicitly helps, which is left for consideration as future work.

4.4 SwiftEC: An efficient encoding for ordinary elliptic curves

This work tackled the problem of finding an efficient encoding for ordinary elliptic curves, which means being able to represent points in an elliptic curve as uniformly random bit strings. This has important applications in protecting against censorship since raw elliptic curve points are easily identifiable and can be filtered out to forbid communications with elliptic curve-based protocols. Additionally, under the right conditions, the inverse of the encoding can be seen as a hash function that turns random bit strings into random elliptic curve points, which is useful for constructing many other cryptographic primitives.

In practice, an encoding is usually defined by first finding a map $f : \mathbb{F} \to E(\mathbb{F})$ and then setting the encoding function to f^{-1}. Finding an efficient f can be hard, especially if it is required to run in constant time where the cost is dominated by the number of field exponentiations. In 2006, Shallue and van de Woestijne [52] proposed a parametrized family of functions f_u which, for any choice of $u \in \mathbb{F}$, mapped to $E(\mathbb{F})$ in constant time using only one exponentiation. However, the image for a given u does not cover all of $E(\mathbb{F})$ and so not only can it not be considered a cryptographic hash function but it cannot even provide an encoding for all of the points.

Tibouchi's Elligator Squared paper [53] in 2014 showed that one could fix both of these issues by instead considering the two-parameter encoding

$$F : (t_1, t_2) \mapsto f_u(t_1) + f_u(t_2),$$

with the addition being performed in the elliptic curve group. It showed that the output was surjective, asymptotically indistinguishable from a random distribution, and that it had all the right conditions for a proper encoding. Having to perform two evaluations of f_u unfortunately brings the cost up to two exponentiations, but this was still the most optimal encoding of its kind known so far.

In this work, we propose a new encoding called SwiftEC, which brings the cost back down to one exponentiation while retaining the indistinguishability property. In a nutshell, the work points out that if u is viewed as a free parameter then the two-parameter function

$$F : (u, t) \mapsto f_u(t)$$

already fills the whole curve and its output distribution is indistinguishable from random. This argument is accompanied by robust mathematical proofs regarding the output distribution, and a proof-of-concept implementation was written in Sage and benchmarked

to count the number of field operations needed. Viewed either as an encoding or as a hash function, the single exponentiation makes our construction the most efficient constant-time construction known to date for a large set of ordinary elliptic curves that it is compatible with.

Since it works over ordinary curves, the construction is not interesting for post-quantum applications. However, it is closely related to our work in that it still utilizes the isogeny framework, and it produces a result that is of great immediate interest for classical applications being used today.

Chapter 5

The SQALE of CSIDH: Scaling up CSIDH to quantum-secure parameters

Abstract. *Recent analyses reported independently by Bonnetain-Schrottenloher and Peikert in Eurocrypt 2020, significantly reduce the estimated quantum security provided by the isogeny-based commutative group action protocol CSIDH. The work in this chapter, published in collaboration with Sam Jacques, Jesus Chi and Francisco Rodríguez in the Journal of Cryptographic Engineering [4], revisits CSIDH quantum security through a comprehensive analysis of the computational cost associated to the quantum collimation sieve attack. Furthermore, it proposes a set of primes that can be applied to obtain large instantiations of CSIDH achieving the NIST security levels 1, 2, and 3. Along with this work, C-code constant-time implementation of those CSIDH large instantiations was provided, which is supported by the new sublinear Vélu formulae.*

Based on supersingular elliptic curve isogenies defined over a prime field $\mathbb{F}_p$, the commutative isogeny-based key exchange protocol CSIDH is a promising isogeny-based protocol that has received considerable attention since its proposal in Asiacrypt 2018 by Castryck, Lange, Martindale, Panny and Renes [3].

CSIDH can be used analogously to the Diffie-Hellman protocol to produce a non-interactive key exchange scheme between two parties. Moreover, CSIDH can be adapted as the underlying cryptographic primitive for more elaborate applications such as key encapsulation mechanisms, signatures and other primitives. It has remarkably small public keys (in fact, even with the parameter scaling proposed in this chapter it still has shorter keys than the four public key encryption round-3 finalists of the NIST post-quantum standardization process [14] [1]), and allows a highly efficient key validation procedure. This latter feature aids in making CSIDH better suited than most (if not all) post-quantum schemes

[1] The SIKE protocol, which is also isogeny-based, does have shorter keys than our scaled version of CSIDH, but is classified as an "alternate candidate"

for resisting Chosen Ciphertext Attacks (CCA) and for supporting static-dynamic and static-static key exchange settings. On the downside, CSIDH has a significantly higher latency than other isogeny-based protocols such as SIDH and SIKE [15, 54]. Furthermore, as this chapter will discuss in detail, several recent analyses revised CSIDH's true quantum security downwards (see for example [50, 49]).

CSIDH's Security From a classical perspective, the security of CSIDH is related to the problem of finding an isogeny path from the isogenous supersingular elliptic curves E_0 and E_A. Now, random-walk-based attacks on the whole class group (of rough size $\sqrt{p}$) have a complexity of $\tilde{O}(\sqrt[4]{p})$ steps with constant space (for more details see, [45]). Thus, in order to provide a security level of 128 classical bits, the prime p needs to be large enough to support 2^{256} ideal classes, hence the choice of a 512-bit prime in the original CSIDH proposal. The parameter m should then be chosen in such a way that the private key space is also composed of 2^{256} different secret keys, which we heuristically expect to fill nearly all ideal classes.

From a quantum attack perspective, Childs, Jao, and Soukharev tackled in [26] the problem of recovering the secret $\mathfrak{a}$ from the relation $E_A = \mathfrak{a} * E_0$. They managed to reduce this computational task to the abelian hidden-shift problem on the class group, where the hidden shift corresponds to the secret $\mathfrak{a}$ that one wants to find. Previously in 2003 and 2004, Kuperberg and Regev had presented two sieving algorithms that could solve this problem in subexponential time if they were executed in a quantum setting [47, 55]. In particular, Kuperberg's procedure has a quantum time and space complexity of just $\exp\left(O(\sqrt{\log p})\right)$. Later, in 2011, Kuperberg refined his algorithm by adding a *collimation sieving* phase [48]. The time complexity of this new variant was still $\exp\left(O(\sqrt{\log p})\right)$, but the quantum space complexity was just $O(\log p)$.

In a nutshell, a Kuperberg-like approach for solving the hidden-shift problem consists of two main components:

1. A quantum oracle that evaluates the group action on a uniform superposition and produces random *phase vectors*

2. A sieving procedure that destructively combines low-quality phase vectors into high-quality phase vectors

The sieving procedure gradually improves the quality of the phase vectors until they can be measured and reveal some bits of the hidden shift, and thus the CSIDH secret key.

Recent analyses of this quantum algorithm that were presented in Eurocrypt 2020 [50, 49], point to a significant reduction of the quantum security provided by CSIDH. Concretely, the original 511-bit prime CSIDH instantiation was deemed to achieve NIST security level 1 in [3]. However, the authors of [50] recommended that the size of the CSIDH prime p should be upgraded to at least 2260 or 5280 bits, according to what they named as *aggressive* and *conservative* modes, respectively.

Both [50] and [49] focus on breaking the originally proposed instantiations of CSIDH, rather than an exhaustive analysis of the quantum attack. [50] focuses mainly on Kupberberg's first attack and Regev's attack by providing a thorough accounting of a quantum group action circuit. [49] gives a thorough practical and theoretical analysis of Kuperberg's second algorithm and provides many optimizations. While [49] simulates the full algorithm to give very precise estimates, this method will not extend to the larger primes we consider here because, by design, even the classical aspects of the attack should be infeasible to compute. We use the results of the theoretical analysis in [49] to count resource use without a full simulation. This allows us to evaluate very large primes and to explore depth-width trade-offs and thus to compare to NIST's security levels. We argue that for the primes we consider, CSIDH's quantum security depends mainly on the cost of the collimation sieve, not the current isogeny evaluation costs.

The SQALE of CSIDH We use the acronym SQALE for "***S**ublinear Vélu **Q**uantum-resistant isogeny **A**ction with **L**ow **E**xponents*". The SQALE of CSIDH is a CSIDH instance such that $p = 4 \cdot \prod_{i=1}^{n} \ell_i - 1$ is a prime number with small odd primes $\ell_1, \ldots, \ell_n$, and the key space size $N \ll \sqrt{p}$ is determined by using only the $k \leq n$ smallest ℓ_i's, where the exponents e_i of the ideal class $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$, are drawn from a small range, possibly $\{-1, 0, 1\}$.

The original CSIDH protocol chose exponents large enough that the key space is approximately equal to the class group. We show in Section 5.1 that a SQALE'd CSIDH preserves classical security. We also argue in Section 5.3 that quantum attackers need to attack the entire class group, regardless of the subset that keys are drawn from, so we can choose low exponents and preserve quantum security as well. With this change, we improve the trade-off between the performance of the key exchange and its quantum security. To further improve performance of the large CSIDH instances considered in this work, we incorporate the Vélu's improved $\mathrm{O}(\sqrt{\ell})$ algorithm for isogeny computations.

Contributions In this work we present a detailed classical and quantum cryptanalysis of CSIDH using our revised prime sizes as shown in Table 5.1, which, according to our analysis, are required to achieve the NIST security levels 1, 2 and 3. We also present a constant-time C implementation of CSIDH for several large instantiations of CSIDH as reported in Table 5.6.

Our concrete contributions can be summarized as follows:

1. A concrete computational cost analysis of the CSIDH group action for different sizes of primes p. Our study takes into account different options for the exponentinterval m going all the way from the minimal setting $[\![-1 \,..\, 1]\!]$ (including and not including the zero exponent), and the more common choice $[\![-5 \,..\, 5]\!]$. In particular, for each interval $[\![-m \,..\, m]\!]$ we directly apply the framework described in [56], by selecting

Table 5.1: Summary of results. Quantum security is depth×width, including a hardware limit of 2^{80} for Level 1, 2^{100} for Level 2, and 2^{119} for Level 3, as well as a 2^{10} overhead for error correction, and assuming a quantum oracle free of cost. Performance based on the CSIDH variant OAYT-style (cf. Subsection 5.1.2).

NIST Security level	CSIDH quantum security in bits	CSIDH prime size in bits	Performance (gigacycles)
Level 1	124	4,096	23.2
Level 1	135	5,120	42.2
Level 2	148	6,144	74.8
Level 3	>160	8,192	199.1
Level 3	>171	9,216	292.4

optimal bound vectors, (*i.e*, a customized m_i per each e_i along with their respective optimal strategies.

2. The first C-code implementation of a constant-time version of the CSIDH protocol using the $O(\sqrt{\ell})$ algorithm for constructing and evaluating isogenies as proposed in [25].

3. Extending the cost analysis of the quantum collimation sieve to account for larger primes, depth limits, improved quantum memory circuits, and several small optimizations. We also argue that a small exponent interval maintains quantum security while improving classical performance.

Outline Section 5.1 gives background on CSIDH, efficient methods for computing its group action, and the quantum cost models we use. In Section 5.2 we describe the quantum collimation sieve attack and explain how to estimate its cost. We account for larger primes, depth limits, improved memory circuits, and find several small optimizations. The sieve only seems able to attack the full class group, and not any smaller generating subset. We give several arguments for this in Section 5.3, ultimately concluding that for a quantum attacker, only the size of the class group affects the total quantum attack cost. These conclusions suggests that an ideal scheme will operate on isogenies of a number of degrees, but with small exponents for each. Section 5.4 summarizes the quantum and classical security and the effects of hardware limits.

We then give a concrete cost analysis of the CSIDH group action for a key exchange with different sizes of primes p in Section 5.5. We account for different options of the exponent interval m, from the minimal setting $[\![-1 \,..\, 1]\!]$ (with or without zero) up to the original proposal of $[\![-5 \,..\, 5]\!]$. For each interval, we apply the framework reported in [56] to select optimal bounds (different m_i for each prime) and their corresponding optimal

strategies. Starting from the Python-3 CSIDH library reported in [57], we present the first constant-time implementation of large CSIDH instantiations supporting the $O(\sqrt{\ell})$ isogeny-evaluation algorithm from [25]. Our C library also includes a companion script that estimates quantum attack costs. Our software is freely available at https://github.com/JJChiDguez/sqale-csidh-velusqrt.

5.1 Background

This section presents some of the main concepts required for performing classical and quantum attacks on CSIDH.

5.1.1 Construction and evaluation of odd degree isogenies using Vélu Square-root Algorithm

Let ℓ be an odd prime number, $\mathbb{F}_p$ a finite field of large characteristic, and A a Montgomery coefficient of an elliptic curve $E_A/\mathbb{F}_p\colon y^2 = x^3 + Ax^2 + x$. Given an order-$\ell$ point $P \in E_A(\mathbb{F}_p)$, the construction of an isogeny $\phi : E_A \mapsto E_{A'}$ of kernel $\langle P\rangle$ and its evaluation at a point $Q = (\alpha, \beta) \in E_A(\mathbb{F}_p)\backslash\langle P\rangle$, consist of the computation of the Montgomery coefficient $A' \in \mathbb{F}_p$ of the co-domain curve $E_{A'}/\mathbb{F}_p\colon y^2 = x^3 + A'x^2 + x$ and the x-coordinate $\phi_x(\alpha)$ of $\phi(Q)$.

Using the recent Vélu square-root algorithm (aka $\sqrt{\text{élu}}$) as presented by Bernstein, De Feo, Leroux and Smith in [25], A' and $\phi_x(\alpha)$ can be computed as (see also [58], [59], [60] and [57]),

$$A' = 2\,\frac{1+d}{1-d} \quad \text{and} \quad \phi_x(\alpha) = \alpha^{\ell}\,\frac{h_S(1/\alpha)^2}{h_S(\alpha)^2},$$

$$\text{where } d = \left(\frac{A-2}{A+2}\right)^{\ell}\left(\frac{h_S(1)}{h_S(-1)}\right)^{8},$$

$$S = \{1, 3, \ldots, \ell-2\}, \text{ and}$$

$$h_S(X) = \prod_{n\in S}(X - x([n]P)).$$

Hence, the main cost associated to computing A' and $\phi_x(\alpha)$, corresponds to the computation of $h_S(X)$. Given $E_A/\mathbb{F}_p$ an order-ℓ point $P \in E_A(\mathbb{F}_p)$, and some value $\alpha \in \mathbb{F}_p$ we want to efficiently evaluate the polynomial,

$$h_S(\alpha) = \prod_{i}^{\ell-1}(\alpha - x([i]P)).$$

From Lemma 4.3 of [25],

$$(X - x(P+Q))(X - x(P-Q)) = X^2 + \frac{F_1(x(P), x(Q))}{F_0(x(P), x(Q))}X + \frac{F_2(x(P), x(Q))}{F_0(x(P), x(Q))}$$

where,

$$\begin{aligned} F_0(Z, X) &= Z^2 - 2XZ + X^2; \\ F_1(Z, X) &= -2(XZ^2 + (X^2 + 2AX + 1)Z + X); \\ F_2(Z, X) &= X^2Z^2 - 2XZ + 1. \end{aligned}$$

This suggests a rearrangement à la Baby-step Giant-step as,

$$h_S(\alpha) = \prod_{i \in \mathcal{I}} \prod_{j \in \mathcal{J}} (\alpha - x([i + s \cdot j]P))(\alpha - x([i - s \cdot j]P)),$$

where s is a fixed integer representing the size of the giant steps and $\mathcal{I}$, $\mathcal{J}$ are two sets of indices such that $\mathcal{I} \pm s\mathcal{J}$ covers S.

Now $h_S(\alpha)$ can be efficiently computed by calculating the resultants of two polynomials in $\mathbb{F}_p[Z]$, of the form

$$\begin{aligned} h_I(Z) &:= \prod_{x_i \in \mathcal{I}} (Z - x_i) \\ E_{J,\alpha}(Z) &:= \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j)\alpha^2 + F_1(Z, x_j)\alpha + F_2(Z, x_j)\right) \end{aligned}$$

The most demanding operations of √élu require computing four different resultants $\mathtt{Res}_Z(f(Z), g(Z))$ of two polynomials $f, g \in \mathbb{F}_p[Z]$. Those four resultants are computed using a remainder tree approach supported by carefully tailored Karatsuba polynomial multiplications. In practice, the computational cost of computing degree-ℓ isogenies using √élu is close to $K(\sqrt{\ell})^{\log_2 3}$ field operations for a constant K. For more details about these computations see [25, 57].

5.1.2 Summary of CSIDH

Here, we give a general description of CSIDH. A more detailed description of the CSIDH group action computation can be found in [3, 61, 62, 63].

The most demanding computational task of CSIDH is evaluating its class group action, whose cost is dominated by performing a number of degree-ℓ_i isogeny constructions. Roughly speaking, three major variants for computing the CSIDH group action have been proposed, which we briefly outline next.

Let $\pi\colon (x,y) \mapsto (x^p, y^p)$ be the Frobenius map and $N \in \mathbb{Z}$ be a positive integer. Working now with points over the extension field $\mathbb{F}_{p^2}$, let $E[N]$ denote the N-torsion subgroup of $E/\mathbb{F}_{p^2}$ defined as, $E[N] = \{P \in E(\mathbb{F}_{p^2})\colon [N]P = \mathcal{O}\}$. Let also

$$E[\pi - 1] = \{P \in E(\mathbb{F}_{p^2})\colon \pi P = P\}$$

and

$$E[\pi + 1] = \{P \in E(\mathbb{F}_{p^2})\colon \pi P = -P\}.$$

Note that $E[\pi - 1]$ corresponds to the original set of $\mathbb{F}_p$-rational points, whereas $E[\pi + 1]$ is a set of points of the form (x, iy) where $x, y \in \mathbb{F}_p$ and $i = \sqrt{-1}$ so that $i^p = -i$. We call the later the set of *zero-trace points*.

The MCR-style [62] of evaluating the CSIDH group action takes as input a secret integer vector $e = (e_1, \ldots, e_n)$ such that $e_i \in [\![0 \,..\, m]\!]$. From this input, isogenies with kernel generated by $P \in E_A[\ell_i] \cap E_A[\pi - 1]$ are constructed for exactly e_i iterations. In the case of the OAYT-style [63], the exponents are drawn from $e_i \in [\![-m \,..\, m]\!]$, and P lies either on $E_A[\ell_i] \cap E_A[\pi - 1]$ or $E_A[\ell_i] \cap E_A[\pi + 1]$ (the sign of e_i determines which one will be used). We stress that for constant-time implementation of CSIDH adopting the MCR and OAYT styles, the group action evaluation starts by constructing isogenies with kernel generated by $P \in E_A[\ell_i] \cap E_A[\pi - \text{sign}\ (e_i)]$ for e_i iterations, followed by dummy isogeny constructions that are performed for the remaining $(m - e_i)$ iterations.

On the other hand, the dummy-free constant-time CSIDH group action evaluation, proposed in [61], takes as secret integer vector $e = (e_1, \ldots, e_n)$ such that $e_i \in [\![-m \,..\, m]\!]$ has the same parity as m. Then, one starts constructing isogenies with kernel generated by $P \in E_A[\ell_i] \cap E[\pi - \text{sign}\ (e_i)]$ for exactly e_i iterations. Thereafter, one alternatingly computes $E_A[\ell_i] \cap E_A[\pi - 1]$ and $E_A[\ell_i] \cap E_A[\pi + 1]$ isogenies for the remaining $m_i - e_i$ iterations (for more details see [61]).

5.1.3 Quantum computing

We refer to [64] for the basics and notation of quantum computing. Following [65], we treat a quantum computer as a memory peripheral of a classical computer, which can modify the quantum state with certain operations called "gates". We give the cost of a quantum algorithm in terms of these operations (specifically Clifford + T gates), which we treat as a classical computation cost. With this we can directly add and compare quantum and classical costs, since we measure quantum computation costs in classical operations. We use the "*DW*"-cost, which assumes that the controller must actively correct all the qubits at every time step to prevent decoherence. This means the total cost is proportional to the total number of qubits (the "width"), times the total circuit depth.

We depart from [65] by giving an overhead of 2^{10} classical operations for each unit of *DW*-cost, to represent the overhead of quantum error correction. With surface code error correction, every logical qubit is formed of many physical qubits, which continuosly run

through measurement cycles. We assume each cycle of each physical qubit is equivalent to a classical operation. By this metric, Shor's algorithm has an overhead of 2^{17} for each logical gate [66]. The algorithm we analyze will need much more error correction, but we assume continuing advances in quantum error correction will reduce this overhead to 2^{10}. Since a surface code needs to maintain a distance between logical qubits in two physical dimensions and one dimension of time [67], we assume the 2^{10} overhead is the cube of the code distance, and thus every logical qubit is composed of $2^{10 \cdot \frac{2}{3}}$ physical qubits.

5.2 Quantum Attack

We follow Peikert [49] and analyze only Kuperberg's second algorithm [48]. Because of this, and our assumption that classical operations are only 2^{10} times cheaper than quantum, the trade-offs of [68, 69] do not help for our analysis.

Kuperberg's algorithm can be divided into 3 stages:

1. Constructing phase states, where we compute an arbitrary isogeny action in superposition, perform a quantum Fourier transform, then measure the result. This leaves a single qubit in a random *phase state* with some associated classical data, which forms the input to the next stage.

2. A sieving stage, where we use a process called "collimation" to destructively combine phase states to produce "better" phase states. This requires some quantum arithmetic, but the main costs are quantum access, in superposition, to a large table of classical memory, and subsequent classical computations on this table.

3. A measurement stage, where we measure a sufficiently "good" phase state and recover some number of bits of the secret key.

We repeat these steps until we recover enough bits of the secret key to exhaustively search the remainder.

Asymptotically, the sieving stage is the most costly, so we focus on that. In Section 5.2.6 we justify our choice to ignore the cost of constructing phase states.

5.2.1 Overview of Kuperberg's algorithm

We start with an abelian group G (the class group) of order N and two injective functions $f : G \to X$ and $h : G \to X$ such that $h(x) = f(x - S)$ for some secret S. For this description we assume G is cyclic. This is generally untrue for class groups, but a quantum attacker can recover the group structure as a polynomial-cost precomputation (see [50, Section 4]). They can then decompose the group into cyclic subgroups, perform a quantum Fourier transform on each, and collimate them independently. The total amount of collimation will be the same, so we focus on a cyclic group as it is easier to describe.

For CSIDH, the function f will identify an element of the class group with an isogeny from E_A to some other curve E, and output the j-invariant of that curve. The function h is the same, but starts with a public key curve $E_{A'}$.

To begin, we generate a superposition over G (ignoring normalization), $\sum_{g\in G} |g\rangle$. Then we initialize a single qubit in the state $|+\rangle = |0\rangle + |1\rangle$, and use it to control applying either f or h:

$$\sum_{g\in G} |0\rangle\,|g\rangle\,|f(g)\rangle + |1\rangle\,|g\rangle\,|h(g)\rangle \tag{5.1}$$

Then we measure the final register, finding $f(g) = h(g+S)$ for some g. Because f and h are injective, this leaves only two states in superposition:

$$|0\rangle\,|g\rangle + |1\rangle\,|g+S\rangle\,. \tag{5.2}$$

This is the ideal state. Naive representations of the group will not produce precisely this state. Section 5.3.1 explains why our best option is to fix a generator g, and produce superpositions $\sum_{x=0}^{N-1} |x\rangle\,|xg\rangle$, which leads to a final state

$$|0\rangle\,|x\rangle + |1\rangle\,|x+s\rangle \tag{5.3}$$

where $S = sg$. At this point, we apply a quantum Fourier transform (QFT), modulo the group order N, to produce

$$|0\rangle \sum_{k=0}^{N-1} e^{2\pi i \frac{xk}{N}}\,|k\rangle + |1\rangle \sum_{j=0}^{N-1} e^{2\pi i \frac{(x+s)j}{N}}\,|j\rangle\,. \tag{5.4}$$

Then we measure the final register and find some value b, leaving us with the state

$$|0\rangle\, e^{2\pi i \frac{xb}{N}} + |1\rangle\, e^{2\pi i \frac{(x+s)b}{N}} \equiv |0\rangle + e^{2\pi i \frac{sb}{N}}\,|1\rangle\,. \tag{5.5}$$

From this point, we define $\zeta_s^b = e^{2\pi i \frac{bs}{N}}$. We emphasize that it is critical that the QFT acts as a homomorphism between the elements of the group and phases modulo N, even an approximate homomorphism as in [50].

A classical computer with knowledege of s can easily simulate input phase vectors, and the cost of the remainder of the algorithm is mainly classical. Peikert thus simulated the remaining steps of the algorithm for a precise security estimate [49]. We hope to choose parameters such that the remaining steps are infeasible, so we cannot classically simulate them. Instead we extrapolate Peikert's results to estimate the full cost, with some small algorithmic improvements we now describe.

Phase vectors with data. Kuperberg works with states of the form in Equation 5.5 to save quantum memory; however, we will maintain the factor b in quantum memory.

We define a phase vector with data to have a length L, a height S, an altitude A, and a phase function $B : [L] \to [S]A$ (defining $[N] := \{0, \ldots, N-1\}$ and $[N]M := \{0, M, 2M, \ldots, M(N-1)\}$), as follows:

$$\sum_{j=0}^{L-1} \zeta_s^{B(j)} |j\rangle |B(j)\rangle . \tag{5.6}$$

The phase function B is known classically.

The vector in Equation 5.5 almost has this form, with $L = 2$, $B(0) = 0$ and $B(1) = b$ (in fact $B(0) = 0$ for all phase vectors), and $S = b$. To add the data to it, we simply use the qubit to control a write of the value of b to a new register.

Starting from an initial phase vector with data, we can double its length with a new initial phase vector. We describe the procedure for a power-of-two length, which is much easier, but other lengths are possible with relabelling. We first concatenate the new phase vector, then treat the new qubit as the most significant bit of the index j:

$$\begin{aligned} &\left(|0\rangle + \zeta_s^{b'} |1\rangle\right) \otimes \left(\sum_{j=0}^{L-1} \zeta_s^{B(j)} |j\rangle |B(j)\rangle\right) \\ =&\sum_{j=0}^{L-1} \zeta_s^{B(j)} |j\rangle |B(j)\rangle + \sum_{j=L}^{2L-1} \zeta_s^{B(j-L)+b'} |j\rangle |B(j-L)\rangle . \end{aligned} \tag{5.7}$$

On the left sum, the first bit of j is 0, and on the right sum it is 1. We then redefine the phase function to be $B' : [2L] \to [S + b']$, where $B'(j) = B(j)$ if $j < L$ and $B'(j) = B(j - L) + b'$ if $j \geq L$. To update the phase register, we perform an addition of b', controlled on the first qubit (which is now the leading bit of the index j). The state is now twice as long, at the cost of just one quantum addition, and classical processing of the table of values representing B.

We can produce initial phase vectors with data of length $L = 2^\ell$ by starting with an initial phase vector, adding its phase function to a quantum register, then repeating this doubling process $\ell - 1$ times. The height of such a vector will be the maximum of ℓ uniformly random values from 0 to 2^n; we assume this is simply 2^n. The altitude will be the least common multiple of these vectors and we assume this is 1.

The next part of the algorithm is to *collimate* phase vectors until their height equals approximately their length. A collimation takes r phase vectors of some length L, height S, and altitude A, and destructively produces a new phase vector of length L', height S', and altitude A', where $S' < S$ and $A' \geq A$. For efficiency, we try to keep $L' = L$.

Once the height equals the length, say S_0, we perform a QFT and hopefully recover $\lg S_0$ bits of the secret s, starting from the bit at $\lg(A)$. To recover all of the secret bits, we run the same process but target different bits each time, sequentially or in parallel. Classical simulations show that each run recovers only $\lg S_0 - 2$ bits on average [49].

Adaptive Strategy . The length of the register in Equation 5.3, which undergoes to the QFT, governs the cost of the sieve. Ideally, after finishing one sieve, we would use the known bits of the secret to reduce the size of the problem. For example, if the group order is $N = 2^n$ for some n, then if the secret is $s = s_1 2^k + s_0$ and we know s_0, we start with a state $|0\rangle |x\rangle + |1\rangle |x + s \mod 2^n\rangle$ for some random, unknown x. We can subtract s_0 from the second register, controlled by the first qubit, to obtain

$$|0\rangle |x\rangle + |1\rangle |x + s_1 2^k \mod 2^n\rangle \tag{5.8}$$

The least significant k bits of the second register are the same in both states, so we can remove or measure these states, and only apply the QFT to the remaining bits. Then our initial phase vectors start with a height of 2^{n-k}, rather than 2^n.

This is Kuperberg's original technique. Peikert analyzed a non-adapative attack, using a high-bit collimation in case of non-smooth group orders. We remain uncertain whether an attack can be adaptive with a prime-order group. With prime orders, there is little correlation between the bits of x and $x + s \mod N$, even if we know most of the bits of s.

Alternatively, we could represent group elements by exponent vectors. In that case, we end up with the state

$$|0\rangle |\vec{x}\rangle + |1\rangle |\vec{x} + \vec{s} \mod L\rangle \tag{5.9}$$

where L is the lattice representing the kernel of the map from exponent vectors to class group elements. However, a direct, bit-wise QFT does not define a homomorphism from vectors modulo a lattice are to phases (see Section 5.3.1).

We could try to represent integer exponent vectors $\vec{x}$ by vectors $\vec{v}$ such that $B_L \vec{v} = \vec{x}$, where B_L is a matrix of the basis vectors of the lattice. We would find all bits of a single component, then clear that component for future sieves. Since $\vec{v} = B_L^{-1} \vec{x}$, and $B_L^{-1} = \frac{1}{\det(B_L)} \mathrm{adj}(B_L)$, and the adjugate of an integer matrix is an integer matrix, the smallest non-zero entry of B_L^{-1} in absolute value is at least $1/\det(B_L)$. This means one needs $\lg \det(B_L)$ bits of precision for each component $\vec{v}$. However, $\det(B_L) = \det(L) = N$, the size of the class group, so each component is as hard to solve as the entire problem under a generator-based representation, and we still cannot adaptively sieve *within* each component.

It is possible that adaptive sieving on a prime-order group is inherently difficult. There is a large gap between the classical difficulty of discrete log in a prime-order group compared to a smooth-order group, so a similar gap may exist in the highly similar abelian hidden shift problem. In summary, we assume that partial knowledge of the bits of a secret s in an abelian hidden shift problem gives no advantage in finding unknown bits for groups of prime order. More formally:

Assumption 5.2.1. *If it costs* C *to recover* t *secret bits in an abelian hidden shift problem for a group of prime order, it will still cost* $\max\{\mathsf{C}, O(2^{n-k})\}$ *to recover* t *bits even if* k *bits out of* n *are already known.*

Each run of the sieve recovers about $\lg S_0 - 2$ bits on average, so the total number of sieves is $\frac{\lg N}{\lg S_0 - 2}$. If this assumption is wrong, then in the worst case, the total sieving cost will be dominated by the first run of the sieve, leading to a reduction of ≈ 7 bits of security.

5.2.2 Collimation

From vectors of length L and height S, we repeatedly *collimate* to a height S' as follows: First we concatenate the vectors and add together their phase functions, which will match the new phase. Addition is done in-place on one of the phase registers. Let $\vec{j} = (j_1, j_2)$ so that $|j_1\rangle |j_2\rangle = |\vec{j}\rangle$, and let $B(\vec{j}) := B_1(j_1) + B_2(j_2)$. The resulting state will be:

$$\begin{aligned}&\sum_{j_1=0}^{L-1} \zeta_s^{B_1(j_1)} |j_1\rangle |B_1(j_1)\rangle \sum_{j_2=0}^{L-1} \zeta_s^{B_2(j_2)} |j_2\rangle |B_1(j_1) + B_2(j_2)\rangle \\ &= \sum_{j_1,j_2=0}^{L-1} \zeta_s^{B(\vec{j})} |\vec{j}\rangle |B_1(j_1)\rangle |B(\vec{j})\rangle . \end{aligned} \tag{5.10}$$

Then we divide $B(\vec{j})$ by S' and compute the remainder and modulus:

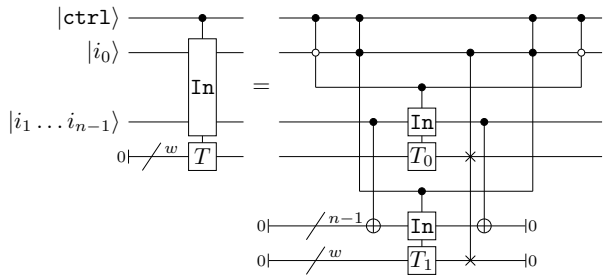
$$\sum_{j_1,j_2=0}^{L-1} \zeta_s^{B(\vec{j})} |\vec{j}\rangle |B_1(j_1)\rangle \left|\left\lfloor \tfrac{B(\vec{j})}{S'} \right\rfloor\right\rangle |B(\vec{j}) \mod S'\rangle . \tag{5.11}$$

We then measure the value of $\left\lfloor \frac{B(\vec{j})}{S'} \right\rfloor$, which gives some value K. Let $J \subseteq L \times L$ be the set of indices j_1 and j_2 such that $\left\lfloor \frac{B_1(j_1)+B_2(j_2)}{S'} \right\rfloor = K$. Since we know K, B_1, and B_2 classically, we can find find the set J and use it to construct a permutation $\pi : J \to [L']$, where $L' = |J|$. Defining a new phase function $B' : [L'] \to [S/S']$ where $B'(j) = B(\pi^{-1}(j)) \mod S'$, we find that $B(\vec{j}) = K + B'(\pi(\vec{j}))$ for all $\vec{j} \in J$. Equation 5.12 shows that the factor of K only introduces a global phase and thus we can ignore it.

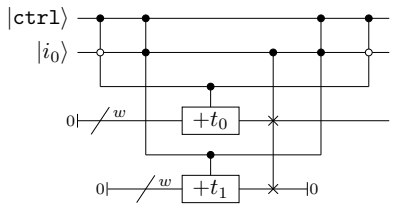
We now fix the phase vector that was left after measurement. First, we must erase $B_1(j_1)$. We use a quantum random access classical memory (QRACM) look-up uncomputation, which only needs to look up values of j_1 which are part of a pair in J. We expect L' such values.

Then we compute $\pi(\vec{j})$ in another register. This is a QRACM look-up from a table of L' indices with words of size $\lg L'$. Letting $j' = \pi(\vec{j})$, this leaves the state

$$\begin{aligned}&\sum_{\vec{j}\in J} \zeta_s^{B(\vec{j})} |\vec{j}\rangle |\pi(\vec{j})\rangle |B(\vec{j}) \mod S'\rangle \\ &= \sum_{j'=0}^{L'-1} \zeta_s^{K+B'(j')} |\pi^{-1}(j')\rangle |j'\rangle |B'(j')\rangle \end{aligned} \tag{5.12}$$



(a) Recursive case



(b) Base case

Figure 5.1: A short, wide look-up circuit for a table $T = [t_0, t_1, \dots]$, where T_0 and T_1 are two halves of T.

We now do a QRACM look-up uncomputation in a table of L' indices to erase $\pi^{-1}(j')$.

This technique is analogous with $r > 2$. We uncompute $B_1(j_1)$, $B_2(j_2)$, …, $B_{r-1}(j_{r-1})$ with a *single* look-up. We can do this because each value of j_i that appears in a tuple in J likely appears in a unique tuple, since there are only L possible values of j_i and it appears in L_i tuples. Since this is an uncomputation, the extra word size is irrelevant [70]. The greatest cost here seems to be computing the permutation π.

QRAM. Collimations repeatedly perform look-ups in quantum random access classical memory (QRACM), also known as quantum read-only memory (QROM). Given a large table of classical data $T = [t_0, \dots, t_{n-1}]$ of w-bit words, we want a circuit to perform the following:

$$|i\rangle\,|0\rangle \mapsto |i\rangle\,|t_i\rangle\,. \tag{5.13}$$

The simplest method is a sequential look-up from Babbush *et al.* [70], while Berry *et al.* [71] provide a version that parallelizes nicely. Beyond the minimum depth of that circuit, we use a wide circuit, Figure 5.1. Our cost estimation checks the cost of each of these circuits and chooses whichever has the lowest cost under each depth constraint; often this is Berry *et al.*'s circuit with $k \approx 8$.

Following Peikert we assume that if our target length is L, the actual look-ups will need to access $L_{max} = 8L$ words.

Memory latency has no effect on our final costs. For both the look-ups and the permutation computation, we added a depth of $(100W)^{1/2}$, where W is the total hardware

(classical and quantum) needed. Signal propagation over a single bit should be faster than execution of a single gate, which is our unit of depth, so $(100W)^{1/2}$ should safely overestimate the latency of accessing two-dimensional memory. This still had no effect on our final costs except under extreme conditions of more than about 2^{130} classical processors.

5.2.3 Permutation

To compute the permutation π, we start with r sorted lists of L elements in the range $[S]$. We want to find *all* tuples that add up to a specified value K in $[rS]$. For our estimation, we checked the cost of three different approaches and different r and chose the cheapest, which was often $r = 2$.

Problem 5.2.1 (Collimation permutation)**.** *Let L, S_1, and S_2 be integers such that $S_1 \gg S_2 \gg L$. On an input of r sorted lists $B_1, \ldots, B_r$ of L random numbers from 0 to S_1 and an integer K, list all r-tuples from $B_1 \times \cdots \times B_r$ such that their sum is in $\{KS_2, KS_2 + 1, \ldots, KS_2 + S_2 - 1\}$.*

One approach is to iterate through all $(r-1)$-tuples of elements from B_1 to B_{r-1}, compute the sum for each tuple, then search through B_r to find all elements that produce a sum in the correct range. This has a cost of approximately $L^{r-1} \lg L$, since we expect to check only $1/L^{r-1}$ elements in B_r for each $(r-1)$-tuple. With appropriate read-write controls, this parallelizes perfectly.

The structure of the sieve guarantees $S_2 \geq L^r$ for all but the final collimation. This means we cannot guess a value for the sum of the first $r/2$ lists, then search for a matching sum in the remaining lists, because we would need to guess $\frac{r}{2} S_2$ values, raising the cost over L^r. This prevents divide-and-conquer strategies like with a subset-sum, as in [69].

A lower-cost but memory intensive algorithm first merges s of the lists into a single sorted list of L^s s-tuples and their sums, at cost $L^s(s \lg L)$. Then it exhaustively searches the remaining L^{r-s} tuples, and searches for matches in the merged, sorted list. The total cost is $O(L^s + L^{r-s} s \lg L)$. We choose $s = \lfloor r/2 \rfloor$.

We assume both classical approaches parallelize perfectly, but we track the total numbers of classical processors required to fit in any depth limit.

Grover's algorithm . A simple quantum approach is Grover's algorithm, searching through the set of L^r r-tuples for those whose sum is in the correct range. This requires $O(L^{r/2})$ iterations, but each iteration requires r look-ups, which each cost $O(L)$. Each Grover search returns 1 possible tuple, creating a coupon-collector problem, so we repeat the Grover search $L \lg L$ times. The cost thus grows as $L^{\frac{r+3}{2}} \lg L$, which improves on the classical approach for $r \geq 5$.

The cost of Grover's algorithm gets much worse under a depth limit. Grover oracles should minimize their depth as much as possible, and since the look-up circuits parallelize

almost perfectly, we analyze only the wide look-up as a Grover oracle subroutine. We assume the $L \lg L$ search repetitions are parallel as well.

5.2.4 Sieving

To find the cost of each sieve repetition, we first find the depth of the tree of sieves. First we follow [72] to derive some facts about the distribution of phase vectors after sieving. Let $K = \{K_1, \ldots, K_s\}$ be all possible measurement results from collimation. We treat each of the L^r states in superposition as i.i.d. random variables X_i with values in K, defining $p_i = \mathbb{P}[X = K_i]$. Since the states are in uniform superposition, we imagine that measurement selects one such state X_j. Let W_j be the number of other states in the superposition with the same value as X_j; it equals $1 + \sum_{i \neq j} 1_{X_i = X_j}$. Conditioning on $X_j = K_m$ gives us

$$W_j | (X_j = K_m) = 1 + \sum_{i \neq j} 1_{X_i = K_m} \sim 1 + \mathrm{Bin}(L^r - 1, p_m).$$

This means

$$\begin{aligned} \mathbb{P}[W_j = w] &= \sum_{m=1}^{s} \mathbb{P}[W_j = w | X_j = K_m] \mathbb{P}[X_j = K_m] \\ &= \binom{L^r - 1}{w - 1} \sum_{m=1}^{s} p_m^w (1 - p_m)^{L^r - w}. \end{aligned} \tag{5.14}$$

The size of the collimated list is the expected value of W_j:

$$\mathbb{E}[W_j] = \sum_{w=0}^{L^r} w \binom{L^r - 1}{w - 1} \sum_{m=1}^{s} p_m^w (1 - p_m)^{L^r - w} \tag{5.15}$$

$$= \sum_{m=1}^{s} \frac{1}{L^r} \underbrace{\sum_{w=0}^{L^r} \binom{L^r}{w} w^2 p_m^w (1 - p_m)^{L^r - w}}_{(A_m)} \tag{5.16}$$

In the first layer of collimation X is uniformly random so $p_m = \frac{S_1}{S_0}$ and W_j is binomial, giving $\mathbb{E}[W_j] = \frac{S_1}{S_0}(L^r - 1) + 1$.

(A_m) is the expected value of the square of $\mathrm{Bin}(L^r, p_m)$, implying $\mathbb{E}[W_j]$ equals

$$\sum_{m=1}^{s} \frac{1}{L^r} \left((L^{2r} + L^r) p_m^2 - L_r p_m \right) = (L^r + 1) \sum_{m=1}^{s} p_m^2 - 1.$$

To find p_m for later collimations, we assume X is a sum of r i.i.d. uniformly random variables with values in $[0, \ldots, s]$ where $s = S_i / S_{i+1}$. By the central limit theorem this converges to a $N(r\mu, r\sigma^2)$ random variable, where $\mu = s/2$ and $\sigma^2 \approx \frac{s^2}{12}$.

We approximate $\sum_{m=1}^{s} p_m^2$ as the integral of the square of the probability density function for $N(\mu, \sigma^2)$, which is $\frac{1}{2\sqrt{\pi}\sigma}$. This gives us

$$\mathbb{E}[W_j] \approx (L^r + 1)\frac{\sqrt{3}}{\sqrt{r\pi}s} - 1. \tag{5.17}$$

This means the size of a new list is approximately $\frac{S_{i+1}}{S_i}\sqrt{\frac{3}{r\pi}}L^r$. We use $c_r := \sqrt{\frac{3}{r\pi}}$ as an "adjustor". Peikert takes this as $\frac{2}{3}$ for $r = 2$. Using the central limit theorem might be innacurate for small r, but in fact our adjustor gives ≈ 0.69 for $r = 2$, so we assume it is also accurate for $r \geq 3$.

This derivation replicates Peikert's result that each collimation reduces the height by a multiplicative factor of $L^{r-1}c_r$, with a more precise expression for c_r.

We start with a height of $N = \sqrt{p}$ and we want to reach a height of S_0, so the height of the tree must be

$$h = \left\lceil \frac{\lg(N/S_0)}{\lg(L^{r-1}/c_r)} \right\rceil. \tag{5.18}$$

Because of the rounding, we might need vectors of length less than L in the initial layer. Thus, we recalculate: The height of the phase vectors in the second layer (after the first collimation) must be $S_{h-1} = S_0(L^{r-1}/c_r)^{h-1}$.

The top layer has height $S_h = N$, the height of random new phase vectors. Since S_{h-1}/S_h is larger than any other layer, the phase vectors in the top layer only need a length L_0 which is less than L. Following Section 3.3.1 of Peikert and the previous derivation, the sieve requires $L_0 = (L\frac{N}{S_{h-1}})^{1/r}$. For this top layer we do not have the adjusting factor of c_r because the sum of r uniformly random values up to N, modulo N, will still be uniformly random.

This tells us how many oracle calls must be performed: There will be r^h leaf nodes in the tree, and each one must have length L_0. We adjust this slightly: Since each layer has some probability of failing, we divide this total by $(1 - \delta)^h$ for $\delta = 0.02$, which is an empirical value from Peikert. We also add a $2^{0.3}$ "fudge factor" from Peikert. The above analysis gives the number of oracle calls.

5.2.5 Fitting the sieve in a depth limit

We focus on NIST's security levels, which have a fixed limit `MAXDEPTH` on circuit depth, forcing the sieve to parallelize. The full algorithm consists of recursive sieving steps, producing a tree, where we collimate nodes together at one level to produce a node at the the next level. This parallelizes extremely well, though a tree of height h must do at least h sequential collimations.

From this, we use `MAXDEPTH`$/h$ as the depth limit for *each* collimation. The cost of collimation is mainly QRACM look-ups, which parallelize almost perfectly.

If each collimation has depth d_c and the tree has height h, then $\texttt{MAXDEPTH} - hd_c$ is the maximum depth available for oracle calls. We divide this by the depth for each oracle call, d_o, and then by the number of total oracle calls. This determines the number of oracle calls one must make simultaneously.

We also check whether collimation must be parallelized. We compute the total number of collimations in the tree, then multiply this by the depth of each collimation. Since one can start collimating as soon as the first oracle calls are done, the depth available for collimating is $\texttt{MAXDEPTH} - d_o$. This tells us how many parallel oracle calls the sieve must make, P_o, and the number of parallel collimations, P_c.

If $P_o > \lg(L_0)P_c$, then we will need to store extra phase vectors. We compute the depth to finish all the oracle calls, then subtract the number of phase vectors that are collimated in that time, to find the number that must be stored.

If $P_o \leq \lg(L_0)P_c$, the algorithm cannot parallelize the collimation as much as required, because the input rate of phase vectors is too low. Hence, we must increase P_o to $\lg(L_0)P_c$. This slightly overestimates the oracle's parallelization, since we can occupy the collimation circuits by collimating at higher levels in the tree, but since the number of vectors in successive levels of the tree decreases exponentially, we expect negligible impact.

5.2.6 Oracle costs

We propose that the cost of the oracle is the most likely factor for future algorithmic improvements to reduce CSIDH quantum security. Any improvement in basic quantum arithmetic will apply to computing the CSIDH group action in superposition; thus, using estimates from current quantum arithmetic techniques like [50], will almost certainly overestimate costs (indeed, the costs they reference have since been reduced [73]). The alternative approach of [9] was to produce a classical constant-time implementation to give a lower bound on cost, since latency, reversibility, and fault tolerance will add significant overheads.

However, there is some possibility that quantum implementations may be cheaper than reversible classical methods. A prominent example is the recent idea of "ghost pebbles" [74], which shows that the lower bounds on the costs of reversibly computing classical straight-line programs [75] do not hold for quantum computers.

We give some rough estimates for the oracle cost here. We start with [9] and assume the number of non-linear bit operations scales quadratically with the size of the prime. The √élu memory costs $8b + 3b\log_2 b$ field elements, where $b \approx \sqrt{\ell_{max}} \approx \sqrt{\frac{\log p}{\log\log p}}$ is the largest isogeny computed. Each field element is $\log_2 p$ bits. We assume that this is enough to hold the "state" of the group action evaluation, and thus we can apply straight-line ghost pebbling techniques. This is likely not optimal but it is a first approximation. We assume that the depth is equal to the number of operations, though with perfect parallelization up to a factor of $\log_2 p$. We treat each non-linear bit operation as a quantum AND gate, and do not include linear bit operation costs.

Pebbling. Reversible computers cannot delete memory, and "pebbling" is the process of managing a limited amount of memory ("pebbles") to compute a program. We refer to [75] for details. Ghost pebbling [74] is a quantum technique where we measure a state in the $\{|+\rangle, |-\rangle\}$-basis, which releases the qubits but may add an unwanted phase that must be cleaned up. For our purposes, a pebble will be a state of many qubits, so with near certainty, a measurement-based uncomputation will leave a phase that we need to remove.

Our strategy is as follows: Suppose we have enough qubits to hold s states simultaneously and n steps remaining in the program. From one state we can compute the next step, uncompute the previous state with measurements, and then repeat this; this only requires 2 states at a time. As a base case for $s = 3$, this gives the "Constant Space" strategy from [74], which requires $\frac{n(n+1)}{2}$ steps. In fact we only need 2 states, since we either consider the final state separately from this accounting, or we only need to clear the phase from the final state.

For a recursive strategy, we pick some $k < n$, and repeat the 2-states-at-a-time method to reach step $n-k$. We then recurse with $s-1$ states for the final k steps, then uncompute the state at step $n-k$ with a measurement. To clean up the phase from this measurement, we repeat the 2-states-at-a-time to reach step $n-2k$, then recurse for the next k steps. We repeat this process until all phases are removed.

If $C(k, s-1)$ is the cost for the recursive step, this has total cost

$$\left\lceil \frac{n}{k} \right\rceil C(k, s-1) + \sum_{i=0}^{\left\lfloor \frac{n}{k} \right\rfloor} ik. \tag{5.19}$$

Based on some simple optimization, we choose $k = n^{\frac{s-1}{s}}$. We find the total costs numerically, and test initial values of s between $\frac{1}{2}\lg n$ and $5\lg n$ to find an optimal value. Table 5.2 gives the costs of one call to the oracle.

5.3 Security of Low Exponents

One of our main contributions is low exponents as secret keys. Our key space is thus a small subset of the class group. We believe that this extra information does not help a quantum adversary, for the following reasons:

1. The representation of group elements as a bitstring must be homomorphic to bitstrings representing integers;
2. Creating an incomplete superposition of states will not produce properly formed phase vectors; and
3. Incorrect phase vectors as input are likely undetectable, uncorrectable, and quickly render the sieve useless.

Table 5.2: Estimated CSIDH group action oracle costs in log base 2, including 2^{10} overhead for total cost and $2^{6.7}$ hardware overhead for each logical qubit.

Prime Size	**Logical Operations**	**Depth**	**Hardware**	**Cost** (DW)
512	44.9	44.0	26.5	73.4
1024	46.9	46.0	28.0	77.4
1792	48.5	47.6	29.3	80.3
3072	50.1	49.2	30.6	83.1
4096	50.9	50.0	31.2	84.6
5120	51.6	50.6	31.8	85.7
6144	52.1	51.2	32.2	86.7
8192	52.9	52.0	32.8	88.2
9216	53.2	52.3	33.1	88.8

We will explain each point in detail. These support our main assumptions:

- Quantum adversaries will still need to search the *entire* class group;
- The oracle for a quantum adversary will need to evaluate arbitrary group actions, not just small exponents.

Both points mean that the quantum security depends only on the size of the class group, not the size of the subset we draw keys from. Importantly, these assumptions fail if we restrict the keys to a small sub*group* of the class group. It is critical that the subset of keys generates the entire class group.

5.3.1 Group Representations

To create the input states, we must use a QFT which computes a homomorphism between elements of the group and phases of quantum states. Circuits to do this are well-known only for modular integers, represented as bitstrings. With a different representation of group elements (e.g., vectors in lattice), we either need a custom-built QFT circuit for that representation, or we first change the representation to modular integers. However, a custom-built QFT is equivalent to a change of representation: we could apply the custom QFT, then the inverse of the usual QFT to integers, and this will map our group elements to modular integers.

This seems to restrict us to representing elements of the class group as multiples of a generator. We might be able to reduce the cost of the search if we only used small multiples of this generator; however, low exponents do not correspond to small multiples.

Hence, the exponent vectors will likely be indistinguishable from random multiples of the generator.

The state before the QFT has the form $|0\rangle|x\rangle + |1\rangle|x+s\rangle$, where x is the coefficient of the generator for the group element that we measured. Hence, if x is randomly distributed, we will still need $\lg|G|$ qubits to represent it, and the QFT will produce random phase vectors of height up to $|G|$. Since the cost of the sieve is governed by the height of the input phase vectors, the cost of the sieve will be the same.

In short, to exploit the fact that secrets are restricted, we require a representation of group elements that can be homomorphically compressed to fewer than $\lg|G|$ qubits. We see no method to do this.

5.3.2 Incomplete Superpositions

The first step of producing phase vectors involves a superposition over all of G. If we know that the secret s is in a smaller subset $H_1 \subseteq G$, we could instead sample from H_1. We could even sample from another set H_0 for f, though it must be the same size for the normalization to match. This produces a superposition

$$|0\rangle \sum_{g\in H_0} |g\rangle|f(g)\rangle + |1\rangle \sum_{g\in H_1} |g\rangle|h(g)\rangle\,. \tag{5.20}$$

Measuring the final register returns a particular value $z = f(g)$ for some $g \in H_0$ or $z = h(g) = f(g-S)$ for some $g \in H_1$. Let $Z = f(H_0) \cup h(H_1)$, and partition it into 3 subsets: $Z_0 = f(H_0) \setminus h(H_1)$, $Z_+ = f(H_0) \cap h(H_1)$, and $Z_1 = h(H_1) \setminus f(H_0)$. If we measure $z \in Z_0$, then the state after the QFT is just $|0\rangle$, since there was no value $g \in H_1$ such that $h(g) = z$. Similarly, measuring $z \in Z_1$ leaves the state $|1\rangle$. Only if we measure $z \in Z_+$ will we have a "successful" phase vector, i.e., one that is not just $|0\rangle$ or $|1\rangle$ and has some information about s.

The size of Z_+ is $|H_0 \cap (S + H_1)| \leq |H_0|$, and the probability of measuring $z \in Z_+$ is $|Z_+|/|H_0|$. Choosing H_0 and H_1 to make this probability large, without knowing S, seems very challenging. For example:

Theorem 5.3.1. *If we generate a uniform superposition of exponent vectors with elements in $\{-m,\ldots,+m\}$, then for a key in $\{-1,1\}^n$, the probability of a successful phase vector is*

$$\left(\frac{2m}{2m+1}\right)^n. \tag{5.21}$$

Proof. There are $2(2m+1)^n$ states in superposition when we measure: $(2m+1)^n$ exponent vectors in superposition for each value $|0\rangle$ or $|1\rangle$ of the leading qubit. Each state has equal probability. We measure curves, meaning that a curve reached by both E_0 and E_1 is twice as likely as a curve reached by only one or the other.

For small m, the set of curves reached by E_0 is close to a bijection with a hypercube of exponent vectors of width $(2m+1)$ and centered at 0. The set of curves reached by E_1 is in bijection with a hypercube of exponent vectors of the same width centered at s, the exponent vector of the secret key. The intersection of these hypercubes has volume $(2m)^n$, giving Equation 5.21. □

5.3.3 Effects of Incomplete Superpositions

We define a defective phase vector with fidelity q of length L as a triple $(B, J, |\phi\rangle)$, where $B : \{0, \ldots, L\} \to [N]$ is classically known, $J \subseteq [L]$ is *not* classically known and $|J| = qL$, and

$$|\phi\rangle = \sum_{j \in J} \zeta_s^{B(j)} |j\rangle . \tag{5.22}$$

If we measure a $|0\rangle$ or $|1\rangle$ state from an oracle that produces incomplete superpositions, then $q = \frac{1}{2}$, $B(1) = b$, but $B(0) = 0$ and $J = \{0\}$ or $J = \{1\}$.

In short, a phase vector with $q < 1$ is one where our classical beliefs about the set of phases in superposition are wrong. We know the function B correctly, but it only matches the real state on the unknown subset J. The issue is that the oracle cannot tell us the fidelity of a new phase vector; our measurements do not tell us whether we succeeded or not.

We call this fidelity because it represents quantum fidelity with respect to the state we believe we have, given the classical information of the function B. This means that if k input phase vectors are defective, the fidelity of the entire input state degrades to 2^{-k}. If our final phase vector before measurement has fidelity q with respect to the state we want, then q is the probability of measuring the same result [64, Section 9.2.2]. That is, the final fidelity gives us the probability of actually recovering any bits of the secret. We need this to be very high, since there is no efficient method to test whether a small number of key bits are correct. Such a method would trivially break the scheme by guessing and checking.

Hence, if our input states have fidelity q, we need the fidelity to increase by the time we reach the final state. Quantum circuits without measurement are unitary operations and thus preserve fidelity, but measurements may increase it, so we first argue that collimation does not appreciably increase the fidelity.

Theorem 5.3.2. *Starting with an initial phase vector of length L and fidelity $q < \frac{1}{2}$, with height S, if we collimate to a new height S', the resulting phase vector is a new defective phase vector with expected fidelity at most*

$$q + 4\sqrt{\frac{\ln(L')}{L'}}, \tag{5.23}$$

for $L' := \frac{S}{S'} Lq \geq 40$.

Proof. The probability of measuring any phase is uniform in the first collimation. This means p_m is constant in Equation 5.14, so the length of any state after measurement, which we denote X, has distribution $1 + \mathrm{Bin}(|J| - 1, S'/S) = 1 + \mathrm{Bin}(qL - 1, p)$ for $p = S'/S$ and $qL = |J|$. The length of phases that we incorrectly believe we have will have distribution $Y \sim \mathrm{Bin}(L - qL, p)$.

The fidelity of the measured state is $\frac{X}{X+Y}$. We use Chernoff bounds to concentrate X and Y to be within a factor of $(1 \pm \delta)$ of their means, except with probability $\epsilon := 2\exp(-\mathbb{E}[x]\delta^2/3) + 2\exp(-\mathbb{E}[y]\delta^2/3)$. With $\delta = \sqrt{3\ln(L')/L'}$, since $q < \frac{1}{2}$, this gives $\epsilon < \frac{5}{L'}$.

We know $\frac{X}{X+Y} \leq 1$ so we can bound $\mathbb{E}[\frac{X}{X+Y}]$ as

$$\mathbb{E}\left[\frac{X}{X+Y}\right] \leq \frac{1+\delta}{1-\delta}\frac{p(qL-1)+1}{p(qL-1)+1+p(L-qL)} + \epsilon. \tag{5.24}$$

With careful rearranging we find

$$q + \frac{q}{L'} + 2\sqrt{3}\sqrt{\frac{\ln(L')}{L'}} + \frac{5}{L'}. \tag{5.25}$$

For sufficiently large L' this fits the required bound. □

Theorem 5.3.2 shows that for small q, the fidelity increases only linearly with each collimation. The factor of L' is approximately equal to the *actual* number of states in superposition in the collimated phase vector. Each phase vector is only collimated once for each level of the tree and there are only $\approx 2^7$ sequential collimations, even at very large prime sizes. Hence the sieve could only tolerate ≈ 7 defective input phase vectors. Sieving over a 6144-bit prime needs 2^{89} input phase vectors, so we would need the probability of failure to be approximately 2^{-86}. Given Theorem 5.3.1, this nearly rules out sampling low exponents.

Since sieving is ineffective, it would be desirable instead to take many phase vectors, some of which may be defective, and produce good vectors out of them. We summarize this as the following problem:

Problem 5.3.1 (Probabilistic Phase Vector Distillation (PPVD))**.** *Let s be an unknown secret value. As input, there are n input states $|\phi_k\rangle$ with labels k, such that with probability p, $|\phi_k\rangle = |0\rangle + e^{iks/N}|1\rangle$, with probability $\frac{1-p}{2}$, $|\phi_k\rangle = |0\rangle$, and with probability $\frac{1-p}{2}$, $|\phi_k\rangle = |1\rangle$.*

With some probability ϵ, either output 0 for failure or output 1 and t states $|\phi_{j_1}\rangle, \ldots, |\phi_{j_t}\rangle$ and their associated phase multipliers j_i, such that, for all i:

$$|\phi_{j_i}\rangle = |0\rangle + e^{ij_is/N}|1\rangle. \tag{5.26}$$

The PPVD problem cannot be solved with *epsilon* > 0 for $n = 1$:

Lemma 5.3.1. *There is no quantum channel (circuit plus measurement) that distinguishes a single phase vector from $|0\rangle$ or $|1\rangle$ without calling the group oracle or learning the secret s.*

Proof. Suppose such a quantum channel Φ exists. Since the states we want to distinguish are constrained to a 2-dimensional subspace, any measurement will produce a state in a 1-dimensional space, which is a single vector. Since we want the output to be a phase vector, our measurement must produce a valid phase vector $|\phi'\rangle$. Suppose $|\phi'\rangle$ has some associated phase j. The vector $|\phi'\rangle$ is the basis of our measurement, and thus cannot depend on the input states nor the secret s, since we assume we do not learn s. Hence, for an input $|\phi\rangle = |0\rangle + e^{iks/N}|1\rangle$, the secret is s, so we require $|\phi'\rangle = |0\rangle + e^{ijs/N}|1\rangle$. But if we instead had an input for a secret $s' \neq s$, then $|\phi'\rangle$ is not a correct phase vector. □

The argument of Lemma 5.3.1 does not readily extend to $n > 1$, but we assume that similar arguments exist. The central issue is that our distillation process must project inputs onto phase vectors that are correct for an *unknown* secret phase multiplier s. We see no way to do this without learning s and without being able to produce correct phase vectors from "blank" inputs of $|0\rangle$ and $|1\rangle$. Either of these cases implies a more efficient solution to the dihedral hidden subgroup problem. We make that last statement more precise and argue that we cannot expect to "gain" phase vectors on average:

Lemma 5.3.2. *If the collimation sieve gives the optimal query complexity for the dihedral hidden subgroup problem, then no process can solve PPVD with $t\epsilon > pn$.*

Proof. For a contradiction, let $t\epsilon > pn$. Assume we have a perfect phase vector oracle, from which we make n initial queries. We then take pn of our phase vectors and shuffle them together with $|0\rangle$ and $|1\rangle$ vectors. Then we run the process that solves the PPVD. If it succeeds, it produces t new phase vectors, which we add to a growing list; if it fails, we call the phase vector oracle another t times. Either way we have $t - np$ new phase vectors, and in the first case we did not need to call the oracle. Thus each iteration calls the oracle $t(1-\epsilon)$ times on average. We repeat this process to create all the phase vectors that the collimation sieve needs.

If the collimation sieve requires Q states, this process only calls the oracle $\frac{Q}{t-np}t(1-\epsilon)$ times. If $t\epsilon > pn$, then

$$\frac{Q}{t-np}t(1-\epsilon) < \frac{Q}{t-t\epsilon}t(1-\epsilon) = Q \tag{5.27}$$

and thus we solve the dihedral hidden subgroup problem with fewer than Q states. □

5.4 Discussing secure CSIDH instatiations

In this section we discuss the various factor that play a role in the security analysis of CSIDH instantiations. We consider the two facets of security, beginning with quantum security in Subsection 5.4.1 and then discussing classical security in Subsection 5.4.2.

5.4.1 Quantum-secure CSIDH instatiations

Table 5.4 presents estimated costs for quantum sieve attacks against different prime sizes, based on the analysis in Section 5.2. NIST defines post-quantum security levels relative to the costs of key search against AES (we assume an offline single-target attack) and collision search against SHA-3 [14], for which the most efficient attacks, respectively, are Grover's algorithm (which is quantum) and van Oorschot & Wiener's (vOW) algorithm (which is classical) [42].

To compare these three algorithms, which have distinct space-time trade-offs, we include fixed hardware limits and add a fault tolerance overhead. These assumptions are stronger than the assumptions used in the analyses of other post-quantum schemes, particularly proposed NIST standards. Since CSIDH, and our 'SQALE'd version, are not being considered for standardization, we use riskier assumptions in our cost model. This means the performance is not directly comparable to other post-quantum schemes at the same security level. Our recommended parameters are a 4096-bit prime for Level 1, 6144 bits for level 2, and 8192 bits for level 3.

Quantum Oracle Costs The number of oracle calls decreases with the size of the prime, relative to the total computational expense. To increase our estimate of the attack cost against a 4096-bit prime, the isogeny oracle must cost at least 2^{54} gates, and at least 2^{79} gates to change the 8192-bit prime cost estimate. These are high but may be realistic. Bonnetain and Schrottenloher [50] estimate 2^{63} T-gates just for CSIDH-1792; however, their estimate is based on costs for modular arithmetic that have subsequently been improved. Bernstein *et al.* [9] gave a circuit with 2^{40} non-linear bit operations, leaving the full quantum cost to be determined. Both predate the $\sqrt{\text{élu}}$ technique and neither exploit any fully quantum techniques. Since basing security on current isogeny evaluation costs seems precarious, we only account for the costs of the collimation sieve itself.

Hardware Limits Grover-like quantum algorithms parallelize very badly, but the collimation sieve parallelizes almost perfectly. Thus the threshold for security increases as depth decreases, but CSIDH's bits of security remain the same. To an adversary with a high depth budget of 2^{96}, SQALE'd CSIDH-4096 costs much more to break than AES-128, but costs much less to break if the adversary must finish their attack in depth 2^{40}. There is therefore not a straightforward answer to whether SQALE'd CSIDH-4096 is as secure as AES-128.

We assert that it does not matter if an adversary with access to more than 2^{80} qubits could attack AES-128 at a higher cost than attacking CSIDH-4096, since such an adversary is unrealistic. We constrain an adversary's amount of "hardware", the total of classical processors, memory, and *physical* qubits (see Subsection 5.1.3). All three are given equal weight. Under limits of both hardware and depth, certain attacks are impossible. The

depths in Table 5.4 are the minimum depths for which the collimation sieve can finish under our hardware constraint. Because Grover search becomes more expensive at lower depths, this removes high-cost attacks on AES.

Our hardware limit for NIST level 1 is 2^{80}, based on [41]. For level 2 we use 2^{100}, the memory contained in a "New York City-sized memory made of petabyte micro-SD cards" [76], and for level 3 we use 2^{119}, the memory of a 15 mm shell of such cards around the Earth [76].

5.4.2 Classical Security

Assume we want to find a CSIDH key that connects two given supersingular Montgomery curves E_0 and E_1 defined over $\mathbb{F}_p$ for a prime $p = 4 \cdot \prod_{i=1}^{n} \ell_i - 1$. Let N denote the key space size.

Notice, large primes $p \gg 2^{512}$ permit smaller key space sizes $N \ll p^{1/2}$ than the class group order; and then, random-walk-based attacks are costlier than Meet-In-The-Middle (MITM) procedures. In fact, MITM performs about $N^{1/2} \ll p^{1/4}$ steps.

To illustrate the MITM approach, let us assume that for $i \coloneqq 1, \ldots, n$, we require the computation of isogenies of degree-ℓ_i, each of which we repeat $m \in \mathbb{Z}^+$ times. The first step is to split the set $\{\ell_1, \ldots, \ell_n\}$ into two disjoint subsets $\mathcal{L}_0$ and $\mathcal{L}_1$, both of size $\frac{n}{2}$. Next, for $i = 0, 1$, let $\mathcal{S}_i$ be the table with elements $(\vec{e}, g_{\vec{e}})$ where $g_{\vec{e}}$ corresponds to the output of the group action evaluation with inputs E_i, and a CSIDH key $\vec{e} = (e_1, \ldots, e_n)$ such that $e_j = 0$ for each $\ell_j \in \mathcal{L}_{1-i}$. The MITM procedure on CSIDH looks for a collision between $\mathcal{S}_0$ and $\mathcal{S}_1$; that is, two pairs $(\vec{e}, g_{\vec{e}}) \in \mathcal{S}_0$ and $(\vec{f}, g_{\vec{f}}) \in \mathcal{S}_1$ such that $g_{\vec{e}} = g_{\vec{f}}$; consequently, the concatenation of $\vec{e}$ and $\vec{f}$, maps E_0 to E_1.

The tables $\mathcal{S}_0$ and $\mathcal{S}_1$ each have about $N^{1/2}$ elements [2]. The size of the class group $\#\mathrm{cl}(\mathcal{O})$ is asymptotically close to $p^{1/2}$, and the key space size N must be (approximately) equal to $2^{2\lambda}$ to ensure $\lambda \in \{128, 192\}$ bits of classical security. Consequently, for large primes $p \gg 2^{1024}$, we have

$$\#\mathcal{S}_0 \approx \#\mathcal{S}_1 \approx 2^{\lambda} \ll \#\mathrm{cl}(\mathcal{O})^{1/2} \approx p^{1/4}. \tag{5.28}$$

Then $(\#\mathcal{S}_1)(\#\mathcal{S}_0) \ll \#\mathrm{cl}(\mathcal{O})$, and the birthday-paradox probability of a collision between $\mathcal{S}_0$ and $\mathcal{S}_1$ (other than the one expected by construction) happening by chance is negligible. The expected running-time of MITM is $1.5N^{1/2}$ and it requires $N^{1/2} \approx 2^{\lambda}$ cells of memory. Here, the classical security of CSIDH falls into the same case as SIDH, where van Oorschot & Wiener (vOW) Golden Collision Search (GCS) is cheaper than MITM, and a small key space still provides $\lambda \in \{128, 192\}$ bits of classical security. In fact, the van Oorschot & Wiener Golden Collision search procedure [41, 42] applied to CSIDH has an expected running-time of

[2] In general, when m_i degree-ℓ_i isogeny constructions are required for each $i = 1, \ldots, n$, where the cardinality of the sets $\mathcal{L}_0$ and $\mathcal{L}_1$ should be $\#\mathcal{S}_0, \#\mathcal{S}_1 \approx N^{1/2}$.

Bound m	Classical security					
	128-bits			**192-bits**		
	OAYT	MCR	Dummy-free	OAYT	MCR	Dummy-free
5	64	86	86	89	119	119
4	70	95	95	97	132	132
3	79	111	111	109	153	153
2	95	139	139	132	193	193
1	139	221	221	193	306	306

Table 5.3: Number of small odd primes ℓ_i's required for ensuring 128 and 192 bits of classical security given the hardware bounds we set.

$$\frac{1}{\mu}\left(7.1 \times \frac{N^{3/4}}{w^{1/2}}\right) \tag{5.29}$$

when only μ processors and w cells of memory are allowed to be used. As a consequence, the number k of small odd primes ℓ_i's that allows λ-bits of classical security is

$$k \approx \frac{4}{3}\left(\frac{\lambda + \frac{1}{2}\log_2(w) - \log_2(7.1)}{\log_2(\delta m + 1)}\right), \tag{5.30}$$

where $N = (\delta m + 1)^k$ and $(\delta m + 1)$ determine the size of either $[\![-m \,..\, m]\!]$ ($\delta = 2$, OAYT-style [63]), $[\![0 \,..\, m]\!]$ ($\delta = 1$, MCR-style [62]) or $\{\,(\,\}\, m) = \{e \in [\![-m \,..\, m]\!] \mid e \equiv m \bmod 2\}$ ($\delta = 1$, dummy-free style [61]).

Assuming the previously mentioned technological limits of $w = 2^{80}, w = 2^{100}, w = 2^{119}$ cells of classical memory for NIST levels 1, 2, 3 (resp.), Table 5.3 summarizes and compares the number k of small odd primes required as a function of the maximun number m of isogeny constructions per prime. In each case, we then found independent bounds m_i for each degree-ℓ_i isogeny construction to optimize the cost using the approach reported in [56]. Note that any increase in our classical memory budget w will imply a higher value of k, thus forcing us to re-parameterize the collection of k isogenies that must be processed.

Quantum collision-finding. For quantum security we analyze only the collimation sieve, but a quantum attacker could attack the meet-in-the-middle problem, just as a classical attacker. With the cost model we use, the best attack is Multi-Grover with distinguished points [77]. SIKE-434 and SIKE-610 have larger search spaces than we consider, and likely have higher oracle costs. Under the Multi-Grover attack, these SIKE

parameters meet NIST security levels 1 and 3, respectively, so we conclude that our parameters are also secure against this attack.

5.5 Experimental results

In this section, we discuss larger and safer CSIDH instantiations. We report the first constant-time C-coded implementation of the CSIDH group action evaluation that uses the new fast isogeny algorithm of [25], as reported in [57]. The C-code implementation allows an easy application for any prime field, which requires the shortest differential addition chains (SDACs), the list of small odd primes (SOPs), and the optimal strategies presented in [56]; in particular, our C-code implementation is a direct application of the algorithm and Python-code presented in [57], and thus all the data framework required (for each different prime field) can be obtained from its corresponding Python-code version.

Our experiments focus on instantiations of CSIDH with primes of the form $p = 4\prod_{i=1}^{n} \ell_i - 1$ of 1024, 1792, 2048, 3072, 4096, 5120, 6144, 8192, and 9216 bits (see Table 5.5). We compared the three variants of CSIDH, namely, i) MCR-style, ii) OAYT-style, and ii) Dummy-free-style. All of our experiments were executed on a Intel(R) Core(TM) i7-6700K CPU 4.00GHz machine with 16GB of RAM, with Turbo boost disabled and using `clang` version 3.8. Our software library is freely available from

https://github.com/JJChiDguez/sqale-csidh-velusqrt .

To illustrate the impact of using low exponents, Figure 5.3 shows experimental results for all instantiations of CSIDH using exponent bounds ranging from $m = 1$ to $m = 5$. Each exponent bound is parameterized to reach the same security, meaning fewer ℓ_i for larger m. In all cases we started from the global bound and then optimized for the bounds per individual small prime and evaluation strategies as in [56]. Note that some configurations of the 1024- and 1792-bit primes do not have enough ℓ_i's to support the $m = 1$ and $m = 2$ bounds.

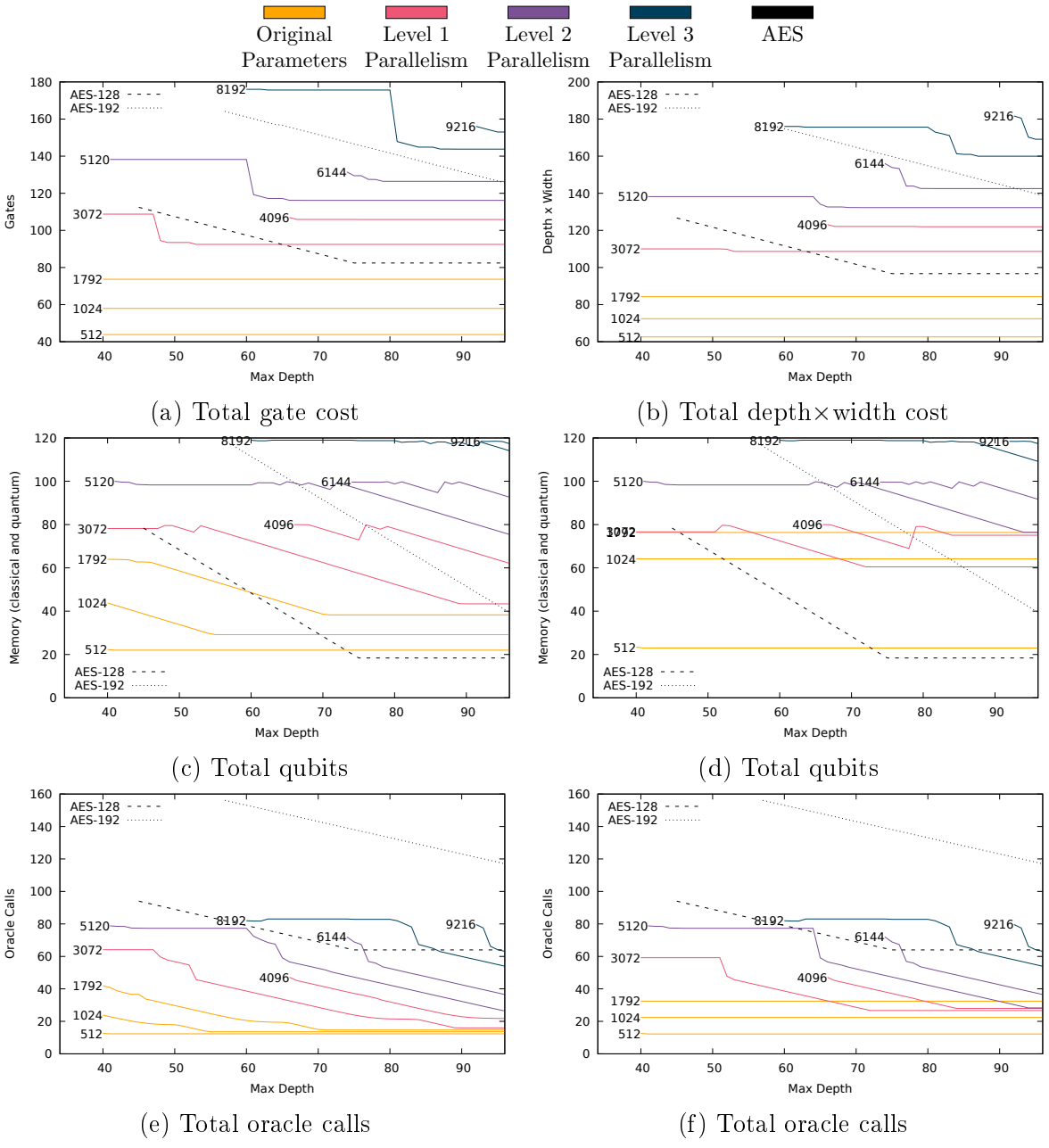
Our results show a slight drop in performance with the $m = 1$ bound in both the dummy-free and MCR-style versions, then for $m = 2$ onwards, higher m steadily performs worse. For OAYT style, on the other hand, $m = 1$ was always optimal. Because the performance bump at $m = 1$ appears to get ameliorated at higher primes, we decided to use the $m = 1$ bound for all cases due to its simplicity and security. The results for these instantiations, which provide NIST security leves 1, 2, and 3, are in Table 5.6. These results correspond with the measurement of 1024 random instances.

5.6 Discussion

As the quantum security analysis of CSIDH has become more robust, it seems clear now that its original parameters must be updated by considering larger primes.

Prime Length	**Depth (min.)**	**Oracle Calls**	**Qubits**	**Classical Hardware**	**Cost** (DW)	**Hardware**	**Cost** (DW)
NIST Level 1 (hardware limit 2^{80})							
CSIDH						AES-128	
512	40	21	13	24	63	*89*	132
1024	40	23	22	64	72	*89*	132
1792	40	36	33	74	83	*89*	132
3072	40	55	59	77	110	*89*	132
4096	66	70	48	80	124	36	106
5120	81	77	44	80	135	18	97
NIST Level 2 (hardware limit 2^{100})							
CSIDH						SHA-256	
5120	41	73	77	99	139	*105*	146
6144	74	89	72	100	156	72	146
NIST Level 3 (hardware limit 2^{119})							
CSIDH						AES-192	
6144	40	74	96	115	146	*151*	195
8192	60	78	82	119	176	111	175
9216	92	102	79	118	181	47	143
CSIDH, lowest cost with no hardware constraints							
3072	47	49	46	94	103		
4096	45	56	59	108	117		
5120	44	64	68	121	130		
6144	52	70	73	132	142		
8192	51	83	88	151	160		
9216	54	87	91	161	171		

Table 5.4: Quantum attack costs against SQALE'd CSIDH. Depth is the minimum possible under the given hardware limit. The final two columns give the lowest cost of attacking {AES,SHA} in depth at least as much as the minimum to break the associated CSIDH instance, based on [78, 79, 14]. Italics highlights where such a break exceeds the hardware limit.



(a) Total gate cost

(b) Total depth×width cost

(c) Total qubits

(d) Total qubits

(e) Total oracle calls

(f) Total oracle calls

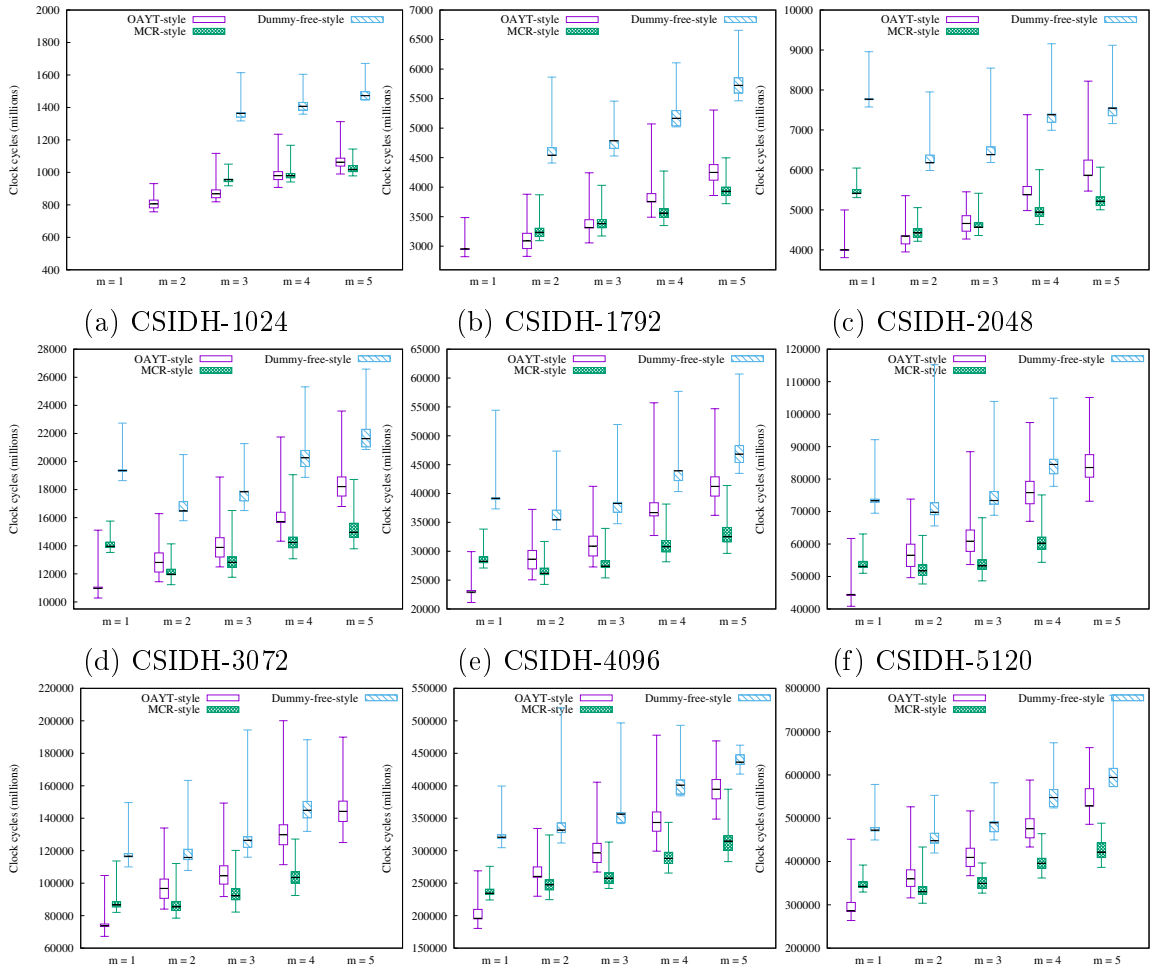
Figure 5.2: Costs of the quantum collimation sieve attack under various hardware limits. Coloured solid lines are the costs of the collimation sieve at primes of bit lengths from 512 to 9126; dotted lines are the cost of key search on AES, from [79], with the same memory limits and overhead as our analysis. All figures are logarithmic in base 2. Plots on the left are parameterized to minimize gate cost, plots on the right to minimize DW-cost. Larger primes achieving lower depth (e.g., 5120 vs. 4096) is due to increased memory limits.

$\log_2(p)$	n	**Excluded**	**Included**
1024	130	739	983
1792	207	149	1289
2048	231	5	3413
3072	326	37, 2053	2203, 2007
4096	417	1151	2897
5120	504	5	4133
6144	590	71, 4289	4337, 4339
8192	757	4937, 5749	5783, 5791
9216	838	263, 6373	6473, 6481

Table 5.5: Shape of the primes: $p = 4\prod_{i=1}^{n} \ell_i - 1$, where $\ell_1, \ldots, \ell_n$ are the first n odd prime numbers, excluding and including the listed primes in columns **Excluded** and **Included**, respectively.

Table 5.6: Clock Cycles (in **gigacycles**) corresponding to CSIDH instantiations with 4096, 5120, 6144, 8192, and 9216 bits. Each CSIDH instantiation uses $m = 1$ (one isogeny construction per each ℓ_i). The measured clock cycles are the average of 1024 random instances without key validation.

Instantiation	**Style**			**NIST**
	OAYT	MCR	Dummy-free	**Security**
CSIDH-4096	23.21	28.50	39.35	Level 1
CSIDH-5120	44.56	53.39	73.57	Level 1
CSIDH-6144	74.88	87.09	117.57	Level 2
CSIDH-8192	199.15	236.13	322.57	Level 3
CSIDH-9216	292.41	346.46	475.64	Level 3



(a) CSIDH-1024 (b) CSIDH-1792 (c) CSIDH-2048

(d) CSIDH-3072 (e) CSIDH-4096 (f) CSIDH-5120

(g) CSIDH-6144 (h) CSIDH-8192 (i) CSIDH-9216

Figure 5.3: Group action evaluation cost (excluding key validation) for each CSIDH instantiation from 1024 to 9216 bits. The CSIDH configurations are according to Table 5.3. Each experiment considers the cost of 1024 random instances, except for experiments corresponding to the 8192- and 9216-bit instances which consider a smaller set of 128 experiments.

In this chapter, we have proposes a set of primes large enough to make the protocol quantum-secure. Taking as a basis the Python 3 library reported in [57], we provide a freely available software library coded in C, which implements CSIDH instantiations that were built using these large primes.

Since the introduction of CSIDH in 2018, it has been the norm to try to approximate the key space to its maximum theoretical size of $\#\mathrm{cl}(\mathcal{O}) \approx \sqrt{p}$. Nevertheless, as quantum security demands a larger prime, this key space has become unnecessarily large. It is therefore important to prove that leaving a portion of this space unused does not compromise the CSIDH security, which is an important conjecture that our analysis supports.

To make larger prime field instantiations of CSIDH more viable, our implementation combines techniques such as exponent strategy optimization, low exponents, and the new Vélu formulas presented in [25]. Our results are the first of their kind for these larger primes, hoping that these designs will pave the path forward for future refinements of CSIDH.

From our analysis, the main computational cost of the quantum sieve comes from the classical cost of merging lists to find permutations. Improvements to this subroutine would lower the security of CSIDH. Given that CSIDH's relative security and it's 'SQALE'd performance depend on hardware limits, our analysis highlights the need for consensus on the resources of far-future attackers.

Chapter 6

Supersingular Isogeny Path Finding with Limited Memory

Abstract. *This work has been published in INDOCRYPT 2022 in collaboration with E. Bellini, JJ. Chi-Domínguez, A. Esser, S. Ionica, L. Rivera-Zamarripa, F. Rodríguez-Henríquez, M. Trimoska and F. Zweydinger [5]. It studies the computational hardness of finding an isogeny between two supersingular isogenous curves defined over a prime field $\mathbb{F}_q$ with q a power of a large prime p, which is the underlying problem on which the security guarantees of most isogeny-based protocols rely. In a classical setting, a meet-in-the-middle algorithm is the fastest known strategy for solving this problem. However, due to its stringent memory requirements, it quickly becomes infeasible for moderately large instances. One has therefore to resort to time-memory trade-offs to instantiate attacks, particularly in GPU platforms, which are inherently more memory-constrained than CPU architectures. In such a setting, a van Oorschot-Wiener-based collision finding algorithm offers a better asymptotic scaling. We present a precise estimation of the costs of both strategies considering most recent algorithmic improvements, and substantiate our estimations via optimized software implementations of both algorithms. In this context, we provide the first optimized GPU implementation of the van Oorschot-Wiener approach for attacking isogenies. Based on practical measurements we extrapolate the running times for solving different-sized instances. Finally, we give estimates of the costs of computing a degree-2^{88} isogeny using our CUDA software library running on an NVIDIA A100 GPU server.*

Let E_0 and E_1 be two supersingular isogenous elliptic curves defined over a finite field $\mathbb{F}_q$, with q a power of a large prime p which can be taken to be $q = p^2$ without loss of generality. Computing an isogeny $\phi\colon E_0 \to E_1$ is believed to be hard in the classical as well as the quantum setting and is known as the Supersingular Isogeny Path (*SIP*) problem. In many scenarios, the isogeny is of known degree ℓ^e for some small prime ℓ and

we refer to this variant as the Supersingular Fixed-Degree Isogeny Path (*SIPFD*) problem. Investigating the concrete computational hardness of *SIPFD* and the best approaches to tackle it in multi- and many-core CPU and GPU platforms, is the main focus of this work.

In the context of cryptographic protocols, *SIP* was first studied by Charles, Goren and Lauter [31]. They reduced the collision resistance of a provably secure hash function to the problem of finding two isogenies of equal degree ℓ^n for a small prime ℓ and $n \in \mathbb{Z}$ between any two supersingular elliptic curves. This in turn may also be tackled as a *SIPFD* problem.

Variants of the *SIPFD* problem form the basis of several isogeny-based signatures [32, 16]. Further, *SIPFD* has been used as foundation of recently proposed cryptographic primitives, including Verifiable Delay Functions such as the one of De Feo *et al.* [80] and the one presented in Chapter 7. Based on the intractability of the *SIPFD* problem, Jao and De Feo proposed the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) [1, 2]. Apart from revealing the isogeny degree, SIDH also reveals the evaluation of its secret isogenies at a large torsion subgroup. This weaker variant of *SIPFD* was dubbed as the *Computational Supersingular Isogeny* (CSSI) problem [1]. SIKE [15], a variant of SIDH equipped with a key encapsulation mechanism, was one of the few schemes that made it to the fourth round of the NIST standardization effort as a KEM candidate [14]. Until recently, the best-known algorithms for breaking SIDH or SIKE had an exponential time complexity in both, classical and quantum settings.

However, in July 2022, Castryck and Decru [21] proposed a surprising attack that (heuristically) solves the CSSI problem in polynomial-time. This attack relies on the knowledge of three crucial pieces of information, namely, (i) the degree of the isogeny ϕ; (ii) the endomorphism ring of the starting curve E_0; and (iii) the images $\phi(P_0), \phi(Q_0)$ of Alice's generator points $\langle P_0, Q_0 \rangle = E[2^a]$, where the prime $p = 2^a 3^b - 1$ is the underlying prime used by SIKE instantiations. Recall that (ii) and (iii) are only known in the specific case of the CSSI problem, but not in the more general case of the *SIPFD* problem. Furthermore, another attack by Maino and Martindale [38] and yet another one by Robert [39] quickly followed. Maino and Martindale's attack relies on several crucial steps used in [21], but does not require knowledge of the endomorphism ring associated to the base curve. Robert's attack can also break SIDH for any random starting supersingular elliptic curve.

Despite the short time elapsed since the publication of Castryck and Decru's attack, several countermeasures have already been proposed by trying to hide the degree of the isogeny [35], the endomorphism ring of the base curve [36], or the images of the torsion points [37]. At this point, only time will tell if SIDH/SIKE will ever recover from the attacks on the CSSI problem. But even if this never happens, the theoretical and practical importance of the *SIPFD* problem still stands. For instance, the constructions from [32, Section 4], [33, Section 5.3] and the one presented in Chapter 7 do not append images of auxiliary points to their public keys. In turn the Castryck-Decru family of attacks does not apply, making the security of those applications entirely based on the *SIPFD*

problem.

Known attacks on the *SIPFD* problem. Even before the publication of the attack in [21], it was wildly believed that the best approaches for solving the CSSI problem are classical and not quantum [65]. Here we present a brief summary of the different assumptions made across the last decade about the cost of solving the *SIPFD* problem. We stress that while all these advances were made with SIKE as main motivation, the fact that they did not make use of the torsion point images means that they still represent the state-of-the-art for attacks against the general *SIPFD* problem. The fastest known algorithm for solving *SIP* has computational complexity $\tilde{O}(\sqrt{p})$ [81, 45, 82]. However, if the secret isogeny is of known degree ℓ^e, there might exist more efficient algorithms for solving the *SIPFD*. Indeed, in their NIST first round submission, the SIKE team [15] argued that the best classical attack against the CSSI problem was to treat it as an *SIPFD* problem and use a MitM approach with a time and memory cost of $O(\ell^{\frac{e}{2}})$, which is more efficient for SIKE and all instantiations of the *SIPFD* where $\ell^e \leq p$.

By assuming an unlimited memory budget and memory queries with zero time cost, the MitM attack is indeed the best attack against the *SIPFD* problem. Nevertheless, in [41], the authors argued that the van Oorschot-Wiener (vOW) golden collision search, which yields a better time-memory trade-off curve, is the best classical approach for large instances. The rationale used is that the $O(\ell^{\frac{e}{2}})$ memory requirement for launching the MitM attack is infeasible for the cryptographic parameter sizes. Since the best known generic attacks against AES use a negligible amount of memory, it is just natural to set an upper bound on the available classical memory when evaluating the cost of solving *SIPFD* instantiations in the context of NIST security levels 1 to 5.

To increase interest in studying the CSSI problem Costello published in [83] two Microsoft $IKE challenges, a small and a large one using a 182- and a 217-bit prime number, respectively. These two CSSI instances are known as $IKEp182 and $IKEp217 challenges.[1] A few months later, the solution of $IKEp182 was announced by Udovenko and Vitto in [44]. The authors treated this challenge as an instance of *SIPFD*, and then used a MitM approach largely following the description given in [43] along with several clever sorting and sieving tricks for optimizing data queries for their disk-based storage solution. The authors reported that their attack had a timing cost of less than 10 core-years, but at the price of using 256 TiB of high-performance network storage memory.

It is obvious that this memory requirements quickly render the strategy unfeasible for larger non-toy instances. As mentioned in [41], there exists a time-memory trade-off variant of the MitM algorithm (*cf.* Subsection 6.1.2), which was adopted by Udovenko and Vitto to bring the storage requirements of their attack down to about 70 TiB.

However, determining the best algorithmic choice for solving instances of given size under a certain memory budget and computational platform remains so far an open problem.

[1] The precise specifications can be found in https://github.com/microsoft/SIKE-challenges.

In this work we present a framework predicting that both MitM variants are outperformed by the vOW golden collision approach even for moderately large *SIPFD* instances. We then substantiate our claims by extrapolating results of our implementations, accounting for practical effects such as memory access costs.

Our contributions In [41] it was found that vOW is a better approach than MitM to tackle large *SIPFD* instances. However, the small Microsoft challenge $IKEp182 was broken, before the Castryck-Decru attack was known, using a MitM strategy [44]. As discussed in [44], it remains unclear for which instance sizes and memory availability, vOW outperforms MitM. In this work we answer this question from a theoretical and practical perspective. Theoretically, we give a precise estimation of the costs of both strategies including most recent algorithmic improvements. Practically we substantiate our estimations via optimized implementations and extensive benchmarking performed in CPU and GPU platforms.

Moreover, in the case of CPU platforms, we present a detailed framework that for a fixed memory budget and prime size, predicts when a pure MitM approach, batched (limited memory) MitM or vOW approach becomes the optimal design choice for attacking *SIPFD* (see Section 6.2 and Figure 6.2). The predictions of our model are backed up by practical experiments on small *SIPFD* instances and extrapolations based on the obtained practical timings of our implementations.

We additionally provide the first optimized GPU implementation of the vOW attack on *SIPFD*, outperforming a CPU based implementation by a factor of almost two magnitudes. We provide medium sized experimental data points using our GPU implementation including extrapolations to larger instances. More concretely, our implementation solves *SIPFD* instances with isogeny degree 2^{88} with primes of bit size 180 (comparable to the instance solved in [44]) using 16 GPUs each equipped with only 80 GiB of memory in about 4 months. Based on our experimental results we conclude that vOW is the preferred choice for any larger *SIPFD* instances on reasonable hardware.

Our CPU and GPU software libraries are open-source and available at `https://github.com/TheSIPFDTeam/SIPFD`.

Outline. The remainder of this work is organized as follows. In Section 6.1 we present a formal definition of *SIPFD* and relevant mathematical background. We also give a detailed explanation of the MitM and vOW strategies. In Section 6.2 we present a careful estimation of the cost of the MitM and vOW strategies and their corresponding trade-offs in the context of the *SIPFD*. In Section 6.3 we present our implementation and results of the CPU-based MitM attack, which is used as baseline to compare against the more complex GPU-based vOW implementation. The later implementation is described in Section 6.4, and the results obtained from it in Section 6.5. Finally, Section 6.6 discusses our results and their implications by extrapolating them to larger instances.

6.1 Preliminaries

In this section we provide some preliminary background for this work. We start with a more detailed explanation of the setting of the problem that we are attacking in Subsection 6.1.1. Then, we describe the two basic approaches for attacking the problem: the Meet in the Middle attack and the Parallel Collision Search, in Subsection 6.1.2 and Subsection 6.1.3, respectively.

6.1.1 Our setting for the SIPFD problem

In this subsection we describe our concrete setting for attacking the SIPFD problem, which is formally defined as follows:

Definition 6.1.1 (*SIPFD* problem)**.** *Let p, ℓ be two prime numbers. Consider E_0 and E_1 two supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ such that $\#E_0(\mathbb{F}_{p^2}) = \#E_1(\mathbb{F}_{p^2})$. Given $e \in \mathbb{N}$ find an isogeny of degree ℓ^e from E_0 to E_1, if it exists.*

For concreteness, we will assume that the degree of the secret isogeny is 2^e, that this number divides $p+1$, and that the bitlength of p is around $2e$. We also assume that the curves have Frobenius trace $t = -p$, so that the curve cardinality is $\#E(\mathbb{F}_{p^2}) = p^2 - t + 1 = (p+1)^2$ and we can find a basis $P, Q \in E(\mathbb{F}_{p^2})$ for the torsion subgroup $E[2^e]$ (that is, the set of points whose order divides 2^e). All of the assumptions mentioned in this paragraph are efficiency-oriented decisions that are commonly adhered to in applications, but it must be stressed that we only make them for concreteness and they are not exploited by our attacks in any significant way (other than in benefiting in performance in the same way that a protocol would), so our results should be considered as applying to the SIPFD problem in general.

The basis P, Q can be used to parametrize all the possible isogenies, in the sense that the kernel of any degree-2^e isogeny can be written as either $\langle P + [k]Q \rangle$ or $\langle [k]P + Q \rangle$ for some integer $k \in \mathbb{Z}_{2^e}$. For simplicity we will assume that kernels can only take the form $\langle P + [k]Q \rangle$ for some choice of basis, but there is little loss in generality since recovering the general case would at most require repeating the attack with a flipped basis.

An isogeny $\phi : E_0 \to E_1$ of degree 2^e can be written as a composition $\phi = \phi_1 \circ \phi_0$ of two isogenies of degree $2^{e/2}$ (assuming an even e for simplicity), where $\phi_0 : E_0 \to E_m$ and $\phi_1 : E_m \to E_1$ for some middle curve E_m. Since there exists a dual isogeny $\hat{\phi}_1 : E_1 \to E_m$, one can conduct a Meet in the Middle (MitM) attack by exploring all the possible $2^{e/2}$-isogenies emanating from E_0 and E_1, and finding the pair of isogenies that arrive to the same curve E_m (up to isomorphism). The largest attack recorded on the *SIPFD* problem, conducted by Udovenko and Vitto[2] [44], used this strategy to break an instance with $e = 88$.

[2]This work was realized as an attack on SIKE, but does not exploit the torsion point images and can be regarded as an attack on *SIPFD* in general.

6.1.2 Meet in the Middle (MitM)

Let us briefly recall the MitM procedure to solve the *SIPFD* for $\ell = 2$. We first compute and store all $2^{e/2}$-isogenous curves to E_0 in a table T (identified via their j-invariants). Then we proceed by computing each $2^{e/2}$-isogenous curve to E_1 and check if its j-invariant is present in table T. Any matching pair then allows to recover the secret isogeny as outlined in the previous section.

Complexity. Let $N := 2^{e/2}$. The worst-case time complexity of the MitM attack is $2N$ evaluations of degree-$2^{e/2}$ isogenies, while in the average case $1.5N$ such evaluations are necessary. The space complexity is dominated by the size of the table to store the N j-invariants and scalars.

In a memory restricted setting, where the table size is limited to W entries, the MitM attack is performed in batches. In each batch, we compute and store the output of W isogenies from E_0, then compute and compare against each of the N isogenies from E_1 without storing them. The number of batches is N/W where each batch performs W isogenies from E_0 and N isogenies from E_1, yielding a total of $\frac{N}{W}(N+W)$ evaluations of $2^{e/2}$-isogenies.

Depth-First Search methodology. In 2018, Adj *et al.* [41, §3.2] exploited the fact that an ℓ^e-isogeny can be regarded as the composition of e isogenies of degree ℓ, and showed that computing the isogenies from each side in a depth-first tree fashion yields performance improvements. The improvement stems from the fact that, whenever two isogenies share the same initial path, the depth-first approach avoids re-computation of those steps.

In order to adapt to the limited-memory scenario, let us assume that the available memory can hold $W = 2^\omega$ entries. Then each batch of isogenies from E_0 can be obtained by following a fixed path for the first $e/2-\omega$ steps, and then computing the whole subtree of depth ω from this node.

Also, the attack is easy to parallelize. Assuming 2^c threads are used, all trees can be branched sequentially for c steps to obtain 2^c subtrees, each of which is assigned to a different core. This methodology for evaluating trees in batches and with multiple cores is summarized in Figure 6.1.

Since a binary tree of depth ω has $2^{\omega+1}-2$ edges, each batch is computing $e/2-\omega+2^{\omega+1}-2$ isogenies of degree 2 for the side corresponding to E_0, and the whole tree with $2^{e/2+1}-2$ isogenies for the side corresponding to E_1. The expected cost corresponding to half of the batches is then

$$\frac{1}{2}2^{e/2-\omega}\left(2^{e/2+1}+2^{\omega+1}+e/2-\omega-4\right) \approx 2^{e-\omega}$$

computations of 2-isogenies.

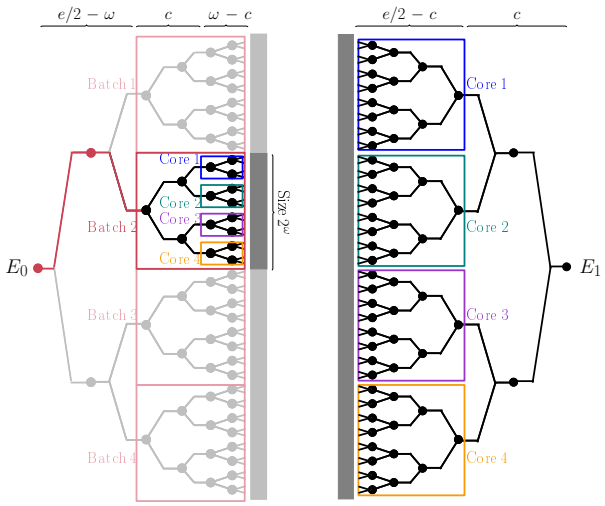


Figure 6.1: The batched Meet-in-the-Middle depth-first approach: finding a 2^e-isogeny between E_0 and E_1 with 2^c cores and 2^ω memory. In this example, $e = 12$, $c = 2$, and $\omega = 4$.

6.1.3 Parallel Collision Search

Given a random function $f : S \to S$, van Oorschot and Wiener's method [42] is a parallel procedure to find collisions in f. The main idea of the algorithm is to construct in parallel several chains of evaluations $x_i = f(x_{i-1})$, starting from random seeds x_0. Further, a small fraction of the points in S is called *distinguished* based on an arbitrary criterion (e.g. that the binary representation of $x \in S$ ends with a certain number of zeros). A chain continues until it reaches a distinguished point. Then this point is compared against a hash table including all previously found distinguished points. Further, to avoid infinite loops, chains are aborted after their length exceeds a specified threshold.

Two chains ending in the same distinguished point indicate a collision between those chains. This collision can be efficiently reconstructed if the seeds x_0, x_0' and the lengths d, d' of the colliding chains are known. Therefore, assuming $d > d'$ we take $d - d'$ steps on the longer chain (starting from x_0). From there on we take simultaneous steps on both chains, checking after each step if the collision has occurred. Hence, the hash table stores for each found distinguished point the triplet (x_0, x_d, d) indexed by x_d.

Complexity. Let N be the size of the set S, θ the proportion of points that are distinguished, and W the amount of distinguished triplets that we can store. Since each chain has an average length of $1/\theta$, the chains represented by the stored triplets (once the hash

table is completely filled) include an average of W/θ points. Therefore the probability that a given evaluation of f collides with any of these points is $W/N\theta$. After a collision takes place, the chain needs to continue for an additional $1/\theta$ steps on average before it reaches a distinguished point and the collision is detected. At this point, the two involved chains must be reconstructed from the start to find the exact step at which the collision occurred, yielding a total of $N\theta/W + 3/\theta$ evaluations of f to find a collision. The optimal choice for θ is $\sqrt{3W/N}$ yielding a cost of $2\sqrt{3N/W}$ per collision. Note, however that this analysis assumes a table that already contains W triplets. To capture the transition effect of the table filling up, van Oorschot and Wiener [42] model $\theta = \alpha\sqrt{W/N}$ for a parameter α that is experimentally measured to be optimal at $\alpha = 2.25$. The resulting cost per collision is found to be linear in $\sqrt{N/W}$ as long as $2^{10} < W < N/2^{10}$.

Note that any random function from S to itself is expected to have $N/2$ collisions, however, many applications, including the *SIPFD*, require looking for one specific collision that we refer to as the "golden collision" [84, 85, 86, 87]. This means that the attack has to find $N/4$ different collisions on average before stumbling upon the golden collision, bringing the total cost to $\mathcal{O}(\sqrt{N^3/W})$ function evaluations.

Application to the *SIPFD* problem To attack the *SIPFD* problem and find the kernel of a degree-2^e isogeny between E_0 and E_1, we assume for simplicity that e is even and define $S = \{0,1\} \times \{0,\ldots,2^{e/2}-1\}$ so that $N = 2^{e/2+1}$. We also define the map $g : S \to \mathbb{F}_{p^2},\ (c,k) \mapsto j(E_c/\langle P_c + [k]Q_c\rangle)$, where (P_c, Q_c) are a predefined basis of the $2^{e/2}$-torsion on either side as before. As explained in Section 6.1.1, the function g yields a bijection between S and the set of $2^{e/2}$-isogenies with kernel $\langle P_c + [k]Q_c\rangle$ from the curves on either side. A collision $g(c,k) = g(c',k')$ with $c \neq c'$ implies two isogenous curves starting on opposite sides and meeting at a middle curve (up to isomorphism).

To apply the parallel collision search, we need a function f that maps S back to itself. Hence, we have to work with the composition $f = h \circ g$ where h is an arbitrary function mapping j-invariants back to S. This composition introduces several fake collisions that are produced by the underlying hash function while there is still only one (golden) collision that leads to the secret isogeny.

Note that for a certain (unlucky) choice of hash function h the golden collision might not be detectable.[3] Therefore, we have to periodically switch the hash function h. More precisely, we switch the function whenever we found a certain amount C of distinguished points. If we model $C = \beta \cdot W$ for some constant β, then each hash function will have a probability of $2\beta W/N$ for finding the golden collision. Experimentally, van Oorschot and Wiener [42] found $\beta = 10$ to perform best, and the average running time of the attack is measured to be $(2.5\sqrt{N^3/W})/m$, where m is the number of processors computing paths in parallel.

[3]For instance, one of the points that leads to the golden collision might be part of a cycle that does not reach a distinguished point.

6.2 Accurate formulas for vOW and MitM

So far, we have provided theoretical cost functions for the golden collision search in terms of the number of evaluations of the function f, and for the batched depth-first MitM in terms of the number of 2-isogeny evaluations. We now provide a more detailed cost model in terms of elliptic curve operations to make these costs directly comparable. These formulas give a first indication of which memory regime favors which algorithm and, further, they form the starting point for parameter selection in our implementation.

6.2.1 Meet in the Middle

For the depth-first MitM, we have counted only the 2-isogeny evaluations but the total cost involves also obtaining the kernel points of each isogeny and pushing the basis points through the isogeny. As described in [41], the total cost of processing a node at depth d can be summarized as:

- $2^{e/2-d}$ point doublings to compute the kernel points
- 2 isogeny constructions to compute the children nodes
- 1 point doubling, 1 point addition, and 6 isogeny evaluations to push the basis through the isogenies.

Nodes at the second-to-last level represent an exception since once we obtain the leaves, we no longer require pushing the bases and instead we need to compute the j-invariant.

Let us refer by `ADD, DBL, ISOG, EVAL, JINV` to the cost of a point addition, point doubling, 2-isogeny construction, 2-isogeny evaluation at a point, and j-invariant computation, respectively. The total cost of computing a tree of depth $e/2$ is then

$$\begin{aligned}\mathrm{DFS}(e/2) &= \sum_{d=0}^{e/2-2} 2^d \left((2^{e/2-d}+1)\mathtt{DBL} + 2\mathtt{ISOG} + 1\mathtt{ADD} + 6\mathtt{EVAL}\right) \\ &\quad + 2^{e/2-1}\left(2\mathtt{DBL} + 2\mathtt{ISOG} + 2\mathtt{JINV}\right) \\ &= 2^{e/2-1}\left((e+1)\mathtt{DBL} + 4\mathtt{ISOG} + 1\mathtt{ADD} + 6\mathtt{EVAL} + 2\mathtt{JINV}\right) + O(1).\end{aligned}$$

The expected time of the whole MitM attack using 2^ω memory entries, which computes a tree of depth ω on one side and a tree of depth $e/2$ on the other side for each batch, is then

$$\begin{aligned}\mathrm{MitM}(e,\omega) &= \frac{2^{e/2}}{2\cdot 2^\omega}(DFS(\omega) + DFS(e/2)) \\ &\approx (2^{e-\omega-2} + 2^{e/2-2})\left(\mathtt{DBL} + 4\mathtt{ISOG} + 1\mathtt{ADD} + 6\mathtt{EVAL} + 2\mathtt{JINV}\right) \\ &\quad + (2^{e-\omega-2}e/2 + 2^{e/2-2}\omega)\mathtt{DBL}.\end{aligned}$$

6.2.2 Golden Collision Search

For the golden collision search, the cost of an evaluation of the random function, given a scalar $k \in \mathbb{Z}_{2^{e/2}}$ and a bit $c \in \{0,1\}$, consists of

- computing the kernel point $P_i + [k]Q_i$,
- constructing a single $2^{e/2}$-isogeny with said kernel and
- computing the j-invariant of the output curve.

The first step is usually done with a three-point Montgomery ladder which has an average cost of $\frac{e}{2}(\mathtt{DBL} + \mathtt{ADD})$. For the second step, it is shown in [1] that a "balanced" strategy for computing a $2^{e/2}$-isogeny costs about $\frac{e}{4}\log(e/2)\mathtt{DBL} + \frac{e}{4}\log(e/2)\mathtt{EVAL} + \frac{e}{2}\mathtt{ISOG}$. Hence, the total expected sequential time of the golden collision search is

$$\begin{aligned}\mathrm{GCS}(e,\omega) = 2.5 \cdot 2^{3(e/2+1)/2-\omega/2} \\ \times \left(\frac{e}{4}\log(e/2)(\mathtt{DBL}+\mathtt{EVAL}) + \frac{e}{2}\mathtt{ISOG} + \mathtt{JINV} + \frac{e}{2}(\mathtt{DBL}+\mathtt{ADD})\right).\end{aligned}$$

6.2.3 Simplified Cost Models for Montgomery Curves

Assuming that we use Montgomery curve arithmetic, then the cost of curve operations can be expressed in terms of field additions, multiplications, squares and inverses (`A`, `M`, `S`, `I`, respectively) as follows (compare to [15])

$$\begin{aligned}&\mathtt{DBL} = 4\mathtt{A} + 4\mathtt{M} + 2\mathtt{S}, \quad \mathtt{ADD} = 6\mathtt{A} + 4\mathtt{M} + 2\mathtt{S}, \quad \mathtt{ISOG} = \mathtt{A} + 2\mathtt{S} \quad \text{and} \\ &\mathtt{EVAL} = 6\mathtt{A} + 4\mathtt{M}, \qquad \mathtt{JINV} = 8\mathtt{A} + 3\mathtt{M} + 4\mathtt{S} + \mathtt{I}.\end{aligned}$$

Moreover, we assume $\mathtt{M} = 1.5\mathtt{S} = 100\mathtt{A} = 0.02\mathtt{I}$ which we have obtained experimentally from our quadratic field arithmetic implementation. The cost models can then be written in units of `M` as

$$\mathrm{MitM}(e,\omega)/\mathtt{M} \approx \frac{22799}{600}\left(2^{e-\omega} + 2^{e/2}\right) + \frac{403}{300}\left(2^{e-\omega}\cdot e/2 + 2^{e/2}\cdot\omega\right) \tag{6.1}$$

and

$$\mathrm{GCS}(e,\omega)/\mathtt{M} \approx 2.5 \cdot 2^{3(e/2+1)/2-\omega/2}\left(\frac{4181}{75} + \frac{1211}{200}e + \frac{283}{120}e\log(e/2)\right) \tag{6.2}$$

For a given value of e and a memory budget ω, we can now determine which algorithm is favorable. Figure 6.2 visualizes three different regions. For $\omega \geq e/2$ the full MitM attack without batching can be applied. The batched MitM attack is found to have a narrow area of application at the border of the region where the golden collision search is optimal, which dominates the largest part of the limited-memory area.

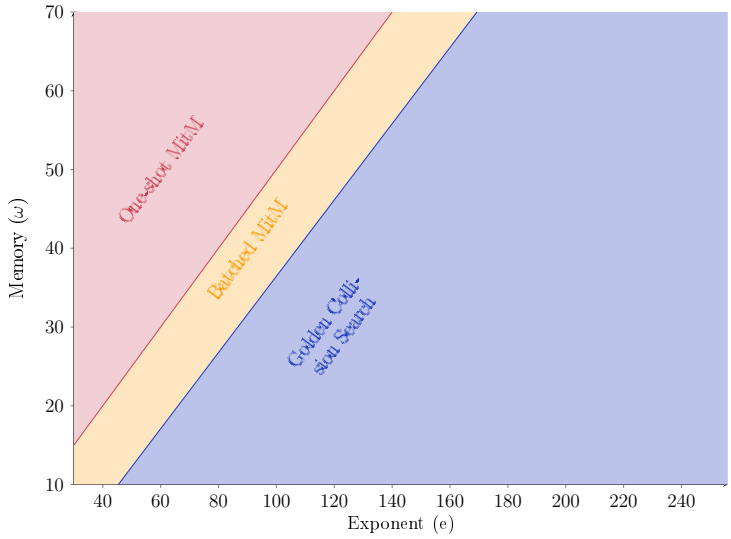


Figure 6.2: Regions in the (e, ω) space where each attack is optimal for solving a SIPFD problem of size 2^e with memory limited to 2^w entries.

We would like to stress, that this comparison is based on idealized models involving only underlying field arithmetic operations. They do not take into account any practical effects, as e.g. memory access timings or parallelization issues. Nevertheless, it gives a first indication of the superiority of the golden collision search in the limited memory setting.

6.3 Practical Results of our MitM CPU Implementation

We have implemented the batched depth-first MitM attack and run experiments on an AMD EPYC 7763 64-Core processor at 2.45 GHz, running 32 threads in parallel.

The j-invariants in each batch are stored in RAM, along with the corresponding scalar k. Each processor maintains an array with j-invariants that have been calculated and sort lexicographically, to reduce the number of memory accesses when searching for the collision.

For measuring the performance of the batched depth-first MitM with the memory parameter ω, we fix a small instance with exponent $e = 50$ and benchmark the attack for ω with $18 \leq \omega \leq 25$. These timings are compared to Equation 6.1, using a separate benchmark for the cost of `M`, i.e. a multiplication operation of our implementation. As shown in Figure 6.3, the experimental measurements are found to adhere to the model up to an overhead factor of about 2, which is explained by the memory access times and sorting overheads that are not accounted for in Equation 6.1.

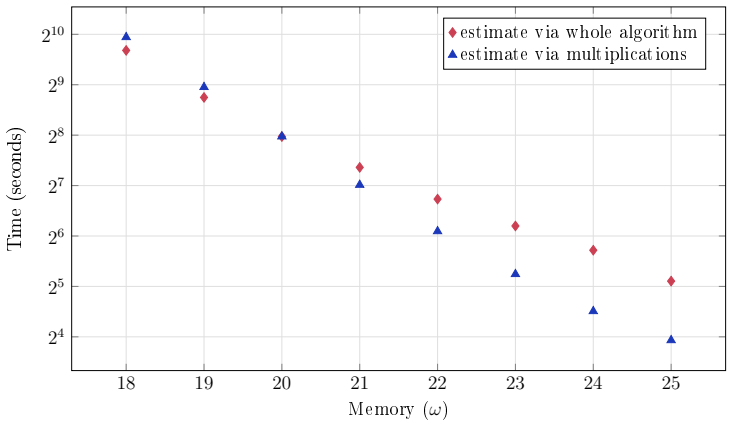


Figure 6.3: Completion time of the MitM attack for an exponent $e = 50$ using 32 physical processors and different memory bounds compared to the prediction in Equation 6.1.

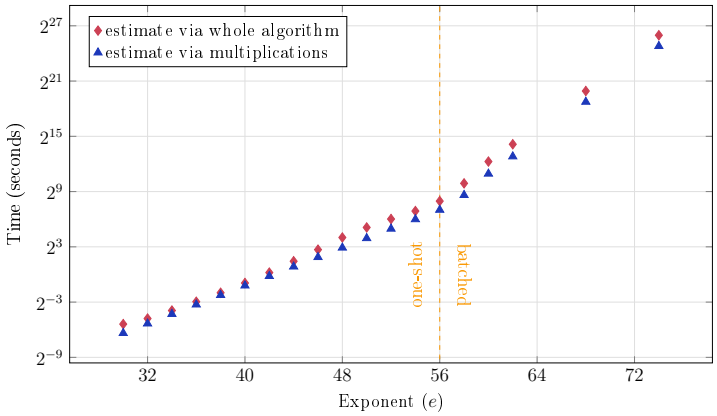


Figure 6.4: Completion time of the MitM attack for various exponent sizes.

We then tested the attack for increasing values of e while limiting the memory to $\omega \leq 28$. For $e > 56$, the batched MitM must be used and we have estimated the complexity of the whole attack by completing a single batch. As expected, Figure 6.4 shows that the slope of the cost changes drastically once we enter the limited-memory region. The overhead factor between the experimental results and the theoretical model is always found to be less than 2.6. We conclude that Equation 6.1 can be used to estimate the cost of the attack for larger parameters without significant overhead.

For comparison, the instance solved by Udovenko and Vitto in [44] was in the unlimited-memory setting using $e = 88$ and $\omega = 44$. Based on our model and adjusting to their clock frequency, we obtain an estimate of 9.47 core-years for the attack. This is close to Udovenko and Vitto's experimental result of 8.5 core-years, despite the fact that they used network storage.

6.4 Practical considerations for our vOW GPU implementation

We now describe our GPU-based implementation of the parallel collision search. For context, we first give a brief explanation of the GPU architecture we used, followed by a summary of practical features of our implementation.

6.4.1 GPU architecture

An NVIDIA CUDA device allows to execute thousands of threads in parallel. Following the Single Instructions Multiple Thread (SIMT) paradigm, a collection of 32 threads is bundled in *warps* that can only perform the same instruction on different data. One of the main challenges when programming CUDA devices is to decrease the memory latency, i.e., the time the threads are waiting for the data to be loaded into the corresponding registers. Therefore all CUDA devices have a multiple-level memory hierarchy incorporating memory and caches of different size and speed.

The NVIDIA A100 has an 80GB sized main memory, connected to other GPUs in the same cluster via a high throughput bus called *NVLINK*. However, for performing computations, data must be propagated through the two levels of caches down to the registers. Each thread has only a very limited amount of these registers. Whenever more registers are addressed than physically available, the memory must be outsourced to other memory levels, causing latency and stalls. Further, whenever more threads are requested than the hardware can handle concurrently, a scheduling is performed, by swapping active threads against queued ones. As a consequence, caches must be invalided, which leads to further memory latency. However, there is usually an optimal number of concurrent threads such that memory latency can be minimized by an optimal scheduling.

GPU potential of vOW. Note that the major task performed inside the vOW algorithm is the computation of chains of evaluations of the given function on different inputs. Therefore, it fits into the SIMT paradigm and can effectively be parallelized on the GPU. Further, since the devices are inherently memory-constrained, they profit from the good asymptotic trade-off curve of the vOW collision search.

6.4.2 Practical features

We briefly describe the various optimizations that our implementation adopts:

Hash function. For performance improvements, we heuristically model hash functions with ℓ-bit output as the projection to the first ℓ bits of the input. To obtain a randomized version we xor a fixed random nonce to the output. That is, for a given nonce $\mathbf{r} \in \mathbb{F}_2^\ell$ the

hash function $h_{\mathbf{r}}\colon \mathbb{F}_2^* \mapsto \mathbb{F}_2^\ell$ is defined as $h_{\mathbf{r}}(\mathbf{x}) := (x_1, \ldots, x_\ell) + \mathbf{r}$. This is justified by the fact that the inputs usually inherit already enough randomness, which is confirmed in our experiments.

Memory optimizations. The bit-size of every triplet (x_0, x_d, d) is roughly $e+\log(20/\theta)$, since x_0, x_d encode $2^{e/2}$-isogenies and the length of each chain is $d < 20/\theta$. However, due to our hash function choice, we can omit $\log W$ bits of x_d referring to its address in the table, plus another $\log(1/\theta)$ bits from the fact that it is a distinguished point, giving a size of roughly $e + \log(20) - \log W$ bits per triplet.

PTX assembly. We provide core functionalities of our GPU implementation in PTX (Parallel Thread eXecution) assembly, which is the low level instruction set of NVIDIA CUDA GPUs. This includes our own optimized $\mathbb{F}_p$ arithmetic. In this context, we provide optimized version of both the schoolbook and the Karatsuba algorithm for integer multiplication, as well as the Montgomery reduction.

Data structure. For storing distinguished points we compare the performance of a standard hash table against the Packed Radix-Tree-List (PRTL) proposed in [88]. The PRTL is a hash table that stores a linked list at each address, instead of single elements. This avoids the need for element replacement in case of hash collisions. Further it identifies the address of an element via its prefix (*radix*) and stores only the prefix-truncated element. The *packed* property of the PRTL relates to distinguished point triplets being stored as a single bit-vector, thus, avoiding the waste of space due to alignment. We ran CPU experiments with both data structures to identify the optimal choice prior to translating the code to the GPU setting. Eventually, we adopted the packed property and the use of prefixes, while we found no improvement in performance from using linked lists.

Precomputation. As discussed in [43], the time T_f required for a function evaluation can be decreased via precomputation. For a depth parameter d, one can precompute the 2^d curves corresponding to all the 2^d-isogenies from E_0 and E_1. When computing a $2^{e/2}$-isogeny, the initial d steps are replaced by a table lookup and we end up computing only a $2^{e/2-d}$-isogeny. Note that the memory needed for precomputation grows exponentially with d and so asymptotically it does not play a relevant role. However, for relatively small parameters it can provide valuable savings and speed up our experiments without affecting metrics such as the number of calls to f.

6.5 Practical results of our vOW GPU implementation

We now present results obtained by our GPU implementation and put them together with the known theoretical behavior to extrapolate the time to solve larger instances. In the original work of van Oorschodt-Wiener the time complexity of the procedure was found to be well approximated by

$$\frac{1}{m}(2.5\sqrt{N^3/W}) \cdot T_f, \tag{6.3}$$

where T_f is the cost per function evaluation. Therefore, we measure the cost T_f of our implementation which then allows us to derive an estimate for arbitrary instances. Further, we compare this estimate against the theoretical estimate via Equation 6.2 and an estimate based on collecting a certain amount of distinguished points.

Additionally, we verify that our GPU implementation using the functions specified in Subsection 6.1.3 has a similar behavior as the CPU implementation using random functions of [42]. This increases the reliability in our estimates, as it shows that the time complexity of our implementation is still well approximated by Equation 6.3. Let us start with this verification.

6.5.1 Verifying the theoretical behavior

In [42] van Oorschot and Wiener find that on average it takes $\frac{0.45N}{W}$ randomized versions of the function to find the solution, which in our case corresponds to random choices of the hash function (compare to Subsection 6.1.3). In their experiments, the function is changed after $\beta \cdot W$ distinguished points have been discovered, where a value of $\beta = 10$ is found to be optimal. Further, chains are aborted after they reach a length of $20\theta^{-1}$, i.e., 20 times their expected length.

Optimal value of β. Let us first verify that an amount of $10 \cdot W$ distinguished points until we abort the collision search for the current version of the function is still a suitable choice for our implementation. Table 6.1 shows the average running time of our vOW implementation using different values of β. We conclude that the values around $\beta = 10$ give comparable performance, with $\beta = 10$ being optimal in most of the experiments. The results are averaged over 100 ($e = 34$) and 50 ($e = 36$) runs respectively.

Expected number of randomized versions of the function. Now that we confirmed the optimal choice of β, we expect that the required amount of random functions until success also matches the one from [42]. In this case, the number of required randomizations of the function until the golden collision is found should follow a geometric distribution with parameter close to $\frac{W}{0.45N}$.

e	ω	$\beta = 5$	$\beta = 10$	$\beta = 15$	$\beta = 20$
34	8	405.08	384.74	371.67	**335.88**
	9	244.30	**198.86**	238.60	285.97
	10	173.73	207.37	**136.80**	179.93
36	9	704.65	**567.89**	654.15	599.61
	10	419.87	**373.16**	489.71	542.00
	11	398.72	365.62	**314.26**	290.49

Table 6.1: Running time in seconds for different values of β.

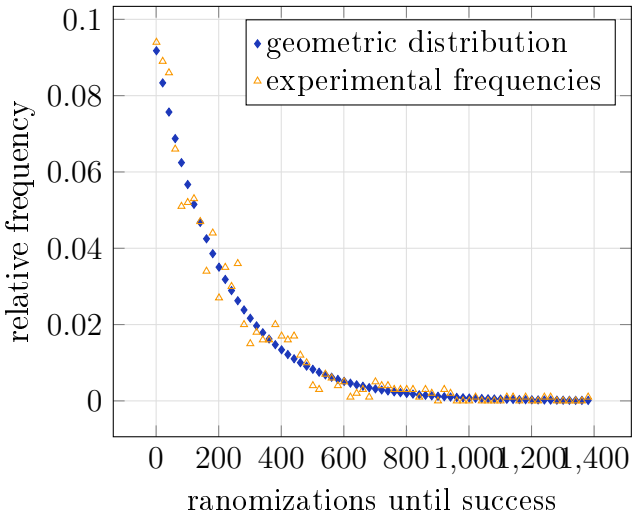


Figure 6.5: Number of used randomizations to find the solution for $e = 30$, $W = 2^7$

We confirm this distribution in an experiment for $e = 30$, in which case we have $N = 2^{e/2+1} = 2^{16}$ and use a hash table that can store up to $W = 2^7$ distinguished points. We then solved 1000 such instances and recorded for each the number of randomized versions of the function until the solution was found. On average, it took 208.28 versions compared to the approximation of $\frac{0.45N}{W} = 230.4$, despite slightly surpassing the $W \leq N/2^{10}$ limit where the vOW experiments took place. In Figure 6.5 we visualize the obtained frequencies (triangles) and give as comparison the probabilities of the geometric distribution with parameter $\frac{1}{208.27}$ (diamonds). In this figure we accumulated the frequencies in each interval of size 20 to allow for a better visualization.

6.5.2 Measuring the time per function evaluation

Next we measured the time per function evaluation that the GPU implementation requires on our hardware for different values of e. To pick our parameters, we first set W to the

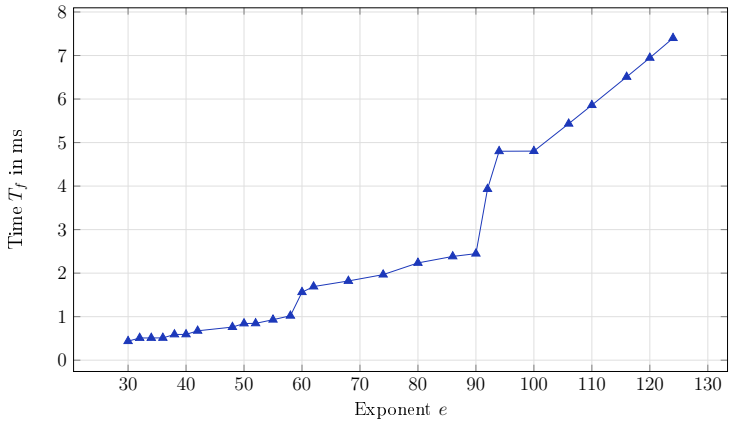


Figure 6.6: Cost per function evaluation using 6912 threads in parallel. Each data point is averaged over 4096 evaluations.

largest power of 2 such that the memory would not surpass our GPU's 80 GB budget, then chose the largest precomputation depth that would fit in the remaining memory. In the smaller instances, the memory and precomputation depth were additionally subject to a cap of $W \leq N/2^8$ and $d \leq e/4$ in the smaller instances. After performing the precomputation, we measured the time per function evaluation as illustrated in Figure 6.6. The jumps in the graph indicate that the bitsize of the used prime, which is roughly $2e$, exceeds the next 64-bit boundary. In those cases the prime occupies an additional register, which leads to a slowdown of the $\mathbb{F}_{p^2}$-arithmetic.

6.5.3 Performance estimation using a single GPU

Now, the measured timings allow us to estimate the time required by our implementation to solve larger instances. To compute this estimate we use Equation 6.3 with the measured value for T_f and the number of concurrent threads m used on the GPU. The resulting estimate is shown in Figure 6.7 (diamonds).

Note that the steeper incline in the estimation for $e > 62$ stems from the fact that for $e = 62$ we reach the maximum number of concurrent threads for our implementation, which we find to be $27,648$ threads. Further, from $e = 80$ onwards we additionally hit our hash table memory limit of $W = 2^{33}$ elements. We summarize in Table 6.2 optimal configurations for the *SIPFD* instances executed on our single GPU platform.

We also obtain an alternative estimate based on the time to finish one version of the random function in the full implementation of the attack. That is, we measure the time to obtain $10 \cdot W$ distinguished points and then multiply by the average number $\frac{0.45N}{W}$ of random functions needed. This method should capture the performance more accurately as it includes practical effects such as the memory access costs. For $e \leq 62$ we averaged 100 experiments of completing a random function, while for larger instances we decreased

e	30	32	34	36	38	40	42	48	50	52	56	62	68	74	80
d	9	9	10	11	11	12	12	14	14	15	16	17	19	21	22
$\log W$	8	9	10	11	12	13	14	17	18	19	21	24	27	30	33

Table 6.2: Optimal configurations for vOW on single GPU with 80GB memory. Configurations for $e > 80$ match the one of $e = 80$.

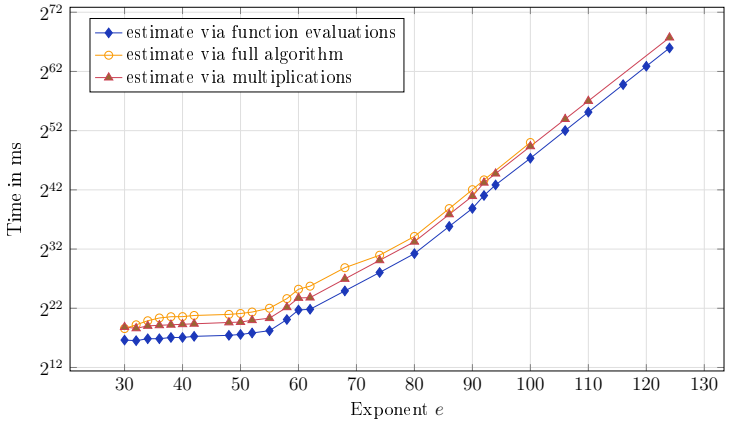


Figure 6.7: Estimated time to solve instances of *SIPFD* on a single GPU

the number of experiments and for $e \geq 76$ we only computed a $1/2^{10}$ fraction of the needed points and scaled the resulting time accordingly. The results of this second estimation are also shown in Figure 6.7 (circles) and present an overhead factor of about 8. This overhead is likely the result of imperfect parallelization speedups in GPUs, as well as the cost of memory accesses, but it is observed to decrease towards larger instances.

Finally, we benchmarked the average cost of field multiplications in our GPU setup to obtain a third estimate based on Equation 6.2, which is also presented in Figure 6.7 (triangles). This estimate closely matches the estimate via the full algorithm, especially for larger instances where distinguished points are rare and memory accesses are more sporadic.

Overall, our measurements support the use of any of the three methods described to obtain accurate extrapolations of the algorithm's running time. For a concrete example, we estimate that a problem with $e = 88$ which corresponds to the instance solved by Udovenko and Vitto in [44], would take about 44 years on a single GPU with 80GB memory limit. While this is not yet very impressive, compared to the 10 CPU years reported in [44], a single GPU is far less expensive and powerful than the 128TB network storage cluster used for that record. Therefore in the following section we give an estimate of the attack when scaling to a multiple GPU architecture.

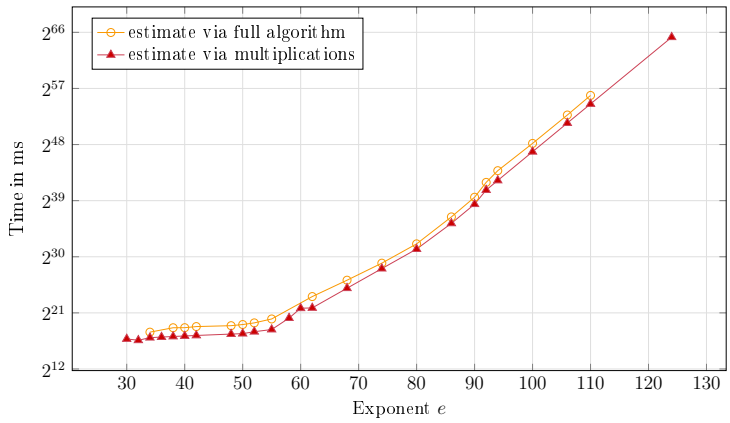


Figure 6.8: Estimated time to solve instances of *SIPFD* on 4 GPUs connected via an *NVLINK* bus

6.5.4 Multiple GPU estimation

We explored different strategies for parallelizing the vOW algorithm across multiple GPUs. In the first strategy, every GPU independently runs its own instantiation of the algorithm. The advantage of this approach lies its simplicity, which minimizes overhead since no communication between GPUs is necessary. On the downside, it provides only a linear speedup in the number of GPUs, since additional memory resources are not shared. In our second approach, GPUs report distinguished points to the same hash table, which is stored distributed over the global memory of all GPUs. The advantage here clearly lies in the increase of the overall memory, which allows to make use of the good time-memory trade-off behavior inherent to the vOW algorithm. However, this approach introduces a communication overhead due to the distributed memory access. On top of that, the data needs to be send over the slower *NVLINK* instead of the internal memory bus of the GPU.

We performed an extrapolation of the time to solve different sized instances in the distributed setting, similar to the extrapolation via the full algorithm in the single GPU setting. In this experiment, we allocated a hash table able to store up to $W = 2^{34}$ distinguished triplets, which for large instances corresponds to about 200GB, across the memory of four GPUs connected via an *NVLINK* bus. We then measured the time to collect and store a certain amount X of distinguished points. Multiplying this time by $\frac{10 \cdot W}{X} \cdot 0.45 \cdot \frac{2^{e/2+1}}{W} = 4.5 \cdot 2^{e/2+1}/X$, gives an extrapolation of the running time of completing the whole attack.

Figure 6.8 visualizes the obtained extrapolations (circles) in comparison to the estimate via the multiplication benchmark (triangles), i.e., using Equation 6.2. We observe, similar to the single GPU case, a slight underestimation by using Equation 6.2, which for larger instances vanishes. For the larger instances we obtain an underestimation by a factor

of roughly two, which corresponds to the performance difference of the *NVLINK* bus in comparison to the internal memory bus. However, since for larger instances with fixed memory budget the time to compute distinguished points dominates, the factor is expected to vanish. Hence, we finally conclude that using the distributed memory architecture does not lead to unexpected performance slowdowns.

Comparing both strategies. Let us determine, which of the parallelization strategies is preferable for a specific amount of GPUs. For large instances, the computational cost of the multi-GPU as well as the single-GPU setting, are well approximated by Equation 6.2. Therefore the speedup when parallelizing via distributed memory using X GPUs is

$$\frac{\mathrm{GCS}(e,\omega)}{\mathrm{GCS}(e,\omega+\log X)/X} = X^{3/2},$$

and, hence, preferable over the strategy via independent executions with a speedup of only X. Also, if comparing the exact numbers obtained from the estimate via the full algorithm in the distributed memory setting and the single GPU setting, we find that the distributed setting offers a better practical performance already for $e \geq 62$.

6.6 Extrapolating to cryptographic sizes

Based on our practical timings we estimate the time to solve an instance with $e = 88$ on 4 GPUs to about 32 GPU years in comparison to roughly 44 GPU years in the single GPU setting. Moreover, if we scale the attack to 16 GPUs, which is the maximum that the *NVLINK* bus currently supports, we estimate the time to only 5.6 GPU years, which means the experiment would finish in about 4 months. We therefore conclude from our experiments that for larger instances, with a memory budget of 128TB in the MitM case and 80GB per device in the GPU case, the vOW algorithm is the preferred choice.

In Figure 6.9 we visualize the result of the estimation via Equation 6.1 and 6.2 in both settings assuming 256 cores with 128TB of memory in the CPU case and 16 NVIDIA A100s connected via an *NVLINK* bus in the GPU case. This figure illustrates the estimate for running the MitM on the CPU (solid line) and the vOW on the GPU system (dashed line). We find that under these fixed resources, the break-even point from where vOW offers a better performance lies at $e = 96$. Additionally, we provide the estimate if we instead execute vOW on the corresponding CPU system (dash dotted line). Observe, that even under the unrealistic assumption that the 128TB of memory would allow for efficient random access (for the vOW hash table), it does not outperform the GPU based approach for any instance size. Moreover, even under this memory advantage in case of $e = 96$, the GPU implementation offers a speedup of almost two magnitudes (82x). We conclude that the way forward when tackling larger instances of the *SIPFD* clearly favors vOW implementations on GPU platforms.

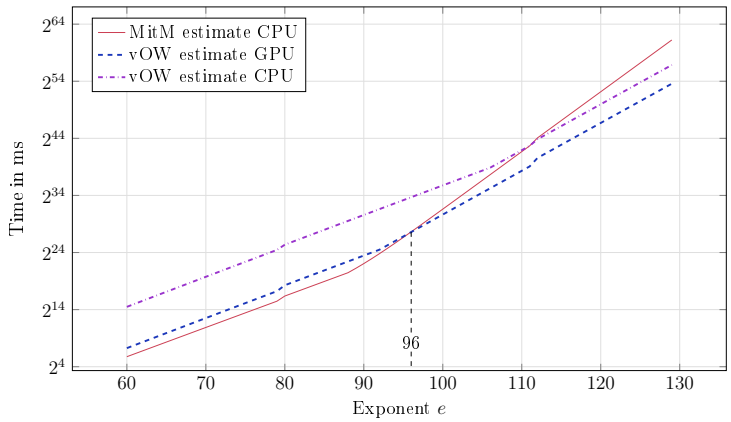


Figure 6.9: Estimated time to solve instances of *SIPFD* on 16 NVIDIA Ampere GPUs with 80GB each connected via an *NVLINK* bus in comparison to a cluster with 128TB storage and 256 cores.

The explicit formulas can also be used to derive cryptographically secure choices for concrete parameters. Assuming the 16 NVIDIA A100 setup, we find that an instance with $e = 190$ already has an expected cost as high as 2^{128} field multiplications. This shows that previous protocols have considerably overestimated parameter sizes, and more aggressive options are possible. For comparison, the SIKE proposal [15] had recommended $e = 216$ to target 128−bit security.

Chapter 7

Verifiable Isogeny Walks: Towards a post-quantum isogeny-based VDF

Abstract. *The work in this chapter, presented in collaboration with Francisco Rodríguez and Mehdi Tibouchi in Select Areas in Cryptography 2021 [6], investigates the problem of constructing postquantum-secure verifiable delay functions (VDFs), particularly based on supersingular isogenies. Isogeny-based VDF constructions have been proposed before, but since verification relies on pairings, they are broken by quantum computers. We propose an entirely different approach using succinct non-interactive arguments (SNARGs), but specifically tailored to the arithmetic structure of the isogeny setting to achieve good asymptotic efficiency. We obtain an isogeny-based VDF construction with postquantum security, quasi-logarithmic verification, and requiring no trusted setup. As a building block, we also construct non-interactive arguments for isogeny walks in the supersingular graph over $\mathbb{F}_{p^2}$, which may be of independent interest.*

A *Verifiable Delay Function* (VDF) is a cryptographic primitive first formalized by Boneh, Bonneau, Bünz and Fisch in 2019 [89], which has since gathered increasing interest due to its various applications such as power-efficient blockchains, benchmarking, and randomness beacons (these and other applications are discussed in [89, 80]).

The intuitive idea of a VDF is that it acts as a function whose value is uniquely determined at the moment that we pick an input, but no one is able to compute its output faster than a guaranteed prescribed wall-clock time T. To achieve this, it is crucial that the only known approaches for computing a VDF must be inherently sequential, such that no reasonable amount of parallelism could be effective on speeding up the VDF evaluation. At the same time, we also require the peculiar feature that the VDF's output must be efficiently and publicly verifiable, meaning that any other party can confirm its correctness, without relying on secret parameters nor on repeating the lengthy evaluation work that was required to produce it in the first place.

Constructing an ordinary delay function is a simple task, as it suffices to use T iterations of any function that can be composed with itself and whose output is unpredictable (such as a hash function). However, achieving an efficient verification is usually a much bigger challenge. For this, one may rely on general techniques for verifiable computation, specifically on *Succinct Non-interactive Arguments* (SNARGs) which allow for efficient proofs of any computation, where the time complexity of the proof construction is asymptotically close to that of the original computation, and its verification is polylogarithmic. This chapter presents a quantum-resistant VDF, whose evaluation involves isogeny walks over supersingular elliptic curves that can be publicly verified by means of a SNARG-based validation process.

In terms of quantum security, none of the current VDF constructions proposed as of today manage to achieve an exponential time gap between evaluation and verification while still being based on commonly studied postquantum assumptions. Our construction benefits from being derived from isogeny-based cryptography, which provides security guarantees that have been carefully scrutinized in the postquantum setting for over a decade. These studies add confidence in our security assumptions as well as in providing accurate estimates of how fast isogeny evaluations can be performed using optimized software and hardware libraries.

Previous Work The usage of SNARGs for constructing a VDF was first proposed by Boneh et al. [89], and independently by Döttling et al. [90]. The concept of verifiable computation branched out from probabilistic checkable proofs, as proposed by Babai et al. [91], and its development towards proofs that are short, efficient and non-interactive began with Micali's work [92].

In the context of verifiable delay functions, as of today the only isogeny-based construction was proposed by De Feo et al. in 2019 [80]. The main computational task for the evaluation of this VDF is that of finding images of points under a fixed large degree isogeny of a supersingular elliptic curve, whereas its verification essentially consists of performing a bilinear pairing computation. This verification is much more efficient than a SNARG-based verification, but the trade-off is that a quantum attacker can compute the VDF output by solving an associated discrete logarithm problem rather than going through the intended isogeny evaluation. Moreover, the construction has the added drawbacks of requiring a trusted setup and the setup itself being slow (requiring about as much time as an evaluation).

More recently, Leroux [93] proposed an isogeny-based verifiable random function that makes use of a proof of knowledge of a secret isogeny. While it is pointed out that this proof provides an exponential gap between prover and verifier, it cannot be adapted to a VDF since it only convinces the verifier that the prover knows *some* isogeny without any guarantee that the isogeny was somehow derived from an input. The evaluation of the random function actually maps the input to points, and then evaluates the images of those points, but this is not quantum-resistant unless it is treated as a single-use function.

Our Contribution We present an isogeny-based VDF that is free of the three main drawbacks suffered by the construction of De Feo et al.: the setup is fast, is not required to be trusted, and the construction is postquantum. By using a SNARG verification, we are able to obtain an evaluator with $\tilde{\mathcal{O}}(T)$ time complexity using $\mathcal{O}((\log T)^4)$ parallelism, and a quasi-logarithmic verifier with $\tilde{\mathcal{O}}((\log T)^{4+\log\log T})$ time complexity using no parallelism. This result is of interest not only due to its asymptotic complexity, but also because of the fact that it could directly benefit from future advances in SNARG constructions.

Since our VDF construction performs a random walk in the supersingular isogeny graph over $\mathbb{F}_{p^2}$, it can be seen as an instance of the Charles-Lauter-Goren hash function [31] augmented with a SNARG verification of this random walk. While there exist general-purpose SNARGs that can be applied for any computation in a straightforward fashion (see for example [94, 95]), we save as much as possible on overhead by specializing to the isogeny setting and constructing a SNARG over the field $\mathbb{F}_{p^2}$, which verifies the computation at the field-arithmetic level as opposed to the ALU-operation level. This implies that we have to generalize various SNARG results to work efficiently over a prescribed field, which leads us to present a framework that connects the SNARG with the isogeny walk setting in a natural way. Moreover, we describe the process for "ordering" the possible isogenies at each vertex of the isogeny graph so that the walk can be derived from an input string. This was never presented explicitly in [31], and in order to be compatible with our SNARG, our method selects an isogeny using only $\mathbb{F}_{p^2}$ arithmetic (i.e. we refrain from using arbitrary rules that look at the bit-representation of the field elements).

Organization The remainder of the chapter is organized as follows. Section 7.1 contains an overview and background of both isogeny-based cryptography and time-sensitive cryptography. In Section 7.2 we present the evaluation method of our isogeny-based delay function, and in Section 7.3 we present its SNARG-based verification. We then provide our security analysis in Section 7.4, and summarize our results in Section 7.5.

7.1 Background

In this section we present some basic definitions and background material used throughout the chapter.

7.1.1 Time-sensitive cryptography and Verifiable Delay Functions

Time-sensitive cryptography was first proposed in 1996 by Rivest, Shamir and Wagner [96]. The authors of [96], presented time-lock puzzle constructions that must be computed by performing a prescribed number of sequential squarings over an RSA modulus of unknown order. More recently, Lenstra and Wesolowski introduced in [97], a slow-timed hash function dubbed *sloth*. The evaluation of sloth is accomplished by the iterated com-

putation of a fixed number of sequential functions. Also, the notion of Proof of Sequential Work (PoSW) was introduced by Cohen and Pietrzak in Eurocrypt 2018 [98]. Then, Boneh, Bonneau, Bünz and Fisch formalized this branch of cryptography by rigorously defining verifiable delay functions.

A Verifiable Delay Function (VDF) as defined in [97, 89] is a function $f : \mathcal{X} \mapsto \mathcal{Y}$ that cannot be computed in less than a prescribed delay, regardless of the amount of parallelization available for its evaluation. At the same time, once a VDF has been computed, it can be easily verified by any third party, typically with the help of a companion proof produced during the evaluation. Moreover, the verification should be achievable with a limited amount of parallel cores, and ideally, by performing a single-core computation. Formally, a VDF is composed of three main algorithms:

- `Setup`: takes as input a security parameter λ and a delay parameter T and outputs public parameters pp.
- `Eval`: Takes a certain input x and public parameters pp and calculates an output y and a proof π.
- `Verify`: Takes as input x, y, π and pp and outputs 1 if and only if π is a valid proof for the input-output pair (x, y).

Moreover, a secure VDF satisfies the following properties:

- *Sequentiality*: The `eval` procedure can be completed in time $\mathcal{O}(T, \lambda)$ using $polylog(T)$ parallelism, but cannot be completed in time $o(T)$ even when $poly(T)$ parallelism is available.
- *Completeness*: An honest evaluation always causes the verifier to accept.
- *Soundness*: If y is not the output of $\texttt{Eval}(x, pp)$, then no PPT adversary can find a proof π such that the verifier accepts (x, y, π).

VDFs have important applications for Blockchain proof of work, space and stake [99], constructing a trustworthy randomness beacon [100], benchmarking of high-end servers and many more [101, 102]. Several examples of VDFs proposed in the literature can be found on [89, 80, 103, 104].

7.1.2 Isogeny-based VDFs

The only isogeny-based VDF construction proposed as of today is the one presented by De Feo, Masson, Petit, and Sanso in Asiacrypt 2019 [80].

The authors of [80], proposed an isogeny-based VDF where the evaluator must compute the image of a point under a large smooth-degree isogeny ϕ (consisting of the composition

of T isogenies each of prime degree ℓ), between two ℓ^T-isogenous supersingular curves E and E'. The order of the elliptic curves E and E' has a large prime divisor N, and their N-torsion subgroups are used for the verification via a pairing comparison using a point P with known image $\phi(P)$ (these points are public parameters and are computed at setup time).

This VDF construction has three important drawbacks. The first one is that it requires a trusted setup. This implies that a dishonest party computing the setup, can easily backdoor the function to make the evaluation much faster (as discussed in [80], knowledge of the random walk that generated E, can be used to compute the endomorphism ring of the curve E, which can be used to reduce the degree-ℓ^T isogeny to a shorter one). A second issue is that the setup computation is slow, taking as long as the delay from the evaluation itself. A third drawback is that the verification crucially depends on the computation of a pairing, which opens the door against any quantum attack targeting discrete logarithm computations.

7.1.3 VDFs from iterated sequential functions

Given an input parameter x, the authors of [89, 80] gave as an example of a naive VDF the chained computation of a one-way function as,

$$x_i = H(x_{i-1}), \text{ for } i = 1, \ldots, T,$$

with $x_0 = x$, and where the output $y = x_T$ can only be calculated sequentially independently of the amount of parallelism available for the evaluator. Notice however, that if the evaluator publishes some of the intermediate values x_i for $i = 1, \ldots, T$ (see Figure 7.1), then a verifier with access to many independent processors, can verify the work of the evaluator in a wall-clock delay significantly shorter than the time invested by the evaluator for producing y. This simple version of a VDF was discussed in [97, §3.1] as a *trivial design*, and later proposed by Yakovenko as a *Proof of History* consensus protocol with direct applications to blockchains [105]. Although this type of construction cannot achieve a polylogarithmic-time verification without requiring $poly(T)$ parallelism from the verifier, they are still sufficiently efficient for various applications. In particular, Yakovenko's Proof of History consensus protocol is massively used by the cryptocurrency Solana as its main consensus mechanism.

In order to obtain an asymptotically efficient verification without parallelism, the verification can be improved by means of verifiable computation. Verifiable computation can be used by the evaluator to compute a succinct non-interactive argument (SNARG), which certifies that a given computation was performed honestly. An important characteristic of a SNARG is that its verification can achieve a complexity that is logarithmic in the size of the original computation.

Both Boneh et al. [89] and Döttling et al. [90] proposed that any iterative sequential function can be augmented with a SNARG to produce an asymptotically efficient VDF.

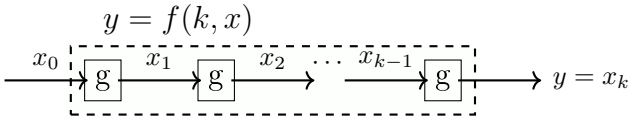


Figure 7.1: Illustration of an Iterated Sequential Function $f : \mathbb{N} \times \mathcal{X} \mapsto \mathcal{X}$, defined as $f(k, x) = g \circ g \circ \ldots \circ g$. In the figure, $x_0 = x$, and the output of the function $f = (k, x)$ is $y = x_k$.

This is precisely the steps that we follow for our isogeny-based VDF, with the iterative function being instantiated by a step in the supersingular isogeny graph and the SNARG being constructed at the $\mathbb{F}_{p^2}$ arithmetic level.

7.2 An Isogeny-based delay function

We now present an overview for the evaluation method of our VDF, leaving the SNARG-based verification for Section 7.3.

Our function involves computing a walk of length T in the 2-isogeny graph of supersingular curves over $\mathbb{F}_{p^2}$, where $p^2 \equiv 9 \mod 16$ (which is required for applying Kong's square-root algorithm [106]) and $p = poly(T)$ (which is required to make field arithmetic efficient for the verifier). The walk itself is determined from a string that is derived from the input to the VDF, starting from a prescribed initial curve, and the $j-$invariant of the final curve is taken as the output of the VDF. Therefore, our output exactly matches the output from an instantiation of the Charles-Goren-Lauter hash function [31], where the isogeny at each step is determined after assigning some ordering to the outgoing isogenies. In order to make the procedure suitable for a SNARG construction, however, we develop a procedure that makes this ordering not only explicit but also verifiable using only field arithmetic.

7.2.1 Evaluation overview

Given a delay parameter T, we ask the evaluator to compute a walk of length T on the 2-isogeny graph, where the exact path is determined by a string s and is non-backtracking. We would not be able to specify kernel points of order ℓ^T as in the SIDH setting, since doing so would require either $p = \mathcal{O}(\ell^T)$ or working over an $\mathcal{O}(T)$ field extension, both of which would make all field arithmetic inefficient for the verifier. Therefore, each step in the isogeny walk has to be determined "on the fly", and we chose to derive it from the modular polynomial root-finding problem since it is naturally expressed in $\mathbb{F}_{p^2}$ arithmetic.

Given two curves with j-invariants j_i and j_{i+1}, they are 2-isogenous over $\mathbb{F}_{p^2}$ if and only if the modular polynomial $\Phi_2(j_i, j_{i+1})$ vanishes. Thus, for fixed j_i, the next curve

in the path can be computed by finding a root of $\Phi_2(j_i, X)$. This is a cubic polynomial, but we can exploit the fact that we already know one of the roots (namely j_{i-1}, the previous curve in the walk) to factor out a linear term: if $X = j_{i-1}$ is a known root of $\Phi_2(X) = X^3 + aX^2 + bX + c$ then we can rewrite

$$\Phi_2(X) = (X - j_{i-1})(X^2 + (a + j_{i-1})X + b + aj_{i-1} + j_{i-1}^2)$$

and focus on finding the roots of the quadratic factor. This accomplishes three distinct goals:

1. It ensures the walk is non-backtracking by discarding the $X = j_{i-1}$ root
2. It reduces the root-finding problem to a quadratic equation (reducing the size of the computation yields heavy savings on SNARG overhead)
3. It enables the step in the walk to be defined by a canonical square-root along with a bit indicating its sign

Taking into account the explicit form of Φ_2, the other two roots are given by

$$j_{i+1} = \frac{1}{2}\left(j_i^2 - 1488j_i - j_{i-1} + 162000 \pm \sqrt{D_i}\right) \tag{7.1}$$

where

$$\begin{aligned} D_i = & j_i^4 - 2976j_i^3 + 2j_i^2 j_{i-1} + 2532192j_i^2 - 2976j_i j_{i-1} \\ & - 645205500j_i - 3j_{i-1}^2 + 324000j_{i-1} - 8748000000 \end{aligned} \tag{7.2}$$

The evaluator computes a canonical square root $S_i = \sqrt{D_i}$ using Kong's algorithm [106]: first fix any quadratic nonresidue d and precompute $t = d^{\frac{p^2-9}{8}}$, then compute

$$R_i = (2D_i)^{\frac{p^2-9}{16}} \tag{7.3}$$

and set

$$S_i = \begin{cases} R_i D_i (2D_i R_i^2 - 1) & \text{if } (2aR_i^2)^2 = -1 \\ R_i t d D_i (2R_i^2 t^2 d^2 D_i - 1) & \text{if } (2aR_i^2)^2 = +1 \end{cases} \tag{7.4}$$

which can be combined into

$$S_i = \left(\frac{1 - (2aR_i^2)^2}{2}\right) R_i D_i (2D_i R_i^2 - 1) + \left(\frac{1 + (2aR_i^2)^2}{2}\right) R_i t d D_i (2R_i^2 t^2 d^2 D_i - 1) \tag{7.5}$$

After the square root has been calculated, the evaluator uses the input string to choose the sign, yielding a deterministic process for the walk. Note that the input string cannot

have length $\mathcal{O}(T)$ (since the verifier must receive it and process it in time $polylog(T)$), so we will construct the signs pseudorandomly from a smaller string, as detailed in Section 7.3.1.

Note also that the evaluator could use any algorithm for computing square roots, but for the SNARG verification process it is convenient to use a procedure that is deterministic and also produces a fixed choice of sign, hence the choice of Kong's algorithm. For the verification process, the evaluator will keep track of j_i, D_i, R_i, S_i at each step and construct a SNARG that shows that equations (7.1), (7.2), (7.3) and (7.5) are satisfied.

Initial conditions. Since equation (7.1) requires knowledge of the $j-$invariant two steps into the past, we need to specify the first two curves as initial conditions. The 2-isogeny graph is a 3-regular graph without repeated edges or self-loops except for the two special vertices $j = 1728$ and $j = 0$. At $j = 1728$ there is one self-loop and two edges going to the vertex $j = 287496$, so we use the initial curve $j_0 = 287496$ and take $j_{-1} = 1728$ to avoid going back to the non-regular vertex. Note that the SNARG will only access values at indices 0 through $T - 1$, so we replace $j_{-1} = 1728$ by an equivalent condition on D_0 using equation (7.2).

7.3 SNARG-based Verification

In this section we deal with the verification process of the VDF, specifically how to fit the evaluation into a SNARG framework.

The notion of verifiable computation emanated from *Probabilistic Checkable Proofs* (PCP), a term first coined by Arora and Safra [107] to refer to a protocol between a prover who generates a proof of membership in a language (known as the PCP witness) for a given input and a randomized verifier which interacts with it. Both Babai et al. [91] and Feige et al. [108] independently proposed algorithms for transforming any $\mathcal{NP}$ witness into a PCP witness which, at the cost of making the verification probabilistic and the witness polynomially larger, allow for logarithmic-time verification by sampling only a logarithmic number of bits from the witness.

The results from [91] and [108] show that the history of any computation can be put into an alternate form which can be verified with an exponential speedup. In practice, however, these PCP constructions have two major drawbacks. First, they require interaction between the two parties throughout the protocol, and second, the amount of communication is still inefficient since the full PCP witness must be transmitted even if only a few bits of it are sampled. *Succinct Non-interactive Arguments* (SNARGs) are an alternative primitive which overcomes both limitations (they do not require interaction and are succinct in the sense that the witness is of logarithmic size). It was shown by Micali [92] that any PCP construction can be efficiently transform into a SNARG.

General-purpose SNARGs aim to verify a computation by working at the level of a RAM model and translating the correctness of the computation into either a circuit satisfiability

or an algebraic constraint problem. For our construction we have focused on the latter approach, and use a checksum-type PCP to verify said constraint problem.

In this section we review how the different construction ingredients are adapted to our particular problem. Section 7.3.1 describes the process of transforming the correctness of our computation into an algebraic constraint problem (known as *arithmetization*). Section 7.3.2 then gives a high-level summary for the rest of the SNARG construction and the complexities of the resulting verification scheme. Finally, Section 7.3.3 discusses the parallelization of the SNARG proof construction to obtain a concrete proof construction time close to the evaluation time.

7.3.1 Arithmetization

The sumcheck protocol that we work with is a PCP that verifies conditions of the form

$$\sum_{\vec{x}\in B^n} P(\vec{x}) = 0, \tag{7.6}$$

where P is a polynomial in n variables over some ring $R \supset B$.

To turn the verification of our VDF into an instance of this problem we *arithmetize* by storing the intermediate values into polynomials that act as lookup tables. Specifically, for the computation with T steps, we pick a base b and integer n such that $T \approx b^n$, and let $B = \{0, 1, \ldots, b-1\}$. The time steps $t \in \{0, 1, \ldots, T-1\}$ can then be expressed as $b-$ary strings of length n, where we refer with $t_{\vec{x}}$ to the integer represented by string $\vec{x} \in B^n$.

For $\vec{y} \in B^n$ we define the polynomial

$$\delta_{\vec{y}}(\vec{x}) = \prod_{j=0}^{n-1} \prod_{z\in B-\{y_j\}} \frac{x_j - z}{y_j - z} \tag{7.7}$$

which maps $\vec{x} \in B^n$ to 1 if $\vec{x} = \vec{y}$ and 0 if $\vec{x} \neq \vec{y}$. Regarding B as a subset of $\mathbb{F}_{p^2}$, the above formula can be seen as a polynomial over $(\mathbb{F}_{p^2})^n$, and is in fact the unique degree-$(b-1)$ polynomial that agrees with the δ function on B^n (note that throughout this chapter, the "degree" of a multivariate polynomial refers to the maximum degree of any individual variable).

The δ polynomial can be used as an auxiliary tool to select a specific index when summing over all indices. Given the sequence j_i of j-invariants at each step, we can define the polynomial

$$j(\vec{x}) = \sum_{y\in B^n} j_{t_{\vec{y}}} \delta_{\vec{y}}(\vec{x}). \tag{7.8}$$

This polynomial encodes the history of the computation since it maps $\vec{x} \in B^n$ to $j_{t_{\vec{x}}}$, but can also be evaluated (with less predictable outcome) over all of $(\mathbb{F}_{p^2})^n$. We can

then define similar polynomials $D(\vec{x})$, $R(\vec{x})$, $S(\vec{x})$ for the quantities D_i, R_i, S_i defined in Section 7.2, and also use the constants

$$L_{t_1,t_2} = \begin{cases} 1 & \text{if } \ t_2 = t_1 + 1 \\ 0 & \text{else} \end{cases}$$

to define the polynomial

$$L(\vec{x},\vec{y}) = \sum_{\vec{x}',\vec{y}' \in B^n} L_{t_{\vec{x}'},t_{\vec{y}'}} \delta_{\vec{x}'}(\vec{x}) \delta_{\vec{y}'}(\vec{y}) \tag{7.9}$$

which vanishes in B^{2n} unless $\vec{x}$ and $\vec{y}$ represent consecutive integers. Similarly to the δ polynomial, this polynomial will be used as a sequential counter that selects only consecutive indices when summing over all pairs of indices.

With this in hand, equations (7.1), (7.2), (7.3) and (7.5) can be represented as polynomial conditions. For instance, (7.1) becomes

$$P^j(\vec{x},\vec{y},\vec{z}) = 0 \quad \forall\ (\vec{x},\vec{y},\vec{z}) \in B^{3n}, \tag{7.10}$$

where

$$\begin{aligned} &P^j(\vec{x},\vec{y},\vec{z}) := \\ &\left[2j(\vec{z}) - j(\vec{y})^2 + 1488j(\vec{y}) + j(\vec{x}) - 162000 - s(\vec{y})S(\vec{y})\right] L(\vec{x},\vec{y})L(\vec{y},\vec{z}). \end{aligned}$$

Here, $s(\vec{x})$ represents the choice of sign at step $t_{\vec{x}}$, which we have also represented as a polynomial. We have not specified how the sign is derived from the input, but it will be necessary for the verifier to be able to efficiently evaluate $s(\vec{x})$. Therefore, we will take

$$s(\vec{x}) = \prod_{i=0}^{n-1} s_i(x_i), \tag{7.11}$$

where s_i are single-variable polynomials of degree $b-1$ mapping B to $\{+1,-1\}$. Since a polynomial of degree $b-1$ is determined by its values at any b points, we only need b bits to uniquely specify each s_i. This means that we use a total of nb bits to define $s(\vec{x})$, and we now define these bits as the input to the VDF (the input bits can be passed through a hash function first to enforce pseudorandomness of the resulting polynomial).

We then turn equations (7.2) and (7.5) into polynomial conditions

$$P^D(\vec{x},\vec{y}) = 0 \quad \forall\ (\vec{x},\vec{y}) \in B^{2n}$$

and

$$P^S(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n$$

in an analogous way.

As for equation (7.3), the polynomial that we would obtain by arithmetizing it directly would be of large degree, which is undesirable for the SNARG construction (the sumcheck protocol verification, described in [109], is linear in the degree of the polynomial). Therefore, we introduce additional state and break the exponentiation down into a right-to-left strategy: let $K = \lceil \log((p^2-9)/16) \rceil$ and $e_0, e_1, \ldots, e_{K-1}$ be the bits of $(p^2-9)/16$. We define $R_i^{(0)} = 1$, $D_i^{(0)} = D_i$ and for $0 < k < K$,

$$D_i^{(k)} = (D_i^{(k-1)})^2, \tag{7.12}$$

and

$$R_i^{(k)} = R_i^{(k-1)} (D_i^{(k-1)})^{e_k} \tag{7.13}$$

so that $R_i^{(k-1)} = R_i$. For each $0 \leq k < K$ we define polynomials $D^{(k)}(\vec{x})$ and $R^{(k)}(\vec{x})$ from the values of $D_i^{(k)}$ and $R_i^{(k)}$, respectively, and use them to write polynomial conditions

$$P^{R,k}(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n$$

and

$$P^{D,k}(\vec{x}) = 0 \quad \forall\ \vec{x} \in B^n.$$

Note that we now have $K = \log p$ pairs of polynomials to work with, but each being of degree $d = \mathcal{O}(b)$ just as P^j, P^D and P^S (this is inherited from the degree of the δ polynomial).

Finally, to wrap the whole verification into the form of (7.6), we use the weighted sum

$$\begin{aligned} P(\vec{x},\vec{y},\vec{z}) = {} & w^J(\vec{x},\vec{y},\vec{z}) P^J(\vec{x},\vec{y},\vec{z}) \\ & + w^D(\vec{x},\vec{y}) P^D(\vec{x},\vec{y}) + w^S(\vec{x}) P^S(\vec{x}) \\ & + \sum_k \left(w^{D,k}(\vec{x}) P^{D,k} + w^{S,k}(\vec{x}) P^{S,k} \right), \end{aligned} \tag{7.14}$$

where each w is a weight polynomial. These polynomials are defined in the same way as $s(\vec{x})$ in equation (7.11), but are chosen randomly by the verifier so that proving that

$$\sum_{\vec{x},\vec{y},\vec{z} \in B^n} P(\vec{x},\vec{y},\vec{z}) = 0$$

is enough to convince the verifier that $P^J, P^D, P^S, P^{k,D}, P^{k,S}$ vanish at all points, and hence that the whole computation is correct.

We stress that general-purpose SNARGs exist that can be applied to any program (see for example [94] and [110]), but they usually perform arithmetization at the ALU level, which increases the overhead cost. For instance, they may have lookup polynomials that encode the values of CPU registers at each time step and obtain polynomial conditions

that represent the correctness of ALU operations (which in our case would include even the breakdown of all $\mathbb{F}_{p^2}$ arithmetic into basic ALU operations). The arithmetization that we propose is performed directly at the field arithmetic level, meaning that the values encoded into our polynomials are $\mathbb{F}_{p^2}$ elements and our polynomial conditions represent equality over this field, without actually including the correctness of $\mathbb{F}_{p^2}$ arithmetic procedures in the SNARG proof since it is within the verifier's capabilities to perform them directly.

7.3.2 Overview of the SNARG construction

We now present a high-level summary of the steps required to complete the SNARG construction. For a more detailed account, the reader may consult the appendix in the extended version of this work [111].

We have reduced the verification process to the verification of a condition of the form[1]

$$\sum_{\vec{x}\in B^n} P(\vec{x}) = 0,$$

where P is of degree $d = \mathcal{O}(b)$ in n variables, and $T = n^b$. This verification is handled by the sumcheck protocol of Lund, Fortnow, Karloff and Nisan [109], detailed in the appendix of [111]. The sumcheck protocol is a PCP that reduces the problem to the verifier's ability to evaluate P at a random point $\vec{x} \in (\mathbb{F}_{p^2})^n$. To enable this, the prover publishes a PCP witness that contains the table of values for each of the polynomials $j(\vec{x}), D(\vec{x}), S(\vec{x}), R^{(k)}(\vec{x}), S^{(k)}(\vec{x})$ in all of $(\mathbb{F}_{p^2})^n$. The table of values for $L(\vec{x}, \vec{y})$ is assumed to be precomputed and publicly available (since it is independent of the input), while the polynomials for the sign and the random weights are directly evaluated by the verifier.

The next step is to apply Micali's transform [92], which both eliminates interactiveness and shortens the proof to a polylogarithmic size. The core idea is to encode the tables of values into a Merkle tree and publish only the root of the tree as a commitment, answering specific queries with a value along with its verification path in the Merkle tree. We then apply the Fiat-Shamir transform [112] to replace all random choices from the verifier (including the choice of the weight polynomials in (7.14)).

The resulting verification scheme has a prover time complexity of $\mathcal{O}((n^2b)^n b \log b)$ with $\mathcal{O}(n^2 \log p)$ parallelism and space complexity of $\mathcal{O}((n^2b)^n \log p)$ field elements (from computing and storing the tables of values), and a verifier time complexity of $\mathcal{O}(n^3 b \log(nb) \log p)$ with no parallelization nor significant storage requirements (from performing a *degree test* on said table). Both of these complexities are derived in the appendix of the extended version of this work [111]. We argue in Section 7.4 that a choice of $p = poly(T)$ is natural, so the evaluator parallelism is $polylog(T)$.

There is still some freedom regarding the choice of parameters n and b, since they only

[1] For ease of notation we collapse $\vec{x}, \vec{y}, \vec{z}$ into a single vector, implicitly substituting $n \mapsto 3n$ throughout.

need to satisfy $n^b \approx T$. Choosing $b \approx (\log T)^{1/\epsilon}$ for some $\epsilon > 0$ means that

$$n \approx \frac{\log T}{\log b} = \frac{\epsilon \log T}{\log \log T},$$

and

$$n^n \approx \left(\frac{\epsilon \log T}{\log \log T}\right)^{\frac{\epsilon \log T}{\log \log T}} < \log T^{\frac{\epsilon \log T}{\log \log T}} = T^\epsilon,$$

so the prover complexity becomes $\tilde{\mathcal{O}}(T^{1+2\epsilon})$ time (which can be made arbitrarily close to linear) and $\mathcal{O}(T^{1+2\epsilon} \log p)$ space, while the verification complexity becomes $\tilde{\mathcal{O}}((\log T)^{4+1/\epsilon})$.

Another option is to chose a slowly decreasing function $\epsilon = 1/\log \log T$. This causes the proof construction to be strictly quasi-linear, but at the cost of making the verification only quasi-polylogarithmic.

7.3.3 Parallelization of the Proof Construction

It is also desirable for the evaluation algorithm of a VDF to take time that is concretely (as opposed to asymptotically) close to T. Both Boneh et. al. [89] and Döttling et. al. [90] independently proposed similar methods for exploiting parallelism to finish evaluation of an iterative function and its SNARG proof construction at the same time by working with subsegments of geometrically decreasing size. Assuming that computing the proof is slower than the function evaluation by a factor of α, the evaluator stops after completing $T/(1+\alpha)$ iterations of the computation and then starts a proof for this partial computation in parallel with the remaining $\alpha T/(1+\alpha)$ steps. This is repeated recursively, so the evaluator does proof constructions of size $T\left(\frac{\alpha}{1+\alpha}\right)^i$ for $i = 1, 2, 3, \ldots$ until $i \approx \log(T)/\log(1+\frac{1}{\alpha})$ when approximately a single step remains and it can be computed directly by the verifier without proof. Since $\log(1+\frac{1}{\alpha}) = \frac{1}{\alpha} + \mathcal{O}(\frac{1}{\alpha^2})$, this increases the parallelization requirement by a factor of $\alpha \log T$.

Note that in the case when $b = (\log T)^{1/\epsilon}$ for constant ϵ this results in an amount of parallelism polynomial in T, which is coined a *weak VDF* by Boneh et al. [89, Definition 5]. We favor the case when $\epsilon = 1/\log \log T$ instead, which means the parallelism is strictly logarithmic at the cost of making the verification slightly slower (quasi-polylogarithmic).

7.4 Security Analysis

The soundness of the VDF relies entirely on that of the SNARG proof, which is discussed in [92]. In this section we discuss the other crucial security property of a VDF, namely its sequentiality.

As a side note, we point out that any protocol where the isogeny walk is not prescribed in some way is insecure in terms of sequentiality. For instance, one could have asked the evaluator for a SNARG proof of *any* large-degree isogeny and naively hope that this makes

for a good proof of sequential work even if the output is not unique. However, much like the proof of isogeny knowledge proposed by Leroux [93], this does not constitute proof of a sequential computation if the evaluator is free to choose the path. Indeed, even if backtracking is avoided it has been shown by Adj, Ahmadi and Menezes [113] that it is easy to find cycles of any length in the $\ell-$isogeny graph, which allows a cheating evaluator to construct a SNARG for a "long" walk by repeating a short cycle as many times as necessary. Of course, the prescribed walks in our construction may still contain cycles, but this is not a problem so long as they cannot be forced nor predicted by the evaluator.

In our context, sequentiality relies on a similar version of the *Isogeny Shortcut Problem* defined by De Feo et al. [80] :

Problem 7.4.1 (Isogeny Shortcut Problem)**.** *Let $E/\mathbb{F}_p$ be a random supersingular elliptic curve and $\phi : E \to E'$ an isogeny of degree ℓ^T to a curve $E'/\mathbb{F}_{p^2}$. After a precomputation time $poly(T, \lambda)$, find the image of a given point whose order is coprime to ℓ in time $o(T)$.*

Our setting differs from the one in this problem in three important ways:

1. Our problem is not to find images of points, but codomain curves. Since the codomain curve can be computed from any three point evaluations, the problem in our setting could be considered more general. However, all known point-evaluation methods have complexities asymptotically equal to those of codomain-evaluation, so we do not expect our security assumption to be significantly stronger.

2. The precomputation time that we allow in our setting is granted before learning the isogeny to be evaluated, which reflects the fact that our VDF uses a different isogeny for each input as opposed to fixing the isogeny at setup time. In this regard, our security is stronger since it relies on a much weaker assumption.

3. We do not assume that the starting curve was randomly sampled, which is done in [80] to prevent shortcut attacks (see below) when the endomorphism ring is known. However, we argue that such attacks are unimportant in our setting precisely due to the previous point. Starting from a public curve means we do not need a trusted setup.

Despite these differences, our security analysis is very similar to De Feo's et al., as we still distinguish two types of attacks: either finding a way to perform the isogeny walk faster (possibly exploiting parallelism), or attempting to find an equivalent isogeny walk of smaller degree (which we call *isogeny shortcuts*).

7.4.1 Faster Isogenies

The best known method for computing a degree-ℓ^T isogeny is by sequentially performing the composition of T consecutive ℓ-isogenies, where each ℓ-isogeny is computed using

Vélu's formulas. We consider the time of this computation as a lower bound for the delay parameter offered by our VDF, even though our specification of the evaluation algorithm does not use kernel points. While Vélu's formulas parallelize almost perfectly, attempting to directly evaluate an isogeny of degree ℓ^T in time $O(T)$, would require an amount of parallelism exponential in T which exceeds the evaluator's capabilities. This is leaving aside the fact that our isogeny is not readily presented as a kernel that could be plugged directly into these formulas, and even if such a kernel could be obtained it would be defined over a degree-T field extension. Assuming then that the best strategy is to decompose into small-degree isogenies, it is unlikely that any algorithm would be able to compute the composition of all such isogenies without going through each intermediate curve.

While none of the above premises are often studied as security assumptions, they are long-standing conjectures that have endured the test of time, despite considerable incentives to optimize isogeny evaluations. For instance, finding a way to compute an isogeny of degree ℓ^T in linear time without exploiting the smooth decomposition would be equivalent to computing an arbitrary-degree isogeny in time logarithmic in the degree, which would be ground-breaking for all isogeny-based cryptography.

It should be noted that recent improvements to the evaluation of $\ell-$isogenies do exist, most notably the algorithm of Bernstein et al. [25], which achieves a square-root time complexity improvement over the previous state-of-the-art. Although this does not affect the asymptotic complexity of computing T isogenies in series, having the possibility for variations in the concrete cost is still problematic for a VDF. However, the gains in these new formulas are asymptotic in ℓ and we only use $2-$isogenies for which the formulas are so simple that they can be conjectured to be already optimal.

7.4.2 Isogeny Shortcuts

The second possibility is for the evaluator to produce a different isogeny path between the two end curves, and hope that it is shorter than the original. Because construction of the proof requires the evaluator to know each of the j-invariants in the original isogeny path, one might mistakenly assume that such an attack would be useless. However, an attacker that is able to predict the output in a shorter time, even if unable to produce a proof for it, would still violate the security of the VDF.

A shorter path always exists because the isogeny graph is of size $\mathcal{O}(p)$ and the optimal expander property of Ramanujan graphs [27] implies the distance between any two curves is bounded by $\mathcal{O}(\log p)$ (which is necessarily logarithmic in T, otherwise all field arithmetic would be inefficient). However, the sequentiality property only requires that such path cannot be found in time $o(T)$. In the case of arbitrary curves, when the endomorphism ring is not known, the best one can do is to try to solve the isogeny problem between the curves, disregarding the already known isogeny. The best algorithm for this is a birthday attack, which takes $\mathcal{O}(p^{1/4})$ time using a quantum Grover search [40]. This sets a theoretic limit on the admissible delay parameters of $T = \mathcal{O}(p^{1/4})$. However, it should be noted

that even this attack would not apply directly since it requires previous knowledge of the codomain curve.

In our construction, we take an additional risk of fixing the starting curve $j = 1728$ where not only is the endomorphism ring known, but also elements of norm ℓ^n can be easily found with the KLPT algorithm [114]. This leads to an attack analogous to the CGL hash collision-finding algorithm that was described in [115] and optimized by [80], which computes a shorter isogeny in time $poly(T, \log p)$. One could always start from a curve of unknown endomorphism ring to avoid this attack at the cost of requiring a trusted setup, but in our case even this attack is admissible because there is a different isogeny used in each input and computing the shortcut is still slower (in the VDF from De Feo et al. [80], this is more of a problem because the same isogeny is always used, so the shortcut breaks the VDF for all inputs after being computed once).

More generally, it is unlikely that any kind of reduction could be computed fast enough since our isogeny is not readily represented as an ideal in the quaternion algebra. Any reduction that works over the endomorphism ring would have to translate the isogeny at each step into the language of quaternion ideals, necessarily resulting in a $\Omega(T)$ complexity, and the concrete time of whatever parsing need to be performed is unlikely to be much faster than a simple degree-2 isogeny.

7.5 Discussion

We have presented a framework for applying a SNARG at the field-arithmetic level to verify an isogeny walk, and used it to obtain a postquantum isogeny-based VDF that does not require a trusted setup and is less susceptible to isogeny shortcut attacks since it uses a different isogeny walk for each input.

In terms of asymptotic complexity, our VDF is less efficient relative to other constructions: for example, the VDF from De Feo et al. [80] has $\mathcal{O}(T)$ evaluator-space complexity and verification time constant in T. However, no previous VDF construction achieves post-quantum security.

Although SNARG-based VDFs have been deemed mainly of theoretical interest by Boneh et al. [89], alluding to the fact that current SNARG constructions have concrete costs about 100,000 times larger than the original computation, it should be noted that this is the case for general-purpose constructions which verify every step of the computation at the bit-operation level. Since our construction verifies steps of the computation at the field-arithmetic level, it has the potential to save significantly on overhead. Therefore, it could prove an interesting future work to implement and benchmark our construction.

We leave it also as future work to propose concrete parameters for our construction. We stress that definitions such as 128-bit security level are not meaningful for the sequentiality property of a VDF, whereas for soundness our security is completely derived from Micali's work [92] and based on symmetric cryptography. This means that the choice of parameters

should be based only on the desired delay time, which is impossible to fine-tune until a working implementation is obtained.

We also point out that there is a factor of $\log p$ in both the evaluator's parallelism complexity and the verifier's total complexity that emanates from the breakdown of an exponentiation to verify a square-root computation, as represented by equations (7.12) and (7.13). If the square-root computation was verified via a squaring rather than repeating the computation, this factor could be eliminated. However, the longer computation is required due to the fact that we perform our arithmetization at the field arithmetic level, meaning that we need a deterministic way of picking a sign that uses only field operations. We leave it as an open question whether one can design a method to, given both roots of a quadratic polynomial, choose one of them deterministically using only field arithmetic and without resorting to a large exponentiation.

Finally, it should also be noted that the SNARG mechanism we have described is fairly rudimentary and there are various recent developments that achieve slight optimizations. However, most of these improvements rely on reducing the arithmetic over ad hoc fields (such as [110], which uses a binary field) whereas our SNARG is constrained to work in the field of the elliptic curve. We note that this problem is likely to be ubiquitous when adapting SNARGs to work with existing cryptographic frameworks, since such frameworks usually include arithmetic over a prescribed field. Working directly over this prescribed field is bound to save significantly on overhead when constructing SNARG proofs, so we also encourage further optimizations of SNARG constructions over arbitrary fields.

Chapter 8

SwiftEC: An efficient encoding for ordinary elliptic curves

Abstract. *The work in this chapter, published in ASIACRYPT 2022 in collaboration with Franchisco Rodríguez and Mehdi Tibouchi [7], presents a novel construction for a uniform encoding of points in ordinary elliptic curves and an associated hash function. This construction has the perk of requiring a single exponentiation in the curve's field as opposed to all previous constructions which required at least two, making it the most efficient known way of encoding and hashing into a large class of ordinary curves. Since it works over ordinary curves, the construction is not interesting for post-quantum applications. However, it is closely related to our work in that it still utilizes the isogeny framework, and it produces a result that is of great immediate interest for classical applications being used today.*

Indifferentiable hashing to elliptic curves. Numerous cryptographic primitives and protocols constructed over elliptic curve groups involve *hashing* to an elliptic curve: they assume the existence of a public function $\mathfrak{H}$ mapping arbitrary bit strings to elliptic curve points / group elements. Moreover, the function $\mathfrak{H}$ is supposed to behave "like a random oracle". Such a functionality is required for example for many password-authenticated key exchange protocols, identity-based encryption schemes, short signature schemes, verifiable random functions, oblivious PRFs and more. It is therefore important to understand how it can be efficiently instantiated in practice, and moreover with constant-time implementations, since the data that is hashed to the curve is often sensitive and can thus be compromised by timing side-channel attacks. This problem is in fact currently the subject of an IETF standardization effort within the Crypto Forum Research Group [116].

It became an active research topic about a decade ago, particularly after the work of Brier et al. [117], which applied Maurer et al.'s *indifferentiability* framework [118] to properly formalize what it meant for $\mathfrak{H}$ to "behave like a random oracle", and proposed several constructions satisfying the required properties. The design paradigm that emerged at the

time as the main approach to hashing to elliptic curve groups combines so-called *encoding functions* to the elliptic curve, which are algebraic (or piecewise algebraic) maps from the base field to the group of points on the curve, with random oracles to the base field and other sets that are "easy to hash to", as well as simple arithmetic operations on the curve.

More precisely, consider for instance[1] the problem of hashing to the subgroup $\mathbb{G}$ of cofactor h in $E(\mathbb{F}_q)$, where E is an elliptic curve defined over the finite field $\mathbb{F}_q$ and such that $E(\mathbb{F}_q)$ is cyclic of order n with generator P. Then Brier et al. [117] showed that the following construction:

$$\mathfrak{H}_{\text{slow}}(m) = [h] \cdot \Big(f\big(\mathfrak{h}_1(m)\big) + [\mathfrak{h}_2(m)]P \Big) \tag{8.1}$$

is *indifferentiable from a random oracle* when $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are modeled as independent random oracles to $\mathbb{F}_q$ and $\mathbb{Z}/n\mathbb{Z}$ respectively (which are easy to realize, heuristically, using bitstring-valued hash functions) and $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ is a mapping (the encoding function) satisfying mild conditions. This means that whenever[2] a cryptographic scheme or protocol is proved secure in the random oracle model with respect to a $\mathbb{G}$-valued random oracle $\mathfrak{H}$, that random oracle can be instantiated securely with the construction $\mathfrak{H}_{\text{slow}}$.

As we have mentioned, the construction above requires a suitable *encoding function* $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$. A number of candidates were known at the time for various classes of elliptic curves, such as those of Shallue and van de Woestijne [52], Ulas [120] or Icart [121], and many more have been proposed since [122, 123, 124, 125, 126, 127, 128]. All of them can be computed in constant time at the cost of one full size exponentiation in $\mathbb{F}_q$ (typically a square root or cube root computation), which dominates the complexity, plus a few other less costly operations in the field, like multiplications, inversions and Jacobi symbol computations.

In contrast, the second term of $\mathfrak{H}_{\text{slow}}$ is a full-size *scalar multiplication* over the curve, which typically exceeds the computationally cost of a field exponentiation by a factor of 10 or more depending on base field size and curve arithmetic. This makes $\mathfrak{H}_{\text{slow}}$ a fairly inefficient construction.

To alleviate this issue, Brier et al. also proved that the following construction is also indifferentiable from a random oracle:

$$\mathfrak{H}_{\text{square}}(m) = [h] \cdot \Big(f\big(\mathfrak{h}_1(m)\big) + f\big(\mathfrak{h}_2(m)\big) \Big) \tag{8.2}$$

when $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are modeled as independent random oracles to $\mathbb{F}_q$, and when f is specifically Icart's function. The result was later extended by Farashahi et al. [129], who showed that basically all of the known encoding functions f could also be plugged into that construction. This provides indifferentiable hashing to arbitrary elliptic curves at the cost of essentially *two* base fields exponentiations.

[1] The general case of a non-cyclic $E(\mathbb{F}_q)$ can be treated similarly. We refer to Brier et al. [117] for details.

[2] Technically, this holds in the case of *single-stage* security games, as clarified by Ristenpart et al. [119]. This limitation is rarely of concern in our context.

On the other hand, in certain primitives and protocols proved secure with respect to a $\mathbb{G}$-valued random oracle $\mathfrak{H}$, one can show that $\mathfrak{H}$ can be securely instantiated using the following simpler construction:

$$\mathfrak{H}_{\text{non-unif}}(m) = [h] \cdot f\big(\mathfrak{h}(m)\big) \tag{8.3}$$

where $\mathfrak{h}$ is modeled as a random oracle to $\mathbb{F}_q$. This construction is not nearly as well-behaved as (8.2). In fact, f usually only reaches a fraction of the points on $E(\mathbb{F}_q)$, and induces a non-uniform distribution over its image, so that $\mathfrak{H}_{\text{non-unif}}$ can typically be efficiently distinguished from a random oracle, and in particular it is *not* indifferentiable in the sense discussed so far. Nevertheless, certain primitives and protocols do not require the full strength of indifferentiability, and $\mathfrak{H}_{\text{non-unif}}$ is sometimes sufficient to let their security proofs go through.

A rough idea of why this happens is that, in a random oracle proof of security, the simulator generally wants to program the random oracle by setting the hash of some message m to a value Q, but that point Q itself can usually be anything depending on some randomness. So assuming that $h = 1$, the simulator might typically want to set $\mathfrak{H}(m)$ to $Q = [r] \cdot P$ for some random r, say. Now if $\mathfrak{H}$ is defined in the protocol using a construction like (8.3), the simulator would pick a random r and set $\mathfrak{h}(m)$ to one of the preimages $u \in f^{-1}(P)$ if $P \in f(\mathbb{F}_q)$. If however P is not in the image of f, the simulator would pick another random r and try again.

Therefore, construction (8.3), while less general and well-behaved than (8.2), is sometimes good enough for security at half the computational cost. This is a substantial difference in terms of efficiency that practitioners may be sensitive to, so much so that *both* of these constructions are in fact proposed in the current IETF draft [116]. Construction (8.3), however, comes with the caveats that applications using it "SHOULD carefully analyze the security implications of nonuniformity", and that "cryptographic protocols whose security analysis relies on a random oracle that outputs points with a uniform distribution MUST NOT" use it. This results in the somewhat unfortunate situation that implementers have to choose between two approaches for implementing hashing to elliptic curves: one which is secure in all cases but slower, and one which is faster but requires a careful analysis to ascertain that it does not fully compromise the security of the scheme.

The quest for fast indifferentiable hashing. Ideally, one would prefer to have the best of both worlds: indifferentiable hashing at the cost of a single exponentiation in the base field instead of two. Obtaining this for general elliptic curves is a long-standing open problem.

In special cases, solutions exist: this is particularly the case for supersingular curves of j-invariant 0 and 1728, for which it has long been known [130, 123] that an "almost bijective" encoding function f exists; it is then easy to check that plugging that f into construction (8.3) does achieve indifferentiability. Unfortunately, those types of supersingular curves, which were popular to reach the 80-bit security level in pairing applications in the

early 2000s, are no longer used today due to exceedingly large parameters at higher security levels. Moreover, there are strong reasons to believe that almost bijective encodings cannot exist for general elliptic curves [131].

Progress towards adressing the general open problem was made by Tibouchi and Kim [132], who extended the statistical results of Farashahi et al., and established in particular that, asymptotically, it was possible to achieve indifferentiable hashing at a cost of less than two exponentiations by tweaking construction (8.1) with a random oracle $\mathfrak{h}_2$ mapping to a short interval. That result is mostly of theoretical significance, however, since it requires very large base fields to provide meaningful error bounds.

Recently, Koshelev [133] made a practically significant advance, by showing that indifferentiable hashing at the cost of a single exponentiation was possible for certain *ordinary* curves of j-invariant 0 over suitable base fields. This is still a negligible fraction of all elliptic curves, but it is practically relevant since it includes pairing-friendly curves like some of the BLS12 used today. Koshelev's approach is also the first one considered in the last decade or so that substantially departs from the framework of constructions (8.1)–(8.3) above. While those earlier techniques reduce the problem of indifferentiable hashing to the encoding function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$, which is defined over a one-dimensional domain, Koshelev bases its construction on a map $F\colon \mathbb{F}_q^2 \to E(\mathbb{F}_q)$ with a two-dimensional range. Looking back at Brier et al.'s original proof for the indifferentiability of construction (8.2) using Icart's encoding function, this is fairly natural (since that proof was constructed around a two-dimensional argument), but it is an important shift in perspective.

In this chapter, we use a similar idea (albeit very different techniques) to settle the open problem for a large class of elliptic curves: for essentially all curves over fields $\mathbb{F}_q$ with $q \equiv 1 \pmod 3$ with either odd order or order divisible by 4 (this includes almost all elliptic curves in current use), we are able to construct a new indifferentiable hashing, which we call SwiftEC, at the cost of a single exponentiation in the base field.

Representing points as uniform random strings. A very different problem, but which has been tackled using similar techniques, was introduced in Bernstein et al.'s *Elligator* paper [127]: the problem of representing a uniform point on $E(\mathbb{F}_q)$ in a public way as a close to uniform random bit string. The stated goal was to achieve a form of steganography for censorship circumvention. Indeed, network traffic containing points on a certain elliptic curve (e.g. public keys for encryption or signature) represented in usual ways (either as full coordinates (x, y), in compressed form $(x, \operatorname{sgn} y)$ or in x-only form) can be easily distinguished from random, which may lead to automated traffic interruption or targeted surveillance.

As a countermeasure, Bernstein et al. suggested to use an encoding function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ with the property that it maps an interval $I \subset \mathbb{F}_q$ of length $\approx q/2$ *injectively* into $E(\mathbb{F}_q)$. Then, any point in $f(I)$ can be represented by its unique preimage under f in I. In particular, if q is close to a power of two, this readily gives a simple representation of random elements in $f(I) \subset E(\mathbb{F}_q)$ as uniform random bit strings (and when q is far from

a power of two, it suffices to represent elements of I as uniform random bit strings, which can be easily done by expanding the representation and introducing randomness).

This approach has two drawbacks. First, suitable encodings f that are injective over a large interval are hard to construct, and only known for limited families of elliptic curves [124, 126, 127], all of order divisible by 3 or 4 (and hence not including curves of prime order, for example). Second, one needs to address the issue of points falling outside $f(I)$. Since the goal is to represent *random* points on $E(\mathbb{F}_q)$ as bit strings, the assumption is that in the cryptographic protocol under consideration, the point to represent is obtained by some sort of random process, and it is possible to use rejection sampling until reaching $f(I)$. Since the image size covers roughly half of all points on the curve, this will require about two iterations on average, often an acceptable cost. However, if the process generating the point is expensive, rejecting may be less than ideal.

Tibouchi's Elligator Squared paper [53] addressed these shortcomings by, in essence, applied construction (8.2) above "in reverse". One of the key properties that makes construction (8.2) an indifferentiable hash function is the fact that, for an encoding function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$, the following map:

$$\begin{aligned} f^{\otimes 2}\colon \mathbb{F}_q^2 &\to E(\mathbb{F}_q) \\ (u, v) &\mapsto f(u) + f(v) \end{aligned} \tag{8.4}$$

induces a close-to-uniform distribution on its image. In particular, a uniformly random preimage of a uniformly random point in $E(\mathbb{F}_q)$ is close to uniform in $\mathbb{F}_q^2$. This provides a simple solution to the point representation problem that works for general elliptic curves and can represent all points, avoiding the need for rejection sampling inside the protocol to reach a particular subset of the curve. However, representation size is about twice as large as Elligator (a drawback partially addressed in subsequent work [132]) and the representation function, computing uniformly random preimages under $f^{\otimes 2}$, is also somewhat more complicated and costly than that of Elligator.

Basically, to compute a random preimage of $P \in E(\mathbb{F}_q)$, one picks a uniform $v \in \mathbb{F}_q$ and computes u as a preimage of $P - f(v)$. However, rejection sampling is necessary to ensure the uniformity of the distribution, which requires multiple iterations, each of them evaluating the function f (at a cost of a field exponentiation each).

In this work, as a by-product of our new SwiftEC construction, we also obtain ElligatorSwift, a much faster variant of Elligator Squared over all the curves over which SwiftEC is defined. The idea is that fully computing the underlying encoding in the forward direction becomes unnecessary, saving many field exponentiations in the process.

Contributions and technical overview. The starting point of our work is to revisit the first construction of an encoding function to general elliptic curves, originally due to Shallue and van de Woestijne [52]. We observe that that construction actually had a number of interesting properties that have not been considered so far, and that we manage

to build upon with suitable additional analysis. To describe them, we need to first recall a few facts about the Shallue–van de Woestijne encoding itself.

Given an elliptic curve $E\colon y^2 = g(x) = x^3 + ax + b$ over a finite field $\mathbb{F}_q$ of characteristic ≥ 5, Shallue and van de Woestijne construct a certain algebraic surface S in the affine space over $\mathbb{F}_q$ together with three rational functions x_1, x_2, x_3 such that the product $g(x_1)g(x_2)g(x_3)$ is a square. This means in particular that, when evaluated at any point P of $S(\mathbb{F}_q)$ (outside of the locus of poles), at least one of $x_1(P)$, $x_2(P)$ or $x_3(P)$ must be the x-coordinate of a point in $E(\mathbb{F}_q)$. Indeed, the product $g\big(x_1(P)\big)g\big(x_2(P)\big)g\big(x_3(P)\big)$ is a square in $\mathbb{F}_q$, and since the product of three nonsquares in $\mathbb{F}_q$ is a nonsquare, at least one of the factors must be square, yielding the x-coordinate of a point in $E(\mathbb{F}_q)$. Based on that, we can define an encoding function from $S(\mathbb{F}_q)$ to $E(\mathbb{F}_q)$ simply by mapping a point P to one of the points of x-coordinate $x_i(P)$ that works (selecting the index i and the sign of the y-coordinate in a predetermined way).

The second step of the construction is to note that the specific surface S under consideration can in fact be seen as a one-parameter family of conics over $\mathbb{F}_q$. Based on that, Shallue and van de Woestijne fix the value of the parameter, obtain a single non-degenerate conic over $\mathbb{F}_q$, and use the fact that such a conic always admits a rational parametrization to obtain a map $\mathbb{F}_q \to S(\mathbb{F}_q)$ to the chosen conic. Composing with the previous map finally gives an encoding $\mathbb{F}_q \to E(\mathbb{F}_q)$ as desired, which can be used in constructions (8.1)–(8.3) above for hashing, and in the Elligator Squared framework: this is what is usually known as the Shallue–van de Woestijne encoding.

Our contributions rely on two novel observations regarding that original construction:

- first, for a large class of elliptic curves E which we characterize in detail, the surface S regarded as a family of conics actually admits a global, two-parameter parametrization over $\mathbb{F}_q$. This means that one can effectively construct a rational map $\mathbb{F}_q^2 \to S(\mathbb{F}_q)$ that is essentially a bijection. This result is obtained using techniques due to van Hoeij and Cremona [134] classifying conics over function fields;

- second, unlike each of the maps defined by individual conics, the map from $S(\mathbb{F}_q)$ as a whole to the set $X_{E,\mathbb{F}_q}$ of elements of $\mathbb{F}_q$ which are x-coordinates on $E(\mathbb{F}_q)$ is *admissible*: it satisfies the sufficient conditions of Brier et al. [117] to construct indifferentiable hashing. The most important of those conditions is regularity: the image of a uniform point in $S(\mathbb{F}_q)$ is close to uniform in $X_{E,\mathbb{F}_q}$. We are able to establish that property by giving a precise description of the preimage of an $x \in X_{E,\mathbb{F}_q}$: it consists of the union of one algebraic curve drawn on S (the set of points P such that $x_1(P) = x$, say) and two halves of two other curves (the subset of the curves given by $x_2(P) = x$ and $x_3(P) = x$ respectively, with the condition that $g(x_1(P))$ is a nonsquare). By counting points on those curves and curve subsets, we are able to establish the required statistical properties, and deduce that $S(\mathbb{F}_q) \to X_{E,\mathbb{F}_q}$ is admissible.

Combining those two observations, we obtain, for a large, explicit class of elliptic curves E (including almost all curves in practical use), an admissible encoding $\mathbb{F}_q^2 \to X_{E,\mathbb{F}_q}$. Adding a sign bit to choose the y-coordinate on E yields an admissible encoding $F\colon \mathbb{F}_q^2 \times \{0,1\} \to E(\mathbb{F}_q)$ as well, which can be computed at the cost of a single exponentiation in $\mathbb{F}_q$ (namely, the square root computation needed to derive the y-coordinate). This has the two consequences mentioned above, over the elliptic curves E of interest:

- given a hash function $\mathfrak{h}$ modeled as a random oracle with values in $\mathbb{F}_q^2 \times \{0,1\}$ (which is easy to heuristically instantiate), the map $m \mapsto F\big(\mathfrak{h}(m)\big)$ is indifferentiable from a random oracle, and can be computed at the cost of a single exponentiation. This is the SwiftEC construction;

- given a uniform point on the curve, we can efficiently sample a uniform preimage of it under F, and this becomes a close-to-uniformly distributed element of $\mathbb{F}_q^2 \times \{0,1\}$. Since such an element is easy to represent as a uniform bit string, we thus obtain an Elligator Square-like representation technique which is much faster than Elligator Square itself, as it requires far fewer field exponentiations on average. This is the ElligatorSwift construction.

In addition, we also get indifferentiable hashing to the set $X_{E,\mathbb{F}_q}$ without any field exponentiation at all. This even faster construction, XSwiftEC, is particularly interesting in context where x-only arithmetic is feasible, such as for example BLS signatures [135].

8.1 Background

We begin by providing some background regarding quadratic residuosity and various statistical notions which will lay the ground for mathematical proofs regarding our construction.

8.1.1 Quadratic Residuosity

Throughout this chapter, $\mathbb{F}_q$ denotes the finite field with q elements. We only consider finite fields of characteristic $\neq 2, 3$. The *quadratic character* $\chi_2\colon \mathbb{F}_q \to \{-1, 0, 1\}$ is the map that sends 0 to 0, nonzero squares to 1 and nonzero nonsquares to -1. It is well-defined, multiplicative, and extends the unique nontrivial multicative group morphism $\mathbb{F}_q^\times \to \{-1, 1\}$. A related map is IsSquare, which sends all squares to 1 and nonsquares to 0.

When q is prime, the quadratic character coincides with the Legendre symbol, and can be computed efficiently by repeated applications of quadratic reciprocity. This can be implemented in fast constant time [136, 137, 138], similar to the constant-time binary GCD technique of Bernstein–Yang for field inversion [139]. Similarly, the quadratic character

over extension extension fields can be computed fast by descending to the prime field, and IsSquare can be trivially computed from χ_2.

We also fix an efficiently computable map $\operatorname{sgn}\colon \mathbb{F}_q \to \{-1, 0, 1\}$ called the "sign", with the property that $\operatorname{sgn} 0 = 0$, $\operatorname{sgn} x \neq 0$ for $x \neq 0$, and $\operatorname{sgn}(-x) = -\operatorname{sgn} x$. The choice is arbitrary, but for example over prime fields, it is customary to use the sign of an integer representative in the interval $(-q/2, q/2)$ (over extension fields, one might choose the sign of the first nonzero coefficient in some basis over the prime field).

An element $x \in \mathbb{F}_q$ which is a square has exactly two square roots (except 0 which has just one), exactly one of which is of nonnegative sign. We denote it by $\sqrt{x}$; it typically requires a single base field exponentiation to compute (although slightly faster approaches may exist over extension fields).

8.1.2 Point counting and character sums

A generalization of the result regarding the number of rational points of an elliptic curve is that any (absolutely irreducible) smooth curve of bounded genus over $\mathbb{F}_q$ has a number of points over $\mathbb{F}_q$ close to q. More precisely, the following celebrated result holds:

Lemma 8.1.1 (Hasse–Weil bound)**.** *For any smooth projective absolutely irreducible curve $X/\mathbb{F}_q$ of genus g, we have:*

$$\left|\#X(\mathbb{F}_q) - (q+1)\right| \leq 2g\sqrt{q}.$$

For curves of bounded degree, the number of points at infinity is also bounded, and we thus get a bound of the form $\#X^{\text{aff}}(\mathbb{F}_q) = q + c\sqrt{q} + O(1)$ ($|c| \leq 2g$) on the number of affine points on X.

A related result concerns character sums on such curves. Let χ be a multiplicative character of $\mathbb{F}_q$ (a group homomorphism $\mathbb{F}_q^\times \to \mathbb{C}^\times$ extended by 0 at 0), and $f \in \mathbb{F}_q(X)$ a rational function on the curve X. We consider the following character sum:

$$W(X, \chi, f) = \sum_{\substack{P \in X(\mathbb{F}_q) \\ f(P) \neq \infty}} \chi\big(f(P)\big).$$

Using the Bombieri–Weil methodology, Perret [140] proves the following bound. See also [141, 132].

Lemma 8.1.2 (Perret)**.** *Let X be a smooth projective absolutely irreducible curve of genus g over $\mathbb{F}_q$, χ a nontrivial multiplicative character of order $m | q-1$, and $f \in \mathbb{F}_q(X)$ a rational function which is not a perfect m-th power in $\bar{\mathbb{F}}_q(X)$. The character sum $W(X, \chi, f)$ can be bounded as:*

$$\left|W(X, \chi, f)\right| \leq (2g - 2 + 2\deg f)\sqrt{q}.$$

8.1.3 Quadratic residuosity over function fields

Many results of classical arithmetic over $\mathbb{Q}$ and number fields have analogues over function fields. This is in particular the case for quadratic reciprocity. We recall some of the relevant results below. An exhaustive treatment is provided in Rosen's textbook [142, pp. 23-31].

For a fixed monic irreducible polynomial $f \in \mathbb{F}_q[t]$, we define the quadratic residue symbol $\left(\frac{g}{f}\right)_2$ for any $g \in \mathbb{F}_q[t]$ as the image of g under the quadratic character of the finite field $\mathbb{F}_q[t]/(f)$. In other words:

$$\left(\frac{g}{f}\right)_2 = \begin{cases} 0 & \text{if } f \text{ divides } g; \\ 1 & \text{if } g \text{ is coprime to } f \text{ and a square modulo } f; \\ -1 & \text{if } g \text{ is coprime to } f \text{ and a nonsquare modulo } f. \end{cases}$$

We then extend this symbol to not necessarily irreducible f's by multiplicativity, similarly to how the Jacobi symbol extends the Legendre symbol. If $f = \alpha f_1^{e_1} \cdots f_n^{e_n}$ with $\alpha \in \mathbb{F}_q^\times$ and the f_i irreducible, we let:

$$\left(\frac{g}{f}\right)_2 = \prod_{i=1} n \left(\frac{g}{f_i}\right)_2.$$

Note that the symbol does not depend on the leading coefficient $\mathrm{lc}(f) = \alpha$ of f.

Lemma 8.1.3. *The quadratic residue symbol has the following properties.*

- *If* $g_1 \equiv g_2 \pmod f$, $\left(\frac{g_1}{f}\right)_2 = \left(\frac{g_2}{f}\right)_2$.
- $\left(\frac{g_1 g_1}{f}\right)_2 = \left(\frac{g_1}{f}\right)_2 \left(\frac{g_2}{f}\right)_2$.
- $\left(\frac{g}{f_1 f_2}\right)_2 = \left(\frac{g}{f_1}\right)_2 \left(\frac{g}{f_2}\right)_2$.
- $\left(\frac{g}{f}\right)_2 \neq 0$ *if and only if* f *and* g *are coprime.*
- *If* g *is a nonzero square modulo* f*, then* $\left(\frac{g}{f}\right)_2 = 1$ *(but the converse does not need to hold).*

Furthermore, it satisfies the following law of quadratic reciprocity. For $f, g \in \mathbb{F}_q[t]$ *coprime and nonzero, it holds that:*

$$\left(\frac{g}{f}\right)_2 \left(\frac{f}{g}\right)_2 = (-1)^{\frac{q-1}{2} \deg f \deg g} \mathrm{lc}(f)^{\frac{q-1}{2} \deg g} \mathrm{lc}(g)^{\frac{q-1}{2} \deg f}.$$

8.1.4 Statistical notions

For $\mathscr{D}$ a probability distribution on a finite set S, we write $\Pr[s \leftarrow \mathscr{D}]$ for the probability assigned to the singleton $\{s\} \subset S$ by $\mathscr{D}$. The uniform distribution on S is denoted by $\mathscr{U}_S$ (or just $\mathscr{U}$ if the context is clear).

Definition 8.1.1 (Statistical distance)**.** *Let $\mathscr{D}$ and $\mathscr{D}'$ be two probability distributions on a finite set S. The* statistical distance *between them is defined as the ℓ_1 norm:*

$$\Delta_1(\mathscr{D}, \mathscr{D}') = \frac{1}{2} \sum_{s \in S} \left| \Pr[s \leftarrow \mathscr{D}] - \Pr[s \leftarrow \mathscr{D}'] \right|.$$

We simply denote by $\Delta_1(\mathscr{D})$ the statistical distance between $\mathscr{D}$ and $\mathscr{U}_S$:

$$\Delta_1(\mathscr{D}) = \frac{1}{2} \sum_{s \in S} \left| \Pr[s \leftarrow \mathscr{D}] - \frac{1}{\#S} \right|,$$

and say that $\mathscr{D}$ is ε-statistically close to uniform *when $\Delta_1(\mathscr{D}) \leq \varepsilon$. When $\Delta_1(\mathscr{D})$ is negligible, we simply say than $\mathscr{D}$ is* statistically close to uniform.

Definition 8.1.2 (Pushforward)**.** *Let S, T be two finite sets and F any mapping from S to T. For any probability distribution $\mathscr{D}_S$ on S, we can define the* pushforward $F_*\mathscr{D}_S$ *of $\mathscr{D}_S$ by F as the probability distribution on T such that sampling from $F_*\mathscr{D}_S$ is equivalent to sampling a value $s \leftarrow \mathscr{D}_S$ and returning $F(s)$. In other words:*

$$\Pr\left[t \leftarrow F_*\mathscr{D}_S\right] = \Pr\left[s \leftarrow \mathscr{D}_S;\ t = F(s)\right] = \mu_S\big(F^{-1}(t)\big) = \sum_{s \in F^{-1}(t)} \Pr[s \leftarrow \mathscr{D}_S],$$

where μ_S is the probability measure defined by $\mathscr{D}_S$.

Definition 8.1.3 (Regularity)**.** *Let S, T be two finite sets and F any mapping from S to T. We say that F is* ε-regular *when $F_*\mathscr{U}_S$ is ε-close to the uniform distribution. We may omit ε if it is negligible.*

8.1.5 Admissible encodings

In their work on the construction of indifferentiable hashing to elliptic curves, Brier et al. [117] define the notion of an *admissible* map $F\colon S \to R$ between two sets. The definition, which generalizes an early notion introduced by Boneh and Franklin [130], is as follows.

Definition 8.1.4 (Admissible encoding)**.** *A function $F\colon S \to R$ between finite sets is an ε-admissible encoding if it satisfies the following properties:*

Computable: *F is computable in deterministic polynomial time.*

Regular: *F is ε-regular (in the sense of the previous section).*

Samplable: *there is an efficient randomized algorithm $\mathscr{I}\colon R \to S \sqcup \{\perp\}$ such that for any $r \in R$, $\mathscr{I}(r)$ induces a distribution that is ε-statistically close to the uniform distribution in $F^{-1}(r)$.*

F is an admissible encoding if it is ε-admissible for some negligible ε.

That notion satisfies the suitable properties such that, given an S-valued random oracle $\mathfrak{h}$, the composition $F \circ \mathfrak{h}$ is indifferentiable from a R-valued random oracle.

Moreover a similar results holds for arbitrary compositions of admissible functions (even though admissibility need not be preserved under composition). Namely, if $F_i\colon S_i \to S_{i-1}$ are admissible encodings for $i = 1, \ldots, n$, then it also holds that, given an S_n-valued random oracle $\mathfrak{h}$, the composition $F_1 \circ \cdots \circ F_n \circ \mathfrak{h}$ is indifferentiable from a S_0-valued random oracle (even though it does not always hold that $F_1 \circ \cdots \circ F_n$ is admissible).

8.2 The SW Encoding Family

In their seminal ANTS–VII paper [52], Shallue and van de Woestijne constructed the first encoding function to arbitrary elliptic curves. In this section, we give a description of that construction (restricted for simplicity to base fields of characteristic ≥ 5) that is slightly different but essentially equivalent to the original one, and then we state new properties of that construction.

In the entire section, we fix an elliptic curve $E\colon y^2 = x^3 + ax + b$ over the finite field $\mathbb{F}_q$ (q prime power not divisible by 2 or 3), and denote by $X_{E,\mathbb{F}_q}$ the subset of $\mathbb{F}_q$ consisting of x-coordinates of points in $E(\mathbb{F}_q)$; in other words:

$$X_{E,\mathbb{F}_q} = \big\{x \in \mathbb{F}_q \ ; \ \exists y, \ (x,y) \in E(\mathbb{F}_q)\big\}.$$

8.2.1 Construction of the Shallue–van de Woestijne encoding

Let g and h be the polynomials over $\mathbb{F}_q$ defined by:

$$g(u) = u^3 + au + b \quad \text{and} \quad h(u) = 3u^2 + 4a.$$

The starting point of the Shallue–van de Woestijne construction is the construction of a rational map $\psi\colon S \to V$ from the following quasi-affine surface in the (x, y, u) affine space:

$$S\colon x^2 + h(u)y^2 = -g(u), \quad y \neq 0 \tag{8.5}$$

to the following threefold in the (x_1, x_2, x_3, z) affine 4-dimensional space:

$$V\colon z^2 = g(x_1)g(x_2)g(x_3).$$

The rational map ψ is given by the following explicit equations and clearly defined everywhere on S:

$$\begin{aligned} x_1 &= \frac{x}{2y} - \frac{u}{2} \quad & x_2 &= -\frac{x}{2y} - \frac{u}{2} \\ x_3 &= u + 4y^2 \quad & z &= \frac{g(u+4y^2)}{2y} \cdot R\Big(u, \frac{x}{2y} - \frac{u}{2}\Big) \end{aligned} \tag{8.6}$$

where $R(u,v) = u^2 + uv + v^2 + a$. When referring to a point P on S, we will denote by $x_1(P)$, $x_2(P)$, $x_3(P)$ and $z(P)$ the corresponding coordinates of $\psi(P)$ in V. In particular, this defines x_1, x_2, x_3 and z as rational functions on the surface.

A remarkable property of the threefold V is that for any point $(x_1, x_2, x_3, z) \in V(\mathbb{F}_q)$, at least one of the three values x_1, x_2, x_3 must be in $X_{E,\mathbb{F}_q}$. Indeed, $g(x_1)g(x_2)g(x_3)$ is a square in $\mathbb{F}_q$, so by multiplicativity of the quadratic character, they cannot be all nonsquares (and in fact, there must be exactly one or three squares among them, except possibly when $z = 0$).

As a result, one can therefore map points on $S(\mathbb{F}_q)$ to $X_{E,\mathbb{F}_q}$ by first mapping to $V(\mathbb{F}_q)$ with ψ, and then selecting one of the coordinates x_1, x_2, x_3 in a prescribed order. For example, in this chapter we will consider the following map:

$$\begin{aligned} F_0 \colon S(\mathbb{F}_q) &\to X_{E,\mathbb{F}_q} \\ P &\mapsto \begin{cases} x_3(P) & \text{if } g\big(x_3(P)\big) \text{ is a square;} \\ x_2(P) & \text{if } g\big(x_3(P)\big) \text{ is not a square but } g\big(x_2(P)\big) \text{ is;} \\ x_1(P) & \text{if neither } g\big(x_3(P)\big) \text{ or } g\big(x_2(P)\big) \text{ are squares.} \end{cases} \end{aligned} \tag{8.7}$$

Note that $F_0(P)$ is very efficient to compute from the coordinates (x, y, u) of P using the formulas of (8.6) and a few quadratic character computations. In particular, it requires no field exponentiation.

Of course, once we have an element $\bar{x} \in X_{E,\mathbb{F}_q}$, it is easy to deduce a point in $E(\mathbb{F}_q)$: simply compute a square root of $g(\bar{x})$ to get the y-coordinate up to sign. Since we prefer to select the sign separately, we define the following extended map to $E(\mathbb{F}_q)$ which takes an additional input bit b:

$$\begin{aligned} F_0^+ \colon S(\mathbb{F}_q) \times \{0,1\} &\to E(\mathbb{F}_q) \\ (P, b) &\mapsto \Big(F_0(P),\ (-1)^b \sqrt{g\big(F_0(P)\big)}\Big). \end{aligned} \tag{8.8}$$

The construction offers a way to map to $E(\mathbb{F}_q)$ provided that one can construct rational points on the surface S itself, which may not be a priori obvious. Fortunately, as seen from equation (8.5), each of the curves S_{u_0} on S obtained by fixing u to some $u_0 \in \mathbb{F}_q$ are simply conics over $\mathbb{F}_q$, with equations:

$$x^2 + h(u_0)y^2 = -g(u_0), \quad y \neq 0.$$

Now, a conic over $\mathbb{F}_q$ always admits a rational parametrization. Therefore, we can construct a map $\mathbb{F}_q \to S_{u_0}(\mathbb{F}_q)$ that can then be composed with F_0^+ to obtain an encoding function $F_{0,u_0}\colon \mathbb{F}_q \to X_{E,\mathbb{F}_q}$ (and similarly to $E(\mathbb{F}_q)$). This is basically the approach taken in the original paper of Shallue and van de Woestijne [52].

Note that obtaining the parametrization of the conic S_{u_0} for a fixed u_0 requires an a priori costly precomputation (it requires finding a point on the conic, typically by trial-and-error: this costs a square root, and a number of quadratic character computations that is hard to bound uniformly). Therefore, while it may be tempting to try and define a two-parameter map $\mathbb{F}_q^2 \to X_{E,\mathbb{F}_q}$ by $(t,u) \mapsto F_{0,u}(t)$, this is not usually workable for hashing purposes, since a new parametrization would have to be computed for any new input u.

Nevertheless, we show in the remainder of this section that the maps F_0 and F_0^+ on the surface $S(\mathbb{F}_q)$ *as a whole* have nice statistical properties, and it would therefore be beneficial to overcome the difficulty of efficiently parametrizing it. That problem will then be addressed, at least for a large class of elliptic curves E, in Section 8.3 below.

8.2.2 Geometry of the SW family

For a fixed element $\bar{x} \in X_{E,\mathbb{F}_q}$, we now want to describe the set of points in $S(\mathbb{F}_q)$ that map to x under the encoding F_0 of (8.7). By the previous description of the encoding, this is the union of three disjoint sets:

$$F_0^{-1}(\bar{x}) = C_{\bar{x}}^{(3)}(\mathbb{F}_q) \sqcup C_{\bar{x}}^{(2)}(\mathbb{F}_q)^+ \sqcup C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+,$$

where $C_{\bar{x}}^{(i)}$ are algebraic curves on S defined by the condition that $x_i = \bar{x}$ $(i = 1,2,3)$ and $C_{\bar{x}}^{(i)}(\mathbb{F}_q)^+$ is the subset of $C_{\bar{x}}^{(i)}(\mathbb{F}_q)$ under the condition that $g\big(x_j(P)\big)$ is not a square for $j \neq i$. Note that since there are always exactly only 1 or 3 squares, it suffices to define

$$C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ := \{P \in C_{\bar{x}}^{(1)}(\mathbb{F}_q);\ x_2(P) \text{ not a square}\}$$

$$C_{\bar{x}}^{(2)}(\mathbb{F}_q)^+ := \{P \in C_{\bar{x}}^{(2)}(\mathbb{F}_q);\ x_1(P) \text{ not a square}\}$$

We would like to count the number of points in each of these sets. The first step is to understand the geometry of the curves $C_{\bar{x}}^{(i)}$. It is easy to see that, for a generic $\bar{x}$, they are hyperelliptic curves of genus 2.

Consider for example $C_{\bar{x}}^{(3)}$. It is given by the equations (*cf.* (8.6)):

$$u + 4y^2 = \bar{x} \quad \text{and} \quad x^2 + h(u)y^2 = -g(u).$$

Eliminating $u = \bar{x} - 4y^2$ between those two equations, we see that that $C_{\bar{x}}^{(3)}$ is isomorphic to the curve in the (y,x) affine plane given by the equation:

$$x^2 = -g(\bar{x} - 4y^2) - h(\bar{x} - 4y^2)y^2.$$

The right-hand side is a polynomial of degree 6 in y, namely:

$$16y^6 - 24\bar{x}y^4 + 9\bar{x}^2y^2 - g(\bar{x}),$$

whose discriminant is a polynomial of degree exactly 11 in $\bar{x}$ (or exactly 9 if $a = 0$). We thus get that $C^{(3)}_{\bar{x}}$ is a hyperelliptic curve of genus 2, except for at most 11 points $\bar{x}$. Other than for those exceptional points, we have:

$$\#C^{(3)}_{\bar{x}}(\mathbb{F}_q) = q + c_3\sqrt{q} + O(1), \quad \text{for some } c_3 \text{ such that } |c_3| \leq 4.$$

by the Hasse–Weil bound. Note that the $O(1)$ term comes from the fact that we consider the affine situation rather than the projective one, and we could easily provide an explicit bound for it, but this is typically not of interest for cryptographic applications.

Similarly, $C^{(2)}_{\bar{x}}$ is given by the equations:

$$-\frac{x}{2y} - \frac{u}{2} = \bar{x} \quad \text{and} \quad x^2 + h(u)y^2 = -g(u).$$

Eliminating $x = -y(u + 2\bar{x})$ between those two equations, we see that that $C^{(2)}_{\bar{x}}$ is isomorphic to the curve in the (u, y) affine plane given by the equation:

$$y^2\big[(u + 2\bar{x})^2 + h(u)\big] = -g(u),$$

which is again isomorphic to the curve in the (u, v) affine plane, $v = y\big[(u + 2\bar{x})^2 + h(u)\big]$, of equation:

$$v^2 = -g(u) \cdot \big[(u + 2\bar{x})^2 + h(u)\big].$$

The right-hand side is a polynomial of degree 5 in u, namely:

$$-4\big(u^5 + \bar{x}u^4 + (\bar{x}^2 + 2a)u^3 + (a\bar{x} + b)u^2 + (a\bar{x}^2 + b\bar{x} + a^2)u + b(\bar{x}^2 + a)\big),$$

and its discriminant is always of degree 14 in $\bar{x}$ (the degree 14 coefficient is $2^{16} \cdot 3 \cdot (4a^3 + 27b^2) \neq 0$). Thus, $C^{(2)}_{\bar{x}}$ is a hyperelliptic curve of genus 2, except for at most 14 points $\bar{x}$. Other than for those exceptional points, we therefore have:

$$\#C^{(2)}_{\bar{x}}(\mathbb{F}_q) = q + c_2\sqrt{q} + O(1) \quad \text{for some } c_2 \text{ such that } |c_2| \leq 4$$

by the Hasse–Weil bound.

A similar computation gives the same result for $C^{(1)}_{\bar{x}}$, given by the equations:

$$\frac{x}{2y} - \frac{u}{2} = \bar{x} \quad \text{and} \quad x^2 + h(u)y^2 = -g(u).$$

Indeed, eliminating $x = y(u + 2\bar{x})$ between those two equations shows that $C^{(1)}_{\bar{x}}$ is also isomorphic to the curve in the (u, y) affine plane given by the equation:

$$y^2\big[(u + 2\bar{x})^2 + h(u)\big] = -g(u),$$

the same as above. It therefore holds again that, except for at most 14 points $\bar{x}$, $C_{\bar{1}}^{(2)}$ is a hyperelliptic curve of genus 2, and:

$$\#C_{\bar{x}}^{(1)}(\mathbb{F}_q) = q + c_1\sqrt{q} + O(1) \quad \text{for some } c_1 \text{ such that } |c_1| \leq 4$$

by the Hasse–Weil bound.

It remains to evaluate the cardinality of the subsets $C_{\bar{x}}^{(i)}(\mathbb{F}_q)^+ \subset C_{\bar{x}}^{(i)}(\mathbb{F}_q)$ for $i = 1, 2$. One can do so in various ways, but the simplest is probably to relate them to character sums. Consider for example the following character sum on $C_{\bar{x}}^{(1)}$:

$$W_1 := W\big(C_{\bar{x}}^{(1)}, \chi_2, g \circ x_2\big) = \sum_{P \in C_{\bar{x}}^{(1)}(\mathbb{F}_q)} \chi_2\Big(g\big(x_2(P)\big)\Big),$$

where χ_2 is the quadratic multiplicative character of $\mathbb{F}_q$. The term $\chi_2\Big(g\big(x_2(P)\big)\Big)$ is equal to -1 if $g\big(x_2(P)\big)$ is not a square in $\mathbb{F}_q$, which is exactly when $P \in C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+$. Moreover, it is otherwise equal to 1 (for points outside $C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+$ such that $x_2(P) \neq 0$) or 0 (for points outside $C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+$ such that $x_2(P) = 0$). As a result, we have:

$$\begin{aligned} W_1 &= (-1) \cdot \#C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ + 1 \cdot (\#C_{\bar{x}}^{(1)}(\mathbb{F}_q) - \#C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ - N_0) + 0 \cdot N_0 \\ &= \#C_{\bar{x}}^{(1)}(\mathbb{F}_q) - 2 \cdot \#C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ - N_0, \end{aligned}$$

where $N_0 = O(1)$ is the number of points in $C_{\bar{x}}^{(1)}(\mathbb{F}_q)$ such that $x_2(P) = 0$. This gives:

$$\#C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ = \frac{1}{2}\#C_{\bar{x}}^{(1)}(\mathbb{F}_q) - \frac{W_1}{2} + O(1) = \frac{q}{2} + \frac{c_1}{2}\sqrt{q} - \frac{W_1}{2} + O(1),$$

where the $O(1)$ term accounts both for N_0 and for the fact that we consider an affine situation instead of a projective one.

Then, by the character sum estimate of 8.1.2, we have:

$$|W_1| \leq \big(4 - 2 + 2\deg(g \circ x_2)\big)\sqrt{q} + O(1) = (2 + 2 \cdot 3 \cdot 2)\sqrt{q} + O(1) = 14\sqrt{q} + O(1)$$

since $x_2 = u - \bar{x}$ on $C_{\bar{x}}^{(1)}$ is a rational function of degree 2. It then follows that:

$$\#C_{\bar{x}}^{(1)}(\mathbb{F}_q)^+ = \frac{q}{2} + c_1^+\sqrt{q} + O(1) \quad \text{for some } c_1^+ \text{ such that } |c_1^+| \leq \frac{4 + 14}{2} = 9.$$

Obviously, the exact same argument applies to $C_{\bar{x}}^{(2)}$, yielding:

$$\#C_{\bar{x}}^{(2)}(\mathbb{F}_q)^+ = \frac{q}{2} + c_2^+\sqrt{q} + O(1) \quad \text{for some } c_2^+ \text{ such that } |c_2^+| \leq 9.$$

Combining all the previous estimates, we finally obtain the following result.

Theorem 8.2.1. *For all $\bar{x} \in X_{E,\mathbb{F}_q}$ except at most 39 of them, the number of preimages of $\bar{x}$ under the F_0 map of equation (8.7) is close to $2q$, and the difference is bounded as:*

$$\left|\#F_0^{-1}(\bar{x}) - 2q\right| \leq 22\sqrt{q} + O(1).$$

Proof. Indeed, except for the at most $11 + 14 + 14 = 39$ exceptional points mentioned above, we have:

$$\#F_0^{-1}(\bar{x}) = \Big(1 + \frac{1}{2} + \frac{1}{2}\Big)q + \big(c_1^+ + c_2^+ + c_3\big)\sqrt{q} + O(1)$$

and since $\left|c_1^+ + c_2^+ + c_3\right| \leq 4 + 9 + 9 = 22$, the result follows. □

8.2.3 The SW family is admissible

Using Theorem 8.2.1, we are now in a position to prove that the encoding function F_0 is *admissible* in the sense of Subsection 8.1.5. The main step in doing so is to prove that it is *regular*.

Lemma 8.2.1. *The map $F_0 \colon S(\mathbb{F}_q) \to X_{E,\mathbb{F}_q}$ of equation (8.7) is ε-regular for $\varepsilon = \big(6 + o(1)\big)q^{-1/2}$.*

Proof. Let $\Delta = \Delta_1\big((F_0)_*\mathscr{U}_{S(\mathbb{F}_q)}\big)$ be the statistical distance between the distribution induced by F_0 on $X_{E,\mathbb{F}_q}$ and the uniform distribution. By definition, we have:

$$\Delta = \frac{1}{2}\sum_{\bar{x} \in X_{E,\mathbb{F}_q}} \left|\frac{\#F^{-1}(\bar{x})}{\#S(\mathbb{F}_q)} - \frac{1}{\#X_{E,\mathbb{F}_q}}\right|.$$

Now for each element $\bar{x} \in X_{E,\mathbb{F}_q}$, there are exactly two points of $E(\mathbb{F}_q)$ with x-coordinate equal to $\bar{x}$, except if $g(\bar{x}) = 0$, in which case there is exactly one (and this happens for at most three values of $\bar{x}$). Taking the point at infinity into account, we therefore get:

$$\#X_{E,\mathbb{F}_q} = \frac{1}{2}\#E(\mathbb{F}_q) + O(1) = \frac{q}{2} + c_E\sqrt{q} + O(1) \quad \text{for some } c_E \text{ with } |c_E| \leq 1$$

by yet another application of the Hasse–Weil bound. Up to sign, the constant c_E is half the normalized Frobenius trace of E.

Moreover, $S(\mathbb{F}_q)$ is the disjoint union of the various affine conics $\big\{x^2 + h(u_0)y^2 = -g(u_0),\ u = u_0\big\}$ for all $u_0 \in \mathbb{F}_q$. Those conics are nondegenerate whenever $g(u_0)h(u_0) \neq 0$, in which case they have $q + O(1)$ points. In remaining exceptional cases, they have at most $2q$ points. As a result, we get:

$$\#S(\mathbb{F}_q) = \big(q - O(1)\big) \cdot \big(q + O(1)\big) + O(1) \cdot O(q) = q^2 + O(q).$$

As for the number of preimages of F, we know by Theorem 8.2.1 that for each $\bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}$, where X_{bad} is a set of 39 points, there exists $c_{0,\bar{x}} \in [-22, 22]$ such that:

$$\#F^{-1}(\bar{x}) = 2q + c_{0,\bar{x}}\sqrt{q} + O(1) \qquad \forall \bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}$$

For $\bar{x} \in X_{\text{bad}}$, we can still obtain a less strict but simpler bound: note that for any fixed $u = u_0 \in \mathbb{F}_q$ the equations $\bar{x} = x_1(x, y, u_0)$, $\bar{x} = x_2(x, y, u_0)$ and $\bar{x} = x_3(x, y, u_0)$ have at most 2, 2, and 4 solutions in S, respectively (these solutions are given explicitly in Section 8.6). Hence, any point can have at most 8 preimages for any fixed u_0 and at most $8q$ preimages in all.

We can now bound Δ as follows:

$$\begin{aligned}
2\Delta &= \sum_{\bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}} \left| \frac{\#F^{-1}(\bar{x})}{\#S(\mathbb{F}_q)} - \frac{1}{\#X_{E,\mathbb{F}_q}} \right| + \sum_{\bar{x} \in X_{\text{bad}}} \left| \frac{\#F^{-1}(\bar{x})}{\#S(\mathbb{F}_q)} - \frac{1}{\#X_{E,\mathbb{F}_q}} \right| \\
&= \sum_{\bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}} \left| \frac{2q + c_{0,\bar{x}}\sqrt{q} + O(1)}{q^2 + O(q)} - \frac{1}{q/2 + c_E\sqrt{q} + O(1)} \right| + \\
&\qquad \sum_{\bar{x} \in X_{\text{bad}}} \left| \frac{c_{\text{bad},\bar{x}} q}{q^2 + O(q)} - \frac{1}{q/2 + c_E\sqrt{q} + O(1)} \right| \\
&= \sum_{\bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}} \frac{1}{q} \left| \left(2 + c_{0,\bar{x}} q^{-1/2} + O(q^{-1})\right) - \left(2 - c_E q^{-1/2} + O(q^{-1})\right) \right| + \\
&\qquad \sum_{\bar{x} \in X_{\text{bad}}} \frac{1}{q} \left| \left(c_{\text{bad},\bar{x}} + O(q^{-3})\right) - \left(2 - c_E q^{-1/2} + O(q^{-1})\right) \right| \\
&= \sum_{\bar{x} \in X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}} \frac{1}{q} \left| \left(c_{0,\bar{x}} + c_E\right) q^{-1/2} + O(q^{-1}) \right| + \sum_{\bar{x} \in X_{\text{bad}}} \frac{1}{q} \left| c_{\text{bad},\bar{x}} - 2 + O(q^{-1/2}) \right|
\end{aligned}$$

where each of the constants $c_{0,\bar{x}}$ is in $[-22, 22]$ and each of the constants $c_{\text{bad},\bar{x}}$ is in $[0, 8]$. In particular, $|c_{0,\bar{x}} + c_E| \le 23$ and $|c_{\text{bad},\bar{x}} - 2| \le 6$ for all $\bar{x}$, and we have:

$$\begin{aligned}
2\Delta &\le \frac{\#\left(X_{E,\mathbb{F}_q} \setminus X_{\text{bad}}\right)}{q} \cdot \left(23q^{-1/2} + O(q^{-1})\right) + \frac{\#X_{\text{bad}}}{q} \cdot \left(6 + O(q^{-1/2})\right) \\
&= \frac{\frac{1}{2}q + O(\sqrt{q})}{q} \cdot \left(23 + o(1)\right) q^{-1/2} + \frac{39}{q} \cdot \left(6 + o(1)\right) \\
&= \left(\frac{23}{2} + o(1)\right) q^{-1/2} \le 2 \cdot \left(6 + o(1)\right) q^{-1/2}
\end{aligned}$$

as required. □

As an easy consequence, we obtain the following theorem.

Theorem 8.2.2. *The map $F_0\colon S(\mathbb{F}_q) \to X_{E,\mathbb{F}_q}$ of equation (8.7) is ε-admissible for $\varepsilon = \big(6 + o(1)\big)q^{-1/2}$. In particular, if $\mathfrak{h}$ is a random oracle with values in $S(\mathbb{F}_q)$, $F_0 \circ \mathfrak{h}$ is indifferentiable from an $X_{E,\mathbb{F}_q}$ random oracle.*

Moreover, the same results hold for $F_0^+\colon S(\mathbb{F}_q) \times \{0,1\} \to E(\mathbb{F}_q)$.

Proof. By definition, we need to prove that F_0 is efficiently computatable, ε-regular and ε-samplable. Computability is obvious. Regularity is the result of 8.2.1. And 0-samplability is obtained using the preimage sampling algorithm discussed in Section 8.6 below. To fix ideas, we sketch its construction.

Fix $\bar{x} \in X_{E,\mathbb{F}_q}$. As previously mentioned, for any fixed $u_0 \in \mathbb{F}_q$, there are at most 8 preimages $(x, y, u) \in F^{-1}(\bar{x})$ such that $u = u_0$ (at most two coming from each of x_1 and x_2 and four coming from x_3). We can efficiently compute all those preimages and in particular count them. Therefore, the following simple rejection sampling algorithm has an output distribution uniform in $F^{-1}(\bar{x})$: pick u_0 uniformly at random, compute the list L_{u_0} of preimages with $u = u_0$, restart with probability $1 - \#L_{u_0}/8$ and otherwise return a random element of L_{u_0}.

Finally, the extension to F_0^+ is straightforward. □

8.3 Parametrizing the SW Conic

We now describe our approach for parametrizing the family of conics that appear in the Shallue–van de Woestijne construction. The base prerequisites that must be met are described in Subsection 8.3.1 in terms of the conics, and this is translated into a set of requirements for the elliptic curve in Subsection 8.3.2. We then describe some workarounds that help us deal even with curves that do not meet these base requirements in Subsection 8.3.3.

8.3.1 Parametrizability conditions

In the previous section, we have seen how the Shallue–van de Woestijne construction could be leveraged to construct admissible encodings $F_0\colon S(\mathbb{F}_q) \to X_{E,\mathbb{F}_q}$ and $F_0^+\colon S(\mathbb{F}_q) \times \{0,1\} \to E(\mathbb{F}_q)$. However, we have also seen that mapping to $\mathbb{F}_q$-points on the surface S efficiently (without base field exponentiations) is a priori not straightforward, since the most naive approach involves finding points on new conics for all inputs.

Fortunately, the surface S has a fairly simple description: it can be seen as a *one-parameter family* of conics (the conics S_u; this is also called a relative conic over the u-line, or a fibration in conics, etc.). In any case, finding a global, two-parameter parametrization of S is thus a function field analogue of the classical problem, studied by Legendre, of finding rational points on conic over $\mathbb{Q}$.

In their paper [134], van Hoeij and Cremona show that Legendre's original approach can be directly adapted to the function field case. They provide necessary and sufficient

conditions for the existence of solutions, as well as an effective algorithm to compute the parametrization if it exists.

A special case of their main result in as follows.

Lemma 8.3.1 (van Hoeij–Cremona)**.** *Let r, s be polynomials in $\mathbb{F}_q[u]$ that are coprime, squarefree, and such that at least one of them is of odd degree. Then, the following projective conic over $\mathbb{F}_q(t)$:*

$$X^2 + rY^2 + sZ^2 = 0$$

admits rational points over $\mathbb{F}_q(u)$ (i.e., a global rational parametrization) if and only if the following two conditions hold:

1. *$-r$ is a square in $\mathbb{F}_q[u]/(s)$*
2. *$-s$ is a square in $\mathbb{F}_q[u]/(r)$.*

Moreover, if this is the case, there is an efficient algorithm to compute those points.

Proof. This is a special case of [134, Th. 1]. More precisely, the assumptions ensure that the conic is in reduced form and in "case 1", in the terminology of van Hoeij and Cremona, and the squareness conditions are equivalent to the existence of a "solubility certificate". □

The proof presented by van Hoeij and Cremona is constructive in that it yields an explicit algorithm for finding the rational parametrization. Our case of interest, corresponding to the surface S, is $r = h(u) = 3u^2 + 4a$ and $s = g(u) = u^3 + au + b$ (except when $a = 0$, in which case a slight adjustment is necessary to meet the assumptions of the theorem). In that case, if a parametrization exists, it can be put in the form where $Z = 1$, and X, Y are polynomials of degree 2 and 1 in u respectively, as will be shown below. These polynomials depend only on the parameters a, b of the target elliptic curve, so the polynomial coefficients can be precomputed while their evaluation at a given u is done at runtime.

8.3.2 Curves with a parametrizable SW conic

Due to the conditions in 8.3.1, the SwiftEC encoding is not applicable to every ordinary elliptic curve. We present a different characterization of these conditions from the point of view of the target curve's geometric properties.

Theorem 8.3.1. *The surface S, as a one-parameter family of conics, admits a global two-parameter parametrization if and only if the following three conditions are satisfied.*

1. *The size of the field satisfies $q \equiv 1 \mod 3$ (i.e., -3 is a square in $\mathbb{F}_q$).*
2. *The discriminant $\Delta_E = -16(4a^3 + 27b^2)$ is a square in $\mathbb{F}_q$ (i.e. E has either zero or three points of order 2).*

3. At least one of the constants $\nu_\pm = \frac{1}{2}(-b \pm \sqrt{-3\Delta_E}/36)$ *is a square in* $\mathbb{F}_q$.

Proof. As a first observation, note that if we let $r = h(u)$ and $s = g(u)$, then r and s are indeed coprime (their resultant is $4a^3 + 27b^2 = -\Delta_E/16 \neq 0$) and s is of odd degree and squarefree. Moreover, r is squarefree if and only if $a \neq 0$. For now, we assume that $a \neq 0$, so that 8.3.1 applies directly. We will treat the special case of $a = 0$ at the end.

Let us first assume that $-h$ is a square in $\mathbb{F}_q[u]/(g)$ and $-g$ is a square in $\mathbb{F}_q[u]/(h)$. Note that h and g are coprime since their resultant is $4a^3 + 27b^2 = -\Delta_E/16 \neq 0$, so the law of quadratic reciprocity over function fields gives

$$\begin{aligned}\left(\frac{-h}{g}\right)_2 \left(\frac{g}{-h}\right)_2 &= (-1)^{\frac{q-1}{2}\deg g \deg h} \chi_2(1)^{\deg h} \chi_2(-3)^{\deg g} \\ 1 \cdot \left(\frac{g}{-h}\right)_2 &= 1 \cdot 1 \cdot \chi_2(-3), \end{aligned} \tag{8.9}$$

where $\left(\frac{\cdot}{f}\right)_2$ and $\chi_2(\cdot)$ denote quadratic residue symbols over $\mathbb{F}_q[u]/(f)$ and $\mathbb{F}_q$, respectively.

On the other hand, we have

$$1 = \left(\frac{-g}{h}\right)_2 = \left(\frac{-1}{h}\right)_2 \left(\frac{-g}{h}\right)_2 = \chi_2(-1)^2 \left(\frac{g}{-h}\right)_2 = \left(\frac{g}{-h}\right)_2,$$

so (8.9) reduces to $\chi_2(-3) = 1$, which shows the necessity of condition 1.

Next, since $-g$ is a square in $\mathbb{F}_q[u]/(h)$, there exists $\alpha, \beta \in \mathbb{F}_q$ such that:

$$\begin{aligned} -g &\equiv (\alpha u + \beta)^2 \pmod{h} \\ -u^3 - au - b &\equiv \alpha^2 u^2 + 2\alpha\beta u + \beta^2 \pmod{3u^2 + 4a} \\ \frac{4a}{3}u - au - b &\equiv -\frac{4a}{3}\alpha^2 + 2\alpha\beta u + \beta^2 \pmod{3u^2 + 4a} \\ \frac{a}{3}u - b &= 2\alpha\beta u + \big(-\frac{4a}{3}\alpha^2 + \beta^2\big). \end{aligned}$$

It follows that the constants α, β satisfy

$$\frac{a}{3} = 2\alpha\beta \tag{8.10}$$

$$b = \frac{4a}{3}\alpha^2 - \beta^2. \tag{8.11}$$

Recalling that $a \neq 0$, it follows from (8.10) that $\alpha, \beta \neq 0$ and we can substitute $\beta = a/(6\alpha)$ into (8.11) to obtain

$$48a\alpha^4 - 36b\alpha^2 - a^2 = 0, \tag{8.12}$$

which is a quadratic equation on α^2 whose discriminant is $36^2b^2 + 192a^3 = -3\Delta_E$. Since -3 is a square, it follows that Δ_E must also be a square for α^2 to exist, showing the necessity of condition 2. The solution to (8.12) is then given by

$$\alpha^2 = \frac{36b \pm \sqrt{-3\Delta_E}}{96a} = \frac{-3}{4a}\nu_{\pm}. \tag{8.13}$$

If a is a square this means that at least one of $\nu_{\pm}$ must be a square for α to exist. On the other hand, if a is not a square then the same condition always holds since the product $\nu_+\nu_- = -a^3/27$ is a non-square.

The proof of the converse is similar: if conditions 2 and 3 are met then there exists $\alpha, \beta \in \mathbb{F}_q$ that are solutions to (8.10) and (8.11), which shows that $-g$ has a square root mod h, and then condition 1 together with (8.9) shows that $-h$ is a square mod g.

Finally, consider the special case $a = 0$. In that case, since $h(u) = 3u^2$, we can apply the change of variables $Y' = uY$ to reduce to the case of the conic:

$$X^2 + 3Y^2 + gZ^2 = 0,$$

i.e., $r = 3$ and $s = g$. It is then clear that r and s are coprime, squarefree, and one of them is of odd degree. Moreover, the condition that $-s$ is a square modulo r is vacuous, and the condition that $-r$ is a square modulo s simply says that -3 is a square in $\mathbb{F}_q[u]/(g)$; since that etale algebra admits either $\mathbb{F}_q$ or $\mathbb{F}_{q^3}$ as a factor, this is equivalent to -3 being a square in $\mathbb{F}_q$, namely $q \equiv 1 \pmod 3$ as required. This shows that in this case, condition 1 is necessary and sufficient. The result still holds, however, because conditions 2 and 3 become vacuous: the discriminant $\Delta_E = -16(27b^2) = -3 \cdot 12^2b^2$ is always a square, and one of $\nu_{\pm}$ is always zero. □

Out of the three conditions in Theorem 8.3.1, condition 1 is the most restrictive discarding half of the prime fields. Condition 3 only fails about 1/4 of the time, whereas condition 2 fails half of the time but can be circumvented half of those times as discussed in the next section. Notable curves that satisfy the conditions for SwiftEC include the NIST `P-256` curve, the curve `secp256k1` used in Bitcoin [143] and the pairing-friendly curve `BLS12-381` [144] as well as all BN curves [145] and BLS curves [146] over any field with $q \equiv 1 \mod 3$. On the other hand, curves such as the Ed448-Goldilocks curve [147] and the NIST `P-384` curve are incompatible due to the field cardinality alone.

8.3.3 Reaching more curves with 2-isogenies

While Theorem 8.3.1 discards the possibility of applying SwiftEC directly to curves with a non-square discriminant, here we present a small modification that can work around this condition, at least some of the time. The condition that the discriminant be a square is invariant under isomorphisms, but not under isogenies. Hence, we may hope that there is an isogenous curve that satisfies the condition and compose the SwiftEC encoding to

this curve with the isogeny to obtain a map to the original curve. Curves with a non-square discriminant always contain exactly one point of order 2, so one may be tempted to exploit the small 2-isogeny that is available. The following result shows that this intuition is correct, and indicates exactly when this is possible.

Theorem 8.3.2. *Let $E/\mathbb{F}_q$ be an elliptic curve with non-square discriminant. There exists a curve E' with square discriminant isogenous to E over $\mathbb{F}_q$ if and only if $E(\mathbb{F}_q)$ has a point of order 4. In this case, the isogeny can always be taken to be of degree 2.*

Proof. First suppose we have a point $P_4 \in E(\mathbb{F}_q)$ of order 4, and let $P_2 = 2P_4$ be the unique point of order 2 in $E(\mathbb{F}_q)$. If $\phi : E \to E'$ is the isogeny with kernel $< P_2 >$, then $\phi(P_4)$ is a point of order 2 in E'. There must also exist a point $P_2' \in E'(\mathbb{F}_q)$ of order 2 generating the dual isogeny $\hat{\phi}$, and we cannot have $\phi(P_4) = P_2'$ because $\hat{\phi}(P_2') = 0$ but $\hat{\phi}(\phi(P_4)) = 2P_4 \neq 0$. This means we have two distinct points of order 2 in E', and their addition yields a third point of order 2, so E' must have a square discriminant as desired.

Conversely, if E has no point of order 4 then the group order is divisible by 2 exactly once, so any isogenous curve will also have exactly one point of order 2 and hence have a non-square discriminant. □

Note that the application of the 2-isogeny is a 2-to-1 map that would make the distribution easily distinguishable from uniform. However, in essentially all cases of interest, one needs to sample points only in a specific subgroup orthogonal to the 2-torsion subgroup. For instance, consider `Curve25519` [148] which is non-compatible with our construction because it does not have a square discriminant. The curve is given by

$$E_{25519} : y^2 = x^3 + 486662x^2 + x$$

over the prime field of size $p = 2^{255} - 19$. The group order for this curve is $\#E_{25519} = 8\ell$ where ℓ is a large prime, and points in the ℓ-torsion subgroup are used in the ECDH scheme. We can use SwiftEC to map onto the 2-isogenous curve

$$E' : y^2 = x^3 - 10231483777459 2x + 398341948567736549376$$

which does satisfy all conditions of Theorem 8.3.1. By composing with the 2-isogeny generated by $P_2' = (-11679888, 0)$ and the multiplication-by-4 map, we are able to hash into the $\ell-$torsion subgroup of Curve25519 at the cost of only an additional 20 field multiplications, 7 squarings and 11 additions. This is to our knowledge the only currently known way of hashing deterministically and indistinguishably into this subgroup using a single square root.

Unfortunately, some curves remain out of reach for SwiftEC due to condition 3 alone, even with this isogeny trick. One such example is NIST curve `P-521`.

8.4 The SwiftEC Encoding

We now put everything together to present an overview of our encoding construction.

8.4.1 Efficient computation

As a proof of principle, we have prepared a Sage implementation of SwiftEC that allows adding new compatible curves in a simple way. This implementation makes explicit the number of field operations needed and uses a constant number of them, but is non-constant time to the degree that the built-in field operations are. Our implementation is freely available at https://github.com/Jchavezsaab/SwiftEC.

For curves with $a \neq 0$, the implementation makes use of the polynomials $X_0(u), Y_0(u)$ that evaluate a point in S_u as discussed in Section 8.3. Since these polynomials only depend on the curve coefficients a, b, they are precomputed and stored in the form of five field elements. On input u, t, the initial point $(X_0(u), Y_0(u)) \in S_u$ is evaluated and then a second point $(X, Y) \in S_u$ is obtained from the parametrization

$$\begin{aligned} X(u,t) &= \frac{g(u) + h(u)(Y_0(u) - tX_0(u))^2}{X_0(u)(1 + t^2 h(u))}, \\ Y(u,t) &= Y_0(u) + t(X - X_0(u)). \end{aligned} \tag{8.14}$$

In the case where $a = 0$, we have simply $g(u) = u^3 + b$ and $h(u) = 3u^2$. In this case the van Hoeij-Cremona algorithm described in Section 8.3 always yields the point at infinity $(X_0 : Y_0 : Z_0) = (\sqrt{-3} : 1 : 0)$, so the formulas for the parametrization have to be adjusted. We can skip the computation of $X_0(u), Y_0(u)$ altogether and apply the following formulas directly:

$$\begin{aligned} X(u,t) &= \frac{u^3 + b - t^2}{2t}, \\ Y(u,t) &= \frac{X(u,t) + t}{u\sqrt{-3}}. \end{aligned} \tag{8.15}$$

Finally, we apply the map ψ from (8.6) to get a point $(x_1, x_2, x_3, z) \in V(\mathbb{F}_q)$. It is not actually necessary to compute the z-coordinate of this point, and the x_i coordinates are computed projectively so that what we actually obtain is a projective triplet $(x_1 : x_2 : x_3 : \lambda)$. Note that this introduces a small bias towards the point at infinity: if any of the x_i are infinite then we have to set $\lambda = 0$ and all three points will be interpreted as being infinite. However, we neglect this since the bias is negligible and dealing with this case explicitly would produce a non-constant-time implementation.

We must then find which of the x_i is the x-coordinate of a point in $E(\mathbb{F}_q)$, choosing one arbitrarily but deterministically if all three are. This can be implemented in constant time as shown in Algorithm 1 which prioritizes x_3.

Algorithm 1 x-picking algorithm.

Input: The projective x_i coordinates $(x_1 : x_2 : x_3 : \lambda)$ of a point in $V(\mathbb{F}_q)$
Output: One of the x_i which is the x-coordinate of a point in $E(\mathbb{F}_q)$.

1: $s_2 \leftarrow x_2^3\lambda + ax_2\lambda^3 + b\lambda^4$
2: $s_3 \leftarrow x_3^3\lambda + ax_3\lambda^3 + b\lambda^4$
3: $c_2 \leftarrow IsSquare(s_2)$
4: $c_3 \leftarrow IsSquare(s_3)$
5: $cswap(c_2, x_1, x_2)$
6: $cswap(c_3, x_1, x_3)$
7: **return** $(x_1 : \lambda)$

Finally, we use a single inverse to compute the affine x-coordinate and a square root computation (the only one throughout the whole program) to recover the y-coordinate. Note that there is a free choice for the sign of y in the end, which we integrate as an additional input bit.

8.4.2 XSwiftEC: x-only computation without exponentiation

Note that the only inverse and square root needed for SwiftEC are at the very end when the affine x, y coordinates are computed. However, there are many applications where obtaining an output in x-only projective coordinates is acceptable, and these operations can be omitted. The resulting XSwiftEC algorithm requires no inversions, square roots or exponentiations of any kind, but only two Jacobi symbol computations that are considerably cheaper and other elementary field operations.

This is particularly useful for the cases when SwiftEC is composed with a 2-isogeny as described in Subsection 8.3.3: even if an affine x, y output is desired, we are better off using XSwiftEC and recovering the affine coordinates until after applying the projective x-only 2-isogeny formulas.

Although the output $(x : \lambda)$ that is obtained is indistinguishable from uniform as a projective pair, the individual values of x and λ are not and may leak information about the input. This can be easily circumvented by multiplying both coordinates by a random field element, or it may be ignored to avoid relying on randomness in applications where this leakage is not a concern.

8.5 Implementation results

We summarize in Table 8.1 the cost in operations for each version of SwiftEC. The most noteworthy feature is the requirement of only one square root computation (and none when the y coordinate is not required), which is an improvement on previous admissible encodings to ordinary elliptic curves.

Table 8.1: Cost in operations of our implementations of SwiftEC for field additions, squarings, multiplications, Jacobi symbol computations, inversions, and square roots.

	Add	Sqr	Mul	Jac	Inv	Sqrt
SwiftEC	25	7	18	2	1	1
SwiftEC with isogeny	36	14	38	2	1	1
XSwiftEC	22	9	23	2	0	0
XSwiftEC with isogeny	33	14	35	2	0	0

The results shown are for the $a \neq 0$ implementation. The implementations for $a = 0$ always save exactly 7 additions and 6 multiplications due to the simpler formulas in (8.15).

8.6 SwiftEC For Point Representation: ElligatorSwift

In this section we describe an algorithm to efficiently compute a uniformly random preimage of any point under SwiftEC. The existence of this algorithm is required for the encoding to be *admissible*, which is crucial for using SwiftEC as part of a cryptographically secure hash function as described in Section 8.1. Moreover, it is important in practice because it allows us to encode points in an elliptic curve as uniform bitstrings, as is done in Elligator [127] and Elligator Squared [53].

Compared to Elligator Squared, our ElligatorSwift construction has the advantage that it does not need to compute any encodings in the forward direction. Indeed, all we need is to sample a random $u \in \mathbb{F}_q$ and then find an inverse $F_{0,u}^{-1}(P)$ of the SW encoding.

We first focus on inverting the map Ψ and note that under a change of variables $v = x/2y - u/2$ and $w = 2y$, the image in (8.6) becomes

$$\begin{aligned} x_1 &= v \\ x_2 &= -u - v \\ x_3 &= u + w^2, \end{aligned}$$

while the equation for the conic becomes

$$w^2(u^2 + uv + v^2 + a) = -(u^3 + au + b). \tag{8.16}$$

This yields up to four possible preimages for a given point $(x, y) \in E(F)$, namely:

1. $v = x$ and w^2 derived from (8.16), if x was drawn from x_1
2. $v = -u - x$ and w^2 derived from (8.16), if x was drawn from x_2

3,4. $w^2 = x - u$ and v derived from (8.16), if x was drawn from x_3,

where the last case actually contains two preimages since (8.16) is a quadratic equation for v with solutions

$$v = \frac{-u}{2} \pm \frac{\sqrt{-w^2(4u^3 + 4au + 4b + 3w^2u^2 + 4aw^2)}}{2w^2}.$$

Moreover, all cases have a duplicity from choosing the sign of $w = \sqrt{w^2}$, so there are up to 8 preimages in total. Of course, some of the square roots needed may not exist and so different values of u will yield a different number of preimages of a given point (including possibly none). On top of this, if the preimage comes from cases 1 or 2 but results in values where all three x_i yield points in $E(\mathbb{F}_q)$, then the preimage will be invalid even if the square root is well-defined since Algorithm 1 in the forward encoding would have prioritized x_3 over the intended one. Care must therefore be taken to check for the existence of the various square roots and restart the procedure when appropriate, as shown in Algorithm 2. This makes the algorithm run in non-constant time but ensures that the preimage is uniformly sampled.

What remains is just to switch back to x, y coordinates and invert the parametrization (8.14) to recover the parameter t.

Remark: For implementations with $a = 0$ we must take into account the different parametrization formulas in (8.15). In this case, lines 29 and 30 of Algorithm 2 can be replaced by simply $t \leftarrow Yu\sqrt{-3} - X$, where the constant $\sqrt{-3}$ is part of the precomputed parameters.

We assume that the square root function makes a random choice of sign each time it is called, and that it returns *Null* for non-squares. It is easy to see that the output of Algorithm 2 is uniformly distributed since each u is attempted with a random choice of one of the 4 cases, so the probability of each u being successful is proportional to how many preimages exist under it.

The main cost of Algorithm 2 is an average of 1.5 square root computations per iteration. Since most points have roughly $2q$ preimages as per Theorem 8.2.1, we can expect each choice of u to contain on average 2 valid preimages out of the 8 possible ones, and so the expected number of iterations is 4. Notice however that a failed iteration can be aborted before computing any square roots by first computing the corresponding Jacobi symbols, which can be done much more efficiently with constant-time efficient implementations such as [136, 137, 138]. The cost of ElligatorSwift is therefore always exactly 1 or 2 square root computations, and 6 Jacobi symbol computations on average. This is a considerable improvement over Elligator Squared, where each failed iteration would have contributed an additional square root from computing the forward map and the average total cost is 6.5 square roots.

Note that in the case of x-only arithmetic there are no savings on ElligatorSwift since Algorithm 2 is already agnostic to y but still requires several square roots. As for

Algorithm 2 ElligatorSwift.

Input: $(x, y) \in E(\mathbb{F}_q)$

Output: $u, t, b \xleftarrow{\$} \text{SwiftEC}^{-1}(x, y)$

$u \xleftarrow{\$} \mathbb{F}_q$
$case \xleftarrow{\$} \{1, 2, 3, 4\}$
if $case == 1$ **then**
    $v \leftarrow x$
    if IsSquare$((-v - u)^3 + a(-v - u) + b)$ **then**
        go to 1
    end if
    $w^2 \leftarrow -(u^3 + au + b)/(u^2 + uv + v^2 + a)$
else if $case == 2$ **then**
    $v \leftarrow -x - u$
    if IsSquare$(v^3 + av + b)$ **then**
        go to 1
    end if
    $w^2 \leftarrow -(u^3 + au + b)/(u^2 + uv + v^2 + a)$
else
    $w^2 \leftarrow x - u$
    $r \leftarrow \sqrt{-w^2(4u^3 + 4au + 4b + 3w^2u^2 + 4aw^2)}$
    if $r == Null$ **then**
        go to 1
    end if
    $v \leftarrow -u/2 + r/2w^2$
end if
$w \leftarrow \sqrt{w^2}$
if $w == Null$ **then**
    go to 1
end if
$Y \leftarrow w/2$
$X \leftarrow 2Y(v + u/2)$
Evaluate $X_0(u)$ and $Y_0(u)$ from precomputed polynomials
$t \leftarrow (Y - Y_0)/(X - X_0)$
$b \leftarrow sign(y)$
return u, t, b

curves where we need to compose SwiftEC with a 2-isogeny, we can obtain a corresponding variant of ElligatorSwift by composing with the dual isogeny, but this has the side effect of introducing a multiplication-by-2 in the round trip. This can be circumvented by starting with a point halving before applying ElligatorSwift, which is important for demonstrating that the encoding with the isogeny trick is still admissible. However, the resulting ElligatorSwift construction is unappealing in terms of efficiency.

8.7 Discussion

The SwiftEC construction presented in this chapter is the first admissible and constant-time encoding using a single square root that is applicable to a large class of ordinary elliptic curves. While some curves are still incompatible, the construction applies to roughly $9/16$ of all curves over fields with $q \equiv 1 \mod 3$. The inverse encoding also results in an Elligator-like encoding that is significantly more efficient than previous constructions, using more than 4 times less square roots on average than Elligator Squared, while retaining the same data transmission size of two field elements.

It is still an open problem to determine if there are any workarounds that could extend this encoding to more of the non-compatible curves, or even to find a single-square root admissible encoding that could be applied to all ordinary elliptic curves.

Chapter 9

Conclusions and Future Work

This project has had important contributions to the development of post-quantum cryptography and specifically to isogeny-based cryptography, as well as classical elliptic curve cryptography as a side result. The main contributions can be summarized as:

- We have contributed vastly to the quantum cryptanalysis of isogeny-based cryptography and developed a methodology for simulating circuits that attack CSIDH.
- We have proposed new instances of the protocol with concrete sets of parameters that were derived from our analysis.
- We have provided several constant-time C implementations that are the first of their kind for various algorithms relevant to isogeny-based cryptography, including the faster Vélu formulas and the CSIDH protocol as a whole.
- We have presented large-scale parallel implementations for classical attacks on isogeny-based cryptography to provide concrete estimates for its security
- We have exploited the use of the isogeny framework in other cryptographic primitives, and adapted it to other paradigms such as that of SNARGs.
- We have presented the first verifiable delay function with post-quantum security, using the isogeny framework.
- We have used other mathematical techniques, combined with isogenies, to present the most efficient encoding known to date onto ordinary elliptic curves.

Derived from the works in this project, there are still a number of topics that can branch out as future work, as well as other topics already being worked on. We conclude the manuscript with a short overview of each of these topics.

Implementing CSIDH into the TLS protocol
While the performance of CSIDH has consistently been found to be far behind other post-quantum candidates, its status as the only non-interactive public key encapsulation candidate can be exploited by an alternative to the TLS protocol called OPTLS, which requires one less round-trip of communication to establish a connection. Whether or not this benefit outweighs CSIDH's poor performance can only be assessed by integrating CSIDH to a full implementation of OPTLS, which is already being worked on in a collaboration of this project.

Optimizing the sublinear Vélu formulas with parallelization
The new sublinear Vélu formulas, which we have been the first to implement in constant-time in C in our *SQALE of CSIDH* project, also allow for a good degree of parallelization that has not yet been exploited. As part of another collaboration, we aim to make an implementation in parallel using OpenMP to benchmark the parallelization savings versus the cost of the OpenMP launch, both in the CSIDH setting and in SIDH variants.

Extending the set of applicable curves for our SwiftEC encoding
Our encoding construction is only applicable to about half the ordinary elliptic curves depending on the three conditions of Theorem 8.3.1. However, an extended version of the work is already being considered to include a method for bypassing condition 3 almost all the time by using odd isogenies to find a similar curve that meets it, and we are also studying an early idea that could bypass condition 1 by working with a quadratic field extension but reducing to the original field before having to do the exponentiation.

Implementing our VDF construction
As was stressed in Chapter 7, performing the SNARG proof at the field arithmetic level should save significantly on overhead, but we have not provided any quantitative estimates as to how much. In a subsequent work, Cong *et al.* [34] have used our framework to implement a proof of knowledge. While their proof is not exactly the same as ours, they have obtained promising results showing that the overhead is indeed not unreasonably large. Therefore, a task that is currently not being worked on but could be considered for future work is to provide an implementation of the VDF at least for small parameters, which should help assess how close we are to a practical application.

List of publications

The publications resulting from this project are three conference talks and one journal article:

[4] **J. Chavez-Saab**, JJ. Chi-Domínguez, S. Jaques ✉, F. Rodríguez-Henríquez. The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. *J Cryptogr Eng* **12**, 349-368 (2022). https://doi.org/10.1007/s13389-021-00271-w

[5] E. Bellini, **J. Chavez-Saab** ✉, JJ. Chi-Domínguez, A. Esser, S. Ionica, L. Rivera-Zamarripa, F. Rodríguez-Henríquez, M. Trimoska, F. Zweydinger (2022). In: Isobe, T., Sarkar, S. (eds) Progress in Cryptology – INDOCRYPT 2022. *INDOCRYPT 2022*. Lecture Notes in Computer Science, vol 13774. Springer, Cham. https://doi.org/10.1007/978-3-031-22912-1_13

[6] **J. Chavez-Saab** ✉, F. Rodríguez-Henríquez, M. Tibouchi (2022). Verifiable Isogeny Walks: Towards an isogeny-based postquantum VDF. In: AlTawy, R., Hülsing, A. (eds) Selected Areas in Cryptography. *SAC 2021*. Lecture Notes in Computer Science, vol 13203. Springer, Cham. https://doi.org/10.1007/978-3-030-99277-4_21

[7] **J. Chavez-Saab**, F. Rodríguez-Henríquez, M. Tibouchi ✉(2022). SwiftEC: Shallue–van de Woestijne Indifferentiable Function to Elliptic Curves. In: Agrawal, S., Lin, D. (eds) Advances in Cryptology – ASIACRYPT 2022. *ASIACRYPT 2022*. Lecture Notes in Computer Science, vol 13791. Springer, Cham. https://doi.org/10.1007/978-3-031-22963-3_3

Bibliography

[1] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[2] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.

[3] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427, Cham, 2018. Springer International Publishing.

[4] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, 2021.

[5] Emanuele Bellini, Jorge Chavez-Saab, Jesús-Javier Chi-Domínguez, Andre Esser, Sorina Ionica, Luis Rivera-Zamarripa, Francisco Rodríguez-Henríquez, Monika Trimoska, and Floyd Zweydinger. Parallel isogeny path finding with limited memory. In Takanori Isobe and Santanu Sarkar, editors, *Progress in Cryptology – INDOCRYPT 2022*, pages 294–316, Cham, 2022. Springer International Publishing.

[6] Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 441–460, Cham, 2022. Springer International Publishing.

[7] Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Swiftec: Shallue-van de woestijne indifferentiable function to elliptic curves. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 63–92, Cham, 2022. Springer Nature Switzerland.

[8] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[9] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2019.

[10] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n+2 qubits with Toffoli based modular multiplication. *arXiv e-prints*, page arXiv:1611.07995, Nov 2016.

[11] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. *arXiv e-prints*, page arXiv:1706.06752, Jun 2017.

[12] Lov K. Grover. A Fast quantum mechanical algorithm for database search. 1996. https://arxiv.org/abs/quant-ph/9605043.

[13] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[14] NIST. NIST Post-Quantum Cryptography Standardization Process. Third Round Candidates, July 2020.

[15] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. Third Round Candidate of the NIST's post-quantum cryptography standardization process, 2020. Available at: https://sike.org/.

[16] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, volume 10322 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2017.

[17] Daniel J. Bernstein et al. Sphincs+ – submission to the 2nd round of the nist post-quantum project. 2019. Available at https://sphincs.org/resources.html.

[18] Daniel J. Bernstein et al. Classic mceliece: conservative code-based cryptography, 2017. Availabel at https://classic.mceliece.org/nist/mceliece-20171129.pdf.

[19] NTRUEncrypt and pqNTRUsign submissions to NIST. Available at https://www.onboardsecurity.com/nist-post-quantum-crypto-submission.

[20] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[21] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh (preliminary version). Cryptology ePrint Archive, Paper 2022/975, 2022. https://eprint.iacr.org/2022/975.

[22] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin, Heidelberg, 2003.

[23] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Chapman & Hall/CRC, 2 edition, 2008.

[24] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones Mathematicae*, 22:134—-144, 1966.

[25] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *IACR Cryptol. ePrint Arch.*, 2020:341, 2020.

[26] Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.*, 8(1):1–29, 2014.

[27] Arnold K. Pizer. Ramanujan graphs and Hecke operators. *Boulletin of the American Mathematical Society*, 23:127–137, 1990.

[28] David Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, UC Berkeley, December 1996.

[29] Max Deuring. Die typen der multiplikatorenringe elliptischer funktionenkörper. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 14:197–272, 1941.

[30] Craig Costello. B-sidh: Supersingular isogeny diffie-hellman using twisted torsion. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 440–463, Cham, 2020. Springer International Publishing.

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

[32] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.

[33] Luca De Feo and Samuel Dobson and Steven D. Galbraith and Lukas Zobernig. SIDH proof of knowledge. *IACR Cryptol. ePrint Arch.*, page 1023, 2021. To appear in ASIACRYPT 2022.

[34] Kelong Cong, Yi-Fu Lai, and Shai Levin. Efficient isogeny proofs using generic techniques. Cryptology ePrint Archive, Paper 2023/037, 2023. https://eprint.iacr.org/2023/037.

[35] Tomoki Moriya. Masked-degree sidh. Cryptology ePrint Archive, Paper 2022/1019, 2022. https://eprint.iacr.org/2022/1019.

[36] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 302–326. Springer, 2021.

[37] Tako Boris Fouotsa. SIDH with masked torsion point images. Cryptology ePrint Archive, Paper 2022/1054, 2022. https://eprint.iacr.org/2022/1054.

[38] Luciano Maino and Chloe Martindale. An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Paper 2022/1026, 2022. https://eprint.iacr.org/2022/1026.

[39] Damien Robert. Breaking SIDH in polynomial time. Cryptology ePrint Archive, Paper 2022/1038, 2022. https://eprint.iacr.org/2022/1038.

[40] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, volume 8885 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2014.

[41] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343. Springer, 2018.

[42] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[43] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of sike in practice. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 505–534, Cham, 2020. Springer International Publishing.

[44] Aleksei Udovenko and Giuseppe Vitto. Breaking the $ikep182 challenge. Cryptology ePrint Archive, Paper 2021/1421, 2021. https://eprint.iacr.org/2021/1421.

[45] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathfrak{O}_{\mathrm{p}}$. *Des. Codes Cryptogr.*, 78(2):425–440, 2016.

[46] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. Csi-fish: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[47] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.

[48] G. Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC 2013*, LIPIcs 22, pages 20–34, 2013.

[49] Chris Peikert. He gives c-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492. Springer, 2020.

[50] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020-Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 493–522. Springer, 2020.

[51] Quantum security analysis of aes. 2019.

[52] Andrew Shallue and Christiaan E. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *Algorithmic Number Theory, 7th International Symposium, ANTS-VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006.

[53] Mehdi Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 139–156. Springer, Heidelberg, March 2014.

[54] Patrick Longa. Practical quantum-resistant key exchange from supersingular isogenies and its efficient implementation. *Latincrypt 2019* Invited Talk., 2019.

[55] Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space, June 2004.

[56] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. *IACR Cryptol. ePrint Arch.*, 2020:417, 2020.

[57] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. On new Vélu's formulae and their applications to CSIDH and B-SIDH constant-time implementations. *IACR Cryptol. ePrint Arch.*, 2020:1109, 2020.

[58] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 303–329. Springer, 2017.

[59] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, volume 11356 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2018.

[60] Dustin Moody and Daniel Shumow. Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. *Math. Comput.*, 85(300):1929–1951, 2016.

[61] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019*, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019.

[62] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019.

[63] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (short paper) A faster constant-time algorithm of CSIDH keeping two points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019*, volume 11689 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2019.

[64] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[65] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 32–61. Springer, 2019.

[66] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, 2019.

[67] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[68] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. A trade-off between classical and quantum circuit size for an attack against CSIDH. *Journal of Mathematical Cryptology*, pages 1–16, August 2019.

[69] Xavier Bonnetain. Improved Low-qubit Hidden Shift Algorithms. working paper or preprint, December 2019.

[70] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Phys. Rev. X*, 8:041015, Oct 2018.

[71] Dominic W. Berry, Craig Gidney, Mario Motta, Jarrod R. McClean, and Ryan Babbush. Qubitization of Arbitrary Basis Quantum Chemistry Leveraging Sparsity and Low Rank Factorization. *Quantum*, 3:208, December 2019.

[72] (https://stats.stackexchange.com/users/173082/ben) Ben O'Neill. Distribution of urns for non-uniform distribution. Cross Validated. (version: 2020-05-06).

[73] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 425–444, Cham, 2020. Springer International Publishing.

[74] Craig Gidney. Spooky pebble games and irreversible uncomputation. (2019, Aug 19).

[75] Emanuel Knill. An analysis of Bennett's pebble game, 1992.

[76] John M. Schanck. *Improving post-quantum cryptography through cryptanalysis*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2020.

[77] Samuel Jaques and André Schrottenloher. Low-gate quantum golden collision finding. In Michael J. Jacobson Jr., Orr Dunkelman, and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, Lecture Notes in Computer Science. Springer, 2020.

[78] James H. Davenport and Benjamin Pring. Improvements to quantum search techniques for block-ciphers, with applications to AES. In Michael J. Jacobson Jr., Orr Dunkelman, and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, Lecture Notes in Computer Science. Springer, 2020.

[79] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020.

[80] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277. Springer, 2019.

[81] Steven D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS J. Comput. Math*, 2:118–138, 1999.

[82] Craig Costello Maria Corte-Real Santos and Jia Shi. Accelerating the Delfs-Galbraith algorithm with fast subfield root detection. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 285–314. Springer, 2022.

[83] Craig Costello. The case for SIKE: A decade of the supersingular isogeny problem. *IACR Cryptol. ePrint Arch.*, page 543, 2021.

[84] Christine van Vredendaal. Reduced memory meet-in-the-middle attack against the ntru private key. *LMS Journal of Computation and Mathematics*, 19(A):43–57, 2016.

[85] Claire Delaplace, Andre Esser, and Alexander May. Improved low-memory subset sum and lpn algorithms via multiple collisions. In *IMA International Conference on Cryptography and Coding*, pages 178–199. Springer, 2019.

[86] Andre Esser and Alexander May. Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 94–122. Springer, 2020.

[87] Alexander May. How to meet ternary lwe keys. In *Annual International Cryptology Conference*, pages 701–731. Springer, 2021.

[88] Monika Trimoska, Sorina Ionica, and Gilles Dequen. Time-memory analysis of parallel collision search algorithms. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):254–274, 2021.

[89] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[90] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020*, volume 12238 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2020.

[91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 21–32, New York, NY, USA, 1991. Association for Computing Machinery.

[92] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, October 2000.

[93] Antonin Leroux. Proofs of isogeny knowledge and application to post-quantum one-time verifiable random function. *IACR Cryptol. ePrint Arch.*, 2021:744, 2021.

[94] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In

Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[95] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*. The Internet Society, 2015.

[96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, MIT, 1996. Available at: https://tinyurl.com/time-lock-puzzles.

[97] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017.

[98] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467. Springer, 2018.

[99] Chia Network Collaboration. Chia DAQ. Chia network, 2021. Available at: https://www.chia.net/faq/.

[100] Justin Drake. Minimal VDF randomness beacon. ETH Research, 2018. Available at: https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566.

[101] Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. Towards uncheatable benchmarks. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11. IEEE Computer Society, 1993.

[102] VDF Alliance. VDF research. VDF Alliance, 2021. Available at: https://vdfresearch.org/.

[103] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[104] Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, 2020.

[105] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13. Solana cryptocurrency whitepaper, 2020. Available at: https://tinyurl.com/solana-whitepaper.

[106] Fanyu Kong, Zhun Cai, Jia Yu, and Daxing Li. Improved generalized Atkin algorithm for computing square roots in finite fields. *Inf. Process. Lett.*, 98(1):1–5, 2006.

[107] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[108] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[109] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[110] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[111] Jorge Chávez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. *IACR Cryptol. ePrint Arch.*, page 1289, 2021.

[112] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[113] Gora Adj, Omran Ahmadi, and Alfred Menezes. On isogeny graphs of supersingular elliptic curves over finite fields. *Finite Fields and Their Appl.*, 55:268–283, 2019.

[114] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion ℓ-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014.

[115] Christophe Petit and Kristin E. Lauter. Hard and easy problems for supersingular isogeny graphs. *IACR Cryptol. ePrint Arch.*, 2017:962, 2017.

[116] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to elliptic curves. https://tools.ietf.org/id/draft-irtf-cfrg-hash-to-curve-14.html, February 2022.

[117] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

[118] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

[119] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Heidelberg, May 2011.

[120] Maciej Ulas. Rational points on certain hyperelliptic curves over finite fields. *Bull. Pol. Acad. Sci. Math.*, 55(2):97–104, 2007.

[121] Thomas Icart. How to hash into elliptic curves. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 303–316. Springer, Heidelberg, August 2009.

[122] Jean-Gabriel Kammerer, Reynald Lercier, and Guénaël Renault. Encoding points on hyperelliptic curves over finite fields in deterministic polynomial time. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, pages 278–297. Springer, Heidelberg, December 2010.

[123] Pierre-Alain Fouque and Mehdi Tibouchi. Deterministic encoding and hashing to odd hyperelliptic curves. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, pages 265–277. Springer, Heidelberg, December 2010.

[124] Reza Rezaeian Farashahi. Hashing into Hessian curves. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 278–289. Springer, Heidelberg, July 2011.

[125] Pierre-Alain Fouque and Mehdi Tibouchi. Indifferentiable hashing to Barreto-Naehrig curves. In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 1–17. Springer, Heidelberg, October 2012.

[126] Pierre-Alain Fouque, Antoine Joux, and Mehdi Tibouchi. Injective encodings to elliptic curves. In Colin Boyd and Leonie Simpson, editors, *ACISP 13*, volume 7959 of *LNCS*, pages 203–218. Springer, Heidelberg, July 2013.

[127] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.

[128] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES*, 2019(4):154–179, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8348.

[129] Reza Rezaeian Farashahi, Pierre-Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and José Felipe Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Math. Comput.*, 82(281):491–512, 2013.

[130] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[131] Mehdi Tibouchi. Impossibility of surjective Icart-like encodings. In Sherman S. M. Chow, Joseph K. Liu, Lucas C. K. Hui, and Siu-Ming Yiu, editors, *ProvSec 2014*, volume 8782 of *LNCS*, pages 29–39. Springer, Heidelberg, October 2014.

[132] Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. *Des. Codes Cryptogr.*, 82(1-2):161–177, 2017.

[133] Dmitrii Koshelev. Indifferentiable hashing to ordinary elliptic $\mathbb{F}_q$-curves of $j = 0$ with the cost of one exponentiation in $\mathbb{F}_q$. *Des. Codes Cryptogr.*, 90(3):801–812, 2022.

[134] Mark van Hoeij and John Cremona. Solving conics over function fields. *Journal de Théorie des Nombres de Bordeaux*, 18(3):595–606, 2006.

[135] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[136] Thomas Pornin. Faster modular inversion and Legendre symbol, and an X25519 speed record. https://research.nccgroup.com/2020/09/28/faster-modular-inversion-and-legendre-symbol-and-an-x25519-speed-record/, September 2020.

[137] Mike Hamburg. Computing the Jacobi symbol using Bernstein–Yang. Cryptology ePrint Archive, Paper 2021/1271, 2021. https://eprint.iacr.org/2021/1271.

[138] Diego F. Aranha and Conrado P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic/blob/symbol-asm/src/fp/relic_fp_smb.c, 2021.

[139] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR TCHES*, 2019(3):340–398, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8298.

[140] Marc Perret. Multiplicative character sums and kummer coverings. *Acta Arith.*, 59:279–290, 1991.

[141] Francis N. Castro and Carlos J. Moreno. Mixed exponential sums over finite fields. *Proc. Amer. Math. Soc.*, 128(9):2529–2537, 2000.

[142] Michael Rosen. *Number Theory in Function Fields*. Springer New York, NY, 2002.

[143] Certicom research, standards for efficient cryptography 2: Recommended elliptic curve domain parameters, January 2010.

[144] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, 2017.

[145] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

[146] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003.

[147] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. https://eprint.iacr.org/2015/625.

[148] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.