university of groningen / faculty of science and engineering

Efficient and secure elliptic curves

Abstract

In this paper, we study various factors that affect the efficiency of elliptic curve cryptosystems for fixed security levels of 128, 192 and 256 bits of security, with a focus on elliptic curves over finite fields $\mathbf{F}_p$ with p elements, where p is a prime number. Since elliptic curve cryptography involves many algebraic operations over the field $\mathbf{F}_p$, it is important to choose p for which modular arithmetic is relatively efficient. The most important factor regarding efficiency is the model of the curve. Group operations such as addition, inversion, and doubling are more effective in some models. We introduce the Weierstrass model, the (twisted) Edwards model, the Montgomery model and the Hessian model. We then compare their cryptographic efficiency and security. We mainly focus on the 2016 paper of Bos, Costello, Longa and Naehrig, in which the authors present efficient and secure elliptic curve families for each model. After this analysis, we present explicit examples of elliptic curves suitable for cryptography, in terms of efficiency and security.

Bachelor's Project Mathematics - July 2021

Student: D.J. Davies-Batista

Primary supervisor: Dr. M. Djukanović

Secondary supervisor: Dr. P. Kılıçer

Contents

Notation

$\mathbf{F}_p$ a field with p elements where p is prime

$\mathbf{F}_q$ a field with q elements

$\mathbf{F}_q^\times$ the multiplicative group of $\mathbf{F}_q$

$\overline{\mathbf{K}}$ an algebraic closure of a field $\mathbf{K}$

$O(g(x))$ a function bounded by $Mg(x)$ for a constant $M > 0$ and all x sufficiently large

$o(g(x))$ a function bounded by $Mg(x)$ for all constants $M > 0$ and all x sufficiently large

$\mathrm{L}_n[a, b]$ the function $\exp((b + o(1))(\ln n)^a(\ln \ln n)^{1-a})$ for $0 \leq a \leq 1$ and $b > 0$

1 Introduction

We begin with a short historical background of modern public-key cryptography. Before public-key cryptography, methods of encryption were constrained to private-key cryptographic methods in which distinct parties were required to exchange some predetermined secret key through insecure means, be that through in-person meetings or courier pigeons. Already, the logistic burden of private-key methods is evident. With this in mind, many attempted to develop methods of producing a shared secret key over a public channel. The first hint of this was in 1874 when William Jevons published *The Principles of Science* in which he wrote:

"Can the reader say what two numbers multiplied together will produce the number 8616460799? I think it unlikely that anyone but myself will ever know."[1, p. 141]

To William's dismay, even a poorly optimised implementation of Lenstra's elliptic curve factorisation algorithm yields

$$8616460799 = 89681 \cdot 96079.$$

Nevertheless, it is impressive that Jevons proposed the problem whose difficulty ensures the security of what would, almost a century later, become one of the most well-known methods of public-key cryptography: i.e. the RSA cryptosystem [2], which was published in 1973, whose security relies on the difficulty of integer factorisation.

Only a year preceding the release of the RSA cryptosystem, another method of public-key cryptography was introduced, which is now known as the Diffie-Hellman key exchange protocol [3]. Though both the RSA cryptosystem and Diffie-Hellman key exchange protocol are methods of securely exchanging secret keys over public channels, they have notable and important differences. As already mentioned, the security of the former is reliant on the difficulty of integer factorisation, an easily posed problem with currently no easy solution. With the rapid improvement of hardware on which one may perform integer factorisation, as well as the rapid development of quantum computing, the RSA cryptosystem has fallen out of favour over the previous few decades, with the Diffie-Hellman key exchange protocol taking its place. This is in part due to the problem on which the security of the latter relies, that of computing discrete logarithms. There is no good reason to believe that the discrete logarithm problem is one whose potential weaknesses will be made clear in the near future.

The Diffie-Hellman key exchange protocol makes use of computing integer multiples (or integer exponents if the group in question is multiplicative) in a cyclic group $(G, +, 0)$. One should take G such that computing integer multiples are efficient. As for the security of performing the Diffie-Hellman key exchange protocol, one would ideally choose G such that the discrete logarithm problem in G is difficult to solve: i.e. given P and $Q = [k]P$, one

should not be able to find the smallest positive integer k satisfying this equality easily. To satisfy both criteria, one might earlier have looked to the multiplicative group of integers modulo some prime p but with the development of the index calculus algorithm and its adaptations, this is not advised. Convenient alternatives are groups determined by elliptic curves over finite fields.

Why elliptic curves?

To convince someone that they should consider elliptic curves over finite fields for the Diffie-Hellman key exchange protocol, one should present both efficient methods of computing integer multiples in elliptic curve groups and should be able to show that the discrete logarithm problem in elliptic curve groups is sufficiently difficult to solve, which is precisely what will be considered in this paper.

Elliptic curves over finite fields have other uses. For example in the Lenstra elliptic curve factorisation method [4], a sub-exponential running time integer factorisation algorithm, or in primality testing algorithms such as the Atkin-Morain elliptic curve primality test algorithm [5].

Outlook

We begin with a list of preliminaries of projective geometry, required to introduce elliptic curves in sufficient detail. Then, the chord-tangent law is applied to elliptic curves in order to derive the elliptic curve addition law, which in turn will act as the group law. Integer multiplication in elliptic curve groups is then considered, with some efficient methods for computing such integer multiples in a general elliptic curve group. The underlying group structure is then treated, including bounds on the group order and some isomorphism relations. We detail the Diffie-Hellman key exchange protocol over elliptic curve groups and the difficulty of solving the discrete logarithm problem in such groups. Methods of securely computing large integer multiples are discussed with a particular interest in omitting vulnerabilities to side-channel attacks. After this, well known elliptic curve models are considered, with an analysis of the efficiency of algorithms for addition on elliptic curves in such models, along with the security of posing the discrete logarithm problem. We conclude by constructing elliptic curves in the Weierstrass model that attain 32, 64 and 96 bits of security along with an explanation the construction process.

2 Preliminaries

To provide a sufficiently detailed introduction to elliptic curves, we must first discuss the relevant preliminary theory. We begin by defining two-dimensional affine space and then two-dimensional projective space, on which we embed elliptic curves.

2.1 The two-dimensional affine and projective spaces

Definition 2.1. *The two-dimensional affine space* $\mathbf{A}^2(\mathbf{K})$ *over a field* $\mathbf{K}$ *is the set of all tuples in* $\mathbf{K} \times \mathbf{K}$*, i.e.*

$$\mathbf{A}^2(\mathbf{K}) = \{(x, y) : x, y \in \mathbf{K}\}.$$

Definition 2.2. *The two-dimensional projective space* $\mathbf{P}^2(\mathbf{K})$ *over a field* $\mathbf{K}$ *is the set of equivalence classes of* $\mathbf{K}^3 \backslash \{(0, 0, 0)\}$ *under the equivalence relation* $\sim$ *where* $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ *if and only if there exists a non-zero scalar* $\lambda \in \mathbf{K}$ *such that* $(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_2)$.

The equivalence class of a tuple $(x, y, z) \in \mathbf{K}^3 \backslash \{(0, 0, 0)\}$ is the projective point $(x : y : z) \in \mathbf{P}^2(\mathbf{K})$. In $\mathbf{A}^2(\mathbf{K})$ lines intersect at a unique point if and only if they are not parallel. The construction of $\mathbf{P}^2(\mathbf{K})$ helps to ensure that parallel lines in projective space intersect at a unique point. Embedding $\mathbf{A}^2(\mathbf{K})$ in $\mathbf{P}^2(\mathbf{K})$ can be done by mapping each affine point $(x, y) \in \mathbf{A}^2(\mathbf{K})$ to the projective point $(x : y : 1) \in \mathbf{P}^2(\mathbf{K})$. Projective points of the form $(x : y : 0)$ can be thought of as points at infinity, forming a line at infinity given by the equation $z = 0$, which all lines in $\mathbf{P}^2(\mathbf{K})$ intersect. The notion of points at infinity will prove useful later on when we consider elliptic curve group structure. With this in mind, $\mathbf{P}^2(\mathbf{K})$ can be thought of as $\mathbf{A}^2(\mathbf{K})$ along with the line at infinity. Though affine coordinates provide an advantage in their intuitive interpretation, there are computational processes where the use of projective coordinates proves to be more efficient [6].

2.2 Projective curves

Definition 2.3. *Given a field* $\mathbf{K}$ *a non-constant polynomial* $F(x_1, \ldots, x_n) \in \mathbf{K}[x_1, \ldots, x_n]$ *is homogeneous of degree* d *if the sum of the exponents of each monomial of* $F(x_1, \ldots, x_n)$ *is* d.

Note that for a homogeneous polynomial in three variables $F = F(x, y, z)$ of degree d we necessarily have $F(\lambda x, \lambda y, \lambda z) = \lambda^d F(x, y, z)$ for all $\lambda \in \mathbf{K}$. A convenient consequence of this is that if $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ then (x_1, y_1, z_1) is a zero of F if and only if (x_2, y_2, z_2) is a zero of F.

Definition 2.4. *A projective curve* C *over a field* $\mathbf{K}$ *is the set of zeros in* $\mathbf{P}^2(\overline{\mathbf{K}})$ *of a non-constant homogeneous polynomial in three variables* $F(x, y, z) \in \mathbf{K}[x, y, z]$ *whose set of* $\mathbf{K}$*-rational points on* C *is given by*

$$C(\mathbf{K}) := \{(x : y : z) \in \mathbf{P}^2(\mathbf{K}) : F(x, y, z) = 0\},$$

where $F(x, y, z)$ *is irreducible over* $\overline{\mathbf{K}}$.

Notation 2.1. *For the remainder of this paper, we write* $C/\mathbf{K}$ *to denote a projective curve* C *given by a polynomial whose coefficients are in a field* $\mathbf{K}$.

Another property we are required to discuss before introducing elliptic curves is that of singular points of projective curves.

Definition 2.5. *For a field* $\mathbf{K}$ *a singular point on a projective curve* $C/\mathbf{K} : F(x, y, z) = 0$ *is a point* $P = (x : y : z) \in C(\overline{\mathbf{K}})$ *at which*

$$\frac{\partial F}{\partial x}(P) = \frac{\partial F}{\partial y}(P) = \frac{\partial F}{\partial z}(P) = 0,$$

where $\frac{\partial}{\partial x}x^n = nx^{n-1}$ *for all* $n \in \mathbf{Z}$ *and the usual rules of derivatives apply. We call a projective curve* $C/\mathbf{K} : F(x, y, z) = 0$ *non-singular if its set of* $\overline{\mathbf{K}}$*-rational points contains no singular points, otherwise we call* $C/\mathbf{K}$ *singular.*

A consequence of the property non-singularity is that the polynomial $F(x, y, z)$ of degree d, by which a non-singular projective curve C over a field $\mathbf{K}$ is given, necessarily has d distinct zeros in $\overline{\mathbf{K}}$.

Example 2.1. *For a field* $\mathbf{K}$ *the projective curve* $C/\mathbf{K}$ *given by*

$$F(x, y, z) = x^3 - y^2z$$

has a singular point at $(0 : 0 : 1)$*. Since* $(0 : 0 : 1)$ *is* $\overline{\mathbf{K}}$*-rational we have that* $C/\mathbf{K}$ *is singular.*

Example 2.2. *A Montgomery curve* $M_{A,B}$ *is a projective curve over a field* $\mathbf{K}$ *given by*

$$F(x, y, z) = x^3 + Ax^2z + xz^2 - By^2z$$

where $A, B \in \mathbf{K}$ *such that* $B(A^2-4) \neq 0$*. After computing the necessary partial derivatives, it is clear that* $M_{A,B}/\mathbf{K}$ *is non-singular.*

2.3 Birational equivalence

With the introduction of projective curves in place, we can now consider an important notion of 'equivalent' projective curves: birationally equivalent projective curves.

Definition 2.6. *Projective curves* $C_1/\mathbf{K}$ *and* $C_2/\mathbf{K}$ *over a field* $\mathbf{K}$ *are birationally equivalent if there exists a rational map* $\psi : C_1(\overline{\mathbf{K}}) \to C_2(\overline{\mathbf{K}})$ *defined at all but finitely many points in* $C_1(\overline{\mathbf{K}})$ *such that an inverse rational map* $\psi^{-1} : C_2(\overline{\mathbf{K}}) \to C_1(\overline{\mathbf{K}})$ *exists defined at all but finitely many points in* $C_2(\overline{\mathbf{K}})$*. Such a map* ψ *is referred to as a birational map.*

It is not immediately clear from the definition, nor even the explicit form of a given birational map, but birational maps preserve the underlying structure of a projective curve. As such, we think of birationally equivalent projective curves $C_1/\mathbf{K}$ and $C_2/\mathbf{K}$ as, in a sense, the same. We will see later on that birational maps come in handy when comparing the efficiency of arithmetic on birationally equivalent elliptic curves.

Example 2.3. *Given a Montgomery curve $M_{A,B}/\mathbf{K}$ over a field $\mathbf{K}$ given by*

$$By^2 = x^3 + Ax^2 + x,$$

the map

$$\psi : (x, y) \mapsto \left(\frac{x}{y}, \frac{x-1}{x+1}\right) = (X, Y)$$

defines a birational map $M_{A,B} \to \xi_{a,d}$, with change of coefficients $a = \frac{A+2}{B}$ and $d = \frac{A-2}{B}$, where $\xi_{a,d}$ is the (twisted) Edwards curve given by

$$aX^2 + Y^2 = 1 + dX^2Y^2,$$

with inverse map $\xi_{a,d} \to M_{A,B}$ given by

$$\psi^{-1} : (X, Y) \mapsto \left(\frac{1+Y}{1-Y}, \frac{1}{X} \cdot \frac{1+Y}{1-Y}\right) = (x, y).$$

As such, any Montgomery curve is birationally equivalent to a (twisted) Edwards curve, and vice versa.

2.4 Elliptic curves

Now that the preliminary theory is in place, we are able to provide a suitably detailed definition of an elliptic curve.

Definition 2.7. *Up to birational equivalence, an elliptic curve $E/\mathbf{K}$ is a non-singular projective curve over a field $\mathbf{K}$ of genus 1 with a fixed base point $\mathcal{O} \in E(\mathbf{K})$.*

Remark 2.1. *The genus g of a projective curve $C/\mathbf{K}$ over a field $\mathbf{K}$ defined by a polynomial of degree d is given by the* genus-degree formula

$$g = \frac{1}{2}(d-1)(d-2) - s$$

where s is the number of singularities of $C/\mathbf{K}$.

A more detailed explanation of the genus of a projective curve can be found in [7, p. 67-86].

Example 2.4. *Montgomery curves, discussed in* Example 2.2*, have degree* 3 *and no singular points and so their genus is* 1*, according to the genus-degree formula. We see that a Montgomery curve, with the fixed base point $\mathcal{O}_M = (0 : 1 : 0)$, is an elliptic curve.*

A common misconception of elliptic curves is that they are necessarily given by polynomials of degree 3, but this is not always the case; for example (twisted) Edwards curves

which are discussed in Section 7.3. Such a misconception can be accredited to the following theorem.

Theorem 2.1. *An elliptic curve $E_1/\mathbf{K}$ over a field $\mathbf{K}$ is birationally equivalent to an elliptic curve $E_2/\mathbf{K}$ over $\mathbf{K}$ given by a long Weierstrass equation*

$$y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3$$

for $a_1, a_2, a_3, a_4, a_6 \in \mathbf{K}$ with base point $\mathcal{O} = (0 : 1 : 0)$.

Proof. Consider Proposition III.3.3 of [8]. □

The motivation behind the base point $\mathcal{O} = (0 : 1 : 0)$ is as follows. For an elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$, a projective point $(x : y : z) \in E(\overline{\mathbf{K}})$ with $z \neq 0$ can be 'dehomogonized' to obtain an affine point $(x/z, y/z) \in \mathbf{A}_2(\overline{\mathbf{K}})$. The set of all such points can be thought of as the 'affine part' of $E(\overline{\mathbf{K}})$. The only points on $E(\overline{\mathbf{K}})$ that remain are of the form $(x : y : 0)$ with x and y not both zero, which lie on the line at infinity. Substituting any point on the line at infinity into a long Weierstrass equation yields $x^3 = 0$ and so $x = 0$. Since the curve intersects the line at infinity at three points, all of which are the same, we have only one point at infinity. As such, the only point that we are unable to 'dehomogenize' is $(0 : y : 0) = (0 : 1 : 0)$. This point is clearly $\mathbf{K}$-rational and a brief calculation shows that it is non-singular, and so $\mathcal{O} = (0 : 1 : 0) \in E(\mathbf{K})$.

Remark 2.2. *For an elliptic curve $E/\mathbf{K} : F(x, y, z) = 0$ over a field $\mathbf{K}$ we write $E(\mathbf{K})$ in affine form as*

$$E(\mathbf{K}) := \{(x, y) \in \mathbf{A}^2(\mathbf{K}) : f(x, y) = 0\} \cup \{\mathcal{O}\}$$

where $f(x, y) = F(x, y, 1)$ for all projective points $(x : y : z)$ with $z \neq 0$.

For this paper, it is enough to consider only the affine form of the equation by which an elliptic curve is given. As such, we write a long Weierstrass equation in affine form as

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

with $a_1, a_2, a_3, a_4, a_6 \in \mathbf{K}$, and $\mathcal{O} = (0 : 1 : 0)$ left implicitly defined.

A consequence of the property of non-singularity of elliptic curves is that any elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$ given by $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ has non-zero discriminant given by

$$\Delta_E := -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6,$$

where $b_2 = a_1^2+4a_2$, $b_4 = 2a_4+a_1a_3$, $b_6 = a_3^2+4a_6$ and $b_8 = a_1^2a_6+4a_2a_6-a_1a_3a_4+a_2a_3^2-a_4^2$.

For a field $\mathbf{K}$ of characteristic 2, the shortest form the equation by which an elliptic curve $E/\mathbf{K}$ is given can take is as in Theorem 2.1. Fortunately, over a field $\mathbf{K}$ of characteristic not 2, an elliptic curve $E_1/\mathbf{K}$ is birationally equivalent to an elliptic curve $E_2/\mathbf{K}$ given by

an equation consisting of fewer terms than that in Theorem 2.1, eliminating the need for the a_1xy and a_3y terms.

Lemma 2.1. *An elliptic curve $E_1/\mathbf{K}$ over a field $\mathbf{K}$ of characteristic not 2 given by*

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

is birationally equivalent to an elliptic $E_2/\mathbf{K}$ given by

$$Y^2 = X^3 + AX^2 + BX + C$$

with $A, B, C \in \mathbf{K}$.

Proof. Adding an additional term to both sides and completing the square yields the desired result, as shown in [9, p. 10]. □

For a field $\mathbf{K}$ of characteristic not 2 or 3, an elliptic curve $E/\mathbf{K}$ is birationally equivalent to an elliptic curve $E_{A,B}/\mathbf{K}$ given by an equation consisting of fewer terms than in Lemma 2.1, eliminating the need for the aX^2 term.

Lemma 2.2. *An elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$ of characteristic not 2 or 3 given by*

$$Y^2 = X^3 + AX^2 + BX + C$$

is birationally equivalent to an elliptic curve $E_{a,b}/\mathbf{K}$ given by a short Weierstrass equation

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

with $a, b \in \mathbf{K}$.

Proof. Consider [9, p. 10]. □

As might be expected, the number of terms in the discriminant of an elliptic curve given by a short Weierstrass equation, $E/\mathbf{K} : y^2 = x^3 + ax + b$, reduces heavily, yielding only two terms and is given by

$$\Delta_{E_{A,B}} := 4a^3 + 27b^2.$$

The importance of this expression is made clear when considering j-invariants of an elliptic curves given by a short Weierstrass equation.

Remark 2.3. *Strictly speaking, the discriminant includes an additional factor and is of the form $-16(4a^3 + 27b^2)$, but since the field over which an elliptic curve given by a short Weierstrass equation is taken is of characteristic not 2 or 3 we see that the inclusion of this factor has no effect on whether or not the discriminant is non-zero (and so whether or not $E/\mathbf{K}$ is non-singular), so it is usually left unwritten.*

2.5 Elliptic curve group structure

Before describing elliptic curve group structure we discuss the geometric interpretation of elliptic curve addition, a process in which we take two points on an elliptic curve to produce a third point on the curve, which in turn is the elliptic curve group law. We then derive explicit formulae for elliptic curve addition on an elliptic curve given by a long Weierstrass equation.

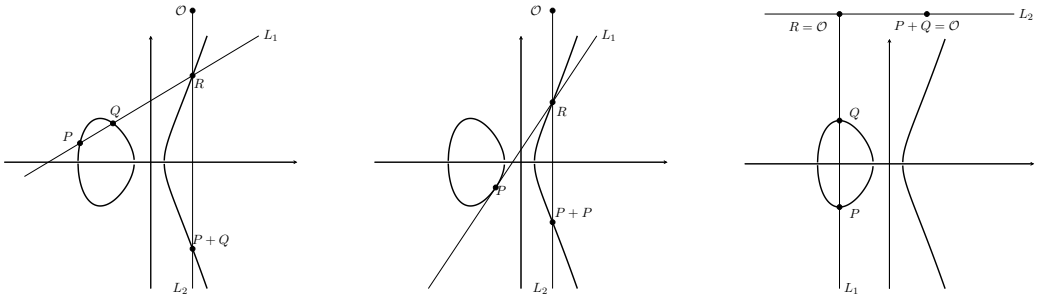
The chord-tangent law

Historically, given a projective curve $C/\mathbf{K}$ over a field $\mathbf{K}$, it has been of interest to try to find a way of using known $\mathbf{K}$-rational points P and Q to find additional $\mathbf{K}$-rational points in $C(\overline{\mathbf{K}})$. To do this, Bézout's theorem for plane curves is utilised.

Theorem 2.2. *If $C_1/\mathbf{K}$ and $C_2/\mathbf{K}$ are projective curves over a field $\mathbf{K}$ given by polynomials of degrees d_1 and d_2 respectively, which do not have a common 'sub-curve', then the number of intersections of $C_1/\overline{\mathbf{K}}$ and $C_2/\overline{\mathbf{K}}$ is $d_1 d_2$.*

Proof. Consider [10]. □

With Bézout's theorem in mind, given $\mathbf{K}$-rational points P and Q on an elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$, a line L_1 through P and Q must intersect $E(\overline{\mathbf{K}})$ at a third, not necessarily distinct, point R. If $P = Q$ then L_1 is taken to be the line tangent to E at P. Otherwise L_1 is the necessarily unique line passing through the distinct points P and Q. In either case, L_1 intersects $E(\overline{\mathbf{K}})$ at P, Q and a third point R. Using this third point R we construct the line L_2 through R and $\mathcal{O}$ which intersects $E(\overline{\mathbf{K}})$ at R, $\mathcal{O}$ and a third point S.



From here Elliptic curve addition, which we denote by $+$, can be defined as follows. Given $\mathbf{K}$-rational points P and Q, along with the third point of intersection R of the line L_1 we require that

$$P + Q + R = \mathcal{O},$$

i.e. we require that the line passing through P, Q and R intersects $E(\overline{\mathbf{K}})$ precisely these three points and $\mathcal{O}$. As such, the third intersection S of the line L_2 through R and $\mathcal{O}$ is

precisely the sum $P+Q$. Note that if $R=\mathcal{O}$ then L_2 is the line at infinity from $\mathcal{O}$ to itself which will only intersect $E(\overline{\mathbf{K}})$ at $\mathcal{O}$ and so $P+Q=\mathcal{O}$. Otherwise $R=(x,y)$ is an affine point and $P+Q=-R$ is also affine.

Explicit formulae for elliptic curve addition

For an elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$ given by

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6, \tag{1}$$

let P and Q be $\mathbf{K}$-rational points on E. In order to derive explicit formulae for the sum $P+Q$ in $E(\overline{\mathbf{K}})$, we first discuss the third intersection $S=-R$ of the line L_2 through a $\mathbf{K}$-rational point R and $\mathcal{O}$.

If $R=\mathcal{O}$ then L_2 is the line at infinity and so the third point of intersection is simply $-R=\mathcal{O}$. Otherwise $R=(x_1,y_1)$ and $-R=(x_2,y_2)$ are affine and the line L_2 through R and $\mathcal{O}$ is given by $x=x_1$ which we substitute into (1) to obtain

$$y^2+(a_1x_1+a_3)y+(-x_1^3-a_2x_1^2-a_4x_1-a_6)=0. \tag{2}$$

Since y_1 and y_2 are roots of (2), Vieta's formulae yield

$$y_2=-y_1-a_1x_1-a_3.$$

Hence, the third point of intersection of L_2 in $E(\overline{\mathbf{K}})$ is given by $-R=(x_2,y_2)$ where

$$\begin{aligned} x_2&=x_1\\ y_2&=-y_1-a_1x_1-a_3. \end{aligned}$$

From here we can easily derive a formula for $P+Q$ with $P\neq Q$. The case $P=\mathcal{O}$ is detailed precisely above, and so, without further explanation, we have $P+Q=Q$. A similar argument is made for the case $Q=\mathcal{O}$ which yields $P+Q=P$.

If $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ are affine and $x_1=x_2$ then the intersections in $E(\overline{\mathbf{K}})$ of the line L_1 through P and Q are P, Q and $R=\mathcal{O}$. In this case the line L_2 through R and $\mathcal{O}$ is the line at infinity and so $P+Q=\mathcal{O}$. Otherwise $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ are affine and $x_1\neq x_2$ then the line L_1 through both points is given by

$$y=\lambda(x-x_1)+y_1 \tag{3}$$

where $\lambda=(y_2-y_1)(x_2-x_1)^{-1}$. To find the third point of intersection $R=(x_3,y_3)$ on L_1, we substitute (3) into (1) to obtain

$$(\lambda(x-x_1)+y_1)^2+(a_1x+a_3)(\lambda(x-x_1)+y_1)=x^3+a_2x^2+a_4x+a_6. \tag{4}$$

Since x_1, x_2 and x_3 are roots of (4), Vieta's formulae yield

$$x_3 = \lambda^2 - x_1 - x_2 + a_1\lambda - a_2$$

and so

$$y_3 = \lambda(x_3 - x_1) + y_1.$$

Finally, the line L_2 through $R = (x_3, y_3)$ and $\mathcal{O}$ intersects $E(\overline{\mathbf{K}})$ at R, $\mathcal{O}$ and $P + Q = (x_3, y_3)$ where

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 + a_1\lambda - a_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3, \end{aligned}$$

by Bézout's theorem.

We now consider the case $P = Q$ in which computing the sum $P + Q = P + P$ can be thought of as doubling P. If $P = Q = \mathcal{O}$ then drawing any line through P and $\mathcal{O}$ yields the line at infinity and so $P + Q = \mathcal{O}$. If $P = Q = (x_1, y_1)$ is affine then the line L_1 is taken to be the line tangent to $E(\overline{\mathbf{K}})$ at P given by

$$y = \lambda(x - x_1) + y_1 \tag{5}$$

where

$$\lambda = (\partial F/\partial x)(\partial F/\partial y)^{-1} = (3x^3 + 2a_2x + a_4 - a_1y)(2y + a_1x + a_3)^{-1}.$$

Upon substituting (5) into (1) we once again have

$$(\lambda(x - x_1) + y_1)^2 + (a_1x + a_3)(\lambda(x - x_1) + y_1) = x^3 + a_2x^2 + a_4x + a_6. \tag{6}$$

Since x_1 is a double root and x_3 is a root of (6), Vieta's formulae yield

$$x_3 = \lambda^2 - 2x_1 + a_1\lambda - a_2$$

and so

$$y_3 = \lambda(x_3 - x_1) + y_1.$$

Finally, the line L_2 through $R = (x_3, y_3)$ and $\mathcal{O}$ intersects $E(\overline{\mathbf{K}})$ at R, $\mathcal{O}$ and $P + Q = (x_3, y_3)$ where

$$\begin{aligned} x_3 &= \lambda^2 - 2x_1 + a_1\lambda - a_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3, \end{aligned}$$

by Bézout's theorem.

We summarise the formulae for elliptic curve addition on elliptic curves given by a long Weierstrass equation in affine coordinates in Algorithm 1.

Algorithm 1: Addition in $(E(\overline{\mathbf{K}}), +, \mathcal{O})$, in affine coordinates where $E/\mathbf{K} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.

Input: $P, Q \in E(\mathbf{K})$ such that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
Output: $P + Q \in E(\overline{\mathbf{K}})$.

if $P = Q$ **then**
  if $P = \mathcal{O}$ *or* $y_1 = 0$ **then**
    return $\mathcal{O}$
  $\lambda \leftarrow (3x_1^2 + 2a_2x_1 - a_1y_1 + a_4)(2y_1 + a_1x_1 + a_3)^{-1}$ `// this step invokes the extended Euclidean algorithm`
  $x_3 \leftarrow \lambda^2 - 2x_1 + a_1\lambda - a_2$
  $y_3 \leftarrow \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3$
  return (x_3, y_3)
else
  if $P = \mathcal{O}$ **then**
    return Q
  if $Q = \mathcal{O}$ **then**
    return P
  if $x_1 = x_2$ **then**
    return $\mathcal{O}$
  $\lambda \leftarrow (y_2 - y_1)(x_2 - x_1)^{-1}$ `// this step invokes the extended Euclidean algorithm`
  $x_3 \leftarrow \lambda^2 - x_1 - x_2 + a_1\lambda - a_2$
  $y_3 \leftarrow \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3$
  return (x_3, y_3)

Remark 2.4. *For an elliptic curve $E/\mathbf{K}$ over a field $\mathbf{K}$ and points $P, Q \in E(\overline{\mathbf{K}})$, we will refer to the act of computing $P + P$ as point doubling and computing $P + Q$ as point addition.*

Example 2.5. *Consider the elliptic curve $E/\mathbf{F}_{1447} : y^2 = x^3 + 748x + 255$ and points $P = (217, 981)$ and $Q = (1335, 405)$. It is easily verified that $P, Q \in E(\mathbf{F}_{1447})$ and easily seen that $P \neq Q$ and $217 \neq 1335$, so by* Algorithm 1 *we begin by computing*

$$\lambda = (405 - 981)(1335 - 217)^{-1} = 871 \cdot 1118^{-1} = 1330.$$

From here we are able to directly compute the coordinates of $P + Q$ as

$$x_3 = 1330^2 - 217 - 1335 + 0 \cdot 1330 - 0 = 561$$
$$y_3 = 1330(217 - 561) - 981 - 0 \cdot 561 - 0 = 198$$

and so $P + Q = (561, 198)$.

Lemma 2.3. *Given an elliptic curve* $E/\mathbf{K}$ *over a field* $\mathbf{K}$, $(E(\overline{\mathbf{K}}), +, \mathcal{O})$ *is an abelian group.*

Proof. Let $E/\mathbf{K}$ be an elliptic curve over a field $\mathbf{K}$ given by

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

Commutativity of elliptic curve addition in $E(\overline{\mathbf{K}})$ is immediately clear from its geometric interpretation and associativity can be proven both geometrically [11, chapter 2] and algebraically [12], but such a proof is rather exhaustive and so is left unwritten. Note then that $\mathcal{O}$ is the additive identity in $E(\overline{\mathbf{K}})$ which is clear from both the geometric interpretation as well as Algorithm 1. Finally, we see that $\mathcal{O}$ is its own additive inverse by Algorithm 1 and that the additive inverse $-P$ of an affine point $P \in E(\overline{\mathbf{K}})$ is given by the third intersection of the line L_2 through P and $\mathcal{O}$ which resides in $E(\overline{\mathbf{K}})$ by Bézout's theorem. □

Remark 2.5. *Given an elliptic curve* $E/\mathbf{K}$ *over a field* $\mathbf{K}$,

$$E(\mathbf{K}) \subset E(\overline{\mathbf{K}})$$

is a subgroup. This can be seen directly from the elliptic curve addition formulae in which all operations are made over $\mathbf{K}$ *and that, by Vieta's formulae, if a polynomial of degree* 3 *has two* $\mathbf{K}$*-rational roots then its third root is also* $\mathbf{K}$*-rational.*

3 Elliptic curves over finite fields

One might question why in cryptography we only consider elliptic curves over $\mathbf{F}_q$ as opposed to other extensively studied fields. A disadvantage of elliptic curves over fields such as $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{C}$ is precision. It is well known that representing real or complex numbers can result in problems with floating point precision and memory limitations, causing rounding errors, which is naturally inappropriate for cryptography. The latter renders elliptic curve addition for elliptic curves over such fields impractical. After only a few additions the number of bits required to store the numerators and denominators of the coordinates of such points is far too high. In contrast, working over $\mathbf{F}_q$ is 'clean' in the sense that the number of bits required to store points is known precisely and properties of the underlying group itself, such as the order of elements and the group order, are far easier to analyse. Not only that, but arithmetic in $\mathbf{F}_q$ is far faster than arithmetic in $\mathbf{Q}$, $\mathbf{R}$ or $\mathbf{C}$.

As for which finite fields are appropriate for cryptographic purpose, historically, a mixture of binary fields $\mathbf{F}_{2^m}$ and prime fields $\mathbf{F}_p$ have been used. Recently though, binary fields have fallen out of favour after progress was made in solving *the discrete logarithm problem* on elliptic curves over such fields. As such, for simplicity, we will only consider elliptic curves $E/\mathbf{F}_p$ over fields of prime order.

3.1 Integer multiplication

Before discussing properties pertaining to the structure of $E(\mathbf{F}_p)$ for an elliptic curve $E/\mathbf{F}_p$, we briefly discuss efficient methods of computing large integer multiples of points in $E(\mathbf{F}_p)$.

Definition 3.1. *If $E/\mathbf{F}_p$ is an elliptic curve and $P \in E(\mathbf{F}_p)$ then, for a non-negative integer n, the integer multiple $[n]P$ of P is the sum of P and itself n-many times, i.e.*

$$[n]P = \underbrace{P + \cdots + P}_{n\text{-many times}}$$

and the empty sum is simply $[0]P = \mathcal{O}$.

To compute such a sum, one could consider manually taking the sum of P and itself n-many times using Algorithm 1. If n is relatively large then such an approach is naive, as the computational costs grow to unfeasible amounts. Fortunately, there are far faster methods for integer multiplication, such as the binary method in which we write n using its binary representation as

$$n = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^r n_r$$

with $n_i \in \{0, 1\}$ for $i = 0, \ldots, r$ and compute $[n]P$ as

$$[n]P = [n_0]P + [2n_1]P + [2^2 n_2]P + \cdots + [2^r n_r]P.$$

Example 3.1. *Consider an elliptic curve $E/\mathbf{F}_p$ and let $P \in E(\mathbf{F}_p)$. Suppose that we would like to compute $[62]P$. The binary representation of 62 is $(1, 1, 1, 1, 1, 0)_2$ and so computing*

$$[62]P = [2^5]P + [2^4]P + [2^3]P + [2^2]P + [2]P$$

using the binary method requires only 5 point doublings and 4 point additions, as opposed to 61 point additions had we computed the sum naively.

In practice, the binary method requires $\lceil \log_2(n) \rceil = r$ point doublings and on average $\frac{1}{2}r$ point additions. Adaptations to the binary method include the (sliding-) window method [13] and the Montgomery ladder [14], the latter of which is an example of a constant-time algorithm whose purpose is to mask the number of point doublings and point additions made in computing $[n]P$, as to reduce vulnerability to timing or power consumption side-channel attacks [15] which we discuss in Section 6.2. We can do better than this, allowing for point subtractions, using the ternary representation of n,

$$n = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^r n_r$$

for $n_i \in \{-1, 0, 1\}$ for $i = 0, \ldots r$. It turns out that on average around two thirds of the coefficients in the ternary representation of an integer are 0, and so integer multiplication

using the ternary representation of n uses r or $r+1$ point doublings and on average $\approx \frac{1}{3}r$ point additions. As such, the ternary method requires the same number or one more point doubling as the binary method but on average $\approx \frac{1}{6}r$ less point additions.

Example 3.2. *Consider an elliptic curve $E/\mathbf{F}_p$ and let $P \in E(\mathbf{F}_p)$. To compute $[62]P$ using the ternary method, note that the ternary representation of 62 is $(1,0,0,0,0,-1,0)_t$ and so computing*

$$[62]P = [2^6]P - [2]P$$

using the ternary method requires 6 point doublings and 1 point addition, which in general is less computationally expensive than 5 point doublings and 4 point additions as in the binary method.

3.2 Hasse's theorem

An important result regarding the number of points on an elliptic curve $E/\mathbf{F}_p$ is Hasse's theorem on elliptic curves over finite fields.

Theorem 3.1 (Hasse)**.** *For an elliptic curve $E/\mathbf{F}_p$ we have the following inequality*

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

where $\#E(\mathbf{F}_p)$ denotes the number of elements in $E(\mathbf{F}_p)$.

Proof. One can prove this by considering the Frobenius endomorphism ϕ, mapping $E(\overline{\mathbf{F}_p})$ to itself given by

$$\phi : (x, y) \mapsto (x^p, y^p),$$

and noting that if $P \in E(\overline{\mathbf{F}_p})$ then $P \in E(\mathbf{F}_p)$ if and only if $\phi(P) = P$. From here it is clear that $E(\mathbf{F}_p) = \ker(1-\phi)$ and so

$$\#E(\mathbf{F}_p) = \#\ker(1-\phi) = \deg(1-\phi).$$

An appropriate use of a Cauchy-Schwarz inequality yields the desired result. □

Methods of obtaining the number of points $\#E(\mathbf{F}_q)$ on an elliptic curve $E/\mathbf{F}_p$ were relatively slow in the early stages of the development of the theory of elliptic curves, until 1985 when René Schoof developed what is now known as Schoof's algorithm. Until then, methods of counting the number of points on an elliptic curve were probabilistic and consisted of using the baby-step giant-step algorithm, detailed later on. Schoof's algorithm was the first deterministic polynomial time algorithm for counting the number of points on an elliptic curve, making use of the Frobenius endomorphism [9, p 98], and is detailed in [16].

3.3 The Frobenius trace

Definition 3.2. *For an elliptic curve $E/\mathbf{F}_p$ we define the Frobenius trace t as*

$$t = \#E(\mathbf{F}_p) - (p+1).$$

Note that by Hasse's theorem we have $|t| \leq 2\sqrt{p}$. We will see later on that an elliptic curve $E/\mathbf{F}_p$ should be chosen in a way such that its Frobenius trace t is not a value that would render particular problems posed on $E(\mathbf{F}_p)$ vulnerable to certain attacks. When discussing the elliptic curve discrete logarithm problem in Section 5, we will see that elliptic curves with Frobenius trace 0 and 1 are particularly weak, and so are not considered suitable for cryptographic purpose.

3.4 Isomorphism relations

An important result pertaining to the underlying structure of an elliptic curve group is the following.

Lemma 3.1. *Given an elliptic curve $E/\mathbf{F}_p$ we have that $E(\mathbf{F}_p)$ is cyclic or isomorphic to the direct product of two cyclic groups.*

Proof. Since $E(\mathbf{F}_p)$ is a finite abelian group we may write

$$E(\mathbf{F}_p) \cong \mathbf{Z}_{d_1} \times \cdots \times \mathbf{Z}_{d_r}$$

for some integer $r \geq 1$ such that $d_i | d_{i+1}$ for $i = 1, \ldots, r$. Given some $i \in \{1, \ldots, r\}$ we observe that $\mathbf{Z}_{n_i}$ has n_1 elements whose order divides n_1 and so $E(\mathbf{F}_p)$ must have n_1^r elements dividing n_1. Since $[n_i]P = \mathcal{O}$ has at most n_i^2 solutions in $E(\mathbf{F}_p)$ we conclude that $r \leq 2$ which yields the desired result. □

We will see later on in Sections 4 and 5 that the property of an elliptic curve $E/\mathbf{F}_p$ being cyclic is handy in defining and implementing Diffie-Hellman key exchange and the discrete logarithm problem.

4 Diffie-Hellman key exchange

The basis of modern public-key cryptography is the Diffie-Hellman key exchange protocol, developed in the 1970s allowing distinct parties to securely communicate over a public channel. Before Diffie-Hellman key exchange, methods of communicating securely over public channels required parties to first communicate a secret key in person. As such, methods of communicating secret keys can be expensive in a sense, involving constraints far harder to satisfy than the constraints of the Diffie-Hellman key exchange protocol, which we now detail.

4.1 The Diffie-Hellman key exchange protocol

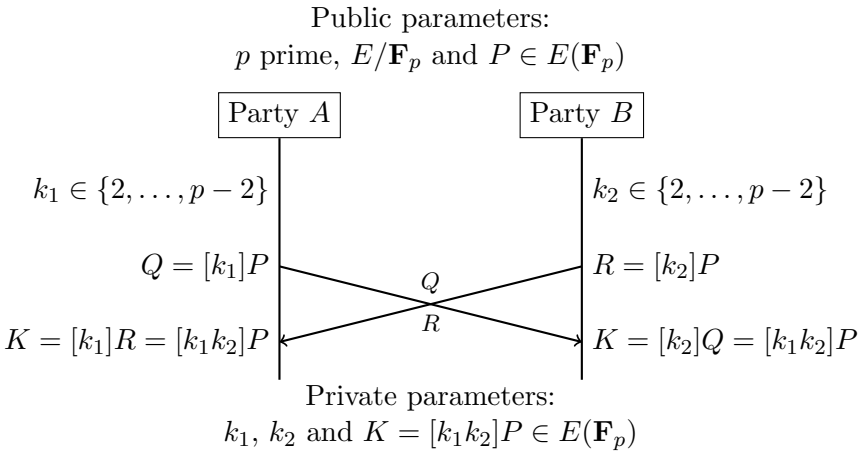
Consider distinct parties A and B that would like to securely communicate a secret key over a public channel. The Diffie-Hellman key exchange protocol between A and B is described as follows.

- A chooses a positive integer k_1, a finite multiplicative cyclic group G and generator $g \in G$. A computes the first public key $p_A = g^{k_1}$ and sends the group structure G, the generator g and p_A to B.
- B chooses a positive integer k_2, computes the second public key $p_B = g^{k_2}$ and sends p_B to A. B computes the shared secret key $s = p_A^{k_2} = g^{k_1 k_2}$.
- A computes the shared secret key $s = p_B^{k_1} = g^{k_1 k_2}$.

In practice, both A and B may agree beforehand to hash the secret key in some way, for an added layer of security. Both parties may then communicate securely over any public channel using the hashed secret key.

Remark 4.1. *For the remainder of the paper we refer to the Diffie-Hellman key exchange protocol as Diffie-Hellman.*

Upon seeing Diffie-Hellman for the first time, one may propose the idea of simply taking the 'logarithm' of g^{k_1} or g^{k_2}, however such a process may be performed in G, to obtain at least one of the secretly kept integers k_1 or k_2. If an adversary E were able to do this then E could easily obtain the shared secret key by direct computation. As such, the security of Diffie-Hellman is solely dependent on the difficulty of computing such *discrete logarithms* in G.



Given an elliptic curve $E/\mathbf{F}_p$ we can perform Diffie-Hellman in $E(\mathbf{F}_p)$ as written in Section 4.1 with generator $P \in E(\mathbf{F}_p)$. In this case, the shared secret key is some (ideally large) integer multiple $[k_1k_2]P$. The suitability of Diffie-Hellman in $E(\mathbf{F}_p)$ is due to the difficulty of solving the discrete logarithm problem in $E(\mathbf{F}_p)$, which we discuss in Section 5.

4.2 Brute force attacks and key-space size

As the name suggests, the method of brute force breaks a cryptographic algorithm by attempting every possible key in the corresponding key-space. This method is easily avoided by ensuring that the key-space size is suitably large with respect to the rate at which an attacker could perform such an attack. Though such a method seems to lack any ingenuity or even thought, it is still worth taking into account. Given current available hardware, a key-space size of at least 2^{128} is considered sufficiently large. Even so, some are skeptical that the development of quantum computers may fuel the need for larger key-space sizes of public-key cryptographic algorithms. Algorithms such as Grover's algorithm [17] could reduce key-size spaces from 2^{128} to 2^{64}; a significant reduction which would yield a system far more vulnerable to brute force attacks.

To understand the level of security that a cryptographic algorithm offers we consider the associated key-space size. The key-space size is the least upper-bound on a cryptographic algorithm's level of security, measuring the logarithm of the complexity of the best known algorithm that 'breaks' the cryptographic algorithm. To better define this we introduce *n-bit security*.

Definition 4.1. *A cryptographic algorithm C is n-bit secure if the best known algorithm to break C has complexity $O(2^n)$.*

Roughly put, we say that an algorithm *breaks* a cryptographic algorithm C if it exploits C in a way such that its running time is less than brute force. Many cryptosystems are designed in such a way that, on release, they attain some level of security, say 256 bits of security. It may be the case that later on an algorithm designed to break such a cryptographic algorithm is developed, with average running time far less than 2^{256} steps, say around 2^{128} steps. In this case, we say that the cryptographic algorithm in question offers 128 bits of security.

4.3 Small subgroup attacks

Typically, the order of an elliptic curve group on which we would like to perform elliptic curve Diffie-Hellman is the product of a small cofactor h and a large prime r: i.e. $\#E(\mathbf{F}_p) = h \cdot r$. Suppose E is an adversary that listens in on an exchange of secret keys between A and B and that B has no process of verifying received points, accepting any offered. If E sends a point of small order P to B then, oblivious to the attack, B computes

and sends $[k_2]P$ to E, as in the exchange. Then, E may simply compute the small number of possible integer multiples of P and see which matches $[k_2]P$, yielding $k_2 \pmod{\operatorname{ord} P}$. With the structure of the order of $E(\mathbf{F}_p)$, $\#E(\mathbf{F}_p) = h \cdot r$, as long as B chooses k_2 uniformly randomly then this attack reduces the number of possibilities of k_2 to r. In this case, E's best option is to resort to Pollard's rho algorithm.

Example 4.1. *Suppose the elliptic curve in question is* $E/\mathbf{F}_p : y^2 = x^3 - 3x + 19$ *with* $p = 1447$ *and that* B *chooses the secretly kept integer* $k_2 = 703$*.* E *may send the points* $P_2 = (313, 0)$ *and* $P_3 = (218, 1337)$*, of orders* 2 *and* 3 *respectively, to* B*. In response to this, oblivious to any attempt of an attack,* B *computes and sends* $Q_2 = [703]P_2 = (313, 0)$ *and* $Q_3 = [703]P_3 = (218, 1337)$ *to* E*. From this,* E *notices that* $Q_2 = [1]P_2$ *and* $Q_3 = [1]P_3$ *which yields* $k_2 \equiv 1 \pmod 2$ *and* $k_3 \equiv 1 \pmod 3$*, and so* $k_2 \equiv 1 \pmod 6$ *by the Chinese remainder theorem. Knowing that* $k_2 = 1 + 6j$ *for some positive integer* j*,* E *may continue with a brute force attack computing the integer multiples* $[1+6k]P$ *until the desired collision is met. In this particular example, the computational cost of a brute force attack is reduced from* 703 *integer multiples of* P *to* $\lfloor 703/6 \rfloor = 117$*.*

In implementation, B can avoid this by simply ensuring that $[h]P \neq \mathcal{O}$ for any proposed base point $P \in E(\mathbf{F}_p)$.

4.4 Invalid-curve attacks

A more elaborate attack, which also makes use of the Chinese remainder theorem, is the invalid-curve attack. In the invalid-curve attack, an adversary E utilises the fact that addition on elliptic curves given by short Weierstrass questions is independent of the coefficient b where $y^2 = x^3 + ax + b$ is the equation by which the elliptic curve in question is given.

Suppose B is performing elliptic curve Diffie-Hellman on the elliptic curve $E/\mathbf{F}_p$ given by $y^2 = x^3 + ax + b$. E may send a point $P = (x, y)$ on a distinct curve $y^2 = x^3 + ax + c$ from which B computes and sends $[k_2]P$ to E, completely oblivious to the attack. With this idea in mind, E may choose a series of points P_2, P_3, P_5, …, P_r of small prime orders 2, 3, 5, …, r respectively. Upon receiving the integer multiples $Q_2 = [k_2]P_2$, $Q_3 = [k_2]P_3$, $Q_5 = [k_2]P_5$, …, $Q_r = [k_2]P_r$, E is easily able to find $k_2 \pmod 2$, $k_2 \pmod 3$, $k_2 \pmod 5$, …, $k_2 \pmod r$ from which k_2 may be partially, or even fully, constructed using the Chinese remainder theorem.

This is a clear improvement on the attack detailed in Section 4.3, as we can choose various 'fake' curves $E_{a,c}/\mathbf{F}_p : y^2 = x^3 + ax + c$ such that $E_{a,c}(\mathbf{F}_p)$ contains points of small prime order, which are not guaranteed to exist in $E_{a,b}(\mathbf{F}_p)$.

Example 4.2. *Suppose, as in* Example 4.1*, that parties* A *and* B *are performing* elliptic curve Diffie-Hellman *on the Weierstrass curve* $E_{-3,19}/\mathbf{F}_p : y^2 = x^3 - 3x + 19$ *with* $p = 1447$*. As in* Example 4.1*, an adversary* E *may send the points* $P_2 = (313, 0)$ *and* $P_3 = (218, 1337)$

and analyse $Q_2 = [k_2]P_2$ *and* $Q_3 = [k_2]P_3$ *from* B *to see that* $k_2 \equiv 1 \pmod 2$ *and* $k_2 \equiv \pmod 3$. *The adversary can then send the points* $P_5 = (1143, 1310)$ *and* $P_7 = (606, 871)$ *on the 'fake' curves* $E_{-3,5}/\mathbf{F}_p : y^2 = x^3 - 3x + 5$ *and* $E_{-3,27}/\mathbf{F}_p : y^2 = x^3 - 3x + 27$ *respectively, the first with order* 5 *and the second with order* 7. *Completely oblivious to the fact that these are not points on the originally proposed curve,* B *responds by sending* E *the points* $Q_5 = (1429, 66) \in E_{-3,5}(\mathbf{F}_p)$ *and* $Q_7 = (1380, 1426) \in E_{-3,27}(\mathbf{F}_p)$. E *is then easily able to compute and see that* $Q_5 = [3]P_5$ *and* $Q_7 = [3]P_7$ *which yields* $k_2 \equiv 3 \pmod 5$ *and* $k_2 \equiv 3 \pmod 7$. *All together, with the help of the Chinese remainder theorem,* E *knows that* $k_2 \equiv 73 \pmod{210}$ *and so* $k_2 = 73 + 210k$ *for some positive integer* k. *All together, this process has reduced the cost of a brute force attack to the computational cost of computing* 3 *integer multiples of* P.

In practice, this is easily avoided by verifying that any point that B receives is a point on the chosen curve. In performing elliptic curve Diffie-Hellman, A and B may agree to exchange points $P = (x, y) \in E(\mathbf{F}_p)$ in the form $(x, \pm 1)$ where $(x, 1)$ corresponds to the point $(x, y_1) \in E(\mathbf{F}_p)$ where $y_1 \in \{\frac{p+1}{2}, \ldots, p-1\}$ and $(x, -1)$ corresponds to the point $(x, y_2) \in E(\mathbf{F}_p)$ where $y_2 \in \{0, \ldots, \frac{p-1}{2}\}$.

5 Discrete logarithms

The use of Diffie-Hellman key exchange over a group G makes it clear that the discrete logarithm problem in G should be sufficiently difficult to solve. To understand the discrete logarithm problem, its definition in a general group is given.

Definition 5.1. *Let* G *be a cyclic multiplicative group and let* $g \in G$. *The problem of finding the smallest positive integer* k *such that*

$$h = g^k \tag{7}$$

for some $h \in \langle g \rangle$ *is the discrete logarithm problem in* $\langle g \rangle \subset G$.

Example 5.1. *The group* $\mathbf{F}_p^\times$ *is cyclic and a generator can be easily found, for example by some brute force search. Consider the discrete logarithm problem in* $\mathbf{F}_{257}^\times$ *given by*

$$2 \equiv 3^k \pmod{257}.$$

It is easily verified that $3^{128} \equiv -1 \pmod{257}$ *and so* 3 *is a generator of* $\mathbf{F}_{257}^\times$. *To understand the 'random look' of exponents modulo a prime consider the exponents of* 3 *modulo* 257 :

k	1	2	3	4	...	9	10	11	12
$3^k \pmod{257}$	3	9	27	81	...	151	196	74	222

With such a small modulus methods of brute force are not unreasonable, and can be used to see that the smallest integer solution is $k = 48$.

Solving the discrete logarithm problem in $\mathbf{F}_q^\times$ through brute force has $O(q)$ running time, which is far too computationally expensive when q is sufficiently large. Naturally, methods far faster than brute force for solving the discrete logarithm problem in extensively studied groups, such as $\mathbf{F}_q^\times$, have been developed, an example of which is the index calculus algorithm.

5.1 The index calculus algorithm

The index calculus algorithm, which we abbreviate by index calculus, is an algorithm with sub-exponential running time that solves the discrete logarithm problem in $\mathbf{F}_q^\times$ whose complexity is $L_q[1/3, (64/9)^{1/3}]$ and is described as follows.

Consider the discrete logarithm function L such that for the discrete logarithm problem (7) we have $L(h) \equiv k \pmod{p-1}$. The discrete logarithm function L acts similarly to the traditional logarithm function in that it 'turns multiplication into addition', i.e. if we have $h_1 \equiv g^{k_1} \pmod{p}$ and $h_2 \equiv g^{k_2} \pmod{p}$ then

$$g^{L(h_1 h_2)} \equiv h_1 h_2 \equiv g^{L(h_1)} g^{L(h_2)} \equiv g^{L(h_1)+L(h_2)} \pmod{p}$$

and so

$$L(h_1 h_2) \equiv L(h_1) + L(h_2) \pmod{p-1}.$$

Index calculus makes use of this property by first choosing a 'factor base'

$$B = \{-1, p_1, \ldots, p_r\}$$

and storing the set

$$S = \{L(-1), L(p_1), \ldots, L(p_r)\},$$

where p_i is the ith prime starting with $p_1 = 2$ and r is a positive integer to be chosen. From here, we choose various values of j until $h \cdot g^j \pmod{p}$ is the product of elements in B, i.e. we search for a positive integer j satisfying

$$h \cdot g^j \equiv \prod_{i=1}^{n} b_i \pmod{p} \tag{8}$$

such that $b_i \in B$ for $i = 1, \ldots, n$. Applying the discrete logarithm function L to both sides of (8) yields

$$L(h) + j \equiv L\left(\prod_{i=1}^{n} b_i\right) \equiv \sum_{i=1}^{n} L(b_i) \pmod{p-1}$$

and so the discrete logarithm of h is given by

$$L(h) \equiv \sum_{i=1}^{n} L(b_i) - j \pmod{p-1},$$

which can be quickly computed since $L(b_i) \in S$ for $i = 1, \ldots, n$.

Example 5.2. *Consider the discrete logarithm problem as in* Example 5.1*,*

$$2 \equiv 3^k \pmod{257},$$

in which $k \equiv L(2) \pmod{256}$*. First, we choose the factor base* $B = \{-1, 2, 3, 5\}$ *and note that* $\operatorname{ord}(3) = 256$ *and so* $L(-1) \equiv 3^{\frac{p-1}{2}} \equiv 128 \pmod{256}$ *and* $L(3) \equiv 1 \pmod{256}$*. From here, we compute exponents of* 3 *whose only prime factors are* 2 *and* 5*. We have*

$$3^{39} \equiv 2^5 \cdot 5 \pmod{256}$$

$$3^{23} \equiv -1 \cdot 2^2 \cdot 5 \pmod{256}$$

which yields

$$39 \equiv 5L(2) + L(5) \pmod{256}$$

$$152 \equiv 2L(2) + L(5) \pmod{256}.$$

Solving these linear equations yields $L(2) \equiv 48 \pmod{256}$ *and so* $k = 48$*, as desired. Note that in this example* $h \equiv 2 \pmod{257}$ *was small enough to not have to perform the final step of finding a positive integer* j *such that* $h \cdot g^j$ *is the product of elements of* B*, which is not often the case.*

The choice of a suitable factor base is key to the efficiency of index calculus. If the factor base is too small then it may be that finding a positive integer j as in (8) takes an unreasonable amount of time, while if the factor base is too large then there may be too many cases to cover before progressing through the various stages of the algorithm.

Index calculus and its abundance of adaptations make the discrete logarithm problem relatively straight forward in $\mathbf{F}_q^\times$. We will see later on that in certain cases, a relatively difficult-to-solve discrete logarithm problem in an elliptic curve group can be reduced to a discrete logarithm problem in $\mathbf{F}_q^\times$, which in general is easier to solve.

5.2 The baby-step giant-step algorithm and Pollard's rho algorithm

In a general group G, the discrete logarithm problem has, so far, no easy solution and it is clear that for $n = \#G$ sufficiently large, brute force is impractical. With this in mind, we consider the baby-step giant-step algorithm, which we abbreviate by baby-step giant-step; a *space-time trade-off* algorithm used to compute discrete logarithms in an arbitrary cyclic

multiplicative group G with n elements, whose running time is $O(\sqrt{n})$. Given such a group G and a generator $g \in G$, baby-step giant-step utilises the fact that if

$$h = g^k$$

with $h \in G$ for a positive integer k, then we may write k in base $m := \lceil \sqrt{n} \rceil$ as

$$k = im + j$$

at which point we see that

$$g^j = h \cdot (g^{-m})^i.$$

Note that g, h and g^{-m} are fixed and so need to be computed only once. With this in mind, we can compute g^j for various values of j and then compute $(g^{-m})^i$ for various values of i in hope that we find a collision.

We summarise the baby-step giant-step algorithm in Algorithm 2.

Algorithm 2: Discrete logarithm in an arbitrary finite cyclic subgroup $\langle g \rangle \subset G$.

Input: A finite cyclic group G, $g \in G$ and $h \in \langle g \rangle$ with $\operatorname{ord}(g) = n$.
Output: A positive integer k such that $h = g^k$.

```
1  m = ⌈n⌉
2  S = ∅
3  for j = 0, ..., m − 1 do
4      S ← S ∪ {(j, g^j)}
5  λ ← g^{−m} // this step invokes the extended
       Euclidean algorithm
6  for i = 0, ..., m − 1 do
7      for γ = 0, ..., m − 1 do
8          if (γ, h(g^{−m})^γ) ∈ S then
9              return im + γ
10     else
11         λ ← λg^{−m}
```

Example 5.3. *Consider the discrete logarithm problem as in* Example 5.1*,*

$$2 \equiv 3^k \pmod{257}$$

in the group $\mathbf{F}_{257}^{\times}$*, whose solution has already been demonstrated to be* $k = 48$*. Writing* k *in base* $m = \lceil \sqrt{257 - 1} \rceil = 16$ *yields* $k = 16i + j$*. From here,* $g^{-m} \equiv 32 \pmod{257}$ *and so we search for collisions of* 3^j *and* $2 \cdot 32^i$ *modulo* 257 *for various values of* i *and* j *in* $\{0, 1, \ldots, 15\}$*. In computing* 3^j *modulo* 257 *for* $j = 0, \ldots, 15$ *we find:*

j	0	1	2	3	...	12	13	14	15
$3^j \pmod{257}$	1	3	9	27	...	222	152	199	83

Computing $2 \cdot 32^i$ *modulo* 257 *for* $i = 0, \ldots, 15$ *until we reach a collision, we find:*

i	0	1	2	3
$2 \cdot 32^i \pmod{257}$	2	64	249	1

We see that a collision occurs at $i = 3$ and $j = 0$ and so $k = 16 \cdot 3 + 0 = 48$, as desired.

With $O(\sqrt{n})$ running time, baby-step giant-step might appear an attractive option when looking to solve the discrete logarithm problem in well known groups, such as $\mathbf{F}_q^\times$, but this can be easily avoided by taking q sufficiently large. If, for example, we take q around 2^{256} then baby-step giant-step solves the discrete logarithm problem in $\mathbf{F}_q^\times$ in around $\sqrt{2^{256}} = 2^{128}$ steps, which is unfeasible. This emphasises the need for algorithms specially fit for a group in question, such as index calculus for $\mathbf{F}_q^\times$. Not only that, the memory required to store the exponents of g can be monstrously large. If $\#G = n$ and each exponent of g requires k bits to store then storing every exponent of g would require $k \cdot \sqrt{n}$ bits. For example if $\#G = 2^{128}$ and each exponent required 64 bits of storage then storing every exponent of g would require $1.18 \cdot 10^{21}$ bits, over 130000 Petabytes.

To avoid this, while maintaining a running time of $O(\sqrt{n})$, one can use *Pollard's Rho algorithm*, which we abbreviate by Pollard's algorithm. The implementation of Pollard's algorithm is slightly more tricky than baby-step giant-step, though its running time is the same and avoids the need more enormous amounts of memory. As such, when assessing the security of the discrete logarithm problem in an arbitrary group, we consider Pollard's algorithm instead of baby-step giant-step. An accurate estimate for the complexity of Pollard's algorithm is $O(\sqrt{\frac{\pi}{4} \cdot \#E(\mathbf{F}_p)})$.

5.3 Elliptic curves with Frobenius trace 0 or 1

As for an elliptic curve $E/\mathbf{F}_p$, the discrete logarithm problem is to find the smallest positive integer k such that

$$Q = [k]P \tag{9}$$

with $P \in E(\mathbf{F}_p)$ and $Q \in \langle P \rangle \subset E(\mathbf{F}_p)$.

Examples of classes of elliptic curves known to be explicitly vulnerable to attack are supersingular curves, anomalous curves and certain elliptic curves over binary fields. An elliptic curve $E/\mathbf{F}_p$ is supersingular if the Frobenius trace of $E(\mathbf{F}_p)$ satisfies $t \equiv 0 \pmod{p}$, which is equivalent to $t = 0$ when $p > 5$. The vulnerability of such curves is due to algorithms such as the MOV attack [18]. In this case, the discrete logarithm problem in $E(\mathbf{F}_p)$ can be reduced to a discrete logarithm problem in $\mathbf{F}_{p^m}^\times$ for $m \in \{2, 3, 4, 6\}$. In the worst case scenario the problem is reduced to a discrete logarithm problem in $\mathbf{F}_{p^6}^\times$ which, in general, is easier to solve. If $p \geq 5$, as is often the case, the problem is reducible to a discrete logarithm problem in $\mathbf{F}_{p^2}^\times$, which is far easier to solve. A similar statement can be made for *anomalous curves*, elliptic curves $E/\mathbf{F}_p$ with Frobenius trace $t = 1$. There exist algorithms for the discrete logarithm problem in anomalous curve groups far faster than $O(\sqrt{n})$ and even the MOV attack for supersingular curves. As for elliptic curve $E/\mathbf{F}_{2^m}$ over binary fields, where m is not prime, major vulnerabilities in the discrete logarithm problem over such curves is due to Weil descent attacks [19].

With such weaknesses in mind, it seems only natural to avoid the use of supersingular and anomalous curves and elliptic curves over binary fields when choosing elliptic curves suitable for cryptographic purpose. With this in mind, it seems only natural to ask what must be taken into account when one would like to make the discrete logarithm problem in an elliptic curve group difficult; how does one choose the coefficients of the elliptic curve and what finite fields $\mathbf{F}_p$ are suitable?

5.4 Rigidly constructing efficient and secure elliptic curves

It should be clear that having an elliptic curve $E/\mathbf{F}_p$ be secure is not enough for implementation, as it could be that computing integer multiples of points on such a curve is rather slow. A similar statement can be made for elliptic curves on which computing integer multiples is efficient. Namely that the curve is not necessarily secure, with the discrete logarithm problem being potentially easy to solve on such a curve. As such, in practice we require that an elliptic curve $E/\mathbf{F}_p$ is both secure, in that the discrete logarithm problem in $E(\mathbf{F}_p)$ is difficult to solve, and efficient, in that computing integer multiples in $E(\mathbf{F}_p)$ can be performed sufficiently quickly and ideally even on 'primitive' hardware.

In practice, one could opt to use elliptic curves published in a standard by a trusted entity. For example, in 1999 NIST published a document discussing the curve parameters for a set of elliptic curves they deemed fit for cryptographic use. In this document NIST detailed the 'desirable' properties of their curves and their approach in determining the parameters of curves yielding these properties. Controversy arose when it was noticed that the seed(s) of some parameter(s) pertaining to their suggested elliptic curves was not left unjustified. This is an example of a set of elliptic curves that are not *fully rigid*: i.e. their generation process is not made completely clear for potential users. From that point, many considered this standard to be influenced in a way such that NIST (or the NSA who had a say in these choices) could exploit implicit weaknesses of the curves in question.

As such, the construction of efficient and secure elliptic curves would ideally be fully rigid: i.e. the seed of each parameter pertaining to the curve is explained in sufficient detail, as to ensure that the curve has not been influenced in such a way that it omits implicit weaknesses. As for how to generate efficient and secure curves, one considers a large list of criteria. In practice, the generation of a curve is rarely constructive. One example of where a parameter is chosen constructively is in the Weierstrass model in which a field operation may be saved in point doubling by taking $a = -3$. Thus, when generating an efficient elliptic curve in the Weierstrass model, one may choose $a = -3$ specifically and look to meet the overall criteria through their choice of b and p alone. Typically, the remainder of the search is done by brute force. For example running some while loop over various candidates for the remaining parameters until all criteria pertaining to efficiency and security is met. This is discussed and exemplified in Section 8.

6 Efficiently and securely computing integer multiples

Factors that contribute to the efficiency of arithmetic on an elliptic curve $E/\mathbf{F}_p$ are primarily the model in which addition is performed and the prime p corresponding to the underlying field $\mathbf{F}_p$. In this section we consider prime numbers that offer fast reduction in $\mathbf{F}_p$. We then consider some 'desirable' properties of algorithms or procedures that parties should consider when performing Diffie-Hellman, including point validation and offline-computation based addition.

6.1 Arithmetic-friendly primes

Certain prime numbers offer faster multiplication and reduction modulo p and hence faster curve arithmetic. Examples of such primes are (pseudo-) Mersenne primes and Montgomery-friendly primes, which we discuss in this subsection.

Mersenne primes

A Mersenne prime p is a prime number that can be written as

$$p = 2^\alpha - 1$$

for some positive integer α. The largest Mersenne prime currently known was discovered in December of 2018, and is given by $2^{82589933} - 1$, though larger Mersenne primes are continuously searched for. Mersenne primes offer faster modular reductions via Algorithm 3, which has been read and written directly from Algorithm 1 in [20].

Algorithm 3: Reduction modulo a Mersenne prime.

Input: An integer k and a Mersenne prime $p = 2^\alpha - 1$.
Output: An integer l such that $0 \leq l < p$ and $l \equiv k \pmod{p}$.

1 Write $k = k_1 2^\alpha + k_0$
2 $r \leftarrow k_0 + k_1$
3 $r' \leftarrow r + 1$
4 **if** $r' \geq 2^\alpha$ **then**
5 $\quad r \leftarrow r' \pmod{2^\alpha}$
6 **return** r

An example of an elliptic curve that employs a Mersenne prime is FourQ; a (twisted) Edwards curve $\xi_{-1,d}/\mathbf{F}_{p^2} : -x^2 + y^2 = 1 + dx^2y^2$ where d is a non-square in $\mathbf{F}_{p^2}$ with $p = 2^{127} - 1$. Developed by Microsoft Research, FourQ was constructed specifically for secure key exchange and verification methods such as elliptic curve Diffie-Hellman and elliptic curve digital signature algorithms.

Pseudo-Mersenne primes

A pseudo-Mersenne prime p is a prime number that can be written as

$$p = 2^\alpha - \gamma$$

where α and γ are positive integers, typically with α a multiple of 64 and $\gamma < 2^{\alpha/2}$. Pseudo-Mersenne primes offer faster modular reduction in $\mathbf{F}_p$ than Mersenne primes. We describe reduction modulo a pseudo-Mersenne prime in Algorithm 4, which has been read and written directly from Algorithm 2 in [20].

Algorithm 4: Reduction modulo a pseudo-Mersenne prime.

Input: An integer k and a pseudo-Mersenne prime $p = 2^\alpha - \gamma$.
Output: An integer l such that $0 \leq l < p$ and $l \equiv k \pmod{p}$.

Write $k = k_1 2^\alpha + k_0$
$b \leftarrow k \cdot \gamma + k_0$
Write $b = b_1 2 + b_0$
$r \leftarrow b_1 \cdot \gamma + b_0$
$r' \leftarrow r + \gamma$
if $r' \geq 2^\alpha$ **then**
    $r \leftarrow r' - 2^\alpha$
    $r' \leftarrow r + \gamma$
    if $r' \geq 2^\alpha$ **then**
        $r \leftarrow r' - 2^\alpha$
return r

A well known example of an elliptic curve that makes use of a pseudo-Mersenne prime is Daniel Bernstein's Curve25519; developed for elliptic curve Diffie-Hellman offering 128 bits of security and detailed in [21]. Curve25519 is the Montgomery curve $M_{486662,1}/\mathbf{F}_p$ with $p = 2^{255} - 19$, hence the name.

Montgomery-friendly primes

Faster field multiplication and reduction in $\mathbf{F}_p$ is given by Montgomery-friendly primes, prime numbers of the form

$$p = 2^\alpha(2^\beta - \gamma) - 1$$

for positive integers α, β and γ. Field multiplication and reduction modulo p in $\mathbf{F}_p$ for Montgomery-friendly primes p is more efficient than (pseudo-) Mersenne primes and is described in detail in [22]. An example of an elliptic curve that utilises a Montgomery-friendly prime p is Curve448; the (twisted) Edwards curve given by $\xi_{1,-39011}/\mathbf{F}_p$ where $p = 2^{224}(2^{224} - 1) - 1$.

6.2 Side-channel attacks

Any attack made on the basis of information attained through the implementation of an algorithm, for example on some piece of hardware, is what we will call a side-channel attack. The more popular implementations of side-channel attacks are timing and power consumption attacks. These methods can be described in a rather intuitive fashion in the context of computing integer multiples of points on elliptic curves. In practice, given an elliptic curve $E/\mathbf{F}_p$, a point $P \in E(\mathbf{F}_p)$ and an integer k, to compute $[k]P$ we can use the binary or ternary method in which we compute a series of point doublings and point additions in accordance with the corresponding representation of k. If the hardware on which these integer multiples are computed perform point doublings and point additions

in distinct amounts of time (or consume distinct amounts of power) then one can easily reconstruct k, by counting the number of times the hardware performs a point doubling or point addition, and in which order.

Naturally, adaptations of currently used methods of computing integer multiples have been adapted with side-channel attacks in mind. As mentioned earlier, the Montgomery ladder is an algorithm for computing integer multiples in which each full computation is executed in a fixed number of steps, thus rendering timing or power consumption attacks useless. The Montgomery ladder is described in Algorithm 5.

Algorithm 5: The Montgomery ladder on an elliptic curve.

Input: An elliptic curve $E/\mathbf{F}_p$ with base point $\mathcal{O}$, a positive integer $n = (n_r, \ldots, n_1, n_0)_2$ with $n_i \in \{0, 1\}$ for $i = 1, \ldots, r$ and $P \in E(\mathbf{F}_p)$.

Output: $[n]P \in E(\overline{\mathbf{F}_p})$.

```
1  Q ← O
2  R ← P
3  for i = r, ..., 0 do
4      if n_i = 0 then
5          R ← Q + R // this invokes a point addition algorithm in the given model
6          Q ← [2]Q // this invokes a point doubling algorithm in the given model
7      else
8          Q ← Q + R // this invokes a point addition algorithm in the given model
9          R ← [2]R // this invokes a point doubling algorithm in the given model
10 return R
```

For a more detailed description of side-channel attacks and masking techniques, consider [15].

6.3 Offline-computation based addition

In practice, one can pre-compute a series of integer multiples of a point P offline in order to speed up the computation of integer multiples later on. An example of this is wNAF addition. Before describing an algorithm for wNAF addition explicitly, we consider the non-adjacent representation of an integer, beginning with its definition.

Definition 6.1. *Given an integer k, the non-adjacent form (NAF) of k is a ternary representation $(k_r, k_{r-1}, \ldots, k_1, k_0)_t$ of k such that no non-zero digits occur adjacently.*

Example 6.1. *We may write $13 = 8 + 4 + 1$ in ternary as $(1, 1, 0, 1)_t$, in which two consecutive non-zero values occur. As such $(1, 1, 0, 1)_t$ is not the non-adjacent normal form of 13. We can instead write $13 = 16 - 4 + 1$ as $(1, 0, -1, 0, 1)_t$ in which we see that no consecutive non-zero values appear. As such $(1, 0, -1, 0, 1)_t$ is the non-adjacent form of 13.*

To compute the non-adjacent form of an integer, consider Algorithm 6 with window size w, a positive integer to be chosen.

Algorithm 6: The non-adjacent form of a positive integer.

Input: A positive integer k.
Output: The ternary representation of k written as $(k_r, \ldots, k_1)_t$ such that if $k_j \neq 0$ then $k_{j-1} = k_{j+1} = 0$ for all $j = 2, \ldots, r-1$.

```
r ← 0
while k > 0 do
    if k ≡ 1 (mod 2) then
        k_r ← k mods 2^w
        k ← k − k_r
    else
        k_r ← 0
    k ← k/2
    i ← r + 1
return (k_{r−1}, ..., k_1, k_0)_t
```

As for wNAF addition, given a point $P \in E(\mathbf{F}_p)$ on an elliptic curve $E/\mathbf{F}_p$, one pre-computes the set $\{\pm P, \pm[3]P, \pm[5]P, \ldots, \pm[2^w - 1]P\}$ offline, for some positive integer w which we call the window size. After which one can quickly compute $[k]P$ by utilising the non-adjacent form $(k_r, \ldots, k_1)$ as follows.

Algorithm 7: wNAF integer multiplication on an elliptic curve $E/\mathbf{F}_p$.

Input: The non-adjacent form of a positive integer k and a point $P \in E(\mathbf{F}_p)$.
Output: $[k]P \in E(\overline{\mathbf{F}_p})$

```
Q ← O
for j = i, ..., 1 do
    Q ← 2Q
    if k_j ≠ 0 then
        Q = Q + k_j P
return Q
```

Algorithm 7 makes use of the signed modulo operator *mods* given by

$$k \text{ mods } 2^w = \begin{cases} (k \pmod{2^w}) - 2^w, \text{ if } k \pmod{2^w} \geq 2^{w-1} \\ k \pmod{2^w}, \text{ otherwise} \end{cases}$$

By pre-computing this set of points, one reduces the cost of computing integer multiples to r point doublings and around $\frac{r}{w+1}$ point additions, where w is the chosen window size, by using wNAF addition.

7 Efficient elliptic curve models and their security

In this section we consider some well known elliptic curve models in which curve arithmetic is known to be efficient. We then discuss criteria that elliptic curves in each model should satisfy to attain improved efficiency and security.

7.1 The Weierstrass model

Remark 7.1. *For the remainder of this paper it is assumed that the field over which we take an elliptic curve given by a Weierstrass equation is of characteristic not 2 or 3. As such, the only curves in the Weierstrass model that we consider are given by short Weierstrass equations.*

Definition 7.1. *A curve $E_{a,b}/\mathbf{F}_p$ in the Weierstrass model is an elliptic curve given by*

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

where $a, b \in \mathbf{F}_p$ such that $4a^3 + 27b^2 \neq 0$ with base point $\mathcal{O}_E = (0 : 1 : 0)$.

In Algorithm 1, methods of performing point addition and point doubling on elliptic curves over a field $\mathbf{K}$ given by a long Weierstrass equation were given. In both point addition and point doubling we saw that a field inversion in $\mathbf{K}$ is necessary. In fields such as $\mathbf{F}_p$, such an inversion is often more computationally expensive than a fair number of field multiplications. For elliptic curves $E_{a,b}/\mathbf{F}_p$ in the Weierstrass model, we can avoid this by computing point addition and point doubling using projective coordinates. Point addition in projective coordinates in the Weierstrass model is detailed in Algorithm 8.

Algorithm 8: Point addition in $(E_{a,b}(\overline{\mathbf{F}_p}), +, \mathcal{O}_E)$ in projective coordinates where $E_{a,b}/\mathbf{F}_p : Y^2Z = X^3 + aXZ^2 + bZ^3$.

Input: $P, Q \in E_{a,b}(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$.
Output: $P + Q \in E_{a,b}(\overline{\mathbf{F}_p})$.

$\lambda_1 \leftarrow Y_2Z_1$
$\lambda_2 \leftarrow Y_1Z_2$
$\lambda_3 \leftarrow X_2Z_1$
$\lambda_4 \leftarrow X_1Z_2$
if $\lambda_3 = \lambda_4$ **then**
  if $\lambda_1 \neq \lambda_2$ **then**
    return $\mathcal{O}_E$
  else
    return the output of Algorithm 9 with input $\{a, b, p, P\}$
$U \leftarrow \lambda_1 - \lambda_2$
$V \leftarrow \lambda_3 - \lambda_4$
$W \leftarrow Z_1Z_2$
$\alpha \leftarrow V^2$
$\beta \leftarrow \alpha V$
$\gamma \leftarrow \lambda_4\beta$
$\omega \leftarrow U^2W - \beta^3 - 2\gamma$
$X_3 \leftarrow \omega V$
$Y_3 \leftarrow U(\gamma - \omega) - \beta\lambda_2$
$Z_3 \leftarrow \beta W$
return $(X_3 : Y_3 : Z_3)$

We see that the computational cost of point addition in projective coordinates in the Weierstrass model is 12 field multiplications and 2 field squarings. For an implementation of Algorithm 8 in Python consider Listing 3. In Algorithm 9 we summarise point doubling

in projective coordinates in the Weierstrass model.

Algorithm 9: Point doubling on $(E_{a,b}(\overline{\mathbf{F}_p}), +, \mathcal{O}_E)$ in projective coordinates where $E_{a,b}/\mathbf{F}_p : Y^2Z = X^3 + aXZ^2 + bZ^3$.

Input: $P \in E_{a,b}(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$.
Output: $[2]P \in E_{a,b}(\overline{\mathbf{F}_p})$.

1 $\alpha \leftarrow aZ_1^2 + 3X_1^2$
2 $\beta \leftarrow Y_1Z_1$
3 $\gamma \leftarrow \beta X_1Y_1$
4 $\delta \leftarrow \alpha^2 - 8\gamma$
5 $\mu \leftarrow \beta^2$
6 $X_2 \leftarrow 2\beta\delta$
7 $Y_2 \leftarrow \alpha(4\gamma - \delta) - 8Y_1^2\mu$
8 $Z_2 \leftarrow 8\beta\mu$
9 **return** $(X_2 : Y_2 : Z_2)$

We see that the computational cost of point doubling in projective coordinates in the Weierstrass model is 7 field multiplications and 5 field squarings. In practice, one can take $a = -3$ so that in step 1 we have

$$\alpha = -3Z_1^2 + 3X_1^2 = 3(X_1 - Z_1)(X_1 + Z_1),$$

omitting the need for 2 field squarings in favour of a field multiplication, reducing the computational cost to 8 field multiplications and 3 field squarings. For an implementation of Algorithm 9 in Python consider Listing 3.

Remark 7.2. *Usually, when considering the computational cost of an algorithm for point addition or point doubling, the number of field additions (and subtractions) and field multiplications by a constant are left uncounted. This is because the computational cost of field multiplication and field squaring overshadows the cost of other field operations, yielding the computational cost of these other field operations negligible.*

Choosing suitable Weierstrass curves

To choose an efficient Weierstrass curve $E_{a,b}/\mathbf{F}_p$ we take $a = -3$, as to ensure faster curve arithmetic in projective coordinates. This imposes a slight restriction on how we choose b. To ensure a non-zero discriminant it is required that $b \in \mathbf{F}_p \backslash \{-2, 2\}$. Since the choice of b has no effect on the efficiency of point addition or point doubling, it seems only natural to choose it in accordance with satisfying the remaining criteria. The final factor for efficiency is the prime p, for which we choose one of the primes listed earlier; a (pseudo-) Mersenne prime of a Montgomery-friendly prime.

As far as security is concerned, we require the following:

- the Frobenius trace t of $E_{-3,b}(\mathbf{F}_p)$ satisfies $|t| > 1$.
- $\#E_{-3,b}(\mathbf{F}_p) = r$ is prime.
- $\#E'_{-3,b}(\mathbf{F}_p) = r'$ is prime, where E' is the quadratic twist of E [9, p 47].

The motivation for the first point is discussed in Section 5.3. Curves that fail to meet this property are immediately excluded due to their blatant weakness. The second condition helps to ensure that choosing a point $P \neq \mathcal{O}_E$ has large order, namely $\text{ord}(P) = r$ is prime, and that the $E_{-3,b}(\mathbf{F}_p)$ is not vulnerable to subgroup attacks. Similar statements can be made for the third condition, namely that the discrete logarithm problem should not be easily solved on the twist of the curve in question.

7.2 The Montgomery model

Definition 7.2. *As discussed in* Example 2.4*, a curve $M_{A,B}/\mathbf{F}_p$ in the Montgomery model is an elliptic curve given by*

$$By^2 = x^3 + Ax^2 + x$$

with $A, B \in \mathbf{F}_p$ such that $B(A^2 - 4) \neq 0$ and base point $\mathcal{O}_M = (0 : 1 : 0)$.

The motivation behind the base point $\mathcal{O}_M = (0 : 1 : 0)$ is precisely the motivation of the base point $\mathcal{O} = (0 : 1 : 0)$ on an elliptic curve given by a long Weierstrass equation.

Remark 7.3. *If B is a quadratic residue in $\mathbf{F}_p$ then a curve $M_{A,B}/\mathbf{F}_p$ in the Montgomery model given by $By^2 = x^3 + Ax^2 + x$ is birationally equivalent to the curve $M_{A,1}/\mathbf{F}_p$ in the Montgomery model given by $y^2 = x^3 + Ax^2 + x$.*

Lemma 7.1. *A Montgomery curve $M_{A,B}$ over a field $\mathbf{K}$ given by*

$$By^2 = x^3 + Ax^2 + x$$

is birationally equivalent to the curve $E_{a,b}/\mathbf{F}_p$ in the Weierstrass model given by

$$Y^2 = X^3 + aX + b$$

with $a = \frac{3-A^2}{3B^2}$ and $b = \frac{2A^3-9A}{27B^3}$.

Proof. The map

$$\psi : (x, y) \mapsto \left(\frac{x}{B} + \frac{A}{3B}, \frac{y}{B} \right) = (X, Y)$$

defines a birational map $M_{A,B} \to E_{a,b}$ with change of coefficients $a = \frac{3-A^2}{3B^2}$ and $b = \frac{2A^3-9A}{27B^3}$, where $E_{a,b}$ is given by

$$Y^2 = X^3 + aX + b.$$

□

The addition law on Montgomery curves is practically identical to the addition law on elliptic curves given by a long Weierstrass curve. As for explicit formulae for point addition and point doubling on Montgomery curves, the usual approach for deriving these

is precisely the approach used in deriving explicit formulae for addition on elliptic curves given by a long Weierstrass equation; looking for intersections of certain lines and the curve and applying Vieta's formulae. We omit this derivation and immediately move to detailing the formulae explicitly. Point arithmetic in affine coordinates in the Montgomery model is detailed in Algorithm 10.

Algorithm 10: Addition in $(M_{A,B}(\overline{\mathbf{F}_p}), +, \mathcal{O}_M)$ in affine coordinates where $M_{A,B}/\mathbf{F}_p : By^2 = x^3 + Ax^2 + x$.

Input: $P, Q \in M_{A,B}(\mathbf{F}_p)$ such that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
Output: $P + Q \in M_{A,B}(\overline{\mathbf{F}_p})$.

if $P = Q$ **then**
  if $P = \mathcal{O}_M$ *or* $y_1 = 0$ **then**
    return $\mathcal{O}_M$
  $\lambda \leftarrow (3x_1^2 + 2Ax_1 + 1)(2By_1)^{-1}$
  $x_3 \leftarrow B\lambda^2 - A - 2x_1$
  $y_3 \leftarrow \lambda(x_1 - x_3) - y_1$
  return (x_3, y_3)
else
  if $P = \mathcal{O}_M$ **then**
    return Q
  if $Q = \mathcal{O}_M$ **then**
    return P
  if $x_1 = x_2$ **then**
    return $\mathcal{O}_M$
  $\lambda \leftarrow (y_2 - y_1)(x_2 - x_1)^{-1}$
  $x_3 \leftarrow B\lambda^2 - A - x_1 - x_2$
  $y_3 \leftarrow \lambda(x_1 - x_3) - y_1$
  return (x_3, y_3)

Unlike other models, the interest of Montgomery curves is not the efficiency of full addition in projective coordinates or any other coordinate system but of computing the X and Z coordinates of multiples of points in projective coordinates, which can be done exceptionally quickly. For convenience let X_k and Z_k represent the X and Z coordinates of an integer multiple $[k]P = (X_k : - : Z_k)$ for a point $P \in M_{A,B}(\mathbf{F}_p)$. Then X_{m+n} and Z_{m+n}, corresponding to the X and Z coordinates of $[m+n]P$ in projective coordinates, are given by

$$X_{m+n} = Z_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2$$
$$Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2.$$

The obvious consequence of this is that to compute $[m+n]P = (X_{m+n} : - : Z_{m+n})$ we first need $[m-n]P = (X_{m-n} : - : Z_{m-n})$, which we can remedy during the computation of integer multiples by using the Montgomery ladder. It's worth noting that the use of the Montgomery ladder invokes the use of a point addition and a point doubling for each bit in the binary representation of k.

As for point doubling, X_{2n} and Z_{2n}, corresponding to $[2n]P = (X_{2n} : - : Z_{2n})$, are

computed via

$$\begin{aligned} 4X_nZ_n &= (X_n + Z_n)^2 - (X_n - Z_n)^2 \\ X_{2n} &= (X_n + Z_n)^2(X_n - Z_n)^2 \\ Z_{2n} &= 4X_nZ_n\left((X_n - Z_n)^2 + \left(\frac{A+2}{4}\right) \cdot 4X_nZ_n\right). \end{aligned}$$

It is easily verified that such a doubling costs only 2 field multiplications and 2 field squarings, which is far faster than the cost of point doubling in the models already considered. For an implementation of these formulae, as well as the Montgomery ladder and Y-coordinate recovery, in Python consider Listing 4.

Y-coordinate recovery

Given that the Montgomery ladder returns the X and Z coordinates of an integer multiple $[k]P = (X_k : - : Z_k)$, it is of interest to be able to 'recover' the corresponding Y coordinate, yielding the integer multiple $[k]P = (X_k : Y_k : Z_k)$ in full. To do this, firstly, if $Z_k = 0$ then $[k]P$ is simply the point at infinity $\mathcal{O}_M$. Otherwise, one may notice that given X_k and Z_k the corresponding Y-coordinate Y_k must satisfy

$$BY_k^2Z_k = X_k^3 + AX_k^2Z_k + X_kZ_k^2$$

and so one may recover Y_k by first multiplying by $(BZ_k)^{-1}$ and taking a modular square root. One could find such a modular square root with the help of the Tonelli-Shanks algorithm [23]. All together, the computational cost of this can be approximated to $m + n + 100$ field multiplications, where m is the number of bits in the binary representation of k and n is the number of 1s that appear in the representation. Note that this square root is guaranteed to exist as X_k and Z_k, and thus Y_k, correspond to a point on the curve. The only issue with this method is that taking the modular square root yields two potential points, corresponding to the two possibilities of the modular square root. Methods more considerate of a problem such as this are developed and well documented.

Remark 7.4. *Since one can always construct a birational map from a Montgomery curve to a (twisted) Edwards curve, and vice versa, we consider tailoring some Montgomery curve to be efficient and secure in accordance with tailoring a (twisted) Edwards curve. As such, we leave the discussion of choosing Montgomery curves to the end of* Subsection 7.3 *which considers curves in the (twisted) Edwards model.*

7.3 The (twisted) Edwards model

The (twisted) Edwards model was introduced by Daniel Bernstein in 2008 as an adaptation to the already-known Edwards model. Curves in the (twisted) Edwards model are known

to have efficient algorithms for point addition and point doubling.

Definition 7.3. *A curve $\xi_{a,d}/\mathbf{F}_p$ in the (twisted) Edwards model is an elliptic curve given by*

$$ax^2 + y^2 = 1 + dx^2y^2$$

where $a, d \in \mathbf{F}_p$ such that $a \neq d$ and $a \cdot d \neq 0$ with base point $\mathcal{O}_\xi = (0 : 1 : 0)$.

Lemma 7.2. *A (twisted) Edwards curve $\xi_{a,d}/\mathbf{K}$ over a field $\mathbf{K}$ is birationally equivalent to the Montgomery curve $M_{A,B}/\mathbf{K}$ where $A = 2\frac{a+d}{a-d}$ and $B = 4\frac{1}{a-d}$.*

Proof. This is shown in Example 2.3. □

As such, (twisted) Edwards curves are indeed elliptic curves. To obtain the addition law on (twisted) Edwards curves, we use Cauchy-Desboves' formulae [24]. Explicit formulae for arithmetic in affine coordinates in the (twisted) Edwards model is summarised in Algorithm 11.

Algorithm 11: Addition in $(\xi_{a,d}(\overline{\mathbf{F}_p}), +, \mathcal{O}_\xi)$ in affine coordinates where $\xi_{a,d}/\mathbf{F}_p : ax^2 + y^2 = 1 + dx^2y^2$.

Input: $P, Q \in \xi_{a,d}(\mathbf{F}_p)$ such that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
Output: $P + Q \in \xi_{a,d}(\overline{\mathbf{F}_p})$.

if $P = Q$ **then**
  if $P = \mathcal{O}_\xi$ *or* $x_1 = 0$ **then**
    return $\mathcal{O}_\xi$
  $x_3 \leftarrow (2x_1y_1)(ax_1^2 + y_1^2)^{-1}$
  $y_3 \leftarrow (y_1^2 - ax_1^2)(2 - ax_1^2 - y_1^2)^{-1}$
  return (x_3, y_3)
else
  if $P = \mathcal{O}_\xi$ **then**
    return Q
  if $Q = \mathcal{O}_\xi$ **then**
    return P
  if $y_1 = y_2$ **then**
    return $\mathcal{O}_\xi$
  $x_3 \leftarrow (x_1y_2 + x_2y_1)(1 + dx_1x_2y_1y_2)^{-1}$
  $y_3 \leftarrow (y_1y_2 - ax_1x_2)(1 - dx_1x_2y_1y_2)^{-1}$
  return (x_3, y_3)

Similarly to Algorithm 1, we see that point addition and point doubling in affine coordinates requires two modular inversions. To avoid this we can instead perform point addition and point doubling in projective coordinates.

In Algorithm 12 we detail point addition in projective coordinates in the (twisted) Edwards model.

Algorithm 12: Point addition in $(\xi_{a,d}(\overline{\mathbf{F}_p}), +, \mathcal{O}_\xi)$ in projective coordinates where $\xi_{a,d}/\mathbf{F}_p : aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$.

Input: $P, Q \in \xi_{a,d}(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$.
Output: $P + Q \in \xi_{a,d}(\overline{\mathbf{F}_p})$ in projective coordinates.

1 $A \leftarrow Z_1Z_2$
2 $B \leftarrow A^2$
3 $C \leftarrow X_1X_2$
4 $D \leftarrow Y_1Y_2$
5 $E \leftarrow dCD$
6 $F \leftarrow B - E$
7 $G \leftarrow B + E$
8 $X_3 \leftarrow AF((X_1 + Y_1)(X_2 + Y_2) - C - D)$
9 $Y_3 \leftarrow AG(D - aC)$
10 $Z_3 \leftarrow FG$
11 **return** $(X_3 : Y_3 : Z_3)$

We see that the computational cost of point addition in projective coordinates in the (twisted) Edwards model is 10 field multiplications and 1 field squaring. For an implementation of Algorithm 12 in Python consider Listing 5. Point doubling in projective coordinates in the (twisted) Edwards model is detailed in Algorithm 13.

Algorithm 13: Point doubling in $(\xi_{a,d}(\overline{\mathbf{F}_p}), +, \mathcal{O}_\xi)$ in projective coordinates where $\xi_{a,d}/\mathbf{F}_p : aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2$.

Input: $P \in \xi_{a,d}(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$.
Output: $[2]P \in \xi_{a,d}(\overline{\mathbf{F}_p})$ in projective coordinates.

1 $B \leftarrow (X_1 + Y_1)^2$
2 $C \leftarrow X_1^2$
3 $D \leftarrow Y_1^2$
4 $E \leftarrow aC$
5 $F \leftarrow E + D$
6 $H \leftarrow Z_1^2$
7 $J \leftarrow F - 2H$
8 $X_3 \leftarrow J(B - C - D)$
9 $Y_3 \leftarrow F(E - D)$
10 $Z_3 \leftarrow FJ$
11 **return** $(X_3 : Y_3 : Z_3)$

We see that the computational cost of point doubling in projective coordinates in the (twisted) Edwards model is 3 field multiplications and 4 field squarings. For an implementation of Algorithm 13 in Python consider Listing 5.

Choosing suitable (twisted) Edwards and Montgomery curves in parallel

The following is a list comprising the criteria that a pair of birationally equivalent Montgomery and (twisted) Edwards curves should satisfy for the sake of both efficiency and security.

- take $a = -1$.
- take $A \in 2 + 4\mathbf{Z}$.

- ensure that $\#M_{A,1} = 4r$ where r is prime.
- ensure that $\#M'_{A,1} = 4r'$ where r' is prime.

The first condition simply ensures that addition in projective coordinates in the (twisted) Edwards model is efficient. Requiring that $A \in 2 + 4\mathbf{Z}$ condition ensures that B is a quadratic residue in $\mathbf{F}_p$ and so addition on the $M_{A,B}$ can be reduced to addition on the Montgomery curve $M_{A,1}/\mathbf{F}_p : y^2 = x^3 + Ax^2 + x$. Since the order of a (twisted) Edwards curve group is always a multiple of 4, the best that can be done in ensuring that the elliptic curve group in question is almost-prime is to take our group such that the order is $4r$ for a large prime p. This condition helps to ensure security against small subgroup attacks. The final point is simply to ensure twist security [25].

7.4 The Hessian model

An example of a less well known model in which arithmetic is efficient 'by design' is the Hessian model.

Definition 7.4. *A curve $H_D/\mathbf{F}_p$, with $p \equiv 2 \pmod 3$, in the Hessian model is an elliptic curve given by*

$$X^3 + Y^3 + Z^3 = 3DXYZ$$

with $D \in \mathbf{F}_p$ such that $D^3 \neq 1$ and base point $\mathcal{O}_H = (1 : -1 : 0)$.

Though we have defined a Hessian curve as an elliptic curve, it can also be seen from Lemma 7.3, in which it is shown that a Hessian curve is birationally equivalent to an elliptic curve given by a long Weierstrass equation.

Lemma 7.3. *Consider the elliptic curve $E/\mathbf{F}_q$, with $q \equiv 2 \pmod 3$, given by*

$$Y^2 + a_1XY + a_3Y = X^3$$

with $a_1, a_3 \in \mathbf{F}_q$, in which an element $\epsilon \in E(\mathbf{F}_q)$ such that $\epsilon^3 = -27a_3\delta^2 - \delta^3$ is assumed to exist. $E/\mathbf{F}_q$ is birationally equivalent to the Hessian curve H_D given by

$$x^3 + y^3 + z^2 = 3Dxyz$$

with $D \in \mathbf{F}_q$ such that $D^3 \neq 1$.

Proof. The discriminant of $E/\mathbf{F}_q$ is given by

$$\Delta_E = a_3^3(a_1^3 - 27a_3) = a_3^3\delta,$$

where $\delta = a_1^3 - 27a_3$. Letting $\mu = \frac{1}{3}((-27a_3\delta^2 - \delta^3)^{1/3} + \delta)$, we have that

$$\psi : (X, Y) \mapsto \left(a_1\frac{2\mu - \delta}{3\mu - \delta}X + Y + a_3, -a_1\frac{\mu}{3\mu - \delta}X - Y, -a_1\frac{\mu}{3\mu - \delta}X - a_3\right) = (x, y, z)$$

defines a birational map $E \to H_D$, with change of coefficient $D = \frac{\mu - \delta}{\mu}$. $\square$

To understand the motivation behind the base point $\mathcal{O}_H = (1 : -1 : 0)$, substitute any point $(x : y : 0)$ on the line at infinity into the equation by which a Hessian curve is given. In doing this we obtain $x^3 + y^3 = 0$ and so $y = -x \neq 0$ which yields $\mathcal{O}_H = (1 : -1 : 0)$.

The addition law on Hessian curves can be described through the use of the chord-tangent law, as seen earlier, while utilising the symmetry about the line given by $y = x$. To begin, we derive the coordinates of the additive inverse $-P = (x_2, y_2)$ for an affine point $P = (x_1, y_1)$. The line L_1 intersecting $\mathcal{O}_H$ and $P = (x_1, y_1)$ in $H_D(\overline{\mathbf{F}_q})$ is given by $y = -x + (x_1 + y_1)$. Substituting the equation by which L_1 is given into the equation by which H_D is given yields

$$(3 + D)(x_1 + y_1)x^2 - (3 + D)(x_1 + y_1)^2 x + (x_1 + y_1)^3 + 1 = 0$$

whose roots are x_1 and x_2. By Vieta's formulae we have $x_2 = y_1$. Upon substituting $x_2 = y_1$ into the equation by which L_1 is given, we see that $y_2 = x_1$ and so

$$-P = (y_1, x_1),$$

i.e. that the additive inverse of a point P is simply its reflection across the line given by $y = x$. Applying the Cauchy-Desboves' formulae to the polynomial

$$F(x, y, z) = x^3 + y^3 + z^3 - 3Dxyz$$

of a Hessian curve yields the following formulae for addition on Hessian curves for distinct points $P = (x_1 : y_1 : z_1)$ and $Q = (x_2 : y_2 : z_2)$; $[2]P = (x_3 : y_3 : z_3)$ where

$$\begin{aligned} x_3 &= y_1(z_1^3 - x_1^3) \\ y_3 &= x_1(y_1^3 - z_1^3) \\ z_3 &= z_1(x_1^3 - y_1^3) \end{aligned}$$

and $P + Q = (x_4 : y_4 : z_4)$ where

$$\begin{aligned} x_4 &= y_1^2 x_2 z_2 - y_2^2 x_1 z_1 \\ y_4 &= x_1^2 y_2 z_2 - x_2^2 y_1 z_1 \\ z_4 &= z_1^2 x_2 y_2 - z_2^2 x_1 y_1. \end{aligned}$$

With explicit formulae for Hessian curve addition in place, points of order 2 and 3 on a Hessian curve H_D over $\mathbf{F}_q$ are easily found. Points $P = (x : y : z)$ of order 2 must satisfy $P = -P$ which holds if and only if

$$(x : y : z) = (y : x : z)$$

and so $P = (1 : 1 : \lambda)$ for $\lambda \in \mathbf{F}_q$. Points $(x : y : z)$ of order 3 must satisfy $[2]P = -P$ which holds if and only if

$$(y(z^3 - x^3) : x(y^3 - z^3) : z(x^3 - y^3)) = (y : x : z).$$

Finding solutions to this equation boils down to solving the equations $x^3 = 1$ and $y^3 = 1$. In $\mathbf{F}_q$ these equations have either one solution, when 3 divides $q-1$ or three solutions, when 3 is relatively prime to $q-1$. After some rearrangement, it is seen that for any $\mathbf{F}_q$ solutions in $H_D(\mathbf{F}_q)$ are $(0 : 1 : -1)$, $(1 : 0 : -1)$ and $(1 : -1 : 0)$. Note that when the finite field in question has three roots of unity, there exist more points of order 3. We know already that $(1 : -1 : 0)$ is the additive identity in $H_D(\mathbf{F}_q)$. As such, when $\gcd(3, q-1) = 1$, $(0 : 1 : -1)$ and $(1 : -1 : 0)$ are the only points of order 3 in $H_D(\mathbf{F}_q)$. From here we know that the number of points on a Hessian curve $\#H_D$ is a multiple of 6.

As for the efficiency of arithmetic on Hessian curve, consider Algorithm 14 which details point addition in projective coordinates in the Hessian model.

Algorithm 14: Point addition in $(H_D(\overline{\mathbf{F}_p}), +, \mathcal{O}_H)$ in projective coordinates where $H_D/\mathbf{F}_p : X^3 + Y^3 + Z^3 = 3DXYZ$.

Input: $P, Q \in H_D(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$.
Output: $P + Q \in H_D(\overline{\mathbf{F}_p})$.

1 $\lambda_1 \leftarrow Y_1X_2$
2 $\lambda_2 \leftarrow X_1Y_2$
3 $\lambda_3 \leftarrow X_1Y_2$
4 $\lambda_4 \leftarrow Z_1X_2$
5 **if** $\lambda_1 = \lambda_2$ **and** $\lambda_3 = \lambda_4$ **then**
6 | return the output of Algorithm 15
7 | with input $\{p, D, P\}$
8 $\lambda_5 \leftarrow Z_1Y_2$
9 $\lambda_6 \leftarrow Z_2Y_1$
10 $s_1 \leftarrow \lambda_1\lambda_6$
11 $s_2 \leftarrow \lambda_2\lambda_3$
12 $s_3 \leftarrow \lambda_4\lambda_5$
13 $t_1 \leftarrow \lambda_2\lambda_5$
14 $t_2 \leftarrow \lambda_1\lambda_4$
15 $t_3 \leftarrow \lambda_3\lambda_6$
16 $X_3 \leftarrow s_1 - t_1$
17 $Y_3 \leftarrow s_2 - t_2$
18 $Z_3 \leftarrow s_3 - t_3$
19 **return** $(X_3 : Y_3 : Z_3)$

We see that the computational cost of point addition in projective coordinates in the Hessian model is 12 field multiplications and 3 field squarings.

For point doubling in projective coordinates in the Hessian model consider Algorithm 15.

Algorithm 15: Point doubling on $(H_D(\overline{\mathbf{F}_p}), +, \mathcal{O}_H)$ in projective coordinates where $H_D/\mathbf{F}_p : X^3 + Y^3 + Z^3 = 3DXYZ$.

Input: $P \in H_D(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$.
Output: $[2]P \in H_D(\overline{\mathbf{F}_p})$ in projective coordinates.

1 $\lambda_1 \leftarrow X_1^2$
2 $\lambda_2 \leftarrow Y_1^2$
3 $\lambda_3 \leftarrow Z_1^2$
4 $\lambda_4 \leftarrow X_1\lambda_1$
5 $\lambda_5 \leftarrow Y_1\lambda_2$
6 $\lambda_6 \leftarrow Z_1\lambda_3$
7 $\lambda_7 \leftarrow \lambda_5 - \lambda_6$
8 $\lambda_8 \leftarrow \lambda_6 - \lambda_4$
9 $\lambda_9 \leftarrow \lambda_4 - \lambda_5$
10 $X_2 \leftarrow y_1\lambda_8$
11 $Y_2 \leftarrow x_1\lambda_7$
12 $Z_2 \leftarrow z_1\lambda_9$
13 **return** $(X_2 : Y_2 : Z_2)$

We see that the computational cost of point doubling in projective coordinates in the Hessian model is 6 field multiplications and 3 field squarings.

Efficient point tripling in $H_D(\mathbf{F}_q)$

An interesting property of the Hessian model is the efficiency of point tripling: i.e. taking a point $P \in H_D(\mathbf{F}_p)$ and computing $[3]P$ in fewer steps than first computing $[2]P$ and then $[2]P + P = [3]P$. This is achieved in projective coordinates by Algorithm 16, as detailed below.

Algorithm 16: Point tripling on $(H_D(\overline{\mathbf{F}_p}), +, \mathcal{O}_H)$ in projective coordinates where $H_D/\mathbf{F}_p : X^3 + Y^3 + Z^3 = 3DXYZ$.

Input: $P \in H_D(\mathbf{F}_p)$ such that $P = (X_1 : Y_1 : Z_1)$.
Output: $[3]P \in H_D(\overline{\mathbf{F}_p})$ in projective coordinates.

1 $\lambda_1 \leftarrow X_1^3$
2 $\lambda_2 \leftarrow Y_1^3$
3 $\lambda_3 \leftarrow Z_1^3$
4 $\mu_1 \leftarrow \lambda_3 - \lambda_1$
5 $\mu_2 \leftarrow \lambda_2 - \lambda_3$
6 $\mu_3 \leftarrow \lambda_1 - \lambda_2$
7 $X_3 \leftarrow 3D(\lambda_2\mu_1\mu_3 - \lambda_1\mu_2^2)$
8 $Y_3 \leftarrow 3D(\lambda_1\mu_2\mu_3 - \lambda_2\mu_1^2)$
9 $Z_3 \leftarrow (\lambda_1 + \lambda_2 + \lambda_3)(\mu_1\mu_2 - \mu_3^2)$
10 **return** $(X_3 : Y_3 : Z_3)$

We see that the computational cost of point tripling in projective coordinates in the Hessian model is 8 field multiplications and 6 field squarings. Thus, tripling requires only two more field operations than point addition on Hessian curves. This may be utilised by an adaptation of the ternary method, in which for a given integer k we compute $[k]P$ using a representation of k in base 3.

7.5 Mapping between models

Cost of addition in each model

When measuring the computational cost of an addition algorithm, it is also worth considering the cost of *mixed point addition* and *mixed point doubling* in which, for inputs $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$, at least one of Z_1 or Z_2 is 1. As one may expect, in practice, this typically reduces the number of operations needed. The table below contains the computational cost of (mixed) point addition and (mixed) point doubling in projective coordinates in each model considered.

Model - Coordinate system	Doubling	Addition	Mixed doubling	Mixed addition
Weierstrass - Projective	11	14	11	8
Weierstrass (with $A = -3$) - Projective	10	14	11	8
Hessian - Projective	8	12	10	6
Edwards - Projective	7	11	10	6
Montgomery - Projective	4	6*	3	6*

Improving efficiency with birational maps

Birational maps between models provide a way of mapping points between models while preserving the underlying group structure. This is a huge convenience but is not necessarily computationally cheap. Typically, even efficient implementations of birational maps between models will involve an operation as computationally expensive as a modular inversion. It also isn't necessarily the case that a birational map between given models is easily constructed. For example mapping Weierstrass to Montgomery is conditional: i.e. a small set of constraints must be satisfied for a birational map to be constructed.

Cost of converting coordinates between models (affine only)				
	Multiplications	Inversions	Square roots	Conditional?
Weierstrass - Montgomery	3	1	1	Yes
(twisted) Edwards - Montgomery	2	2	0	No
Montgomery - (twisted) Edwards	2	1	0	No
Montgomery - Weierstrass	2	1	0	No

Birational maps between models		
Models	Coordinate mapping	Coefficient mapping
$M_{A,B}$ (affine) to $\xi_{a,d}$ (affine)	$(x, y) \mapsto (\frac{x}{y}, \frac{x-1}{x+1})$	$(A, B) \mapsto (\frac{A+2}{B}, \frac{A-2}{B}) = (a, d)$
$\xi_{a,d}$ (affine) to $M_{A,B}$ (affine)	$(x, y) \mapsto (\frac{1+y}{1-y}, \frac{1}{x}\frac{1+y}{1-y})$	$(a, d) \mapsto (2\frac{a+d}{a-d}, 4\frac{1}{a-d}) = (A, B)$
$\xi_{a,d}$ (projective) to $M_{A,B}$ (affine)	$(- : Y : Z) \mapsto (\frac{Z+Y}{Z-Y}) = (x, -)$	$(a, d) \mapsto (2\frac{a+d}{a-d}, 4\frac{1}{a-d}) = (A, B)$

Given that a birational map is not always easily constructed, an attractive option may be to work solely in models that are easy to map to a more efficient model. Take for

example the Montgomery and (twisted) Edwards models; a birational map between both, in either direction, is easily constructed and costs a total of 1 field multiplication and 1 field inversion in each case. With this in mind, we are already able to consider faster methods of computing integer multiples in a given model with the help of birational maps.

For example, consider the problem of having to compute the integer multiple $[k]P_\xi$ for a large integer k and a point $P_\xi \in \xi_{a,d}$ on the (twisted) Edwards curve $\xi_{a,d}/\mathbf{F}_p$. Given that mapping to the corresponding Montgomery curve $M_{A,B}/\mathbf{F}_p$ and back is always a possibility, one may wonder when it is more efficient to first map P_ξ to its corresponding point $P_M = \varphi(P_\xi)$ using a birational map φ, compute $[k]P_M$ in the Montgomery model and then map $[k]P_M$ back to the (twisted) Edwards model via $[k]P_\xi = \varphi^{-1}([k]P_M)$. Though an interesting, we leave this as a question for future consideration.

8 Constructing efficient and secure elliptic curves

To finish off, we consider explicit examples of efficient and secure elliptic curves in the Weierstrass, Montgomery and (twisted) Edwards models. Before this, we discuss the methodology taken to explicitly construct efficient and secure elliptic curves, with respect to the criteria already given, at 32-bits, 64-bits and 96-bits of security.

8.1 Curves in the Weierstrass model

To construct suitable curves $E_{a,b}/\mathbf{F}_p : y^2 = x^3 + ax + b$ in the Weierstrass model, an approach similar to as in [22] is taken. For the sake of efficiency, we take $a = -3$ and a Montgomery-friendly prime $p = 2^\alpha(2^\beta - \gamma) - 1$ where α is a multiple of 8, $\beta = 2s - \alpha$, γ is some small positive integer and s is the desired level of security of the curve. With this, one can use a relatively short script in Magma, using a while loop to increment b by 1, starting with $b = 3$, until all criteria is met. To compute the group orders $\#E_{a,b}(\mathbf{F}_p)$ and $\#E'_{a,b}(\mathbf{F}_p)$, an efficient implementation of SEA in Magma is used. This is equipped with an early abort feature which aborts the current value of b if either the group order or the order of its twist is seen to have a small factor. The Magma script used is given below.

```
alpha := ?;
beta  := ?;
gamma := ?;
p := (2^alpha)*(2^beta-gamma)-1;

b := 3;
while b gt 0 do
    E := EllipticCurve([GF(p)|-3,b]);
    if SEA(E : MaxSmooth := 1) ne 0 then
        t := TraceOfFrobenius(E);
```

```
        if IsPrime(p+1-t) and IsPrime(p+1+t) and (t gt 1 or -t gt 1) then
         break;
        end if;
    end if;
    b +:= 1;
end while;

b;
```

Note that to find the 32-, 64- and 96- bit Montgomery-friendly primes used for this construction, Listing 2 in the Appendix was used. We see that once a curve satisfying all efficiency and security criteria is found, the incrimination stops and returns the corresponding value of b. For explicit examples pertaining 32, 64 and 96 bits of security, consider the following table.

bits of security	prime p	b	ρ complexity	Frobenius trace
32 bits	$2^{32}(2^{32}-49)-1$	2375	31.8	142D9309
	$2^{48}(2^{16}-31)-1$	4604	31.8	17ACE5819
	$2^{16}(2^{48}-47)-1$	6873	31.8	2EADF711
64 bits	$2^{32}(2^{96}-199)-1$	642	63.8	9D9630168CF05133
	$2^{48}(2^{80}-26)-1$	5912	63.8	5B71448E3F37B76B
	$2^{16}(2^{112}-4)-1$	22266	63.8	-1745866B6C66213DD
96 bits	$2^{128}(2^{64}-142)-1$	7050	95.8	-40B4E6A71E36C0941A928B57

For each curve, it can be verified using Magma or Sage that the orders of both the elliptic curve group and its twist are both prime. In each case, the Frobenius trace t satisfies $|t| > 1$ and the complexity of Pollard's rho algorithm, given by $\log_2(\sqrt{\pi/4} \cdot \sqrt{\#E(\mathbf{F}_p)})$, is roughly the desired level of security. Due to the computational expense of finding these curves, security levels of 128, 192 and 256 bits were out of reach. Instead of constructing curves curves at such security levels ourselves, we consider some already constructed curves given in [22].

target security level	curve name	prime p	b	ρ complexity
128 bits	**w-256-mont**	$2^{240}(2^{16}-88)-1$	85610	127.8
	w-256-mers	$2^{256}-189$	152961	127.8
192 bits	**w-384-mont**	$2^{376}(2^{8}-79)-1$	27798	191.5
	w-384-mers	$2^{384}-317$	-34568	191.8
256 bits	**w-512-mont**	$2^{496}(2^{16}-491)-1$	99821	255.8
	w-512-mers	$2^{512}-569$	121243	255.8

A table containing the Frobenius trace of each curve presented below.

Curve name	Frobenius trace
w-256-mont	3AE8AEC191AF8B462EF3A1E5867A815
w-256-mers	1BC37D8A15D9A39FDF54DFD6B8AE571F
w-384-mont	456480EB358AEDAC85B1232C7583BE25D641B76B4D671145
w-384-mers	29E150E114A2977E412562C2B3C81D859FB27E0984F19D0B
w-512-mont	9C757286D118AFD67F9B550F47B6719E20C2C66AF9B128C46C69D70E81670237
w-512-mers	A4C35B046B187CE4B03DA712682F4239C4A974C99F832DBC31EAC0C6FBCCA86B

For each curve, one can verify using Magma or Sage that the orders of the elliptic curve group and twist in question are prime and that the Frobenius trace t satisfies $|t| > 1$. These curves, with respect to the criteria listed in [22], are suitable for cryptosystems used in practice, i.e. an attacker's best bet to solving ECDLP on these curves is Pollard's rho algorithm. Additionally, arithmetic on these curves is fast.

8.2 Curves in the Montgomery (and (twisted) Edwards) model

Due to the lack of support for Montgomery curves in Magma, if one would like to brute force search for Montgomery curves satisfying certain criteria then Sage is likely currently the best option. In practice, for the sake of efficiency and general ease, one only considers curves $M_{A,B}$ in the Montgomery model with $B = 1$. As such, one can use Sage's

```
EllipticCurve([a1,a2,a3,a4,a6])
```

with $a_1 = a_3 = a_6 = 0$, $a_4 = 1$ and $a_2 = A$. That said, the best option for point counting on elliptic curves in Sage is an implementation of SEA that does not allow elliptic curves taken over fields of suitably large prime order as input. As such, the explicit construction of Montgomery curves satisfying the list of criteria mentioned has not been done. Instead, we discuss some already-constructed efficient and secure Montgomery curves developed by Microsoft Research, as presented below.

target security level	curve name	prime p	A	ρ complexity
128 bits	**ed-256-mont**	$2^{240}(2^{16} - 88) - 1$	-54314	126.8
	ed-256-mers	$2^{256} - 189$	-61370	126.8
192 bits	**ed-384-mont**	$2^{376}(2^{8} - 79) - 1$	-113758	190.5
	ed-384-mers	$2^{384} - 317$	-1332778	190.8
256 bits	**ed-512-mont**	$2^{496}(2^{16} - 491) - 1$	-305778	254.8
	ed-512-mers	$2^{512} - 569$	-2550434	254.8

The coefficient d in the twisted Edwards curves corresponding to these curves can be computed through the formulae presented earlier. A table containing the Frobenius trace of each curve presented below.

Curve name	Frobenius trace
ed-256-mont	13AAD11411E6330DA649B44849C4E1154
ed-256-mers	106556A94BD650E6C691EC643BB752C90
ed-384-mont	2A4BE076C762D8C9825225944DFC2407E406C7167336DD94
ed-384-mers	4CA0BB84A976997697B17EE9C7182C6EB8A4A3823EF64630
ed-512-mont	CCC0A98C8F32E3CBBF3E7EBB024842CB2099437935363F81733ADE04D1C927EC
ed-512-mers	1606BDFD840951119676E1EC2EDAAE83C8C56803CD1FFC1DAC61CB8D3D283F7A4

Similar to Subsection 8.1, each of these curves satisfies the efficiency and security criteria for Montgomery and (twisted) Edwards curves stated earlier. Each curve is seen to have order the product of 4 and a suitably large prime and the Frobenius trace t of each curve satisfies $|t| > 1$. As for efficiency, for each (twisted) Edwards curve $\xi_{a,d}$, we have $a = -1$ to reduce the cost of point addition by 1 multiplication. Similarly, for corresponding Montgomery curves $M_{A,B}$, these curves ensure that B is a quadratic residue in $\mathbf{F}_p$ to ensure that one can perform addition on the birationally equivalent Montgomery curve given by $y^2 = x^3 + Ax^2 + x$. On top of this, these curves guarantee a 'minimality' in how A and d are taken. Since these curves are intended for efficient arithmetic on both the (twisted) Edwards curve itself and its corresponding Montgomery curve, the condition of ensuring small coefficients for both curves is no easy task, and is detailed well in [22].

9 Further discussion

In this paper, we have considered properties pertaining to elliptic curves over finite fields, some common attacks that exploit poor implementations of the Diffie-Hellman key exchange protocol and then the discrete logarithm problem whose difficulty ensures the security of a well-posed implementation of Diffie-Hellman. Then, primes p for which multiplication and reduction in $\mathbf{F}_p$ is particularly efficient were discussed, as well as some efficient methods for computing integer multiples of points. Important and well known elliptic curve models were then overviewed, with a brief analysis of their efficiency, and properties that the curves in question should satisfy to ensure that the discrete logarithm problem is sufficiently difficult. The explicit construction of curves in the Weierstrass model attaining 32, 64 and 96 bits of security is then presented, along with a brief analysis of curves that attain 128, 256 and 512 bits of security whose means of construction were similar.

With the field of elliptic curve cryptography being in its youth, there is naturally a lot to be done. One might, and arguably should expect notable development over the next few decades within the field. Whether that development is due to making progress in solving the discrete logarithm problem on some classification of elliptic curves or strides made in the efficiency of arithmetic on curves in some (potentially new) model, can only be answered with time. How this development affects currently implemented elliptic curve cryptosystems is of similar interest.

As for us, there are a variety of interesting avenues to pursue. This includes attempts to improve the efficiency of computing integer multiples on curves in the Weierstrass model using birational maps or perhaps investigating the efficiency and security implications of efficient point tripling in the Hessian model. One might also be interested in considering the applicability of side-channel attacks on methods of improved efficiency in the (twisted) Edwards model using birational maps to the Montgomery model.

References

[1] W.S. Jevons. *The principles of science*. 1877.

[2] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[3] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[4] H.W. Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[5] Oliver L. Morain F. Atkin, A. Elliptic curves and primality proving. *Mathematics of computation*, 61(203):29–68, 1993.

[6] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 51–65. Springer, 1998.

[7] J.R. Sendra, F. Winkler, and S. Pérez-Díaz. Rational algebraic curves. *Algorithms and Computation in Mathematics*, 22, 2008.

[8] J.H. Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[9] L.C. Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.

[10] R.V. Gurjar and A.K. Pathak. A short Proof of Bezout's Theorem in P2. *Communications in Algebra*, 38(7):2585–2587, 2010.

[11] J.H. Silverman and J. Tate. *Rational Points on Elliptic Curves*. Undergraduate Texts in Mathematics. Springer New York, 2013.

[12] K. Fujii and H. Oike. An Algebraic Proof of the Associative Law of Elliptic Curves. *Advances in Pure Mathematics*, 07:649–659, 01 2017.

[13] Z. Zhen. Adaptive Elliptic Curve Sliding Window Scalar Multiplication Algorithm. *Journal of Beijing Jiaotong University*, 2, 2007.

[14] M. Joye and S. Yen. The Montgomery powering ladder. In *International workshop on cryptographic hardware and embedded systems*, pages 291–302. Springer, 2002.

[15] L.D. Olson. Side-channel attacks in ECC: A general technique for varying the parametrization of the elliptic curve. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 220–229. Springer, 2004.

[16] T. Izu, J. Kogure, M. Noro, and K. Yokoyama. Efficient Implementation of Schoof's Algorithm. volume 1514, pages 66–79, 01 1998.

[17] L.K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[18] T. Scholl. Isolated Curves and the MOV Attack. *IACR Cryptol. ePrint Arch.*, 2018:307, 2018.

[19] I.F. Blake, G. Seroussi, and N.P. Smart. *Advances in elliptic curve cryptography*, volume 317. Cambridge University Press, 2005.

[20] J.C. Bajard and S. Duquesne. Montgomery-friendly primes and applications to cryptography. *Journal of Cryptographic Engineering*, pages 1–17, 2021.

[21] D.J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[22] J. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–286, 2016.

[23] R. Kumar. An algorithm for finding square root modulo p. *arXiv preprint arXiv:2008.11814*, 2020.

[24] M. Joye and J. Quisquater. Hessian elliptic curves and side-channel attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 402–410. Springer, 2001.

[25] M. Lochter and A. Wiemers. Twist Insecurity. *IACR Cryptol. ePrint Arch.*, 2015:577, 2015.

Appendices

```python
from random import randint

def is_comp(a,n,q,s):
    if pow(a,q,n) == 1:
        return False
    for i in range(s):
        if pow(a,q*pow(2,i),n) == n-1:
          return False
    return True

def mra(n,k): #probabilistic primality test

    if n in [2, 3, 5, 7, 11]: #special cases
        return True

    for j in [2, 3, 5, 7, 11]: #trial division
        if n % j == 0 or n == 1:
            return False

    s, q = 0, n-1
    while q % 2 == 0: #rewriting n-1 as q*2^s
      q >>= 1
      s  += 1

    for i in range(k):
      a = randint(2,n-2)
      if is_comp(a,n,q,s):
        return False
    return True
```

Listing 1: Miller-Rabin primality test

```python
from miller_rabin import mra

def s_bit_prime(delta, s): #generate $2s$-bit Montgomery-friendly prime

    alpha = 8*delta
    beta  = 2*s-alpha
    gamma = 0

    p = pow(2,alpha+beta)-0*pow(2,alpha)-1
    while not mra(p,10) or not p % 4 == 3:
        p -= pow(2,alpha)
        gamma += 1
    return p, alpha, beta, gamma
```

Listing 2: Find s-bit secure Montgomery-friendly prime

```sage
#integer multiplication in the Weierstrass model

O = [0,1,0]

def wei_mult_p(m,p,a,b,X,Y,Z):
    if Z == 0:
        return O

    Q = O
    for k in "{0:b}".format(m):
        Q = wei_double_p(p,a,b,Q[0],Q[1],Q[2])
        if int(k) == 1:
            Q = wei_add_p(p,a,b,X,Y,Z,Q[0],Q[1],Q[2])
    return Q

def wei_double_p(p,a,b,X,Y,Z): #point doubling

    if Z == 0:
        return O

    alpha = (a * pow(Z,2,p) + 3 * pow(X,2,p)) % p
    beta  = Y * Z % p
    gamma = X * Y * beta % p
    delta = pow(alpha,2,p) - 8 * gamma
    mu    = pow(beta,2,p)

    X3 = 2 * beta * delta % p
    Y3 = (alpha * (4 * gamma - delta) - 8 * pow(Y,2,p) * mu) % p
    Z3 = 8*beta*mu % p

    if Z3 == 0:
        return O

    return [X3,Y3,Z3]

def wei_add_p(p,a,b,X1,Y1,Z1,X2,Y2,Z2): #point addition

    if Z1 == 0:
        return [X2,Y2,Z2]
    if Z2 == 0:
        return [X1,Y1,Z1]
    if [X1,Y1,Z1] == [X2,Y2,Z2]:
        return wei_double_p(p,a,b,X1,Y1,Z1)

    lam1 = Y1 * Z2 % p
    lam2 = X1 * Z2 % p
    lam3 = Z1 * Z2 % p

    t0 = Y2 * Z1 % p
    u = t0 - lam1
```

```
    uu = pow(u,2,p)
    t1 = X2 * Z1 % p
    v = t1 - lam2
    vv = pow(v,2,p)
    vvv = v * vv % p
    R = vv * lam2 % p
    t2 = 2 * R
    t3 = uu * lam3 % p
    t4 = t3 - vvv
    A = t4 - t2
    t5 = R - A
    t6 = vvv * lam1 % p
    t7 = u * t5 % p

    X3 = v * A % p
    Y3 = (t7 - t6) % p
    Z3 = vvv * lam3 % p

    return [X3,Y3,Z3]
```

Listing 3: Integer multiples in projective coordinates for curves in the Weierstrass model

```
#integer multiplication in the Mongtomgery model

from mod_sqrt    import mod_sqrt
from sympy       import mod_inverse

def mont_double_XZ(p, X, Z, A24): #point doubling

    if X == 0:
        return [0, 0]

    u  = X + Z
    v  = X - Z
    uu = pow(u,2,p)
    vv = pow(v,2,p)
    uv = uu - vv
    t  = A24 * uv + vv % p

    X3 = uu * vv % p
    Z3 = uv * t % p

    return [X3, Z3]

def mont_add_XZ(p, Xm, Zm, Xn, Zn, X1, Z1): #point addition

    u = (Xm - Zm) * (Xn + Zn) % p
    v = (Xm + Zm) * (Xn - Zn) % p
    w = u + v
    t = u - v
    ww = pow(w,2,p)
    tt = pow(t,2,p)
    X = ww * Z1 % p
    Z = tt * X1 % p

    return [X, Z]

def mont_ladder(k, p, X1, Z1, A): #the Montgomery ladder

    if k == 0:
        return [0, 0]

    A24 = (A + 2) * mod_inverse(4,p)
    P1, P2 = X1, Z1
    Q1, Q2 = mont_double_XZ(p, X1, Z1, A24)
    for i in bin(k)[3:]:
        if int(i) == 1:
            P1, P2 = mont_add_XZ(p, Q1, Q2, P1, P2, X1, Z1)
            Q1, Q2 = mont_double_XZ(p, Q1, Q2, A24)
        else:
            Q1, Q2 = mont_add_XZ(p, P1, P2, Q1, Q2, X1, Z1)
            P1, P2 = mont_double_XZ(p, P1, P2, A24)
```

```
51
52      return [P1, P2]
53
54  def y_recover(p,A,B,X,Z): #Y-coordinate recovery
55
56      if Z == 0:
57          return [0,1]
58
59      LHS = B * Z % p
60      RHS = (pow(X,3,p) + A * pow(X,2,p) * Z + X * pow(Z,2,p)) % p
61
62      inv_LHS = mod_inverse(LHS,p)
63      Y_sq_co = RHS * inv_LHS % p
64
65      Y = mod_sqrt(Y_sq_co,p)
66
67      x = B * X * inv_LHS % p
68      y = B * Y * inv_LHS % p
69
70      return [x,[y,p-y]] #return both possibilities of (x,y)
```

Listing 4: Integer multiples in projective coordinates for curves in the Montgomery model along with Y-coordinate recovery

```
#integer multiplication in the (twisted) Edwards model

O = [0,1,0]

def ted_mult_p(m,p,a,d,X,Y,Z):
    if Z == 0:
        return O

    Q = O
    for k in "{0:b}".format(m):
        Q = ted_double_p(p,a,d,Q[0],Q[1],Q[2])
        if int(k) == 1:
            Q = ted_add_p(p,a,d,X,Y,Z,Q[0],Q[1],Q[2])
    return Q

def ted_double_p(p,a,d,X,Y,Z): #point doubling

    if Z == 0:
        return O

    B = pow(X + Y,2,p)
    C = pow(X,2,p)
    D = pow(Y,2,p)
    E = a * C % p
    F = E + D
    H = pow(Z,2,p)
    J = F - 2 * H

    X3 = (B - C - D) * J % p
    Y3 = F * (E - D) % p
    Z3 = F * J % p

    return [X3,Y3,Z3]

def ted_add_p(p,a,d,X1,Y1,Z1,X2,Y2,Z2): #point addition

    if Z1 == 0:
        return [X2,Y2,Z2]
    if Z2 == 0:
        return [X1,Y1,Z1]

    A = Z1 * Z2 % p
    B = pow(A,2,p)
    C = X1 * X2 % p
    D = Y1 * Y2 % p
    E = d * C * D % p
    F = B - E
    G = B + E

    X3 = A * F* ((X1 + Y1) * (X2 + Y2)- C - D) % p
```

```
51      Y3 = A * G * (D - a * C) % p
52      Z3 = F * G % p
53
54      return [X3,Y3,Z3]
```

Listing 5: Integer multiples in projective coordinates for curves in the (twsited) Edwards model