YAŞAR UNIVERSITY

GRADUATE SCHOOL

MASTER THESIS

# EFFICIENT COMPUTATION OF ELLIPTIC CURVE PRIMITIVES

MERT YASSI

THESIS ADVISOR: ASSIST. PROF. (PHD) HÜSEYİN HIŞIL

COMPUTER ENGINEERING

PRESENTATION DATE: 11.08.2022

BORNOVA / İZMİR

AUGUST 2022

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

| **Jury Members:** | **Signature:** |
|---|---|
| Prof. Ahmet Hasan KOLTUKSUZ, Ph.D.<br>Yaşar University | . . . . . |
| Assist. Prof. Erdem ALKIM, Ph.D.<br>Dokuz Eylül University | . . . . . |
| Assist. Prof. Hüseyin HIŞIL, Ph.D.<br>Yaşar University | . . . . . |

Prof. Yücel ÖZTÜRKOĞLU, Ph.D.
Director of the Graduate School

# ABSTRACT

EFFICIENT COMPUTATION OF ELLIPTIC CURVE PRIMITIVES

Yassı, Mert
MSc, Computer Engineering
Advisor: Assist. Prof. Hüseyin HIŞIL, Ph.D.
August 2022

In elliptic curve cryptography, it is of great importance to be able to perform computations associated with elliptic curves efficiently. To achieve this, the primitives of the elliptic curve cryptography, such as elliptic curve scalar multiplication and elliptic curve point counting, should be done using fast methods. Elliptic curve scalar multiplication is the process of multiplying a non-trivial point on an elliptic curve with a scalar. It is one of the bottleneck-forming challenges in elliptic curve cryptography, and different efficient methods, such as the Montgomery ladder, are used to overcome this challenge. A fast 4-way vectorized Montgomery ladder that works with all types of Montgomery curves is presented in the first part of this thesis. One of the other fundamental primitives emphasized in this thesis is the point counting operation on elliptic curves. Accurately determining the number of points on an elliptic curve defined over a finite field $\mathbb{F}_p$ has always been an essential and exhausting task. The difficulty level of the discrete logarithm problem, which is used in elliptic curve cryptography, and thus its security, is related to the number of points on the elliptic curve. Therefore, point counting is an essential operation in choosing a safe curve. In 1985, the first polynomial time elliptic curve point counting algorithm was found by René Schoof. The developments of Atkin and Elkies followed this, and in 1995, the SEA algorithm was introduced. Even today, improved versions of the SEA algorithm are used to find the number of points on an elliptic curve defined over a large finite field. Detecting whether two elliptic curves are isogenous is related to point counting, and this detection is generally done using the SEA algorithm. According to Sato-Tate's Isogeny Theorem from 1966, two elliptic curves are isogenous if and only if the numbers of points on them are equal. Based on this theorem, Schoof's and SEA algorithms are explained in detail, the implementations of these algorithms are given, and a new early abort method for detecting (non-)isogenous elliptic curves is presented in the second part of this thesis.

**Keywords:** Elliptic curve scalar multiplication, Montgomery ladder, Kummer lines, Elliptic curve point counting, Schoof's algorithm, SEA algorithm, trace computation, modular polynomials, isogeny, isogenous curves

# ÖZ

## ELİPTİK EĞRİ PRİMİTİFLERİNİN VERİMLİ HESAPLANMASI

Yassı, Mert
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Dr. Öğr. Üyesi Dr. Hüseyin HIŞIL
Ağustos 2022

Eliptik eğri kriptografisinde, eliptik eğriler ile ilgili hesaplamaların verimli yapılabilmesi büyük önem taşımaktadır. Bunu başarmak için, eliptik eğri skaler çarpması ve eliptik eğri nokta sayımı gibi eliptik eğri kriptografisinin primitifleri hızlı yöntemler kullanılarak yapılmalıdır. Eliptik eğri skaler çarpımı eliptik bir eğri üzerindeki basit olmayan bir noktayı bir skaler ile çarpma işlemidir. Bu, eliptik eğri kriptografisinde darboğaz oluşturan zorluklardan biridir ve Montgomery merdiveni gibi verimli yöntemler bu zorluğun üstesinden gelmek için kullanılır. Bu tezin ilk bölümünde, tüm Montgomery eğrileri ile çalışan hızlı 4 kanallı vektörleştirilmiş bir Montgomery merdiveni sunulmaktadır. Bu tezde üzerinde durulan diğer temel primitiflerden biri de eliptik eğriler üzerindeki noktaları sayma işlemidir. Sonlu bir $\mathbb{F}_p$ alanı üzerinde tanımlanan bir eliptik eğri üzerindeki noktaların sayısını doğru bir şekilde belirlemek, her zaman gerçekleştirmesi önemli ve yorucu bir görev olmuştur. Eliptik eğri kriptografisinde kullanılan ayrık logaritma probleminin zorluk derecesi, ve dolayısıyla güvenliği, eliptik eğri üzerindeki nokta sayısı ile doğrudan ilişkilidir. Bu nedenle, güvenli bir eğri seçiminde nokta sayımı büyük önem taşımaktadır. 1985 yılında, ilk polinomsal zamanda çalışan eliptik eğri nokta sayma algoritması René Schoof tarafından bulunmuştur. Bunu Atkin ve Elkies'in geliştirmeleri izlemiştir ve 1995'te SEA algoritması ortaya konmuştur. Bugün bile, geniş bir sonlu alan üzerinde tanımlanan bir eliptik eğri üzerindeki noktaların sayısını bulmak için SEA algoritmasının geliştirilmiş versiyonları kullanılmaktadır. İki eliptik eğrinin izojen olup olmadığının tespiti, nokta sayımı ile doğrudan ilişkilidir ve bu tespit genellikle SEA algoritması kullanılarak yapılır. Sato-Tate'in 1966 tarihli İzojeni Teoremi'ne göre, iki eliptik eğri, ancak ve ancak üzerlerindeki noktaların sayısı eşitse izojendir. Bu teoreme dayanarak, bu tezin ikinci bölümünde Schoof'un ve SEA algoritmaları ayrıntılı olarak açıklanmış, ilgili uygulamalar verilmiş ve izojen eliptik eğrileri tespit etmek için yeni bir erken iptal yöntemi sunulmuştur.

**Anahtar Kelimeler:**Eliptik eğri skaler çarpma, Montgomery merdiveni, Kummer doğruları, Eliptik eğri nokta sayımı, Schoof'un algoritması, SEA algoritması, iz hesaplama, izojeni, izojen eğriler

## ACKNOWLEDGEMENTS

First of all, I would like to denote my most profound appreciation to my advisor Assist. Prof. Dr. Hüseyin HIŞIL. I have worked with Dr HIŞIL on elliptic curves and cryptography since my sophomore year. Without his support and deep knowledge of elliptic curves, I would not be able to overcome the problems I faced while writing this thesis. I would like to sincerely thank him for being an outstanding supervisor by showing great patience and effort during my education progress.

Secondly, I want to thank my beloved wife and best friend, Benan, who always stood by me throughout this challenging period. I owe a lot to her love, care and constant support.

I would also like to express my deepest gratitude to my family. My parents, Lale and Halil, and my brother, my "dude" Ege, were always there for me. They were very supportive and caring in all respects.

I additionally want to thank my cat, my "son" Osman. He always boosted my mood thanks to his intense purring and extreme cuteness.

Last but not least, I would like to thank my office mates, my friends and Şefik, who manages to make me laugh every time.

Mert YASSI
İzmir, 2022

## TEXT OF OATH

I declare and honestly confirm that my study, titled "EFFICIENT COMPUTATION OF ELLIPTIC CURVE PRIMITIVES" and presented as a Master's Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Mert Yassı

Signature:

_____

August 11, 2022

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# LIST OF CODES

# SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

| | |
|---|---|
| ECC | Elliptic Curve Cryptography |
| ECDLP | Elliptic Curve Discrete Logarithm Problem |
| SIMD | Single-Instruction Multiple-Data |
| AVX | Advanced Vector Extensions |
| SEA | Schoof-Elkies-Atkin |
| AGM | Arithmetic-Geometric Mean |
| ECM | Elliptic Curve Factorization Method |
| ECDH | Elliptic Curve Diffie–Hellman |
| RSA | Rivest-Shamir-Adleman |
| CPU | Central Processing Unit |
| GCD | Greatest Common Divisor |
| CRT | Chinese Remainder Theorem |
| PGL | Projective General Linear Group |
| BSGS | Baby-Step Giant-Step |

SYMBOLS:

| | |
|---|---|
| $E$ | Elliptic curve |
| $\#E$ | Number of points on $E$ |
| $E[\ell]$ | $\ell$-th torsion group of $E$ |
| $\mathbb{Z}$ | Set of integers |
| $GF$ | Galois Field |
| $\mathbb{F}_p$ | Finite field with prime order $p$ |
| $\mathbb{Q}$ | Field of rational numbers |
| $\mathbb{C}$ | Field of complex numbers |

| | |
|---|---|
| $\mathbb{R}$ | Field of real numbers |
| $\mathcal{O}$ | Point at infinity |
| $\psi_\ell$ | $\ell$-th division polynomial |
| $\Phi_\ell$ | $\ell$-th modular polynomial |
| $\Lambda$ | Lattice |
| $\wp$ | Weierstrass elliptic function |
| $\mathcal{H}$ | Upper half-plane |
| $\mathcal{F}$ | Fundamental domain |
| $\pi$ | Frobenius endomorphism |
| $\rho$ | Characteristic polynomial |
| $\zeta$ | Primitive $r$-th root of unity |
| $[[q]]$ | Laurent series with variable $q$ |
| $\mathfrak{B}$ | Prime ideal |

# CHAPTER 1
# INTRODUCTION

Various cryptographic components are widely used to maintain an adequate level of security in digital communication, such as key exchange schemes, digital signatures, hash functions and secure pseudorandom number generators. These components are some basic building blocks that make up the concepts of symmetric and asymmetric (public-key) cryptography. The public-key cryptography relies on creating two separate keys and keeping one of the keys private for the creator while allowing the public use of the other. The main idea to consider here is the reciprocal encryption-decryption mechanism, i.e., a message encrypted using a public key can only be decrypted using a private key, and vice-versa applies. Detailed information about public-key cryptography can be found in (Menezes, Katz, van Oorschot, & Vanstone, 2018).

Elliptic curve cryptography (ECC) is one of the fundamental concepts of public-key cryptography. It is mainly based on elliptic curves over finite fields. The foundations of ECC were laid by the independent works of Miller (1985) and Koblitz (1987) on elliptic curves. ECC gains an advantage over other alternatives by using small-sized keys to provide the same level of security, increasing its popularity. In ECC, there is a problem named the elliptic curve discrete logarithm problem (ECDLP). This problem is a computational problem defined on an elliptic curve $E/\mathbb{F}_p$ where $P$ and $Q$ are two given points in $E(\mathbb{F}_p)$. The security of ECDLP is ensured with the difficulty of finding an integer $k$ that satisfies $Q = [k]P$. Here, point $Q$ is found with elliptic curve scalar multiplication methods. Performing an efficient elliptic curve scalar multiplication routine is one of the significant challenges faced in ECC. Montgomery (1987) proposed the Montgomery ladder algorithm to multiply points on a Montgomery curve (i.e., a particular elliptic curve used in efficient implementations) in constant time. Montgomery curves allow working only with the $x$-coordinate, speeding up the computations. This property has increased the popularity of Montgomery curves, and various Montgomery ladder-based implementations currently exist. Recently, these ladders are being implemented using single-instruction multiple-data (SIMD) instructions and multiple channel vectorizations, as architectures that allow parallel computation have become widespread. The vectorized implementations of the Montgomery ladder are generally gathered around 2-way vectorization. Karati and Sarkar (2017) stated that the Montgomery ladder structure is unsuitable for producing a sufficiently parallelized SIMD implementation.

However, it was seen that the Montgomery ladder could be efficiently 4-way vectorized by performing proper arrangements in the ladder shape. Therefore, utilizing AVX2 and AVX-512 instruction sets and performing 4-way SIMD operations in implementing the Montgomery ladder would make perfect sense.

Returning to ECDLP, the elliptic curve chosen for use should be defined on a reasonably large finite field to provide adequate security. In this way, the cardinality of $E(\mathbb{F}_p)$ will be high, and it would be harder to find the required integer $k$. Therefore, it is crucial to correctly determine the number of points on an elliptic curve to ensure the safety of ECDLP. Initially, a naive approach was used to count the points on an elliptic curve. The naive point counting method is the process of selecting points in the field that satisfy the curve equation. Although it could be used when the field size on which the elliptic curve is defined is small, the need for more efficient methods has increased as the field size has grown. At first, Shanks (1971) developed a more efficient algorithm named Baby-step Giant-step (BSGS) to count points on an elliptic curve. This algorithm and its variants were extensively used in point counting until Schoof (1985) found a point counting algorithm that works in polynomial time. Schoof's algorithm had become a pioneer in point counting and led to numerous improvements in space and time efficiency. Thanks to the developments of Atkin and Elkies, Schoof (1995) introduced the Schoof-Elkies-Atkin (SEA) algorithm. SEA algorithm employs lower degree modular polynomials in the computations, unlike Schoof's algorithm, which utilizes division polynomials with much higher degrees. The efficiency of the SEA algorithm comes from this improvement, and it is still the most efficient algorithm used for counting points on elliptic curves defined over very large finite fields.

A structure-preserving map can be defined between two elliptic curves. These maps are called isogenies. If there exists an isogeny between two elliptic curves, they are identified as isogenous curves. Tate (1966) showed that two elliptic curves are isogenous if and only if the numbers of points on these elliptic curves are equal. Therefore, it is reasonable to use the SEA algorithm to check whether the elliptic curves are isogenous since it is the most efficient algorithm used in elliptic curve point counting.

In this thesis, at first, preliminary information will be given for creating the necessary infrastructure to understand upcoming chapters about elliptic curve scalar multiplication and point counting methods. Then, a fast 4-way vectorized Montgomery ladder that works with the complete set of Montgomery curves will be introduced. Next, Schoof's and SEA algorithms will be explained in detail and illustrated with toy examples. Finally, the thesis will be concluded by introducing an early abort method for detecting both isogenous and non-isogenous curves, based on checking the equality of small traces obtained in the Elkies parts of the SEA algorithm.

## 1.1. MOTIVATION

Protocols and applications derived from the ECC are widely used in industry, and optimizing these protocols and applications as much as possible is of great importance. Therefore, it will be helpful to improve the efficiency of the elliptic curve primitives of the ECC. The primary motivation for writing this thesis is to provide efficient methods for the computation of elliptic curve primitives, such as elliptic curve scalar multiplication and elliptic curve point counting. At first, the subject of elliptic curve scalar multiplication was studied, and consequently, a new 4-way vectorized Montgomery Ladder was introduced. Then, the problem of counting points on elliptic curves was discussed. Brief research on the methods of counting points was done, and Schoof's algorithm was found interesting for investigation. After understanding Schoof's algorithm, efficient and up-to-date point counting algorithms were examined, and the focus was shifted to the SEA algorithm.

While searching the areas where the SEA algorithm is used, it was seen that Bosma, Cannon, and Playoust (1997) use the SEA algorithm in the `IsIsogenous` function to check whether the elliptic curves are isogenous. Another motivation is to find an alternative method for detecting (non-)isogenous curves by thoroughly investigating the SEA algorithm. This investigation was strengthened by implementing the algorithm. To produce comprehensive and informative public implementations of Schoof's and SEA algorithms is another motivation for this thesis.

## 1.2. AIMS AND OUTCOMES

The objectives intended to be achieved by writing this thesis are as follows:

- Producing an efficient implementation of the Montgomery ladder by utilizing SIMD architectures and parallelism.
- Elaborately explaining efficient methods used to determine the number of points on elliptic curves, such as Schoof's and SEA algorithms.
- Studying the improvements that will speed up the SEA algorithm in theory and practice.
- Presenting implementations of Schoof's and SEA algorithms.
- Investigating the methods for detecting (non-)isogenous curves.

Based on these aims, the following outcomes were obtained as a result of this thesis:

- A new 4-way vectorized Montgomery ladder that works on the complete set of Montgomery curves was presented, and efficient 4-way SIMD AVX2 and 8-way

AVX-512 implementations of this ladder were produced.

- Schoof's algorithm and SEA algorithm were explained in detail. Numerical examples for these algorithms were provided.
- Methods and optimization techniques that can increase the efficiency of the SEA algorithm have been studied and added to the implementation.
- Magma implementations of Schoof's algorithm and SEA algorithm were presented.
- An early abort method was developed for detecting (non-)isogenous curves, and a Magma implementation for this method was presented.

## 1.3. LITERATURE REVIEW

The literature review of Chapter 3 is given in Section 3.1 not to disrupt the self-contained structure of the chapter. This section only provides the literature review on elliptic curve point counting methods.

Point counting operation on elliptic curves has always been exhausting. Point counting was done using a naive approach until efficient procedures were discovered. Sundriyal (2008) explained the naive algorithm in detail. This algorithm stems from the Lang-Trotter method, expressed by Washington (2008). Various number theorists and computer scientists struggled for many years to find a way to count points on elliptic curves effectively. Harris (2005) stated that the first seemingly efficient way to deal with the problem of point counting was using the BSGS algorithm (Shanks, 1971). Zelenova (2011) gave descriptive information about this algorithm and discussed the improvements the algorithm provided. As field prime started to grow to enormous sizes with cryptographic advances, this algorithm became inefficient due to its running-time complexity $O(p^{1/4+o(1)})$. Although it is impractical to count points on elliptic curves of genus one, Matsuo, Chao, and Tsujii (2002) used the BSGS algorithm to count numbers on hyperelliptic curves efficiently. Also, there is an improved version of the BSGS algorithm in the context of point counting, called the Shank-Mestre algorithm (Cohen, 1993). In this method, Mestre decreased the overall computational effort needed to find the trace $a_p$ by applying the BSGS algorithm to the quadratic twist of the elliptic curve $E$ to compute $\#E = p + 1 - a_p$. The Shank-Mestre algorithm utilizes Hasse's Bound (Hasse, 1936) to determine the possible number of points on an elliptic curve and Lagrange's Theorem (Roth, 2001), demonstrating that the order of any point on an elliptic curve must divide the total number of points on the same curve. This method was also implemented and heavily used in the PARI computer algebra system (Batut, Belabas, Bernardi, Cohen, & Olivier, 1990). Additionally, there are methods preferred

in terms of space efficiency, such as Pollard's Rho algorithm and Pollard's Kangaroo algorithm, proposed by Pollard (1978). However, all of these methods had exponential running time complexity.

The popularity of the BSGS algorithm and its variants started to decline with the emergence of Schoof's algorithm (Schoof, 1985). Schoof's groundbreaking work on counting points on elliptic curves paved the way for various developments in the field. The algorithm was the first elliptic curve point counting algorithm that ran in polynomial time. It achieved this by computing trace value $t$ modulo different small prime numbers and building $t$ with the help of Hasse's bound and Chinese Remainder Theorem (CRT) at the end of the algorithm. The running-time complexity of Schoof's algorithm is $O(log^8 p)$. Menezes (1993) proposed Schoof's algorithm implementation for counting points on an elliptic curve defined over a field of characteristic two. The $\mathbb{F}_{2^m}$-points are chosen because of their suitability for efficient computation. As a side note, McGee (2006) implemented Schoof's algorithm using Mathematica (Inc., n.d.), and Hsieh and Yang (2010) published a study about designing a graphical interface to implement Schoof's algorithm on Windows platforms. Detailed explanations of Schoof's algorithm and its extensions are given by (Blake, Seroussi, & Smart, 1999). Before heading to these extensions, the other methods used in elliptic and hyperelliptic curve point counting will be mentioned.

Lercier, Lubicz, and Vercauteren (2005) gave overviews of both $p$-adic and $\ell$-adic methods used in elliptic curve point counting. The $p$-adic methods include approaches that benefit from zeta functions and canonical lifts to count the number of points on an elliptic curve. At first, Kato and Lubkin (1982) found the relation between $p$-adic cohomology theory and zeta matrices of elliptic curves. They devised a $p$-adic algorithm that finds the number of points on an elliptic curve in polynomial time. Then, Satoh (2000) showed that canonical lifts could be used to create efficient implementations of point counting, more efficient than $\ell$-adic methods when $p$ is small enough. Also, Satoh, Skjernaa, and Taguchi (2003) discussed the fast computation methods of canonical lifts of elliptic curves and their relation with point counting. Satoh's work was crucial for the progress of $p$-adic methods since several improvements started to be proposed by different people after the emergence of his fast algorithm. In the same year as the appearance of Satoh's work, Fouquet, Gaudry, and Harley (2000) presented an extension of his algorithm that optimises the computations in small characteristics, especially in characteristics two and three. Then, an improvement over Satoh's algorithm in terms of space efficiency was published (Vercauteren, Preneel, & Vandewalle, 2001). In 2002, Satoh surveyed advancements in $p$-adic algorithms (Satoh, 2002). Furthermore, Skjernaa (2003) provided an efficient variant of Satoh's algorithm specific to characteristic two. Also, Madsen (2005) applied Satoh's $p$-adic

point counting methods to the elliptic curves in Legendre form.

Apart from Satoh's algorithm and its variants, Arithmetic-Geometric Mean (AGM) based algorithms fall under the $p$-adic methods category. Mestre (2000) used this method to count points on elliptic curves. Next, he provided an algorithm to count points on hyperelliptic curves in a separate work (Mestre, 2002). Kedlaya's algorithm (Kedlaya, 2001) is a point counting method also used for counting points on hyperelliptic curves. This algorithm works with the help of zeta functions and Monsky–Washnitzer cohomology group (Monsky & Washnitzer, 1968; Monsky, 1968). Denef and Vercauteren (2002) extended Kedlaya's algorithm to Artin-Schreier curves in characteristic two. They also developed an algorithm that expanded Kedlaya's algorithm on general hyperelliptic curves in characteristic two (Denef & Vercauteren, 2006). In addition, Gaudry and Gürel (2003) explained how the point counting operation with Kedlaya's algorithm works on hyperelliptic curves of medium characteristics. Also, the same year, Lercier and Lubicz (2003) provided an implementation-friendly algorithm for counting points on elliptic curves over finite fields of small characteristics while running in quasi-quadratic time. Study of Lauder (2005) reviewed rigid cohomology-based algorithms that compute the zeta function of an algebraic variety over a finite field. Then, Carls and Lubicz (2009) designed an algorithm that computes the number of points on the Jacobian variety of a generic ordinary hyperelliptic curve by combining the canonical lifting of Satoh and AGM-based algorithm of Mestre with having a time complexity of $O(log(p)^{2+o(1)})$.

Schoof's and SEA algorithms fall into the $\ell$-adic methods category. SEA algorithm arose from the works of Atkin and Elkies. These improvements revealed the real potential of Schoof's algorithm. Atkin (1988, 1991) put forward an algorithm to detect the order of the Frobenius endomorphism in the projective general linear group acting on the $\ell$-th torsion group $E[\ell]$ of an elliptic curve $E/\mathbb{F}_p$. Elkies (1991) found a method to decrease the degree of the $\ell$-th division polynomial used in Schoof's algorithm. Instead of using the $\ell$-th division polynomial of degree $(\ell^2-1)/2$ in the computations, Elkies used a kernel polynomial of degree $(\ell-1)/2$, and this led to a significant speedup in the algorithm. Schoof (1995) combined all these advancements and explained the complete SEA algorithm in detail. The running-time complexity of the SEA algorithm is $\tilde{O}(log^4 p)$, which indicates a significant improvement in efficiency. This work of Schoof is of great importance since it also includes several enhancements to existing algorithms used in point counting. Schoof (1995) also provided an extension of Mestre's theorem on the bounds of the order of the group of points. The original theorem was valid for $p > 457$, yet Schoof expanded it to $p > 229$, where $p$ is the prime of a prime field $\mathbb{F}_p$. Cremona and Sutherland (2010) extended this bound to $p > 49$ for all finite fields and $p > 29$ for all prime fields. Additionally, Schoof (1995) explained a method

of elliptic curve point counting with the knowledge of the endomorphism ring of the elliptic curve $E$ by using Cornacchia's algorithm (Cornacchia, 1908).

After the improvements of Atkin and Elkies on Schoof's algorithm, many studies on the SEA algorithm began to appear. Couveignes and Morain (1994) investigated attributes of the isogeny cycles. They showed how to efficiently use the powers of good primes by using isogenies between curves over the ground field. Lercier and Morain (1995) considered different strategies for point counting and discussed the performances while presenting an efficient point counting implementation for an elliptic curve over a field $GF(2^n)$ for $n \leq 500$. Then, Lercier and Morain (1996) gave the first successful implementation of Couveignes's algorithm. Elkies (1997) discussed the computational issues dependent on elliptic and modular curves over finite fields. Dewaghe (1998) described different methods to apply in the SEA algorithm when any small prime $\ell$ is considered "good" or "bad", according to being an Elkies or Atkin prime, respectively. Izu, Kogure, Noro, and Yokoyama (1998) described how to efficiently implement Schoof's algorithm by integrating Atkin and Elkies' improvements, the Match and Sort algorithm of Atkin and the isogeny cycles method. Optimization of the SEA algorithm for the case of characteristic two was presented by Vercauteren (2000). Next, Joux and Lercier (2001) suggested an alternative method to the Match and Sort algorithm called the Chinese and Match algorithm, which uses less memory space while having similar time complexity. Joux and Lercier (2006) also gave a point counting algorithm specific for elliptic curves of medium characteristics.

The eigenvalue computation phase is one of the most crucial parts of the SEA algorithm. Maurer and Müiler (2001) discussed the eigenvalue search algorithms suitable for Elkies' part of the procedure. Moreover, Gaudry and Morain (2006) provided a few improvements for the eigenvalue computation phase. These improvements advanced the SEA algorithm both theoretically and practically. One year later, Mihailescu, Morain, and Schost (2007) found a method to compute the eigenvalue using Abelian lifts. The Abelian properties of division polynomials allowed them to build fast eigenvalue computation algorithms. Bostan, Morain, Salvy, and Schost (2008) discussed isogeny computation algorithms for Elkies' part of the SEA algorithm. They provided a fast isogeny computation algorithm that does not need the sum of roots to operate.

Modular polynomials are important mathematical structures employed in the SEA algorithm's kernel polynomial construction. Numerous theorists also heavily work on these structures to get the maximum benefit. Broker, Lauter, and Sutherland (2010) propounded a new algorithm for computing classical modular polynomials using isogeny volcanoes. Moreover, Sutherland (2013) proposed new procedures to evaluate modular polynomials. He also set a point-counting record with this work by utilizing his proposed methods. These studies helped Sutherland break the new point counting record in the

area of elliptic curves of genus one. Sutherland (2013) managed to compute the number of points on an elliptic curve defined over a prime field, with the prime having more than 5000 digits.

The $\ell$-adic point counting methods also apply when the genus of the curve is greater than one. Gaudry and Harley (2000) provided several algorithms to get the number of points on hyperelliptic curves and their Jacobians defined over finite fields. Lercier et al. (2005) comprehensively described methods for finding the cardinality of hyperelliptic curves. Lercier and Lubicz (2006) provided an algorithm for counting points on Jacobians of ordinary hyperelliptic curves in quasi-quadratic time. In 2007, Hubrechts addressed this topic and gave a polynomial time point counting algorithm suitable for the families of hyperelliptic curves (Hubrechts, 2007a) and specifically for characteristic two (Hubrechts, 2007b). Then, Gaudry, Kohel, and Smith (2011) proposed an efficient point counting algorithm suitable for genus two curves which utilized real multiplication endomorphism in the computations. Also, Gaudry and Schost (2012) improved the point counting method for Jacobians of genus two curves over a large prime field. They detected a cryptographically secure curve with a state-of-the-art security level of approximately $2^{128}$. A comprehensive overview of hyperelliptic curve point counting can be found in (Doliskani, 2011) and (Abelard, 2018).

There are many implementations of the SEA algorithm on different platforms. One of the most commonly used implementations is the one in the PARI-GP software (Batut et al., 1990). Also, it exists in SageMath Mathematics Software System (T. S. Developers et al., 2020), Magma Computer Algebra System (Bosma et al., 1997) and the MIRACL Core Cryptographic Library (T. M. Developers, 2018). In addition, there are also individual implementations arising from various articles and theses. For instance, Hsieh, Yang, and Ahn (2008) implemented the SEA algorithm suitable for working on Windows platforms. Then, Kok (2013) implemented the SEA algorithm with the NTL library (Shoup & Developers, 1990). Morain, Scribot, and Smith (2016) suggested a practically faster point counting algorithm than SEA while providing a C++ implementation of their algorithm using the NTL library. Furthermore, Stankovic (2017) implemented the SEA algorithm in a Python-Sage environment. Besides these examples, this thesis also presents an open-source Magma implementation of the SEA algorithm. A clear explanation of the SEA algorithm can be found in (Blake et al., 1999), (Csirik, 2000) and (Galin, 2007).

## 1.4. CONTRIBUTIONS

The introduction of a fast 4-way vectorized Montgomery ladder is the first main contribution of this thesis. The 4-way AVX2 implementation got competitive results among

the other fast vectorized implementations in the field. The 8-way AVX-512 implementation set the new speed record in the area with an improvement of approximately nine per cent.

The second main contribution of this thesis is the presented early abort method for detecting (non-)isogenous curves. The detection of cases when given two elliptic curves are non-isogenous can be done very quickly with this method. This method was implemented using Magma. This thesis also presents publicly available Magma implementations of Schoof's and SEA algorithms. SEA algorithm was implemented using both classical modular polynomials and Atkin modular polynomials, and a fast kernel polynomial computation technique is employed in the version using Atkin modular polynomials. In addition, the literature review section summarises various works on elliptic and hyperelliptic curve point counting. This thesis can also be a helpful resource for non-experts interested in elliptic curve point counting.

## 1.5. ORGANIZATION

The rest of the thesis is organized as follows. Several necessary preliminaries are given in Chapter 2 to fully understand the inner workings of elliptic curve scalar multiplication, Montgomery ladder, Schoof's algorithm and SEA algorithm. Chapter 3 provides all the necessary information about the fast 4-way vectorized Montgomery ladder for the complete set of Montgomery curves, including the implementation details. Chapter 4 introduces a new early abort method to detect (non-)isogenous curves. Before heading into the process, detailed explanations of Schoof's and SEA algorithms are given. For Schoof's algorithm, trace computations modulo two and consecutive odd primes are expressed, and a numerical example is given. For the SEA algorithm, the improvements of Atkin and Elkies are fully clarified, Match and Sort algorithm is explained, and a toy example is given. After presenting the SEA algorithm in detail, the method for detecting (non-)isogenous curves with implementation details is provided. In the last chapter, the work done in this thesis is summarized, and possible future work related to this thesis is discussed.

# CHAPTER 2
# PRELIMINARIES

To better understand the algorithms described in Chapter 3 and Chapter 4, gathering information about some structures and their properties would be helpful. This preliminary chapter gives several necessary definitions and properties of elliptic curves first. Then, the group law defined on a group of points on an elliptic curve is mentioned. After that, isogenies between elliptic curves and the Frobenius endomorphism map are described. Then, Hasse's bound related to the number of points on an elliptic curve is briefly explained. Finally, division polynomials, which are frequently used in Schoof's algorithm and SEA algorithm, and modular polynomials that play a significant role in obtaining the improvements in the SEA algorithm are described.

## 2.1. ELLIPTIC CURVES

Nowadays, elliptic curves are used extensively in many areas of cryptography, such as key exchanges, digital signatures and pseudorandom number generators. Although elliptic curves have been known for many years throughout history, they have recently been used in cryptographic applications. They were first seen in Diophantus' Arithmetica in the second century AD. Number theorists such as Cauchy, Levi, Lucas, Sylvester and Poincaré worked on them until the 1920s. They were used in the Mordell–Weil Theorem (Mordell, 1922), one of the four theorems that constructed the Diophantine Geometry. Miller (1985) and Koblitz (1987) independently proposed the relation of elliptic curves with cryptography under a notion called ECC. Their first application in ECC happened with the Lenstra elliptic-curve factorization method (ECM) (Lenstra, 1987). ECC variants of the existing cryptographic schemes have been proposed (e.g., Elliptic Curve Diffie-Hellman key exchange, Elliptic Curve Digital Signature Algorithm), which have widened the usability of elliptic curves significantly. Elliptic Curve Diffie-Hellman key exchange (ECDH) is one of the significant contributions of elliptic curves to the world of digital security. It is based on a complicated problem and is considered a reliable key agreement protocol because of this problem. The problem that ensures the security of ECDH is named ECDLP. The safety of ECDLP is based on the difficulty of reaching a number $k$ that $R = [k]P$ while knowing two points $P$ and $R$ on an elliptic curve $E$.

ECC follows a public-key cryptographic (or asymmetric cryptographic) approach by employing public keys in both parties, as in Rivest-Shamir-Adleman (RSA) cryptosystem. The main advantage of ECC over RSA is that it can provide equivalent security, with the key sizes being much smaller. For example, according to (Mahto & Yadav, 2017), an equal cryptographic strength can be ensured by using 256-bit keys in ECC, whilst in the RSA cryptosystem, 3072-bit keys are needed. The smaller keys mean more memory space efficiency, enabling the usage of elliptic curve-related methods on devices with small memory sizes and low CPU consumption due to faster arithmetic operations. Also, the key generation phase in ECC takes less time than the one in the RSA. Considering these advantages, working with elliptic curves would be a sensible choice.

Elliptic curves are broad concepts to go into detail. Providing some background information about these curves would be handy to cope with the complex nature of the elliptic curve scalar multiplication methods and point counting algorithms.

**Definition 2.1.1.** An elliptic curve is a non-singular algebraic curve of genus one denoted with $E$ and defined with the affine Weierstrass equation,

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.1}$$

having $\mathcal{O} \in E$ as the base point (or called the point at infinity) and defined over a field $K$. The coefficients $a_1, a_2, a_3, a_4$ and $a_6 \in K$ and the characteristic of $K$ is represented with $p = char(K)$, multiplicative identity with $1_K$, additional identity with $0_K$ satisfying $\underbrace{1_K + 1_K + \ldots + 1_K}_{\text{p times}} = 0_K$.

The field $K$ can be chosen as, for example, $\mathbb{Q}, \mathbb{R}$ or $\mathbb{C}$. Usually, $E$ is defined over a finite field and in such cases, the number of points on $E$ becomes countable. In the cryptographic setting, the chosen finite field is bounded by a large prime $p$, and the field is denoted with $\mathbb{F}_p$. The representation of the curve can change according to the characteristic of the underlying field $K$. The isomorphism [1] of elliptic curves can help simplify the Weierstrass form. This simplification can be achieved with the help of the transformation map in the following definition.

[1] An isomorphism is an invertible bijective map between two mathematical structures which maintains the structural properties.

**Definition 2.1.2.** Let $E_1$ and $E_2$ be two elliptic curves over a field $K$ in the Weierstrass form as

$$\begin{aligned} E_1 &: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \\ E_2 &: y^2 + a_1'xy + a_3'y = x^3 + a_2'x^2 + a_4'x + a_6'. \end{aligned} \tag{2.2}$$

An isomorphism between $E_1$ and $E_2$ exists if and only if there exist some variables $u, r, s, t \in K, u \neq 0$ which satisfy the map

$$(x, y) \mapsto (u^2x + r, u^3y + u^2sx + t) \tag{2.3}$$

that causes a transformation from the equation of $E_1$ to $E_2$. These types of transformations are called *admissible changes of variables*.

For an elliptic curve $E/\mathbb{F}_p$ in the Weierstrass form, when $char(\mathbb{F}_p) = 2$ and the curve is supersingular, meaning that the number of points on $E$ is equal to $p + 1$, the transformation map in Definition 2.1.2 turns the equation of $E/\mathbb{F}_p$ to

$$E/\mathbb{F}_p : y^2 + a_3y = x^3 + a_4x + a_6 \tag{2.4}$$

by the use of the map

$$(x, y) \mapsto (x + a_2, y).$$

In the non-supersingular case, the equation appears as

$$E/\mathbb{F}_p : y^2 + xy = x^3 + a_2x^2 + a_6 \tag{2.5}$$

by the use of the map

$$(x, y) \mapsto (a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2a_4 + a_3^2}{a_1^3}).$$

If $char(\mathbb{F}_p) = 3$ and the curve is supersingular, then the equation of $E/\mathbb{F}_p$ appears as

$$E/\mathbb{F}_p : y^2 = x^3 + a_4x + a_6 \tag{2.6}$$

by the use of the map

$$(x, y) \mapsto (x, y + a_1x + a_3).$$

In the non-supersingular case, it turns into

$$E/\mathbb{F}_p : y^2 = x^3 + a_4x^2 + a_6 \tag{2.7}$$

by the use of the map

$$(x, y) \mapsto (x + \frac{a_4 - a_1 a_3}{a_1^2 + a_2}, y + a_1 x + a_1 \frac{a_4 - a_1 a_3}{a_1^2 + a_2} + a_3).$$

At last, if $char(\mathbb{F}_p) \neq 2, 3$, the equation turns into

$$E/\mathbb{F}_p : y^2 = x^3 + a_4 x + a_6 \tag{2.8}$$

by the use of the map

$$(x, y) \mapsto (\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24}).$$

Equation (2.8) is called the short Weierstrass equation. Defining elliptic curves in the short Weierstrass form simplifies group law and the expressions related to the curves. For example, the discriminant of an elliptic curve in the Weierstrass form is defined as

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6 \tag{2.9}$$

where

$$d_2 = a_1^2 + 4a_2, \qquad d_4 = 2a_4 + a_1 a_3,$$
$$d_6 = a_3^2 + 4a_6, \qquad d_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2,$$

while the discriminant of a short Weierstrass curve is defined in a much shorter way as

$$\Delta = -16(4a_4^3 + 27a_6^2). \tag{2.10}$$

This situation also applies the expressions of the $j$-invariants of elliptic curves. One property to note here is that the $j$-invariant of two elliptic curves are equal if and only if these are isomorphic. The admissible change of variables becomes applicable only when the curves have the same $j$-invariant. The $j$-invariant of an elliptic curve $E$ in the Weierstrass form is defined as

$$j(E) = (d_2^2 - 24d_4)^3/\Delta. \tag{2.11}$$

If $E$ is defined in short Weierstrass form, then the expression shortens and takes the form of

$$j(E) = 1728a_4^3/4\Delta. \tag{2.12}$$

More information about the $j$-invariants is given in Section 2.7.

There are other elliptic curves, such as Montgomery, Edwards, Twisted Edwards, and

Hessian. They have different representations from the short Weierstrass form. Also, the arithmetic on them is done with different formulas. Information about Montgomery curves is given in Section 3.1. Schoof's and SEA algorithms employ elliptic curves in short Weierstrass form. Therefore, examining the group law on these curves will be helpful.

## 2.2. GROUP LAW

Let $E : y^2 = x^3 + a_4x + a_6$ be an elliptic curve in short Weierstrass form over a finite field $\mathbb{F}_p$. The set of points on $E$, including the point at infinity, constitutes a group with a binary group operation (also called the composition law on $E$) represented by $\oplus$. Every group element can be obtained thanks to the binary operation and geometric construction of points. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on $E(\mathbb{F}_p)$. By simply drawing a line which connects these points, another point $R = (x_3, -y_3) = -(P \oplus Q)$ is found from the intersection of the curve. Projecting point $R$ according to the $x$-axis allows obtaining point $R'$ that $R' = (x_3, y_3) = P \oplus Q$. This operation is named *point addition* on elliptic curves, and the visual demonstration of the operation is given in Figure 2.1.

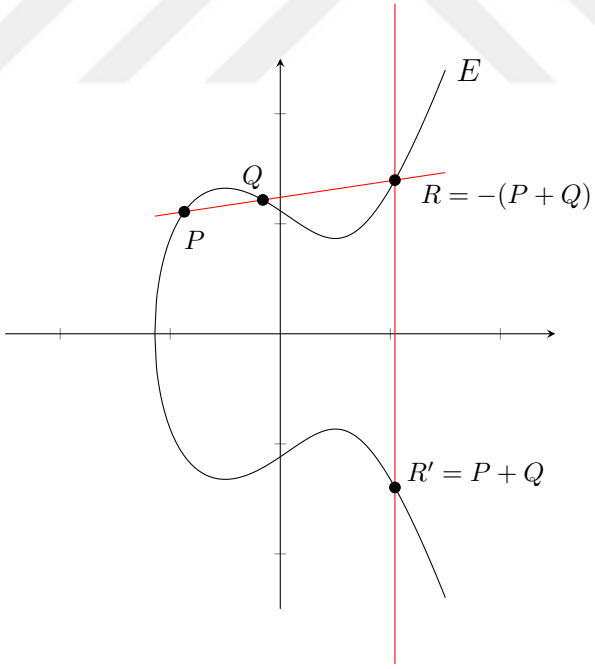


**Figure 2.1.** Point Addition on Elliptic Curves

Adding a point $P = (x_1, y_1)$ to itself can also be constructed geometrically by drawing a tangent line at $P$ and again taking the projection at the $x$-axis. This operation is named *point doubling* on elliptic curves, and the visual demonstration of the procedure is given in Figure 2.2.

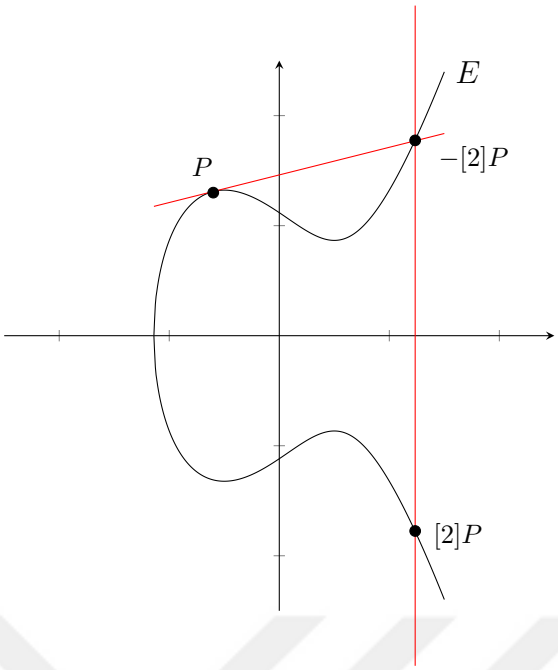


**Figure 2.2.** Point Doubling on Elliptic Curves

Here, the point at infinity $\mathcal{O}$ can be imagined as a point parallel to the $y$-axis and located very far away from the origin. It is the identity element of the group that for a point $P = (x_1, y_1) \in E(\mathbb{F}_p), P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$ holds. Also, the addition of $P$ and its negative $-P = (x_1, -y_1)$ results in $P \oplus -P = \mathcal{O}$ and is visualized by a line that connects $P$ with $-P$, meanwhile intersecting with $\mathcal{O}$.

The explicit formulas to detect the coordinates of the resulting points in addition and doubling are derived by first finding the slope of the intersecting line as

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } x_1 \neq x_2 \text{ i.e., } P \neq \pm Q, \\ \dfrac{3x_1^2 + a_4}{2y_1} & \text{if } x_1 = x_2, y_1 = y_2 \text{ i.e., } P = Q. \end{cases} \tag{2.13}$$

Then, the coordinates of the point $R = (x_3, y_3)$ are found as

$$x_3 = \begin{cases} \lambda^2 - x_1 - x_2 & \text{if } P \neq \pm Q, \\ \lambda^2 - 2x_1 & \text{if } P = Q, \end{cases} \tag{2.14}$$

and

$$y_3 = \lambda(x_1 - x_3) - y_1, \tag{2.15}$$

whether $P$ is equal to $Q$ or not. Detailed information about the group law of elliptic curves and proof of the composition law can be found in (Silverman, 2009, pp. 51-58).

## 2.3. ISOGENIES

Isogenies of elliptic curves have become a prominent topic recently, especially with the increasing importance of post-quantum cryptography. The Supersingular Isogeny Diffie Hellman key exchange is an extensively studied scheme. It is built with supersingular isogenies and the isogeny graphs created with these isogenies. Isogenies are also used in the Elkies part of the SEA algorithm. Thus, discussing isogenies and their properties in this thesis will be beneficial. Isogenies can be established among various algebraic varieties. Here, the isogeny of elliptic curves will be discussed.

**Definition 2.3.1.** Let $E_1/K$ and $E_2/K$ be two elliptic curves where $K$ is a field. An isogeny is a non-constant surjective morphism defined between $E_1$ and $E_2$, mapping the identity $\mathcal{O}_1$ of $E_1$ to the identity $\mathcal{O}_2$ of $E_2$ as

$$\begin{aligned} &\phi : E_1 \mapsto E_2, \\ &\phi(\mathcal{O}_1) = \mathcal{O}_2. \end{aligned} \tag{2.16}$$

The isogeny $\phi$ associates the elliptic curves $E_1$ and $E_2$ with each other. Therefore $E_1$ and $E_2$ can be called *isogenous curves*. Theorem 2.3.1 gives a critical property related to this case.

**Theorem 2.3.1.** *Let $E_1$ and $E_2$ be two elliptic curves that are isogenous over the field $K$ and $\phi : E_1 \mapsto E_2$ is an isogeny which maps the identity of $E_1$ to the identity of $E_2$. Then, $\phi$ is a group homomorphism from $E_1(K)$ to $E_2(K)$.*

*Proof.* See (Silverman, 1986, pp.75-76). □

An isogeny can either be separable, inseparable or purely inseparable. If $\phi$ is identified as

$$\phi(x, y) = \left( \frac{u(x)}{v(x)}, \frac{s(x)}{t(x)} y \right), \tag{2.17}$$

where $u, v$ and $s, t$ are pairwise coprime polynomials in $K$, then $\phi$ is a separable isogeny if $\left(\frac{u}{v}\right)' \neq 0$. The isogenies involved in cryptography are mostly separable ones. The degree of a separable isogeny is determined by the number of elements in its kernel (Silverman, 1986, Theorem III.4.10). The following definition helps understand this statement better.

**Definition 2.3.2.** The kernel of an isogeny is a set of points in $E_1(\bar{K})$ that every element in $E_1(\bar{K})$ is mapped to the identity element $\mathcal{O}_2$ of $E_2$ with the isogeny $\phi$, i.e.,

$$\ker(\phi) = \{P \in E_1(\bar{K}) : \phi(P) = \mathcal{O}_2\}. \tag{2.18}$$

If every element $P$ of $E_1(\bar{K})$ is mapped to $\mathcal{O}_2$ with $\phi$, then $\phi$ is a constant map named *zero isogeny*. As the name suggests, its degree is zero. If the degree is one, meaning the kernel has only one element, $\ker(\phi) = \{\mathcal{O}\}$, the mapping is an isomorphism. Isomorphisms are invertible, and an isomorphism can be merged with the inverse of it to build an identity map. Isogenies do not have inverses. However, they have *dual* isogenies. They are combined with the isogenies to create multiplication-by-$d$ maps where $d$ is the degree of the isogeny used. The definition given below solidifies this statement.

**Definition 2.3.3.** Let $E_1$ and $E_2$ be isogenous elliptic curves defined as in Theorem 2.3.1, and $\phi$ be an isogeny that $\phi : E_1 \mapsto E_2$. Then,

$$\hat{\phi} : E_2 \mapsto E_1 \tag{2.19}$$

is a unique map called the dual isogeny. The composition

$$\hat{\phi} \circ \phi = [d]_{E_1} \tag{2.20}$$

gives the multiplication-by-$d$ map on $E_1$. The opposite case,

$$\phi \circ \hat{\phi} = [d]_{E_2}, \tag{2.21}$$

gives the multiplication-by-$d$ map on $E_2$.

The kernel of a multiplication-by-$d$ map forms the *d-torsion group*. This property is very crucial for both the Schoof's and SEA algorithms, as the scalar multiplication of points on elliptic curves is frequently used in them. Definition 2.3.4 gives insight into the torsion groups and points.

**Definition 2.3.4.** Let $E$ be an elliptic curve defined over a field $K$. An $\ell$-torsion subgroup of $E$ is defined as

$$E[\ell] = \{P \in E(\bar{K}) : [\ell]P = \mathcal{O}\}, \tag{2.22}$$

where $\ell \geq 1$ and $\mathcal{O}$ is the identity of $E$. Also, every element in $E[\ell]$ is called an $\ell$-torsion point.

In addition, for points $P, Q \in E(\bar{K})$, every point $R \in E(K)$ can be written as $[\alpha]P + [\beta]Q$ where $\alpha, \beta \in \mathbb{Z}$. This statement can be generalized as

$$E[\ell] \cong \mathbb{Z}/\ell \times \mathbb{Z}/\ell. \tag{2.23}$$

The following theorem paves the way for the improvements of Atkin and Elkies over Schoof's algorithm.

**Theorem 2.3.2.** *Let $E$ be an elliptic curve defined over a finite field $\mathbb{F}_p$ and $\ell$ be a prime where $\gcd(\ell, p) = 1$. Then, the $\ell$-torsion subgroup $E[\ell] = \mathbb{Z}/\ell \times \mathbb{Z}/\ell$ has precisely $\ell + 1$ cyclic subgroups $C_i$ for $i = 0, \ldots, \ell$ and $|C_i| = \ell$.*

*Proof.* See (Galin, 2007, p. 19). □

## 2.4. FROBENIUS ENDOMORPHISM

Frobenius endomorphism is a special type of endomorphism frequently used in elliptic curve point counting. It is a map defined between groups with the characteristic $p$, where $p$ is a prime number, and it is generally associated with elliptic curves and their arithmetic. In this thesis, Frobenius endomorphism will always be related to elliptic curves over finite fields since the operations in the algorithms that will be explained are always done in finite fields. To be able to grasp the idea of what the Frobenius Endomorphism is, the homomorphism property between groups should be defined first.

**Definition 2.4.1.** A homomorphism is a function between two groups keeping each group's structure the same after the application of binary operation. Let $G$ and $H$ be the groups with the binary operations $*$ and $+$, respectively. Then $f$ is a homomorphism that $f : G \to H$ between $G$ and $H$ if

$$f(x * y) = f(x) + f(y), \tag{2.24}$$

for every pair $x, y \in G$.

In light of this, the endomorphism map for group $G$ can be defined as the following.

**Definition 2.4.2.** An endomorphism is a homomorphism of a group $G$ to itself. It preserves the originality of the group structure. Then, $g$ is an endomorphism that $g : G \to G$ for a group $G$ with the binary operation $*$ related to $G$ if

$$f(x * y) = f(x) * f(y), \tag{2.25}$$

for every pair $x, y \in G$.

After defining the endomorphism for groups, it is now suitable to define the Frobenius endomorphism.

**Definition 2.4.3.** (Frobenius Endomorphism) Consider an elliptic curve $E/\mathbb{F}_p$ and a point $P = (x, y)$, which lies on $\bar{\mathbb{F}}_p$, the algebraic closure of $\mathbb{F}_p$. The Frobenius endomorphism $\varphi \in \mathrm{End}_{\mathbb{F}_p}(\mathrm{E})$ (i.e., the endomorphism ring of $\mathbb{F}_p$-endomorphisms of elliptic curve $E$) and is defined as

$$\begin{aligned} \varphi : E/\bar{\mathbb{F}}_p &\to E/\bar{\mathbb{F}}_p, \\ \varphi(x, y) &= (x^p, y^p), \\ \varphi(\mathcal{O}) &= \mathcal{O}. \end{aligned} \tag{2.26}$$

With the Frobenius endomorphism, every element in $\bar{\mathbb{F}}_p$ is mapped to its $p$-th power, which is already present in $\bar{\mathbb{F}}_p$. The characteristic equation used for detecting the trace of Frobenius in Schoof's algorithm employs the Frobenius endomorphism. Such an equation can be built thanks to Hasse's Bound.

## 2.5. HASSE'S BOUND

Hasse (1936) showed that there is a bound that determines a range for the possible number of points on a non-singular elliptic curve $E/\mathbb{F}_p$. This bound restricts the number of points on $E/\mathbb{F}_p$ with the inequality

$$| \#E(\mathbb{F}_p) - p - 1 | \leq 2\sqrt{p} \tag{2.27}$$

where $\#E(\mathbb{F}_p)$ is the number of points on $E/\mathbb{F}_p$ and $p$ is the characteristic of the finite field $\mathbb{F}$.

It would be convenient to give proof of Hasse's theorem on elliptic curves after some necessary background information.

Let $\alpha$ and $\beta$ be endomorphisms defined from $E/\mathbb{F}_p$ to itself. These endomorphisms can be considered as separable isogenies from $E/\mathbb{F}_p$ to $E/\mathbb{F}_p$. Here, the orders of the kernels of isogenies are equal to the degrees of isogenies. The degree of an isogeny can also be determined by looking at the highest degree term of the irreducible polynomial if and only if the isogeny is defined with an irreducible polynomial. As stated by Tolkov (2009), the multiplicative property of the degree of polynomials can be used for the degrees of endomorphisms as

$$\deg(\alpha \circ \beta) = \deg(\alpha)\deg(\beta), \tag{2.28}$$

and since these endomorphisms form a ring of endomorphisms of $E$, $\mathrm{End}_{\mathbb{F}_p}(\mathrm{E})$, they

also have the addition and composition properties for a point $P$ on $E/\mathbb{F}_p$ such as

$$(\alpha + \beta)(P) = \alpha(P) + \beta(P), (\alpha \circ \beta)(P) = \alpha(\beta(P)). \tag{2.29}$$

Then, the following inequality holds for the endomorphisms $\alpha$ and $\beta$.

$$| \deg(\alpha - \beta) - \deg(\alpha) - \deg(\beta) | \leq 2\sqrt{\deg(\alpha)\deg(\beta)}. \tag{2.30}$$

The proof of the above statement can be found in (Silverman, 1986). Keeping Equation (2.30) in mind will help us understand the proof of Hasse's theorem of elliptic curves. The proof declared here is taken from (Tolkov, 2009).

*Proof.* Let $\alpha$ be the Frobenius Endomorphism of the elliptic curve $E$ over $\mathbb{F}_p$. The degree of $\alpha$ is $p$ since it sends a point $P = (x, y)$ to $(x^p, y^p)$. With $x^p \equiv x \bmod p$, which comes from Fermat's Little Theorem, every point $P$ on $E/\mathbb{F}_p$ is fixed using Frobenius endomorphism $\alpha$ as $\alpha(P) = P$. Then, $\alpha(P) - P = 0$ and $(\alpha - 1)(P) = 0$. So, it can be deduced that $P$ is in the kernel of $\alpha - 1$. This kernel is isomorphic to $E/\mathbb{F}_p$, and the number of elements in this kernel equals the number of points on $E/\mathbb{F}_p$. Therefore,

$$\deg(\alpha - 1) = \#E/\mathbb{F}_p. \tag{2.31}$$

If Equation (2.30) and Equation (2.31) are associated,

$$| \deg(\alpha - 1) - \deg(\alpha) - \deg(1) | \leq 2\sqrt{\deg(\alpha)\deg(1)},$$

and $\deg(\alpha - 1) = \#E/\mathbb{F}_p, \deg(\alpha) = p, \deg(1) = 1$. In the end, by combining all,

$$| \#E(\mathbb{F}_p) - p - 1 | \leq 2\sqrt{p}.$$

□

## 2.6. DIVISION POLYNOMIALS

Division polynomials play a significant role in elliptic curve point counting algorithms such as Schoof's algorithm and SEA algorithm. Working modulo $\ell$-th division polynomial causes a remarkable speed up while performing necessary operations.

To describe the division polynomials, let $E$ be an elliptic curve defined over a field $\mathbb{F}_p$ with prime characteristic $p$ and given in short Weierstrass form as $E/\mathbb{F}_p : y^2 = x^3 + a_4x^2 + a_6$. For $\ell > 0$, there exist some polynomials $\psi_\ell, \phi_\ell, \omega_\ell \in \bar{\mathbb{F}}_p[x, y]$ and the existence of these polynomials enables getting the coordinates of the $\ell$-th multiple of

a point $P$, where $P = (x, y) \in E(\mathbb{F}_p)$ and $[\ell]P \neq \mathcal{O}$. The $\ell$-th multiple of $P$ can be expressed as

$$[\ell]P = \left(\frac{\phi_\ell(x,y)}{\psi_\ell(x,y)^2}, \frac{\omega_\ell(x,y)}{\psi_\ell(x,y)^3}\right). \tag{2.32}$$

Here, $\psi_\ell(x, y)$ is called the $\ell$-th division polynomial and $\phi_\ell$ and $\omega_\ell$ can be derived with the help of $\psi_\ell$ as

$$\phi_\ell = x\psi_\ell^2 - \psi_{\ell-1}\psi_{\ell+1}, \tag{2.33}$$

and

$$\omega_\ell = \frac{\psi_{\ell+2}\psi_{\ell-1}^2 - \psi_{\ell-2}\psi_{\ell+1}^2}{4y}, \tag{2.34}$$

where $\ell \geq 1$. Combining the expressions leads to the emergence of

$$[\ell]P = \left(x - \frac{\psi_{\ell-1}\psi_{\ell+1}}{\psi_\ell^2}, \frac{\psi_{\ell+2}\psi_{\ell-1}^2 - \psi_{\ell-2}\psi_{\ell+1}^2}{4y\psi_\ell^3}\right). \tag{2.35}$$

The proof of the derivation of this expression can be found in (Galin, 2007).

One important property of the division polynomials is that they can be generated recursively. Since $E$ is a short Weierstrass curve, the variables $a_1 = a_2 = a_3 = 0$. Therefore in the expressions, $x, y, a_4$ and $a_6$ will be free variables in $\mathbb{F}_p$. The recursive construction of the $\ell$-th division of polynomial of $E$ can be demonstrated as the following:

$$\begin{aligned}
\psi_0 &= 0, \\
\psi_1 &= 1, \\
\psi_2 &= 2y, \\
\psi_3 &= 3x^4 + 6a_4x^2 + 12a_6x - a_4^2, \\
\psi_4 &= 4y(x^6 + 5a_4x^4 + 20a_6x^3 - 5a_4^2x^2 - 4a_4a_6x - 8a_6^2 - a_4^3), \\
&\vdots \\
\psi_{2\ell} &= \left(\frac{\psi_\ell}{2y}\right)(\psi_{\ell+2}\psi_{\ell-1}^2 - \psi_{\ell-2}\psi_{\ell+1}^2), && \text{where } \ell \geq 3, \\
\psi_{2\ell+1} &= \psi_{\ell+2}\psi_\ell^3 - \psi_{\ell-1}\psi_{\ell+1}^3, && \text{where } \ell \geq 2.
\end{aligned}$$

According to Lang (1978, p. 37), the division polynomial formulas for both odd and even degrees can be represented in the $x$-coordinate only form. This conversion is done using induction techniques and the proper substitution of $y$-coordinates with $x$-coordinates using the equation $y^2 = x^3 + a_4x + a_6$. The formulas resulting from these processes are given below. Note that $f_\ell$ is equal to $\psi_\ell$ or $\psi_\ell/2y$ depending on

whether $\ell$ is odd or even, respectively.

$$
\begin{aligned}
f_0 &= 0, \\
f_1 &= 1, \\
f_2 &= 1, \\
f_3 &= 3x^4 + 6a_4x^2 + 12a_6x - a_4^2, \\
f_4 &= 2(x^6 + 5a_4x^4 + 20a_6x^3 - 5a_4^2x^2 - 4a_4a_6x - 8a_6^2 - a_4^3), \\
&\vdots \\
f_{2\ell} &= f_\ell(f_{\ell+2}f_{\ell-1}^2 - f_{\ell-2}f_{\ell+1}^2), \\
f_{2\ell+1} &= \begin{cases} f_{\ell+2}f_\ell^3 - 16(x^3 + a_4x + a_6)^2 f_{\ell-1}f_{\ell+1}^3, & \text{where } \ell \text{ is odd,} \\ 16(x^3 + a_4x + a_6)^2 f_{\ell+2}f_\ell^3 - f_{\ell-1}f_{\ell+1}^3, & \text{where } \ell \text{ is even.} \end{cases}
\end{aligned}
$$

Galin (2007) gave the construction of such polynomials with a theorem and the corresponding proof.

## 2.7. MODULAR POLYNOMIALS

In Schoof's Algorithm, division polynomials are utilized to increase computational efficiency in calculating $t_\ell$ values. Nonetheless, division polynomials' feasibility is only valid to a certain degree. Schoof's algorithm uses division polynomials of degree $(\ell^2 - 1)/2$ for an odd prime $\ell$. Due to the $\ell$ values getting large, the usage of division polynomials in trace computations will lose its practicality quickly as the workload increases. That is why Atkin and Elkies thought that when the discriminant of the characteristic polynomial is a square modulo a prime $\ell$ (will be called an Elkies Prime), a factor with the degree of $(\ell - 1)/2$ of the division polynomial $\psi_\ell$ can be found and used in the trace computation (Schoof, 1995). This factor is called kernel polynomial, and its usage decreases the computational workload significantly. Section 4.2.1 gives detailed information about selecting such primes. According to Schoof (1995), modular polynomials are needed to work with Elkies primes. Modular polynomials are also used in the classification part of the primes as Atkin or Elkies, so it would be helpful to be informed about them.

Modular polynomials are named according to their corresponding prime number $\ell$. The $\ell$-th classical modular polynomial $\Phi_\ell(x, y)$ is a symmetric bivariate polynomial having coefficients in $\mathbb{Z}$, each of whose degree is $\ell + 1$, where $(x, y) \in E$. A critical feature of this polynomial is that there exist $\ell + 1$ $\ell$-isogenous curves to $E$, with the j-invariant values being the roots of $\Phi_\ell(x, j) = 0$ (Blake et al., 1999, p. 51). The $\ell$-th

classical modular polynomial can be used in kernel computation by benefiting from this feature, as stated in Section 4.2.2.1.

The most well-known method for computing the classical modular polynomials includes elliptic curves, which are isogenous over a complex field $\mathbb{C}$. These elliptic curves can be uniquely associated with lattices (Silverman, 1986). The complex analytic side of elliptic curves must be investigated to understand the computation process clearly.

Let $\Lambda$ be a lattice defined as

$$\Lambda = \omega_1\mathbb{Z} + \omega_2\mathbb{Z},$$

where $\Lambda \subset \mathbb{C}$, $\omega_1, \omega_2 \in \mathbb{C}$ are the periods constructing the basis of the lattice $\Lambda$ and are $\mathbb{R}$-linearly independent. By using the lattice $\Lambda$, the Weierstrass elliptic function $\wp$ can be defined as

$$\wp(z, \Lambda) = \frac{1}{z^2} + \sum_{\omega \in \Lambda \setminus \{0\}} \frac{1}{(z-\omega)^2} - \frac{1}{\omega^2}, \tag{2.36}$$

which has a double pole on $\mathbb{C}/\Lambda$ and has the Laurent expansion $\wp(z) = z^{-2} + 0z^0 + g_2z^2 + g_3z^4 + O(z^6)$. The theorem below builds the connection between lattices and elliptic curves over complex fields.

**Theorem 2.7.1.** *Let $E/\mathbb{C}$ be an elliptic curve with the Weierstrass Equation (2.1) and $\mathbb{C}$ is a complex field. Between $E$ and a lattice, $\Lambda \subset \mathbb{C}$, there exists an isomorphism that,*

$$\begin{aligned} &\mathbb{C}/\Lambda \to E, \\ &\wp(z) \mapsto \begin{cases} (x_\Lambda, (\wp'(z) - a_1x_\Lambda - a_3)/2), & z \notin \Lambda \\ \mathcal{O}, & z \in \Lambda \end{cases} \end{aligned} \tag{2.37}$$

*with $\wp(z) = -\dfrac{a_1^2 + 4a_2}{12}$. This map is a bijection. Therefore, for any lattice $\Lambda$, there is a unique elliptic curve $E/\mathbb{C}$ satisfying the given map.*

*Proof.* See (Silverman, 1986, p. 161). □

Theorem 2.7.1 is valid when $E$ is in the Short Weierstrass form. In this case, the map can be defined as

$$\wp(z) \mapsto (\wp(z), \wp'(z)/2), z \notin \Lambda,$$

and the coefficients can be identified as

$$a_4 = -g_2/\sqrt[3]{4}, \qquad\qquad a_6 = -g_3,$$

where

$$g_2 = 60 \sum_{\omega \in \Lambda \setminus \{0\}} \frac{1}{\omega^4}, \qquad\qquad g_3 = 140 \sum_{\omega \in \Lambda \setminus \{0\}} \frac{1}{\omega^6}.$$

Note that any unique elliptic curve $E/\mathbb{C}$ that complies with Theorem 2.7.1 can also be stated as $E = \{(x, y) \in \mathbb{C}^2, y^2 = 4x^3 - g_2 x - g_3\}$ and the functions for coefficient computations can also be defined as

$$g_2(\tau) = 60 \sum_{(\alpha,\beta) \neq (0,0)} \frac{1}{(\alpha + \tau\beta)^4}, \qquad g_3(\tau) = 140 \sum_{(\alpha,\beta) \neq (0,0)} \frac{1}{(\alpha + \tau\beta)^6},$$

using a variable $\tau = \omega_1/\omega_2$. This variable is actually an element of an upper half-plane $\mathcal{H} = \{\alpha + i\beta \mid \beta > 0;\ \alpha, \beta \in \mathbb{R}\}$. For this reason, the j-invariant of the elliptic curve $E/\mathbb{C}$ is a function of $\tau$ in the upper half-plane. The j-invariant of the corresponding lattice $(1, \tau)$ is constructed as

$$j(\tau) = 1728 \frac{g_2^3}{g_2^3 - 27 g_3^2}. \tag{2.38}$$

Furthermore, let $SL_2(\mathbb{Z})$ be the modular group of $2 \times 2$ matrices with determinant 1, which acts on $\mathcal{H}$ via fractional linear transformations as

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \tau = (a\tau + b)/(c\tau + d). \tag{2.39}$$

Then, it appears that the lattices $(1, \tau)$ and $(1, \delta\tau)$ correspond to isomorphic elliptic curves for any $\tau \in \mathcal{H}$ and any $\delta \in SL_2(\mathbb{Z})$. So, $j(\tau) = j(\delta\tau)$ and with $\delta = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ or $\delta = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, $j(\tau) = j(-1/\tau)$ or $j(\tau) = j(1 + \tau)$ can be obtained, respectively. Thus, the j-invariant has a Fourier expansion in $q = e^{2\pi i \tau} \in \mathbb{C}$ (Schoof, 1995). Assuming $\Lambda$ is a lattice associated with an elliptic curve $E/\mathbb{C}$. Then j-invariant of this curve can be denoted with

$$j(q) = \frac{1}{q} + 744 + \sum_{n=1}^{\infty} c_n q^n, \tag{2.40}$$

where $c_n \in \mathbb{Z}$. It is clear that the choice of $\tau$ has no effect on the construction of $j(q)$, and $\tau$ can be picked from a fundamental domain $\mathcal{F} = \{\tau \in \mathcal{H} \ :\ |Re(\tau)| \leq 1/2,\ |\tau| \geq 1\}$.

Relevant proof can be found in (Silverman, 1986, p. 343).

In addition, the set of matrices is defined

$$\mathcal{C}(\ell) = \left\{ \begin{pmatrix} a & b \\ 0 & d \end{pmatrix} \; : \; a, b, d \in \mathbb{Z},\; ad = \ell,\; a > 0,\; 0 \leq b < d,\; \gcd(a, b, d) = 1 \right\} \tag{2.41}$$

and for all $\gamma = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix} \in \mathcal{C}(\ell)$, the action on $\mathcal{H}$ is

$$\gamma\tau = \frac{a\tau + b}{d}. \tag{2.42}$$

**Definition 2.7.1.** The $\ell$-th classical modular polynomial $\Phi_\ell(x, y)$ is a polynomial such that

$$\Phi_\ell(x, j(\tau)) = \prod_{\gamma \in \ell} (x - j(\gamma\tau)), \tag{2.43}$$

where $(x, y) \in \mathbb{C},\; |\mathcal{C}(\ell)| = \ell + 1$ and also it is monic in $x$ with degree $\ell + 1$.

The arithmetic properties of the $\ell$-th modular polynomial are given in the following theorem.

**Theorem 2.7.2.** *Let $n$ be a positive integer.*

1. $\Phi_n(x, y) \in \mathbb{Z}[x, y]$
2. $\Phi_n(x, y)$ *is irreducible as a polynomial in* $x$.
3. $\Phi_n(x, y) = \Phi_n(y, x)$.
4. *If $n$ is not a perfect square, then the degree of $\Phi_n(x, x) > 1$, whose leading coefficient is $\pm 1$.*
5. *If $n$ is prime, then* $\Phi_n(x, y) = (x^n - y)(x - y^n) \bmod n\mathbb{Z}[x, y]$.

*Proof.* See (Cox, 2013, p. 210). □

# CHAPTER 3

# FAST 4-WAY VECTORIZED LADDER FOR THE COMPLETE SET OF MONTGOMERY CURVES

**DECLARATION:**

- This chapter has been published in *Hüseyin Hışıl, Berkan Eğrice, and Mert Yassı, "Fast 4-way vectorized ladder for the complete set of Montgomery curves", International Journal of Information Security Science, vol. 11, no. 2, pp. 12-24, Jun. 2022*.
- A part of this work is supported by Yaşar University Research Project SRP-057.
- To keep the text self-contained, the relevant sections, such as the introduction, literature review, and contributions, are not given in Chapter 1 but this chapter.
- We thank Erdem Alkım, Sedat Akleylek, and members of the Cyber Security and Cryptology Laboratory, Ondokuz Mayis University, for providing us access to `OMU-i9`, a Skylake i9-7900X machine. We developed the AVX-512 implementation on `OMU-i9`. The related measurements were taken on `OMU-i9`. We thank the reviewers for their valuable comments.

This paper introduces the 4-way vectorization of the Montgomery ladder on any Montgomery form elliptic curve. Our algorithm takes $2\mathbf{M}^4 + 1\mathbf{S}^4$ ($\mathbf{M}^4$: A vector of four field multiplications, $\mathbf{S}^4$: A vector of four field squarings) per ladder step for variable-scalar variable-point multiplication. It is a theoretical improvement over the squared Kummer ladder, which takes $2\mathbf{M}^4 + 1\mathbf{S}^4 + 1\mathbf{d}^4$ per ladder step. This paper also introduces new formulas for doing arithmetic over $GF(2^{255} - 19)$. We provide two implementations of `curve25519` using our proposed algorithm. The first implementation uses the AVX2 instruction set and takes 98484 cycles, which is competitive with the current speed record of 95437 cycles by Nath and Sarkar. The second implementation uses the AVX-512 instruction set and takes 74368 cycles, which sets the new speed record over Faz-Hernández, López, and Dahab's implementation, which takes 81600 cycles.

## 3.1. INTRODUCTION

Elliptic curve cryptography was proposed by Miller (1985) and Koblitz (1987) in the late 80s. In the past three decades, elliptic curves have become one of the central objects in public key cryptography. The group law computations on elliptic curves are particularly interesting as they allow efficient computer arithmetic. In addition, hard instances of the discrete logarithm problem can be defined on elliptic curves over finite fields of reasonably small size. These two properties of elliptic curves make them perfect candidates for many cryptographic primitives, such as key exchange, key encapsulation mechanisms, and digital signatures. In these primitives, the bottleneck operation is the multiplication of a point on an elliptic curve with a scalar. This operation is called scalar multiplication. Optimizing scalar multiplication is one of the main challenges in elliptic curve cryptography.

An elliptic curve can be represented in several different forms. One of these forms was introduced by Peter L. Montgomery in his celebrated article (Montgomery, 1987). An elliptic curve in Montgomery form is written as in Equation (3.1)

$$By^2 = x^3 + Ax^2 + x \tag{3.1}$$

with constants $A$ and $B$ satisfying $B(A^2 - 4) \neq 0$. Let $P$ be a point on this curve. Let $x(P)$ be the $x$-coordinate of $P$. Let $k$ be a positive integer. The Montgomery ladder algorithm, also proposed by Montgomery (1987), computes $x(kP)$ by accessing a point doubling and a point addition operation per iteration of its main loop. In this setting, Montgomery provides doubling formulas to compute $x(2P)$ given $x(P)$ and differential addition formulas to compute $x(P + Q)$ given $x(P)$, $x(Q)$, and $x(P - Q)$. The ladder maintains the auxiliary value $x(P - Q)$ naturally. This regular structure of the Montgomery ladder made it a perfect candidate to be used in elliptic curve cryptography.

D. Bernstein (2006) proposed an elliptic curve Diffie-Hellman key exchange function, Curve25519, which uses the Montgomery ladder and a twist-secure Montgomery curve over the field $GF(2^{255} - 19)$. D. Bernstein (2006) also provided fast software that implements Curve25519, runs in constant-time, and can defend against timing attacks. Bernstein's design is later re-specified by the Internet Research Task Force in RFC 7748 memorandum.

The Montgomery ladder was also adapted to other elliptic curve forms. For example, (Brier & Joye, 2002) presented formulas for any elliptic curve written in short Weierstrass form $y^2 = x^3 + a_4x + a_6$, covering all elliptic curves over a field $k$ with char$(k) \neq 2, 3$. Analogous formulas over a field of characteristic 2 were given by

López and Dahab (1999). Additional alternative differential additions formulas can be found in (Castryck, Galbraith, & Farashahi, 2008), (D. J. Bernstein, Lange, & Rezaeian Farashahi, 2008), (Rezaeian Farashahi & Hosseini, 2016) and (Rezaeian Farashahi & Hosseini, 2017).

Building on an earlier work of Chudnovsky and Chudnovsky (1986), Gaudry introduced doubling and differential addition analogues on genus 2 Kummer surfaces (Gaudry, 2007). As follow-up work, Gaudry and Lubicz introduced genus 1 analogues of Kummer surfaces (Gaudry & Lubicz, 2009). Their study covers both odd and even characteristics. We refer to these Kummer lines as canonical Kummer lines in this work, following the language of (Renes & Smith, 2017). Explicit formulas for squared Kummer lines appeared in EFD[1] with credits to (Gaudry, 2007) and (Gaudry & Lubicz, 2009).

Emerging hardware trends in single-instruction multiple-data (SIMD) circuits led researchers to develop vectorized implementations of ladders. A SIMD implementation of the Gaudry-Schost squared Kummer surface (Gaudry & Schost, 2012) was introduced by D. Bernstein, Chuengsatiansup, Lange, and Schwabe (2014). Their implementation is currently the speed leader in the genus 2 setting. The genus 1 setting is actively in development. Chou (2015, Alg. 3.1) put forward a 2-way vectorized implementation of the Montgomery ladder using the inherent 2-way parallelism in the classic formulas. Chou's implementation uses the 2-way vectorized $32 \times 32 \rightarrow 64$-bit multipliers on Sandy Bridge and Ivy Bridge. A 4-way vectorized implementation of squared Kummer lines was presented by Karati and Sarkar (2017). Their implementation uses the 4-way vectorized $32 \times 32 \rightarrow 64$-bit multipliers on Haswell and Skylake. Karati and Sarkar report that their implementation offers competitive performance in Kummer line-based scalar multiplication for genus one curves over prime order fields using SIMD operations. Faz-Hernández, López, and Dahab provided a $2 \times 2$ way implementation of Montgomery ladder on Haswell and Skylake, see (Faz-Hernández, López, & Dahab, 2019). The arithmetic of the underlying field is 2-way vectorized in their implementation (hence the notation $2 \times 2$).

Putting the vectorization option of the underlying field aside (which is also an option for squared Kummer lines), the sequence of recent advances in ladder implementations may lead to the illusion that Montgomery curves are less vectorization-friendly than Kummer lines. In this work,

- we show that Montgomery curves are efficiently 4-way vectorizable. See Section 3.3.
- we propose a new 9-limb representation of field elements which has the potential

[1] `http://www.hyperelliptic.org/EFD/` (last accessed 2022-03-18)

to be faster than the widely applied 10-limb representation in implementations without using field-level vectorization. See Section 3.4.

- we provide timings for our $4 \times 1$ way vectorized implementation on AVX2. See Section 3.6.
- we provide timings for our $4 \times 2$ way vectorized implementation on AVX-512. See Section 3.6. This implementation sets the new speed record in variable-scalar variable-point multiplication over the field $GF(2^{255} - 19)$.

Source code related to this project is publicly available at the following URL.

`https://github.com/crypto-ninjaturtles/montgomery4x`

## 3.2. MONTGOMERY LADDER

This section provides preliminaries on the Montgomery ladder. We will skip detailed discussions on the group law, the pseudo-group structure, working solely on the $x$-line, point recovery etc. These are all very well understood and available in several texts in the literature, cf. (D. Bernstein & Lange, 2017, Chapter 4) and (Costello & Smith, 2018). Our approach will be more implementation-oriented. Therefore, the treatment in this section is far from comprehensive.

The abscissa $x(P)$ of a point $P$ is represented in homogenous projective space $\mathbb{P}$ in the form $(x(P) : 1)$. In this projective representation, $(X : Z) = (\lambda X : \lambda Z)$ for all non-zero $\lambda \in \mathbb{K}$. The point $(1 : 0)$ is the pseudo-identity element. From now on, we update the definition of $P$ and use the projective notation.

Given the points $(X_3 : Z_3)$, $(X_2 : Z_2)$, and $(X_1 : Z_1) = (X_3 : Z_3) - (X_2 : Z_2)$, we have $(X_5 : Z_5) = (X_3 : Z_3) + (X_2 : Z_2)$ and $(X_4 : Z_4) = 2(X_2 : Z_2)$. Montgomery (1987) provided the explicit formulas in (3.2) for $(X_5 : Z_5)$ and $(X_4, Z_4)$:

$$\begin{aligned} &\left(Z_1(X_2X_3 - Z_2Z_3)^2 : X_1(X_2Z_3 - Z_2X_3)^2\right), \\ &\left((X_2^2 - Z_2^2)^2 : 4X_2Z_2(X_2^2 + AX_2Z_2 + Z_2^2)\right), \end{aligned} \tag{3.2}$$

respectively.

These doubling and differential addition formulas are the building blocks of the Montgomery ladder. Before providing the ladder, we simplify our notation and define `DBLADD` and `SWAP` functions. The function `DBLADD` inputs three points, where the third is the difference between the first two and outputs the sum of the two initial points and the double of the second input point. The output is overwritten to $(X_3 : Z_3)$ and

$(X_2 : Z_2)$, respectively, as given in (3.3).

$$\texttt{DBLADD}\left((X_3 : Z_3), (X_2 : Z_2), (X_1 : Z_1)\right) \tag{3.3}$$

The function `SWAP` inputs two points and a single bit. If `swap` is 0, then the output is identical to the input. If `swap` is 1, then the output is the swapped input points. The Montgomery ladder is provided succinctly in Algorithm 1.

In cryptographic applications, the output of Algorithm 1 is typically normalized as $X_2/Z_2$ to obtain a unique representative of the output. In addition, $\ell$ is fixed to fix the number of iterations. Moreover, one can force $k$ to be multiple of a small power of 2 to surpass active attacks exploiting the existence of small subgroups. Cryptographic applications which are required to run in constant time must have each sub-operation run in constant time. We refer to `curve25519` specification for full detail (D. Bernstein, 2006).

**Algorithm 1:** Montgomery ladder

**Require:** $P = (X : Z) \neq (1 : 0)$ and $k = \sum_{i=0}^{\ell-1} k_i 2^i$ with $k_{\ell-1} = 1$, $k_i \in \{0, 1\}$.
**Ensure:** $kP$.

1: $(X_3 : Z_3) \leftarrow P, ;$
   $(X_2 : Z_2) \leftarrow (1 : 0)$
2: $(X_1 : Z_1) \leftarrow P, ;$
   $\texttt{prevbit} \leftarrow 0$
3: **for** $i = \ell - 1$ down to $0$ **do**
4: $\quad \texttt{swap} \leftarrow \texttt{prevbit} \oplus k[i]$
5: $\quad \texttt{prevbit} \leftarrow k[i]$
6: $\quad \texttt{SWAP}(\texttt{swap}, (X_3 : Z_3), (X_2 : Z_2))$
7: $\quad \texttt{DBLADD}((X_3 : Z_3), (X_2 : Z_2), (X_1 : Z_1))$
8: **end for**
9: $\texttt{SWAP}(k[0], (X_3 : Z_3), (X_2 : Z_2))$
10: **return** $(X_2 : Z_2)$

## 3.3. 4-WAY MONTGOMERY LADDER

Montgomery's formulas (3.2) lie at the heart of `curve25519`. Several implementations of `curve25519` are available in the public domain. Karati and Sarkar (Karati & Sarkar, 2017) commented for the ladder step used in `curve25519` specification (D. Bernstein, 2006, Appendix B):

> *"The structure of this ladder is not as regular as the ladder step on the Kummer line. This makes it difficult to optimally group together the multiplications for SIMD implementation."*

In this work, we aim to show that a higher level of parallelism can be achieved with new tweaks on the ladder step, see Figure 3.1. In the figure, $\mathcal{H}$ stands for Hadamard transformation, which inputs two coordinates, $X$ and $Z$ and outputs $X + Z$ and $X - Z$.

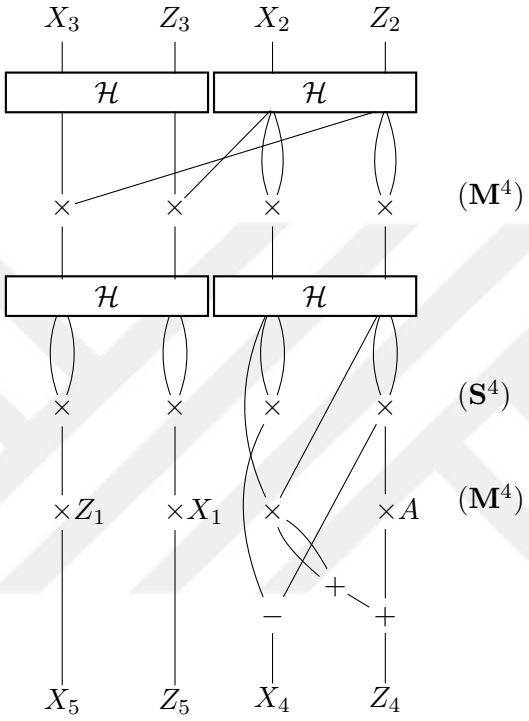


**Figure 3.1.** `DBLADD`: 4-Way Vectorized Ladder Step for the Curve $By^2 = x^3 + Ax^2 + x$.

The point doubling side of Figure 3.1 is recognizably different from Bernstein's diagram. Specifically, the squaring step now utilizes all 4 channels in vectorized form. On the other hand, an inspection of Figure 3.1 reveals that the outputs $X_4$, $Z_4$, $X_5$, and $Z_5$ agree with (3.2) up to a multiplication of the coordinates by a constant with no effect on the correctness of `DBLADD` routine.

The ladder step in Figure 3.1 takes $2\mathbf{M}^4 + 1\mathbf{S}^4$. Karati and Sarkar's 4-way vectorized ladder step (Karati & Sarkar, 2017, Fig. 1) takes $2\mathbf{M}^4 + 1\mathbf{S}^4 + 1\mathbf{d}^4$ ($\mathbf{d}^4$: A vector of four field multiplications by four small constants). There is a speed trade-off between these two approaches, which is not apparent immediately from the high-level operation counts:

- *Multiplication with constants:* A squared Kummer line requires one multiplication by $[a^2 + b^2, a^2 - b^2, a^2 + b^2, a^2 - b^2]$ followed by reduction (denoted $\mathbf{d}^4$), per ladder step. Such a multiplication-reduction does not occur in Figure 3.1.

- *Extra permutations:* Data transfers between SIMD channels occur in Hadamard transform and constant-time conditional point swap operations in both types of ladder steps. Our algorithm requires additional transfers and linear operations following the second Hadamard transform.

These two items constitute a speed trade-off which depends heavily on the cost $\mathbf{d}^4$ and the comparative throughput of SIMD multiplication and data transfer instructions, which can significantly vary depending on the micro-architecture.

## 3.4. IMPLEMENTATION ON AVX2

This section provides implementation details for the 4-way vectorization of the Montgomery ladder. Implementers are not limited to the specification of this section because Figure 3.1 is independent of the choices made here. The same applies to Section 3.5.

We fix $p = 2^{255} - 19$ and work over $GF(p)$. We start by explaining field multiplication. The discussion is narrowed to a field multiplication. On the other hand, the implementation computes 4 field multiplications simultaneously in vector form. We refer to (D. Bernstein et al., 2014) for a comprehensive explanation of the concept. We use core ideas from (D. Bernstein & Schwabe, 2012), (D. Bernstein et al., 2014), (Chou, 2015), and (Karati & Sarkar, 2017). Yet, we made different implementation choices.

**Multiplication.** We represent reduced field elements in 9 limbs rather than 10 and keep unreduced products in 11 limbs rather than 10. We justify how intermediate values always fit into $64$-bit registers without producing overflow. This operation is a hybridization of two commonly followed methods:

- doing the $255 \times 255 \rightarrow 510$-bit multiplication first and then reducing to $255$ bits, cf. (Karati & Sarkar, 2017) and
- merging reduction with integer multiplication and keeping elements always in the specified number of limbs, cf. (D. Bernstein, 2006).

These scenarios are not in the context of the 4-way ladder (Figure 3.1) and are thus omitted in this work.

We designed a two-layer implementation to carry out field multiplications with a redundant representation of elements. Both layers use a 3-way splitting strategy. Therefore, a field element is represented by 9 limbs, each of which can accommodate non-negative values smaller than $2^{64}$.

The higher layer is described as follows. A field element $u$ is represented by integers $u_0$, $u_1$, and $u_2$ such that $u = u_0 + 2^{85}u_1 + 2^{170}u_2$. We note that this is not a unique representation. Let $v$ be an integer also represented in the same way. We then have

congruence (3.4).

$$\begin{aligned} uv \equiv \quad & 2^0( \ u_0v_0 + \ 19u_1v_2 + \ 19u_2v_1 \ ) + \\ & 2^{85}( \ u_0v_1 + \quad u_1v_0 + \ 19u_2v_2 \ ) + \\ & 2^{170}( \ u_0v_2 + \quad u_1v_1 + \quad u_2v_0 \ ) \qquad (\mathrm{mod}\ p) \end{aligned} \tag{3.4}$$

The congruence $255 \equiv 0 \pmod 3$ helps significantly in obtaining simple formulas. If we did not have this condition, the given formulas would have contained several multiplications by 2 and multiplications by 19. Such a situation would have added more linear operations to the ladder step.

The nine long multiplications in the form $u_iv_j$ are reduced to six by three Karatsuba optimizations capable of sharing the sub-expressions $u_iv_i$. These operations are provided in (3.5).

$$\begin{aligned} & 2^0( \quad u_0v_0 + 19((u_1+u_2)(v_1+v_2) - u_1v_1 - u_2v_2) \ ) + \\ & 2^{85}( \ 19u_2v_2 + \quad (u_0+u_1)(v_0+v_1) - u_0v_0 - u_1v_1 \ ) + \\ & 2^{170}( \quad u_1v_1 + \quad (u_0+u_2)(v_0+v_2) - u_0v_0 - u_2v_2 \ ) \end{aligned} \tag{3.5}$$

This variant leads to more additions/subtractions, some of which can be shared. We eliminated these repeating operations at the cost of using more registers in our implementation. The additions of the form $u_i + u_j$ are 3-limb additions. All other additions and subtractions are 5-limb additions.

These high-level operations do not provide low-level details. For instance, we do not have hardware multipliers that can accommodate $85 \times 85 \rightarrow 170$-bit integer multiplications. Therefore, we further split each digit in the higher layer into three limbs provided in Equations 3.6.

$$\begin{aligned} u_0 &= a_0 + 2^{29}a_1 + 2^{57}a_2, \quad v_0 = b_0 + 2^{29}b_1 + 2^{57}b_2, \\ u_1 &= a_3 + 2^{29}a_4 + 2^{57}a_5, \quad v_1 = b_3 + 2^{29}b_4 + 2^{57}b_5, \\ u_2 &= a_6 + 2^{29}a_7 + 2^{57}a_8, \quad v_2 = b_6 + 2^{29}b_7 + 2^{57}b_8. \end{aligned} \tag{3.6}$$

For instance, $u_0v_0$ can be computed with the formulas given in Equation (3.7).

$$
\begin{aligned}
u_0 v_0 = \quad & 2^{0}(\ a_0 b_0 & & & & ) + \\
& 2^{29}(\ a_0 b_1 + & a_1 b_0 & & & ) + \\
& 2^{57}(\ a_0 b_2 + & a_2 b_0 + & 2a_1 b_1 & & ) + \\
& 2^{86}(\ a_1 b_2 + & a_2 b_1 & & & ) + \\
& 2^{114}(\ a_2 b_2 & & & & )
\end{aligned}
\tag{3.7}
$$

These operations take 9 multiplications and 5 additions, all of which can be directly carried out by the target hardware. Karatsuba optimization is not used here since the trade-off between multiplications and additions does not provide a practical speed-up at this level. The registers $a_0$, $a_1$, and $a_2$ are bounded carefully to prevent overflowing of the 64-bit registers, allowing the final carries to be delayed to the end of the field operation. More explicitly, the multiplication algorithm takes 9-limb integers and produces the 11 limbs given in Equations 3.8.

$$
\begin{aligned}
w_0 &= a_0 b_0 + 19(a_3 b_6 + a_6 b_3), \\
w_1 &= a_0 b_1 + a_1 b_0 + 19(a_3 b_7 + a_4 b_6 + a_6 b_4 + a_7 b_3), \\
w_2 &= a_0 b_2 + 2a_1 b_1 + a_2 b_0 + 19(a_3 b_8 + a_8 b_3 + 2(a_4 b_7 + a_7 b_4) + a_5 b_6 + a_6 b_5), \\
w_3 &= a_0 b_3 + a_3 b_0 + 2(a_1 b_2 + a_2 b_1) + 19(a_6 b_6 + 2(a_4 b_8 + a_5 b_7 + a_7 b_5 + a_8 b_4)), \\
w_4 &= a_0 b_4 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_4 b_0 + 19(a_5 b_8 + a_6 b_7 + a_7 b_6 + a_8 b_5), \\
w_5 &= a_0 b_5 + a_2 b_3 + a_3 b_2 + a_5 b_0 + 2(a_1 b_4 + a_4 b_1) + 19(a_6 b_8 + 2a_7 b_7 + a_8 b_6), \\
w_6 &= a_0 b_6 + a_3 b_3 + a_6 b_0 + 2(a_1 b_5 + a_2 b_4 + a_4 b_2 + a_5 b_1 + 19(a_7 b_8 + a_8 b_7)), \\
w_7 &= a_0 b_7 + a_1 b_6 + a_2 b_5 + a_3 b_4 + a_4 b_3 + a_5 b_2 + a_6 b_1 + a_7 b_0 + 19 a_8 b_8, \\
w_8 &= a_0 b_8 + a_2 b_6 + a_3 b_5 + a_5 b_3 + a_6 b_2 + a_8 b_0 + 2(a_1 b_7 + a_4 b_4 + a_7 b_1), \\
w_9 &= 2(a_1 b_8 + a_2 b_7 + a_4 b_5 + a_5 b_4 + a_7 b_2 + a_8 b_1), \\
w_{10} &= a_2 b_8 + a_5 b_5 + a_8 b_2,
\end{aligned}
\tag{3.8}
$$

satisfying the congruence in (3.9).

$$\begin{aligned} uv \equiv w \equiv (w_0 + 2^{29}w_1 + 2^{57}w_2)+ \\ 2^{85}(w_3 + 2^{29}w_4 + 2^{57}w_5)+ \\ 2^{170}(w_6 + 2^{29}w_7 + 2^{57}w_8)+ \\ 2^{255}(w_9 + 2^{29}w_{10}) \pmod{2^{255} - 19} \end{aligned} \tag{3.9}$$

We do not perform all these $9 \times 9 = 81$ multiplications but just $9 \times 6 = 54$. It is due to the shared-Karatsuba approach explained earlier.

**Input/output specification.** We set essential bounds in inequalities (3.10) for the input and output limbs.

$$\begin{aligned} 0 \leq a_0, a_3, a_6 < 2^{29} + k \\ 0 \leq a_1, a_2, a_4, a_5, a_7, a_8 < 2^{28} + k. \end{aligned} \tag{3.10}$$

The $k = 173$ is a constant whose role will become apparent in the reduction step. We always ensure the accuracy of these bounds after a reduction step, which provides an easy-to-follow input/output specification.

The limbs $w_i$ are displayed explicitly (in the item list) to help easily check the output's boundaries. In particular, we need to show that these limbs cannot exceed $2^{64}$. Now, inputting the largest possible values for each limb of $u$ and $v$ and evaluating the formulas provided in inequalities (3.11),

$$\begin{aligned} &w_0 < 2^{63.29}, \quad w_1 < 2^{63.29}, \quad w_2 < 2^{63.88}, \\ &w_3 < 2^{63.91}, \quad w_4 < 2^{62.95}, \quad w_5 < 2^{62.98}, \\ &w_6 < 2^{62.59}, \quad w_7 < 2^{61.05}, \quad w_8 < 2^{60.17}, \\ &w_9 < 2^{59.59}, \quad w_{10} < 2^{57.59}. \end{aligned} \tag{3.11}$$

All of these values can be accommodated without overflow in 64-bit registers $w_i$.

Even if we have computed all $w_i$, we are not entirely done yet. We only have a semi-reduced $w$ satisfying the congruence (3.12).

$$w \equiv uv \pmod{2^{255} - 19} \tag{3.12}$$

We must do the carries to eliminate $w_9$ and $w_{10}$ and match the output requirements in

inequalities (3.13).

$$\begin{aligned} 0 \leq w_0, w_3, w_6 < 2^{29} + k \\ 0 \leq w_1, w_2, w_4, w_5, w_7, w_8 < 2^{28} + k \end{aligned} \tag{3.13}$$

The outputs' bounds agree with the input specification of $u$ and $v$ in (3.10).

**Carries (Reduction after multiplication).** This operation is composed of several steps. Each step transforms $w$ towards satisfying the input/output specification without violating the congruence in (3.12) and producing an overflow. The reduction proceeds as in the set of assignments (3.14).

$$\begin{aligned} \mathbf{1}: & \ t \leftarrow \lfloor w_9/2^{29} \rfloor,\ w_9 \leftarrow w_9 \bmod 2^{29},\ w_{10} \leftarrow w_{10} + t, \\ \mathbf{2}: & \ w_0 \leftarrow w_0 + 19w_9,\ w_9 \leftarrow 0, \\ \mathbf{3}: & \ w_1 \leftarrow w_1 + 19w_{10},\ w_{10} \leftarrow 0, \\ \mathbf{4}: & \ t \leftarrow \lfloor w_0/2^{29} \rfloor,\ w_0 \leftarrow w_0 \bmod 2^{29},\ w_1 \leftarrow w_1 + t, \\ \mathbf{5}: & \ t \leftarrow \lfloor w_1/2^{28} \rfloor,\ w_1 \leftarrow w_1 \bmod 2^{28},\ w_2 \leftarrow w_2 + t, \\ \mathbf{6}: & \ t \leftarrow \lfloor w_2/2^{28} \rfloor,\ w_2 \leftarrow w_2 \bmod 2^{28},\ w_3 \leftarrow w_3 + t, \\ \mathbf{7}: & \ t \leftarrow \lfloor w_3/2^{29} \rfloor,\ w_3 \leftarrow w_3 \bmod 2^{29},\ w_4 \leftarrow w_4 + t, \\ \mathbf{8}: & \ t \leftarrow \lfloor w_4/2^{28} \rfloor,\ w_4 \leftarrow w_4 \bmod 2^{28},\ w_5 \leftarrow w_5 + t, \\ \mathbf{9}: & \ t \leftarrow \lfloor w_5/2^{28} \rfloor,\ w_5 \leftarrow w_5 \bmod 2^{28},\ w_6 \leftarrow w_6 + t, \\ \mathbf{10}: & \ t \leftarrow \lfloor w_6/2^{29} \rfloor,\ w_6 \leftarrow w_6 \bmod 2^{29},\ w_7 \leftarrow w_7 + t, \\ \mathbf{11}: & \ t \leftarrow \lfloor w_7/2^{28} \rfloor,\ w_7 \leftarrow w_7 \bmod 2^{28},\ w_8 \leftarrow w_8 + t, \\ \mathbf{12}: & \ t \leftarrow \lfloor w_8/2^{28} \rfloor,\ w_8 \leftarrow w_8 \bmod 2^{28},\ w_0 \leftarrow w_0 + 19t \\ \mathbf{13}: & \ t \leftarrow \lfloor w_0/2^{29} \rfloor,\ w_0 \leftarrow w_0 \bmod 2^{29},\ w_1 \leftarrow w_1 + t. \end{aligned} \tag{3.14}$$

In this sequence of operations, we are accumulating on registers $w_i$, which contain values potentially very close to $2^{64}$. Once more, we need to justify that these additions do not constitute overflow.

- **Step 1:** $t = \lfloor w_9/2^{29} \rfloor < 2^{59.59-29} = 2^{30.59}$. So, $w_{10} + t < 2^{57.59} + 2^{30.59} < 2^{57.60}$. Therefore, the updated value of $w_{10}$ still fits into 64 bits. A bit of care is needed now to track the updated $w_9$. Although we computed $w_9 \leftarrow w_9 \bmod 2^{29}$ for maximum possible inputs, the updated value of $w_9$ can still get values as large as $2^{29} - 1$ for some other input. Therefore, we assume that we take $w_9 = 2^{29} - 1$ from here for the sake of our inspection.

- **Step 2:** Now, we must have $w_0 + 19w_9 < 2^{63.29} + 19(2^{29} - 1) < 2^{63.30}$. Multiplication by 19 here is performed with $32 \times 32 \rightarrow 64$-bit multiplication instruction `vpmuludq` since both 19 and $w_9$ are smaller than $2^{32}$.
- **Step 3:** Similarly, we must have $w_1 + 19w_{10} < 2^{63.29} + 19(2^{57.60}) < 2^{63.75}$. We note that $19w_{10}$ is computed as $19w_{10} = 16w_{10} + 2w_{10} + w_{10}$ by using `vpaddq` and `vpsllq` instructions because $w_{10}$ can exceed $2^{32}$ and, thus, is unsuitable to be inputted to `vpmuludq`. We note that $w_9 \leftarrow 0$ and $w_{10} \leftarrow 0$ are displayed just for mathematical correctness.
- **Steps 4-11:** Repeating the same inspection by computing each step sequentially, we get $w_{1,\ldots,8} < 2^{64}$ after additions as expected. Limbs $w_{0,\ldots,7}$ obey the input/output specification after reducing $w_{1,2,4,5,7}$ modulo $2^{28}$ and $w_{3,6}$ modulo $2^{29}$. Again, we assume that $w_{0,3,6} = 2^{29} - 1$ and $w_{1,2,4,5,7,8} = 2^{28} - 1$ after the modular reductions are performed for these digits.
- **Step 12:** We get $t = \lfloor w_8/2^{28} \rfloor < 2^{60.17-28} = 2^{32.17}$. So, $w_0 + 19t < (2^{29} - 1) + 19(2^{32.17}) < 2^{36.43}$. Now, $w_8$ obeys the input/output specification after being reduced to modulo $2^{28}$. We note that $19t$ is computed as $19t = 16t + 2t + t$ since $w_8$ can exceed $2^{32}$.
- **Step 13:** We get $w_1 + t < (2^{28} - 1) + (2^{36.43-29}) < 2^{28} + 173$. This upper bound explains the value of $k$. We note that a lower upper bound can be found with increased precision in calculations. Moreover, much larger values for $k$ work without producing overflow in reduction[2], but 173 is adequate to test the stability of limbs.

Now, all $w_i$ agree with the input/output specification of $u_i$ and $v_i$. We intentionally added $k$ to all limbs in the input/output specification rather than adding just to $w_1$ because

- these additions simplify the notation, and
- we need such extra additions when designing parallel carry chains.

The reduction step can be summarized as $h_9 \rightarrow h_{10}$, followed by a very long sequence (3.15).

$$h_8 \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 \rightarrow h_5 \rightarrow h_6 \rightarrow h_7 \rightarrow h_8 \rightarrow h_0 \rightarrow h_1 \tag{3.15}$$

[2]We reiterate that we use a redundant representation. Therefore, reduction does not produce a unique representative. Nevertheless, we still call it *reduction* since we can do arithmetic in this form.

We do it faster by computing two sequences given in (3.16)

$$\begin{aligned} &h_4 \longrightarrow h_5 \rightarrow h_6 \rightarrow h_7 \rightarrow h_8 \rightarrow h_0 \rightarrow h_1, \\ &h_9 \rightarrow h_{10}, \quad h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 \rightarrow h_5 \end{aligned} \tag{3.16}$$

in parallel at the processor's ports. We refer to (D. Bernstein & Schwabe, 2012) and (Chou, 2015) for similar optimizations.

In this parallel reduction, not only $w_1$ but also $w_5$ can exceed $2^{28} - 1$ by $k$. But we have already relaxed $w_5$ (like all other limbs) by additions of $k$ in our inspection.

**Squaring.** Squaring can be explained as a simplified multiplication routine given in (3.17).

$$\begin{aligned} 2^0(\quad & u_0^2 + \; 19((u_1+u_2)^2 - u_1^2 - u_2^2) \;) + \\ 2^{85}(\; & 19u_2^2 + \quad (u_0+u_1)^2 - u_0^2 - u_1^2 \;) + \\ 2^{170}(\quad & u_1^2 + \quad (u_0+u_2)^2 - u_0^2 - u_2^2 \;) \end{aligned} \tag{3.17}$$

The nine long multiplications in the form $u_i v_j$ are now reduced to six squares. In addition, the computation of $u_0^2$ can be further optimized at the lower level in the form given in Equation (3.18).

$$\begin{aligned} u_i^2 = \quad & 2^0(\; a_0^2 \qquad\qquad\quad) + \\ & 2^{29}(\; (2a_0)a_1 \qquad\quad) + \\ & 2^{57}(\; (2a_1)a_1 + \; (2a_0)a_2 \;) + \\ & 2^{86}(\; (2a_1)a_2 \qquad\quad) + \\ & 2^{114}(\; a_2^2 \qquad\qquad\quad) \end{aligned} \tag{3.18}$$

Similar applies to the other squarings. Our implementation delays multiplication by twos and pushes them towards the higher layer.

**Squeeze/Unsqueeze.** A field element $w$ satisfying the input/output specification can be squeezed from 9 limbs to 5 by computing (3.19).

$$w_{i+4} \leftarrow w_{i+4} \oplus 2^{32}\, w_i \quad \text{for } \; i = 0, 1, 2, 3 \tag{3.19}$$

Now, $w$ is represented only by $w_4, w_5, w_6, w_7$, and $w_8$. Linear operations such as (field) additions and subtractions can be handled in this form, provided that computed values do not exceed $2^{32} - 1$. It is always the case in our implementation.

A squeezed field element is unsqueezed into the original form by computing (3.20).

$$w_i \leftarrow w_{i+4}/2^{32} \quad \text{for } i = 0, 1, 2, 3 \quad \text{and}$$
$$w_{i+4} \leftarrow w_{i+4} \bmod 2^{32} \quad \text{for } i = 0, 1, 2, 3 \tag{3.20}$$

at multiplication, squaring, and reduction moments. We note that we skip computing $w_{i+4} \leftarrow w_{i+4} \bmod 2^{32}$ before multiplication and squaring since the higher 32 bits are not taken into consideration by `vpmuludq` instruction. See also (D. Bernstein et al., 2014).

This squeeze/unsqueeze method is adapted from the software introduced by (D. Bernstein et al., 2014). The difference is that we group the limbs of a field element where Bernstein, Chuengsatiansup, Lange, and Schwabe group points on a genus 2 Kummer surface.

Despite the added cost of squeezing and unsqueezing, linear operations in squeezed form can be done faster and save cycles in total.

**Double Hadamard.** This step can be put in a 4-way vectorized form in modulus $2^{255} - 19$, as given in Equation (3.21).

$$\begin{aligned}
&(\mathcal{H} \times \mathcal{H})(X_3, Z_3, X_2, Z_2) = \\
&\quad (X_3 + Z_3, X_3 - Z_3, X_2 + Z_2, X_2 - Z_2) = \\
&\quad (X_3 + Z_3, X_3 + (3p - Z_3), X_2 + Z_2, X_2 + (3p - Z_2))
\end{aligned} \tag{3.21}$$

The additions of $3p$ are to ensure that $\mathcal{H} \times \mathcal{H}$ (double Hadamard) produces non-negative values for output limbs. We drop the word "double" for simplicity. This $3p$ needs to be prepared with some care, as given in (3.22).

$$\begin{aligned}
&2^0([3(2^{29} - 19)] + 2^{29}[3(2^{28} - 1)] + 2^{57}[3(2^{28} - 1)]) + \\
&2^{85}(\ [3(2^{29} - 1)] + 2^{29}[3(2^{28} - 1)] + 2^{57}[3(2^{28} - 1)]) + \\
&2^{170}(\ [3(2^{29} - 1)] + 2^{29}[3(2^{28} - 1)] + 2^{57}[3(2^{28} - 1)])
\end{aligned} \tag{3.22}$$

Observe that each limb[3] is greater than the corresponding maximum bound in the input/output specification.

All of the limbs of $X_3 + Z_3$, $X_3 + (3p - Z_3)$, $X_2 + Z_2$, and $X_2 + (3p - Z_2)$ are always less than $2^{32}$ after the first Hadamard operation in Figure 3.1. To show this, we concentrate on the linear operations at the right of the bottom of the figure.

[3]The value of each limb appears in square brackets.

- $Z_4$ is computed as the sum of three values. To simplify our analysis, we assume that all inputs to these additions take the largest possible values. Then, $w_{0,3,6} = 3((2^{29}-1)+k) < 2^{31}$ and $w_{1,2,4,5,7,8} = 3((2^{28}-1)+k) < 2^{30}$.
- $X_4$ is computed as the difference between two values. We assume that minuend takes the largest and the subtrahend takes the smallest possible value. Then, $w_{0,3,6} = ((2^{29}-1)+k) + (2(2^{29}-1)-0) < 2^{31}$ and $w_{1,2,4,5,7,8} = ((2^{28}-1)+k) + (2(2^{28}-1)-0) < 2^{30}$. Observe that we added $2p$ rather than $3p$ this time, which is adequate because $2(2^{29}-1) > (2^{29}-1)+k$ and likewise $2(2^{28}-1) > (2^{28}-1)+k$. So, even if the subtrahend takes the maximum possible value, the limbs are still non-negative.

Up to this point, we showed that $w_i$ of both $X_4$ and $Z_4$ fit into 31 bits. We now feed these extreme values[4] to the first Hadamard operation. Clearly, we have $0 \leq w_i < 2^{32}$ for $X + Z$. Separately, assuming that $w_i = 0$ for $Z$, we have $0 \leq w_i < 2^{32}$ for $X + (3p - Z)$. Analyzing the second Hadamard is even simpler since its inputs are already reduced values.

**Fast carries (Fast reduction after Hadamard).** Following a Hadamard step, a reduction operation must be applied to the output to match the input/output specification. This time, the reduction can be performed faster since we do not have limbs $w_9$ and $w_{10}$. Therefore, a fast reduction can be defined as a trimmed version of the reduction after multiplication with steps given in a set of assignments (3.23).

$$
\begin{aligned}
\mathbf{1}: t &\leftarrow \lfloor w_0/2^{29} \rfloor,\ w_0 \leftarrow w_0 \bmod 2^{29},\ w_1 \leftarrow w_1 + t,\\
\mathbf{2}: t &\leftarrow \lfloor w_1/2^{28} \rfloor,\ w_1 \leftarrow w_1 \bmod 2^{28},\ w_2 \leftarrow w_2 + t,\\
\mathbf{3}: t &\leftarrow \lfloor w_2/2^{28} \rfloor,\ w_2 \leftarrow w_2 \bmod 2^{28},\ w_3 \leftarrow w_3 + t,\\
\mathbf{4}: t &\leftarrow \lfloor w_3/2^{29} \rfloor,\ w_3 \leftarrow w_3 \bmod 2^{29},\ w_4 \leftarrow w_4 + t,\\
\mathbf{5}: t &\leftarrow \lfloor w_4/2^{28} \rfloor,\ w_4 \leftarrow w_4 \bmod 2^{28},\ w_5 \leftarrow w_5 + t,\\
\mathbf{6}: t &\leftarrow \lfloor w_5/2^{28} \rfloor,\ w_5 \leftarrow w_5 \bmod 2^{28},\ w_6 \leftarrow w_6 + t,\\
\mathbf{7}: t &\leftarrow \lfloor w_6/2^{29} \rfloor,\ w_6 \leftarrow w_6 \bmod 2^{29},\ w_7 \leftarrow w_7 + t,\\
\mathbf{8}: t &\leftarrow \lfloor w_7/2^{28} \rfloor,\ w_7 \leftarrow w_7 \bmod 2^{28},\ w_8 \leftarrow w_8 + t,\\
\mathbf{9}: t &\leftarrow \lfloor w_8/2^{28} \rfloor,\ w_8 \leftarrow w_8 \bmod 2^{28},\ w_0 \leftarrow w_0 + 19t\\
\mathbf{10}: t &\leftarrow \lfloor w_0/2^{29} \rfloor,\ w_0 \leftarrow w_0 \bmod 2^{29},\ w_1 \leftarrow w_1 + t.
\end{aligned}
\tag{3.23}
$$

[4]Noticed that all these operations can be performed in squeezed form.

We do better by computing these operations in squeezed form and computing (3.24)

$$\begin{aligned} &h_0 \to h_1 \to h_2 \to h_3 \longrightarrow h_4 \to h_5, \\ &h_4 \to h_5 \to h_6 \to h_7 \to h_8 \to h_0 \to h_1 \end{aligned} \tag{3.24}$$

in parallel on two 32-bit SIMD channels. We do not further exploit the processor's port-level parallelism since the sequence is short enough to produce low latency.

## 3.5. IMPLEMENTATION ON AVX-512

`AVX-512` provides 8-way SIMD multiplication with the `vpmuludq` instruction, providing twice as many $32 \times 32 \to 64$-bit multipliers as `AVX2`. Therefore it is reasonable to question whether the 4-way vectorized ladder can be computed faster on `AVX-512`. Since Figure 3.1 supports up to 4-way vectorization, we need to vectorize the field arithmetic in 8/4=2-way form to get a $4 \times 2$ way ladder.

Although our 9-limb multiplication fits nicely on $4 \times 1$ ladder, it does not seem to be the best choice for its $4 \times 2$ counterpart. Yet, there is room for research in finding a fast 2-way vectorization of the 9-limb multiplication described in Section 3.5. We do not pursue this idea further here.

As a practical solution, we decided to use a 2-way vectorized version of the 10-limb multiplication algorithm using Radix-$2^{25.5}$ (D. Bernstein, 2006). This algorithm was previously used with minor modifications by (D. Bernstein & Schwabe, 2012) and (Chou, 2015). Fortunately, we could reuse optimized codes freely available in the public domain. In particular, we used the 2-way `AVX2` targeted `intmul` and `intsqr` functions from

`hp-ecc-vec/src/eltfp25519_2w_redradix.c`

by Faz Hernández, López, Dahab [5] and have those functions run on `AVX-512`. Then, we applied the ladder step in Figure 3.1 to get a $4 \times 2 = 8$-way vectorized implementation of the Montgomery ladder over the field $GF(2^{255} - 19)$. The speed comparison is given in Section 3.6.

## 3.6. RESULTS

*The final inversion.* Our implementation reduces the output of scalar multiplication to a unique representative in the underlying field in radix 256. Therefore, we compute

[5] `https://github.com/armfazh/hp-ecc-vec` (last accessed 2022-03-18)

$X_2/Z_2$ after the main loop. We integrated Nath and Sarkar's (Nath & Sarkar, 2018) freely available and optimized inversion software without further modification. In particular, we used

`pmp-inv-master/p25519/SL-DCC/1`

which requires the `BMI2` instruction set. Nath and Sarkar report 9301 Skylake cycles for this inversion.

*Working environment and AVX instructions.* As stated in the declaration, the vectorized implementations and related measurements were made on an Intel Core™ Skylake i9-7900X CPU. Here, Skylake represents the codename of the micro-architecture family used in the processor. It was launched in August 2015. It is the successor to the Broadwell micro-architecture. Its 14 nm micro-architecture is also used in the following processor generations, such as Kaby Lake, Coffee Lake and Comet Lake. The processor number of the CPU is i9-7900X. According to (Intel, n.d.-c), this number consists of the brand modifier, generator indicator, SKU numeric digits and product line suffix. Intel Core™ i9-7900X CPU (Intel, n.d.-a) has 10 cores and can work on 20 parallel threads with the use of Intel® Hyper-Threading Technology. Its base frequency is 3.30 GHz (with 4.30 GHz max turbo frequency), and it has 13.75 MB L3 CPU cache memory. We decided to work with this CPU because it allows working with both AVX2 and AVX-512 instruction sets. The most commonly used vectorized instructions in our implementations include arithmetic instructions (e.g., addition, subtraction, multiplication), shift instructions (e.g., left shift, right shift) and swizzle instructions (e.g., shuffle, blend). The latency and throughput values of some of these AVX2 and AVX-512 instructions on Skylake micro-architecture can be found in Table 3.1 and Table 3.3, respectively. Note that latency denotes the total number of cycles that must be taken to execute the given instruction. Throughput means the average number of clock cycles per instruction, also indicated as CPI (cycles per instruction). In addition, the values in Table 3.2 and Table 3.4 represent the latency and throughput values of the exact instructions on Icelake micro-architecture. These values are given so that the values on Skylake can be compared with the ones on more recent micro-architectures, such as Icelake, released in September 2019. The values are taken from (Intel, n.d.-b).

*Measuring cycles.* We measure cycles for variable-scalar variable-point multiplication only. Our code changes the base point and scalar at each iteration and excludes extra cycles coming from this randomization. Also, our implementation chains the outputs to prevent the compiler from removing portions of the code. Measured cycle counts are given in Table 3.5, along with selected results from the literature. The table is limited to our results and recently published measurements for the Skylake micro-architecture.

| Instruction | Latency | Throughput(CPI) |
|---|---|---|
| vpaddq | 1 | 0.33 |
| vpsubq | 1 | 0.33 |
| vpmuludq | 5 | 0.5 |
| vpsllq | 1 | 0.5 |
| vpsrlq | 1 | 0.5 |
| vpblendd | 1 | 0.33 |
| vpshufd | 1 | 1 |

**Table 3.1.** Latency and Throughput Values of Selected AVX2 Instructions on Skylake Micro-Architecture

| Instruction | Latency | Throughput(CPI) |
|---|---|---|
| vpaddq | 1 | 0.33 |
| vpsubq | 1 | 0.33 |
| vpmuludq | – | 0.5 |
| vpsllq | 1 | 0.5 |
| vpsrlq | 1 | 0.5 |
| vpblendd | 1 | 0.33 |
| vpshufd | 1 | 0.5 |

**Table 3.2.** Latency and Throughput Values of Selected AVX2 Instructions on Icelake Micro-Architecture

| Instruction | Latency | Throughput(CPI) |
|---|---|---|
| vpaddq | 1 | 0.5 |
| vpsubq | 1 | 0.5 |
| vpmuludq | 5 | 0.5 |
| vpsllq | 1 | 1 |
| vpsrlq | 1 | 1 |
| vpblendmq | 1 | 0.5 |
| vpshufd | 1 | 1 |

**Table 3.3.** Latency and Throughput Values of Selected AVX-512 Instructions on Skylake Micro-architecture

| Instruction | Latency | Throughput(CPI) |
|---|---|---|
| vpaddq | 1 | 0.5 |
| vpsubq | 1 | 0.5 |
| vpmuludq | – | 1 |
| vpsllq | 1 | 1 |
| vpsrlq | 1 | 1 |
| vpblendmq | 1 | 0.5 |
| vpshufd | 1 | 1 |

**Table 3.4.** Latency and Throughput Values of Selected AVX-512 Instructions on Icelake Micro-architecture

**Table 3.5.** Skylake Cycles for Variable-Scalar Variable-Point Multiplication

| ladder | instr. set | limbs | cycles (median) | ratio |
|---|---|---|---|---|
| $4 \times 1$ | AVX2 | 10 | 123 102, (Karati & Sarkar, 2017) | 1x |
| $4 \times 1$ | AVX2 | 10 | 116 654, *this work* | .95x |
| $1 \times 1$ | BMI2 | 4 | 113 874, (Oliveira, López, Hışıl, Faz-Hernández, & Rodríguez-Henríquez, 2017) | .93x |
| $2 \times 2$ | AVX2 | 5 | 99 400, (Faz-Hernández et al., 2019) | .81x |
| $4 \times 1$ | AVX2 | 9 | 98 484, *this work* | .80x |
| $4 \times 1$ | AVX2 | 10 | 95 437, (Nath & Sarkar, 2020) | .78x |
| $2 \times 4$ | AVX-512 | 5 | 81 600, (Faz-Hernández et al., 2019) | 1x |
| $4 \times 2$ | AVX-512 | 5 | 74 368, *this work* | .91x |

Table 3.5 justifies our motivation in proposing the 9-limb representation in Section 3.5. The 9-limb method is faster than 10 *in the context* of our 4-way ladder and specified implementation platform. See (Nath & Sarkar, 2020) for new results on another Skylake CPU with a different micro-architecture. Nath and Sarkar's fastest implementation uses a small curve constant where our proposed algorithm does not require such an

assumption. Our implementation takes precisely the same number of cycles for any curve constant, which can be provided parametrically.In contrast, Nath and Sarkar's reported cycle counts heavily depend on the size of the hard-coded curve constants. We note that both implementations use the same elliptic curve, namely `curve25519`, for a fair comparison.

Figure 3.1 shows its real potential in our AVX-512 implementation. The reported 74368 cycles set the new record among `curve25519` family of implementations, to the best of our knowledge.

*Variable-scalar fixed-base multiplication.* Our implementation can be used directly in fixed-base multiplication without further modification. Nevertheless, one can make precomputation on a fixed-base point to speed up the process. In that case, we refer to (Oliveira et al., 2017, Algorithm 5).

Apart from architecture-dependent discussions, we expect that our 4-way ladder will become even more convenient if the current trend of increasing the level of SIMD parallelism in hardware continues. We reiterate that the speeds we achieve are common for all Montgomery curves, not specific to ones with small constants.

# CHAPTER 4
# DETECTING (NON-)ISOGENOUS CURVES

In Chapter 4, Schoof's algorithm is explained and illustrated with a numerical example as a preparation for the SEA algorithm at first. Then, the SEA algorithm is described in detail. In addition to Schoof's and SEA algorithms, all auxiliary algorithms used in these algorithms are explicitly given. Afterwards, a toy example covering all the cases of the SEA algorithm is given. Finally, an early abort method for detecting (non-)isogenous curves is presented at the end of the chapter.

## 4.1. SCHOOF'S ALGORITHM

Schoof's Algorithm (Schoof, 1985) determines the number of points on an elliptic curve defined over a finite field. The algorithm uses Hasse's Bound, Frobenius endomorphism, division polynomials and CRT to determine the number of points on an elliptic curve. It is the first point counting algorithm that has polynomial-time complexity. The approach Schoof followed led the way to new improvements in the field and caused the discovery of the SEA algorithm. Before heading into the SEA algorithm, at first Schoof's algorithm will be explained in detail.

Let $E$ be an elliptic curve over a finite field $\mathbb{F}_q$ with characteristic $q \neq 2, 3$. It can be defined that $q = p^n$, where $p$ is a prime number and $n$ is the degree of the prime field with $n \geq 1$. The curve $E$ is defined with the short Weierstrass equation $y^2 = x^3 + a_4x + a_6$ where $a_4, a_6 \in \mathbb{F}_q$. The number of points on the elliptic curve $E/\mathbb{F}_q$, which is the cardinality of $E(\mathbb{F}_q)$, can be found as,

$$\#E(\mathbb{F}_q) = q + 1 - t, \tag{4.1}$$

where $t$ is the trace of Frobenius for $E$. The result at the end of Hasse's theorem of elliptic curves can be utilized when determining the boundaries of the trace of Frobenius as $|t| \leq 2\sqrt{q}$.

Elliptic curves used in cryptography are generally defined over very large fields. As $q$ gets larger, the computations performed under this bound will be more challenging to handle. Therefore, the calculation of $t$ must be done modulo a number $N$. However, computing the modulus with $N$ having a bound of $4\sqrt{q}$ will be exhausting. So, instead

of taking $t$ modulo $N$ directly, small prime numbers that are relatively prime with $q$ can be used. The main strategies of Schoof's Algorithm and SEA Algorithm stem from using small primes $\ell$ at the computation of $t$. The computations $t$ modulo $\ell$, with all possible small primes constructing a set $S$, can be stopped according to

$$\prod_{\ell \in S} \ell > 4\sqrt{q}. \tag{4.2}$$

Eventually, the only work to do is to calculate $t$ with the help of CRT. The overall running time complexity of Schoof's algorithm is $O(\log^8 q)$ bit operations. If efficient implementation methods such as fast polynomial and integer arithmetic are used in the computations, this complexity decreases to $\tilde{O}(\log^5 q)$ bit operations. More information about the calculation of the complexity of Schoof's algorithm can be found in (Blake et al., 1999).

In Schoof's algorithm, each trace $t_\ell$ collected modulo prime $\ell$ is stored with the corresponding prime $\ell$. After completing the trace collection, these trace-prime pairs are sent to CRT to detect the trace of Frobenius $t$. The trace gathering steps until the application of CRT are exhausting and should be adequately investigated. The operations for $\ell = 2$ are distinct from the other and more straightforward, so it is good to handle this case separately.

### 4.1.1 TRACE COMPUTATION MODULO 2

The trace computation modulo 2 is the simplest case and does not require complex computation. The trace $t$ modulo 2 can only be 0 or 1, and the choice can be made by checking the existence of a root of $x^3 + a_4x + a_6$ other than 1 in $\mathbb{F}_q$. The presence of a root of $x^3 + a_4x + a_6$ indicates a point of order 2. This point is on the elliptic curve $E/\mathbb{F}_q$ in the form of $(x, 0)$ where $x \in \mathbb{F}_q$. Substituting this point into the equation $y^2 = x^3 + a_4x + a_6$ reveals that there must be a root in the polynomial $x^3 + a_4x + a_6$, which lies on $\mathbb{F}_q$.

The equation

$$\prod_{\alpha \in \mathbb{F}_q} (x - \alpha) = x^q - x.$$

shows that the roots of the polynomial $x^q - x$ are all of the roots in $\mathbb{F}_q$. Hence, computing the Greatest Common Divisor ($\gcd$) of the polynomials $x^q - x$ and $x^3 + a_4x + a_6$ is enough to ensure the existence of a root in a more efficient way than finding the roots. If $\gcd$ is not equal to 1, trace $t_2$ is set to 0; otherwise, it is set to 1. When the value of $q$ is very large (the usual case in the cryptographic setting), computing the $\gcd$ will be tedious and expensive. That is why, first, the polynomial $x^q - x$ is reduced modulo

$x^3 + a_4x + a_6$ by using modular exponentiation at $x^q$ mod $x^3 + a_4x + a_6$, then the $\gcd$ of the result and the polynomial $x^3 + a_4x + a_6$ is taken.

### 4.1.2 TRACE COMPUTATION MODULO $\ell$

For odd primes, the trace computation relies on the characteristic equation

$$\varphi^2 - t\varphi + q = 0 \tag{4.3}$$

where $\varphi$ is the Frobenius Endomorphism that sends a point $P = (x, y) \in E/\mathbb{F}_q$ to $(x^q, y^q) \in E/\bar{\mathbb{F}}_q$ and $t$ is the trace of Frobenius which is wanted to be calculated. This characteristic equation can be arranged and written as

$$(x^{q^2}, y^{q^2}) \oplus q(x, y) = t(x^q, y^q). \tag{4.4}$$

The coordinates given in the equation can be computed with fast elliptic curve point multiplication methods, and different values for $t$ can be searched until equality is found. However, this process will quickly become computationally infeasible due to the large value of $q$. Instead, Schoof (1985) thought that for every different odd prime $\ell$, the coordinates could be computed modulo the $\ell$-th division polynomial $\psi_\ell$. Usage of the $\ell$-th division polynomial makes working in the $\ell$-torsion possible because roots of the $\ell$-th division polynomial correspond to the x-coordinates of $\ell$-torsion points. A quotient ring $Q_\ell = \mathbb{F}_q[x, y]/(y^2 - x^3 - a_4x - a_6, \psi_\ell)$ can be created by utilizing the $\ell$-th division polynomial. Working under this ring reduces the effort needed to compute the coordinates in Equation (4.4).

Schoof (1985) expressed that the problem of finding $t$ in Equation (4.4) can be separated into smaller problems by dealing not with $q$ but with $q_\ell \equiv q \pmod{\ell}$. Hence, the original problem turns into

$$(x^{q^2}, y^{q^2}) \oplus q_\ell(x, y) = t_\ell(x^q, y^q) \tag{4.5}$$

where $t_\ell \equiv t \pmod{\ell}$ and $(x, y)$ is a point $\in E[\ell]$. The only unknown variable in the equation is $t_\ell$, and the terms on the left-hand side can be computed directly. $(x^{q^2}, y^{q^2})$ can be computed with the help of the fast, modular exponentiation methods such as square-and-multiply and $(x_{q_\ell}, y_{q_\ell}) = q_\ell(x, y)$ can be computed by elliptic curve scalar multiplication methods or with Equation (2.35). Herein, there are two different cases which must be considered. These cases differ according to the equality of the points $(x^{q^2}, y^{q^2})$ and $(x_{q_\ell}, y_{q_\ell})$. Since these expressions are less likely to be equal, the case $(x^{q^2}, y^{q^2}) \neq (x_{q_\ell}, y_{q_\ell})$ is handled first to increase the algorithm's efficiency. In this case, the points can be added because of the inequality, i.e., the points do not violate the point

addition formula. The point addition can be done with the point addition formula given in Section 2.2 as

$$\begin{aligned} x' &= \left(\frac{y_{q_\ell} - y^{q^2}}{x_{q_\ell} - x^{q^2}}\right)^2 - x^{q^2} - x_{q_\ell}, \\ y' &= \left(\frac{y_{q_\ell} - y^{q^2}}{x_{q_\ell} - x^{q^2}}\right)(x^{q^2} - x') - y^{q^2}. \end{aligned} \tag{4.6}$$

Computing only the x-coordinate at first can increase efficiency, and this can be achieved if $x'$ consists only of $x$ values. Therefore, $(y_{q_\ell} - y^{q^2})^2$ must be rewritten with x-coordinates. To achieve this, $y^2$ can be left alone, and by using the defining equation of $E/\mathbb{F}_q : y^2 = x^3 + a_4x + a_6$, it can be substituted with $x$ values. Accordingly, $(y_{q_\ell} - y^{q^2})^2 = y^2(y_{q_\ell}/y - y^{q^2-1})^2 = (x^3 + a_4x + a_6)(y_{q_\ell}/y - y^{q^2-1})^2$. The prime power $q$ is odd, so $q^2 - 1$ is even. Then, $y^{q^2-1} = (x^3 + a_4x + a_6)^{\frac{q^2-1}{2}}$. Moreover, by using Equation (2.35), $y_{q_\ell}/y$ can also be written only with $x$-coordinates. Therefore, $x'$ is a function in x-coordinate only.

After the computation of $x'$, scalar multiplication with potential $t'$ values must be tried in the range $t' \in [0, (\ell - 1)/2]$ with a brute force approach. Trying the values up to $(\ell - 1)/2$ is enough because, for every value of $t'$ in the range, scalar multiplication with $t'$ and its additive inverse $\ell - t'$ gives the same results. The scalar multiplication $t'(x^q, y^q) = (x^q_{t'}, y^q_{t'})$ can be done by using a suitable method such as double-and-add or Montgomery Ladder (Montgomery, 1987). For each of the multiplications, $x^q_{t'}$ is compared with $x'$ modulo $\psi_\ell$, since $x' \equiv x^q_{t'} \pmod{\psi_\ell}$ ensures for all the points $(x, y) \in E[\ell]$ that

$$\begin{aligned} (x', y') &\equiv t'(x^q, y^q) && \pmod{\psi_\ell} \\ (x', y') &\equiv (x^{q^2}, y^{q^2}) + (x_{q_\ell}, y_{q_\ell}) && \pmod{\psi_\ell} \\ (x^{q^2}, y^{q^2}) + (x_{q_\ell}, y_{q_\ell}) &\equiv t'(x^q, y^q) && \pmod{\psi_\ell} \\ (x^{q^2}, y^{q^2}) \pm t'(x^q, y^q) + (x_{q_\ell}, y_{q_\ell}) &\equiv \mathcal{O} && \pmod{\psi_\ell}, \end{aligned}$$

i.e., the characteristic equation is satisfied with the given $t'$ value. The y-coordinates of both points $(x', y')$ and $(x^q_{t'}, y^q_{t'})$ are used to determine the sign of $t'$. If $y' \equiv y^q_{t'} \pmod{\psi_\ell}$, then $t_\ell = t'$ as $(x^{q^2}, y^{q^2}) + t'(x^q, y^q) + (x_{q_\ell}, y_{q_\ell}) \equiv \mathcal{O} \pmod{\psi_\ell}$ is satisfied. Otherwise, $t_\ell = -t'$, as $-(x, y) = (x, -y)$ and $(x^{q^2}, y^{q^2}) + t'(x^q, -y^q) + (x_{q_\ell}, y_{q_\ell}) \equiv \mathcal{O} \pmod{\psi_\ell}$ is satisfied.

Despite trying all the values in the range $0 < t' < (\ell - 1)/2$, equality may not be found with a slight chance. In this case, a flag variable can be used to prevent the disruption of the flow of the algorithm. The flag variable is set to 0 at the start of each iteration for different prime $\ell$ and only updated to 1 when a suitable trace $t'$ is found

with brute force search. If the flag variable is still 0 after the search, it is realised that there is no good $t'$ value that satisfies the characteristic equation, and the other case must be tried to find a good $t'$ value. The equality of the points $(x^{q^2}, y^{q^2})$ and $q_\ell(x, y)$ means that the elliptic curve point addition rule cannot be applied. A different approach must be followed to solve this problem.

The points are equal as

$$(x^{q^2}, y^{q^2}) = \pm q_\ell(x, y). \tag{4.7}$$

These two cases also result in different consequences. Considering the positive case, by substituting $(x^{q^2}, y^{q^2}) = q_\ell(x, y)$ to the equation $(x^{q^2}, y^{q^2}) + q_\ell(x, y) = t_\ell(x^q, y^q)$, the equality

$$2q_\ell(x, y) = t_\ell(x^q, y^q) \tag{4.8}$$

is obtained. Multiplying both sides with $t_\ell^2$ causes $t_\ell^2(x^{q^2}, y^{q^2}) = t_\ell^2(q_\ell(x, y))$. Since $(x^{q^2}, y^{q^2}) = \varphi^2(x, y)$, the left hand side can be arranged as $t_\ell(\varphi(t\ell(\varphi(x, y))))$. By using Equation (4.8), $t\ell(\varphi(x, y))$ can be replaced by $2q_\ell(x, y)$. Thus, $t_\ell(\varphi(t_\ell(\varphi(x, y)))) = (2q_\ell)^2(x, y)$. The equivalence $t_\ell^2 q_\ell \equiv (2q_\ell)^2 \pmod{\ell}$ indicates that $q_\ell$ is a quadratic residue modulo $\ell$. So, calculating the modular square root $w$ where $q_\ell \equiv w^2 \pmod{\ell}$ can be used to determine the trace $t_\ell$. For all the points $(x, y) \in E[\ell]$, if $q_\ell \equiv w^2 \pmod{\ell}$, then by the positive case of Equation (4.7),

$$(\varphi^2 - q_\ell)(x, y) = (\varphi - w)(\varphi + w)(x, y) = \mathcal{O}. \tag{4.9}$$

Here, checking whether $\varphi(x, y) = \pm w(x, y)$ holds is significant when ensuring the existence of $w$. Performing a scalar multiplication $(x_w, y_w) = w(x, y)$ in the quotient ring $Q_\ell$ and checking

$$\gcd(x^q - x_w, \psi_\ell)$$

would be enough to be sure about having a factorizable characteristic equation using the root $w$. If there is no common factor, then $t_\ell \equiv 0 \pmod{\ell}$. If a common factor is determined, then Equation (4.9) holds. The sign can be deduced by using the y-coordinate and checking the result of

$$\gcd(y^q - y_w/y_w, \psi_\ell).$$

If there is a common factor, then $t_\ell \equiv 2w \pmod{\ell}$, otherwise $t_\ell \equiv -2w \pmod{\ell}$.

In Equation (4.7), the negative case holds if and only if $q_\ell$ is not a quadratic residue modulo $\ell$. Here, $w$ cannot exist, and the trace $t_\ell \equiv 0 \pmod{\ell}$. Schoof's algorithm is given in Algorithm 2. In addition, Example 4.1.1 is provided to make understanding the algorithm easier. The curve in the example is more illustrative since it covers different cases in the algorithm.

**Algorithm 2:** Schoof's Algorithm

**Input :** An elliptic curve $E : y^2 = x^3 + a_4x + a_6$ over a finite field $\mathbb{F}_q$ with $q = p^n$ is a prime power where $n \geq 1$.

**Output :** The number of points on the elliptic curve $E/\mathbb{F}_q$.

Determine a set $S$ and a bound $N$ with the condition $N = \prod_{\ell \in S} \ell > 4\sqrt{q}$, where every $\ell$ is a prime starting from 2 and $\ell \neq p$.
**if** $\gcd(x^q - x, x^3 + a_4x + a_6) \neq 1$ **then**
    Set $t_2 = 0$
**else**
    Set $t_2 = 1$.
**end if**
**for** every odd prime $\ell \in S$ **do**
    Set the Boolean flag variable $fl =$ **false**.
    Compute the $\ell$-th division polynomial $\psi_\ell$.
    Compute $q_\ell$ as $q_\ell \equiv q \pmod{\ell}$ and $|q_\ell| < \ell/2$.
    Choose a quotient ring $Q_\ell = \mathbb{F}_q[x, y]/(y^2 - x^3 - a_4x - a_6, \psi_\ell)$.
    Compute $(x^q, y^q)$, $(x^{q^2}, y^{q^2})$ and $(x_{q_\ell}, y_{q_\ell}) = q_\ell(x, y)$ in $Q_\ell$.
    **if** $x^{q^2} \neq x_{q_\ell}$ **then**
        Compute $(x', y') = (x^{q^2}, y^{q^2}) \oplus (x_{q_\ell}, y_{q_\ell})$.
        **for** $1 \leq t' \leq (\ell - 1)/2$ **do**
            Compute $(x^q_{t'}, y^q_{t'}) = t'(x^q, y^q)$ in $Q_\ell$.
            **if** $x' = x^q_{t'}$ **then**
                **if** $y' = y^q_{t'}$ **then**
                    Set $t_\ell = t'$.
                **else**
                    Set $t_\ell = -t'$.
                **end if**
                Set $fl =$ **true**.
            **end if**
        **end for**
    **end if**
    **if** $fl \neq$ **true then**
        **if** $q_\ell$ is a quadratic residue mod $\ell$ **then**
            Get modular square root $w$ where $q_\ell \equiv w^2 \pmod{\ell}$.
            Compute $(x_w, y_w) = w(x, y)$ in $Q_\ell$.
            **if** $\gcd(x^q - x_w, \psi_\ell) \neq 1$ **then**
                **if** $\gcd(y^q - y_w)/y_w, \psi_\ell) \neq 1$ **then**
                    Compute $t_\ell = 2w \pmod{\ell}$.
                **else**
                    Compute $t_\ell = -2w \pmod{\ell}$.
                **end if**
            **else**
                Set $t_\ell = 0$.
            **end if**
        **else**
            Set $t_\ell = 0$.
        **end if**
    **end if**
**end for**
Combine $t_\ell$ values by using CRT to get the trace $t \equiv t_\ell \pmod{\ell}$.
**if** $|t| > 2\sqrt{q}$ **then**
    Compute $t = t - N$.
**end if**
**return** $q + 1 - t$.

**Example 4.1.1.** Consider an elliptic curve $E/\mathbb{F}_{13} : y^2 = x^3 + 5x + 7$. If one would like to use Schoof's algorithm to calculate the number of points on this curve, the first task would be determining the set $S$. After that, the procedure will be applied according to the elements of $S$. Since $4\sqrt{13} \approx 14.42$, then $N = 2 \cdot 3 \cdot 5 = 30 > 14.42$. Therefore $S = \{2, 3, 5\}$.

For $\ell = 2$, the existence of a common root between the polynomials $x^3 + 5x + 7$ and $x^{13} - x$ should be checked. So, $\gcd(x^3 + 5x + 7, x^{13} - x) = x^3 + 5x + 7 \neq 1$ which means that there is a point of order 2, $(1, 0)$, and by the existence of this point, the trace for $\ell = 2$ is $t_2 = 0$.

For $\ell = 3$, first, the flag variable $fl$ is set as $false$, and the quotient ring $Q_3 = \mathbb{F}_{13}[x, y]/(y^2 - x^3 - 5x - 7, \psi_3)$ is determined where $x$ and $y$ are the coordinates of a non-trivial point $P$ of order 3 and $\psi_3 = 3x^4 + 4x^2 + 6x + 1$. The scalar multiplication with $q_3$ is unnecessary because $q_3 \equiv 13 \pmod 3 = 1$, so $(x_{q_3}, y_{q_3}) = (x, y)$. The coordinates $(x^{13}, y^{13})$ and $(x^{13^2}, y^{13^2})$ are computed in the ring $Q_3$ as

$$\begin{aligned} x^{13} &= 4x^3 + 6x^2 + 10x + 10, & y^{13} &= y(8x^3 + 12x^2 + 7x + 7), \\ x^{13^2} &= 10x^3 + 9x + 2, & y^{13^2} &= y(8x^3 + 2x + 3). \end{aligned}$$

The x-coordinates of $(x^{13^2}, y^{13^2})$ and $(x_{q_3}, y_{q_3})$ are not equal, so $(x', y') = (x^{13^2}, y^{13^2}) + (x_{q_3}, y_{q_3}) = (x^3 + x^2 + 2x + 8, y(2x^3 + 2x^2 + 4x + 3))$. Then the possible traces in the range $1 \leq t' \leq (3 - 1)/2$ are checked. The only possible value for $t'$ is 1, so $(x^{13}_{t'}, y^{13}_{t'}) = (x^{13}, y^{13})$. None of the coordinates of $(x', y')$ and $(x^{13}_{t'}, y^{13}_{t'})$ is equal. Thus, the flag variable $fl$ did not change. Therefore, it should be checked whether $q$ is a quadratic residue modulo $\ell$. It can be seen that 13 is a square modulo 3 with the square root being $w = 1$ and $(x_w, y_w) = (x, y)$. Then, $\gcd(x^{13} - x, \psi_\ell) = x + 7$ and $\gcd((y^{13} - y)/y, \psi_\ell) = 1$ and $t_3$ is set $-2w \pmod 3 = 1$.

For $\ell = 5$, the quotient ring $Q_5 = \mathbb{F}_{13}[x, y]/(y^2 - x^3 - 5x - 7, \psi_5)$ where $x$ and $y$ are the coordinates of a non-trivial point $P$ of order 5 and $\psi_5 = 5x^{12} + 11x^{10} + 8x^9 + x^8 + 2x^7 + 10x^6 + 10x^5 + 10x^4 + 5x^2 + 5x + 10$. This time $q_\ell = q_5 = 13 \pmod 5 = 3$, and scalar multiplication is needed to compute $(x_{q_5}, y_{q_5}) = q_5(x, y)$ in the ring $Q_5$. After the multiplication,

$$\begin{aligned} x_{q_5} &= 8x^{11} + 12x^{10} + 9x^9 + 9x^8 + 3x^7 + 4x^6 + 12x^5 + 10x^4 + 7x^3 + 10x^2 + \\ &\quad 7x + 12, \\ y_{q_5} &= y(4x^{10} + 12x^9 + 5x^8 + x^7 + 3x^6 + x^5 + 2x^4 + 10x^3 + 2x^2 + 3x + 11). \end{aligned}$$

Also, the coordinates $(x^{13}, y^{13})$ and $(x^{13^2}, y^{13^2})$ are computed in the ring $Q_5$ where,

$$\begin{aligned}
x^{13} &= 3x^{11} + x^{10} + 5x^9 + 10x^8 + 11x^7 + 11x^6 + 11x^5 + 12x^3 + 12x^2 + 11x, \\
y^{13} &= y(4x^{11} + 9x^{10} + 2x^9 + 10x^8 + 6x^7 + 9x^6 + 10x^5 + 9x^4 + 2x^3 + 9x^2 + \\
&\quad 2x + 6), \\
x^{13^2} &= x^{11} + 10x^{10} + 7x^8 + 4x^6 + x^5 + 3x^3 + 3x^2 + 7x + 1, \\
y^{13^2} &= y(4x^{11} + 3x^{10} + 3x^9 + 3x^8 + 9x^7 + 8x^6 + 6x^5 + 4x^4 + 4x^3 + 10x + 2).
\end{aligned}$$

The x-coordinates of $(x^{13^2}, y^{13^2})$ and $(x_{q_5}, y_{q_5})$ are not equal, so $(x', y') = (x^{13^2}, y^{13^2})$ $+(x_{q_5}, y_{q_5})$ where,

$$\begin{aligned}
x' &= 9x^{11} + x^{10} + 12x^9 + 4x^8 + 8x^7 + 8x^6 + 3x^5 + 10x^4 + 6x^3 + 10x + 11, \\
y' &= y(5x^{11} + 9x^{10} + 6x^9 + 5x^8 + 4x^7 + x^6 + 4x^5 + 9x^4 + x^3 + x + 10).
\end{aligned}$$

The possible values for the candidate trace of Frobenius $t'$ for $\ell = 5$ need to be checked. Since $(5-1)/2 = 2$, $t'$ can only be equal to 1 or 2. So, after the scalar multiplication of $(x^{13}, y^{13})$ with $t' = 1$,

$$\begin{aligned}
x_1^{13} &= 3x^{11} + x^{10} + 5x^9 + 10x^8 + 11x^7 + 11x^6 + 11x^5 + 12x^3 + 12x^2 + 11x, \\
y_1^{13} &= y(4x^{11} + 9x^{10} + 2x^9 + 10x^8 + 6x^7 + 9x^6 + 10x^5 + 9x^4 + 2x^3 + 9x^2 + \\
&\quad 2x + 6),
\end{aligned}$$

and with $t' = 2$,

$$\begin{aligned}
x_2^{13} &= 9x^{11} + x^{10} + 12x^9 + 4x^8 + 8x^7 + 8x^6 + 3x^5 + 10x^4 + 6x^3 + 10x + 11, \\
y_2^{13} &= y(8x^{11} + 4x^{10} + 7x^9 + 8x^8 + 9x^7 + 12x^6 + 9x^5 + 4x^4 + 12x^3 + 12x + 3).
\end{aligned}$$

In the end, $x' = x_2^{13}$, but $y' \neq y_2^{13}$. The flag variable $fl$ is set as $true$, and there is no need to do any other operation. Therefore, the trace $t_5 = -2$.

All the collected traces form the congruences

$$\begin{aligned}
t &\equiv t_2 = 0 \pmod 2, \\
t &\equiv t_3 = 1 \pmod 3, \\
t &\equiv t_5 = -2 \pmod 5,
\end{aligned}$$

and by using the CRT, the trace of Frobenius $t \pmod N$ is obtained as $t = 28$. Since $|28| > 2\sqrt{13} \approx 7.21$, $t = t - N = 28 - 30 = -2$. Finally, we find the number of points on $E/\mathbb{F}_{13}$ as $q + 1 - t = 13 + 1 - (-2) = 16$.

## 4.2. SCHOOF-ELKIES-ATKIN ALGORITHM

### 4.2.1 ATKIN'S CLASSIFICATION

Due to the advancements in both technology and cryptography, the effectiveness of the usage of Schoof's algorithm for point counting on elliptic curves was starting to decrease. Atkin (1988, 1991) and Elkies (1991) began to investigate the characteristic polynomial of Frobenius endomorphism

$$\rho(\varphi) = \varphi^2 - t_\ell \varphi + p_\ell \tag{4.10}$$

built from Equation (4.3) defined in Section 4.1.2, they found that the splitting of $\rho(\varphi)$ can lead to possible enhancements in terms of efficiency in Schoof's algorithm. Later, this idea turned into an algorithm called SEA. The underlying thought of the SEA algorithm is the classification of prime numbers according to the splitting of the related modular polynomial. This classification is first proposed by (Atkin, 1991). The following definition gives the method for the classification of prime numbers.

**Definition 4.2.1.** Let $\Delta = t_\ell^2 - 4p_\ell$ be the discriminant of the characteristic polynomial of the Frobenius endomorphism $\rho(\varphi)$ for an elliptic curve $E/\mathbb{F}_p$ where $\ell \neq p$. If the discriminant $\Delta$ is a square modulo $\ell$, then $\rho(\varphi)$ has roots in $\mathbb{F}_\ell$, and $\ell$ is called an Elkies prime; otherwise, it is called an Atkin prime.

Definition 4.2.1 suggests that to detect whether a prime is an Atkin or Elkies prime, the employment of Frobenius endomorphism is needed. However, due to the uncertainty of the value of trace $t_\ell$, such endomorphism cannot be used directly. This problem is overcome by using modular polynomials. The theorem below provides a solution for this problem and is the fundamental building block of the SEA algorithm.

**Theorem 4.2.1** (Atkin)**.** *Let $E/\mathbb{F}_p$ be an ordinary elliptic curve with j-invariant $j \neq 0$ and $j \neq 1728$, and let $\bar{\Phi}_\ell(x, j) = f_1 f_2 \dots f_n$ be the factorization of a modular polynomial $\Phi_\ell(x, j) \in \mathbb{F}_p[x]$ where every $f$ is an irreducible polynomial. Also, let $r$ be the order of the Frobenius endomorphism $\varphi$ in the projective general linear group* $\mathrm{PGL}_2(\mathbb{F}_\ell)$ *which acts on the torsion group $E[\ell]$. Then the following cases occur according to the degrees of the polynomials $f_1, f_2, \dots, f_n$ :*

1. $(1, \ell)$ *or* $(1, 1, \dots, 1)$ : *In both cases the discriminant* $\Delta = t_\ell^2 - 4p_\ell \equiv 0 \pmod{\ell}$. Then for the first case, $r = 1$, and for the second case, $r = \ell$.
2. $(1, 1, r, r, \dots, r)$ : The discriminant $\Delta = t_\ell^2 - 4p_\ell$ is a square modulo $\ell$ and $r \,|\, \ell - 1$ and $\varphi$ acts on E$[\ell]$ as $\begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}$, where $\lambda, \mu \in \mathbb{F}_\ell^*$ are the roots of $\rho$.

3. $(r, r, \ldots, r), r > 1$ : The discriminant $\Delta = t_\ell^2 - 4p_\ell$ is not a square modulo $\ell$, $r \mid \ell + 1$. Therefore, $\varphi$ has an irreducible characteristic polynomial over $\mathbb{F}_\ell$.

*For all the cases above, the trace of Frobenius $t$ ensures*

$$t^2 \equiv p(\zeta + \zeta^{-1})^2 \pmod{\ell}, \tag{4.11}$$

*where $\zeta \in \bar{\mathbb{F}}_\ell$ is a primitive $r$-th root of unity, and the equation*

$$(-1)^s = \left(\frac{p}{\ell}\right) \tag{4.12}$$

*is satisfied, with $s$ being the number of irreducible factors.*

*Proof.* See (Schoof, 1995). □

With the help of Theorem 4.2.1, classification for any prime $\ell \neq p$ can be done. If the degrees for the splitting of $\Phi_\ell(x, j)$ correspond to the first or second case, a root exists in $\mathbb{F}_p$ of the equation $\Phi_\ell(x, j) = 0$, and $\ell$ is an Elkies prime. Otherwise, the degrees for splitting suit the third case, and there is no root of the equation $\Phi_\ell(x, j) = 0$. Therefore, $\ell$ is an Atkin prime. Instead of factorizing $\Phi_\ell(x, j)$ and checking the degrees of each factor, computing

$$g(x) = \gcd(\Phi_\ell(x, j), x^p - x) \tag{4.13}$$

would be sufficient to detect the existence of a root. Here, if the degree of $g(x)$ is 0, i.e., $g(x)$ is constant, $\ell$ is an Atkin prime. Apart from that, the degree of $g(x)$ can only be 1,2 or $\ell$. In one of these cases, a root is found, and $\ell$ is an Elkies prime.

#### 4.2.1.1 Procedure for Atkin Primes

After the classification process, if the prime $\ell$ is determined as an Atkin prime, the method indicated in this section should be employed to complete the Atkin part of the SEA algorithm. The stated approach follows (Schoof, 1995) and (Lercier et al., 2005).

As stated in Section 4.2.1, the classification can be done by computing $g(x) = \gcd(\Phi_\ell(x, j), x^p - x)$ and checking the degree of $g(x)$. Since $\ell$ is an Atkin prime, the third case of Theorem 4.2.1 applies and $r \mid \ell + 1$. The exact value of $r$ must be found to detect the trace $t_\ell$. It can be computed similarly with the computation of $g(x)$ as

$$f(x) = \gcd(\Phi_\ell(x, j), x^{p^i} - x), \tag{4.14}$$

for $i > 1$. Herein, each $i$ that divides $\ell + 1$ is checked until $f(x) = \Phi_\ell(x, j)$. The choice of $i$ can be narrowed down using Equation (4.12) as $(-1)^{(\ell+1)/i} = \left(\frac{p}{\ell}\right)$.

Additionally, the calculation of $x^{p^i}$ can be achieved efficiently by performing modular exponentiation with $\Phi_\ell(x, j)$. The efficiency of the order computation can be further improved by only performing exponentiation for the factors of $\ell + 1$. Since $p^i = p^s p^{i-s}$ for some integer $s \leq i$, exponentiation can be done only up to $i - s$ and the result can be used in the $\gcd$ computation. If $\gcd$ is not equal to $\Phi_\ell(x, j)$, then the value of $s$ is updated to $i$, and the procedure is tried with a new factor of $\ell + 1$. To sum up, the order of the Frobenius endomorphism $r$ can be efficiently computed with Algorithm 3.

**Algorithm 3:** FindOrderFrobenius

**Input :** An Atkin prime $\ell$ and the $\ell$-th modular polynomial $\Phi_\ell(x, j) \in \mathbb{F}_p[x]$.
**Output :** Order $r$ of the Frobenius endomorphism $\pi \in \mathrm{PGL}_2(\mathbb{F}_\ell)$.

Set $f(x) = x, s = 0$.
**for** $i = 1$ **to** $\ell + 1$ **do**
  **if** $r \mid \ell + 1$ and $(-1)^{(\ell+1)/i} = \left(\frac{p}{\ell}\right)$ **then**
    **for** $k = 1$ **to** $i - s$ **do**
      Compute $f(x) = f(x)^p \bmod \Phi_\ell(x, j)$.
    **end for**
    **if** $\gcd(f(x) - x, \Phi_\ell(x, j)) = \Phi_\ell(x, j)$ **then**
      **return** $i$.
    **else**
      Set $s = i$.
    **end if**
  **end if**
**end for**
**return** $-1$.

With the discovery of the order of Frobenius endomorphism, Equation (4.11), one of the outputs of Theorem 4.2.1, can be utilized. If any primitive $r$-th root of unity $\zeta$ that belongs to the algebraic closure of $\mathbb{F}_\ell$ can be found, then the trace of Frobenius $t$ can be reached using this equation.

It is known that $r \mid \ell + 1$ since $\ell$ is an Atkin prime and the third case of the Theorem 4.2.1 applies. So, for an integer $n$, $\ell + 1 = nr$ and $\zeta^{\ell^2-1} = \zeta^{(\ell-1)(\ell+1)} = \zeta^{(\ell-1)nr}$. Since $\zeta^r = 1$, then $\zeta^{(\ell-1)nr} = 1$ and $\zeta \in \mathbb{F}_{\ell^2}$. Therefore, one can get equality by trying different primitive $r$-th roots of unities in $\mathbb{F}_{\ell^2}$. The number of possible primitive $r$-th roots of unities can be found with Euler's totient function of $r$, $\phi_{Eul}(r)$, which gives the number of relatively prime numbers with $r$ in the range between 1 and $r$. Also, symmetry exists in the roots of unities, so the total number of values for $\zeta + \zeta^{-1}$ is $\phi_{Eul}(r)/2$, which is the minimum number of iterations to be performed for the collection of candidate traces. At this point, $\zeta \in \mathbb{F}_{\ell^2}$, and this causes the characteristic polynomial to split into two roots as $\rho(\varphi) = (\varphi - \lambda)(\varphi - \mu)$, where $\lambda, \mu \in \mathbb{F}_\ell^2 \setminus \mathbb{F}_\ell$. Furthermore, from Equation (4.10), $\lambda\mu = p_\ell$ and $\lambda + \mu = t_\ell$. So, $\lambda = a_1 + a_2\sqrt{d}$ and $\mu = a_1 - a_2\sqrt{d}$ where $a_1, a_2, d \in \mathbb{F}_\ell$ with $d$ being a non-square in $\mathbb{F}_\ell$. Such $\lambda$ and $\mu$ can be defined

because of the isomorphism between the extension field $\mathbb{F}_{\ell^2}$ and the multiplicative group $\mathbb{F}_\ell[\sqrt{d}]$. Note that $\gamma = \lambda/\mu$ is a primitive $r$-th root of unity, and it can be written as $\gamma = g_1 + g_2\sqrt{d}$. The variables $a_1$ and $a_2$ are unknown, but $g_1$ and $g_2$ can be found as follows.

$$\gamma = g_1 + g_2\sqrt{d} = \frac{\lambda}{\mu} = \frac{\lambda^2}{\lambda\mu} = \frac{a_1^2 + 2a_1a_2\sqrt{d} + da_2^2}{p_\ell} = \frac{a_1^2 + da_2^2}{p_\ell} + \frac{2a_1a_2}{p_\ell}\sqrt{d}. \quad (4.15)$$

So, $g_1 = \dfrac{a_1^2 + da_2^2}{p_\ell}$. Then, $p_\ell g_1 = a_1^2 + da_2^2$ and by the usage of $p_\ell = a_1^2 - da_2^2$, consequently $a_1^2 = p_\ell(g_1 + 1)/2$. For every different primitive $r$-th root of unity, the equation gets in the form $a_{i_1}^2 = p_\ell(g_{i_1} + 1)/2$. Iterating through $\phi_{Eul}(r)/2$ different roots will be sufficient. The cases for $a_1^2$ is not a square in $\mathbb{F}_\ell$ are passed, and the trace $t \equiv \lambda + \mu = 2a_1 \pmod{\ell}$ is calculated when it is a square in $\mathbb{F}_\ell$. Since it is impossible to know the correct sign of the square root, both $2a_1$ and $-2a_1$ are collected for the candidate traces. For each Atkin prime, candidate traces are stored in a set $A_{t_\ell}$. Every set is also stored in a set $A_t$ as pairs with corresponding prime $\ell$ as $(A_{t_\ell}, \ell)$ to know which set belongs to which Atkin prime in the information combination part of the algorithm. The algorithm of Atkin's procedure is given in Algorithm 4.

After a certain point, it may not make sense to perform Atkin's procedure and store candidate traces even if the prime $\ell$ is an Atkin prime. As the primes get larger, the number of stored candidate traces will increase significantly. Therefore, combining the information obtained from Atkin and Elkies procedures and recovering the correct trace $t$ will be more challenging. For instance, if the number of candidate trace sets collected from Atkin's method is high while the number of exact traces collected from the Elkies' procedure is low, the probability of finding a match with a random point $P$ selected from $E(\mathbb{F}_p)$ will be low. Thus, applications of Atkin's procedures can be stopped at a certain point to reduce the time taken in the information gathering part, which has exponential time complexity. Any Atkin prime $\ell$ after this point is ignored, and the algorithm continues with the following prime number. The implementations related to this thesis adjust the total information gathered from Atkin procedures according to the greatness of $p$. The details of this adjustment are given in Section 4.4. There are also variants of the SEA algorithm that differ in frequency of use of the Atkin or Elkies procedures. Information about these variants can be found in (Kok, 2013).

### 4.2.2 ELKIES' IMPROVEMENTS

In the case that the prime $\ell$ is decided as an Elkies prime, the discriminant $\Delta = t_\ell^2 - 4p_\ell$ is a square modulo $\ell$, according to Theorem 4.2.1. This case leads to the split of the characteristic polynomial of Frobenius endomorphism $\rho(\varphi)$ in $\mathbb{F}_\ell$ as $\rho(\varphi) = (\varphi -$

**Algorithm 4:** AtkinAlgorithm

**Input :** An Atkin prime $\ell$ and the $\ell$-th modular polynomial $\Phi_\ell(x, j) \in \mathbb{F}[x]$.
**Output :** Set of candidate traces and related prime sets $A_t$.

Compute $r = \texttt{FindOrderFrobenius}(\ell, \Phi_\ell(x, j))$.
Set $d = 0$.
**for** $i = 1$ **to** $\ell - 1$ **do**
    **if** $\left(\frac{i}{\ell}\right) = -1$ **then**
        Set $d = i$.
        **break**.
    **end if**
**end for**
Create the multiplicative group $\mathbb{F}_\ell[\sqrt{d}]$.
Get the $r$-th root of unity $g$ in $\mathbb{F}_\ell[\sqrt{d}]$.
Set $S = \{\}$.
**for** $i = 1$ **to** $r - 1$ **do**
    **if** $\gcd(i, r) = 1$ **then**
        Set $S_i = \{g^i\}$.
        Set $S = S \cup S_i$.
    **end if**
**end for**
Set $A_{t_\ell} = \{\}$.
**for** $i = 1$ **to** $\phi_{Eul}(r)/2$ **do**
    Get $g_{1_i}$ from $S_i = g_{1_i} + g_{2_i}\sqrt{d}$.
    Compute $a = p(g_{1_i} + 1)/2 \pmod{\ell}$.
    **if** $\left(\frac{a}{\ell}\right) = 1$ **then**
        Compute $b = \sqrt{a} \pmod{\ell}$.
        Set $A_{t_\ell} = A_{t_\ell} \cup \{2b, -2b\}$.
    **end if**
**end for**
Set $A_{t_\ell} = \texttt{Sort}(A_{t_\ell})$.
Set $A_t = A_t \cup \{A_{t_\ell}, \ell\}$.
**return** $A_t$.

$\lambda)(\varphi - \mu) = 0$ where $\lambda, \mu \in \mathbb{F}_\ell$. This split is, in fact, the main idea of the procedure for Elkies primes which follows (Elkies, 1991). From here, determination of the trace $t$ can be done as

$$t \equiv \lambda + \mu = \lambda + \frac{p_\ell}{\lambda} \pmod{\ell}. \tag{4.16}$$

The correct value for $\lambda$ can be found by trying the values in the range $1 \leq \lambda \leq \ell - 1$ and checking if the equality

$$(x^p, y^p) = [\lambda](x, y) \tag{4.17}$$

holds for a point $P = (x, y) \in E[\ell]$. The values of $\lambda$ and $\mu$ in Equation (4.16) can be equal. For each case, there exist subgroups of $E[\ell]$, which are stable under the action of $\rho(\varphi)$. The details for these cases can be found in (Blake et al., 1999, Section VII.3).

The benefit of working in $\mathbb{F}_\ell$ is the decrease of degrees of division polynomials used at the computations in the quotient ring. Instead of using the $\ell$-th division polynomial $\psi_\ell$ with degree $(\ell^2 - 1)/2$ as in the usual case of Schoof's Algorithm, a factor of $\psi_\ell$ whose degree is $(\ell - 1)/2$ can be used with the knowledge of being $\ell$ an Elkies prime. By employing the quotient ring created with a factor of $\psi_\ell$, the trace search operation can be completed much faster. The real work to be done here is finding the mentioned factor. This factor is also called *kernel polynomial*. The roots of this polynomial come from the kernel of an isogeny of degree $\ell$ between the curve $E/\mathbb{F}_p$ and an isogenous curve to this curve. The kernel of this isogeny is denoted as $C_\ell$, and the kernel polynomial can be created as

$$F_\ell(x) = \prod_{\pm P \in C_\ell \setminus \{\mathcal{O}\}} (x - \mathrm{x}(P)) \tag{4.18}$$

where $\mathrm{x}(P)$ is the $x$-coordinate of the point $P$, an element of the kernel. It is sufficient to take only one point for each pair $\pm P$ since the $x$-coordinate is sign-independent, leading to the mentioned decrease in the degree of $F_\ell$. Section 4.2.2.1 explains how to find such a factor and recover the correct trace value using this factor.

#### 4.2.2.1 Procedure for Elkies Primes

As described in Section 4.2.2, an isogeny needs to be determined to find a factor of the $\ell$-th division polynomial with a degree of $(\ell - 1)/2$. Frobenius endomorphism $\varphi$ acts on the kernel of this isogeny, namely $C_\ell$. This acting applies as multiplication by $\lambda$ or $\mu$ on $C_\ell$, supporting the creation of the polynomial in Equation (4.18). The critical path to follow to get the desired kernel polynomial, $F_\ell$, is to determine an isogenous curve $E/C_\ell$ to $E/F_p$ and get the coefficients of each term of $F_\ell$ by using points of $C_\ell$. Then, the sum of the $x$-coordinates of the points in $C_\ell$ gives the first coefficient of $F_\ell$, and with the help of it, the other coefficients are constructed. The method described after this point follows (Schoof, 1995), based on (Elkies, 1991). More detailed information

can be obtained from these sources.

**Definition 4.2.2.** Let $E/F_p$ be an ordinary elliptic curve whose $j$-invariant $\neq 0$ or $1728$. As stated in Section 2.7, $E$ can be associated with $E/\mathbb{C}$ via an isogeny where $\mathbb{C}$ is a complex field. Let $q = e^{2\pi i\tau} \in \mathbb{C}$, and the $j$-invariant of $E/\mathbb{C}$, $j(q)$, be as in Equation (2.40). Then, the following expressions are valid for Laurent series $\mathbb{Z}[[q]]$:

$$E_2(q) = 1 - 24\sum_{n=1}^{\infty}\frac{nq^n}{1-q^n}, \tag{4.19}$$

$$E_4(q) = 1 + 240\sum_{n=1}^{\infty}\frac{n^3q^n}{1-q^n}, \tag{4.20}$$

$$E_6(q) = 1 - 504\sum_{n=1}^{\infty}\frac{n^5q^n}{1-q^n}, \tag{4.21}$$

where the discriminant

$$\Delta(q) = q\prod_{n=1}^{\infty}(1-q^n)^{24} = \frac{E_4(q)^3 - E_6(q)^2}{1728}, \tag{4.22}$$

and

$$E_4(q)^3 = j(q)\Delta(q), \qquad E_6(q)^2 = (j(q)-1728)\Delta(q). \tag{4.23}$$

Let $f(q) = \sum_n a_nq^n$ be any Laurent series and $f'(q) = \sum_n na_nq^n$ be $q$ times the derivative of it. Then, the following inference can be reached.

**Proposition 4.2.1.** *The following equalities can be established where every variable is defined in* $\mathbb{Z}[[q]]$.

$$\frac{j'(q)}{j(q)} = -\frac{E_6(q)}{E_4(q)}, \tag{4.24}$$

$$\frac{j'(q)}{j(q)-1728} = -\frac{E_4^2(q)}{E_6(q)}, \tag{4.25}$$

$$\frac{j''(q)}{j'(q)} = -\frac{E_2(q)}{6} - \frac{E_4^2(q)}{2E_6(q)} - \frac{2E_6(q)}{3E_4(q)}. \tag{4.26}$$

*Proof.* See (Schoof, 1995, p. 244). □

Next, the power series $\mathbb{Z}[\zeta, 1/(\zeta(1-\zeta))][[q]]$ should be defined to construct the following proposition.

**Proposition 4.2.2.** *The following equalities hold in* $\mathbb{Z}[\zeta, 1/(\zeta(1-\zeta))][[q]]$ :

$$y^2 = x^3 - \frac{E_4(q)}{48}x + \frac{E_6(q)}{864} \tag{4.27}$$

*where*

$$x = x(\zeta; q) = \frac{1}{12} - 2\sum_{n=1}^{\infty} \frac{q^n}{(1-q^n)^2} + \sum_{n\in\mathbb{Z}} \frac{\zeta q^n}{(1-\zeta q^n)^2},$$
$$y = y(\zeta; q) = \frac{1}{2}\sum_{n\in\mathbb{Z}} \frac{\zeta q^n(1+\zeta q^n)}{(1-\zeta q^n)^3},$$

*and*

$$\sum_{\zeta\in\mu_\ell} x(\zeta; q) = \frac{\ell}{12}(E_2(q) - \ell E_2(q^\ell)) \tag{4.28}$$

*where* $\mu_\ell$ *denotes the set of complex* $\ell$*-th roots of unity.*

*Proof.* Proof of the proposition is given in (Schoof, 1995, p. 245). □

This equation takes the shape of the short Weierstrass form, and the coefficients can be reduced modulo a prime ideal $\mathfrak{B}$ with the residue field $\mathbb{F}_p$, which means having an equation defined in $\mathbb{F}_p$. Also, in $\mathbb{F}_p$, the sum in Equation (4.28) stands for the sum of $x$-coordinates of the points in the desired kernel of the isogeny. This sum, denoted by $p_1$, is the coefficient of the term with the degree $(\ell-3)/2$ in the kernel polynomial. The coefficients of other terms can be found using $p_1$ and the involvement of modular polynomials. The procedure given here applies to classical modular polynomials. The different types of modular polynomials can also be used in the kernel polynomial determination phase. Blake et al. (1999) gave a procedure using Müller's modular polynomials. Also, Stankovic (2017) used Atkin modular polynomials efficiently in the kernel polynomial calculation phase. Using Atkin modular polynomials increases efficiency since their coefficients are much shorter than those of the classical modular polynomials. That is why these polynomials are more commonly used in implementations. Among the implementations related to this thesis, the fastest one utilizes Atkin modular polynomials, thanks to the method given in (Stankovic, 2017).

**Theorem 4.2.2.** *Let $j$ denote the Laurent series $j(q)$ and $\tilde{j}$ denote the Laurent series $j(q^\ell)$. For the $\ell$-th classical modular polynomial $\Phi_\ell(x,y) \in \mathbb{Z}[x,y]$, the following equalities of the Laurent series hold:*

$$\Phi_\ell(j,\tilde{j}) = 0, \tag{4.29}$$

$$\tilde{j}' = -\frac{j'\Phi_{\ell_x}(j,\tilde{j})}{\ell\Phi_{\ell_y}(j,\tilde{j})}, \tag{4.30}$$

*where $\Phi_{\ell_x}$ and $\Phi_{\ell_y}$ denote the partial derivatives $\dfrac{\partial\Phi_\ell}{\partial x}$ and $\dfrac{\partial\Phi_\ell}{\partial y}$, respectively,*

$$\frac{j''}{j'} - \ell\frac{\tilde{j}''}{\tilde{j}'} = -\frac{j'^2\Phi_{\ell_{xx}}(j,\tilde{j}) + 2\ell j'\tilde{j}'\Phi_{\ell_{xy}}(j,\tilde{j}) + \ell^2\tilde{j}'^2\Phi_{\ell_{yy}}(j,\tilde{j})}{j'\Phi_{\ell_x}(j,\tilde{j})}, \tag{4.31}$$

*where $\Phi_{\ell_{xx}}$,$\Phi_{\ell_{xy}}$ and $\Phi_{\ell_{yy}}$ denote the partial derivatives $\dfrac{\partial^2\Phi_\ell}{\partial x^2}$,$\dfrac{\partial^2\Phi_\ell}{\partial x\partial y}$ and $\dfrac{\partial^2\Phi_\ell}{\partial y^2}$, respectively.*

*Proof.* See (Schoof, 1995, p. 246). □

One condition to be aware of here is that the partial derivatives can vanish at the condition of $\Phi_{\ell_x}(j,\tilde{j}) = \Phi_{\ell_y}(j,\tilde{j}) = 0$ if $(j,\tilde{j})$ is a singular point of $\Phi_\ell(x,y)$ over $\mathbb{F}_p$. However, the probability of encountering such a situation is minimal due to the very large $p$. Otherwise, the used curve is discarded, and another random curve is chosen.

As stated earlier, all the arithmetic related to Equation (4.27) can be done in $\mathbb{F}_p$ under the reduction of a prime ideal $\mathfrak{B}$. Equation (4.30) and Equation (4.31) are defined over Laurent series owing to $j(q)$ and $j(q^\ell)$. If all the computation is to be done in $\mathbb{F}_p$, these variables must also be defined in $\mathbb{F}_p$. The $\mathbb{F}_p$ analogue of $j(q)$ already exists as the $j$-invariant of $E/\mathbb{F}_p$, but an analogue for $j(q^\ell)$ must be found. It can be found by computing

$$\gcd(\Phi_\ell(x,j), x^p - x),$$

which is the same operation done in Atkin's classification. Now the result of this operation exposes whether there are roots of the $\ell$-th modular polynomial in $\mathbb{F}_p$ or not. Here, the situation complies with the one specified at the end of the Section 4.2.1 and one of the two roots can be decided as the $\tilde{j}$ in $\mathbb{F}_p$. This root is the $j$-invariant of the isogenous curve $E/C_\ell$, according to the proposition given in (Schoof, 1995, p. 236). Then, with the use of Equation (4.30), the equation of the isogenous curve in short Weierstrass form appears as

$$\tilde{E} = E/C_\ell : y^2 = x^3 + \tilde{a}_4 x + \tilde{a}_6 \tag{4.32}$$

where

$$\tilde{a}_4 = -\frac{\tilde{j}'^2}{48\tilde{j}(\tilde{j}-1728)}, \qquad \tilde{a}_6 = -\frac{\tilde{j}'^3}{864\tilde{j}^2(\tilde{j}-1728)}. \tag{4.33}$$

Also, from the equalities in Proposition 4.2.1,

$$\tilde{E}_4 \equiv -48\tilde{a}_4 \,(\mathrm{mod}\ \mathfrak{B}), \qquad \tilde{E}_6 \equiv 864\tilde{a}_6 \,(\mathrm{mod}\ \mathfrak{B}) \tag{4.34}$$

hold, with the property $j(q^\ell) \equiv \tilde{j} \,(\mathrm{mod}\ \mathfrak{B})$.

As in Equation (4.28), in $\mathbb{F}_p$, the sum corresponds to the sum of $x$-coordinates of the points in the kernel. Reducing this sum with a prime ideal $\mathfrak{B}$ and utilizing Equation (4.31), the sum of $x$-coordinates of the points in the kernel is formed as

$$p_1 = \frac{\ell}{2}J + \frac{\ell}{4}\left(\frac{E_4^2}{E_6} - \frac{\ell\tilde{E}_4^2}{\tilde{E}_6}\right) + \frac{\ell}{3}\left(\frac{E_6}{E_4} - \frac{\ell\tilde{E}_6}{\tilde{E}_4}\right), \tag{4.35}$$

where $J = \dfrac{j''}{j'} - \ell\dfrac{\tilde{j}''}{\tilde{j}'}$ as given in Equation (4.31) and $E_4$ and $E_6$ are the analogues of Equation (4.20) and Equation (4.21) in $\mathbb{F}_p$ with the expressions

$$E_4 \equiv -48a_4 \,(\mathrm{mod}\ \mathfrak{B}), \qquad E_6 \equiv 864a_6 \,(\mathrm{mod}\ \mathfrak{B}). \tag{4.36}$$

The elliptic curve $E/\mathbb{F}_p$ is isomorphic to an elliptic curve $E/\mathbb{C}$ defined over a complex field $\mathbb{C}$ as declared in Section 2.7, and it can be associated with a lattice $\Lambda = \omega_1\mathbb{Z} + \omega_2\mathbb{Z}$. The isogeny discussed here is an $\ell$-isogeny modulo the prime ideal $\mathfrak{B}$, defined as

$$\mathbb{C}/(\omega_1\mathbb{Z} + \omega_2\mathbb{Z}) \to \mathbb{C}/(\omega_1\mathbb{Z} + \ell\omega_2\mathbb{Z})$$

with the map $z \mapsto \ell z$. Schoof (1995) states that working with a different isogeny with the same kernel as the above would be better. This isogeny facilitates the computations to be done and is defined as

$$\mathbb{C}/(\omega_1\mathbb{Z} + \omega_2\mathbb{Z}) \to \mathbb{C}/(\frac{\omega_1}{\ell}\mathbb{Z} + \omega_2\mathbb{Z})$$

with the map $z \mapsto z$ and the isogenous curve equation

$$\hat{E} : y^2 = x^3 + \hat{a}_4 x + \hat{a}_6. \tag{4.37}$$

where $\hat{a}_4 = \ell^4\tilde{a}_4$ and $\hat{a}_6 = \ell^6\tilde{a}_6$.

The coefficients of the remaining terms of the kernel polynomial can be found using the relationship between the polynomial and the Weierstrass elliptic function. Rewriting

the Weierstrass elliptic function $\wp$ in Equation (2.36) as

$$\wp(z) = \frac{1}{z^2} + \sum_{k=1}^{\infty} c_k z^{2k}, \tag{4.38}$$

where coefficients $c_k \geq 1$ are computed with the recursion

$$c_1 = -\frac{a_4}{5}, \qquad c_2 = -\frac{a_6}{7}, \tag{4.39}$$

$$c_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} c_j c_{k-1-j}, \qquad \text{where } k \geq 3. \tag{4.40}$$

Also, for the lattice $\frac{\omega_1}{\ell}\mathbb{Z} + \omega_2\mathbb{Z}$, another Weierstrass elliptic function $\hat{\wp}(z)$ can be defined as

$$\hat{\wp}(z) = \frac{1}{z^2} + \sum_{k=1}^{\infty} \hat{c}_k z^{2k}, \tag{4.41}$$

where the coefficients $\hat{c}_k \geq 1$ are computed with the recursion

$$\hat{c}_1 = -\frac{\hat{a}_4}{5}, \qquad \hat{c}_2 = -\frac{\hat{a}_6}{7}, \tag{4.42}$$

$$\hat{c}_k = \frac{3}{(k-2)(2k+3)} \sum_{j=1}^{k-2} \hat{c}_j \hat{c}_{k-1-j}, \qquad \text{where } k \geq 3. \tag{4.43}$$

Based on all this information, the following theorem helps find the remaining coefficients of the kernel polynomial $F_\ell(x)$. Note that the naive linear algebra method given in this theorem finds the kernel polynomial with the complexity of $O(\ell^3)$. More efficient methods are used to reach the kernel polynomial, such as the method given in (Galbraith, 2012), provided by Andrew Sutherland. The running time complexity of this method is $O(\ell^2)$, which is used in the implementations presented in this thesis.

**Theorem 4.2.3.** *Let $F_\ell(x)$ be a polynomial whose roots are the $x$-coordinates of the points in the kernel of an $\ell$-isogeny $\phi_\ell : E \to \hat{E}$. Then,*

$$F_\ell(\wp(z)) = z^{1-\ell} \exp\left( -\frac{p_1}{2} z^2 - \sum_{k=1}^{\infty} \frac{\hat{c}_k - \ell c_k}{(2k+1)(2k+2)} z^{2k+2} \right). \tag{4.44}$$

*Proof.* See (Schoof, 1995, p. 246). □

Theorem 4.2.3 gives the expression of $F_\ell(\wp(z))$, which includes the Laurent series. Since every variable has a counterpart in $\mathbb{F}_p$, expanding the series up to $d = (\ell - 1)/2$ will be enough. Then, $F_\ell(x)$ can be reached with the method given in (Blake et al.,

1999). In the solution, the coefficients are found with the recursion

$$a_{d-i} = [A(w)]_i - \sum_{k=1}^{i} \left( \sum_{j=0}^{k} \binom{d-i+k}{k-j} [C(w)^{k-j}]_j \right) a_{d-i+k}, \tag{4.45}$$

for $1 \leq i \leq d$, where

$$\begin{aligned} w =& z^2, \\ C(w) =& \sum_{k=1}^{d} c_k w^k, \\ A(w) =& \exp\left( -\frac{p_1}{2} w - \sum_{k=1}^{d} \frac{\hat{c}_k - \ell c_k}{(2k+1)(2k+2)} w^{k+1} \right). \end{aligned}$$

In addition, in the beginning, $a_d = 1$ and $[C(w)]_j^{k-j}$ denotes the $k-j$-th power of the coefficient of $w^j$ in an arbitrary power series $C(w)$. If the kernel polynomial $F_\ell(x)$ is defined as $F_\ell(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_0$, then the first three coefficients after $a_d$ appear as

$$\begin{aligned} a_{d-1} =& -\frac{p_1}{2}, \\ a_{d-2} =& \frac{p_1^2}{8} - \frac{\hat{c}_1 - \ell c_1}{12} - \frac{\ell - 1}{2} c_1, \\ a_{d-3} =& -\frac{p_1^3}{48} - \frac{\hat{c}_2 - \ell c_2}{30} + p_1 \frac{\hat{c}_1 - \ell c_1}{24} - \frac{\ell - 1}{2} c_2 + \frac{\ell - 3}{4} c_1 p_1. \end{aligned}$$

The steps for finding the kernel polynomial are given in Algorithm 5. After obtaining the kernel polynomial, the detection of the trace will be an easy task to do. Instead of working under the quotient ring $Q_\ell = \mathbb{F}_p[x,y]/(y^2 - x^3 - a_4 x - a_6, \psi_\ell)$, a new quotient ring can be created by using the kernel polynomial $F_\ell(x)$ as $Q_\ell = \mathbb{F}_p[x,y]/(y^2 - x^3 - a_4 x - a_6, F_\ell(x))$ and then, the approach in Equation (4.17) can be followed. The computations under this ring will be more efficient because of the low degree kernel polynomial usage. In Equation (4.17), scalar multiplication can be performed with repeated point additions to speed up the equality checks. Algorithm 6 handles every case encountered when adding two affine points on an elliptic curve $E/\mathbb{F}_p$ in the short Weierstrass form. When equality is found between $x^p$ and ${}_{\mathrm{X}}([\lambda](x,y))$, $y$-coordinates are checked. If $y^p$ and ${}_{\mathrm{Y}}([\lambda](x,y))$ are equal, the needed value to calculate the trace is $\lambda$; otherwise, it is $-\lambda$. Then, Equation (4.16) is used to get the trace $t \pmod{\ell}$ which is stored in an array with the corresponding prime $\ell$. After concluding the procedure for the Elkies prime $\ell$, the algorithm continues with the following prime number. Application of an Elkies procedure takes around $O(\log^6 p)$ bit operations and employing fast arithmetic in the computations further decreases this to $\tilde{O}(\log^4 p)$, which constitutes a significant

**Algorithm 5:** FindKernelPolynomial

**Input :** An Elkies prime $\ell$ and an elliptic curve $E/\mathbb{F}_p$.

**Output :** A factor of the $\ell$-th division polynomial whose degree is $d = (\ell - 1)/2$.

Compute the $j$-invariant of $E/\mathbb{F}_p$ as $j = 1728\dfrac{4a_4^3}{a_4^3 + 27a_6^2} \in \mathbb{F}_p$.

Compute $E_4 \equiv -48a_4 \pmod{\mathfrak{B}}$ and $E_6 \equiv 864a_6 \pmod{\mathfrak{B}}$.

Compute $j' \equiv j'(q) \pmod{\mathfrak{B}} \equiv -\dfrac{jE_4}{E_6} \pmod{\mathfrak{B}}$.

Set $\tilde{j} \in \mathbb{F}_p$ as a root of $\Phi_\ell(x, j)$.

Compute $\tilde{j}' = -\dfrac{j'\Phi_{\ell_x}(j, \tilde{j})}{\ell\Phi_{\ell_y}(j, \tilde{j})}$.

Compute $\tilde{a}_4 = -\dfrac{\tilde{j}'^2}{48\tilde{j}(\tilde{j} - 1728)}$ and $\tilde{a}_6 = -\dfrac{\tilde{j}'^3}{864\tilde{j}^2(\tilde{j} - 1728)}$.

Compute $\tilde{E}_4 \equiv -48\tilde{a}_4 \pmod{\mathfrak{B}}$ and $\tilde{E}_6 \equiv 864\tilde{a}_6 \pmod{\mathfrak{B}}$.

Compute $J = \dfrac{j''}{j'} - \ell\dfrac{\tilde{j}''}{\tilde{j}'} = -\dfrac{j'^2\Phi_{\ell_{xx}}(j, \tilde{j}) + 2\ell j'\tilde{j}'\Phi_{\ell_{xy}}(j, \tilde{j}) + \ell^2\tilde{j}'^2\Phi_{\ell_{yy}}(j, \tilde{j})}{j'\Phi_{\ell_x}(j, \tilde{j})}$.

Compute $p_1 = \dfrac{\ell}{2}J + \dfrac{\ell}{4}\left(\dfrac{E_4^2}{E_6} - \dfrac{\ell\tilde{E}_4^2}{\tilde{E}_6}\right) + \dfrac{\ell}{3}\left(\dfrac{E_6}{E_4} - \dfrac{\ell\tilde{E}_6}{\tilde{E}_4}\right)$.

Compute $c_1 = -\dfrac{a_4}{5}, c_2 = -\dfrac{a_6}{7}$.

Compute $c_k = \dfrac{3}{(k-2)(2k+3)}\sum_{j=1}^{k-2} c_j c_{k-1-j}$ for $3 \le k \le d$.

Compute $\hat{a}_4 = \ell^4\tilde{a}_4$ and $\hat{a}_6 = \ell^6\tilde{a}_6$.

Compute $\hat{c}_1 = -\dfrac{\hat{a}_4}{5}, \hat{c}_2 = -\dfrac{\hat{a}_6}{7}$.

Compute $\hat{c}_k = \dfrac{3}{(k-2)(2k+3)}\sum_{j=1}^{k-2} \hat{c}_j \hat{c}_{k-1-j}$ for $3 \le k \le d$.

Set $w = z^2 \in \mathbb{Z}[[z]]$.

Compute $C(w) = \sum_{k=1}^{d} c_k w^k$.

Compute $A(w) = \exp\left(-\dfrac{p_1}{2}w - \sum_{k=1}^{d} \dfrac{\hat{c}_k - \ell c_k}{(2k+1)(2k+2)} w^{k+1}\right)$.

Compute $a_{d-i} = [A(w)]_i - \sum_{k=1}^{i}\left(\sum_{j=0}^{k}\binom{d-i+k}{k-j}[C(w)^{k-j}]_j\right) a_{d-i+k}$ for $1 \le i \le d$.

Compute $F_\ell(x) = x^d + \sum_{i=1}^{d} a_{d-i}x^{d-i}$.

**return** $F_\ell(x)$.

improvement compared to the complexity of Schoof's algorithm. More information about the calculation of the complexity of the SEA algorithm can be found in (Blake et al., 1999). The algorithm of Elkies' procedure is given in Algorithm 7.

**Algorithm 6:** AddAffinePointsGeneric

**Input :** Two affine points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on an elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + a_4x + a_6$ and $i_1$ and $i_2$, Boolean indicators of being the identity element $\mathcal{O} = (0, 0)$ for point $P$ and $Q$, respectively.

**Output :** An affine point $R = P + Q = (x_3, y_3)$ on $E/\mathbb{F}_p$ and a Boolean indicator of this point which is the identity element $\mathcal{O}$.

**if** $i_1 =$ **true then**
  **return** $x_2, y_2, i_2$.
**else if** $i_2 =$ **true then**
  **return** $x_1, y_1, i_1$.
**else if** $x_1 = x_2$ **then**
  **if** $y_1 = 0$ **then**
    **return** $0, 0,$ **true**.
  **else if** $y_1 \neq y_2$ **then**
    **return** $0, 0,$ **true**.
  **else**
    Compute $L = \dfrac{3x_1^2 + a_4}{2y_1}$.
    Compute $x_3 = L^2 - 2x_1$.
    Compute $y_3 = L(x_1 - x_3) - y_1$.
    **return** $x_3, y_3,$ **false**.
  **end if**
**else**
  Compute $L = \dfrac{y_2 - y_1}{x_2 - x_1}$.
  Compute $x_3 = L^2 - x_1 - x_2$.
  Compute $y_3 = L(x_1 - x_3) - y_1$.
  **return** $x_3, y_3,$ **false**.
**end if**

### 4.2.3 BRINGING INFORMATION TOGETHER

After completing the trace collection for each prime $\ell$ according to the procedure of Atkin or Elkies, one last operation is needed to obtain the final trace $t$. In Schoof's algorithm, only performing a CRT was enough to find the trace value. However, in the SEA algorithm, CRT fails because of the candidate traces produced in Atkin procedures. That is why another process called Match and Sort algorithm (Müller, 1995) is used to merge the trace values. This method stems from the well-known BSGS algorithm. It is a sub-exponential time algorithm and is described in the following section.

The first task is to split the candidate traces into two distinct sets, $S_1$ and $S_2$ where

**Algorithm 7:** ElkiesAlgorithm

**Input :** An Elkies Prime $\ell$ and an elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + a_4x + a_6$.
**Output :** Set $E_t$ that contains the trace $E_{t_\ell}$ and related prime $\ell$.

Set $E_t = \{\}$.
Compute $F_\ell(x) = \texttt{FindKernelPolynomial}(\ell, E/\mathbb{F}_p)$.
Create the quotient ring $Q_\ell = \mathbb{F}_p[x, y]/(y^2 - x^3 - a_4x - a_6, F_\ell(x))$.
Set $(x_1, y_1) = (0, 0) \in Q_\ell, (x_2, y_2) = (x, y) \in Q_\ell, (x_p, y_p) = (x^p, y^p) \in Q_\ell$.
Set the Boolean identity indicator $i_1 =$ **true**.
**for** $k = 1$ **to** $(\ell - 1)/2$ **do**
    Compute
    $(x_1, y_1) = \texttt{AddAffinePointsGeneric}(x_1, y_1, i_1, x_2, y_2,$ **false**$, a_4)$.
    **if** $x_1 = x_p$ **then**
        **if** $y_1 = y_p$ **then**
            Set $\lambda = k$.
        **else**
            Set $\lambda = -k$.
        **end if**
        Compute $E_{t_\ell} = \lambda + \dfrac{p}{\lambda} \pmod{\ell}$.
        Set $E_t = \{E_{t_\ell}, \ell\}$.
        **break**.
    **end if**
**end for**
**return** $E_t$.

$S$ denotes all the candidate traces gathered from Atkin procedures, and $S_1 \cup S_2 = S, S_1 \cap S_2 = \emptyset$. A nice partitioning of $S$ into $S_1$ and $S_2$ can be achieved, in terms of having the lowest computational cost, by adjusting the cardinalities of the sets obeying the condition $1 \leq |S_1|/|S_2| \leq 2$. After determining the sets, $m_1, m_2$ and $m_3$ can be determined by multiplying corresponding primes for the elements in $S_1, S_2$, and the Elkies trace set $E_t$, respectively. Since $E_t$ has no candidate traces (meaning there is only a single trace modulo a specific prime), the collective trace $t_3 \equiv t \pmod{m_3}$ detected from $E_t$ can be easily found using CRT. Note that the traces for $m_1$ and $m_2$ also follow the same idea as $t_1 \equiv t \pmod{m_1}$ and $t_2 \equiv t \pmod{m_2}$, where $t_1 \in S_1$ and $t_2 \in S_2$. Here the trace that is needed to be found is $t$, and there exists an equation

$$t = t_3 + m_3(m_2 r_1 + m_1 r_2), \tag{4.46}$$

where $m_1, m_2, m_3$ and $t_3$ are known, $m_1 m_2 m_3 > 4\sqrt{p}$ and $r_1, r_2 \in \mathbb{Z}$ are unknown. Under modulo $m_1$ and $m_2$, Equation (4.46) turns into

$$\begin{aligned} t \equiv t_1 \equiv t_3 + m_2 m_3 r_1 \pmod{m_1}, \\ t \equiv t_2 \equiv t_3 + m_1 m_3 r_2 \pmod{m_2}, \end{aligned} \tag{4.47}$$

respectively. Reordering the terms leads to the emergence of

$$\begin{aligned} r_1 \equiv \frac{t_1 - t_3}{m_2 m_3} \pmod{m_1}, \\ r_2 \equiv \frac{t_2 - t_3}{m_1 m_3} \pmod{m_2}. \end{aligned} \tag{4.48}$$

However, there is still not enough information to determine $r_1$ and $r_2$. Fortunately, the Lemma below provides helpful information about the possible boundaries of $r_1$ and $r_2$.

**Lemma 4.2.1.** *If the conditions*

$$\begin{aligned} 0 \leq t_3 < m_3, \\ \lfloor \frac{-m_1}{2} \rfloor < r_1 \leq \lfloor \frac{m_1}{2} \rfloor, \end{aligned}$$

*are satisfied, then* $|r_2| \leq m_2$ *applies.*

*Proof.* See (Blake et al., 1999, Lemma VII.10). □

Let $P$ be a non-trivial random point on an elliptic curve $E/\mathbb{F}_p$. From Equation (4.1) and Equation (4.46),

$$[p+1]P = [t]P = [t_3 + m_3(m_2 r_1 + m_1 r_2)]P,$$

and therefore,

$$[p+1-t_3]P - [m_2m_3r_1]P = [m_1m_3r_2]P. \tag{4.49}$$

The work to be done here is, for every possible $t_1 \in S_1$, to store each $r_1 \equiv \frac{t_1 - t_3}{m_2m_3} \pmod{m_1}$ that also ensures the inequality $|r_1| \leq \lfloor \frac{m_1}{2} \rfloor$ which was given in Lemma 4.2.1, together with the left hand side of Equation (4.49) as $Q_{r_1} = [p+1-t_3]P - [m_2m_3r_1]P$. The stored pairs of $(Q_{r_1}, r_1)$ can be kept sorted to increase the overall performance of the information gathering work. Constructing such pairs corresponds to the *baby-step* parts of the BSGS algorithm.

For every possible $t_2 \in S_2$, determining each value of $r_2$ that keeps $r_2 \equiv \frac{t_2 - t_3}{m_1m_3} \pmod{m_2}$ and $|r_2| \leq m_2$ of Lemma 4.2.1 valid and computing the right-hand side of Equation (4.49) as $Q_{r_2} = [m_1m_3r_2]P$ with the determined $r_2$ values corresponds to the *giant-step* parts of the BSGS algorithm. There is no need to store $Q_{r_2}$ values because a match is sought between the recently calculated $Q_{r_2}$ and $Q_{r_1}$ values of the set created in baby steps. If a match is not found, another giant step is performed, and equality is sought again. This process can be done faster by using the binary search algorithm. Finding a match means that the corresponding $r_1$ and $r_2$ values can be used for detecting the trace $t$, as in Equation (4.46).

### 4.2.4 THE FULL ALGORITHM

The whole procedure for the SEA algorithm starts with the computation of the $j$-invariant of the given elliptic curve $E$. Then, trace modulo 2 is determined by checking the $\gcd$ of $x^p - x$ and the curve equation, as in Schoof's algorithm. The reason for this can be found in Section 4.1.1. Next, the main loop starts, and after getting the next consecutive prime and computing the multiplication of primes used, $M$, the degree of the $\gcd$ of $x^p - x$ and the $\ell$-th modular polynomial whose $y$ coordinate is evaluated by the $j$-invariant of the elliptic curve. If the degree is 0, Atkin's procedure is performed; otherwise, Elkies' procedure is performed. The loop iterates until $M$ reaches $4\sqrt{p}$. Finally, collected traces from Atkin and Elkies' procedures are combined using the Match and Sort algorithm. The trace obtained at the end of this algorithm is the desired trace of Frobenius $t$. Thus, the number of points on $E$ can be calculated as $\#E = p + 1 - t$. The complete SEA algorithm is given in Algorithm 8. Additionally, a toy example for the SEA algorithm is provided in Example 4.2.1.

**Algorithm 8:** SEA Algorithm

**Input :** An elliptic curve $E : y^2 = x^3 + a_4x + a_6$ over a finite field $\mathbb{F}_p$.
**Output :** $\#E(\mathbb{F}_p)$.

Set $M = 1, \ell = 2, A_t = \{\}, E_t = \{\}$.
Compute the $j$-invariant of $E/\mathbb{F}_p$ as $j = 1728\dfrac{4a_4^3}{a_4^3 + 27a_6^2} \in \mathbb{F}_p$.
**if** $\gcd(x^p - x, x^3 + a_4x + a_6) = 1$ **then**
    Set $E_t = \{(1, 2)\}$.
**else**
    Set $E_t = \{(0, 2)\}$.
**end if**
**while** $M < 4\sqrt{p}$ **do**
    Set $\ell = \texttt{NextPrime}(\ell)$.
    Compute $M = M \cdot \ell$.
    **if** $\deg(\gcd(x^p - x, \Phi_\ell(x, j))) = 0$ **then**
        Compute $A_{t_\ell} = \texttt{AtkinAlgorithm}(\ell, \Phi_\ell(x, j))$.
        Set $A_t = A_t \cup \{(A_{t_\ell}, \ell)\}$.
    **else**
        Compute $E_{t_\ell} = \texttt{ElkiesAlgorithm}(\ell, E/\mathbb{F}_p)$.
        Set $E_t = E_t \cup \{(E_{t_\ell}, \ell)\}$.
    **end if**
**end while**
Get $t$ by combining information stored in $A_t$ and $E_t$ with Match and Sort algorithm.
**return** $\#E(\mathbb{F}_p) = p + 1 - t$.

**Example 4.2.1.** Consider an elliptic curve $E/\mathbb{F}_{617} : y^2 = x^3 + 347x + 76$. The SEA algorithm will find the number of points on this curve. Note that every arithmetic operation will be done in $\mathbb{F}_{617}$.

At first, trace mod 2 should be determined. The $\gcd(x^{617} - x, x^3 + 347x + 76) = x + 56$ indicates that the result is not equal to 1, the trace $t_2 = 0$ and it is stored with the related prime $\ell = 2$ as $(0, 2)$. Up to which prime the procedures would be applied is determined by multiplying the consecutive primes and checking if $M = \prod_{i=1}^{k} \ell_i > 4\sqrt{617}$ where $\ell \geq 3$ holds or not. So, $4\sqrt{617} = 99.35$ and $M = 3 \cdot 5 \cdot 7 = 105$, the primes to be used are $3, 5$ and $7$.

For the case $\ell = 3$, the classical modular polynomial

$$\begin{aligned}\Phi_3(x, y) = {} & x^4 + 616x^3y^3 + 381x^3y^2 + 539x^3y + 101x^3 + 381x^2y^3 + 434x^2y^2 + \\ & 377x^2y + 521x^2 + 539xy^3 + 377xy^2 + 407xy + 568x + y^4 + 101y^3 + \\ & 521y^2 + 568y\end{aligned}$$

is first evaluated at $(x, j)$ where $j = 1728 \cdot \dfrac{4 \cdot 347^3}{4 \cdot 347^3 + 27 \cdot 76^2} \pmod{617} = 391$.

The result is $\Phi_3(x, j) = \Phi_3(x, 391) = x^4 + 569x^3 + 520x^2 + 207x + 515$. Then, $\gcd(\Phi_3(x, 391), x^{617} - x) = x^2 + 261x + 140$ is computed. The degree of $\gcd$ is not equal to 0, so $\ell = 3$ is an Elkies prime. The procedure for Elkies primes should be followed by first computing $E_4 \equiv -48 \cdot 347 \pmod{617} = 3$, $E_6 \equiv 864 \cdot 76 \pmod{617} = 262$ and $j' \equiv -\dfrac{391 \cdot 3}{262} \pmod{617} = 199$. Next, one of the roots of $\Phi_3(x, 391)$, which are 13 and 343, is set as $\tilde{j}$. Let $\tilde{j} = 13$. Here, $\Phi_3(j, \tilde{j}) = \Phi_3(391, 13) = 0$. Then, the partial derivatives of $\Phi_3(x, y)$ are computed as

$$\begin{aligned}
\Phi_{3_x}(x, y) &= 4x^3 + 614x^2y^3 + 526x^2y^2 + 383x^2y + 303x^2 + 145xy^3 + 251xy^2 + \\
&\quad 137xy + 425x + 539y^3 + 377y^2 + 407y + 568, \\
\Phi_{3_y}(x, y) &= 614x^3y^2 + 145x^3y + 539x^3 + 526x^2y^2 + 251x^2y + 377x^2 + 383xy^2 + \\
&\quad 137xy + 407x + 4y^3 + 303y^2 + 425y + 568, \\
\Phi_{3_{xx}}(x, y) &= 12x^2 + 611xy^3 + 435xy^2 + 149xy + 606x + 145y^3 + 251y^2 + 137y + \\
&\quad 425, \\
\Phi_{3_{xy}}(x, y) &= 608x^2y^2 + 435x^2y + 383x^2 + 435xy^2 + 502xy + 137x + 383y^2 + \\
&\quad 137y + 407, \\
\Phi_{3_{yy}}(x, y) &= 611x^3y + 145x^3 + 435x^2y + 251x^2 + 149xy + 137x + 12y^2 + 606y + \\
&\quad 425,
\end{aligned}$$

and the evaluations of these derivatives at $(391, 13)$ give

$$\begin{aligned}
\Phi_{3_x}(391, 13) &= 519, \\
\Phi_{3_y}(391, 13) &= 30, \\
\Phi_{3_{xx}}(391, 13) &= 159, \\
\Phi_{3_{xy}}(391, 13) &= 525, \\
\Phi_{3_{yy}}(391, 13) &= 558.
\end{aligned}$$

After that, $\tilde{j}' = -\dfrac{j'\Phi_{3_x}(391, 13)}{\ell\Phi_{3_y}(391, 13)} = -\dfrac{199 \cdot 519}{3 \cdot 30} \pmod{617} = 107$ is computed. Now, the coefficients for the isogenous curve $\tilde{E} : y^2 = x^3 + \tilde{a}_4 x + \tilde{a}_6$ can be computed as $\tilde{a}_4 \equiv -\dfrac{107^2}{48 \cdot 13 \cdot (13 - 1728)} \pmod{617} = 476$ and $\tilde{a}_6 \equiv -\dfrac{107^3}{864 \cdot 13^2 \cdot (13 - 1728)} \pmod{617} = 70$ with $\tilde{E}_4 \equiv -48 \cdot 476 \pmod{617} = 598$ and $\tilde{E}_6 \equiv 864 \cdot 70 \pmod{617} = 14$. The sum of the roots in the kernel of the isogeny, $p_1$, can be computed by first computing

$$J \equiv -\frac{199^2 \cdot 159 + 2 \cdot 3 \cdot 199 \cdot 107 \cdot 525 + 3^2 \cdot 107^2 \cdot 558}{199 \cdot 519} \pmod{617} = 66,$$

and then

$$p_1 \equiv \frac{3}{2} \cdot 66 + \frac{3}{4} \cdot \left( \frac{3^2}{262} - \frac{3 \cdot 598^2}{14} \right) + \frac{3}{3} \cdot \left( \frac{262}{3} - \frac{3 \cdot 14}{598} \right) \pmod{617} = 577.$$

At this point, the coefficients of the Laurent series related to $E$ and the isogenous curve $\hat{E} : y^2 = x^3 + \hat{a}_4 x + \hat{a}_6$ can be computed up to $d = (3-1)/2 = 1$ as

$$c_1 \equiv -\frac{347}{5} \pmod{617} = 54,$$

and $\hat{a}_4 \equiv 3^4 \cdot 476 \pmod{617} = 302$ such that

$$\hat{c}_1 \equiv -\frac{476}{5} \pmod{617} = 63.$$

Setting $w = z^2 \in \mathbb{Z}[[z]]$ results in:

$$\begin{aligned} C(w) &= 54w, \\ A(w) &= \exp\left( -\frac{577}{2} w - \frac{63 - 3 \cdot 54}{3 \cdot 4} w^2 \right) = 1 + 20w + 54w^2 + 470w^3 + \ldots, \\ a_0 &= 20 - \left( \binom{1}{1} \cdot 0 + \binom{1}{0} \cdot 0 \right) \cdot 1 = 20. \end{aligned}$$

Therefore, the kernel polynomial is $x + 20$ and the quotient ring to be worked in can be created as $Q_3 = \mathbb{F}_p[x,y]/(y^2 - x^3 - a_4 x - a_6, x + 20)$ instead of $Q_3 = \mathbb{F}_p[x,y]/(y^2 - x^3 - a_4 x - a_6, \psi_3)$. Under this ring, equality between $(x^{617}, y^{617})$ and $[\lambda](x,y)$ should be checked. The point additions can be made quickly with Algorithm 6. It can be seen that equality is obtained directly as $(x^{617}, y^{617}) = (x,y) = (597, y)$ in $Q_3$, so $\lambda = 1$ and $t_3 \equiv 1 + 617 \pmod 3 = 0$. The trace and prime are stored as $(0, 3)$.

For the case $\ell = 5$, the classical modular polynomial

$$\begin{aligned} \Phi_5(x,y) = {} & x^6 + 616x^5y^5 + 18x^5y^4 + 52x^5y^3 + 195x^5y^2 + 113x^5y + 73x^5 + 18x^4y^5 \\ & + 452x^4y^4 + 323x^4y^3 + 33x^4y^2 + 520x^4y + 31x^4 + 52x^3y^5 + 323x^3y^4 \\ & + 235x^3y^3 + 547x^3y^2 + 334x^3y + 35x^3 + 195x^2y^5 + 33x^2y^4 + 547x^2y^3 \\ & + 13x^2y^2 + 422x^2y + 612x^2 + 113xy^5 + 520xy^4 + 334xy^3 + 422xy^2 \\ & + 483xy + 147x + y^6 + 73y^5 + 31y^4 + 35y^3 + 612y^2 + 147y + 440 \end{aligned}$$

is evaluated at $(x, 391)$. The result is $\Phi_5(x, 391) = x^6 + 432x^5 + 398x^4 + 452x^3 + 239x^2 + 38x + 214$. Then, $\gcd(\Phi_5(x, 391), x^{617} - x) = 1$ is computed. The degree of $\gcd$ is $0$, so $\ell = 5$ is an Atkin prime. The first task to be accomplished is finding the order of the Frobenius endomorphism $\varphi$ that acts on the torsion group $E[5]$, which is done using Algorithm 3. Applying the algorithm results in encountering $\gcd(0, \Phi_5(x, 391)) =$

$\Phi_5(x, 391)$ at the last iteration, so $r = 6$. Next, a non-square $d$ in $\mathbb{F}_5$ is needed to be found to construct the multiplicative group $\mathbb{F}_5[\sqrt{d}]$, where the characteristic polynomial would split into two roots. After a short search, $d$ is determined as 2. Later, a primitive 6-th root of unity $\gamma = g_1 + g_2\sqrt{2}$ is chosen from this multiplicative group where $g_1 = 3$ and $g_2 = 3$. Using $3 + 3\sqrt{2}$ as a generator, the other 6-th roots of unity can be found by computing $(3 + 3\sqrt{2})^i$ when $\gcd(i, 6) = 1$, for $1 \leq i < 6$. As a result, $S = \{3 + 3\sqrt{2}, 3 + 2\sqrt{2}\}$ are determined as all the primitive 6-th roots of unity in $\mathbb{F}_{5^2}$. Due to symmetry, performing $\phi_{Eul}(6)/2 = 1$ iteration will be enough. Thus, $(617 \mod 5) \cdot (3 + 1)/2 = 4$ is a square in $\mathbb{F}_5$ and 3 is the square root of it in $\mathbb{F}_5$. The candidate traces $2 \cdot 3 = 6$ and $-2 \cdot 3 = -6$ are stored as $([1, 4], 5)$ and the algorithm proceeds with the next prime.

The last case to be examined is $\ell = 7$. The classical modular polynomial

$$\begin{aligned}\Phi_7(x, y) = {} & x^8 + 616x^7y^7 + 272x^7y^6 + 451x^7y^5 + 444x^7y^4 + 163x^7y^3 + 295x^7y^2 + \\ & 336x^7y + 84x^7 + 272x^6y^7 + 276x^6y^6 + 464x^6y^5 + 61x^6y^4 + 608x^6y^3 \\ & + 235x^6y^2 + 583x^6y + 428x^6 + 451x^5y^7 + 464x^5y^6 + 134x^5y^5 + \\ & 199x^5y^4 + 242x^5y^3 + 46x^5y^2 + 459x^5y + 588x^5 + 444x^4y^7 + 61x^4y^6 + \\ & 199x^4y^5 + 403x^4y^4 + 100x^4y^3 + 278x^4y^2 + 300x^4y + 280x^4 + 163x^3y^7 \\ & + 608x^3y^6 + 242x^3y^5 + 100x^3y^4 + 379x^3y^3 + 261x^3y^2 + 118x^3y + \\ & 583x^3 + 295x^2y^7 + 235x^2y^6 + 46x^2y^5 + 278x^2y^4 + 261x^2y^3 + 477x^2y^2 \\ & + 110x^2y + 240x^2 + 336xy^7 + 583xy^6 + 459xy^5 + 300xy^4 + 118xy^3 \\ & + 110xy^2 + 380xy + y^8 + 84y^7 + 428y^6 + 588y^5 + 280y^4 + 583y^3 + \\ & 240y^2\end{aligned}$$

is evaluated at $(x, 391)$. The result is $\Phi_7(x, 391) = x^8 + 539x^7 + 17x^6 + 556x^5 + 428x^4 + 283x^3 + 327x^2 + 541x + 421$. Then, $\gcd(\Phi_7(x, 391), x^{617} - x) = x^2 + 564x + 608$ is computed. The degree of $\gcd$ is 2, so $\ell = 7$ is an Elkies prime. Herein, $E_4$, $E_6$ and $j'$ are already calculated at the case where $\ell = 3$. So, one can directly proceed to find the roots of $\Phi_7(x, 391)$, which are 280 and 390. Let $\tilde{j} = 280$. Then, $\Phi_7(j, \tilde{j}) = \Phi_7(391, 280) = 0$.

The partial derivatives of $\Phi_7(x, y)$ are computed as

$$\begin{aligned}\Phi_{7_x}(x,y) = {} & 8x^7 + 610x^6y^7 + 53x^6y^6 + 72x^6y^5 + 23x^6y^4 + 524x^6y^3 + 214x^6y^2 + \\ & 501x^6y + 588x^6 + 398x^5y^7 + 422x^5y^6 + 316x^5y^5 + 366x^5y^4 + \\ & 563x^5y^3 + 176x^5y^2 + 413x^5y + 100x^5 + 404x^4y^7 + 469x^4y^6 + \\ & 53x^4y^5 + 378x^4y^4 + 593x^4y^3 + 230x^4y^2 + 444x^4y + 472x^4 + 542x^3y^7 \\ & + 244x^3y^6 + 179x^3y^5 + 378x^3y^4 + 400x^3y^3 + 495x^3y^2 + 583x^3y + \\ & 503x^3 + 489x^2y^7 + 590x^2y^6 + 109x^2y^5 + 300x^2y^4 + 520x^2y^3 + \\ & 166x^2y^2 + 354x^2y + 515x^2 + 590xy^7 + 470xy^6 + 92xy^5 + 556xy^4 + \\ & 522xy^3 + 337xy^2 + 220xy + 480x + 336y^7 + 583y^6 + 459y^5 \\ & + 300y^4 + 118y^3 + 110y^2 + 380y\end{aligned}$$

$$\begin{aligned}\Phi_{7_y}(x,y) = {} & 610x^7y^6 + 398x^7y^5 + 404x^7y^4 + 542x^7y^3 + 489x^7y^2 + 590x^7y + \\ & 336x^7 + 53x^6y^6 + 422x^6y^5 + 469x^6y^4 + 244x^6y^3 + 590x^6y^2 + \\ & 470x^6y + 583x^6 + 72x^5y^6 + 316x^5y^5 + 53x^5y^4 + 179x^5y^3 + 109x^5y^2 \\ & + 92x^5y + 459x^5 + 23x^4y^6 + 366x^4y^5 + 378x^4y^4 + 378x^4y^3 + \\ & 300x^4y^2 + 556x^4y + 300x^4 + 524x^3y^6 + 563x^3y^5 + 593x^3y^4 + \\ & 400x^3y^3 + 520x^3y^2 + 522x^3y + 118x^3 + 214x^2y^6 + 176x^2y^5 + \\ & 230x^2y^4 + 495x^2y^3 + 166x^2y^2 + 337x^2y + 110x^2 + 501xy^6 + 413xy^5 \\ & + 444xy^4 + 583xy^3 + 354xy^2 + 220xy + 380x + 8y^7 + 588y^6 + \\ & 100y^5 + 472y^4 + 503y^3 + 515y^2 + 480y\end{aligned}$$

$$\begin{aligned}\Phi_{7_{xx}}(x,y) = {} & 56x^6 + 575x^5y^7 + 318x^5y^6 + 432x^5y^5 + 138x^5y^4 + 59x^5y^3 + \\ & 50x^5y^2 + 538x^5y + 443x^5 + 139x^4y^7 + 259x^4y^6 + 346x^4y^5 + \\ & 596x^4y^4 + 347x^4y^3 + 263x^4y^2 + 214x^4y + 500x^4 + 382x^3y^7 + \\ & 25x^3y^6 + 212x^3y^5 + 278x^3y^4 + 521x^3y^3 + 303x^3y^2 + 542x^3y + 37x^3 \\ & + 392x^2y^7 + 115x^2y^6 + 537x^2y^5 + 517x^2y^4 + 583x^2y^3 + 251x^2y^2 + \\ & 515x^2y + 275x^2 + 361xy^7 + 563xy^6 + 218xy^5 + 600xy^4 + 423xy^3 + \\ & 332xy^2 + 91xy + 413x + 590y^7 + 470y^6 + 92y^5 + 556y^4 + 522y^3 + \\ & 337y^2 + 220y + 480\end{aligned}$$

$$\begin{aligned}\Phi_{7_{xy}}(x,y) = {} & 568x^6y^6 + 318x^6y^5 + 360x^6y^4 + 92x^6y^3 + 338x^6y^2 + 428x^6y + 501x^6 \\ & + 318x^5y^6 + 64x^5y^5 + 346x^5y^4 + 230x^5y^3 + 455x^5y^2 + 352x^5y \\ & + 413x^5 + 360x^4y^6 + 346x^4y^5 + 265x^4y^4 + 278x^4y^3 + 545x^4y^2 \\ & + 460x^4y + 444x^4 + 92x^3y^6 + 230x^3y^5 + 278x^3y^4 + \\ & 278x^3y^3 + 583x^3y^2 + 373x^3y + 583x^3 + 338x^2y^6 + 455x^2y^5 + \\ & 545x^2y^4 + 583x^2y^3 + 326x^2y^2 + 332x^2y + 354x^2 + 428xy^6 + \\ & 352xy^5 + 460xy^4 + 373xy^3 + 332xy^2 + 57xy + 220x + 501y^6 + \\ & 413y^5 + 444y^4 + 583y^3 + 354y^2 + 220y + 380\end{aligned}$$

$$\begin{aligned}\Phi_{7_{yy}}(x,y) = {} & 575x^7y^5 + 139x^7y^4 + 382x^7y^3 + 392x^7y^2 + 361x^7y + 590x^7 + \\ & 318x^6y^5 + 259x^6y^4 + 25x^6y^3 + 115x^6y^2 + 563x^6y + 470x^6 + \\ & 432x^5y^5 + 346x^5y^4 + 212x^5y^3 + 537x^5y^2 + 218x^5y + 92x^5 + \\ & 138x^4y^5 + 596x^4y^4 + 278x^4y^3 + 517x^4y^2 + 600x^4y + 556x^4 + 59x^3y^5 \\ & + 347x^3y^4 + 521x^3y^3 + 583x^3y^2 + 423x^3y + 522x^3 + 50x^2y^5 + \\ & 263x^2y^4 + 303x^2y^3 + 251x^2y^2 + 332x^2y + 337x^2 + 538xy^5 + 214xy^4 \\ & + 542xy^3 + 515xy^2 + 91xy + 220x + 56y^6 + 443y^5 + 500y^4 + 37y^3 \\ & + 275y^2 + 413y + 480,\end{aligned}$$

and the evaluations of these derivatives at $(391, 280)$ give

$$\begin{aligned}\Phi_{7_x}(391,280) &= 506,\\ \Phi_{7_y}(391,280) &= 238,\\ \Phi_{7_{xx}}(391,280) &= 168,\\ \Phi_{7_{xy}}(391,280) &= 441,\\ \Phi_{7_{yy}}(391,280) &= 275.\end{aligned}$$

Next, $\tilde{j}' = -\dfrac{j'\Phi_{7_x}(391,280)}{\ell\Phi_{7_y}(391,280)} = -\dfrac{199\cdot 509}{7\cdot 238} \pmod{617} = 4$ and for the isogenous curve $\tilde{E} : y^2 = x^3 + \tilde{a}_4 x + \tilde{a}_6$, coefficients $\tilde{a}_4 \equiv -\dfrac{4^2}{48\cdot 280\cdot(280-1728)} \pmod{617} = 533$, $\tilde{a}_6 \equiv -\dfrac{4^3}{864\cdot 280^2\cdot(280-1728)} \pmod{617} = 329$ and $\tilde{E}_4 \equiv -48\cdot 533 \pmod{617} = 330$, $\tilde{E}_6 \equiv 864\cdot 329 \pmod{617} = 436$. The sum of the roots in the kernel of the isogeny, $p_1$, can be computed by first computing

$$J \equiv -\frac{199^2\cdot 168 + 2\cdot 7\cdot 199\cdot 4\cdot 441 + 7^2\cdot 4^2\cdot 275}{199\cdot 506} \pmod{617} = 249,$$

and then

$$p_1 \equiv \frac{7}{2} \cdot 249 + \frac{7}{4} \cdot \left( \frac{3^2}{262} - \frac{7 \cdot 330^2}{436} \right) + \frac{7}{3} \cdot \left( \frac{262}{3} - \frac{7 \cdot 436}{330} \right) \pmod{617} = 517.$$

Now, the coefficients of the Laurent series related to $E$ and the isogenous curve $\hat{E}$ : $y^2 = x^3 + \hat{a}_4 x + \hat{a}_6$ can be computed up to $d = (7-1)/2 = 3$ as

$$\begin{aligned} c_1 &\equiv -\frac{347}{5} \pmod{617} = 54, \\ c_2 &\equiv -\frac{76}{7} \pmod{617} = 518, \\ c_3 &\equiv \frac{3}{9} \cdot 54 \cdot 54 \pmod{617} = 355, \end{aligned}$$

and $\hat{a}_4 \equiv 7^4 \cdot 533 \pmod{617} = 75$, $\hat{a}_6 \equiv 7^6 \cdot 329 \pmod{617} = 260$ such that

$$\begin{aligned} \hat{c}_1 &\equiv -\frac{75}{5} \pmod{617} = 602, \\ \hat{c}_2 &\equiv -\frac{260}{7} \pmod{617} = 51, \\ \hat{c}_3 &\equiv \frac{3}{9} \cdot 602 \cdot 602 \pmod{617} = 75. \end{aligned}$$

Setting $w = z^2 \in \mathbb{Z}[[z]]$ results in:

$$\begin{aligned} C(w) &= 54w + 518w^2 + 355w^3, \\ A(w) &= \exp\left( -\frac{517}{2} w - \left( \frac{602 - 7 \cdot 54}{3 \cdot 4} w^2 + \frac{51 - 7 \cdot 518}{5 \cdot 6} w^3 + \frac{75 - 7 \cdot 355}{7 \cdot 8} w^4 \right) \right) \\ &= 1 + 50w + 203w^2 + 378w^3 + 276w^4 + \ldots, \\ a_2 &= 50 - \left( \binom{3}{1} \cdot 0 + \binom{3}{0} \right) \cdot 1 = 50, \\ a_1 &= 203 - \left( \left( \binom{2}{1} \cdot 0 + \binom{2}{0} \right) \cdot 50 + \left( \binom{3}{2} \cdot 0 + \binom{3}{1} \cdot 54 + \binom{3}{0} \right) \cdot 1 \right) = 41, \\ a_0 &= 378 - \left( \left( \binom{1}{1} \cdot 0 + \binom{1}{0} \right) \cdot 41 + \left( \binom{2}{2} \cdot 0 + \binom{2}{1} \cdot 54 + \binom{2}{0} \right) \cdot 50 + \right. \\ &\qquad \left. \left( \binom{3}{3} \cdot 0 + \binom{3}{2} \cdot 0 + \binom{3}{1} \cdot 518 \right) \cdot 1 \right) = 211. \end{aligned}$$

Therefore, the kernel polynomial is $x^3 + 50x^2 + 41x + 211$ and the quotient ring to be worked in can be created as $Q_7 = \mathbb{F}_p[x, y]/(y^2 - x^3 - a_4 x - a_6, x^3 + 50x^2 + 41x + 211)$ instead of $Q_7 = \mathbb{F}_p[x, y]/(y^2 - x^3 - a_4 x - a_6, \psi_7)$. Under this ring, equality between $(x^{617}, y^{617})$ and $[\lambda](x, y)$ should be checked. After performing point additions, one can see that $(x^{617}, y^{617}) = [2](x, y) = (445y^4 + 329y^2 + 120, 585y^5 + 244y^3 + 562y)$ in $Q_7$, so $\lambda = 2$ and $t_7 \equiv 2 + 617/2 \pmod{7} = 6$. The trace and prime are stored as $(6, 7)$.

After completing Atkin and Elkies parts, all the information gathered can be combined with Match and Sort algorithm. First, the information collected from Atkin parts can be handled. Since the Atkin procedure was only performed at $\ell = 5$ and the candidate traces were $[1, 4]$, the distribution of $S$ into $S_1$ and $S_2$ can be done as $S_1 = \{([1, 4], 5)\}$ and $S_2 = \{\}$ in order not to separate traces in the same modulo prime. So, $m_1 = 5$ and $m_2 = 1$. Here $m_2$ is set to 1 to avoid errors in the calculations. Then, $m_3 = 2 \cdot 3 \cdot 7 = 42$ and $m_1 \cdot m_2 \cdot m_3 = 5 \cdot 1 \cdot 42 = 210 > 4\sqrt{617} = 99.35$ holds. The CRT for $t_3 \equiv 0 \pmod 2, t_3 \equiv 0 \pmod 3, t_3 \equiv 6 \pmod 7$ gives $t_3 = 6$. After that, the values for $r_1$ and $r_2$ that satisfy the conditions in Lemma 4.2.1 and Equation (4.48) are determined as $R_1 = \{-5, -1, 0, 4\}$ and $R_2 = \{-1, 0\}$, respectively. Let $P = (392, 348)$ be a random affine point on $E/\mathbb{F}_p$. Then, $[p + 1 - t_3]P = [612](392, 348) = (467, 192)$. Next, for every element $R_{1_i}$ of $R_1$, $[p + 1 - t_3]P - [R_{1_i} m_2 m_3]P$ is calculated as

$$
\begin{aligned}
(467, 192) - [-5 \cdot 1 \cdot 42](392, 348) &= (467, 192) - (23, 411) = (102, 533),\\
(467, 192) - [-1 \cdot 1 \cdot 42](392, 348) &= (467, 192) - (467, 192) = (0, 1),\\
(467, 192) - [0 \cdot 1 \cdot 42](392, 348) &= (467, 192) - (0, 1) = (467, 192),\\
(467, 192) - [4 \cdot 1 \cdot 42](392, 348) &= (467, 192) - (102, 533) = (23, 411).
\end{aligned}
$$

These points are sorted according to their $x$-coordinates and stored in a table to find a match later. Furthermore, for every element $R_{2_i}$ of $R_2$, $[R_{2_i} m_1 m_3]P$ is calculated as

$$
\begin{aligned}
[-1 \cdot 5 \cdot 42](392, 348) &= (23, 411),\\
[0 \cdot 5 \cdot 42](392, 348) &= (0, 1).
\end{aligned}
$$

It is easy to see that point $(23, 411)$ exists at both tables. So, necessary $r_1$ and $r_2$ values in order to calculate $(23, 411)$ are $r_1 = 4$ and $r_2 = -1$. At the end, the trace $t = t_3 + m_3(m_2 r_1 + m_1 r_2) = 6 + 42 \cdot (1 \cdot 4 + 5 \cdot -1) = -36$. Therefore, $\#E = p + 1 - t = 617 + 1 - (-36) = 654$.

## 4.3. EARLY ABORT METHOD

The meaning of being isogenous for two elliptic curves was explained in Definition 2.3.1. Besides being able to establish an isogeny in between, these curves have another important property that is closely related to point counting. This property is given in the following well-known theorem, which is due to (Tate, 1966).

**Theorem 4.3.1** (Sato-Tate's Isogeny Theorem (1966))**.** *Let $E_1$ and $E_2$ be two elliptic curves defined over a field $K$. $E_1$ and $E_2$ are isogenous if and only if*

$$\#E_1(K) = \#E_2(K). \tag{4.50}$$

Theorem 4.3.1 states that the isogenous property of elliptic curves can be decided by checking the number of points on elliptic curves. The number of points on an elliptic curve can be found with the SEA algorithm efficiently. Therefore, the straightforward method to detect whether two elliptic curves are isogenous or not is, finding the number of points on both elliptic curves and comparing the numbers.

There is a function in Magma Computer Algebra System (Bosma et al., 1997) called `IsIsogenous`. This function works with different types of input. However, depending on the scope of this thesis, the version working with elliptic curves will be considered. It takes two elliptic curves defined over a field and returns a Boolean output according to whether the given elliptic curves are isogenous. As written on the Magma handbook on the website (Bosma, Cannon, & Playoust, 2010):

> *IsIsogenous(E, F) : CrvEll[FldFin], CrvEll[FldFin] -> BoolElt*
> *Given two elliptic curves E and F defined over the rationals or a finite field, this function returns true if the curves E and F are isogenous over this field and false otherwise. In the rational case, if the curves are isogenous then the isogeny will be returned as the second value. For finite fields the isogeny computation operates via point counting and thus no isogeny is returned.*

The last statement can be checked using the following Magma code.

```
F := GF(2^255-19);
A4_1 := F!17;
A6_1 := F!861;
A4_2 := F!654157;
A6_2 := F!98791;
E1 := EllipticCurve([A4_1,A6_1]);
E2 := EllipticCurve([A4_2,A6_2]);

time #E1 + #E2;

clear;
F := GF(2^255-19);
A4_1 := F!17;
A6_1 := F!861;
A4_2 := F!654157;
A6_2 := F!98791;
E1 := EllipticCurve([A4_1,A6_1]);
E2 := EllipticCurve([A4_2,A6_2]);

time IsIsogenous(E1,E2);
```

**Code 4.1.** IsIsogenous Check Code

A finite field with prime $p = 2^{255} - 19$ is chosen in the code. Then, the coefficients of the elliptic curves $E_1 : y^2 = x^3 + a_{4_1}x + a_{6_1}$ and $E_2 : y^2 = x^3 + a_{4_2}x + a_{6_2}$ are selected in this field and the elliptic curves are created. The number of points on $E_1$ and $E_2$ are added while printing the total time spent for the computation of this operation.

Here, the addition is done to trigger the computation of the number of points with the SEA algorithm for both elliptic curves. The `clear` command was used, and the same procedure was applied again to prevent Magma from caching the result. In the end, the `IsIsogenous` function is called with both curves, and the total time spent is printed. The code was run on the Online Magma Calculator (Bosma, Cannon, & Playoust, n.d.). The output is given below.

```
115792089237316195423570985008687907853332541735889112675281281684049368446066
Time: 13.640
false
Time: 13.680
```

**Figure 4.1.** The Output of the IsIsogenous Check Code

As seen in the output, the times taken for the operations are almost equal. So, Magma does compute the number of points for each curve.

The traces collected under the modulo of different primes from Atkin and Elkies procedures are combined with a CRT-based method called Match and Sort. As given in (Shoup, 2009, p. 24), by the nature of the CRT, there is a one-to-one and onto mapping between the trace of Frobenius $t$ and each trace $t_\ell \pmod{\ell}$. Therefore, the traces of Frobenius computed from two elliptic curves can be equal if and only if every trace $t_\ell$ values are equal under the same modulo $\ell$. In other words, at least one trace $t_\ell$ calculated under the same modulo must be different between two curves if and only if the curves are not isogenous. This property allows early aborting in the calculation phase possible when detecting (non-)isogenous curves. Hence, it leads to faster detection of cases where the curves are not isogenous. The early abort mechanism is only used in Elkies parts of the algorithm because multiple candidate traces come from Atkin procedures, and there is no way to find the correct trace here. Elkies procedure produces a single trace value modulo prime $\ell$. Therefore the traces that appeared after performing Elkies procedures can be compared. Performing all Elkies procedures before Atkin procedures increases the chance of encountering an inequality between traces. The Atkin primes can be collected in every encounter, and the remaining Atkin procedures can be performed after Elkies procedures, using the collected Atkin primes. If the curves are isogenous, then every trace calculated should be equal to each other, and no termination can be done, resulting in the execution of the whole SEA algorithm for both curves and reaching the case in the `IsIsogenous` function of Magma. The algorithm of the described operation from a high-level perspective is given in Algorithm 9.

**Algorithm 9:** CheckIfIsogenous

**Input :** Two elliptic curves $E_1 : y^2 = x^3 + a_{4_1}x + a_{6_1}$ and $E_2 : y^2 = x^3 + a_{4_2}x + a_{6_2}$ over a prime field $\mathbb{F}_p$.

**Output :** A Boolean that shows whether $E_1$ and $E_2$ are isogenous.

Set $M = 1, \ell = 2, E_t = \{\}, A_{t_1} = \{\}, A_{t_2} = \{\}, A_\ell = \{\}$.
Compute $j$-invariants $j_1$ and $j_2$ of $E_1$ and $E_2$, respectively.
Set $e_1 = \gcd(x^p - x, x^3 + a_{4_1}x + a_{6_1})$.
Set $e_2 = \gcd(x^p - x, x^3 + a_{4_2}x + a_{6_2})$.
**if** $e_1 = 1$ and $e_2 \neq 1$ or $e_1 \neq 1$ and $e_2 = 1$ **then**
    **return false**.
**else if** $e_1 = 1$ **then**
    Set $E_t = \{(1, 2)\}$.
**else**
    Set $E_t = \{(0, 2)\}$.
**end if**
**while** $M \leq 4\sqrt{p}$ **do**
    Set $\ell = \texttt{NextPrime}(\ell)$.
    Compute $M = M \cdot \ell$.
    Compute $d_1 = \deg(\gcd(x^p - x, \Phi_\ell(x, j_1)))$.
    Compute $d_2 = \deg(\gcd(x^p - x, \Phi_\ell(x, j_2)))$.
    **if** $d_1 = 0$ and $d_2 \neq 0$ or $d_1 \neq 0$ and $d_2 = 0$ **then**
        **return false**.
    **end if**
    **if** $d_1 \neq 0$ **then**
        Compute $E_{t_{\ell_1}} = \texttt{ElkiesAlgorithm}(\ell, E_1/\mathbb{F}_p)$.
        Compute $E_{t_{\ell_2}} = \texttt{ElkiesAlgorithm}(\ell, E_2/\mathbb{F}_p)$.
        **if** $E_{t_{\ell_1}} \neq E_{t_{\ell_2}}$ **then**
            **return false**.
        **else**
            Set $E_t = E_t \cup (E_{t_{\ell_1}}, \ell)$.
        **end if**
    **else**
        Set $A_\ell = A_\ell \cup \ell$.
    **end if**
**end while**
**for** every prime $\ell \in A_\ell$ **do**
    Compute $A_{t_{\ell_1}} = \texttt{AtkinAlgorithm}(\ell, \Phi_\ell(x, j_1))$.
    Set $A_{t_1} = A_{t_1} \cup (A_{t_{\ell_1}}, \ell)$.
    Compute $A_{t_{\ell_2}} = \texttt{AtkinAlgorithm}(\ell, \Phi_\ell(x, j_2))$.
    Set $A_{t_2} = A_{t_2} \cup (A_{t_{\ell_2}}, \ell)$.
**end for**
Compute $t_1 = \texttt{MatchSort}(A_{t_1}, E_t, E_1/\mathbb{F}_p)$.
Compute $t_2 = \texttt{MatchSort}(A_{t_2}, E_t, E_2/\mathbb{F}_p)$.
**if** $t_1 \neq t_2$ **then**
    **return false**.
**else**
    **return true**.
**end if**

## 4.4. IMPLEMENTATION

There are several elliptic curve point counting implementations as the outputs of this thesis. All these implementations are done using a local Magma Computer Algebra System V2.21-5. The Magma implementations are publicly available at the following URL.

`https://github.com/yassimert/Schoof-SEA`

Firstly, Schoof's algorithm was implemented to make the implementation of the SEA algorithm easier. The necessary functions used in Schoof's and SEA algorithms, such as `AddAffinePointsGeneric` and $x$-only division polynomial generation function `divpols` were implemented in this phase. Then, Algorithm 2 was coded with the help of auxiliary Magma library functions like `Modexp, LegendreSymbol, Modsqrt`, et cetera. The function that contains the main loop of the implementation of Schoof's algorithm is given in Appendix B Code B.1. This function respectively takes the coefficients $a_4, a_6$ of the elliptic curve $E/\mathbb{F}_q$, a bivariate polynomial ring $P$ with variables $x, y$, variable $\_x$ of a univariate polynomial ring $R$ and the prime power $q = p^n$ where $n \geq 1$. After the computations, the function returns the total number of points $q + 1 - t$ on the elliptic curve $E/\mathbb{F}_q$. The algorithm is firstly implemented using classical modular polynomials to demonstrate the basic version of the SEA algorithm described in Section 4.2. This version can be found in the file *SEA-Classical.mag*. The main code structure of the SEA algorithm resembles Schoof's algorithm. After a proper random selection of an elliptic curve, the case for $\ell = 2$ is handled, and the main computation loop starts. Each prime is determined whether it is an Atkin or Elkies prime with the use of the $\ell$-th classical modular polynomial. In *SEA-Classical.mag*, the main loop operates until the product of each prime $\ell$ exceeds $4\sqrt{p}$, as in the usual case. Additionally, a bound called ClassicalMax, equal to 47 is used to check the largeness of $\ell$ at every iteration because the 47-th classical modular polynomial is the largest modular polynomial in the default Magma database. This check was added so that the code can be run effortlessly in any local Magma or Online Magma.

According to the type of prime $\ell$, Atkin or Elkies' procedure is applied. In the timeline of implementing the SEA algorithm in the context of this thesis, Atkin's method was implemented at first. Before entering the procedure, a check is performed to avoid gathering too much information from Atkin's part. Because if the number of collected candidate traces is large, it would be difficult to detect the trace of Frobenius $t$. Therefore, if the multiplication of the number of candidate traces collected from every Atkin procedure exceeds $\log_2 p$, then the prime $\ell$ is discarded, and the iteration continues with the next prime. Details of the auxiliary functions `FindOrderFrobenius` and

`getNonSquare` can be found in *SEA-Classical.mag*. The procedure is given in Appendix B Code B.2

In *SEA-Classical.mag*, Elkies' procedure strictly follows the method given in Section 4.2.2.1. Several assertions are added to the code to verify the equations in Section 4.2.2.1. The code is given in Appendix B Code B.3

The information combination part of the SEA algorithm is done with the Match and Sort algorithm. This part's code is straightforward and can be found in all SEA implementations this thesis provides.

The method described in Section 4.2.2.1 is the naive method to find the kernel polynomial, and it is an inefficient method to utilize in the SEA algorithm. It can be improved with the use of Atkin modular polynomials. This improvement takes the efficiency of the code a step further. Also, it enables the usage of the Atkin modular polynomial database in Magma. The 397-th Atkin modular polynomial is the largest modular polynomial used in Magma without involving additional libraries. So, the usage of Atkin modular polynomials paves the way for working with much larger numbers. The SEA implementation that employs Atkin modular polynomials can be found in *SEA-Atkin-v1.mag*. The acceleration gained by using Atkin modular polynomials can be further increased by changing the kernel polynomial computation method. Using the procedure given in (Galbraith, 2012) and (Stankovic, 2017), which Andrew Sutherland provides, decreases the running time complexity of the kernel polynomial finding procedure from $O(\ell^3)$ to $O(\ell^2)$. The SEA implementation that uses this method can be found in *SEA-Atkin-v2.mag*.

The implementation of the `CheckIfIsogenous` function can be found in *CheckIfIsogenous.mag*. In this function, Atkin and Elkies' procedures of *SEA-Atkin-v2.mag* are used to detect isogenous curves more efficiently. The function is compared with the `IsIsogenous` function of Magma. If two non-isogenous curves are selected, the `CheckIfIsogenous` function terminates much faster than the `IsIsogenous` function because of the early aborts. When a difference between small traces from Elkies procedures is detected, `CheckIfIsogenous` directly ends computations and outputs that the curves are non-isogenous. If two isogenous curves are selected, this is the worst case for the CheckIfIsogenous algorithm. As Magma does, the algorithm fully computes the number of points on both curves. However, Magma's SEA implementation is much faster than the implementation given in this thesis. That is why the `IsIsogenous` function of Magma works faster when two curves are chosen as isogenous. Since this situation is less common in general, it can be observed that the `CheckIfIsogenous` function will terminate more quickly in the vast majority of runs. The output of five runs for non-isogenous curves is given in Appendix A Figure A.1. Note that the runs are performed on an Intel® Core™ i5-7200U CPU while random prime $p$ is between

200 and 250 bits long, and all the curves are randomly generated on the field $\mathbb{F}_p$. The first time shows the total time taken for `CheckIfIsogenous`, and the second time shows the total time taken for `IfIsogenous` to run.

# CHAPTER 5
# CONCLUSION

In this thesis, a fast 4-way vectorized Montgomery ladder for the complete set of Montgomery curves was proposed at first. The efficient, competitive SIMD 4-way AVX2 and speed record holder SIMD 8-way AVX-512 implementations of this ladder were publicly presented. Then, various $\ell$-adic and $p$-adic point counting algorithms and studies related to these algorithms were reviewed. Afterwards, Schoof's algorithm and its improvement SEA algorithm, one of the most efficient and convenient $\ell$-adic point counting algorithms on elliptic curves over large prime fields $\mathbb{F}_p$, were discussed in detail. The preliminaries necessary for clarifying these algorithms were given concisely and descriptively. After gaining enough knowledge about Schoof's and SEA algorithms, the implementations of these algorithms were done using Magma Computer Algebra System. These include the first publicly available Magma implementations of the SEA algorithm using classical modular polynomials and Atkin modular polynomials. Also, a new early abort method was provided for detecting (non-)isogenous curves. A Magma implementation of this method was also provided, and the results are compared with the Magma function `IsIsogenous`. It was seen that the algorithm presented in this thesis gave faster results in the non-isogenous case. Nevertheless, the Magma function `IsIsogenous` terminated faster when two curves are isogenous because of the faster SEA implementation and optimizations. Consequently, working on the developments of the SEA algorithm, presenting a more efficient SEA implementation and investigating an algorithm that gives faster results both in the isogenous and non-isogenous cases are left as future works.

# REFERENCES


Abelard, S. (2018). *Counting points on hyperelliptic curves in large characteristic : algorithms and complexity* (Unpublished doctoral dissertation). Université de Lorraine.

Atkin, A. (1988). *The number of points on an elliptic curve modulo a prime.* (E-mail on the Number Theory Mailing List.)

Atkin, A. (1991). *Several public email messages.* (E-mail on the Number Theory Mailing List.)

Batut, C., Belabas, K., Bernardi, D., Cohen, H., & Olivier, M. (1990). *User's guide to PARI-GP*. Laboratoire A2X, Université Bordeaux I, France.

Bernstein, D. (2006). Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, & T. Malkin (Eds.), *Public key cryptography - PKC 2006, 9th international conference on theory and practice of public-key cryptography, new york, ny, usa, april 24-26, 2006, proceedings* (Vol. 3958, pp. 207–228). Springer. doi: 10.1007/11745853_14

Bernstein, D., Chuengsatiansup, C., Lange, T., & Schwabe, P. (2014). Kummer strikes back: New DH speed records. In P. Sarkar & T. Iwata (Eds.), *Advances in cryptology - ASIACRYPT 2014 - 20th international conference on the theory and application of cryptology and information security, kaoshiung, taiwan, r.o.c., december 7-11, 2014. proceedings, part I* (Vol. 8873, pp. 317–337). Springer. doi: 10.1007/978-3-662-45611-8_17

Bernstein, D., & Lange, T. (2017). Montgomery curves and the montgomery ladder. In J. W. Bos & A. K. Lenstra (Eds.), *Topics in computational number theory inspired by Peter L. Montgomery* (pp. 82–115). Cambridge University Press. doi: 10.1017/9781316271575.005

Bernstein, D., & Schwabe, P. (2012). NEON crypto. In E. Prouff & P. Schaumont (Eds.), *Cryptographic hardware and embedded systems – CHES 2012* (Vol. 7428, pp. 320–339). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bernstein, D. J., Lange, T., & Rezaeian Farashahi, R. (2008). Binary Edwards curves. In E. Oswald & P. Rohatgi (Eds.), *Cryptographic hardware and embedded systems – ches 2008* (pp. 244–265). Berlin, Heidelberg: Springer Berlin Heidelberg.

Blake, I. F., Seroussi, G., & Smart, N. P. (1999). *Elliptic curves in cryptography*. USA: Cambridge University Press. doi: 10.1017/cbo9781107360211

Bosma, W., Cannon, J., & Playoust, C. (n.d.). *The Magma computational algebra system.* Retrieved July 21, 2022, from http://magma.maths.usyd.edu.au/calc.

Bosma, W., Cannon, J., & Playoust, C. (1997). The Magma algebra system. I. The user language. *J. Symbolic Comput.*, *24*(3-4), 235–265. doi: 10.1006/jsco.1996.0125

Bosma, W., Cannon, J., & Playoust, C. (2010). *Magma handbook, operations on curves.* Retrieved July 21, 2022, from http://magma.maths.usyd.edu.au/magma/handbook/text/1491#17197.

Bostan, A., Morain, F., Salvy, B., & Schost, É. (2008). Fast algorithms for computing isogenies between elliptic curves. *Mathematics of Computation*, *77*(263), 1755–1778. doi: 10.1090/s0025-5718-08-02066-8

Brier, E., & Joye, M. (2002). Weierstraß elliptic curves and side-channel attacks. In D. Naccache & P. Paillier (Eds.), *Public key cryptography, 5th international workshop on practice and theory in public key cryptosystems, PKC 2002, paris, france, february 12-14, 2002, proceedings* (Vol. 2274, pp. 335–345). Springer. doi: 10.1007/3-540-45664-3_24

Broker, R., Lauter, K., & Sutherland, A. (2010). Modular polynomials via isogeny volcanoes. *Mathematics of Computation*, *81*, 1201-1231. doi: 10.1090/S0025-5718-2011-02508-1

Carls, R., & Lubicz, D. (2009). A p-adic quasi-quadratic time point counting algorithm. *International Mathematics Research Notices*. doi: 10.1093/IMRN/RNN143

Castryck, W., Galbraith, S., & Farashahi, R. R. (2008). *Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation.* Cryptology ePrint Archive, Report 2008/218. (https://eprint.iacr.org/2008/218)

Chou, T. (2015). Sandy2x: New Curve25519 speed records. In O. Dunkelman & L. Keliher (Eds.), *Selected areas in cryptography - SAC 2015 - 22nd international conference, sackville, nb, canada, august 12-14, 2015, revised selected papers* (Vol. 9566, pp. 145–160). Springer. doi: 10.1007/978-3-319-31301-6_8

Chudnovsky, D., & Chudnovsky, G. (1986). Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, *7*(4), 385–434.

Cohen, H. (1993). *A course in computational algebraic number theory*. New York: Springer Berlin Heidelberg. doi: 10.1007/978-3-662-02945-9

Cornacchia, G. (1908). Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$. *Giornale di Mat. di Battaglini*, 33–90.

Costello, C., & Smith, B. (2018). Montgomery curves and their arithmetic - The case of large characteristic fields. *J. Cryptographic Engineering*, *8*(3), 227–240. doi: 10.1007/s13389-017-0157-6

Couveignes, J.-M., & Morain, F. (1994). Schoof's algorithm and isogeny cycles. In *Algorithmic number theory* (pp. 43–58). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-58691-1_42

Cox, D. A. (2013). *Primes of the form* $x^2 + ny^2$. John Wiley & Sons, Inc. doi: 10.1002/9781118400722

Cremona, J., & Sutherland, A. (2010). On a theorem of Mestre and Schoof. *Journal de Théorie des Nombres de Bordeaux*, *22*(2), 353-358. doi: 10.5802/jtnb.719

Csirik, J. A. (2000). *An exposition of the sea algorithm.* Retrieved July 21, 2022, from http://www.csirik.net/sch-survey.pdf.

Denef, J., & Vercauteren, F. (2002). An extension of Kedlaya's algorithm to Artin-Schreier curves in characteristic 2. In *Algorithmic number theory* (pp. 308–323). Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/3-540-45455-1_25

Denef, J., & Vercauteren, F. (2006). An extension of Kedlaya's algorithm to hyperelliptic curves in characteristic 2. *Journal of Cryptology*, *19*, 1–25. doi: 10.1007/s00145-004-0231-y

Developers, T. M. (2018). *MIRACL, Multiprecision Integer and Rational Arithmetic Cryptographic Library.* Retrieved July 21, 2022, from https://miracl.com.

Developers, T. S., Stein, W., Joyner, D., Kohel, D., Cremona, J., & Eröcal, B. (2020). *Sagemath, the Sage Mathematics Software System (Version 9.0).* Retrieved July 21, 2022, from http://www.sagemath.org.

Dewaghe, L. (1998). Remarks on the Schoof-Elkies-Atkin algorithm. *Mathematics of Computation*, *67*(223), 1247–1252. doi: 10.1090/S0025-5718-98-00962-4

Doliskani, J. N. (2011). *Point counting on genus 2 curves* (Unpublished master's thesis). Western University.

Elkies, N. D. (1991). *Explicit isogenies.* (Manuscript)

Elkies, N. D. (1997). Elliptic and modular curves over finite fields and related computational issues. In *Computational perspectives on number theory: Proceedings of a conference in honor of A. O. L. Atkin* (Vol. 7). AMS/IP Studies in Advanced Mathematics. doi: 10.1090/amsip/007/03

Faz-Hernández, A., López, J., & Dahab, R. (2019, July). High-performance im-

plementation of elliptic curve cryptography using vector instructions. *ACM Trans. Math. Softw.*, *45*(3). Retrieved from https://doi.org/10.1145/3309759 doi: 10.1145/3309759

Fouquet, M., Gaudry, P., & Harley, R. (2000). An extension of Satoh's algorithm and its implementation. *Journal of the Ramanujan Mathematical Society*, *15*, 281–318.

Galbraith, S. D. (2012). *Mathematics of public key cryptography* (1st ed.). USA: Cambridge University Press. doi: 10.1017/CBO9781139012843

Galin, B. (2007). *Schoof-Elkies-Atkin algorithm.* Retrieved July 21, 2022, from http://www.bens.ws/papers/SchoofElkiesAtkinAlgorithm.pdf. (Senior Thesis, Stanford University)

Gaudry, P. (2007). Fast genus 2 arithmetic based on Theta functions. *Journal of Mathematical Cryptology (JMC)*, *1*(3), 243–265.

Gaudry, P., & Gürel, N. (2003). Counting points in medium characteristic using Kedlaya's algorithm. *Experimental Mathematics*, *12*, 395–402. doi: 10.1080/10586458.2003.10504508

Gaudry, P., & Harley, R. (2000). Counting points on hyperelliptic curves over finite fields. In *Lecture notes in computer science* (pp. 313–332). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/10722028_18

Gaudry, P., Kohel, D., & Smith, B. (2011). Counting points on genus 2 curves with real multiplication. In *Proceedings of the 17th international conference on the theory and application of cryptology and information security* (pp. 504–519). Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/978-3-642-25385-0_27

Gaudry, P., & Lubicz, D. (2009). The arithmetic of characteristic 2 Kummer surfaces and of elliptic Kummer lines. *Finite Fields and Their Applications*, *15*(2), 246 - 260. Retrieved from http://www.sciencedirect.com/science/article/pii/S1071579708000804 doi: https://doi.org/10.1016/j.ffa.2008.12.006

Gaudry, P., & Morain, F. (2006). Fast algorithms for computing the eigenvalue in the Schoof-Elkies-Atkin algorithm. In *Proceedings of the 2006 international symposium on symbolic and algebraic computation* (Vol. 2006, pp. 109–115). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1145768.1145791

Gaudry, P., & Schost, E. (2012). Genus 2 point counting over prime fields. *Journal of Symbolic Computation*, *47*(4), 368-400. doi: 10.1016/j.jsc.2011.09.003

Harris, N. (2005). *Math 168A final project: Computing $a_p$ for elliptic curves.* Retrieved

July 21, 2022, from https://www.wstein.org/projects/168/neal_harris/project.pdf. (University of California, San Diego)

Hasse, H. (1936). Zur theorie der abstrakten elliptischen funktionenkörper III. die struktur des meromorphismenrings. die riemannsche vermutung. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, *1936*(175), 193–208. doi: 10.1515/crll.1936.175.193

Hsieh, H.-T., & Yang, C. (2010). *Implementation of Schoof's algorithm on the Windows platforms.* Retrieved July 21, 2022, from http://security.nknu.edu.tw/publications/2010JWIS_SEA.pdf.

Hsieh, H.-T., Yang, C., & Ahn, T.-N. (2008). Implementation of SEA algorithm on the Windows platforms. *2008 International Conference on Convergence and Hybrid Information Technology*, 363–370. doi: 10.1109/ICHIT.2008.248

Hubrechts, H. (2007a). Point counting in families of hyperelliptic curves. *Foundations of Computational Mathematics*, *8*(1), 137–169. doi: 10.1007/s10208-007-9000-2

Hubrechts, H. (2007b). Point counting in families of hyperelliptic curves in characteristic 2. *LMS Journal of Computation and Mathematics*, *10*, 207–234. doi: 10.1112/s1461157000001376

Inc., W. R. (n.d.). *Mathematica, Version 13.1.* Retrieved July 21, 2022, from https://www.wolfram.com/mathematica. (Champaign, IL, 2022)

Intel. (n.d.-a). *Intel core i9-7900x x-series processor.* Retrieved July 21, 2022, from https://ark.intel.com/content/www/us/en/ark/products/123613/intel-core-i97900x-xseries-processor-13-75m-cache-up-to-4-30-ghz.html.

Intel. (n.d.-b). *Intel intrinsics guide.* Retrieved July 21, 2022, from https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

Intel. (n.d.-c). *Intel processor names and numbers.* Retrieved July 21, 2022, from https://www.intel.com/content/www/us/en/processors/processor-numbers.html.

Izu, T., Kogure, J., Noro, M., & Yokoyama, K. (1998). Efficient implementation of Schoof's algorithm. In *Advances in Cryptology — ASIACRYPT'98* (pp. 66–79). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-49649-1_7

Joux, A., & Lercier, R. (2001). "Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm. *Mathematics of Computation*, *70*, 827-836. doi: 10.1090/S0025-5718-00-01200-X

Joux, A., & Lercier, R. (2006). Counting points on elliptic curves in medium characteristic. *IACR Cryptol. ePrint Arch.*, *2006*, 176.

Karati, S., & Sarkar, P. (2017). Kummer for genus one over prime order fields. In T. Takagi & T. Peyrin (Eds.), *Advances in cryptology - ASIACRYPT 2017 - 23rd international conference on the theory and applications of cryptology and information security, hong kong, china, december 3-7, 2017, proceedings, part II* (Vol. 10625, pp. 3–32). Springer. Retrieved from https://doi.org/10.1007/978-3-319-70697-9_1 doi: 10.1007/978-3-319-70697-9_1

Kato, G. C., & Lubkin, S. (1982). Zeta matrices of elliptic curves. *Journal of Number Theory*, *15*(3), 318-330. doi: 10.1016/0022-314X(82)90036-1

Kedlaya, K. S. (2001). *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology.* arXiv. doi: 10.48550/ARXIV.MATH/0105031

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, *48*(177), 203–209. doi: 10.1090/S0025-5718-1987-0866109-5

Kok, Y. S. (2013). *Implementing the Schoof-Elkies-Atkin with NTL* (Unpublished master's thesis). University of Waterloo, UWSpace.

Lang, S. (1978). *Elliptic curves Diophantine analysis*. Springer-Verlag Berlin Heidelberg. doi: 10.1007/978-3-662-07010-9

Lauder, A. G. (2005). Rigid cohomology and p-adic point counting. *Journal de Théorie des Nombres de Bordeaux*, *17*, 169–180. doi: 10.5802/JTNB.484

Lenstra, H. W. (1987). Factoring integers with elliptic curves. *The Annals of Mathematics*, *126*(3), 649–673. doi: 10.2307/1971363

Lercier, R., & Lubicz, D. (2003). Counting points on elliptic curves over finite fields of small characteristic in quasi quadratic time. In *Advances in cryptology – EUROCRYPT 2003* (Vol. 2656, pp. 360–373). doi: 10.1007/3-540-39200-9_22

Lercier, R., & Lubicz, D. (2006). A quasi quadratic time algorithm for hyperelliptic curve point counting. *The Ramanujan Journal*, *12*(3), 399–423. doi: 10.1007/s11139-006-0151-6

Lercier, R., Lubicz, D., & Vercauteren, F. (2005). Point counting on elliptic and hyperelliptic curves. In *Handbook of elliptic and hyperelliptic curve cryptography* (pp. 407–454). Philadelphia, PA: Chapman & Hall/CRC. doi: 10.1201/9781420034981.ch17

Lercier, R., & Morain, F. (1995). Counting the number of points on elliptic curves over finite fields: strategies and performances. In *Advances in cryptology – EUROCRYPT '95* (pp. 79–94). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-49264-X_7

Lercier, R., & Morain, F. (1996). *Counting points on elliptic curves over $\mathbb{F}_{p^n}$ using Couveignes's algorithm.* Retrieved July 21, 2022, from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.5948&rep=rep1&type=pdf.

López, J., & Dahab, R. (1999). Fast multiplication on elliptic curves over GF($2^{\mathrm{m}}$) without precomputation. In Ç. Koç & C. Paar (Eds.), *Cryptographic hardware and embedded systems, first international workshop, ches'99, worcester, ma, usa, august 12-13, 1999, proceedings* (Vol. 1717, pp. 316–327). Springer. Retrieved from https://doi.org/10.1007/3-540-48059-5_27 doi: 10.1007/3-540-48059-5_27

Madsen, M. (2005). A $p$-adic point counting algorithm for elliptic curves on Legendre form. *Finite Fields and Their Applications*, *11*, 71-88. doi: 10.1016/j.ffa.2004.05.002

Mahto, D., & Yadav, D. (2017). RSA and ECC: A comparative analysis. *International Journal of Applied Engineering Research*, *12*, 9053-9061.

Matsuo, K., Chao, J., & Tsujii, S. (2002). An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. In *International algorithmic number theory symposium* (pp. 461–474). Springer. doi: 10.1007/3-540-45455-1_36

Maurer, M., & Müiler, V. (2001). Finding the eigenvalue in Elkies' algorithm. *Experimental Mathematics*, *10*(2), 275-285. doi: 10.1080/10586458.2001.10504448

McGee, J. J. (2006). *René Schoof's algorithm for determining the order of the group of points on an elliptic curve over a finite field* (Unpublished master's thesis). Virginia Polytechnic Institute and State University.

Menezes, A. (1993). Counting points on elliptic curves over $F_{2^m}$. In *Elliptic curve public key cryptosystems* (pp. 101–116). Boston, MA: Springer US. doi: 10.1007/978-1-4615-3198-2_7

Menezes, A., Katz, J., van Oorschot, P., & Vanstone, S. (2018). *Handbook of applied cryptography*. CRC Press. doi: 10.1201/9781439821916

Mestre, J. F. (2000). *Lettre adressée á Gaudry et Harley.* Retrieved July 21, 2022, from http://www.math.jussieu.fr/~mestre/.

Mestre, J. F. (2002). *Applications de l'AGM au calcul du nombre de points d'une courbe de genre 1 ou 2 sur $\mathbb{F}_{2^n}$*. Retrieved from http://www.maths.univ-rennes1.fr/crypto/2001-02/Mestre2203.html. (Talk given to the Séminaire de Cryptographie de l'Université de Rennes)

Mihailescu, P., Morain, F., & Schost, E. (2007). Computing the eigenvalue in the Schoof-Elkies-Atkin algorithm using Abelian lifts. In *Proceedings of the 2007 international symposium on symbolic and algebraic computation* (pp. 285–292). New York, USA: Association for Computing Machinery. doi: 10.1145/1277548.1277587

Miller, V. (1985). Use of elliptic curves in cryptography. In *Advances in cryptology – CRYPTO '85 proceedings* (Vol. 218, pp. 417–426). Springer. doi: 10.1007/3-540-39799-X_31

Monsky, P. (1968). Formal cohomology: II. the cohomology sequence of a pair. *The Annals of Mathematics*, *88*(2), 218. doi: 10.2307/1970572

Monsky, P., & Washnitzer, G. (1968). Formal cohomology: I. *The Annals of Mathematics*, *88*(2), 181. doi: 10.2307/1970571

Montgomery, P. L. (1987). Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, *48*(177), 243–264. doi: 10.1090/S0025-5718-1987-0866113-7

Morain, F., Scribot, C., & Smith, B. (2016). Computing cardinalities of Q-curve reductions over finite fields. *LMS Journal of Computation and Mathematics*, *19*, 115-129. doi: 10.1112/S1461157016000267

Mordell, L. (1922). On the rational solutions of the indeterminate equations of the 3rd and 4th degrees. *Proc. Cambridge Philos. Soc*, *21*, 179–182.

Müller, V. (1995). *Ein algorithmus zur bestimmung der punktanzahl elliptischer kurven über endlichen körpern der charakteristik größer drei* (Doctoral dissertation). doi: 10.22028/D291-25721

Nath, K., & Sarkar, P. (2018). *Efficient arithmetic in (pseudo-)mersenne prime order fields.* Cryptology ePrint Archive, Report 2018/985. (https://eprint.iacr.org/2018/985)

Nath, K., & Sarkar, P. (2020). *Efficient 4-way vectorizations of the Montgomery ladder.* Cryptology ePrint Archive, Report 2020/378. (https://eprint.iacr.org/2020/378)

Oliveira, T., López, J., Hışıl, H., Faz-Hernández, A., & Rodríguez-Henríquez, F. (2017). How to (pre–)compute a ladder – improving the performance of X25519 and X448. In C. Adams & J. Camenisch (Eds.), *Selected areas in cryptography - SAC 2017 - 24th international conference, ottawa, on, canada, august 16-18, 2017, revised selected papers* (Vol. 10719, pp. 172–191). Springer. Retrieved from https://doi.org/10.1007/978-3-319-72565-9_9 doi: 10.1007/978-3-319-72565-9_9

Pollard, J. M. (1978). Monte Carlo methods for index computation ($mod\ p$). *Mathematics of Computation*, *32*, 918-924. doi: 10.1090/S0025-5718-1978-0491431-9

Renes, J., & Smith, B. (2017). qDSA: small and secure digital signatures with curve-based Diffie-Hellman key pairs. In T. Takagi & T. Peyrin (Eds.), *Advances in cryptology - ASIACRYPT 2017* (Vol. 10625, pp. 273–302). Springer. Retrieved from https://doi.org/10.1007/978-3-319-70697-9 doi: 10.1007/978-3-319-70697-9_10

Rezaeian Farashahi, R., & Hosseini, S. G. (2016). Differential addition on binary elliptic curves. In S. Duquesne & S. Petkova-Nikova (Eds.), *Arithmetic of finite fields* (pp. 21–35). Cham: Springer International Publishing.

Rezaeian Farashahi, R., & Hosseini, S. G. (2017). Differential addition on twisted Edwards curves. In J. Pieprzyk & S. Suriadi (Eds.), *Information security and privacy - 22nd australasian conference, ACISP 2017, auckland, new zealand, july 3-5, 2017, proceedings, part II* (Vol. 10343, pp. 366–378). Springer. Retrieved from https://doi.org/10.1007/978-3-319-59870-3_21 doi: 10.1007/978-3-319-59870-3_21

Roth, R. L. (2001). A history of Lagrange's theorem on groups. *Mathematics Magazine*, *74*(2), 99–108. doi: 10.1080/0025570x.2001.11953045

Satoh, T. (2000). The canonical lift of an ordinary elliptic curve over a finite field and its point counting. *Journal of the Ramanujan Mathematical Society*, *15*(4).

Satoh, T. (2002). On p-adic point counting algorithms for elliptic curves over finite fields. In *Lecture notes in computer science* (pp. 43–66). Springer Berlin Heidelberg. doi: 10.1007/3-540-45455-1_5

Satoh, T., Skjernaa, B., & Taguchi, Y. (2003). Fast computation of canonical lifts of elliptic curves and its application to point counting. *Finite Fields and Their Applications*, *9*(1), 89–101. doi: 10.1016/S1071-5797(02)00013-8

Schoof, R. (1985). Elliptic curve over finite fields and the computation of square roots mod $p$. *Mathematics of Computation*, *44*(170), 483–495. doi: 10.1090/S0025-5718-1985-0777280-6

Schoof, R. (1995). Counting points on elliptic curves over finite fields. *Journal de Théorie des Nombres de Bordeaux*, *7*(1), 219–254. doi: 10.5802/JTNB.142

Shanks, D. (1971). Class number, a theory of factorization, and genera. *Proceedings of Symposium of Pure Mathematics 20*. doi: 10.1090/pspum/020/0316385

Shoup, V. (2009). *A computational introduction to number theory and algebra* (2nd

ed.). USA: Cambridge University Press. doi: 10.5555/1529931

Shoup, V., & Developers, T. N. (1990). *NTL: A Library for doing Number Theory.* Retrieved July 21, 2022, from https://libntl.org.

Silverman, J. H. (1986). *The arithmetic of elliptic curves* (1st ed.). Springer New York. doi: 10.1007/978-1-4757-1920-8

Silverman, J. H. (2009). *The arithmetic of elliptic curves* (2nd ed.). Springer New York. doi: 10.1007/978-0-387-09494-6

Skjernaa, B. (2003). Satoh's algorithm in characteristic 2. *Mathematics of Computation*, *72*(241), 477–487. doi: 10.1090/S0025-5718-02-01434-5

Stankovic, A. (2017). *Schoof-Elkies-Atkin point counting algorithm for cryptographic purposes - Python Sage implementation.* Retrieved July 21, 2022, from http://stankovicaleksa.net/Crypto_paper_Aleksa_Stankovic.pdf.

Sundriyal, S. (2008). *Counting points on elliptic curves over Zp* (Unpublished master's thesis). Rochester Institute of Technology.

Sutherland, A. (2013). On the evaluation of modular polynomials. *The Open Book Series*, *1*, 181. doi: 10.2140/obs.2013.1.531

Takagi, T., & Peyrin, T. (Eds.). (2017). *Advances in cryptology - ASIACRYPT 2017 - 23rd international conference on the theory and applications of cryptology and information security, hong kong, china, december 3-7, 2017, proceedings, part II* (Vol. 10625). Springer. Retrieved from https://doi.org/10.1007/978-3-319-70697-9 doi: 10.1007/978-3-319-70697-9

Tate, J. (1966). Endomorphisms of Abelian varieties over finite fields. *Inventiones mathematicae*, *2*, 134-144. doi: 10.1007/BF01404549

Tolkov, I. (2009). *Counting points on elliptic curves: Hasse's theorem and recent developments.* Retrieved July 21, 2022, from https://sites.math.washington.edu/~morrow/336_09/papers/Igor.pdf.

Vercauteren, F. (2000). *The SEA algorithm in characteristic 2.* Retrieved July 21, 2022, from https://www.esat.kuleuven.be/cosic/publications/article-403.pdf. (Preprint)

Vercauteren, F., Preneel, B., & Vandewalle, J. (2001). A memory efficient version of Satoh's algorithm. In *Advances in cryptology — EUROCRYPT 2001* (pp. 1–13). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-44987-6_1

Washington, L. C. (2008). *Elliptic curves: Number theory and cryptography* (2nd ed.). Chapman & Hall/CRC. doi: 10.1201/9781420071474

Zelenova, M. (2011). Counting the order of the group of points of an elliptic curve over a finite field based on shanks's algorithm. *P-Adic Numbers, Ultrametric Analysis, and Applications*, *3*, 157-164. doi: 10.1134/S2070046611020075

# APPENDIX A
# SUPPLEMENTARY FIGURES

```
1
Checking if E1 : y^2 = x^3 + 1047904962440916948235009230631248113535415073108\
8
6344617622031066326019022 6x + 34387566339114205486748486485854650150635779097\
45
3666141408228557537779511 6 and E2 : y^2 = x^3 +
1021599203925083015807333226582389747814978225314531748224194668351299101835x +
9722666031631654428392146343182834765309965253913214181011450598812292199 61 are
isogenous or not:

CheckIfIsogenous:
3 : Elkies_1
3 : Elkies_2
Traces are different mod 5!
Time: 0.020
IsIsogenous:
Time: 5.950
Curves are not isogenous.

2
Checking if E1 : y^2 = x^3 + 5651839393326220676445654006595261814001298025541
9
2533013978480973826407124x + 7080529686903301974175990175136843223394136755562
8
0796597639155403816708 34 and E2 : y^2 = x^3 +
1124660770401659598804364673107863064884932617785845819475804430398999375557x +
9784418863424087373036977108391048497619202330197080419980374516080079416 84 are
isogenous or not:

CheckIfIsogenous:
Traces are different mod 2!
Time: 0.000
IsIsogenous:
Time: 6.980
Curves are not isogenous.

3
Checking if E1 : y^2 = x^3 + 1660545663755572082186583577167096012892377678182
4
8472643820117230390041143x + 7550391870612564803354527308762515901983114478164
0
2789728569333855777855879 and E2 : y^2 = x^3 +
4059638046382284165286753442206835195721280569859092480651766317293545584 19x +
2251359376882394478588280599760889700242575054401414213325014518844883543 43 are
isogenous or not:

CheckIfIsogenous:
Traces are different mod 3!
Time: 0.000
IsIsogenous:
Time: 7.630
Curves are not isogenous.

4
Checking if E1 : y^2 = x^3 + 1209388565073615435364686125109573422812452471452
6
3845216580840389149501609x + 8405589523164187669023038729143304531536350683251
9
1621286235952781569192 55 and E2 : y^2 = x^3 +
8383876561750764423150203059629167981801647981666740619540037098054797674 5x +
8007864608869852070750224631845923587674132013319896491483025785218889704 2 are
isogenous or not:

CheckIfIsogenous:
Traces are different mod 3!
Time: 0.000
IsIsogenous:
Time: 4.870
Curves are not isogenous.

5
Checking if E1 : y^2 = x^3 + 4383726060896192012404091097831321036926796500839\
7
6757004847704627951126227x + 5047169842486937073033500628415919865136508019193\
8
1004332524971453531579228 and E2 : y^2 = x^3 +
8140902647745049711911203197296444437671403183901783104489758863874654390 59x +
2985955583204633451651171786407272930346153315272827219161149275484821743 93 are
isogenous or not:

CheckIfIsogenous:
3 : Elkies_1
3 : Elkies_2
Traces are different mod 5!
Time: 0.020
IsIsogenous:
Time: 6.490
Curves are not isogenous.
```

**Figure A.1.** The Output of CheckIfIsogenous.mag

# APPENDIX B

# SUPPLEMENTARY CODES

```
schoof := function(a4, a6, P, x, y, _x, q)
  S := []; rm := []; i := 1; N := 1; k := 1; t := 1;
  repeat
    t := NextPrime(t);
    S[i] := t; i +:= 1;
    N *:= t;
  until N gt 4*Sqrt(q);

  // l = 2
  if GCD(Modexp(_x,q,_x^3+a4*_x+a6)-_x,_x^3+a4*_x+a6) ne 1 then
    rm[k] := 0;
  else
    rm[k] := 1;
  end if;
  k +:= 1;

  // l = 3,5,...
  for i := 2 to #S do
    l := S[i]; fl := false;
    phil := divpol(l,_x,a4,a6);
    ql := q mod l;
    Q := quo<P | y^2-x^3-a4*x-a6, divpol(l,x,a4,a6)>;
    Qx := Q!x; Qy := Q!y; Qxq := Q!x^q; Qyq := Q!y^q; Qxqs := Q!x^(q*q); Qyqs := Q!y^(q*q);

    xql := 0*Qx; yql := 0*Qy; xqli := true;
    for j := 1 to ql do
      xql, yql, xqli := AddAffinePointsGeneric(xql, yql, xqli, Qx, Qy, false, a4);
    end for;

    xqt := 0*Qx; yqt := 0*Qy; xqi := true;
    if Qxqs ne xql then
      xd, yd, _ := AddAffinePointsGeneric(Qxqs, Qyqs, false, xql, yql, false, a4);
      for td := 1 to (l-1) div 2 do
        xqt, yqt, xqi := AddAffinePointsGeneric(xqt, yqt, xqi, Qxq, Qyq, false, a4);
        if xd eq xqt then
          if yd eq yqt then
            rm[k] := td;
          else
            rm[k] := -td;
          end if;
          k +:= 1; fl := true;
          break;
        end if;
      end for;
    end if;
    if fl ne true then
      if LegendreSymbol(ql,l) eq 1 then
        w := Modsqrt(ql,l);
        xw := 0*Qx; yw := 0*Qy; xwi := true;
        for j := 1 to w do
          xw, yw, xwi := AddAffinePointsGeneric(xw, yw, xwi, Qx, Qy, false, a4);
        end for;
        if GCD(UnivariatePolynomial(Qxq-xw),phil) ne 1 then
          if GCD(UnivariatePolynomial((Qyq-yw)/yw),phil) ne 1 then
            rm[k] := (2*w) mod l;
          else
            rm[k] := (-2*w) mod l;
          end if;
        else
          rm[k] := 0;
        end if;
      else
        rm[k] := 0;
      end if;
      k +:= 1;
    end if;
  end for;

  t := CRT(rm,S);

  if Abs(t) gt 2*Sqrt(q) then // |t| <= 2*sqrt(q)
    t := t - N;
  end if;

  return q+1-t;
end function;
```

**Code B.1.** Schoof's Algorithm Magma Implementation

```
if atkins gt Log(2,p) then // avoid gathering too much Atkin info
  M := M/ell;
else
  r := FindOrderFrobenius(poly, ell, _x, p);

  d := getNonSquare(ell);

  FF := GF(ell);
  FFF<i> := ExtensionField<FF, x|x^2-d>;
  g := RootOfUnity(r, FFF);

  S := []; kk := 1;
  for i := 1 to r-1 do
    if GCD(i,r) eq 1 then
      S[kk] := g^i; kk +:= 1;
    end if;
  end for;

  Atl := []; tct := 1;
  if #S eq 1 then
    Snum := 1;
  else
    Snum := #S div 2;
  end if;
  for i := 1 to Snum do
    g1 := ElementToSequence(S[i])[1];
    z := (FF!p*(g1+1)/2);
    if LegendreSymbol(Integers()!z, ell) ne -1 then
      sz := Sqrt(z);
      Atl[tct] := Integers()!( 2*sz); tct +:= 1;
      Atl[tct] := Integers()!(-2*sz); tct +:= 1;
    end if;
  end for;
  Atl := Sort(Atl);
  atkins *:= #Atl;
  Ap[ai] := Atl; ai +:= 1;
  Ap[ai] := [Integers()!ell]; ai +:= 1;
end if;
```

**Code B.2.** Atkin's Procedure Magma Implementation

```
jt := Roots(poly)[1][1];

PHI_ell := P!ClassicalModularPolynomial(ell);
assert Evaluate(PHI_ell,[j,jt]) eq 0;

dxPHI_ell := Derivative(PHI_ell,x);
dyPHI_ell := Derivative(PHI_ell,y);

eval_dxPHI_ell := Evaluate(dxPHI_ell,[j,jt]);
eval_dyPHI_ell := Evaluate(dyPHI_ell,[j,jt]);

dxdxPHI_ell := Derivative(dxPHI_ell,x);
eval_dxdxPHI_ell := Evaluate(dxdxPHI_ell,[j,jt]);

dydyPHI_ell := Derivative(dyPHI_ell,y);
eval_dydyPHI_ell := Evaluate(dydyPHI_ell,[j,jt]);

dxdyPHI_ell := Derivative(dyPHI_ell,x);
eval_dxdyPHI_ell := Evaluate(dxdyPHI_ell,[j,jt]);

E4 := -48*a4;
E6 := 864*a6;
jd := -j*E6/E4;

jtd := -jd*eval_dxPHI_ell/(ell*eval_dyPHI_ell);

a4t := -jtd^2/(48*jt*(jt-1728));
a6t := -jtd^3/(864*jt^2*(jt-1728));

E4t := -48*a4t;
E6t := 864*a6t;

assert jtd eq -jt*E6t/E4t;
assert jtd eq (-18*a6*eval_dxPHI_ell*j)/(ell*a4*eval_dyPHI_ell);

assert jd*eval_dxPHI_ell eq -ell*jtd*eval_dyPHI_ell;

J := -(jd^2*eval_dxdxPHI_ell + 2*ell*jd*jtd*eval_dxdyPHI_ell + ell^2*jtd^2*eval_dydyPHI_ell)/(jd
    *eval_dxPHI_ell);
p1 := (ell/2)*J + (ell/4)*(E4^2/E6-ell*E4t^2/E6t) + (ell/3)*(E6/E4-ell*E6t/E4t);

a4h := ell^4*a4t;
a6h := ell^6*a6t;
assert IsIsomorphic(EllipticCurve([0,0,0,a4t,a6t]),EllipticCurve([0,0,0,a4h,a6h])) eq true;

d := (ell-1) div 2;

c := [];
c[1] := -a4/5; c[2] := -a6/7;
for k := 3 to d do
  sum := 0;
  for j := 1 to k-2 do
    sum +:= c[j]*c[k-1-j];
  end for;
  c[k] := (3/((k-2)*(2*k+3)))*sum;
end for;
```

```
56
57 ch := [];
58 ch[1] := -a4h/5; ch[2] := -a6h/7;
59 for k := 3 to d do
60   sum := 0;
61   for j := 1 to k-2 do
62     sum +:= ch[j]*ch[k-1-j];
63   end for;
64   ch[k] := (3/((k-2)*(2*k+3)))*sum;
65 end for;
66
67 L<w> := LaurentSeriesRing(Parent(a4));
68
69 sum := 0;
70 for k := 1 to d do
71   sum +:= ((ch[k]-ell*c[k])/((2*k+1)*(2*k+2)))*w^(k+1);
72 end for;
73 Aw := Exp((-p1/2)*w-sum);
74
75 Cw := 0;
76 for k := 1 to d do
77   Cw +:= c[k]*w^k;
78 end for;
79
80 Fl := []; Fl[d+1] := 1;
81 for i := 1 to d do
82   sum2 := 0;
83   for k := 1 to i do
84     sum1 := 0;
85     for j := 0 to k do
86       num := d-i+k;
87       den := k-j;
88       comb := Factorial(num)/(Factorial(den)*Factorial(num-den));
89       if k-j eq 0 then
90         if j eq 0 then
91           sum1 +:= comb;
92         end if;
93       else
94         sum1 +:= comb*Coefficient(Cw^(k-j), j);
95       end if;
96     end for;
97     sum2 +:= sum1*Fl[d-i+k+1];
98   end for;
99   Fl[d-i+1] := Coefficients(Aw)[i+1]-sum2;
100 end for;
101
102 kpol := 0;
103 for i := #Fl to 1 by -1 do
104   kpol +:= Fl[i]*x^(i-1);
105 end for;
106
107 Q := quo<P | y^2-x^3-a4*x-a6, kpol>;
108 Qx := Q!x; Qy := Q!y; xk := 0*Qx; yk := 0*Qy;
109 Qxp := Q!x^p; Qyp := Q!y^p; xki := true;
110 for k := 1 to d do
111   xk, yk, xki := AddAffinePointsGeneric(xk, yk, xki, Qx, Qy, false, a4);
112   if xk eq Qxp then
113     if yk eq Qyp then
114       lmd := k;
115     else
116       lmd := -k;
117     end if;
118     FF := GF(ell);
119     t := FF!lmd + FF!p/FF!lmd;
120     Ep[ei] := [Integers()!t,Integers()!ell]; ei +:= 1;
121     break;
122   end if;
123 end for;
```

**Code B.3.** Elkies' Procedure Magma Implementation