Elliptic curve cryptography

by

Gerard Jacques Louw

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Mathematics in the Faculty of Science at Stellenbosch University

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Supervisor: Prof. F. Breuer

October 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ....

Abstract

Elliptic curve cryptography

G.J. Louw

Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, 7602 Matieland, South Africa.

Thesis: MSc (Mathematics)

October 2016

In this thesis we present a selection of Diffie-Hellman cryptosystems, which were classically formulated using the multiplicative group of a finite field, but which may be generalised to use other group varieties such as elliptic curves. We also describe known attacks on special cases of such cryptosystems, which manifest as solutions to the discrete logarithm problem for group varieties, and the elliptic curve discrete logarithm problem in particular. We pursue a computational approach throughout, with a focus on the development of practical algorithms.

Acknowledgements

I would like to thank my wife, Andrea, and my parents, Nelmarie and Jacques, for their interest in my research, as well as their support, which was unwavering during the most challenging times of my studies. I also wish to thank my supervisor, Florian, for his suggestion of such a fascinating research topic, and for his invaluable advice and guidance, which kept me focused on my goals.

I acknowledge the financial contributions of the National Research Foundation, the Harry Crossley Foundation and the MIH Media Lab to my research. I also thank the MIH Media Lab for providing me with a stimulating environment in which much of my research was conducted.

Contents

List of Figures

List of Algorithms

Notation

Objects

K	a perfect field of characteristic not 2 or 3
L	an algebraic extension of K
$\bar{K}$	a fixed algebraic closure of K
p	a prime number greater than 3
q	a positive power of p
$\mathbb{F}_q$	a fixed finite field of order q
$O(f(n))$	the set of functions $\left\{g(n) : \liminf\limits_{n\to\infty} \frac{g(n)}{f(n)} \geq 0 \text{ and } \limsup\limits_{n\to\infty} \frac{g(n)}{f(n)} < \infty\right\}$
$o(f(n))$	the set of functions $\left\{g(n) : \lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = 0\right\}$
$\Theta(f(n))$	the set of functions $\left\{g(n) : \liminf\limits_{n\to\infty} \frac{g(n)}{f(n)} > 0 \text{ and } \limsup\limits_{n\to\infty} \frac{g(n)}{f(n)} < \infty\right\}$
$\tilde{O}(f(n))$	the set of functions $\bigcup\limits_{k=0}^{\infty} O(f(n)(\log f(n))^k)$
$L_n[\alpha, c]$	the set of functions $\left\{e^{(c+g(n))(\log n)^{\alpha}(\log\log n)^{1-\alpha}} : g(n) \in o(1)\right\}$

Chapter 1

Introduction

Elliptic curves are among the simplest examples of group varieties – algebro-geometric objects which may be adorned with a group structure. Their theory has been fruitfully applied to important problems in number theory, and they have fairly recently found a practical application in the field of cryptography.

The first cryptosystem based on the so-called Diffie-Hellman problem on group varieties was introduced in (Diffie and Hellman, 1976). Many other authors were subsequently inspired to propose cryptosystems based on the same problem, thereby solving a variety of problems in public-key cryptography. These cryptosystems were classically formulated using the multiplicative group $\mathbb{F}_q^\times$, which may be viewed as the affine plane curve $xy = 1$ over $\mathbb{F}_q$. Replacing this group variety with an elliptic curve yields cryptosystems which are more secure than their classic counterparts, thus making this choice very popular in recent years.

In this thesis we present the background on elliptic curves needed to discuss elliptic curve cryptography, and we formulate the Diffie-Hellman problem on elliptic curves, as well as the related discrete logarithm problem. We then give an exposition of known solutions to the discrete logarithm problem, of which some work for general group varieties, while others are specific to special classes of elliptic curves. Finally, we describe a selection of public-key cryptosystems based on the Diffie-Hellman and discrete logarithm problems on elliptic curves.

For the discussion on elliptic curves, we assume a familiarity of basic algebraic geometry. In the analysis of various algorithms throughout the thesis, a knowledge of the time and space complexities of finite field arithmetic is also assumed.

Chapter 2

Elliptic curves

In this chapter, we cover the background on elliptic curves needed throughout the rest of the thesis. Our exposition borrows heavily from the books (Silverman, 2009; Washington, 2008), as well as the lecture notes (Sutherland, 2015).

2.1 Weierstrass equations

2.1.1 Weierstrass equations and elliptic curves

Definition 2.1. A *Weierstrass equation* over K is an equation of the form

$$y^2 = x^3 + a_4 x + a_6,$$

where $a_4, a_6 \in K$. Sometimes we will abbreviate $f(x) := x^3 + a_4 x + a_6$. An *elliptic curve (in Weierstrass form)* over K is the projective closure of a smooth affine plane curve cut out by a Weierstrass equation.

It is clear that the projective point $\mathcal{O} := (0 : 1 : 0)$ is the only point at infinity on an elliptic curve, and it is therefore referred to as *the point at infinity.* This fact allows us to deal with elliptic curves as affine curves, treating the point at infinity separately. Checking on the affine patch where $x = 0$ easily shows that the point at infinity is always smooth, so that elliptic curves are smooth projective curves.

2.1.2 Discriminant

Definition 2.2. The *discriminant* of a Weierstrass equation is the quantity

$$\Delta := -16(4a_4^3 + 27a_6^2)$$

The following theorem gives us a simple criterion for determining whether an arbitrary Weierstrass equation defines an elliptic curve.

Theorem 2.1. *The affine plane curve cut out by a Weierstrass equation is smooth if and only if $\Delta \neq 0$.*

This immediately suggests a probabilistic algorithm of Las Vegas type for randomly sampling from the elliptic curves over a finite field $\mathbb{F}_q$: select each of the coefficients in a Weierstrass equation uniformly at random from $\mathbb{F}_q$, until a combination of coefficients is found which results in a non-zero discriminant. Sage code implementing this approach is given in Algorithm 2.1.

Algorithm 2.1 Random elliptic curve

```
def random_elliptic_curve(K):
    """
    K: a field
    """
    while true:
        a_4, a_6 = random_vector(K, 2)
        if -16 * (4 * a_4^3 + 27 * a_6^2) != 0:
            return EllipticCurve([a_4, a_6])
```

2.2 Group structure

Central to the utility of elliptic curves in cryptography is the fact that the L-points on an elliptic curve may be endowed with an abelian group structure which allows efficient computation. The following theorem describes this group structure.

Theorem 2.2. *Let E be an elliptic curve over K in Weierstrass form. The points in $E(L)$ form an abelian group with $\mathcal{O}$ as its identity element, negation* *of an affine point $P \in E(L)$ defined by*

$$-P := (x_P, -y_P)$$

and addition of two affine points $P, Q \in E(L)$ such that $P \neq -Q$ defined by

$$P + Q := (m_{P,Q}^2 - x_P - x_Q, -m_{P,Q}x_{P+Q} - c_{P,Q}),$$

where

$$m_{P,Q} := \begin{cases} (3x_P^2 + a_4)/2y_P & \text{if } P = Q \\ (y_Q - y_P)/(x_Q - x_P) & \text{if } P \neq Q \end{cases}$$

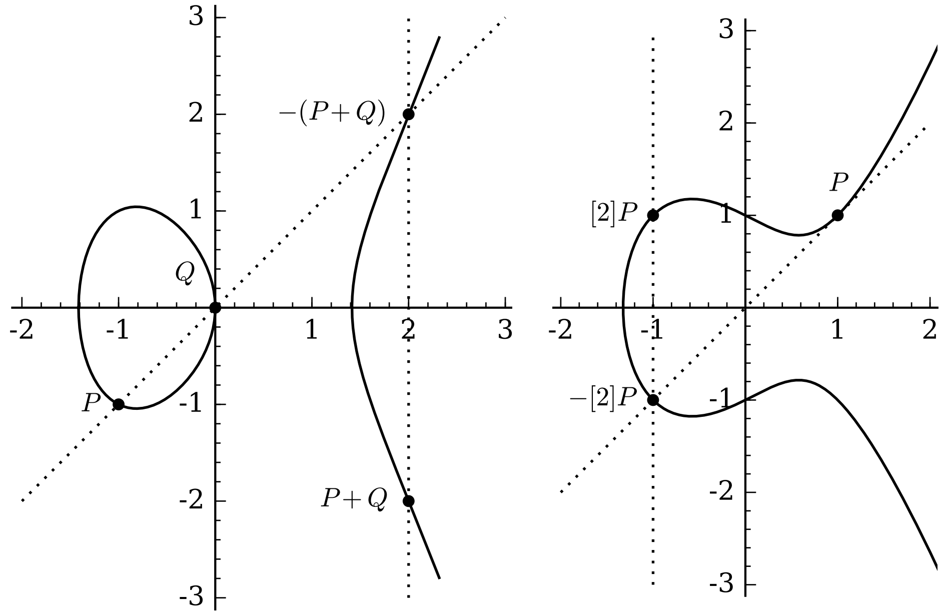
and

$$c_{P,Q} := \begin{cases} (-x_P^3 + a_4 x_P + 2a_6)/2y_P & \text{if } P = Q \\ (x_Q y_P - x_P y_Q)/(x_Q - x_P) & \text{if } P \neq Q. \end{cases}$$

It is clear from the formulae in Theorem 2.2 that addition in $E(L)$ is commutative, and it is straightforward to verify that $E(L)$ is closed under negation and addition by plugging the respective formulae into the elliptic curve's Weierstrass equation. However, it is somewhat tedious to demonstrate that addition in $E(L)$ is associative. We refer the reader to Section 2.4 of (Washington, 2008) for a proof of associativity.

It can also be shown that the maps $- : E \to E$ and $+ : E \times_K E \to E$ are K-morphisms, so that elliptic curves are in fact abelian varieties since $\mathcal{O}$ is furthermore a K-point.

The group structure of an elliptic curve has a useful geometric interpretation. By Bézout's theorem, the line passing through two L-points of a cubic projective plane curve will meet the curve in exactly one other L-point, when counting with multiplicity. Thus, the line passing through a point $P \in E(L)$ and $\mathcal{O}$ meets E at a unique L-point, which is the point $-P$. The quantities $m_{P,Q}$ and $c_{P,Q}$ in Theorem 2.2 are respectively the slope and the intercept of the line passing through the points P and Q, which meets E at the point $-(P + Q)$, so that the sum $P + Q$ is the third intersection point of E and the line passing through $-(P + Q)$ and $\mathcal{O}$. Figure 2.1 depicts the geometric interpretation of group addition on elliptic curves over $\mathbb{Q}$ for both the cases $P = Q$ and $P \neq Q$.



(a) $E : y^2 = x^3 - 2x$, $P = (-1, -1)$, $Q = (0, 0)$

(b) $E : y^2 = x^3 - x + 1$, $P = (1, 1)$

Figure 2.1: Addition on elliptic curves over $\mathbb{Q}$

There is another important interpretation of the group structure on an elliptic curve. It can be shown that there is an isomorphism of abelian varieties

$$E \to J_E, \quad P \mapsto (P) - (\mathcal{O}),$$

where $J_E := \operatorname{Div}^0(E)/\operatorname{Prin}(E)$ denotes the *Jacobian variety* of E.

2.3 Functions and morphisms

2.3.1 Regular functions

Any *regular function* α in the *coordinate ring* $L[E] := L[x, y]/(y^2 - f)$ over L of an elliptic curve E may be put into a simple canonical form. Given a polynomial representative in $L[x, y]$ of α, we may substitute the factor y^{2k}, where $k \in \mathbb{N}_0$ is maximal, with f^k in every term which contains such a factor, yielding a polynomial of the form $\alpha_1 + \alpha_2 y$, where $\alpha_1, \alpha_2 \in L[x]$. Such a polynomial must be the unique representative for α of this form, since any further substitutions using the Weierstrass equation will introduce a factor y^2 in some term.

For any $\alpha \in L[E]$, its *conjugate* $\bar{\alpha}(x,y) := \alpha(x,-y)$ is also a regular function, and the canonical form of its *norm* $N(\alpha) := \alpha\bar{\alpha}$ is a polynomial in x only, since

$$(\alpha_1 + \alpha_2 y)(\alpha_1 - \alpha_2 y) = \alpha_1^2 - \alpha_2^2 y^2 = \alpha_1^2 - \alpha_2^2 f.$$

2.3.2 Rational functions

A *rational function* α in the *function field* $L(E) := \mathrm{Frac}(L[E])$ of E may also be put into a canonical form. If $\alpha = \beta/\gamma$, where $\beta, \gamma \in L[E]$ are in canonical form, then $\alpha = \beta\bar{\gamma}/N(\gamma)$, which can be written in the general form $\alpha_1 + \alpha_2 y$, where $\alpha_1, \alpha_2 \in L(x)$.

The conjugate $\bar{\alpha}$ and norm $N(\alpha)$ of a rational function are defined similarly to the regular function case, and by the same line of reasoning we may see that the canonical representation of $N(\alpha)$ is an element of $L(x)$.

2.3.3 Morphisms

Let $E' : y^2 = x^3 + a_4'x + a_6'$ be another elliptic curve over K. A non-zero *morphism* of group varieties $\phi := (\alpha, \beta) \in \mathrm{Mor}(E, E')$, where $\alpha, \beta \in \bar{K}(E)$, may also be put into a canonical form as follows. Note that by the definition of negation $-\phi = (\alpha, -\beta)$, but since ϕ is a morphism of groups, we also have $-\phi = (\bar{\alpha}, \bar{\beta})$. It follows that $\alpha = \bar{\alpha}$, so that $\alpha \in \bar{K}(x)$ and $-\gamma = \bar{\gamma}$, so that $\gamma \in \bar{K}(x)y$. The canonical form of ϕ is then $(\phi_x, \phi_y y)$, where $\phi_x, \phi_y \in \bar{K}(x)$.

Lemma 2.1. *Let $\phi := (\phi_x, \phi_y y) \in \mathrm{Mor}(E, E')$ be a morphism in canonical form. Then the equation $\phi_y^2 f = \phi_x^3 + a_4'\phi_x + a_6'$ in $\bar{K}(x)$ is satisfied.*

Proof. Substituting $(\phi_x, \phi_y y)$ into the Weierstrass equation for E' and replacing y^2 with f yields the result. □

2.3.4 Isogenies

Definition 2.3. An *isogeny* is an epimorphism of group varieties with a finite kernel. The *degree* of an isogeny $\phi : E \to E'$ is defined as $\deg \phi := [\bar{K}(E) :$

$\phi^*\bar{K}(E')]$, where $\phi^*\alpha := \alpha\phi$ for $\alpha \in \bar{K}(E')$, and ϕ is said to be *separable* if and only if the extension $\bar{K}(E)/\phi^*\bar{K}(E')$ is separable.

From the well known fact every morphism of projective curves is either an epimorphism or constant, it follows that the zero morphism $(0 : 1 : 0)$ is the only morphism of elliptic curves as group varieties which is not an isogeny. For convenience, one may sometimes wish to consider the zero morphism as an isogeny of degree zero.

Theorem 2.3. *If $\phi : E \to E'$ is an isogeny, then $\# \ker \phi \mid \deg \phi$, with $\# \ker \phi = \deg \phi$ if and only if ϕ is separable.*

Proof. See Theorem III.4.10 of (Silverman, 2009). □

The following theorem gives some useful properties for isogenies which have been put into canonical form.

Theorem 2.4. *Let $\phi := (\phi_x, \phi_y y) \in \mathrm{Mor}(E, E')$ be an isogeny in canonical form, where $\phi_x := \phi_{x;1}/\phi_{x;2}$ for some relatively prime $\phi_{x;1}, \phi_{x;2} \in \bar{K}[x]$. Then*

(a) $(x, y) \in \ker \phi$ if and only if $\phi_{x;2}(x) = 0$;

(b) $\deg \phi = \max(\deg \phi_{x;1}, \deg \phi_{x;2})$; and

(c) ϕ is inseparable if and only if $p := \mathrm{char}(K)$ is prime and $\phi_x \in \bar{K}(x^p)$.

Proof. (a) See Corollary 5.23 of (Sutherland, 2015).

(b) See Lemma 9.6.13 of (Galbraith, 2012).

(c) See Corollary II.2.12 of (Silverman, 2009), which along with Lemma 2.1 implies the result.

□

Theorem 2.5. *Let $\phi, \chi \in \mathrm{Mor}(E, E')$ be isogenies with ϕ inseparable. Then $\phi + \chi$ is separable if and only if χ is separable.*

Proof. If χ is inseparable, then $\phi_x, \chi_x \in \bar{K}(x^p)$ by Theorem 2.4, and so $\phi_y^2 f, \chi_y^2 f \in \bar{K}(x^p)$ by Lemma 2.1. Then $(\phi + \chi)_x = m_{\phi,\chi}^2 - \phi_x - \chi_x$, where

$$m_{\phi,\chi} = \begin{cases} (3\phi_x^2 + a_4)/2\phi_y y & \text{if } \phi = \chi \\ (\chi_y - \phi_y)y/(\chi_x - \phi_x) & \text{if } \phi \neq \chi, \end{cases}$$

so that $m^2_{\phi,\chi} \in \bar{K}(x^p)$ in either case, and thus $\phi + \chi$ is inseparable by Theorem 2.4.

If $\phi + \chi$ is inseparable, then $\chi = (\phi + \chi) - \phi$ is inseperable by the result we have just demonstrated, since $-\phi$ is trivially also inseparable. □

2.4 Torsion

2.4.1 Multiplication-by-m endomorphisms

Definition 2.4. For $m \in \mathbb{Z}$, the *multiplication-by-m endomorphism* of E is defined recursively as

$$[m] := \begin{cases} (0 : 1 : 0) & \text{if } m = 0 \\ (x, y) & \text{if } m = 1 \\ [m-1] + [1] & \text{if } m > 1 \\ -[-m] & \text{if } m < 0 \end{cases}$$

Note that $\ker[m] = E[m]$, the *m-torsion subgroup* of E. For a point $P \in E(L)$, the point multiplication $[m]P$ may be calculated with a time complexity of $\Theta(\log m)$ operations in L using a double-and-add algorithm.

2.4.2 Division polynomials

Definition 2.5. For $m \in \mathbb{Z}$ the *m-th division polynomial* of E is the regular function $\psi_m \in K[E]$ defined recursively as (Washington, 2008)

$$\begin{aligned}
\psi_0 &:= 0 \\
\psi_1 &:= 1 \\
\psi_2 &:= 2y \\
\psi_3 &:= 3x^4 + 6a_4x^2 + 12a_6x - a_4^2 \\
\psi_4 &:= 4y(x^6 + 5a_4x^5 + 20a_6x^3 - 5a_4^2x^2 - 4a_4a_6x - a_4^3 - 8a_6^2) \\
\psi_{2m-1} &:= \psi_{m-1}^3\psi_{m+1} - \psi_{m-2}\psi_m^3 \text{ for } m \geq 3 \\
\psi_{2m} &:= \frac{\psi_m}{2y}(\psi_{m-1}^2\psi_{m+2} - \psi_{m-2}\psi_{m+1}^2) \text{ for } m \geq 3 \\
\psi_m &:= -\psi_{-m} \text{ for } m < 0.
\end{aligned}$$

Putting ψ_m in canonical form, a straightforward inductive argument shows that if m is odd then $\psi_m \in K[x]$ has the leading term $mx^{(m^2-1)/2}$, and if m is even then $\psi_m/y \in K[x]$ has the leading term $mx^{(m^2-4)/2}$.

The division polynomials satisfy the following essential property.

Theorem 2.6. *For $m \neq 0$, the multiplication-by-m endomorphism has the form*

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

There exist completely elementary inductive proofs of this statement involving messy calculations. A neater proof is given in Theorem 9.33 of (Washington, 2008), which involves first demonstrating the result in characteristic zero using complex analytic techniques, then extending it to arbitrary fields by showing that it is preserved by reduction to characteristic p.

Furthermore, one may show that $x\psi_m^2 - \psi_{m-1}\psi_{m+1}$ and ψ_m^2 are relatively prime when reduced to their canonical representations in $K[x]$. This allows us to apply the criteria in Theorem 2.4 to this representation of $[m]$, immediately yielding the following corollary.

Corollary 2.1. *If $\operatorname{char}(K) \nmid m$, then $\psi_m(x, y) = 0$ if and only if $(x, y) \in E[m]$.*

Furthermore, using the representation in Theorem 2.6 allows us to determine the degree of $[m]$, and under certain conditions decide whether it is separable.

Theorem 2.7. *The multiplication-by-m endomorphism has degree m^2, and if $\operatorname{char}(K) \nmid m$ then $[m]$ is separable.*

Proof. This follows easily from Theorem 2.4. Each term of the numerator $x\psi_m^2 - \psi_{m-1}\psi_{m+1}$ of $[m]_x$ is of degree m^2, with leading coefficients m^2 and $m^2 - 1$ respectively, so that the numerator is monic of degree m^2. The leading term of the denominator ψ_m^2 of $[m]_x$ is $m^2x^{m^2-1}$, and hence $\deg[m] = m^2$. If $\operatorname{char}(K) = p$ and $p \nmid m$, then clearly $[m]_x \notin K(x^p)$, since the leading term of its numerator is x^{m^2}, so that $[m]$ is separable. □

Remark 2.1. The converse of the separability criterion in Theorem 2.7 is also true, but we will not need this result.

2.4.3 Structure of m-torsion subgroups

Theorem 2.8. *If* $\operatorname{char}(K) \nmid m$ *then the* m*-torsion subgroup* $E[m]$ *of* E *is a free* $\mathbb{Z}/m\mathbb{Z}$*-module of rank two.*

Proof. We give a distilled version of the proof to Theorem 3.2 of (Washington, 2008). Since $E[m] = \ker[m]$ is a finite abelian group of order $\deg[m] = m^2$, there is an isomorphism

$$E[m] \cong \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z}$$

for some $m_1, \ldots, m_k \in \mathbb{N}_0$ such that $m_1 \neq 1$, $m_1 \mid \cdots \mid m_k$ and $m_1 \cdots m_k = m^2$. Since $m_1 \mid m_i$ for $i = 1, \ldots, k$, $E[m]$ contains a subgroup isomorphic to $(\mathbb{Z}/m_1\mathbb{Z})^k$, which forces $k \leq 2$, since $\#E[m_1] = m_1^2$. Furthermore $m_k \mid m$ since $E[m]$ comprises elements of order dividing m. It follows that $m_1 = m_2 = m$, so that $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$. □

At this point, counting roots shows that canonical representation of ψ_m is in fact the polynomial in $K[x, y]$ of minimal degree with the property that its zeros are the affine m-torsion points, since each root of ψ_m (when m is even) or ψ_m/y (when m is odd) must be the x-coordinate of a pair of points in $E[m]$, while the factor y accounts for the three points of order two when m is even.

Corollary 2.2. *If* $K = \mathbb{F}_q$ *is a finite field of characteristic not dividing* $\#E(\mathbb{F}_q)$*, then* $E(\mathbb{F}_q) = \langle P, Q \rangle$ *for some* $P, Q \in E(\mathbb{F}_q)$*.*

Proof. Let $n := \#E(\mathbb{F}_q)$. Note that $E(\mathbb{F}_q) \leq E[n]$, so that the result follows immediately from Theorem 2.8, since $E(\mathbb{F}_q) \cong \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}$ for some positive integers n_1 and n_2 such that $n_1 n_2 = n$. □

2.5 The dual isogeny

The following theorem establishes the existence of a special isogeny related to any isogeny of elliptic curves, and holds more generally for abelian varieties.

Theorem 2.9. *Let* $\phi : E \to E'$ *be an isogeny, then there is a unique isogeny* $\hat{\phi} : E' \to E$*, called the* dual isogeny *of* ϕ*, such that* $\hat{\phi}\phi = [\deg \phi]$*.*

If $\phi : E \to E'$ is an isogeny, then we say that E and E' are *isogenous*. The existence of the dual isogeny shows that the property of being isogenous is an equivalence relation on elliptic curves, which may be viewed as a generalisation of isomorphism, since isomorphisms are isogenies of degree one.

Theorem 2.10. *Let $\phi, \chi \in \mathrm{Mor}(E, E')$ and $\psi : E' \to E''$ be isogenies, and let $m \in \mathbb{Z}$. Dual isogenies satisfy the following properties:*

(a) $\widehat{[m]} = [m]$*;*

(b) $\widehat{\psi\phi} = \hat{\phi}\hat{\psi}$*;*

(c) $\widehat{\phi + \chi} = \hat{\phi} + \hat{\chi}$*;*

(d) $\deg \hat{\phi} = \deg \phi$*;*

(e) $\hat{\hat{\phi}} = \phi$*; and*

(f) $\phi\hat{\phi} = [\deg \phi]$.

Proof. We present the proof given to Theorem III.6.2 of (Silverman, 2009).

(a) Clearly $[m][m] = [m^2] = [\deg m]$, so that the statement follows by uniqueness of the dual isogeny.

(b) $\hat{\phi}\hat{\psi}\psi\phi = \hat{\phi}[\deg \psi]\phi = [\deg \psi]\hat{\phi}\phi = [\deg \psi][\deg \phi] = [\deg(\psi\phi)]$, from which the statement follows by uniqueness.

(c) Refer to Theorem III.6.2(b) of (Silverman, 2009) for a proof.

(d) $\deg \hat{\phi} \deg \phi = \deg(\hat{\phi}\phi) = \deg([\deg \phi]) = (\deg \phi)^2$, so that $\deg \hat{\phi} = \deg \phi$.

(e) $\phi\hat{\phi}\phi = \phi[\deg \phi] = [\deg \phi]\phi$, so that $\phi\hat{\phi} = [\deg \phi] = [\deg \hat{\phi}]$, where right cancellation is possible because ϕ is an epimorphism. By uniqueness of the dual isogeny, it follows that $\hat{\hat{\phi}} = \phi$.

(f) Using (e) and (d), $\phi\hat{\phi} = \hat{\hat{\phi}}\hat{\phi} = [\deg \hat{\phi}] = [\deg \phi]$.

□

Definition 2.6. The *trace* of an isogeny $\phi \in \mathrm{End}(E)$ is the integer

$$\operatorname{tr} \phi := 1 + \deg \phi - \deg([1] - \phi).$$

It is easy to show that $\operatorname{tr}\hat{\phi} = \operatorname{tr}\phi$ using the properties of the dual isogeny.

Lemma 2.2. *Let $\phi \in \operatorname{End}(E)$ be a non-zero endomorphism. Then*

$$\hat{\phi} + \phi = [\operatorname{tr}\phi].$$

Proof.

$$\begin{aligned}[\deg([1]-\phi)] &= (\widehat{[1]-\phi})([1]-\phi)\\ &= ([1]-\hat{\phi})([1]-\phi)\\ &= [1] - (\hat{\phi}+\phi) + [\deg\phi].\end{aligned}$$

□

Theorem 2.11. *Let $\phi \in \operatorname{End}(E)$ be a non-zero endomorphism. Then ϕ satisfies the* characteristic equation

$$\phi^2 - [\operatorname{tr}\phi]\phi + [\deg\phi] = [0].$$

Proof. $[\operatorname{tr}\phi]\phi = \hat{\phi}\phi + \phi^2 = [\deg\phi] + \phi^2$. □

2.6 The Frobenius endomorphism

Definition 2.7. The *Frobenius endomorphism* of an elliptic curve $E/\mathbb{F}_q$ is defined as

$$\phi_q := (x^q, y^q),$$

which has the canonical form $(x^q, f^{\frac{q-1}{2}}y)$.

To see that ϕ_q is indeed an endomorphism, note that $\phi_q(-P) = -\phi_q(P)$ for all $P \in E$ since q is odd, and furthermore

$$\begin{aligned}\phi_q(P+Q) &= (m_{P,Q}^{2q} - x_P^q - x_Q^q, -m_{P,Q}^q x_{P+Q}^q - c_{P,Q}^q)\\ &= (m_{\phi_q P,\phi_q Q}^2 - x_{\phi_q P} - x_{\phi_q Q}, -m_{\phi_q P,\phi_q Q} x_{\phi_q(P+Q)} - c_{\phi_q P,\phi_q Q})\\ &= \phi_q P + \phi_q Q,\end{aligned}$$

where $m_{P,Q}^q = m_{\phi P,\phi Q}$ and $c_{P,Q}^q = c_{\phi P,\phi Q}$ since these are expressions involving only the coordinates of P and Q, and elements of $\mathbb{F}_q$ which are fixed under exponentiation by q.

Theorem 2.12. *The Frobenius endomorphism ϕ_q is an inseparable isogeny of degree q.*

Proof. Since $\phi_{q;x} = x^q \in \mathbb{F}_q(x^p)$, this follows from Theorem 2.4. □

Theorem 2.13. $[1] - \phi_q$ *is a separable isogeny of degree* $\#E(\mathbb{F}_q)$.

Proof. Since $[1]$ is separable by Theorem 2.7 and ϕ_q is inseparable by Theorem 2.12, the separability of $[1] - \phi_q$ follows from Theorem 2.5. Furthermore, $([1] - \phi_q)(x, y) = \mathcal{O}$ if and only if $x = x^q$ and $y = y^q$, which is equivalent to $x, y \in \mathbb{F}_q$, so that $\ker([1] - \phi_q) = E(\mathbb{F}_q)$. It then follows from Theorem 2.3 that $\deg([1] - \phi_q) = \#E(\mathbb{F}_q)$. □

2.7 Hasse's theorem

Theorem 2.14. *Let $E/\mathbb{F}_q$ be an elliptic curve, then*

$$\#E(\mathbb{F}_q) = q + 1 - \operatorname{tr} \phi_q.$$

Proof. From Definition 2.6 we have

$$\operatorname{tr} \phi_q := 1 + \deg \phi_q - \deg([1] - \phi_q) = 1 + q - \#E(\mathbb{F}_q),$$

since $\deg([1] - \phi_q) = \#E(\mathbb{F}_q)$ by Theorem 2.13. □

Theorem 2.15 (Hasse)**.** *Let $E/\mathbb{F}_q$ be an elliptic curve, then*

$$|\operatorname{tr} \phi_q| \leq 2\sqrt{q}.$$

Proof. Let $t := \operatorname{tr} \phi_q$ and recall $\deg \phi_q = q$ from Theorem 2.12. Then

$$\begin{aligned}([t]\hat{\phi}_q - [2q])([t]\phi_q - [2q]) &= [t]^2\hat{\phi}_q\phi_q - [t][2q](\hat{\phi}_q + \phi_q) + [2q]^2 \\ &= [t]^2[q] - [t]^2[2q] + [2q]^2,\end{aligned}$$

so that $\deg([t]\phi_q - [2q]) = 4q^2 - t^2 q = q(4q - t^2)$. Since degrees are non-negative, this means $4q - t^2 \geq 0$, or equivalently $|t| \leq \sqrt{2q}$. □

As an immediate consequence of Hasse's theorem, we have the asymptotic relationship $\#E(\mathbb{F}_q) \sim q$. This will be useful when studying the time complexity of certain algorithms for elliptic curves.

It can be shown that the bound given by Hasse's theorem is tight. In fact, for each $t \in \mathbb{Z}$ such that $|t| \leq 2\sqrt{q}$, there exists an elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q + 1 - t$.

Chapter 3

Pairings

In this chapter, we introduce pairings – that is, maps which are bilinear and non-degenerate – on certain groups related to elliptic curve groups, as well as presenting algorithms for computing these pairings. Our exposition is based on those of (Galbraith, 2012; Washington, 2008).

3.1 Miller's algorithm

Definition 3.1. Let $P \in E(L)[m]$ where $\operatorname{char}(K) \nmid m$. The *$i$-th Miller function at P* is the rational function $f_{P;i} \in L(E)$ defined recursively by $f_{P;0} := f_{P;1} := 1$ and

$$f_{P;i} := f_{P;j} f_{P;i-j} \frac{\ell_{[j]P,[i-j]P}}{\ell_{[i]P,\mathcal{O}}}$$

for $1 < i \leq m$ and $1 \leq j < i$, where $\ell_{P,Q}$ is the rational function in $L(E)$ corresponding to the line passing through P and $Q \in E(L)[m]$ defined as

$$\ell_{P,Q} := \begin{cases} 1 & \text{if } P = Q = \mathcal{O} \\ x - x_Q & \text{if } P = \mathcal{O} \text{ and } Q \neq \mathcal{O} \\ x - x_P & \text{if } P \neq \mathcal{O} \text{ and } Q \in \{\mathcal{O}, -Q\} \\ y - (m_{P,Q}x + c_{P,Q}) & \text{otherwise,} \end{cases}$$

where $m_{P,Q}$ and $c_{P,Q}$ are respectively the slope and the tangent of this line as in Theorem 2.2.

It remains to be shown that $f_{P;i}$ is a well-defined rational function, since j may take on multiple values in the recursive definition. Since the rational function with a given divisor is unique up to multiplication by a constant, this can be seen from the following theorem by checking inductively that

$$\left((x/y)^{-\operatorname{ord}_{\mathcal{O}}(f_{P;i})} f_{P;i}\right)(\mathcal{O}) = 1$$

for $i = 0, \ldots, m$, independant of the choices for j.

Theorem 3.1. *The i-th Miller function at a point $P \in E(L)[m]$ has the divisor* $\operatorname{div} f_{P;i} = i(P) - ([i]P) - (i-1)(\mathcal{O})$*, and in particular* $\operatorname{div} f_{P;m} = m(P) - m(\mathcal{O})$*.*

Proof. The result clearly holds for $f_{P;0}$ and $f_{P;1}$. Let $Q \in E(L)[m]$ and observe that

$$\operatorname{div} \ell_{P,Q} = (P) + (Q) + (-(P+Q)) - 3(\mathcal{O}),$$

and in particular

$$\operatorname{div} \ell_{P,\mathcal{O}} = (P) + (-P) - 2(\mathcal{O}),$$

so that

$$\operatorname{div} \frac{\ell_{P,Q}}{\ell_{P+Q,\mathcal{O}}} = (P) + (Q) - (P+Q) - (\mathcal{O}).$$

Assuming that the result holds for $f_{P;j}$ and $f_{P;i-j}$ where $1 \le j < i$, we have

$$\begin{aligned}\operatorname{div} f_{P;i} = {} & (j + (i-j))(P) - ([j]P) - ([i-j]P) - (j + (i-j) - 2)(\mathcal{O}) \\ & + ([j]P) + ([i-j]P) - ([j + (i-j)]P) - (\mathcal{O}),\end{aligned}$$

so that

$$\operatorname{div} f_{P;i} = i(P) - ([i]P) - (i-1)(\mathcal{O})$$

as required. The result follows by induction, where $\operatorname{div} f_{P;m} = m(P) - m(\mathcal{O})$ since $[m]P = \mathcal{O}$. □

We now turn the problem of computing the m-th Miller function $f_{P;m}$. One can give a heuristic argument that the maximum degree of the polynomials in the canonical representation of $f_{P;i}$ is linear in i, so that directly computing the function $f_{P;m}$ would have at least linear time complexity in m. In practice, when working with an elliptic curve $E/\mathbb{F}_q$, the m-torsion subgroups we are interested in will have m roughly as large as q, thus yielding such a computation infeasible for large q.

However, we may evaluate $f_{P;m}(Q)$ for a given $Q \in E(L)[m]$ more efficiently, by simply computing

$$f_{P;i}(Q) = f_{P;j}(Q) f_{P;i-j}(Q) \frac{\ell_{[j]P,[i-j]P}(Q)}{\ell_{[i]P,\mathcal{O}}(Q)}$$

iteratively for some $1 \leq j < i$. Since j may be chosen arbitrarily, this allows us to use a modified square-and-multiply algorithm, which yields a time complexity of $\Theta(\log m)$ operations in $E(L)$. This is known as *Miller's algorithm*, due to the unpublished manuscript (Miller, 1986a). Sage code implementing Miller's algorithm is given in Algorithm 3.1.

Algorithm 3.1 Miller

```
def line(E, P, Q, R):
    """
    E: an elliptic curve
    (P, Q, R): points on E
    """
    if P == 0:
        if Q == 0:
            return 1
        else:
            return R[0] - Q[0]
    elif Q == 0 or P == -Q:
        return R[0] - P[0]
    elif P == Q:
        m = (3 * P[0]^2 + E.a4()) / (2 * P[1])
        c = (-P[0]^3 + E.a4() * P[0] + 2 * E.a6()) / (2 * P[1])
    else:
        m = (P[1] - Q[1]) / (P[0] - Q[0])
        c = (Q[0] * P[1] - P[0] * Q[1]) / (Q[0] - P[0])
    return R[1] - (m * R[0] + c)

def miller(E, m, P, Q):
    """
    E: an elliptic curve
    m: a positive integer
    (P, Q): m-torsion points on E
    """
    fPi, iP = 1, 0
    fPj, jP = 1, P
    while m > 0:
        if m % 2 == 1:
            fPi *= fPj * line(E, iP, jP, Q) / line(E, iP + jP, 0, Q)
            iP += jP
        fPj *= fPj * line(E, kP, kP, Q) / line(E, 2 * jP, 0, Q)
        jP += jP
        m //= 2
    return fPi
```

Remark 3.1. Miller's algorithm may fail to compute $f_{P;m}(Q)$ at a point $Q \in E(L)[m]$ if a denominator $\ell_{[k]P,\mathcal{O}}(Q) = 0$ is encountered during the computation for some intermediate $1 \leq k < m$. In this case, we gain the knowledge that $Q \in \langle P \rangle$, which will in fact help us to handle the situation in our applications.

3.2 The Weil pairing

Definition 3.2. For $m \in \mathbb{Z}$ such that $\mathrm{char}(K) \nmid m$, the *m-th Weil pairing* on $E[m]$ is the mapping

$$e_m : E[m] \times E[m] \to \bar{K}^\times[m]$$

defined by $e_m(P, P) := e_m(P, \mathcal{O}) := e_m(\mathcal{O}, P) := 1$ for all $P \in E[m]$ and

$$e_m(P, Q) := (-1)^m \frac{f_{P;m}(Q)}{f_{Q;m}(P)}$$

for all $P, Q \in E[m] \setminus \{\mathcal{O}\}$ such that $P \neq Q$ (Weil, 1940).

Note that in the above definition $\bar{K}^\times$ is considered as a group variety, so that $\bar{K}^\times[m]$ denotes the group of m-th roots of unity in $\bar{K}$. It remains to be established that $e_m(P, Q) \in \bar{K}^\times[m]$ as claimed. This is a trivial consequence of the following theorem, which justifies the chain of equalities $e_m(P, Q)^m = e_m([m]P, Q) = e_m(\mathcal{O}, Q) = 1$.

Theorem 3.2. *Let $P, P', Q \in E[m]$, $\phi : E \to E'$ an isogeny, and $Q' \in E'[m]$. Then, the Weil pairings e_m and e'_m on $E[m]$ and $E'[m]$ respectively satisfy the following properties:*

(a) (Alternating) $e_m(P, Q) = e_m(Q, P)^{-1}$;

(b) (Bilinear) $e_m(P+P', Q) = e_m(P, Q)e_m(P', Q)$ and $e_m(Q, P+P') = e_m(Q, P)e_m(Q, P')$;

(c) (Non-degenerate) if $e_m(P, R) = 1$ for all $R \in E[m]$ or $e_m(R, P) = 1$ for all $R \in E[m]$ then $P = \mathcal{O}$; and

(d) (Compatable) $e_m(P, \hat{\phi}Q') = e'_m(\phi P, Q')$.

Proof. The alternating property is immediate from the definition, and implies that bilinearity and non-degeneracy only need to be shown in the first argument. For a proof of the other properties, it is useful to consider an alternative construction of the Weil pairing, as given in Section 11.2 of (Washington, 2008). See Theorem 11.7 of the same text for a proof of the remaining properties using this construction, and Theorem 11.12 for a proof that these definitions of the Weil pairing are equivalent. □

Since the point P and the function $f_{P;m}$ are defined over the same extension L/K by the construction in Definition 3.1, and thus $f_{P;m}(Q) \in L$ if $Q \in E(L)$, it is easy to see that if $E[m] \leq E(L)$, then the m-th Weil pairing on E has the form

$$e_m : E(L)[m] \times E(L)[m] \to L^{\times}[m].$$

As we pointed out in Remark 3.1, Miller's algorithm may fail to compute $f_{P;m}(Q)$, but in this case $[k]P = Q$ for some $k = 1, \ldots, m-1$, so that $e_m(P, Q) = e_m(P, [k]P) = e_m(P, P)^k = 1$. With this observation, a Sage implementation of an algorithm for computing the m-th Weil pairing on E is given in Algorithm 3.2. This algorithm has a time complexity of $\Theta(\log m)$ operations in some extension L over K such that $E[m] \leq E(L)$, since it merely involves two executions of Miller's algorithm.

Algorithm 3.2 Weil pairing

```
def weil_pairing(E, m, P, Q):
    """
    E: an elliptic curve
    m: a positive integer
    (P, Q): m-torsion points on E
    """
    if P == Q or P == 0 or Q == 0:
        return 1
    try:
        return (-1)^m * miller(E, m, P, Q) / miller(E, m, Q, P)
    except ZeroDivisionError:
        return 1
```

3.3 The Tate pairing

Definition 3.3. The *m-torsion embedding field* of $K^{\times}$ is the extension $K_m := K(K^{\times}[m])$ of K obtained by adjoining the elements of $\bar{K}^{\times}[m]$. The *m-torsion* *embedding degree* of $K^\times$ is the degree $d_{K^\times;m} := [K_m : K]$ of this extension.

Definition 3.4. For $m \in \mathbb{Z}$ such that $\operatorname{char}(K) \nmid m$, the *m-th Tate pairing* on E is the mapping

$$t_m : E(K_m)[m] \times E(K_m)/[m]E(K_m) \to K_m^\times/(K_m^\times)^m$$

defined by

$$\begin{aligned} t_m(P, [\mathcal{O}]) &:= 1; \\ t_m(P, [P]) &:= t_m(P, [P] + [Q])/t_m(P, [Q]); \text{ and} \\ t_m(P, [Q]) &:= f_{P;m}(Q) \end{aligned}$$

for all $P \in E(\mathbb{F}_{q^d})[m]$ and $Q \in E(\mathbb{F}_{q^d}) \setminus \{\mathcal{O}, P\}$ (Tate, 1958; Lichtenbaum, 1969).

Note that in the above definition, $t_m(P, [P])$ is multiply defined for $P \in E(K_m)[m]$. However, the following theorem guarantees that these definitions coincide.

Theorem 3.3. *Let $P \in E(K_m)[m]$ and $[Q] \in E(K_m)/[m]E(K_m)$. Then the Tate pairing t_m satisfies the following properties:*

(a) (Bilinear) if $P' \in E(K_m)[m]$ then $t_m(P + P', [Q]) = t_m(P, [Q])t_m(P', [Q])$ and if $[Q'] \in E(\mathbb{F}_{K_m})/[m]E(\mathbb{F}_{K_m})$ then $t_m(P, [Q]+[Q']) = t_m(P, [Q])t_m(P, [Q'])$; and

(b) (Non-degenerate) if $t_m(P, [Q']) = 1$ for all $[Q'] \in E(K_m)/[m]E(K_m)$ then $P = \mathcal{O}$, and if $t_m(P', [Q]) = 1$ for all $P' \in E(K_m)[m]$ then $[Q] = [\mathcal{O}]$.

Proof. See Theorem 3.17 of (Washington, 2008) for a proof of bilinearity and Section 11.7 of the same text for a proof of non-degeneracy. □

As with the Weil pairing, Miller's algorithm may fail to compute $f_{P;m}(Q)$ if $[k]P = Q$ for some $k = 1, \ldots, m - 1$. Unlike the Weil pairing, this does not tell us the value of $f_{P;m}(Q)$, so we deal with the problem by selecting a random $R \in E(K_m)$ and computing $f_{P;m}(Q + R)/f_{P;m}(R)$ instead. On closer inspection of Miller's algorithm, one sees that there are only $\Theta(\log m)$ points $Q \in E(K_m)$ for which the computation of $f_{P;m}(Q)$ will fail for a fixed P, so the expected number of evaluations of Miller's algorithm is $\Theta(1)$, for an overall expected time complexity of $\Theta(\log m)$ operations in K_m. Sage code for computing the m-th Tate pairing on E is given in Algorithm 3.3.

Algorithm 3.3 Tate pairing

```
def tate_pairing(E, m, P, Q):
    """
    E: an elliptic curve
    m: a positive integer
    (P, Q): m-torsion points on E
    """
    if Q == 0:
        return 1
    if P == Q:
        R = E.random_point()
        return tate_pairing(E, m, P, P + R) / tate_pairing(E, m, P, R)
    try:
        return miller(E, m, P, Q)
    except ZeroDivisionError:
        R = E.random_point()
        return tate_pairing(E, m, P, Q + R) / tate_pairing(E, m, P, R)
```

Chapter 4

Point counting

4.1 Naive methods

The most naive approach for counting the number of points $\#E(\mathbb{F}_q)$ on an elliptic curve involves simply enumerating all pairs $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, counting those which satisfy the Weierstrass equation $y^2 = f(x)$ of E, and adding one to the tally for the point at infinity. This approach has a time complexity of $O(q^2)$ field operations.

We may do slightly better by only enumerating values $x \in \mathbb{F}_q$, computing $f(x)$, then counting one point if it is zero or two points if it is a quadratic residue modulo $\mathbb{F}_q$. Testing whether or not $f(x)$ is a quadratic residue involves computing the Legendre symbol $\left(\frac{f(x)}{\mathbb{F}_q}\right)$, which may be done with $O(\log q)$ field operations. Thus, the total time complexity of this approach is $O(q \log q)$ field operations.

4.2 Schoof's algorithm

The first algorithm for computing $\#E(\mathbb{F}_q)$ with time complexity polynomial in $\log q$ was published in (Schoof, 1985). Its discovery was the theoretical breakthrough which allowed (Miller, 1986*b*) and (Koblitz, 1987) to suggest the use of elliptic curves in cryptography soon thereafter. In this section, we describe this algorithm, known as *Schoof's algorithm*, closely following the exposition of (Sutherland, 2015).

The algorithm works by computing the trace of Frobenius $\operatorname{tr} \phi_q$ modulo ℓ for many distinct small primes ℓ, then using the Chinese remainder theorem to compute $\operatorname{tr}\phi_q$ modulo the product of these primes. Since by Hasse's theorem $\operatorname{tr}\phi_q$ can only take values in an interval of width $4\sqrt{q}$, we may fully specify its value by taking the product of primes to be larger than this interval.

The technique we will present for computing $\operatorname{tr}\phi_q \bmod \ell$ works for any prime $\ell \neq p$. Therefore we may simply use the k smallest primes $\ell_1, \dots, \ell_k$ such that the product

$$N_k := \prod_{i=1}^{k} \ell_i$$

satisfies $N_k > 4\sqrt{q}$. It is possible that $\ell_i = p$ for some $i \leq k$, but we ignore this possibility since it does not occur in large characteristic and we could simply substitute ℓ_i with ℓ_{k+1}. Sage code for the main loop of Schoof's algorithm is given in Algorithm 4.1

Algorithm 4.1 Schoof

```
def schoof(E):
    """
    E: an elliptic curve
    """
    q = E.base_field().order()
    residues = [trace_of_frobenius_mod_2(E)]
    moduli = [2]
    l = 3
    while prod(moduli) <= 4 * sqrt(q):
        if q % l == 0:
            l = next_prime(l)
        residues.append(trace_of_frobenius_mod(E, l))
        moduli.append(l)
        l = next_prime(l)
    t = crt(residues, moduli)
    if t > 2 * sqrt(q):
        return t - prod(moduli)
    else:
        return t
```

4.2.1 Computing the trace of Frobenius modulo 2

In order to compute $\operatorname{tr}\phi_q \bmod \ell$ for some prime ℓ, we proceed as follows. The case $\ell = 2$ is dealt with separately by checking whether $(f(x), x^q - x)$ is a constant. We know that the roots of $f(x)$ are the x-coordinates of the points of order 2 of E, while the roots of $x^q - x$ contain the x-coordinates of all of its $\mathbb{F}_q$-rational points. Therefore, $(f(x), x^q - x)$ is a constant if and only if $E(\mathbb{F}_q)$ has no points of order 2 so that $\#E(\mathbb{F}_q) = q + 1 - \operatorname{tr}\phi_q$ is odd, or equivalently $\operatorname{tr}\phi_q$ is odd since p is assumed to be an odd prime, so that $\operatorname{tr}\phi_q \bmod 2$ is determined. Note that reducting $x^q - x$ modulo $f(x)$ will not affect which divisors it has in common with $f(x)$. Therefore, we should rather compute $(f(x), x^q - x \bmod f(x))$, using the square-and-multiply algorithm to first calculate the modular exponent $x^q \bmod f(x)$ (Washington, 2008). Sage code for computing the trace of Frobenius modulo 2 is given in Algorithm 4.2

Algorithm 4.2 Trace of Frobenius modulo 2

```
def trace_of_frobenius_mod_2(E):
    """
    E: an elliptic curve
    """
    F = E.base_field()
    _.<x> = F[]
    f = x^3 + E.a4() * x + E.a6()
    q = F.order()
    if gcd(f, power_mod(x, q, f) - x).is_constant():
        return 1
    else:
        return 0
```

4.2.2 Characteristic equation of Frobenius modulo ℓ

Suppose now that ℓ is an odd prime. Let t_ℓ denote the unique integer of minimum absolute value satisfying $\operatorname{tr}\phi_q \equiv t_\ell \mod \ell$ so that the multiplication endomorphisms $[\operatorname{tr}\phi_q]$ and $[t_\ell]$ are equal when restricted to $E[\ell]$. Furthermore, note that since ϕ_q is a monomorphism, its restriction $\phi_{q;\ell}$ to $E[\ell]$ is a monomorphism. By Theorems 2.11 and 2.12 we have that the characteristic equation of the Frobenius endomorphism is

$$\phi_q^2 + [q] = [\operatorname{tr}\phi_q]\phi_q,$$

from which we derive the equation

$$\phi_{q;\ell}^2 + [q \bmod \ell] = [t_\ell]\phi_{q;\ell} \tag{4.1}$$

which is satisfied for all ℓ-torsion points of E.

Choosing an affine point $(x, y) \in E[\ell]$, we may now test each of the values $t_\ell = 0, \pm 1, \ldots, \pm\frac{\ell-1}{2}$, terminating when we find a value of t_ℓ for which the equation is satisfied at (x, y). Such a value will necessarily be unique, since $\phi_{q;\ell}(x, y)$ has order ℓ. However, there may be no appropriate point (x, y) lying in $E(\mathbb{F}_q)$, thus requiring that we work over a potentially large algebraic extension of $\mathbb{F}_q$.

4.2.3 Computing in the endomorphism ring modulo ℓ

The issue of choosing an appropriate ℓ-torsion point may be addressed by avoiding the choice altogether, instead operating directly with the endomorphisms in (4.1), performing all necessary computations in the endomorphism ring $\mathrm{End}(E[\ell])$.

To this end, we will describe how to canonically represent elements of this endomorphism ring, thus allowing us to easily verify whether (4.1) holds for a particular test value of t_ℓ. We also describe how to perform operations on elements of the endomorphism ring which are represented in this way.

Let

$$\alpha := (\alpha_x, \alpha_y y) \in \mathrm{End}(E)$$

be an endomorphism of E in canonical form, where $\alpha_x, \alpha_y \in \mathbb{F}_q(x)$. A canonical representation for the restriction α_ℓ of α to $E[\ell]$ is obtained by using the ℓ-th division polynomial $\psi_\ell \in \mathbb{F}_q[x]$. Recall that this polynomial has degree $(\ell^2 - 1)/2$, and its roots are the x-coordinates of the points of order ℓ of E. Thus, assuming the denominators of α_x and α_y are relatively prime to ψ_ℓ, we may reduce these rational functions modulo ψ_ℓ to obtain the representation

$$\alpha_\ell = (\alpha_x \bmod \psi_\ell, (\alpha_y \bmod \psi_\ell)y).$$

If the denominator of either α_x or α_y has some non-constant greatest divisor ψ'_ℓ in common with ψ_ℓ, then we fail to represent α_ℓ in this form. However, this means that there are points of degree ℓ in the kernel of α, and hence the kernel of α_ℓ is a non-trivial subgroup $E[\ell]'$ of $E[\ell]$. Since $E[\ell]$ has order ℓ^2, this subgroup is either the whole of $E[\ell]$, in which case $\psi'_\ell = \psi_\ell$, or it has order ℓ and ψ'_ℓ has degree $(\ell - 1)/2$, with roots corresponding to the x-coordinates of affine points in $E[\ell]'$.

In the former case, α_ℓ is the zero endomorphism on $E[\ell]$, for which we may choose some unique representation. In the latter case, we may perform all subsequent computations in $\operatorname{End}(E[\ell]')$, since these endomorphisms still satisfy the characteristic equation (4.1). Canonical representations for the restrictions of endomorphisms to $E[\ell]'$ are obtained by reducing modulo ψ'_ℓ instead.

Note that all endomorphisms will be representable in this form, since the only endomorphisms for which non-invertible denominators will occur are those restricting to the zero endomorphism on $E[\ell]'$. Henceforth we will only refer to ψ_ℓ and $E[\ell]$, with the understanding that they are to be replaced by some ψ'_ℓ and $E[\ell]'$ if necessary.

Given two non-zero endomorphisms α_ℓ and β_ℓ of $E[\ell]$ in canonical form, we may compute their sum and product as usual, respectively by applying the group law of the elliptic curve and by composition of endomorphisms.

Composition yields the endomorphism with canonical form

$$\alpha_\ell\beta_\ell = (\alpha_x \circ \beta_x \bmod \psi_\ell, ((\alpha_y \circ \beta_x)\beta_y \bmod \psi_\ell)y).$$

Note that since α_ℓ and β_ℓ are in canonical form, their kernels must be trivial, and so the kernel of $\alpha_\ell\beta_\ell$ is trivial, thus ensuring that no non-invertible denominators occur.

Computing $\gamma_\ell := \alpha_\ell + \beta_\ell$ by applying the group law of E yields

$$\begin{aligned}\gamma_x &= m^2 - \alpha_x - \beta_x \bmod \psi_\ell\\ \gamma_y &= \frac{m}{y}(\alpha_x - \gamma_x) - \alpha_y \bmod \psi_\ell,\end{aligned}$$

where

$$m = \begin{cases} \frac{\alpha_y - \beta_y}{\alpha_x - \beta_x} y & \text{if } \alpha_x \neq \beta_x \\ \frac{3\alpha_x^2 + A}{2\alpha_y y} & \text{if } \alpha_\ell = \beta_\ell \end{cases}.$$

Note that γ_x and γ_y are functions in x only, since both contain only square factors of y. Furthermore, note that in the case $\alpha_\ell = \beta_\ell$, no non-invertible denominators will occur, since α_ℓ has a trivial kernel and ℓ is odd, so that $[2]\alpha_\ell$ has a trivial kernel. Therefore, the only case where we need to check for non-invertible denominators is the case $\alpha_x \neq \beta_x$.

Algorithm 4.3 gives Sage code for computing the trace of Frobenius modulo ℓ. Since Sage does not have a full implementation of elliptic curve endomorphism rings, with endomorphism rings of torsion subgroups entirely lacking, code has been written to fill this gap. The interested reader may refer to Appendix A for the relevant code – its complexity has been abstracted away intentionally in the present section, since it would distract from the essence of the algorithm. We only point out the detail that the last two lines of Algorithm 4.3 are responsible for replacing ψ_ℓ with a divisor if a non-invertible denominator is encountered.

Algorithm 4.3 Trace of Frobenius modulo ℓ

```
def trace_of_frobenius_mod(E, l):
    """
    E: an elliptic curve
    l: a prime number
    """
    psi_l = E.division_polynomial(l)
    q_l = E.base_field().order() % l
    t_l = 0
    while true:
        try:
            phi = FrobeniusEndomorphismMod(E, psi_l)
            lhs = phi^2 + q_l
            rhs = t_l * phi
            while t_l <= (l - 1) // 2:
                if lhs == rhs:
                    return t_l
                elif lhs == -rhs:
                    return -t_l
                t_l += 1
                rhs += phi
        except ZeroDivisionError as e:
            psi_l = e[0]
```

4.2.4 Time complexity analysis

Now that we have made the rules for computing in the endomorphism ring of $E[\ell]$ explicit, we can analyse the time complexity of Schoof's algorithm. In the worst case, we perform all computations in the full endomorphism ring of $E[\ell]$, never finding a factor of ψ_ℓ. The computation of $\phi_{q;\ell}$, $\phi_{q;\ell}^2$ and $[q \bmod \ell]$ are done upfront, and can be made efficient by computing the components of $\phi_{q;\ell}$

and $\phi_{q;\ell}^2$ using a square-and-multiply algorithm, and by computing $[q \bmod \ell] = (q \bmod \ell)[1]$ using a double-and-add algorithm.

For each prime ℓ, we need to test at most ℓ values for t_ℓ, requiring a total of $O(\ell)$ additions in the endomorphism ring. For each addition, we perform a constant number of operations using polynomials of degree $O(\ell^2)$ in the polynomial ring $\mathbb{F}_q[x]$, the most expensive of which is computing the multiplicative inverse modulo ψ_ℓ. Under the assumption that $\log \ell \in O(\log q)$, which it clearly is, this requires $O(M(\ell^2 \log q) \log \ell)$ operations using the extended Euclidean algorithm, where $\Theta(M(n))$ is the time complexity of the algorithm used for multiplying two n-bit integers (Sutherland, 2015).

The last ingredient in the time complexity analysis is to obtain asymptotic estimates for the number of primes k and the largest prime ℓ_k used in the algorithm. Recall the *prime number theorem*, which gives the asymptotic formula $\pi(x) \sim \frac{x}{\log x}$ for the prime-counting function π. An equivalent form of the prime number theorem is that the Chevyshev function

$$\vartheta(x) := \sum_{p \leq x} \log p,$$

has the asymptotic formula $\vartheta(x) \sim x$ (Apostol, 1976). Since we have chosen the product of primes N_k so that $\log N_k = \vartheta(\ell_k)$, we have $\ell_k \sim \log N_k \sim \frac{1}{2} \log q$, and furthermore we then have $k = \pi(\ell_k) \sim \frac{\log q}{2 \log \log q}$.

Substituting these asymptotic formulae for ℓ_k and k yields a total time complexity of

$$O(M((\log q)^3)(\log q)^2)$$

for Schoof's algorithm. Using schoolbook multiplication gives a time complexity of $O((\log q)^8)$, but this can be brought down to $\tilde{O}((\log q)^5)$ if the Schönhage-Strassen algorithm, with time complexity $M(n) \in O(n \log n \log \log n)$, is used instead. However, the constants in the latter multiplication algorithm are very large, so that other multiplication algorithms are typically more efficient for the values of q currently used in cryptographic applications (Sutherland, 2015).

Chapter 5

Discrete logarithms

In this chapter, we introduce the notion of *discrete logarithms* on a cyclic subgroup of a group variety. We then discuss various approaches for computing discrete logarithms, in general as well as for certain special classes of elliptic curves. As we will see in Chapter 6, elliptic curve cryptography is based on the assumption that it is difficult to compute these discrete logarithms for elliptic curves over finite fields. The books (Galbraith, 2012; Washington, 2008) are extensively used as references throughout this chapter.

5.1 The discrete logarithm and Diffie-Hellman problems

Let $V/\mathbb{F}_q$ be a group variety with a fixed *base point* $G \in V(\mathbb{F}_q)$ of order n such that the addition and negation of points in $\langle G \rangle$ may be computed with time complexity polynomial in $\log n$. Some familiar examples of groups varieties which always have a base point with this property are the additive group $\mathbb{F}_q^+$, the multiplicative group $\mathbb{F}_q^\times$, and an elliptic curve group $E/\mathbb{F}_q$.

Note that the map $\exp_G : k \mapsto [k]G$ is an isomorphism from $\mathbb{Z}/n\mathbb{Z}$ to $\langle G \rangle$ and may be evaluated at any element of $\mathbb{Z}/n\mathbb{Z}$ with time complexity polynomial in $\log n$ using a double-and-add algorithm. The problem of evaluating the inverse function $\log_G := \exp_G^{-1}$ at some point in $\langle G \rangle$ is known as the *discrete logarithm problem* – in the literature, this name often refers to the special case for the group $\mathbb{F}_q^\times$, while the case of an elliptic curve $E/\mathbb{F}_q$ is referred to as the *elliptic curve discrete logarithm problem.*

It is clear that a brute force solution to the discrete logarithm problem exists, where one computes $[0]G, [1]G, \ldots$ until the desired point is found. This will require $\frac{n}{2}$ point additions on average, which leads to an average-case time complexity exponential in $\log n$. However, for the group variety $\mathbb{F}_q^+$, we may evaluate $\log_G P$ with time complexity polynomial in $\log n$ by performing the computation $G^{-1}P$ in $\mathbb{F}_q$. So-called *index calculus algorithms* are a well-known family of algorithms which solve the discrete logarithm problem in $\mathbb{F}_q^\times$ with a time complexity of $L_n[1/3, c]$, which is subexponential in $\log n$ (Galbraith, 2012).

The question arises for which groups varieties we may solve the discrete logarithm problem efficiently, say with a time complexity polynomial in $\log n$, and for which group varieties the discrete logarithm problem is hard, say only solvable with an expected time complexity exponential in $\log n$ (Cohen and Frey, 2015).

In the present chapter, we first describe some algorithms for solving the discrete logarithm problem for general group varieties. These algorithms all have time complexity exponential in $\log n$ – $\Theta(\sqrt{n})$ to be precise. We then describe algorithms for solving the elliptic curve discrete logarithm problem efficiently for special classes of elliptic curves, which are to be avoided in cryptographic applications. In the next chapter, we proceed to define some cryptosystems whose security rely on the difficulty of the discrete logarithm problem.

There is a common variant of the discrete logarithm problem known as the *Diffie-Hellman problem*, which is the problem of computing $\exp_G(\log_G P \log_G Q)$ for two points $P, Q \in \langle G \rangle$. Clearly an efficient solution to the discrete logarithm problem also gives an efficient solution to the Diffie-Hellman problem, but it is not known whether the converse is true for any group varieties. However, the general consensus is that the two problems are roughly equally difficult for the group varieties $\mathbb{F}_q^\times$ and $E(\mathbb{F}_q)$.

5.2 The baby-step giant-step algorithm

In the present section we describe a simple algorithm known as the *baby-step giant-step algorithm*, due to (Shanks, 1971), for solving the discrete logarithm problem for a general group variety with a more favourable time complexity than that of a brute force solution.

Let $m \in [0, n)$ be an integer. Using the Euclidean division of k by m we may uniquely rewrite a point $P := [k]G$ as $P = [mq + r]G$, where $0 \leq q \leq \lfloor n/m \rfloor$ and $0 \leq r < m$. We now precompute the restriction of $\log_G$ to the values $[0]G, [1]G, \ldots, [m-1]G$ and store it efficiently in a lookup table, i.e. we store the key-value pairs $([j]G, j)$ for $j = 0, \ldots, m-1$. Iterating over the values $i = 0, \ldots, \lfloor n/m \rfloor$, we compute $P - [mi]G = [m(q-i) + r]G$, and test for its membership in the lookup table. Clearly, if a pair $([j]G, j)$ is found, then $q = i$, and $r = j$, thus fully determining the discrete logarithm.

The time complexity of the algorithm crucially depends on the parameter m, since the precomputation step requires about m point additions, and the iteration step requires about n/m point additions on average, for a total of $m + n/m$ point additions. Thus, the number of point additions is minimised when $m \approx \sqrt{n}$, thus yielding an algorithm with a time complexity of $\Theta(\sqrt{m})$ point additions. Sage code implementing the baby-step giant-step algorithm is given in Algorithm 5.1.

Algorithm 5.1 Baby-step giant-step

```
def baby_step_giant_step(G, P, n):
    """
    G: a point
    P: a multiple of G
    n: the order of G
    """
    m = round(sqrt(n))
    log_G = {}
    jG = 0 * G
    for j in (0..m-1):
        log_G[jG] = j
        jG += G
    mG = m * G
    miG = 0 * G
    for i in (0..floor(n/m)):
        if P - miG in log_G:
            return m * i + log_G[P - miG]
        miG += mG
```

Note that the space complexity of the algorithm also depends on m – indeed, exactly m points are stored in the lookup table, or $\Theta(\sqrt{n})$ when optimising for the time complexity.

It is worth noting that the precomputation step only needs to be performed once for a given base point G, while the iteration step needs to be executed for each evaluation of $\log_G$ at a point in $\langle G \rangle$. For this reason, one may wish to set m to a larger value than $\sqrt{n}$ if one wishes to solve many discrete logarithms for the same base point G.

Using modern computer hardware, the space complexity of the baby-step giant-step algorithm may prove to be more of a hurdle than its time complexity when solving large instances of the discrete logarithm problem. In Section 5.4, we present two probabilistic algorithms with the same time complexity as the baby-step giant-step algorithm, but with significantly better space complexities.

5.3 The Pohlig-Hellman reduction

In the present section we describe a procedure known as the *Pohlig-Hellman reduction*, first published in (Pohlig and Hellman, 1978), which allows one to reduce the problem of solving a base-G discrete logarithm to solving discrete logarithms in the prime-order subgroups of $\langle G \rangle$.

If $q_1 \ldots q_k$ is the prime factorization of n, where the $q_i := \ell_i^{e_i}$ are powers of distinct primes, we proceed to solve the discrete logarithm $\log_G P$ for some $P \in \langle G \rangle$ as follows.

We proceed by computing the discrete logarithm $\log_{[n/q_i]G}[n/q_i]P$ in the subgroup $\langle [n/q_i]G \rangle$ of order q_i of $\langle G \rangle$ for each $i = 1, \ldots, k$. Since this yields $\log_G P$ modulo q_i, the original discrete logarithm may then be reconstructed using the Chinese remainder theorem.

However, for some fixed factor $q = \ell^e$ in the prime factorization of n, the computation of the discrete logarithm $d := \log_{[n/q]G}[n/q]P$ may be further reduced to computing a sequence of e discrete logarithms in the p-order subgroup $\langle [n/\ell]G \rangle$ of $\langle G \rangle$.

First we represent the discrete logarithm d in base ℓ as

$$d = d_0\ell^0 + \cdots + d_{e-1}\ell^{e-1}.$$

Note that using this representation, we have

$$d \equiv \sum_{j=0}^{k} d_j \ell^j \pmod{\ell^{k+1}}$$

for $k = 0, \ldots, e-1$, so that

$$\log_{[n/\ell^{k+1}]G}[n/\ell^{k+1}]P = \sum_{j=0}^{k} d_j \ell^j.$$

Thus to find d_0, we simply compute $\log_{[n/\ell]G}[n/\ell]P$. Assuming that $d_0, \ldots, d_{k-1}$ are already known, we may find d_k by computing the discrete logarithm

$$\log_{[n/\ell]G}\left([n/\ell^{k+1}]P - \left[\sum_{j=0}^{k-1} d_j \ell^j\right][n/\ell^{k+1}]G\right),$$

which yields d_k since

$$\begin{aligned}
[n/\ell^{k+1}]P - \left[\sum_{j=0}^{k-1} d_j \ell^j\right][n/\ell^{k+1}]G &= \left[\sum_{j=0}^{k} d_j \ell^j - \sum_{j=0}^{k-1} d_j \ell^j\right][n/\ell^{k+1}]G \\
&= [d_k \ell^k][n/\ell^{k+1}]G = [d_k][n/\ell]G.
\end{aligned}$$

Sage code for the Pohlig-Hellman reduction is given in Algorithm 5.2. Note that to compute discrete logarithms in the prime-order subgroups of $\langle G \rangle$, this code uses the baby-step giant-step algorithm described in the previous section. Any other algorithm for solving discrete logarithms may be used in its place, including those which we will describe later in this chapter.

If ℓ is the largest prime factor of n, then the time complexity of the Pohlig-Hellman reduction is $O(\log n\sqrt{\ell})$ point additions, assuming that the baby-step giant-step algorithm is used for computing discrete logarithms in the prime-order subgroups of $\langle G \rangle$. To see this, note that

$$\sum_{i=1}^{k} e_i \leq \log_2 n,$$

and the largest discrete logarithm problem solved will be in a subgroup of order ℓ.

Of course, this analysis assumes that the prime factorisation of n is already known. In practice, this is a reasonable assumption, since the groups which are used for cryptography are typically proposed as standards, which do not change frequently. Furthermore, the *general number field sieve* algorithm can factorise arbitrary integers with time complexity $L_n[1/3, c]$, which is subexponential in $\log n$ and thus significantly more favourable than general techniques for discrete logarithms such as the baby-step giant-step algorithm (Galbraith, 2012).

Algorithm 5.2 Pohlig-Hellman reduction

```
def pohlig_hellman_reduction(G, P, n, n_factors):
    """
    G: a point
    P: a multiple of G
    n: the order of G
    n_factors: the prime factorisation of n
    """
    residues = []
    moduli = [l^e for l, e in n_factors]
    for l, e in n_factors:
        d = 0
        G_prime = (n // l) * G
        for j in (0..e-1):
            P_prime = (n // l^(j + 1)) * (P - d * G)
            d += l^j * baby_step_giant_step(G_prime, P_prime, l)
        residues.append(d)
    return crt(residues, moduli)
```

Remark 5.1. The existence of the Pohlig-Hellman reduction severely restricts which group varieties can be used for cryptographic purposes, since it means that solving the discrete logarithm problem in $\langle G\rangle$ is roughly as difficult as solving the discrete logarithm problem in its largest subgroup of prime order. For this reason, we will henceforth assume that the order of G is a prime number ℓ.

5.4 Pollard's algorithms

5.4.1 Pollard's ρ algorithm

Pollard's ρ algorithm is a probabilistic algorithm of Las Vegas type due to (Pollard, 1978) for solving general discrete logarithm problems, with a favourable expected time complexity, and constant space complexity. The algorithm relies on finding collisions of the form $[a]G+[b]P=[a']G+[b']P$, so that $[a-a']G = [b'-b]P$. If $b \not\equiv b' \pmod{\ell}$, we may then determine $\log_G P = (a-a')(b'-b)^{-1}$, where arithmetic is performed in $\mathbb{F}_\ell$.

In order to find collisions, we would like to repeatedly sample elements of the form $[a]G+[b]P$ randomly from $\langle G\rangle$. A standard statistical argument in the style of the *birthday paradox* shows that the expected number of samples needed before finding a collision is asymptotically equal to $\sqrt{\frac{\pi}{2}\ell}$ (Galbraith, 2012). However, if we store all of the sample elements while searching for a collision, this approach has no better space complexity than the baby-step giant-step algorithm.

To avoid storing the sample elements, we wish to find a deterministic function $f_{G,P} : \langle G \rangle \to \langle G \rangle$ which behaves sufficiently randomly. That is, such that if $Q_0 \in \langle G \rangle$ is randomly selected, then the sequence defined by $Q_{i+1} := f_{G,P}(Q_i)$ will enter a cycle after an expected number of approximately $\sqrt{\frac{\pi}{2}\ell}$ steps. Furthermore, we would like $f_{G,P}$ to have the property that if a_i, b_i are known such that $Q_i = [a_i]G + [b_i]P$, then it is easy to determine a_{i+1}, b_{i+1} such that $Q_{i+1} = [a_{i+1}]G + [b_{i+1}]P$.

Using *Floyd's cycle-finding algorithm*, a collision can then be detected after an expected number of approximately $3\sqrt{\frac{\pi}{2}\ell}$ function evaluations as follows (Floyd, 1967). We initially select $a_0, b_0 \in \mathbb{Z}/n\mathbb{Z}$ at random and compute $Q_0 = [a_0]G + [b_0]P$. For $i = 1, 2, \ldots$, we then compute Q_i and Q_{2i} from Q_{i-1} and $Q_{2(i-1)}$ by applying $f_{G,P}$ and $f_{G,P} \circ f_{G,P}$ respectively.

It is possible that, after finding a collision $Q_i = Q_{2i}$, we in fact have $b_i \equiv b_{2i} \pmod{\ell}$. However, this is an unlikely event, with a probability of $1/\ell$ if the b_i's are assumed to be uniformly distributed. In this case, we may simply restart the computation with a different choice of a_0 and b_0. Pollard's ρ algorithm thus has an expected time complexity of $\Theta(\sqrt{\ell})$ point additions. Furthermore, it is clear that it has a space complexity of $\Theta(1)$ points.

In practice, the following choice for the function $f_{G,P}$ is popular, due to its good heuristic properties. Let m be a fixed positive integer, and randomly select two vectors A and B over $\mathbb{F}_\ell$ of length m. Let $H : \langle G \rangle \to \{0, \ldots, m-1\}$ be a hash function. We then set $f_{G,P}(Q_i) := Q_i + [A_{H(Q_i)}]G + [B_{H(Q_i)}]P$, so that $a_{i+1} = a_i + A_{H(Q_i)}$ and $b_{i+1} = b_i + B_{H(Q_i)}$ may be easily determined. Sage code implementing Pollard's ρ algorithm with this choice of $f_{G,P}$ is given in Algorithm 5.3.

5.4.2 Pollard's λ algorithm

We now describe a variant of Pollard's ρ algorithm due to (van Oorschot and Wiener, 1996), which is referred to as *Pollard's λ algorithm*, *parallel Pollard's ρ algorithm*, and *Pollard's ρ algorithm with distinguished points* in the literature.

Algorithm 5.3 Pollard's ρ

```
def pollard_rho(G, P, l, m=20):
    """
    G: a point of prime order
    P: a multiple of G
    l: the order of G
    """
    A, B = random_matrix(GF(l), 2, m)
    def f(Q, a, b):
        h = hash(Q) % m
        return Q + ZZ(A[h]) * G + ZZ(B[h]) * P, a + A[h], b + B[h]
    a_i, b_i = a_2i, b_2i = random_vector(GF(l), 2)
    Q_i = Q_2i = ZZ(a_i) * G + ZZ(b_i) * P
    for i in (1..):
        Q_i, a_i, b_i = f(Q_i, a_i, b_i)
        Q_2i, a_2i, b_2i = f(*f(Q_2i, a_2i, b_2i))
        if Q_i == Q_2i:
            if b_i != b_2i:
                return ZZ((a_i - a_2i) / (b_2i - b_i))
            else:
                return pollard_rho(G, P, l)
```

A *distinguished point* is a point $Q \in \langle G \rangle$ which satisfies some chosen property D that can be checked efficiently. We denote $\theta := \#\left\{Q \in \langle G \rangle : D(Q)\right\}/\ell$, so that θ is the probability that an element selected uniformly at random from $\langle G \rangle$ satisfies D.

Reusing notation from the previous subsection, the algorithm performs pseudorandom walks in $\langle G \rangle$ by repeated application of $f_{G,P}$. However, rather than performing a single pseudorandom walk which terminates when it enters a cycle, the present algorithm performs multiple pseudorandom walks, each of which terminates when it reaches a distinguished point.

Once a distinguished point Q_i is found, it is stored in a lookup table, along with a_i and b_i. However, if the lookup table already contains an entry (Q'_j, a'_j, b'_j) such that $Q'_j = Q_i$, then we have found a collision with high probability. As in the previous algorithm, this will be the case if and only if $b_i \not\equiv b'_j \pmod{\ell}$, which then gives us the solution $\log_G P = (a_i - a'_j)(b'_j - b_i)^{-1}$ to the discrete logarithm problem.

The key to the efficiency of Pollard's λ algorithm is choosing a property D which yields a favourable value of θ. As in the analysis of Pollard's ρ algorithm, the expected number of iterations of $f_{G,P}$ needed to obtain a collision is about $\sqrt{\frac{\pi}{2}\ell}$. However, the collision will only be detected once the current pseudorandom walk reaches a distinguished point. Since the expected length of a pseudorandom walk before reaching a distinguished point is $1/\theta$, the total number of iterations of $f_{G,P}$ needed, and thus the number of point additions performed, will be roughly

$$\sqrt{\frac{\pi}{2}\ell} + \frac{1}{\theta}$$

The total number of distinguished points found among all of the walks is expected to be a fraction θ of all iterations of $f_{G,P}$, so that roughly $\theta\sqrt{\frac{\pi}{2}\ell}$ group elements will need to be stored.

Selecting some function $g \in o(1)$ and letting $\theta := \frac{1}{g(\ell)\sqrt{\ell}}$ thus yields an algorithm which computes an expected number of $\sqrt{\frac{\pi}{2}\ell}$ point additions, while storing $\sqrt{\frac{\pi}{2}}/g(\ell)$ points. For example, choosing $g(\ell) := \frac{1}{\log \ell}$ yields a modest space complexity of $\Theta(\log \ell)$ points, while maintaining a favourable time complexity of $\Theta(\sqrt{\ell})$.

A choice of property which may yield a particular value of θ in practice is to let $D(Q)$ be the property that $H(Q) = 0$, where $H : \langle G \rangle \to \mathbb{Z}/\lfloor 1/\theta \rfloor\mathbb{Z}$ is a hash function. Sage code implementing Pollard's λ algorithm with this choice of property is given in Algorithm 5.4.

A significant practical advantage of Pollard's λ algorithm over Pollard's ρ algorithm and the baby-step giant-step algorithm, is that it is trivial to convert into a parallel algorithm which runs across multiple processing units. All processing units may perform random walks simultaneously, only communicating with each other when distinguished points are found.

5.5 Pairing-based reductions

In this section we focus specifically on the elliptic curve discrete logarithm problem. We describe two pairing-based approaches for solving the discrete logarithm problem on a subgroup $\langle G \rangle$ of order ℓ of an elliptic curve group $E(\mathbb{F}_q)$ – one using the ℓ-th Weil pairing on E, and one using the ℓ-th Tate pairing.

Algorithm 5.4 Pollard's λ

```
def pollard_lambda(G, P, l, m=20):
    """
    G: a point of prime order
    P: a multiple of G
    l: the order of G
    """
    A, B = random_matrix(GF(l), 2, m)
    def f(Q, a, b):
        h = hash(Q) % m
        return Q + ZZ(A[h]) * G + ZZ(B[h]) * P, a + A[h], b + B[h]
    T = {}
    theta_inv = floor(sqrt(l) / log(l))
    while true:
        a_i, b_i = random_vector(GF(l), 2)
        Q_i = ZZ(a_i) * G + ZZ(b_i) * P
        for i in (1..):
            Q_i, a_i, b_i = f(Q_i, a_i, b_i)
            if hash(Q_i) % theta_inv == 0:
                if Q_i in T:
                    a_j, b_j = T[Q_i]
                    if b_i != b_j:
                        return ZZ((a_i - a_j) / (b_j - b_i))
                else:
                    T[Q_i] = a_i, b_i
                    break
```

5.5.1 The MOV reduction

The approach using the Weil pairing was first published in (Menezes *et al.*, 1993), and it has come to be known as the *MOV reduction* after its authors.

To use the ℓ-th Weil pairing, it will be necessary that $\ell \neq p$. However, in the case $\ell = p$, we will usually be able to use the reduction described in the next section, which solves the discrete logarithm problem with time complexity polynomial in $\log q$.

Lemma 5.1. *Let $P, Q \in E[\ell]$ be points of order ℓ such that $E[\ell] = \langle P, Q\rangle$. Then*

$$\langle P\rangle \to \bar{\mathbb{F}}_q^\times[\ell], \quad R \mapsto e_\ell(R, Q)$$

is an isomorphism of groups.

Proof. The map in question is clearly a morphism of groups by bilinearity of the Weil pairing. To see that it is an epimorphism, observe that if $e_\ell(R,Q) = e_\ell(S,Q)$ then $e_\ell(R-S,Q) = 1$. Furthermore, since $R - S \in \langle P \rangle$, it follows that $e_\ell(R-S,P) = e_\ell(P,P) = 1$ so that $R = S$ by non-degeneracy of the Weil pairing. The result follows since the domain $\langle P \rangle$ and codomain $\bar{\mathbb{F}}_q^\times[\ell]$ of the morphism both have order ℓ. □

Let $Q \in E[\ell]$ be a point of order ℓ such that $Q \notin \langle G \rangle$. Using the isomorphism from Lemma 5.1, we may transfer the discrete logarithm problem on $\langle G \rangle$ to a discrete logarithm problem on $\bar{\mathbb{F}}_q^\times[\ell]$. Concretely, for $P \in \langle G \rangle$, we compute $\log_G P = \log_{e_\ell(G,Q)} e_\ell(P,Q)$.

To analyse the time complexity of this reduction, we first need to determine the smallest integer d such that $E[\ell] \leq E(\mathbb{F}_{q^d})$. We state the following partial solution due to (Balasubramanian and Koblitz, 1998).

Theorem 5.1. *If $\ell \mid \#E(\mathbb{F}_q)$ and $\ell \nmid q-1$, then $E[\ell] \leq E(\mathbb{F}_{q^d})$ if and only if $\ell \mid (q^d - 1)$.*

This result assures us that if ℓ-torsion embedding degree $d := d_{\mathbb{F}_q^\times;\ell}$ of $\mathbb{F}_q$ is not equal to one, then $E[\ell] \leq E(\mathbb{F}_{q^d})$, where d is the smallest integer with this property. In the remainder of this section, we assume $\ell \nmid q-1$. The reduction presented in the next section will deal with the $\ell \mid q-1$ case.

In order to compute the reduction, we must first find a $\mathbb{F}_{q^d}$-point Q of order ℓ such that $e_\ell(G,Q) \neq 1$. This can be done by repeatedly selecting $Q \in E(\mathbb{F}_{q^d})[\ell]$ at random and computing $e_\ell(G,Q)$ to test the condition. Since $E(\mathbb{F}_{q^d})[\ell]$ is a $\mathbb{Z}/\ell\mathbb{Z}$-module of rank two, $e_\ell(G,Q) \neq 1$ for any $Q \notin \langle G \rangle$, so that the probability of selecting an appropriate Q is $1 - 1/\ell$ and we only need to sample $\Theta(1)$ points on average.

Assuming that $\#E(\mathbb{F}_{q^d})$ is known, we may sample from $E(\mathbb{F}_{q^d})[\ell]$ by first randomly selecting $R \in E(\mathbb{F}_{q^d})$, then multiplying R by an appropriate constant to obtain $Q \in E(\mathbb{F}_{q^d})[\ell]$. To this end, we first determine the largest $m \mid \#E(\mathbb{F}_{q^d})$ such that $\ell \nmid m$ by repeated division. We then compute $[m\ell^0]R, [m\ell^1]R, \ldots$ until an element $Q \in E(\mathbb{F}_{q^d})[\ell]$ is encountered. Thus, we can sample a point $Q \in E(\mathbb{F}_{q^d})[\ell]$ with a time complexity of $\Theta(d \log q)$ operations in $\mathbb{F}_{q^d}$, which then also gives the expected time complexity of finding a point such that $e_\ell(G,Q) \neq 1$.

Using an index calculus algorithm, one may solve the discrete logarithm problem in $\mathbb{F}_{q^d}^{\times}[\ell]$ with a time complexity of $L_{q^d}[1/3, c]$, clearly dominating the time complexity of the reduction. Thus, the MOV reduction will yield an algorithm with time complexity subexponential in $\log q$ if and only if $d \in o((\log q/\log\log q)^2)$. In particular, if $d \in O(1)$, this yields a $L_q[1/3, c]$ algorithm for the elliptic curve discrete logarithm problem.

A Sage implementation of the MOV reduction is given in Algorithm 5.5. Note that at the time of writing, Sage does not include an implementation of an index calculus algorithm for solving discrete logarithms in finite fields, so this code will not be efficient in large cases until such an implementation is included.

Algorithm 5.5 MOV reduction

```
def mov_reduction(G, P, l, E, n):
    """
    G: a point on E of prime order
    P: a multiple of G
    l: the order of G
    E: an elliptic curve
    n: the order of E
    """
    m = n
    while m % l == 0:
        m //= l
    g = 1
    while g == 1:
        Q = E.random_point() * m
        while Q * l != 0:
            Q *= l
        g = weil_pairing(E, l, G, Q)
    p = weil_pairing(E, l, P, Q)
    return p.log(g)
```

It is worth mentioning two important results from the literature regarding the MOV reduction.

It was shown in (Menezes *et al.*, 1993) that if $E/\mathbb{F}_q$ is a supersingular elliptic curve, then for every $\ell \mid \#E(\mathbb{F}_q)$, $E[\ell] \leq E(\mathbb{F}_{q^d})$ for some $d = 1, \ldots, 6$, so that the discrete logarithm problem on $E(\mathbb{F}_q)$ is at least as easy as the discrete logarithm problem on $\mathbb{F}_{q^6}$. This rendered an entire class of elliptic curves unsuitable for use in discrete logarithm cryptosystems.

However, (Balasubramanian and Koblitz, 1998) showed that it is exceptionally rare for a randomly selected pair of prime numbers p and ℓ such that $\ell \in [p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ and $d_{\mathbb{F}_p^\times;\ell} \neq 1$ to satisfy the condition $d_{\mathbb{F}_p^\times;\ell} \in o((\log \ell / \log\log \ell)^2)$. Thus, the MOV reduction is necessarily inefficient for an elliptic curve with $\#E(\mathbb{F}_p) = \ell$ using present state-of-the-art techniques for solving discrete logarithms in multiplicative groups of finite fields.

5.5.2 The Frey-Rück reduction

Using the Tate pairing, we may perform another reduction similar to the MOV reduction of the previous section. This reduction was first published in (Frey and Rück, 1994) and is known as the *Frey-Rück reduction* after its authors. As with the MOV reduction, we require that that $\ell \neq p$, but unlike the MOV reduction, we can find an isomorphism using the ℓ-th Tate pairing without the assumption $\ell \nmid q-1$. For the remainder of the section, let $d := d_{\mathbb{F}_q^\times,\ell}$.

Definition 5.1. For a prime number ℓ such that $\ell \neq p$, the *modified ℓ-th Tate pairing* on E is the mapping

$$\tau_m : E(\mathbb{F}_{q^d})[\ell] \times E(\mathbb{F}_{q^d})/[\ell]E(\mathbb{F}_{q^d}) \to \mathbb{F}_{q^d}^\times[\ell]$$

defined by $\tau_\ell(P,[Q]) := t_\ell(P,[Q])^{(q^d-1)/\ell}$ for all $P \in E(\mathbb{F}_{q^d})[\ell]$ and $Q \in E(\mathbb{F}_{q^d})$.

Note that the map

$$\mathbb{F}_{q^d}^\times/[\ell]\mathbb{F}_{q^d}^\times \to \mathbb{F}_{q^d}^\times[\ell], \quad [t] \mapsto t^{(q^d-1)/\ell}$$

is a well-defined isomorphism of groups, since its kernel is exactly the coset $[\ell]\mathbb{F}_{q^d}^\times$, while its domain and codomain are both groups of order ℓ. Therefore, the modified Tate pairing enjoys all of the same properties as the usual Tate pairing, with the practical advantage that evaluations of the modified Tate pairing can easily be checked for equality. However, the exponentiation step requires $\Theta(d \log q)$ operations in $\mathbb{F}_{q^d}$, thus dominating the time complexity of computing the modified Tate pairing, since the usual Tate pairing has a time complexity of $\Theta(\log \ell)$ and clearly $\ell \in O(q^d)$.

Lemma 5.2. *Let $P \in E(\mathbb{F}_{q^d})[\ell]$ be a point of order ℓ. Then there exists a point $Q \in E(\mathbb{F}_{q^d})$ such that*

$$\langle P \rangle \to \mathbb{F}_{q^d}^{\times}[\ell], \quad R \mapsto \tau_\ell(R, [Q])^{(q^d-1)/\ell}$$

is an isomorphism of groups.

Proof. By bilinearity of the modified Tate pairing, the map is clearly a morphism of groups. Since $P \neq \mathcal{O}$, there exists a point $Q \in E(\mathbb{F}_{q^d})$ such that $\tau_\ell(P, [Q]) \neq 1$ by non-degeneracy of the modified Tate pairing. Then $\mathbb{F}_{q^d}^{\times}[\ell]$ is generated by $\tau_\ell(P, [Q])$ since it is a group of prime order, from which it follows that the morphism is an epimorphism, and hence an isomorphism since its domain and codomain both have order ℓ. □

Let $Q \in E(\mathbb{F}_{q^d})$ be a point such that $\tau_\ell(G, Q) \neq 1$. Similarly to the MOV reduction, we may reduce the computation of the discrete logarithm $\log_G P$ for $P \in \langle G \rangle$ to the discrete logarithm $\log_{\tau_\ell(G,Q)} \tau_\ell(P, Q)$ in $\mathbb{F}_{q^d}^{\times}[\ell]$.

While Lemma 5.2 guarantees the existence of an isomorphism, it does not give an indication of how many points $Q \in E(\mathbb{F}_{q^d})$ will yield such an isomorphism. However, by observing that $E(\mathbb{F}_{q^d})/[\ell]E(\mathbb{F}_{q^d})$ is a $\mathbb{Z}/\ell\mathbb{Z}$-module of rank either one or two due to the structure of $E(\mathbb{F}_{q^d})$ described in Corollary 2.2, we see that such points are always common. Indeed, in the first case any point $Q \in E(\mathbb{F}_{q^d}) \setminus \{\mathcal{O}\}$ will suffice, while in the second case the probabilistic approach used for the MOV reduction may be applied.

Repeatedly sampling $Q \in E(\mathbb{F}_{q^d})$ and computing $\tau_\ell(G, Q)$ to test the condition $\tau_\ell(G, Q) \neq 1$ yields a desired point with an expected time complexity of $\Theta(q \log d)$ operations in $\mathbb{F}_{q^d}$. Once again, an index calculus algorithm may be used to solve the discrete logarithm in $\mathbb{F}_{q^d}^{\times}[\ell]$, for a time complexity dominated by $L_{q^d}[1/3, c]$. Sage code implementing the Frey-Rück reduction is given in Algorithm 5.6.

It is worth drawing a comparison between the Frey-Rück reduction and the MOV reduction. Firstly, practical implementations of the modified Tate pairing are often faster than those for the Weil pairing when the embedding degree d is small, since they usually only require one evaluation of Miller's algorithm, whereas the Weil pairing always requires two. However, when d is large, the exponentiation by $(q^d - 1)/\ell$ in computing the modified Tate pairing may result in it actually being outperformed by the Weil pairing. Another

Algorithm 5.6 Frey-Rück reduction

```
def frey_ruck_reduction(G, P, l, E):
    """
    G: a point on E of prime order
    P: a multiple of G
    l: the order of G
    E: an elliptic curve
    """
    q = E.base_field().order()
    g = 1
    while g == 1:
        Q = E.random_point()
        g = tate_pairing(E, l, G, Q)^((q - 1) // l)
    p = tate_pairing(E, l, P, Q)^((q - 1) // l)
    return p.log(g)
```

advantage of the Frey-Rück reduction is that when searching for a point Q that yields an isomorphism, we may directly use a point sampled from $E(\mathbb{F}_{q^d})$ to compute the modified Tate pairing, whereas the MOV reduction requires first multiplying such a point by some constant in $O(q^d)$ to transform it into a point in $E(\mathbb{F}_{q^d})[\ell]$, which usually exceeds the cost of exponentiation in the modified Tate pairing. A situation where the Frey-Rück reduction is always more appropriate is when $d = 1$, since the Weil pairing may require working over a much larger extension field of $\mathbb{F}_q$ in this case.

5.6 Anomalous curve reduction

Definition 5.2. $E/\mathbb{F}_q$ is said to be an *anomalous* elliptic curve if and only if $\#E(\mathbb{F}_q) = q$.

In the present section, we discuss an approach due to (Smart, 1999) for reducing the discrete logarithm problem on a subgroup $\langle G \rangle$ of order p of an anomalous elliptic curve $E/\mathbb{F}_q$ to the discrete logarithm problem on the additive group $\mathbb{F}_p^+$, which may be solved with time complexity polynomial in $\log p$.

In our discussion, we focus on the case of an elliptic curve $E/\mathbb{F}_p$, but similar results may be derived for the more general case of $E/\mathbb{F}_q$ (Washington, 2008).

To apply the reduction, we first find a p-adic lift $\tilde{E}/\mathbb{Q}_p$ of our elliptic curve. That is, we choose $\tilde{a}_4, \tilde{a}_6 \in \mathbb{Q}_p$ such that $\tilde{a}_4 \equiv a_4 \pmod{p}$ and $\tilde{a}_6 \equiv a_6 \pmod{p}$, yielding the elliptic curve $\tilde{E} : y^2 = x^3 + \tilde{a}_4 x + \tilde{a}_6$. We similarly compute p-adic lifts $\tilde{G}, \tilde{P} \in \tilde{E}(\mathbb{Q}_p)$ whose coordinates reduce to the respective coordinates of G and P modulo p, which can be achieved using *Hensel's lemma*.

The following theorem gives an explicit isomorphism from $E(\mathbb{F}_p)$ to $\mathbb{F}_p^+$ by passing through the p-adic lift $\tilde{E}(\mathbb{Q}_p)$.

Theorem 5.2. *There exists a function $\vartheta_p : [p]\tilde{E}(\mathbb{Q}_p) \to \mathbb{Q}_p$, called the* p-adic elliptic logarithm*, with the properties*

(a) $\vartheta_p([p]\tilde{P}) \equiv 0 \pmod{p}$

(b) $\vartheta_p([p]\tilde{P}) \equiv -\frac{([p]\tilde{P})_x}{([p]\tilde{P})_y} \pmod{p^2}$

(c) The map

$$E(\mathbb{F}_p) \to \mathbb{F}_p^+, \quad P \mapsto \frac{\vartheta_p([p]\tilde{P})}{p} \bmod p,$$

where $\tilde{P}$ is a p-adic lift of P, is an isomorphism.

Proof. See Section V.3 of (Blake *et al.*, 1999). □

The isomorphism in Theorem 5.2 is efficiently computable since we only need to store the results of calculations up to two p-adic digits. The reduction of the discrete logarithm to $\mathbb{F}_p^+$ may then be solved by computing $\log_G P = a/g$ in $\mathbb{F}_p$, where a and g are the images of P and G respectively under the isomorphism.

Algorithm 5.7 Anomalous curve reduction

```python
def elliptic_log(E, P):
    """
    E: an elliptic curve over GF(p)
    P: a point on E
    """
    p = E.base_field().order()
    Ep = E.base_extend(QQ).base_extend(Qp(p, 2))
    P_tilde = Ep.lift_x(ZZ(P[0]) + p * randint(0, p - 1))
    if P_tilde[1].residue() != P[1]:
        P_tilde *= -1
    return -(p * P_tilde)[0] / (p * P_tilde)[1]

def anomalous_curve_reduction(G, P, E):
    """
    G: a point on E of order p
    P: a multiple of G
    E: an elliptic curve over GF(p)
    """
    p = E.base_field().order()
    a = (elliptic_log(E, P) // p).residue()
    g = (elliptic_log(E, G) // p).residue()
    return ZZ(a / g)
```

Chapter 6

Elliptic curve cryptography

Suppose two parties, Alice and Bob, wish to communicate with each other securely over a public network. This is the type of problem which is solved by algorithms in the realm of *public-key cryptography*, where each party selects both a *secret key*, which they don't publish, as well as a corresponding *public key*, which they send to the other party over the public network.

In this chapter we describe a number of public-key cryptosystems based on group varieties for which the Diffie-Hellman problem is assumed to be hard, and hence the discrete logarithm problem is also assumed to be hard. In particular, the reader should keep in mind the case of an elliptic curve over a finite field. Multiplicative groups of finite fields are also still widely used in practice.

Henceforth we will assume that Alice and Bob have publicly agreed on a tuple of values (V, G, ℓ, H), where $V/\mathbb{F}_q$ is a group variety, G is an $\mathbb{F}_q$-point of order ℓ of V, and

$$H : \{0,1\}^* \times V \to \mathbb{F}_\ell$$

is a cryptographic hash function. The contents of this tuple are sometimes referred to as the *system parameters*, whereas Alice and Bob's public keys are known as the *user parameters*.

When V is an elliptic curve, the system parameters G and ℓ may be chosen efficiently using repeated sampling if the number of $\mathbb{F}_q$-points on the curve is known. As described in Section 4.2, Schoof's algorithm may be used to count the number of $\mathbb{F}_q$-points efficiently in this case.

In practice, the hash function H is often constructed from a general-purpose cryptographic hash function $\{0,1\}^* \to \{0,1\}^k$, where $k > \log_2 \ell$. We then simply define $H(m, P)$ as the result of applying this hash function to the concatenation of m and the bit-string representation of P, reduced modulo ℓ.

Note that the default hash function in Sage only has a codomain of 64 bits, and is not designed to be cryptographically secure. Furthermore, the default random number generator is also not cryptographically secure. The following code replaces the default functions `hash` and `randint` with cryptographically secure versions. The hash function has a codomain of 512 bits, which is more than sufficient for values of ℓ currently used in applications.

```
from Crypto.Hash import SHA512
from Crypto.Random import random

def hash(m):
    return ZZ(SHA512.new(str(m)).hexdigest(), 16)

def randint(a, b):
    return ZZ(random.randint(int(a), int(b)))
```

6.1 Diffie-Hellman key generation

All of the cryptosystems introduced in this chapter will use the same approach for generating a pair of secret and public keys, which we call *Diffie-Hellman key generation*.

For this approach, each party selects a random element $s \in \mathbb{F}_\ell$ to use as their secret key. They then calculate the point multiplication $K := [s]G$, which they use as their public key. We will use (s_A, K_A) and (s_B, K_B) to denote Alice and Bob's key pairs respectively. Algorithm 6.1 gives Sage code for generating a key pair from the system parameters in this way.

Note that given knowledge of the system parameters and a public key K, the problem of deriving the matching secret key s is an instance of the discrete logarithm problem, which we have assumed to be difficult. This is clearly a desirable property for a key pair to have.

Algorithm 6.1 Diffie-Hellman key generation

```
def diffie_hellman_key_generation(G, l):
    """
    M: the base point
    l: the order of G
    """
    s = randint(0, l - 1)
    K = s * G
    return s, K
```

6.2 Diffie-Hellman key-agreement scheme

Suppose that Alice and Bob wish to establish a common secret which is only known to them, while communicating only over a public network. A cryptosystem which allows one to do this is known as a *key-agreement scheme*. A typical use case of key-agreement schemes is to establish a common secret which can be used as the *symmetric key* for encrypting and decrypting subsequent communication using a *symmetric cryptosystem* such as the *Advanced Encryption Standard (AES)*. This is commonplace in practical applications, since the encryption and decryption procedures for symmetric cryptosystems are typically very fast to compute.

The *Diffie-Hellman key-agreement scheme* was originally proposed by (Diffie and Hellman, 1976), using group varieties of the form $\mathbb{F}_p^\times$. It was the first cryptosystem to be based on the Diffie-Hellman problem, hence the name of the problem. More generally, it was the first successful implementation of a public-key cryptosystem to be published.

To establish a common secret using this scheme, Alice simply computes the point multiplication $[s_A]K_B$ using her secret key and Bob's public key, and Bob similarly computes $[s_B]K_A$. These computation reduce to $[s_A]K_B = [s_A s_B]G$ and $[s_B]K_A = [s_B s_A]G$, which are clearly equal. Sage code for this procedure is given in Algorithm 6.2.

Assuming knowledge of the system parameters and Alice and Bob's public keys K_A and K_B, computing the shared secret $[s_A s_B]G$ is clearly equivalent to solving the Diffie-Hellman problem.

Algorithm 6.2 Diffie-Hellman key-agreement

```python
def diffie_hellman_key_agreement(s_A, K_B):
    """
    s_A: Alice's secret key
    K_B: Bob's public key
    """
    return s_A * K_B
```

6.3 ElGamal encryption scheme

Suppose that Alice wishes to send Bob a message $m \in \{0,1\}^*$ over a public network, in such a way that the contents can only be read by Bob. An *encryption scheme* is a cryptosystem which consists of both an *encryption* and a *decryption* algorithm. The encryption algorithm allows one to use Bob's public key K_A to transform the message m into a *cyphertext* c, from which someone can extract the original message m using the decryption algorithm if and only if they have knowledge of Bob's secret key s_B.

Since messages may be arbitrarily long, we will break m up into a sequence of messages of some specified maximum length, each of which is then to be encrypted individually. Thus, we assume without loss of generality that $m \in \{0,1\}^k$ for some appropriately chosen k.

We now describe two common variants of an encryption scheme known as *ElGamal encryption*, first published in (ElGamal, 1985).

6.3.1 First variant

In the first variant, we restrict the message m to have length at most $\log_2 \ell$ and then encode it as an element $m' \in \mathbb{F}_\ell$ by interpreting it as the binary representation of m'. A random element $e \in \mathbb{F}_\ell^\times$ is then selected as an *ephemeral key* – a key which is used only for a single encryption procedure and then discarded. We compute the first part of the ciphertext as the point multiplication $C_1 := [e]G$.

The second part of the ciphertext is computed by additively perturbing the message m' by the hash of the point multiplication $[e]K_B$. To this end, we use the empty message for the first argument of H to turn it into a hash function on V only. Concretely, we set $c_2 := m' + H(\emptyset, [e]K_B)$ for the second part of the ciphertext, yielding the full ciphertext $C := (C_1, c_2)$. Sage code for the encryption procedure is given in Algorithm 6.3.

Algorithm 6.3 ElGamal encryption (first variant)

```
def elgamal_encrypt(m, K_B, G, l):
    """
    m: the message (encoded as an element of GF(l))
    K_B: Bob’s public key
    G: the base point
    l: the order of G
    """
    e = randint(1, l - 1)
    C_1 = e * G
    c_2 = (m + hash(e * K_B)) % l
    return C_1, c_2
```

The decryption procedure for this variant proceeds by simply computing $c_2 - H(\emptyset, [s_B]C_1)$. This clearly yields m', since $[s_B]C_1 = [s_B e]G = [e]K_B$. Algorithm 6.4 gives the Sage code for the decryption procedure.

Algorithm 6.4 ElGamal decryption (first variant)

```
def elgamal_decrypt((C_1, c_2), s_B, l):
    """
    (C_1, c_2): the ciphertext
    s_B: Bob’s secret key
    l: the order of the base point
    """
    return (c_2 - hash(s_B * C_1)) % l
```

Note that given knowledge of the system parameters, $[e]G$ and K_B, computing $[s_B e]G$ without knowing s_B requires solving the Diffie-Hellman problem. Thus, without $[s_B]$, we cannot compute $H(\emptyset, [e]K_B)$ efficiently. Thus, assuming that H yields a value which is indistinguishable from an element selected uniformly at random from $\mathbb{F}_\ell$ when the input is uniformly distributed, c_2 is also indistinguishable from a uniformly distributed element of $\mathbb{F}_\ell$, and thus m' cannot be recovered.

Remark 6.1. The importance of not reusing the ephemeral key e should be emphasised. If Alice uses the same value of e to encrypt two messages m and $\bar{m}$ to Bob, then their respective representations m' and $\bar{m}'$ will be perturbed by the same hash, namely $H(\emptyset, [e]K_B)$. Computing $C_1 - \bar{C}_1$ then yields $m' - \bar{m}'$, which is certainly no longer guaranteed to be indistinguishable from a uniformly distributed element of $\mathbb{F}_\ell$. Furthermore, if someone discovers the contents of at least one message, all other messages which were encrypted using the same value of e become fully known. An attacker may even ask Alice to encrypt a specific message using Bob's public key, in what is known as a *chosen-message attack*, thereby rendering all other messages encrypted with the same ephemeral key known.

6.3.2 Second variant

The second variant of the ElGamal encryption scheme avoids using a hash function by encoding m as an $\mathbb{F}_q$-point on the group variety V. The approach followed for representing m as an element of V depends on the type of group.

Elements of the group variety $\mathbb{F}_q^\times$ may be encoding using a straightforward process. We restrict m to have length at most $\log_2 q$, which we interpret as the binary representation of an integer. The encoding of m is then taken to be the element of $\mathbb{F}_q^\times$ whose polynomial representation has the p-ary digits of this integer as its coefficients. This does not yield an element of $\mathbb{F}_q^\times$ for the message whose bits are all zero, but we may encode this message as the element of $\mathbb{F}_q^\times$ whose polynomial representation has $p - 1$ as all of its coefficients, since no other message will be encoded to this element as long as p is odd.

For an elliptic curve $E/\mathbb{F}_q$, we suggest a probabilistic approach for encoding messages. This approach fails to encode a given message with a probability of about 2^{-k}, where k is a parameter of the encoding scheme. Here we restrict m to have length at most $\log_2 q - k$, which we use as the least significant bits in a binary sequence with $\lfloor \log_2 q \rfloor$ bits in total. The k most significant bits of this binary sequence are then chosen randomly, with the resulting sequence being interpreted as an element $x \in \mathbb{F}_q$ as described for the group $\mathbb{F}_q^\times$, but without interpreting the sequence of zero bits specially. The k randomly chosen bits are simply ignored during the decoding procedure.

The element of $\mathbb{F}_q$ selected in this way may either fail to be the x-coordinate of a point on the elliptic curve or such a point may not be in the subgroup

$\langle G \rangle$ of $E(\mathbb{F}_q)$, in which case we repeat the procedure. The former happens exactly when $\left(\frac{f(x)}{\mathbb{F}_q}\right) = -1$, which has a probability of about 1/2. The latter has a probability of exactly $\ell/\#E(\mathbb{F}_q)$. Thus, we expect to have to randomly select k bits about $2\#E(\mathbb{F}_q)/\ell$ times, which is in $\Theta(1)$ under the assumption $\ell \in \Theta(\#E(\mathbb{F}_q))$. As previously mentioned, there is a probability of about 2^{-k} that no point in $\langle G \rangle$ will have m as the least significant bits of its x-coordinate, in which case we fail to encode m.

Once the x-coordinate of a point in $\langle G \rangle$ has been found, we may compute the square roots of $f(x)$ and randomly select one of them as its y-coordinate, thus obtaining a point M to use as the encoding of m. Alternatively, we may encode messages which are one bit longer by using the extra bit to decide which of the square roots of $f(x)$ to use as the encoding of the message.

The encryption algorithm proceeds in a similar fashion to the first variant. An ephemeral key $e \in \mathbb{F}_\ell^\times$ is selected at random, the point multiplications $C_1 := [e]G$ and $[e]K_B$ are calculated, and the message is additively perturbed to $C_2 := M + [e]K_B$. The pair (C_1, C_2) then serves as the ciphertext. Sage code for this procedure is given in Algorithm 6.5.

Algorithm 6.5 ElGamal encryption (second variant)

```
def elgamal_encrypt(M, K_B, G, l):
    """
    M: the message (encoded as a multiple of G)
    K_B: Bob's public key
    G: the base point
    l: the order of G
    """
    e = randint(1, l - 1)
    C_1 = e * G
    C_2 = M + e * K_B
    return C_1, C_2
```

The decryption procedure then simply computes $C_2 - [s_B]C_1$, which yields the message M since $[s_B]C_1 = [s_B e]G = [e]K_B$. The Sage code for decryption is given in Algorithm 6.6.

The second variant once again relies on the Diffie-Hellman problem so that it is difficult to compute $[e]K_B$ from C_1 and K_B. Its security also relies on the assumption that given C_1, the point $[e]K_B$ is indistinguishable from a

Algorithm 6.6 ElGamal decryption (second variant)

```
def elgamal_decrypt((C_1, C_2), s_B):
    """
    (C_1, C_2): the ciphertext
    s_B: Bob's secret key
    """
    return C_2 - s_B * C_1
```

uniformly distributed point in $\langle G \rangle$ without knowledge of s_B, so that $M + [e]K_B$ is therefore indistinguishable from a uniformly distributed point.

6.4 Schnorr signature scheme

Suppose Alice would like to send a message $m \in \{0,1\}^*$ to Bob over a public network, along with a *signature* which proves that the message has not been tampered with during transit. This is the problem solved by a *signature scheme*, which provides a *signing* algorithm which Alice may use to generate a signature for some message, and a *verification* algorithm which Bob may use to ensure that the message which he receives is the same as the message sent by Alice. For this to be possible, it must be difficult for a third party to forge Alice's signature for some message given examples of her signatures for other messages.

We describe a cryptosystem known as the *Schnorr signature scheme* due to (Schnorr, 1990). For Alice to sign a message m that she wishes to send to Bob, she randomly selects an emphemeral key $e \in \mathbb{F}_\ell$, computes the point multiplication $[e]G$, and sets the first part of her signature to the hash $s_1 := H(m, [e]G)$. The second part of her signature is simply computed as $s_2 := e - s_A s_1$. Finally, the signature is appended to the message, yielding the tuple (m, s_1, s_2). Sage code for this procedure is given in Algorithm 6.7.

Assuming that the hash function H is secure against inversion, it is not possible to gain knowledge of e from s_1. Furthermore, assuming s_1 is indistinguishable from a randomly selected element of $\mathbb{F}_\ell$ without knowledge of e, it is impossible to learn s_A from s_2. Someone who does not have knowledge of Alice's private key s_A can construct a valid value for s_1 corresponding to m by choosing their own ephemeral key e'. However, they will not be able

Algorithm 6.7 Schnorr signing

```
def schnorr_sign(m, s_A, G, l):
    """
    m: the message
    s_A: Alice's secret key
    G: the base point
    l: the order of G
    """
    e = randint(0, l - 1)
    s_1 = hash((m, e * G)) % l
    s_2 = (e - s_A * s_1) % l
    return m, s_1, s_2
```

to construct a matching value for s_2 without knowledge of s_A, so that Alice's signatures cannot be forged.

To verify that the signature (s_1, s_2) corresponds to the message m that was sent by Alice, Bob may do the following. He computes the point multiplications $[s_1]K_A$ and $[s_2]G$, then evaluates the hash $H(m, [s_1]K_A+[s_2]G)$. If this is equal to s_1, then he accepts that the message has been verified. Otherwise, he rejects it. This verification algorithm is given as Sage code in *Algorithm* 6.8.

Algorithm 6.8 Schnorr verification

```
def schnorr_verify((m, s_1, s_2), K_A, G, l):
    """
    m: the message
    (s_1, s_2): the signature
    K_A: Alice's public key
    G: the base point
    l: the order of G
    """
    return s_1 == hash((m, s_1 * K_A + s_2 * G)) % l
```

Since $[s_2]G = [e]G-[s_As_1]G = [e]G-[s_1]K_A$, we have that $[s_1]K_A+[s_2]G = [e]G$, showing that the verification procedure works for signatures generated by Algorithm 6.7. Furthermore, assuming that the hash function H is resistant against collisions, it is not possible to pick a different message m' such that $H(m', [s_1]K_A + [s_2]G)$ will evaluate to s_1.

Note that the Schnorr signature scheme does not rely on the Diffie-Hellman problem, although it does implicitly rely on the discrete logarithm problem due to the nature of Diffie-Hellman key pairs.

Chapter 7

Conclusion

In this thesis, we surveyed the mathematics underlying elliptic curve cryptography. In particular, we discussed the algorithmic aspects of computations involving elliptic curves over finite fields, thereby demonstrating that they are feasible objects to work with in a cryptographic setting, where the finite fields used are typically of a very large order.

By describing Schoof's algorithm, we showed that it is also possible to compute the number of points on an elliptic curve over a finite field in an efficient manner, allowing us to learn enough about the elliptic curve's group structure to enable its use in cryptographic algorithms.

After formulating the discrete logarithm and Diffie-Hellman problems on elliptic curves, we gave an overview of solutions to these problems. We saw that the known algorithms for solving the discrete logarithm problem on a general group variety are all infeasible when the number of points on the variety is a small multiple of a very large prime number, in which case the Pohlig-Hellman reduction cannot be used effectively, and computations using the baby-step giant-step algorithm or Pollard's algorithms are too cumbersome.

Furthermore, we discussed some reductions of the discrete logarithm problem to the additive or multiplicate group of a finite field which may only be applied to special types of elliptic curves. However, we saw that such elliptic curves are extremely rare, so that the reductions are not an obstacle to finding curves suitable for cryptographic use.

Finally, we presented a selection of cryptosystems based on the difficulty of solving the discrete logarithm and Diffie-Hellman problems on group varieties, for which elliptic curves are a suitable candidate.

Appendix A

Computing in the endomorphism ring modulo ℓ

In this appendix, we give Sage code for performing computations in the endomorphism ring of an elliptic curve $E/\mathbb{F}_q$ modulo the division polynomial ψ_ℓ, which is used in Schoof's algorithm.

```
class EndomorphismMod:
    def __init__(self, E, fx, fy, psi):
        R.<x> = E.base_field()['x'].quotient(psi)
        self.E = E
        self.fx = R(fx)
        self.fy = R(fy)
        self.psi = psi
        self.f = x^3 + E.a4() * x + E.a6()
        self.A = E.a4()

    def __add__(self, other):
        if isinstance(other, Integer):
            other *= IdentityEndomorphismMod(self.E, self.psi)
        if isinstance(other, ZeroEndomorphismMod):
            return self
        if self.fx == other.fx:
            if self.fy == other.fy:
                m = (3 * self.fx^2 + self.A) / (2 * self.fy * self.f)
            else:
                return ZeroEndomorphismMod(self.E, self.psi)
        else:
            div, inv, _ = xgcd((self.fx - other.fx).lift(), self.psi)
            if div.is_constant():
                m = (self.fy - other.fy) * inv
            else:
                raise ZeroDivisionError(div)
        fx = m^2 * self.f - self.fx - other.fx
        fy = m * (self.fx - fx) - self.fy
        return EndomorphismMod(self.E, fx, fy, self.psi)

    def __radd__(self, other):
        return self + other

    def __neg__(self):
        return EndomorphismMod(self.E, self.fx, -self.fy, self.psi)

    def __sub__(self, other):
        return self + -other

    def __rsub__(self, other):
        return -(self + -other)

    def __mul__(self, other):
        if isinstance(other, Integer):
            if other < 0:
                return -self * -other
            alpha = ZeroEndomorphismMod(self.E, self.psi)
            while other > 0:
                if other % 2 == 1:
                    alpha += self
                self += self
                other //= 2
            return alpha
        fx = self.fx.lift()(other.fx)
        fy = self.fy.lift()(other.fx) * other.fy
        return EndomorphismMod(self.E, fx, fy, self.psi)

    def __rmul__(self, other):
        return self * other

    def __pow__(self, n):
        alpha = IdentityEndomorphismMod(self.E, self.psi)
        while n > 0:
            if n % 2 == 1:
                alpha *= self
            self *= self
            n //= 2
        return alpha

    def __eq__(self, other):
        if isinstance(other, Integer):
            other *= IdentityEndomorphismMod(self.E, self.psi)
        if isinstance(other, ZeroEndomorphismMod):
            return False
        return self.fx == other.fx and self.fy == other.fy
```

```
class ZeroEndomorphismMod(EndomorphismMod):
    def __init__(self, E, psi):
        self.E = E
        self.psi = psi

    def __neg__(self):
        return self

    def __add__(self, other):
        return other

    def __mul__(self, other):
        return self

    def __eq__(self, other):
        if isinstance(other, Integer):
            other *= EndomorphismMod(self.E, x, 1, self.psi)
        return isinstance(other, ZeroEndomorphismMod)
```

```
class IdentityEndomorphismMod(EndomorphismMod):
    def __init__(self, E, psi):
        _.<x> = E.base_field()[’x’].quotient(psi)
        EndomorphismMod.__init__(self, E, x, 1, psi)
```

```
class FrobeniusEndomorphismMod(EndomorphismMod):
    def __init__(self, E, psi):
        q = E.base_field().order()
        _.<x> = E.base_field()[’x’].quotient(psi)
        f = x^3 + E.a4()*x + E.a6()
        EndomorphismMod.__init__(self, E, x^q, f^((q - 1) // 2), psi)
```

Bibliography

Apostol, T.M. (1976). *Introduction to Analytic Number Theory.* No. 8 in Undergraduate Texts in Mathematics. Springer.

Balasubramanian, R. and Koblitz, N.J. (1998). The Improbability That an Elliptic Curve Has Subexponential Discrete Log Problem under the Menezes-Okamoto-Vanstone Algorithm. *Journal of Cryptology*, vol. 11, no. 141, pp. 141–145.

Blake, I.F., Seroussi, G. and Smart, N.P. (1999). *Elliptic Curves in Cryptography.* No. 265 in London Mathematical Society Lecture Note Series, 1st edn. Cambridge University Press.

Cohen, H. and Frey, G. (2015). *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Discrete Mathematics and Its Applications, 2nd edn. Chapman and Hall/CRC.

Diffie, W. and Hellman, M. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654.

ElGamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472.

Floyd, R.W. (1967). Nondeterministic Algorithms. *Journal of the ACM*, vol. 14, no. 4, pp. 636–644.

Frey, G. and Rück, H.-G. (1994). A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, vol. 62, no. 206, pp. 865–874.

Galbraith, S.D. (2012). *Mathematics of Public Key Cryptography.* 1st edn. Cambridge University Press.

Koblitz, N.J. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209.

Lichtenbaum, S. (1969). Duality Theorems for Curves over p-Adic Fields. *Inventiones Mathematicae*, vol. 7, pp. 120–136.

Menezes, A.J., Okamoto, T. and Vanstone, S.A. (1993). Reducing Elliptic Curve Logarithms in a Finite Field. *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1639–1646.

Miller, V.S. (1986*a*). Short Programs for Functions on Curves. Available: https://crypto.stanford.edu/miller.

Miller, V.S. (1986*b*). Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.), *Advances in Cryptology – CRYPTO '85 Proceedings*, vol. 218 of *Lecture Notes in Computer Science*, pp. 417–426. Springer.

Pohlig, S. and Hellman, M. (1978). An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance. *IEEE Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110.

Pollard, J.M. (1978). Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924.

Schnorr, C.P. (1990). Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (ed.), *Advances in Cryptology – CRYPTO '89 Proceedings*, vol. 435 of *Lecture Notes in Computer Science*, pp. 239–252. Springer.

Schoof, R. (1985). Elliptic Curves over Finite Fields and the Computation of Square Roots mod p. *Mathematics of Computation*, vol. 44, no. 170, pp. 483–494.

Shanks, D. (1971). Class Number, a Theory of Factorization and Genera. *Proceedings of Symposium of Pure Mathematics*, vol. 20, pp. 415–440.

Silverman, J.H. (2009). *The Arithmetic of Elliptic Curves.* No. 106 in Graduate Texts in Mathematics, 2nd edn. Springer.

Smart, N.P. (1999). The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, vol. 12, no. 3, pp. 193–196.

Sutherland, A. (2015). 18.783 Elliptic Curves. Available: http://ocw.mit.edu/.

Tate, J. (1958). WC-Groups over p-Adic Fields. *Séminaire Bourbaki*, vol. 4, pp. 265–277.

van Oorschot, P.C. and Wiener, M.J. (1996). Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, vol. 12, no. 1, pp. 1–28.

Washington, L.C. (2008). *Elliptic Curves: Number Theory and Cryptography.* Discrete Mathematics and Its Applications, 2nd edn. Chapman and Hall/CRC.

Weil, A. (1940). Sur les Fonctions Algébriques á Corps de Constantes Fini. *Les Comptes rendus de l'Académie des Sciences*, vol. 210, pp. 592–594.