Lancaster University

MSci Mathematics Dissertation

Cubic Curves Over Finite Fields

William Bolton

supervised by
Dr. Nadia Mazza

5th May 2016

Contents

Motivation **2**

1 Prerequisites **3**
1.1 Finite fields . . . 3
1.1.1 Finite fields . . . 3
1.1.2 The euclidean algorithm . . . 3
1.2 The projective plane . . . 5
1.2.1 Affine space and affine curves . . . 5
1.2.2 Projective space . . . 6
1.2.3 Construction 1: Quotient of $\mathbb{A}^{n+1}(k)$ . . . 7
1.2.4 Construction 2: Extension of $\mathbb{A}^n(k)$ . . . 7
1.2.5 Homogenisation and dehomogenisation . . . 8
1.2.6 Projective curves and projective geometry . . . 9
1.3 Elliptic curves . . . 12
1.3.1 Bézout's theorem . . . 12
1.3.2 The addition law on elliptic curves . . . 13
1.3.3 Canonical forms of elliptic curves . . . 14

2 Theory **15**
2.1 Elliptic curves over finite fields . . . 15
2.1.1 Rational points of elliptic curves . . . 15
2.1.2 Addition formulae for points . . . 15
2.2 The number of points on an elliptic curve . . . 16
2.2.1 Naïve counting . . . 16
2.2.2 The Hasse-Weil estimate . . . 16
2.2.3 Schoof's algorithm . . . 17
2.2.4 Gauss' theorem . . . 18
2.3 Lenstra's factorisation algorithm . . . 25
2.3.1 The modular exponentiation algorithm . . . 25
2.3.2 Fermat's little theorem . . . 25
2.3.3 Lenstra's algorithm . . . 26

3 Discussion **30**
3.1 Example RSA decryption . . . 30
3.2 Elliptic curve cryptography . . . 31

A Code **32**

References **39**

Motivation

The main topic of this dissertation is cubic curves over finite fields, and in order to discuss them, there are prerequisite topics that must first be introduced. Therefore, this document is split into three sections. Section 1 covers the necessary prerequisite material as mentioned before. Since this material is not the main focus of the document, some of the lengthier or more unenlightening proofs are not included, and the reader is instead provided with a reference for where they can find a proof and learn more about the topic. The majority of the material in the finite fields subsection comes from [6, 15], and the remainder of the section comes largely from [8] and [1], with [12] also being a useful resource.

Section 2 is the main section and is devoted to cubic curves over finite fields. We establish some fundamental theoretical results in the area and then describe Lenstra's algorithm, which is a number-theoretical factorisation algorithm. The main resources for this section were [13] and [12].

The final section, section 3, contains a brief discussion of the two main applications of the material from the previous chapter. The discussion of RSA is motivated both by lenstra's algorithm from [7], [13] and [9], and the discussion of elliptic curve cryptography is mainly due to [13].

To facilitate understanding, some figures have been included in the document. Figures 2 and 3 were created in Ti*k*Z, figures 1 and 4 were created with OS X's *Grapher* app, figure 5 was made using *Geogebra* and figure 6 was plotted using R.

1 Prerequisites

1.1 Finite fields

1.1.1 Finite fields

Definition 1.1 *A field is a commutative, unital ring in which every non-zero element is invertible.[15]*

Recall the group $\mathbb{Z}_n$, the group of integers modulo some natural number n. For a given n, we have already seen that we can make $\mathbb{Z}_n$ a ring by defining the natural multiplication law by $\widehat{a} \times \widehat{b} = \widehat{ab}$. From now on, when referring to elements of $\mathbb{Z}_n$ or $\mathbb{F}_p$, we will omit the caret that denotes that the elements are actually equivalence classes, and just write them as numerals. This is only to ease notation, and the elements of these groups/fields are still equivalence classes.

However, the ring $\mathbb{Z}_n$ is a field if and only if n is prime, and this is denoted by writing $\mathbb{Z}_n = \mathbb{F}_p$ to emphasise both that the modulus is prime and that the ring is a field.

The reason this is true is obvious when considering inverses; if we attempt to make a field out of $\mathbb{Z}_4$, we find that 2 has no inverse, since

$$\begin{aligned} 2 \times 0 &= 0 \\ 2 \times 1 &= 2 \\ 2 \times 2 &= 0 \\ 2 \times 3 &= 2. \end{aligned}$$

Therefore, $\mathbb{Z}_4$ fails to meet the criteria for a field, since $2 \neq 0$ has no inverse.

It should be noted, though, that it is possible to construct a field with four elements. More generally, it is possible to construct finite fields $\mathbb{F}_{p^e}$ of order p^e, with p prime and $e \in \mathbb{N}$. This is done by taking a quotient of the ring $\mathbb{F}_p[X]$ (the ring of polynomials in the indeterminate X with coefficients in $\mathbb{F}_p$) by an irreducible polynomial F of degree e.

In the case of $\mathbb{F}_4$, this can be done by quotienting $\mathbb{F}_2[X]$ with the (only) irreducible polynomial of degree 2 over $\mathbb{F}_2$ $F = X^2 + X + 1$, i.e., by letting

$$\mathbb{F}_4 = \mathbb{F}_2[X]/(X^2 + X + 1).$$

Let α be a root of F. Then $\alpha^2 + \alpha + 1 = 0$, and we see that $\alpha(\alpha + 1) = 1$. Thus, we have that $\mathbb{F}_4 = \{0, 1, \alpha, \alpha + 1\}$.

For simplicity's sake, from now on, when we refer to finite fields, we limit ourselves to "simple" finite fields of the form $\mathbb{Z}_p$, and generally refer to these as $\mathbb{F}_p$.

1.1.2 The euclidean algorithm

The question of finding inverses in $\mathbb{Z}_n$ has a well understood, general solution, which involves the euclidean algorithm.

Definition 1.2 *The euclidean algorithm takes two numbers $a, b \in \mathbb{N}$ and returns their* greatest common divisor (gcd) *by repeated division with remainder.*

We do not define the euclidean algorithm rigorously, as it has already been introduced in [6]. We provide an example of its usage as a refresher, though. The algorithm looks as follows when applied to 21 and 15:

$$\begin{aligned} \gcd(21,15) : 21 &= 1 \times 15 + 6 \\ 15 &= 2 \times 6 + 3 \\ 6 &= 2 \times 3 + 0 \end{aligned}$$

So $\gcd(21, 15) = 3$ Using the steps of the algorithm, it is possible to calculate

$$\begin{aligned} \gcd(21,15) = 3 &= 1 \times 15 - 2 \times 6 \\ &= 15 - 2 \times (21 - 1 \times 15) \\ &= 15 - 2 \times 21 + 2 \times 15 \\ &= 3 \times 15 - 2 \times 21 \end{aligned}$$

So $3 = 3 \times 15 - 2 \times 21$. This relates to inverses in a finite field in the following way.

Definition 1.3 *Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{N}$. A linear congruence is a congruence of the form*

$$ax \equiv b \mod n$$

Recall this basic definition from `MATH111`. The solutions of such congruences are well understood, as given by this straightforward proposition.

Proposition 1.1 *The congruence $ax \equiv b \mod n$ has solutions if and only if $\gcd(a, n) | b$.*

Proof. Let $d = \gcd(a, n)$. To prove $\Rightarrow$, suppose that the congruence has solutions. Let $x \in \mathbb{Z}$ be a solution, such that we can write $ax = qn + b$ for some $q \in \mathbb{Z}$, and therefore $b = ax - qn$. Since $d|a$ and $d|n$, clearly $d|b$.

For $\Leftarrow$, supposing that $d|b$, or $b = td$ for some $t \in \mathbb{Z}$. We can write $d = ra + sn$ for some $r, s \in \mathbb{Z}$, and therefore

$$b = td = t(ra + sn) = a(tr) + (ts)n \equiv a(tr) \mod n$$

and the congruence has $x = tr$ as a solution. ∎

This sheds light on why $\mathbb{Z}_n$ is a field when n is prime: in $\mathbb{Z}_n$, to find the inverse of a nonzero element a, we need to solve

$$ax \equiv 1 \mod n,$$

where proposition 1.1 clearly implies that a and n must be coprime for an inverse to exist. Therefore all elements $\{1, \ldots, n-1\}$ must be coprime with n for $\mathbb{Z}_n$ to be a field. Clearly, this happens if and only if n is prime.

Proposition 1.1 also tells us how to solve such congruences; by reversing the euclidean algorithm and letting $x \equiv r \mod n$ (note that t is dropped from the general solution since $b = 1$). When finding inverses in a finite field, since a and the modulus p are coprime, it boils down to the following.

To find the inverse of 52 in $\mathbb{F}_{83}$, we apply the euclidean algorithm to 52 and 83:

$$\begin{aligned} \gcd(83, 52) : 83 &= 1 \times 52 + 31 \\ 52 &= 1 \times 31 + 21 \\ 31 &= 1 \times 21 + 10 \\ 21 &= 2 \times 10 + 1 \end{aligned}$$

then write 1 as a linear combination of 52 and 83:

$$\begin{aligned} \gcd(83, 52) = 1 &= 1 \times 52 - 2 \times 10 \\ &= 3 \times 21 - 2 \times 31 \\ &= 3 \times 52 - 5 \times 31 \\ &= 8 \times 52 - 5 \times 83 \end{aligned}$$

and we have that the coefficient of 52 in the expression is 8. Therefore,

$$52^{-1} \equiv 8 \mod 83.$$

In this case, clearly $8 \in \mathbb{F}_{83}$, but on occasion, the fact that the solution is a congruence is important. A python implementation of the euclidean algorithm, together with the inverse-finding capacity, is included in appendix A.

1.2 The projective plane

1.2.1 Affine space and affine curves

The main object of interest in this subsection is the projective plane. Before we introduce it, though, we begin with a prerequisite definition from [8].

Definition 1.4 *Let k be a field. Define affine n-space, or $\mathbb{A}^n(k)$, to be the set of all n-tuples, or points,*

$$\mathbb{A}^n(k) = \{(a_1, a_2, \ldots, a_n) | a_i \in k\}.$$

In this section, we take the underlying field k to be $\mathbb{C}$, as in [1, 8], which differentiates affine space from $\mathbb{R}^n$ since $\mathbb{C}$ is algebraically closed, whereas $\mathbb{R}^n$ is not. This means that we can split all polynomials, which is useful when considering their roots. Also, from now on, if it is clear which field is being worked in, we write $\mathbb{A}^n$ for short instead of $\mathbb{A}^n(k)$. We call affine 2-space, or $\mathbb{A}^2$, the *affine plane*.

In affine n-space, it is possible to define *affine curves* in the following way.

Definition 1.5 *In $\mathbb{A}^n(k)$, for a given polynomial $f \in k[x_1, \ldots, x_n]$, an* affine curve *is the set $V(f)$ of all points $(x_1, \ldots, x_n) \in \mathbb{A}^n(k)$ such that $f(x_1, \ldots, x_n) = 0$. More concisely,*

$$V(f) = \{(x_1, \ldots, x_n) \in \mathbb{A}^n(k) | f(x_1, \ldots, x_n) = 0\}$$

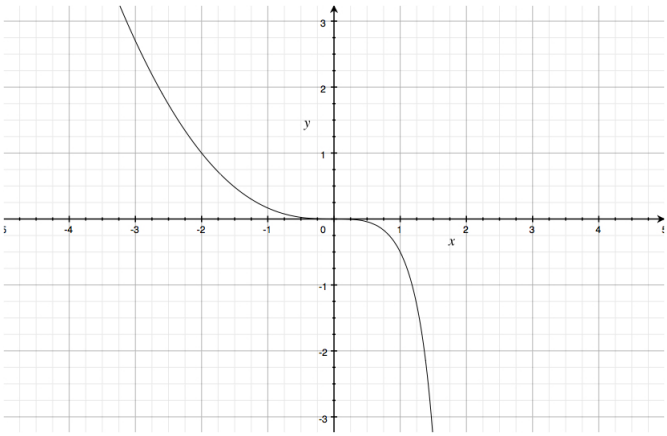

Figure 1: The affine curve $x^3 - 2xy + 4y = 0$ in the visible part of $\mathbb{A}^2$

Figure 1 shows an example of an affine curve. Note that, when plotting affine curves, we only plot the real part of them, as they may have points with complex coordinates.

It is important to distinguish the two objects $V(f)$ and f, as f is simply a polynomial, whereas $V(f)$ is the set of points that are a solution to a polynomial *equation* $f(x_1, \ldots, x_n) = 0$. However, for convenience, we write the curve $V(f)$ as its defining polynomial equation $f(x_1, \ldots, x_n) = 0$, when it is clear to do so, or even refer to just the curve f. The distinction of objects remains, though, and this is purely notational.

More points on notation are that in the affine plane, instead of x_1 and x_2, we write x and y, lowercase. We also write polynomials as lowercase letters f, g etc. The reason for this will be explained shortly.

1.2.2 Projective space

Now that affine space has been introduced, we are ready to introduce projective n-space $\mathbb{P}^n(k)$, and with that, the projective plane $\mathbb{P}^2(k)$.

The projective plane is an idea that grew out of the renaissance study of perspective in art. For example, when stood between two railway lines, they appear to meet at the horizon. Figure 2 is a diagram of the situation. While this is merely a trick of perspective in euclidean space, in the projective plane, parallel lines *do* meet, at a so called 'point at infinity'. The idea is to add these points at infinity to the affine plane so that each line contains exactly one. On top of this is the one exception, the 'line at infinity', which is defined to be the unique line that is simply the collection of all points at infinity. These ideas, while formative, are not rigorous, and we present the proper definition and constructions (there are two) of projective space below.

The two constructions of projective space give two different interpretations of it, both of which are useful to consider. Projective n-space can be thought of as either affine n-space with points at infinity added (mentioned already) or as a quotient of affine $(n + 1)$-space. For both of these constructions, k is the underlying field of affine space $\mathbb{A}^n(k)$.

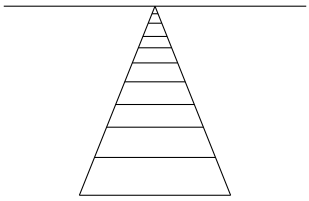

Figure 2: Parallel lines 'meeting' at infinity in euclidean space

1.2.3 Construction 1: Quotient of $\mathbb{A}^{n+1}(k)$

Remove the origin (i.e. the point $(0, 0, \ldots, 0)$) from $\mathbb{A}^{n+1}(k)$ and define an equivalence relation $\sim$ on the remaining points as follows:

$$(a_0, a_1, \ldots, a_n) \sim (b_0, b_1, \ldots, b_n) \iff (a_0, a_1, \ldots, a_n) = \lambda(b_0, b_1, \ldots, b_n)$$

where λ is a non-zero scalar in $\mathbb{C}$. Then we have

$$\mathbb{P}^n = (\mathbb{A}^{n+1}(k) \setminus \{(0, \ldots, 0)\})/\sim,$$

the set of equivalence classes of $\sim$.

Therefore, we see that points in $\mathbb{P}^n$ are equivalence classes of lines in $\mathbb{A}^{n+1}(k) \setminus \{0\}$. For points in $\mathbb{P}^n$, write $P = [p_0, p_1, \ldots, p_n]$, where the square brackets represent the *homogeneous coordinates* of the point. We see that $[p_0, p_1, \ldots, p_n] = \{(\lambda p_0, \lambda p_1, \ldots, \lambda p_n) | \lambda \in \mathbb{C}, \lambda \neq 0\}$.

By the nature of this construction, it is clear that scaling homogeneous coordinates results in coordinates of the same point. Therefore, for a given point, there is no unique representation in homogeneous coordinates, and we may scale coordinates for our convenience.

1.2.4 Construction 2: Extension of $\mathbb{A}^n(k)$

Now we present the second construction of $\mathbb{P}^n$, which can be thought of as adding points at infinity to affine n-space. This is done inductively on n by defining maps α_n and β_n. To begin with, let $\mathbb{P}^0 = \mathbb{A}^1(k) \setminus \{0\}$ and let

$$\begin{aligned} \alpha_n &: \mathbb{A}^n(k) \to \mathbb{P}^n, \quad \alpha_n(x_1, \ldots, x_n) = [x_1, \ldots, x_n, 1] \\ \beta_n &: \mathbb{P}^{n-1} \to \mathbb{P}^n, \quad \beta_n[X_1, \ldots, X_n] = [X_1, \ldots, X_n, 0] \end{aligned}$$

Then we can define

$$\mathbb{P}^n = \alpha_n(\mathbb{A}^n(k)) \cup \beta_n(\mathbb{P}^{n-1})$$

Effectively, we have that α_n is the embedding of affine space in $\mathbb{P}^n$ and β_n represents the points at infinity of $\mathbb{P}^n$. Notice that the image of β_n will always give valid homogeneous coordinates, as we removed zero from $\mathbb{P}^0$, and therefore at least one of the first n coordinates will be nonzero.

1.2.5 Homogenisation and dehomogenisation

Now that projective space has been introduced, we establish how to transition between affine space $\mathbb{A}^n$ and projective space $\mathbb{P}^n$. The second construction of projective space is the most enlightening in this regard. The map α_n gives the image of a point $p = (x_1, \ldots, x_n) \in \mathbb{A}^n$ in $\mathbb{P}^n$ as $[x_1, \ldots, x_n, 1]$, and this process is known as *homogenisation*. From the first construction, which included the definition of homogeneous coordinates, we see that this can be scaled by any nonzero element of k, say λ, for example. Then if we have homogeneous coordinates $[\lambda x_1, \ldots, \lambda x_n, \lambda]$, we can recover p by dividing through by λ (which is safe, since $\lambda \neq 0$) such that the final homogeneous coordinate is 1, where we can recover the coordinates of the point from the first n coordinates. This transition from $\mathbb{P}^n$ to $\mathbb{A}^n$ is known as *dehomogenisation*, since it is the reverse transition to homogenisation. More efficiently, to dehomogenise a point $P = [X_1, \ldots, X_n, X_{n+1}] \in \mathbb{P}^n$ where $X_{n+1} \neq 0$, we let $p = (\frac{X_1}{X_{n+1}}, \ldots, \frac{X_n}{X_{n+1}}) \in \mathbb{A}^n$. Of course, the fact that we cannot dehomogenise points with final homogeneous coordinate 0 is to be expected, since these are the points at infinity, which do not exist in $\mathbb{A}^n$.

If we try to define projective curves in $\mathbb{P}^n$ in the same way as affine curves, i.e. as solutions of a polynomial equation $f(x_1, \ldots, x_n) = 0$ in n variables, we find that it is not always possible, since we saw that the homogeneous coordinates of a point are not unique. For example, if we tried to define a projective curve as $F(X, Y, Z) = X^2 + Y + Z = 0$, we see that $[2, -2, -2]$ is a solution of this equation, and so $[2, -2, -2] \in F$. However, since we work with homogeneous coordinates, the point $[2, -2, -2]$ can also be written as $[-2, 2, 2]$, which clearly does not satisfy $F(X, Y, Z) = 0$. This is a problem, as we do not want the representation of a point to affect whether it lies on a curve or not.

To that extent, when working with the projective plane, it is convenient to define a homogeneous polynomial as a polynomial which satisfies

$$F(tX_1, tX_2, \ldots, tX_n) = t^e F(X_1, X_2, \ldots, X_n)$$

for some $t \in k$ and $e \in \mathbb{N}$. We see that these kinds of polynomials are able to define curves in the projective plane, as for a homogeneous polynomial F,

$$F(X_1, X_2, \ldots, X_n) = 0 \Leftrightarrow F(tX_1, tX_2, \ldots, tX_n) = t^k F(X_1, X_2, \ldots, X_n) = 0$$

It is easy enough to see that a polynomial is homogeneous if and only if the degree of each monomial $= d$, for some $d \in \mathbb{N}$. So the polynomial $\frac{1}{2}x^4 + x^2y^2 + 14xy^3$ is homogeneous with degree 4, whereas the polynomial $17x^5 + 12x^2y + 2$ is not, as the monomials have degrees 5, 3 and 0, from left-to-right.

It is possible to homogenise a given polynomial $f(x_1, \ldots, x_n)$ by introducing another variable x_{n+1} and multiplying each monomial by x_{n+1} until it has degree d, where d is the maximum degree of all monomials of F. In practice, this resembles 'topping up' the monomials with an extra variable as necessary. So the inhomogeneous polynomial $17x^5+12x^2y+2$ discussed earlier can be homogenised as $17X^5+12X^2YZ^2+2Z^5$. The reverse transition can be made also, simply by letting $f = f(x_1, \ldots, x_n) = F(x_1, \ldots, x_n, 1)$, which looks like simply taking out the extra variable.

When writing homogeneous polynomials, we use uppercase letters to distinguish them from affine variables and make the context clear. We also use X, Y and Z (and also x and y) as the three variables when dealing with the projective plane (uppercase) and the affine part thereof (lowercase).

To summarise, it is possible to move to (homogenisation) and from (dehomogenisation) the projective plane with regards to both points and polynomials. A point $(x, y) \in \mathbb{A}^2$ can be homogenised as $[x, y, 1] \in \mathbb{P}^2$, and $[X, Y, Z]$ dehomogenised as $(X/Z, Y/Z)$. A polynomial $f(x, y)$ can be homogenised as $F(X, Y, Z)$ by adding powers of Z to each monomial as appropriate, and $F(X, Y, Z)$ dehomogenised as $f(x, y) = F(x, y, 1)$.

1.2.6 Projective curves and projective geometry

Now that the concepts of homogenisation and dehomogenisation have been established, we introduce a few more ideas necessary for understanding the main subject matter. From now on, we also limit ourselves to the projective plane $\mathbb{P}^2$.

Definition 1.6 *A projective curve is the set of points defined by a* homogeneous *polynomial equation $F(X_1, X_2, \ldots, X_n) = 0$. When discussing the projective plane $\mathbb{P}^2$, we write $F(X, Y, Z) = 0$. The degree of a curve is the degree of the polynomial in the defining equation.*

For example, the polynomial equations $X^3 - YZ^2 = 0$, $13X^2Z + Y^3 + 4XYZ = 0$ and $Z^2 + \frac{1}{2}Y^2 = 0$ all define projective curves in $\mathbb{P}^2$. The first two have degree 3, the third, degree 2.

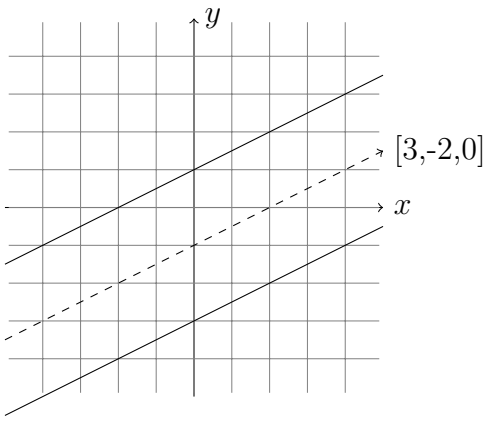

Figure 3: The intersection of $2x - 3y + 1 = 0$ and $2x - 3y - 3 = 0$

While it is not possible to directly plot projective curves, as they contain points with complex coordinates and points at infinity in the projective plane, it is possible to see the affine part of projective curves F by dehomogenising F and looking at the resultant affine curve f. The reverse is also true, and the parallel lines given by $2x - 3y + 1 = 0$ and $2x - 3y - 3 = 0$ in $\mathbb{A}^2$ intersect in the projective plane at the point [3,-2,0]. This is because, when homogenised with respect to Z, the curves are

given by $2X - 3Y + Z = 0$ and $2X - 3Y - 3Z = 0$, which are clearly satisfied by $[X, Y, Z] = [3, 2, 0]$. As we would expect, this is a point at infinity, since $Z = 0$. This is illustrated in figure 3.

Definition 1.7 *Let F be a curve in $\mathbb{P}^2$ and $P = [a, b, c]$ be a point on F. Let L be a line through P with parametrisation*

$$\varphi(S, T) = SP + TQ = [Sa + Tu, Sb + Tv, Sc + Tw]$$

for another point Q on L. The intersection multiplicity *of L and F at P is the multiplicity of $\varphi(1, 0)$ as a solution of $F \circ \varphi = 0$, that is, the largest index i such that T^i is a factor of $F \circ \varphi = 0$ The intersection is* multiple *if the intersection multiplicity is greater than one, else it is* regular*.*

The concept of intersection multiplicity appears frequently in projective geometry, as we will see later.

We give an example of explicitly calculating the intersection multiplicity between the curve $F : YZ^2 - X^3 = 0$ and the line $Y = 0$ at the point $P = [0, 0, 1]$. Let $Q = [1, 0, 1]$ such that L has the parametrisation

$$L : [S, T] \mapsto [T, 0, S + T].$$

Then $F(T, 0, S+T) = 0(S+T)^2 - T^3 = -T^3$, and since T^3 is a factor, the intersection has multiplicity 3.

Definition 1.8 *A singular point on a curve F is a point P where all partial derivatives F_X, F_Y and $F_Z = 0$. An alternative characterisation is that the intersection multiplicity of a line and F at P is always greater than one. A point that is not a singular point is called a regular point.*

Two fundamental examples of projective curves with singular points are provided by the *cusp* $C : Y^2Z - X^3$ and the *node* $N : Y^2Z - X^3 - X^2Z$, which are illustrated in figure 4. We can see that the point $[0, 0, 1]$ (the origin in the affine plane) is a

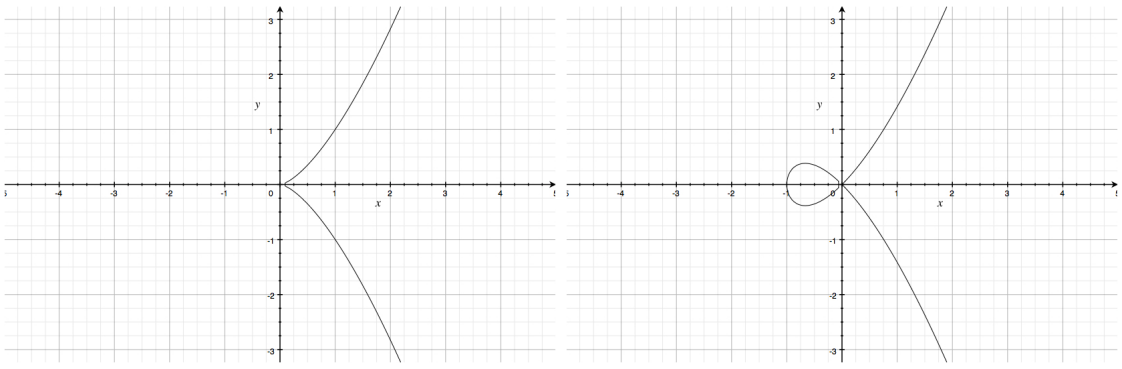

Figure 4: The cusp (left) and node (right) in the visible part of $\mathbb{A}^2$

singular point on both curves, since

$$\begin{aligned} C_X &= -3X^2, & N_X &= -3X^2 - 2XZ, \\ C_Y &= 2YZ, & N_Y &= 2YZ, \\ C_Z &= Y^2, & N_Z &= Y^2 - X^2, \end{aligned}$$

and all partial derivatives vanish at $[0, 0, 1]$.

Singular points are important to consider as they are the points where there is no sensible notion of the tangent to a curve. The importance of this will become apparent later on.

Supposing that a point on a curve is regular, and therefore has a tangent. It is possible to find the equation of the line by examining the partial derivatives of the curve. For example, let $F : X^3 - 3XZ^2 - Y^2Z = 0$ and consider the point $[0, 1, 0]$ on F. We have $F_X(X, Y, Z) = 3X^2 - 3Z^2$, $F_Y(X, Y, Z) = -2YZ$ and $F_Z(X, Y, Z) = -6XZ - Y^2$. The equation of the tangent line L is given by

$$L : F_X(P)X + F_Y(P)Y + F_Z(P)Z = 0$$

In this case, we have $L : -X - Z = 0$, or $L : X + Z = 0$. As alluded to before, being able to find tangent lines plays an important part in our upcoming material.

Definition 1.9 *A curve that contains one or more singular points is a singular curve. Otherwise, the curve is regular.*

Definition 1.10 *A* flex *is a point P on a curve F where the tangent to F at P intersects F with multiplicity at least 3.*

The importance of flexes will become apparent in the next subsection. Notice that the intersection of $F : Y^2Z - X^3 = 0$ and $Y = 0$ that we calculated in definition 1.7 is a flex of F, since it has intersection multiplicity 3.

Definition 1.11 *A projective transformation is a map $\pi : \mathbb{P}^2 \to \mathbb{P}^2$ represented by a matrix*

$$\Pi = \begin{pmatrix} \pi_{11} & \pi_{21} & \pi_{31} \\ \pi_{12} & \pi_{22} & \pi_{32} \\ \pi_{13} & \pi_{23} & \pi_{33} \end{pmatrix}.$$

To find the image P' of a point $P = [X, Y, Z]$ under the projective transformation π, we simply calculate $P' = \Pi[X, Y, Z]^{\mathrm{T}}$. Since we work in the projective plane, and therefore homogeneous coordinates, two projective transformations π and π' are the same if and only if $\exists \lambda \in k$ such that $\Pi = \Pi'$.

We do not go into detail about projective transformations, but merely introduce them for their utility; the existence of projective transformations often allows simplifying assumptions to made about curves and points in the projective plane.

1.3 Elliptic curves

Knowledge of projective geometry is vital to understanding elliptic curves, as they are curves that are defined in the projective plane. Therefore from now on, unless otherwise stated, we assume to be working in $\mathbb{P}^2$. As with all projective curves though, it is possible to visualise parts of elliptic curves as they appear in the affine plane.

Definition 1.12 *An* elliptic curve *is a regular, projective cubic curve.*

Here, a cubic simply means that the polynomial that defines the curve is of degree three. Generally, we use the letter E for an elliptic curve.

So for example, $E(X, Y, Z) = \frac{1}{3}X^3 + 2YZ^2 + 7Y^3$ defines the elliptic curve $E(X, Y, Z) = 0$. Since E is a homogeneous polynomial of degree three, it suffices to check that there are no singular points.

$$E_X = X^2 \tag{1}$$
$$E_Y = 2Z^2 + 21Y^2 \tag{2}$$
$$E_Z = 4YZ \tag{3}$$

Suppose that $E_X = E_Y = E_Z = 0$ at a point $P = [X, Y, Z]$. Equation (1) implies that $X = 0$. Equation (3) now implies that either $Y = 0$ or $Z = 0$. Either way, equation (2) subsequently implies that $X = Y = Z = 0$, which is not a valid point in homogeneous coordinates. Therefore, there are no singular points on the curve E and it is indeed an elliptic curve.

The following theorem gives a fundamental property of elliptic curves.

Theorem 1.1 *Let E be an elliptic curve. Then E has at least one flex in $\mathbb{P}^2$.*

Proof. We do not prove this theorem, but instead refer readers to [1, §12] for more information. ∎

1.3.1 Bézout's theorem

Before we go into more detail on elliptic curves, we require an important theorem of Bézout.

Theorem 1.2 *Let F and G be curves in $\mathbb{P}^n(k)$ of degree d_F and d_G respectively. Suppose that k is algebraically closed, F and G have no common factors and we count each intersection point with its intersection multiplicity. Then $F \cap G$ has intersection multiplicity $= d_F d_G$.*

This is *Bézout's theorem*, which is a fundamental result in projective geometry. Each assumption of Bézout's theorem is important, as each is required for the theorem to hold. This is one reason why we take the base field k of affine and projective space to be $\mathbb{C}$.

Proof. Again, we do not prove Bézout's theorem as it is lengthy and not insightful. We refer readers to [12] or [1]. ∎

1.3.2 The addition law on elliptic curves

Our interest in elliptic curves is mainly due to a special property of theirs. It is possible to turn the set of points of an elliptic curve E into an abelian group by introducing an addition law that satisfies all of the group axioms. We do not prove this in this document but direct readers to [12]. The addition rests upon the so-called

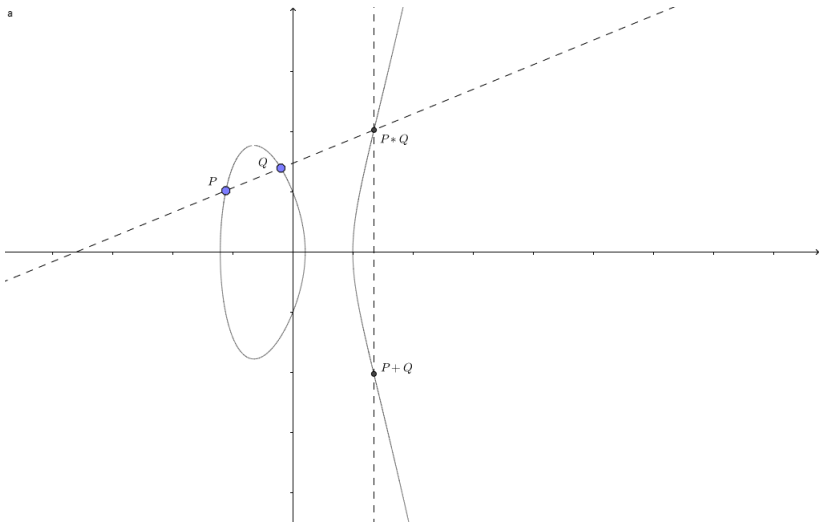

Figure 5: The chord-tangent law $*$ and the addition law $+$ on an elliptic curve

'chord-tangent law', which is illustrated in figure 5, along with the full addition operation. Since an elliptic curve is cubic and regular, every line intersects it with multiplicity exactly three. This means that given two points (not necessarily distinct), the line on which those points lie (or the tangent line to the curve at the point, if the points are the same) will intersect the curve in exactly one other place. The chord-tangent law $*$ for two points $P, Q \in E$ is that we define $P * Q$ to be the third point of intersection on the line containing P and Q (which may be P or Q again, if they intersect E with multiplicity greater than 1).

However, the chord-tangent law alone is not enough to turn E into a group, since the operation is not associative, as is required by a group operation. This can be remedied, though, by defining the elliptic curve addition law as follows.

Theorem 1.3 *For points P, Q and R on an elliptic curve with a flex $\mathcal{O}$, the addition law $+$ defined by*

$$P + Q = \mathcal{O} * (P * Q)$$

turns the curve into an abelian group, where $*$ is the chord-tangent law. The flex $\mathcal{O}$ acts as the identity element of the group.

This is known as Poincaré's theorem. Note that theorem 1.1 ensures that there will be a flex on the curve we can take to be $\mathcal{O}$.

Proof. Let $P, Q, R \in E$. We have

$$P + Q = \mathcal{O} * (P * Q) = \mathcal{O} * (Q * P) = Q + P,$$

so $+$ is commutative. Also,

$$\mathcal{O} + P = \mathcal{O} * (\mathcal{O} * P) = P$$

because P, $\mathcal{O}$ and $\mathcal{O}P$ are the three intersection points of the line through P and $\mathcal{O}$ with E. Therefore $\mathcal{O}$ is indeed the identity element, and $-P = \mathcal{O} * P$. Finally, to show associativity,

$$\begin{aligned} P * (Q + R) &= P * (\mathcal{O} * (Q * R)) \\ &= ((P * Q) * Q) * (\mathcal{O} * (QR)) \\ &= ((P * Q) * \mathcal{O}) * (Q * (Q * R)) \\ &= (\mathcal{O} * (P * Q)) * R \\ &= (P + Q) * R \end{aligned}$$

and thus $P + (Q + R) = \mathcal{O} * (P * (Q + R)) = \mathcal{O} * ((P + Q) * R) = (P + Q) + R$. ■

Formulas exist which allow this addition to be easily calculated, which we postpone providing until the following section.

1.3.3 Canonical forms of elliptic curves

As we have introduced them so far, elliptic curves have no specific form, other than being defined by a homegeneous polynomial of degree three. The following theorem is useful when working with elliptic curves.

Theorem 1.4 *For any elliptic curve given by a polynomial E, there exist projective transformations which put E into one of the following forms:*

- $Y^2Z - F(X, Z) = 0$ *where* $F(X, Z) = X^3 + aX^2Z + bXZ^2 + cZ^3$ *for some* $a, b, c \in k$
- $Y^2Z - F(X, Z) = 0$ *where* $F(X, Z) = 4X^3 - pXZ^2 - qZ^3$ *for some* $p, q \in k$

In either form, the curve is regular if and only if $f(x)$, the dehomogenisation of $F(X, Z)$, has no repeated roots. The point $\mathcal{O} = [0, 1, 0]$ is the unique point at infinity on the curve, and is a flex. The line at infinity $Z = 0$ is tangent to E at $\mathcal{O}$.

When dehomogenising these forms, notice that we have the affine curves $y^2 = x^3 + ax^2z + bxz^2 + cz^3$ and $y^2 = 4x^3 - pxz^2 - qz^3$. Since $\mathcal{O} = [0, 1, 0]$ acts as the identity on curves of this form, the elliptic curve addition law is sometimes stated as $P + Q$ equals the reflection of $P * Q$ in the x-axis, as in figure 5.

When a curve is in the first form given, we say that it is in *normal form*. When in the second form, we say that it is in *Weierstrass form*.

Proof. Again, we do not prove this theorem due to its length, but refer readers to [8, 1]. ■

2 Theory

2.1 Elliptic curves over finite fields

2.1.1 Rational points of elliptic curves

Now that the groundwork for elliptic curves has been laid, we are ready to address the main subject matter, that of elliptic curves over finite fields. As stressed before, we restrict our attention to finite fields $\mathbb{F}_p$ of the form $\mathbb{Z}_p$, the integers modulo a prime p. We also take all curves to be in normal form from here out, and only look at the affine parts of the curves. The underlying field k of $\mathbb{A}^n$ and $\mathbb{P}^n$ we now take to be $\mathbb{F}_p$ instead of $\mathbb{C}$ for the rest of the document, unless otherwise specified.

Therefore we write curves as E, $E(\mathbb{F}_p)$ or $E : y^2 = f(x)$, where $f(x)$ has no repeated roots. Points on the curve are of the form (x, y) such that $x, y \in \mathbb{F}_p$ and $y^2 = f(x)$, and we refer to these as *rational points*. By changing the field k, we have coefficients of $f(x)$ and points on the curve (x, y) with values in $\mathbb{F}_p$, and so the visible part of an elliptic curve E is represented by a collection of points in $\mathbb{A}^2(\mathbb{F}_p)$. The point at infinity $\mathcal{O}$ remains on the curve at $[0, 1, 0]$. Figure 6 gives a visual example of a curve over a finite field.

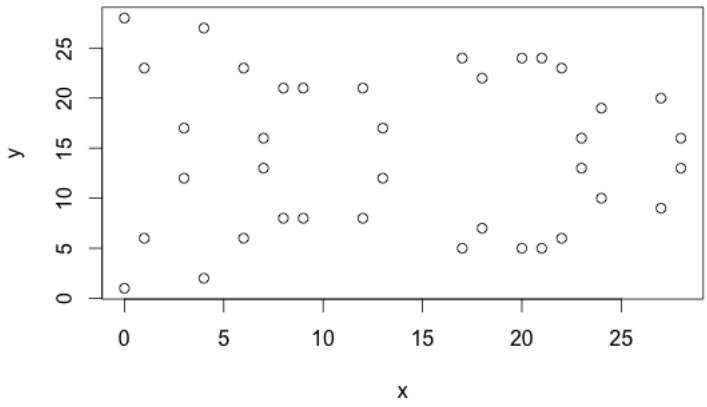

Figure 6: The curve $y^2 = 4x^3 + 2x + 1$ over $\mathbb{F}_{29}$

Since there are now a finite number of points, we can define the order of a curve E to be the number of rational points on the curve $+1$, to include the point at infinity.

2.1.2 Addition formulae for points

For elliptic curves over a finite field $\mathbb{F}_p$, when adding points $(x_1, y_1)+(x_2, y_2) = (x', y')$,

$$x' = \lambda^2 - a - x_1 - x_2, \quad y' = \lambda x_1 - \lambda x' - y_1$$

where $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ if $x_1 \neq x_2$ or $\lambda = \frac{3x_1^2 + 2ax_1 + b}{2y_1}$ otherwise.[13] These formulae are only valid when neither (x_1, y_1) nor $(x_2, y_2) = \mathcal{O}$, since $\mathcal{O}$ has no coordinates in $\mathbb{F}_p \times \mathbb{F}_p$.

It is important to remember that we are now dealing with arithmetic in $\mathbb{F}_p$, and so the division referenced is actually multiplication by the inverse element in $\mathbb{F}_p$, which can be found with the euclidean algorithm as before. Another consequence is that all calculations are performed mod p.

2.2 The number of points on an elliptic curve

2.2.1 Naïve counting

If one has taken an elliptic curve E defined by $y^2 = f(x)$ over a finite field $\mathbb{F}_p$, a fundamental question to ask is about how many points there are on the curve. We have already established that there is still a point at infinity on the curve (in Weierstrass form, at $[0, 1, 0]$), but apart from that, there are a maximum of p^2 points. The naïve approach, of course, is to simply run through all sets of points $(x, y) \in \mathbb{A}^2(\mathbb{F}_p)$ and test whether each point satisfies $y^2 = f(x)$, but depending on the size of the field, this may or may not be practical. Despite this, a basic example of this method of counting is ran through in section 2.2.2.

There are a few approaches possible to take when considering the number of points on an elliptic curve over a finite field, though, and we will cover a number of them.

2.2.2 The Hasse-Weil estimate

The most fundamental theoretical result in finding the order of $E(\mathbb{F}_p)$ is the Hasse-Weil theorem, which bounds the size of the group in terms of the prime modulus. The theorem was initially proved for elliptic curves by Helmut Hasse in [3], [4] and [5], and subsequently generalised to curves of higher genus by André Weil in [14], proved by Pierre Deligne in [2].

Theorem 2.1 *For an elliptic curve $E : y^2 = f(x)$ over a finite field $\mathbb{F}_p$, the number of points on $E = |E(\mathbb{F}_p)| = p + 1 + \epsilon$, where $|\epsilon| \leq 2\sqrt{p}$.[13]*

This is the statement of Hasse's original result, as it pertains to elliptic curves. The generalisation comes from introducing the genus g and allowing F to be any curve, so that the theorem reads $|F(\mathbb{F}_p)| = p + 1 + \epsilon$, where $|\epsilon| \leq 2g\sqrt{p}$. Since elliptic curves have genus 1, we get Hasse's result from the extended theorem, known as the Hasse-Weil theorem.

Sketch of proof. The proof of this theorem is very technical, and so will be omitted. Those interested can consult [3, 4, 5]. We give an intuitive, non-rigorous sketch of proof though, and prove a theorem of Gauss from [13] as a supplement.

Since there are p possibilites for x in $\mathbb{F}_p$, if $f(x)$ is a square in $\mathbb{F}_p$ for a particular x, then this will yield two points on the curve E. Otherwise, it will yield no points. Intuitively, one would not expect there to be a tendency for $f(x)$ to be a square or not to be a square, and so we assume that each x in $\mathbb{F}_p$ has a 50% chance of yielding no points and a 50% chance of yielding two points. Therefore, one would

expect there to be p points from $x \in \mathbb{F}_p$ and one more from the point at infinity $\mathcal{O}$. Allowing for a small error term ϵ, we arrive at

$$|E(\mathbb{F}_p)| = p + 1 + \epsilon.$$

■

We can test the Hasse-Weil estimate by calculating the number of points on an elliptic curve E over a particular finite field $\mathbb{F}_p$. For ease of calculation, we will take $\mathbb{F}_p = \mathbb{F}_7$ and let $E : y^2 = x^3 + x + 1$. First, we write out the squares in $\mathbb{F}_7$:

$$\begin{aligned} 0^2 &= 0 \\ 1^2 &= 1 \\ 2^2 &= 4 \\ 3^2 &= 2 \\ 4^2 &= 2 \\ 5^2 &= 4 \\ 6^2 &= 1 \end{aligned}$$

Then we write out $x^3 + x + 1$ and match to the relevant squares.

$$\begin{aligned} 0^3 + 0 + 1 &= 1 \\ 1^3 + 1 + 1 &= 3 \\ 2^3 + 2 + 1 &= 4 \\ 3^3 + 3 + 1 &= 3 \\ 4^3 + 4 + 1 &= 6 \\ 5^3 + 5 + 1 &= 5 \\ 6^3 + 6 + 1 &= 6 \end{aligned}$$

Which leads us to the four points

$$\begin{aligned} &(0, 1) \\ &(0, 6) \\ &(2, 2) \\ &(2, 5) \end{aligned}$$

Including the point at infinity $\mathcal{O}$, the Hasse-Weil theorem gives $|E(\mathbb{F}_7)| = 7 + 1 + \epsilon$, where $|\epsilon| \leq 2\sqrt{7}$, which bounds the number of points on E as 8 ± 5, or $|E(\mathbb{F}_7)| \in \{3, 4, \ldots, 13\}$. Clearly, 5 agrees with this estimate.

2.2.3 Schoof's algorithm

The Hasse-Weil estimate gives an important bound on the number of rational points over an elliptic curve over a finite field, but otherwise does not give the exact number.

For this purpose, in 1985, René Schoof published a paper detailing what is known as *Schoof's algorithm*. This was the first algorithm of its kind to run in polynomial time, as opposed to exponential time.

We do not go into detail on the algorithm, but mention it for its theoretical significance. Those who are interested can refer to Schoof's paper detailing the algorithm, [11].

2.2.4 Gauss' theorem

Theorem 2.2 *Define M_p to be the number of projective solutions to the equation $x^3 + y^3 + z^3 = 0$ in the field $\mathbb{F}_p$. Then,*

- *if $p \not\equiv 1 \mod 3$, then $M_p = p + 1$.*
- *otherwise $p \equiv 1 \mod 3$, and there exist integers A and B such that*

$$4p = A^2 + 27B^2$$

Since $A^2 \equiv 1 \mod 3$, $A \equiv \pm 1 \mod 3$. If we fix the sign of A so that $A \equiv 1 \mod 3$, then

$$M_p = p + 1 + A.$$

Proof. The proofs of the two separate cases both rely on the same idea, but in the second case requires many more steps before the result is reached. Therefore we prove the first case, then provide an outline of the proof second case before proving it.

For both cases, the proof revolves around examining the group homomorphism

$$\varphi : \mathbb{F}_p^\times \to \mathbb{F}_p^\times \qquad x \mapsto x^3$$

Now in the first case, since 3 does not divide the order of $\mathbb{F}_p^\times$, φ is bijective, and so an isomorphism. This means that each element of $\mathbb{F}_p$ has exactly one cube root, since $0^3 = 0$ in any field. For example, in $\mathbb{F}_{11}$, we have:

$$\begin{aligned} &0^3 = 0, \quad 1^3 = 1, \quad 7^3 = 2, \quad 9^3 = 3, \quad 5^3 = 4, \quad 3^3 = 5, \\ &8^3 = 6, \quad 6^3 = 7, \quad 2^3 = 8, \quad 4^3 = 9, \quad 10^3 = 10. \end{aligned}$$

This implies that every solution of $x + y + z = 0$ gives exactly one solution of $x^3 + y^3 + z^3 = 0$ by taking the cube root of each of x, y, z. Since $x + y + z = 0$ is the equation of a line in the projective plane, it has exactly $p + 1$ solutions. This settles the first case.

Now to outline what happens in the second case.

- We start off again by examining the homomorphism φ, and construct three sets R, S and T that together make up $\mathbb{F}_p$
- By introducing a new notation, we then establish an initial expression for M_p
- Manipulate this expression

- Introduce two new objects, "cubic Gauss sums" and an as yet unknown polynomial, $f(t)$
- Calculate $f(t)$ and its discriminant
- Use this to update our expression for M_p to its final form
- Prove that the expression has the necessary properties

So after getting an initial expression for M_p, we basically update it a lot through many smaller arguments, eventually arriving at the one we need.

Since $p \equiv 1 \mod 3$, write $p = 3m + 1$. Since 3 divides the order of $\mathbb{F}_p^\times = p - 1 = 3m$, φ is a homomorphism that is neither injective nor surjective. For example, in $\mathbb{F}_{13}$, we have:

$$\begin{aligned} 0^3 &= 0 \\ 1^3 = 3^3 = 9^3 &= 1 \\ 7^3 = 8^3 = 11^3 &= 5 \\ 2^3 = 5^3 = 6^3 &= 8 \\ 4^3 = 10^3 = 12^3 &= 12 \end{aligned}$$

Now, we can introduce our sets R, S and T. Let

$$R = \{x^3 : x \in \mathbb{F}_p^\times\}$$

and let $S = sR$ and $T = s^2R$, where s is any element of $\mathbb{F}_p^\times$ not in R. We now have a partition of $\mathbb{F}_p$ as follows:

$$\mathbb{F}_p = \{0\} \cup R \cup S \cup T$$

Notice that $|R| = |S| = |T| = m$. Since $-1 = (-1)^3$, $-1 \in R$ and $R = -R$, which implies that $S = -S$ and $T = -T$. Now that we have our partition of $\mathbb{F}_p$, we can reason about M_p, the object of interest in this proof. First, though, we introduce a new notation. For $X, Y, Z \subset \mathbb{F}_p$:

$$[X, Y, Z] = |\{(x, y, z) \mid x \in X, \quad y \in Y, \quad z \in Z \quad \text{and} \quad x + y + z = 0\}|$$

For simplicity, we will sometimes write $[XYZ]$ instead of $[X, Y, Z]$. Note that this is different to writing homogenous coordinates in the projective plane!

The symbol [ ] has a number of useful properties, such as:

$$\begin{aligned} [XY(Z \cup W)] &= [XYZ] + [XYW] \quad \text{if } Z \cap W = \emptyset \\ [XYZ] &= [aX, aY, aZ] \quad \text{for } a \neq 0 \\ [XYZ] &= [XZY] = \dots \end{aligned}$$

These properties are important and we will use them later to manipulate our initial expression for M_p.

For the first property, observe that if $Z \cap W = \emptyset$, then let $[XYZ] = z$ and $[XYW] = w$. $[XY(Z \cup W)]$ will have z elements that lie in $X \times Y \times Z$, and w that lie in $X \times Y \times W$. Since $Z \cap W = \emptyset$, none of these elements will lie in both $X \times Y \times Z$ and $X \times Y \times W$, and so we see that the $z + w$ elements are exactly those that contribute to $[XY(Z \cup W)]$.

For the second, notice that by definition,

$$[aX, aY, aZ] = |\{(x, y, z) \mid x \in X, \quad y \in Y, \quad z \in Z \quad \text{and} \quad ax + ay + az = 0\}|.$$

The requirement $ax+ay+az = 0 \iff a(x+y+z) = 0$ and since $a \neq 0$, $x+y+z = 0$. The final property follows from the commutativity of addition and by definition of $[\,]$.

Now, if $xyz \neq 0$, then the number of ways of writing 0 as a sum of non-zero cubes is $[RRR]$, simply by the definitions of the symbol $[\,]$ and the set R. But each element of R has three distinct cube roots, and so there are $(3 \times 3 \times 3)[RRR] = 27[RRR]$ solutions of $x^3 + y^3 + z^3 = 0$ of the form (x, y, z) where $xyz \neq 0$. But since we are working in the projective plane, scalar multiples of the same point are in fact the same point, so we have to divide by $p - 1$ for all of the non-zero multiples in $\mathbb{F}_p$, which gives

$$\frac{27[RRR]}{p-1} = \frac{27[RRR]}{3m} = \frac{9[RRR]}{m}$$

projective solutions where $xyz \neq 0$.

Now, when one of x, y or z is zero, say x, neither y nor z can be zero, as either being zero would force the other to also be zero, and the solution $(0, 0, 0)$ is not allowed in the projective plane. Thus there are $p - 1$ choices for y, each of which gives three choices of z (by solving $y^3 = -x^3$ and remembering that each element has three cube roots), which means there are $3(p - 1)$ solutions when $x = 0$. The same goes for y and z being zero, so we have a total of $9(p - 1)$ solutions where $xyz = 0$. However, again, we must divide through by $p - 1$, so there are 9 projective solutions when $xyz = 0$. This covers all cases, and so we have established that

$$M_p = \frac{9[RRR]}{m} + 9 = 9\left(\frac{[RRR]}{m} + 1\right) \tag{4}$$

The rest of the proof now relies on modifying this expression until we have the stated result. First of all, we use some properties of $[\,]$ to simplify (4).

To begin with, recall that $\mathbb{F}_p = 0 \cup R \cup S \cup T$, a disjoint union. Also, notice that $[RR\mathbb{F}_p] = m^2$, since there are m choices for each element of R, say r_1 and r_2, and then let the element of $\mathbb{F}_p$ be $-r_1r_2$. Using these two facts, we see that

$$[RR\{0\}] + [RRR] + [RRS] + [RRT] = m^2 \tag{5}$$

now fix $s \in S$ and $t \in T$.

$$[RRS] = [sR, sR, sS] = [SST]$$

and likewise, $[RRT] = [TTS]$. Substituting these into (5), we have

$$[RR\{0\}] + [RRR] + [SST] + [TTS] = m^2 \tag{6}$$

Again using the disjoint union of $\mathbb{F}_p$ and $[\mathbb{F}_p TS] = m^2$ (for the same reasons as $[RR\mathbb{F}_p]$, we have

$$[\mathbb{F}_p TS] = [\{0\}TS] + [RTS] + [STS] + [TTS] = m^2 \tag{7}$$

Recall that $[STS] = [SST]$. Now notice two things, first, that $[\{0\}TS] = 0$, since $T = -T$, $S = -S$ and $T \cap S = \emptyset$. Secondly, $[RR\{0\}] = m$, since each element of R has exactly one inverse, also in R. Combine these observations with substracting (6) from (7) and we end up with

$$[RTS] = [RRR] + m$$

Now we can substitute this into our initial expression as follows:

$$\begin{aligned} M_p &= 9\left(\frac{[RRR]}{m} + 1\right) \\ &= \frac{9}{m}\left([RRR] + m\right) \\ &= \frac{9[RTS]}{m} \end{aligned}$$

This is the end of our initial manipulations. We now proceed to use *cubic Gauss sums* to find an expression for $[RTS]$. Let $\zeta = e^{2\pi i/p}$, the p-th root of unity, and define three complex numbers α_1, α_2 and α_3 as follows:

$$\alpha_1 = \sum_{r\in R} \zeta^r, \quad \alpha_2 = \sum_{s\in S} \zeta^s, \quad \alpha_3 = \sum_{t\in T} \zeta^t$$

These numbers are each a sum of m distinct p-th roots of unity, and they are the cubic Gauss sums. They also happen to be the three roots of a cubic polynomial $f(t)$, which we now find an expression for. Recall that a general cubic polynomial $P(t)$ in t with roots α, β and γ can be calculated as

$$P(t) = (t-\alpha)(t-\beta)(t-\gamma) = t^3 - (\alpha+\beta+\gamma)t^2 + (\alpha\beta + \alpha\gamma + \beta\gamma)t - \alpha\beta\gamma$$

For the coefficient of t^2, notice that

$$0 = \zeta^p - 1 = (\zeta - 1)(\zeta^{p-1} + \zeta^{p-2} + \cdots + \zeta + 1)$$

and since $\zeta - 1 \neq 0$, $(\zeta^{p-1} + \zeta^{p-2} + \cdots + \zeta + 1) = 0$. Therefore, $\alpha_1 + \alpha_2 + \alpha_3 = -1$, and the coefficient of $t^2 = 1$.

To find the other coefficients, multiply two of the a_is together, say α_2 and α_3. Then

$$\alpha_2\alpha_3 = \sum_{s\in S} \zeta^s \cdot \sum_{t\in T} \zeta^t = \sum_{s\in S, t\in T} \zeta^{s+t} = \sum_{x\in\mathbb{F}_p} N_x \zeta^x$$

where N_x is the number of pairs (s,t) with $s \in S$ and $t \in T$ such that $s + t = x$. For $r \in R$, notice that

$$N_x = [ST - x] = [rS, rT, -rx] = [S, T, -rx] = N_{rx}$$

which shows that N_x depends only on the coset R, S or T in which x lies. Therefore,

$$mN_x = [S, T, Rx] = \begin{cases} [STR] & \text{if } x \in R, \\ [STS] & \text{if } x \in S, \\ [STT] & \text{if } x \in T. \end{cases}$$

Define integers a, b and c by

$$[STR] = ma, \quad [STS] = mb, \quad [STT] = mc$$

Recall our current working expression for $M_p = \frac{9[RTS]}{m}$. Then

$$M_p = 9a$$

and

$$\alpha_2\alpha_3 = a\alpha_1 + b\alpha_2 + c\alpha_3$$

Similarly,

$$\alpha_1\alpha_3 = a\alpha_2 + b\alpha_3 + c\alpha_1$$
$$\alpha_1\alpha_2 = a\alpha_3 + b\alpha_1 + c\alpha_2$$

This allows us to calculate the coefficient of t in $f(t)$ as follows:

$$\alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_3 = (a+b+c)(\alpha_1 + \alpha_2 + \alpha_3) = -(a+b+c)$$

But

$$\begin{aligned} m(a+b+c) &= [STR] + [STS] + [STT] \\ &= [ST(R \cup S \cup T)] \\ &= [ST\mathbb{F}_p] - [ST\{0\}] \\ &= m^2 \end{aligned}$$

So we see that

$$\alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_3 = -m$$

Finally, we need to calculate $\alpha_1\alpha_2\alpha_3$. We do this by calculating the sum of squares of the α_i as follows:

$$\alpha_1^2 + \alpha_2^2 + \alpha_3^2 = (\alpha_1 + \alpha_2 + \alpha_3)^2 - 2(\alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_3) = 1 + 2m$$

Now we can write

$$\alpha_1(\alpha_2\alpha_3) = \alpha_1(a\alpha_1 + b\alpha_2 + c\alpha_3)$$
$$\alpha_2(\alpha_3\alpha_1) = \alpha_2(a\alpha_2 + b\alpha_3 + c\alpha_1)$$
$$\alpha_3(\alpha_1\alpha_2) = \alpha_3(a\alpha_3 + b\alpha_1 + c\alpha_2)$$

and sum these to get

$$\begin{aligned} 3\alpha_1\alpha_2\alpha_3 &= a(\alpha_1^2+\alpha_2^2+\alpha_3^2)+(b+c)(\alpha_1\alpha_2+\alpha_1\alpha_3+\alpha_2\alpha_3) \\ &= a(1+2m)+(b+c)(-m) = a+km \end{aligned}$$

where k is yet another new letter, defined as $k = 2a-b-c = 3a-m$. This is the last coefficient of $f(t)$ we need to calculate, and thus

$$f(t) = t^3+t^2-mt-\frac{a+km}{3}$$

We then update our expression for M_p once more, and have

$$M_p = 9a = 3(k+m) = 3k+p-1$$

Now let D_f be the discriminant of f. Using our formula for the $\alpha_i\alpha_j$, we can calculate a square root of D_f as

$$\begin{aligned} \sqrt{D_f} &= (\alpha_1-\alpha_2)(\alpha_1-\alpha_3)(\alpha_2-\alpha_3) \\ &= \alpha_2\alpha_3(\alpha_2-\alpha_3)+\alpha_3\alpha_1(\alpha_3-\alpha_1)+\alpha_1\alpha_2(\alpha_1-\alpha_2) \\ &= (a\alpha_1+b\alpha_2+c\alpha_3)(\alpha_2-\alpha_3)+(a\alpha_2+b\alpha_3+c\alpha_1)(\alpha_3-\alpha_1)+(a\alpha_3+b\alpha_1+c\alpha_2)(\alpha_1-\alpha_2) \\ &= (b-c)(\alpha_1^2+\alpha_2^2+\alpha_3^2-\alpha_1\alpha_2-\alpha_1\alpha_3-\alpha_2\alpha_3) \\ &= (b-c)(1+3m) \\ &= (b-c)p \end{aligned}$$

Now, we calculate $g(t)$, which is related to $f(t)$; while $f(t)$ has the α_i as its three roots, $g(t)$ has the β_i as its three roots, which are defined as follows:

$$\beta_i = 1+3\alpha_i \quad \text{for } i = 1,2,3$$

Then

$$\begin{aligned} \beta_1+\beta_2+\beta_3 &= 0 \\ \beta_1\beta_2+\beta_1\beta_3+\beta_2\beta_3 &= -3p \\ \beta_1\beta_2\beta_3 &= (3k-2)p \end{aligned}$$

Which gives $g(t)$ as

$$g(t) = t^3-3pt-(3k-2)p$$

Let $A = 2k-2$. Then we can, for the last time, update our expression for M_p:

$$M_p = 3k+p-1 = p+1+A$$

This is indeed the A referred to in the initial statement of the theorem; we now show that it has all the stated properties.

Let D_g be the discriminant of $g(t)$. Then by simply plugging the coefficients of g into the formula for the discriminant of a cubic, we have

$$D_g = -4(-3p)^3-27(Ap)^2 = 4\cdot 27p^3-27A^2p^2$$

Now, for distinct $i, j \in \{1, 2, 3\}$, notice that $\beta_i - \beta_j = 3(\alpha_i - \alpha_j)$, and therefore

$$D_g = 27^2 D_f$$

And thus

$$4 \cdot 27p^3 - 27A^2p^2 = D_g = 27^2 D_f = 27^2(b-c)^2p^2$$

Canceling $27p^2$,

$$4p = A^2 + 27B^2$$

with

$$B = b - c \quad \text{and} \quad A = 3k - 2$$

We now just need to verify that A is uniquely determined by the two conditions $4p = A^2 + 27B^2$ and $A \equiv 1 \mod 3$. For uniqueness, suppose $4p = A_*^2 + 27B_*^2$ is another representation. Then

$$\begin{aligned} 4p(B_*^2 - B^2) &= (A^2 + 27B^2)B_*^2 - (A_*^2 + 27B_*^2)B^2 \\ &= (AB_* + A_*B)(AB_* - A_*B) \end{aligned}$$

Since p divides the LHS, it must divide one of the factors on the RHS, say $p|(AB_* - A_*B)$. Now multiply the two representations of $4p$ together to get

$$16p^2 = A^2A_*^2 + 27B^2A_*^2 + 27B_*^2A^2 + 27B^2B_*^2$$

so that

$$16p^2 - (AA_* + 27BB_*)^2 = 27(AB_* - A_*B)^2$$

Since p divides $AB - AB$, we see that

$$16 - \left(\frac{AA_* + 27BB_*}{p}\right)^2 = 27\left(\frac{AB_* - A_*B}{p}\right)^2$$

Seeing as the LHS is no greater than 16 and the RHS is 27 times the square of an integer, we see that both sides actually equal zero. This implies that $AB_* - A_*B = 0$, so if we let

$$\lambda = \frac{A_*}{A} = \frac{B_*}{B}$$

Substituting into $A^2 + 27B^2 = 4p = A_*^2 + 27B_*^2$ gives $\lambda^2 = 1$, so $\lambda = \pm 1$. The assumption that $A = A_* = 1 \mod 3$ means that $\lambda = 1$, proving the uniqueness of A and completing the proof. ∎

We finish with an example. To calculate M_{73}, for example, we need to solve $4 \times 73 = A^2 + 27B^2$ for some integers A and B. Observe that $A = 7$ and $B = 9$ solve this equation, and also $A \equiv 1 \mod 3$ as required. Therefore $M_{73} = 73 + 7 + 1 = 81$.

2.3 Lenstra's factorisation algorithm

2.3.1 The modular exponentiation algorithm

Before we introduce Lenstra's elliptic curve factorisation algorithm, a prerequisite algorithm, the modular exponentiation algorithm. This, along with the euclidean algorithm, is one of the two dependencies that Lenstra's algorithm requires. Therefore, a python implementation of it is also included in appendix A.

Definition 2.1 *The modular exponentiation algorithm allows highly efficient calculation of expressions of the form $a^k \mod n$, even when the numbers involved are very large. To improve on the naïve method of repeatedly multiplying by a and reducing mod n, observe that, for $k = 500$,*

$$500 = 2^2 + 2^4 + 2^5 + 2^6 + 2^7 + 2^8$$

and therefore a^k may be written as

$$a^{500} = a^{2^2} \times a^{2^4} \times a^{2^5} \times a^{2^6} \times a^{2^7} \times a^{2^8}$$

these terms are much easier to calculate by successively squaring and reducing mod n. Let $A_0 = a$ and calculate

$$\begin{aligned} A_1 &\equiv A_0 \cdot A_0 \equiv a^2 \mod n \\ A_2 &\equiv A_1 \cdot A_1 \equiv a^4 \mod n \\ A_3 &\equiv A_2 \cdot A_2 \equiv a^8 \mod n \\ &\vdots \\ A_l &\equiv A_{l-1} \cdot A_{l-1} \equiv a^{2^l} \mod n \end{aligned}$$

and then calculate $a^{500} = A_2 \times A_4 \times A_5 \times A_6 \times A_7 \times A_8 \mod n$.

As the numbers involved grow larger, the time savings over the trivial method become enormous, as this is an $O(\log_2(k))$ algorithm, and it is this algorithm which we shall use in our arithmetic.

2.3.2 Fermat's little theorem

Fermat's little theorem is a useful result in number theory that can (sometimes) be used to prove that a number is composite.

Theorem 2.3 *If p is prime, then*

$$2^{p-1} \equiv 1 \mod p.$$

Thus if we calculate $2^{n-1} \mod n$ for some $n \in \mathbb{N}$ and find that it does not equal 1, then we know that n is composite The converse statement is not true, however, and so we see that $2^{n-1} \equiv 1 \mod n$ is a necessary but not sufficient condition for n to be prime.

2.3.3 Lenstra's algorithm

Now we are ready to introduce Lenstra's algorithm. First published in [7], the algorithm is essentially an improvement on the older $p-1$ algorithm created by John Pollard [13, 10]:

Definition 2.2 *Lenstra's algorithm goes as follows to factor a composite integer N:*

- *Choose random integers b, x and $y \mod N$.*
- *Let $P = (x, y)$ and, for a given b, let $c = y^2 - x^3 - bx$ such that P is a point on the curve $C : Y^2 = X^3 + bX + c \mod N$. In this way, b determines the curve C.*
- *Compute kP for large k ($k = 10!$, for example).*
- *If the computation of kP is successful, increment b to get a new curve and recalculate kP.*
- *Continue until one of the additions fails.*

A python implementation of lenstra's algorithm is included in appendix A. Here, k simply refers to an exponent in $\mathbb{N}$, instead of the field associated with affine and projective space.

When calculating kP, we take a similar approach to the modular exponentiation algorithm, in that instead of calculating the k-fold sum $\underbrace{P + P + \ldots + P}_{k \text{ summands}}$, we calculate the binary expansion of k and write

$$k = a_0 2^0 + a_1 2^1 + \ldots + a_i 2^i + \ldots + a_n 2^n,$$

where $n \in \mathbb{N}$ and each a_i is either 0 or 1. It follows that

$$kP = a_0 2^0 P + a_1 2^1 P + \ldots + a_i 2^i P + \ldots + a_n 2^n P, \tag{8}$$

which we can calculate by repeatedly doubling P up to $2^n P$ and adding the points that have nonzero coefficients in equation (8).

For example, say we want to factor $N = 2638661449034729$. We can check that N is composite with fermat's little theorem, as mentioned earlier. Using the modular exponentiation algorithm, we calculate $2^{N-1} \mod N = 1386068300154542$, which proves that N is composite. Now we choose $x = 2$ and $y = 1$ such that our point $P = (2, 1)$ and let $b = 1$. Taking $k = 10!$, we attempt to calculate kP by doubling P and adding relevant points as described before. Since $10! = 2^8 + 2^9 + 2^{10} + 2^{11} + 2^{12} + 2^{14} + 2^{16} + 2^{17} + 2^{18} + 2^{20} + 2^{21}$, we double P up to $2^{21}P$ and then perform

$$kP = 10!P = 2^8P + 2^9P + 2^{10}P + 2^{11}P + 2^{12}P + 2^{14}P + 2^{16}P + 2^{17}P + 2^{18}P + 2^{20}P + 2^{21}P.$$

To aid legibility, from now on we write $2^k P = P_k$.

k	P_k
0	$(2, 1)$
1	$(1978996086776085, 1649163405646469)$
2	$(1354531004190202, 712566094192611)$
3	$(3034566536053, 2449122987212961)$
4	$(1301371522167812, 173584894036069)$
5	$(1996689877431974, 1560461237243599)$
6	$(1690032118719931, 1115183255675848)$
7	$(1106989140269887, 1249337210022185)$
8	$(2006000898376031, 739903289262451)$
9	$(360409978329708, 680989406049585)$
10	$(1498346384082037, 1311595726437562)$
11	$(317917266115269, 1139730538479759)$
12	$(1176599249124147, 834137694636709)$
13	$(229831052738857, 33111605954288)$
14	$(514283736870450, 404971411450613)$
15	$(1619865029688614, 1758310567012823)$
16	$(2493326453658282, 878913854327751)$
17	$(2243262160137209, 1701640423934925)$
18	$(2323526509553459, 1123682327919074)$
19	$(777963473649634, 1615413237260867)$
20	$(1420808458933872, 2034011761100882)$
21	$(2061892737190217, 240178323260750)$

Table 1: Generating the list of points via doubling when $b = 1$

To begin with, when $b = 1$, we double P as described, up to P_{21}. Table 1 shows the points generated by doubling. Now, we add up the relevant points to find kP:

$$\begin{aligned}
P_8 &= (2006000898376031, 739903289262451) \\
+P_9 &= (1435983113175084, 866053276340207) \\
+P_{10} &= (1372522607116506, 2279199412810534) \\
+P_{11} &= (2522377969419304, 1963228719476736) \\
+P_{12} &= (1867636445080926, 1800997235517595) \\
+P_{14} &= (396572835101474, 143329732066401) \\
+P_{16} &= (376321152472975, 2055982586016208) \\
+P_{17} &= (1086449006160587, 1492583392899450) \\
+P_{18} &= (804870318814085, 1030995341073004) \\
+P_{20} &= (1245577332479910, 2193527909976663) \\
+P_{21} &= (458028559825619, 2277727531737619)
\end{aligned}$$

Hence, we have successfully calulated

$$kP = (458028559825619, 2277727531737619)$$

What now? Clearly, if we stop here, we have learned nothing about the factorisation of N, so we increment b to give a new curve, and start again. In this particular example, it is possible to calculate kP in this way for $b = 2$, $b = 3$, ..., all the way up to $b = 23387$, where something changes. Table 2 again shows the list of points generated by doubling P, which is successful. However, when adding points to find kP, we find that

$$\begin{aligned}
P_8 &= (514795954556322, 1072900592237302) \\
+P_9 &= (1594423138470272, 757463091827773) \\
+P_{10} &= (2479979706264660, 2333023450224930) \\
+P_{11} &= (2288850814199939, 1125653152307693) \\
+P_{12} &= (406974087705183, 2180109173615353) \\
+P_{14} &= (2517955941215941, 2377002053335673) \\
+P_{16} &= (871996642729558, 279643526813096) \\
+P_{17} &= (540192081160502, 2103753450326003) \\
+P_{18} &= (1950391654929818, 1926026636369457) \\
+P_{20} &= (914230967480337, 1578752046703807)
\end{aligned}$$

This all calculates without issue, as before. When attempting the final addition

$$(914230967480337, 1578752046703807) + P_{21}$$

we need to calculate the inverse of the difference in x-coordinates, as in the addition formulae. The difference in x-coordinates is given by

$$1747536919598074 - 914230967480337 = 833305952117737$$

We then attempt to find the inverse of this with respect to N via the extended euclidean algorithm. In this instance, though, we see that

$$\gcd(833305952117737, N) = 33750191 \tag{9}$$

and the addition law breaks down. As a result, from equation (9) we have found that 33750191 is a non-trivial factor of N, and the algorithm is terminated. Dividing shows that $N = 33750191 \times 78182119$, and furthermore, these numbers are both prime, so this is the full factorisation of N.

k	P_k
0	$(2, 1)$
1	$(1978996223654343, 2307227360301809)$
2	$(2065341991305728, 2602387696447713)$
3	$(706090194730739, 364131284052804)$
4	$(977388443992499, 2229627649103617)$
5	$(1935153559561631, 2093139597695517)$
6	$(175553192385993, 2422801076771166)$
7	$(900832477635742, 1826551381265349)$
8	$(514795954556322, 1072900592237302)$
9	$(2067644682928109, 2124286622646949)$
10	$(2301960260745936, 2560653520820230)$
11	$(2512378902623323, 2537364249709183)$
12	$(2441105821836409, 1109262897760599)$
13	$(1236261064186840, 2217234011447721)$
14	$(384156417337880, 1320270218117776)$
15	$(1306192093717332, 688349719246224)$
16	$(2062019132081890, 832733545528307)$
17	$(1656673193668733, 916976917613322)$
18	$(17008362074755, 1666632592566998)$
19	$(1506916095423602, 1182578326715926)$
20	$(850331238187406, 2319495258218643)$
21	$(1747536919598074, 1619293033009998)$

Table 2: Generating the list of points via doubling when $b = 23387$

3 Discussion

3.1 Example RSA decryption

Now that Lenstra's algorithm has been introduced, we can see a slightly contrived example of a typical use case. Those unfamiliar with RSA encryption can refer to [9] for a complete description of the system. In brief, RSA is a public-key cryptosystem which relies on the difficulty of factoring large integers for its security. Messages are enciphered with a public-key of the form (e, N), where e is an exponent to raise a number to and N is a (preferably large) semiprime that acts as the modulus for encryption calculations. Suffice to say, if one can factor the modulus N, one can recover the private-key and factor any message enciphered with (e, N), as we demonstrate here.

Say we intercepted a message M along with the public key $P = (e, N)$. If we were able to factor N, we would be able to deduce the decryption key, and thus read this and any future messages.

Let's see how this would work with an example. Say we knew that

$$M = 3103229009940552729864, \quad P = (65537, 3160853182090460427047)$$

and we wanted to read M. If we apply Lenstra's algorithm to $N = 3160853182090460427047$, eventually, with $(3, 1)$ as our initial point, the algorithm finds that, with $b = 39850$, adding the two points

$$(2191801374392476491053, 2332211434379395076998) \text{ and}$$
$$(406058948051076877967, 315696859272760266 2096)$$

is impossible, since

$$2191801374392476491053 - 406058948051076877967 = 1785742426341399613086$$

and $\gcd(1785742426341399613086, 3160853182090460427047) = 3992747141$. Of course, we don't mind that we can't add the two points, since 3992747141 is a non-trivial factor of our modulus. A simple division then gives $N = 3992747141 \times 791648724667$. Because we know this is an RSA modulus, we know that this is the complete factorisation of N, but for completeness, a simple brute force check quickly shows that $p = 791648724667$ and $q = 3992747141$ are both prime.

Now that we know this, we calculate $(p-1) \times (q-1) = 3160853181294818955240$ and then find the multiplicative inverse of $e = 65537$ modulo 3160853181294818955240 with the euclidean algorithm. This turns out to be 146040609624497792033, and we finally decrypt our message by taking

$$M^{146040609624497792033} \mod 3160853182090460427047 = 2016$$

so we see that whatever nefarious organisation we are spying on is communicating the current year.

3.2 Elliptic curve cryptography

Elliptic curves over finite fields also appear in *elliptic curve cryptography*. This term refers to multiple schemes of public-key cryptography, all of which rely on elliptic curves over finite fields. Here, public-key cryptography refers to cryptography schemes like RSA, where the encryption key f can be made public, as long as the decryption key f^{-1} is kept private. This differs from classical private-key cryptosystems where the encryption key is either the same as the decryption key or else allows easy calculation of it, and therefore must be kept secret.

Common to all schemes is the difficulty of the *elliptic curve discrete logarithm problem*, in much the same way that schemes like RSA rely on the difficulty of factoring large integers.

Before we introduce the elliptic curve discrete logarithm problem comes the more general *discrete logarithm problem*. The discrete logarithm problem, or DLP, is stated as follows:

$$\text{Given } a, b \in \mathbb{F}_p, \text{ find } m \text{ such that } a^m \equiv b \mod p.$$

Clearly the requirement to work mod p is important, otherwise $m = \log_a b$ and the question is trivial.

The problem can be refactored in terms of elliptic curves by considering two distinct points P and Q on a curve E, and considering the following.

$$\text{Given } P, Q \in E(\mathbb{F}_p), \text{ find } m \text{ such that } mP = Q.$$

This is the elliptic curve discrete logarithm problem, or ECDLP. An important assumption is that such an m exists, as in general it will not.

Of course, given enough time it is possible to calculate $2P, 3P, \ldots$ until one finds that $nP = Q$ for some $n \in \mathbb{N}$, but as the size of the finite field grows larger this trivial method becomes impractical. Currently, the only approach to improve upon this naïve method of counting is by using a collision algorithm as detailed in [13], but this still takes approximately $\sqrt{p}$ steps to solve the ECDLP. There are faster methods in certain special cases of the ECDLP, but none which solve the problem for *any* given curve.

This sets elliptic curves aside from other groups, where the analogues of the DLP can be solved more quickly. For example, in $\mathbb{Z}_n$, one can use the extended euclidean algorithm as described before to solve

$$ma \equiv b \mod n$$

in at most $2 \log n$ steps, which allows efficient solutions for even large n. The important detail, then, is that the ECDLP is much harder than other versions of the DLP, and so this means that finite fields of smaller order may be used than in other systems. This lends itself well to any situation where minimising data storage is important, and is one of the main benefits of using elliptic curve cryptography.

A Code

```
"""
Implementation of lenstra's algorithm for factorisation in Python. Functions to
convert a number from base 10 to binary, to perform modular exponentiation, and
to apply the euclidean algorithm need to be defined first before a fourth
function to implement lenstra is made.

All code assumes curve is in short Weierstrass form, i.e. a = 0
"""

import math

# return the binary representation of an integer
def binary(integer):
    # compare input with increasing powers of two to generate list
    compare = 2
    powers = [1]
    while(compare <= integer):
        powers = powers + [compare]
        compare = compare * 2
    #print(powers)

    # decide which powers are in representation
    l = len(powers)
    representation = [0]*l

    for p in range(1,l+1):
        if powers[l-p]<=integer:
            integer = integer - powers[l-p]
            representation[l-p] = 1

    return representation

# perform modular exponentiation and return a^k mod n
def modex(a=1,k=1,n=1):
    bin = binary(k)

    # generate the Ai's

    A = [a]
    for i in range(0,len(bin)-1):
        x = A[i] * A[i]
        A = A + [x%n]
    # print A

    # perform calculation

```

```
47      ans = 1
48      #print range(0,len(bin))
49      for i in range(0,len(bin)):
50          if bin[i]==1:
51              #print A[i]
52              ans = ans * A[i]
53              ans = ans%n
54      ans = ans%n
55      return ans
56
57  # run quick compositeness test using fermat's little theorem
58  def primetest(N,diag=True):
59      fermat = modex(2,N-1,N)
60      if fermat == 1:
61          if diag:
62              print "Warning: input is either prime or pseudoprime."
63          return 0
64      else:
65          if diag:
66              print "Input is composite"
67          return 1
68
69  # perform the euclidean algorithm on two inputs a and b
70  def euclidean(a,b,prin=True,inv=False):
71      if a<b:# switch a and b if necessary, to ensure that a is larger
72          z = a
73          a = b
74          b = z
75      # save initial inputs
76      A = a
77      B = b
78
79      # initialise quotient and remainder and then perform algorithm
80
81      r = a%b
82      Q = [a/b]
83
84      while r!=0:
85          a = b
86          b = r
87          r = a%b
88          Q = Q + [a/b]
89      # print Q
90
91      # work out X and Y, the coefficients of the inputs when writing gcd
92      length = len(Q)
93      if length == 1:
94          X = 1
```

```
95          Y = Q[0]-1
96      elif length == 2:
97          X = 1
98          Y = Q[0]
99      else:
100         X = 1
101         Y = Q[length-2]
102         # print "X = %i" % (X)
103         # print "Y = %i" % (Y)
104         update = 0 # records whether to alter X or Y
105         for iter in range(1,length-1):
106             if update == 0:
107                 X = X + Q[length-2-iter] * Y
108                 update = 1
109                 # print "X = %i" % (X)
110                 # print "Y = %i" % (Y)
111             else:
112                 Y = Y + Q[length-2-iter] * X
113                 update = 0
114                 # print "X = %i" % (X)
115                 # print "Y = %i" % (Y)
116
117     if prin:# print GCD as linear combination of inputs if prin=True
118         if length <= 2:
119             print "GCD = %i = %i * %i - %i * %i" % (b, X, A, Y, B)
120
121         elif length%2 == 0:
122             print "GCD = %i = %i * %i - %i * %i" % (b, X, A, Y, B)
123
124         else:
125             print "GCD = %i = %i * %i - %i * %i" % (b, X, B, Y, A)
126
127     # return inverse of b in Z_a. Not necessarily valid!
128     if inv:
129         if length <= 2:
130             return (b,(-1*Y)%A)
131
132         elif length%2 ==0:
133             return (b,(-1*Y)%A)
134
135         else:
136             return (b,X)
137
138     else:
139         return b
140
141 # double input point P modulo N, on C: Y2 = X3 + bX + c (mod N)
142 #
```

```
143 # when called within function, returns [success,value]
144 # so either [all fine, point] or [error, offending inverse]
145 def dub(P,N,b=1,func=False):
146     # first calculate lambda, = lamb
147     e = euclidean((2*P[1])%N,N,prin=False,inv=True)
148 
149     if e[0]!=1:# no need for corresponding else
150         if func:
151             return [1,e[0]]
152         return "Found %i as a factor" % (e[0])
153 
154     n = (3 * P[0]**2 + b)%N
155     d = e[1]%N# modulus may be redundant..
156     lamb = (d*n)%N
157 
158     # calculate new coords, x and y
159     x = (lamb**2 - (2*P[0]))%N
160     y = (-1*lamb*x - P[1] + lamb*P[0])%N
161 
162     if func:
163         return [0,(x,y)]
164     return (x,y)
165 
166 # adds two distinct points modulo N, on EC : Y2 = X3 + bX + c
167 # when called within function, returns [success,value]
168 # so either [all fine, point] or [error, offending inverse]
169 def add(P1,P2,N,func=False):
170     #calculate lambda, = lamb
171     e = euclidean((P2[0]-P1[0])%N,N,prin=False,inv=True)
172 
173     if e[0]!=1:# no need for corresponding else
174         if func:
175             return [1,e[0]]
176         return "Found %i as a factor" % (e[0])
177 
178     n = P2[1]-P1[1]
179     d = e[1]%N# modulus may be redundant..
180     lamb = (d*n)%N
181 
182     # calculate new coords, x and y
183     x = (lamb**2 -P1[0]-P2[0])%N
184     y = (lamb*(P1[0] - x) - P1[1])%N
185 
186     if func:
187         return [0,(x,y)]
188     return(x,y)
189 
190 # use trial division on input to return a factor or "prime"
```

```
191 def brute(N):
192     if N % 2 == 0:
193         return "2 is a factor"
194     for x in range(3,int(N**0.5)+1,2):
195         if N % x == 0:
196             return "%i is a factor" % (x)
197     return "Prime"
198 
199 # MAIN FUNCTION
200 
201 # perform lenstra's algorithm on an input
202 def lenstra(N,K=20,b=False,X=2,Y=1):
203     # if b specified, execute only b
204     if b:
205         # initialise stuff
206         k = 10*9*8*7*6*5*4*3*2
207         bin = binary(k)
208         length = len(bin)
209 
210         # generate list of points via doubling
211         points = [(X,Y)]
212         for e in range(0,length-1):
213             new = dub(points[e],N,b,func=True)
214             if new[0]!=0:
215                 print points
216                 return "Found %i as a factor on curve with b = %i" % (new[1],b)
217 
218             points = points + [new[1]]
219         print points
220         print "\n"
221 
222         # compute kP
223         # find first point in binary representation
224         for x in range(0,length+1):
225             if bin[x] ==1:
226                 start = x
227                 kP = points[start]
228                 print kP
229                 break
230 
231         # add relevant points to find kP
232         for x in range(start+1,length):
233             if bin[x] == 1:
234                 new = add(kP,points[x],N,func=True)
235                 if new[0]!=0:
236                     print kP
237                     print points[x]
238                     return "Found %i as a factor on curve with b = %i" % (new[1],b)
```

```
239                 print new[1]
240                 kP = new[1]
241 
242         return "kP = (%i,%i)" % (kP[0],kP[1])
243 
244     # end of single-case
245 
246 
247 
248     # pre-start checks
249 
250     # run quick compositeness test using primetest
251     primetest(N)
252 
253     # test to make sure gcd(6,n) == 1
254     six = euclidean(N,6,prin=False)
255     if six != 1:
256         return "Found %i as a factor" % six
257 
258     # begin algorithm proper
259 
260     # initialise variables
261     k = 10*9*8*7*6*5*4*3*2
262 
263     bin = binary(k)
264 
265     length = len(bin)
266     b = 1
267 
268     while b<=50000:
269         # generate list of points via doubling
270         points = [(X,Y)]
271         for e in range(0,length-1):
272             new = dub(points[e],N,b,func=True)
273             if new[0]!=0:
274                 return "Found %i as a factor on curve with b = %i" % (new[1],b)
275 
276             points = points + [new[1]]
277 
278         # compute kP
279         # find first point in binary representation
280         for x in range(0,length+1):
281             if bin[x] ==1:
282                 start = x
283                 kP = points[start]
284                 break
285 
286         # add relevant points to find kP
```

```
287         for x in range(start+1,length):
288             if bin[x] == 1:
289                 new = add(kP,points[x],N,func=True)
290                 if new[0]!=0:
291                     return "Found %i as a factor on curve with b = %i" % (new[1],b)
292                 kP = new[1]
293 
294         b = b + 1
295 
296     return "Unable to find a factor :("
```

References

[1] R. Bix. *Conics and cubics: a concrete introduction to algebraic curves*. Springer Science & Business Media, 2006.

[2] P. Deligne. La conjecture de Weil. I. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 43(1):273–307, 1974.

[3] H. Hasse. Zur Theorie der abstrakten elliptischen Funktionenkörper I. Die Struktur der Gruppe der Divisorenklassen endlicher Ordnung. *Journal für die reine und angewandte Mathematik*, 175:55–62, 1936.

[4] H. Hasse. Zur Theorie der abstrakten elliptischen Funktionenkörper II. Automorphismen und Meromorphismen. Das Additionstheorem. *Journal für die reine und angewandte Mathematik*, 175:69–88, 1936.

[5] H. Hasse. Zur Theorie der abstrakten elliptischen Funktionenkörper III. Die Struktur des Meromorphismenrings. Die Riemannsche Vermutung. *Journal für die reine und angewandte Mathematik*, 175:193–208, 1936.

[6] N. Laustsen. Math111: Numbers and relations. Lancaster University Lecture Notes, October 2012.

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

[8] N. Mazza. Math323: Elliptic curves. Lancaster University Lecture Notes, October 2014.

[9] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[10] J. M. Pollard. Theorems on factorization and primality testing. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528. Cambridge Univ Press, 1974.

[11] R. Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

[12] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Springer, 2nd edition, 2009.

[13] J. H. Silverman and J. T. Tate. *Rational Points on Elliptic Curves*. Springer Science & Business Media, 2013.

[14] A. Weil. *Sur les courbes algébriques et les variétés qui s' en déduisent*. Number 1041 in Actualités scientifiques et industrielles. Hermann, 1948.

[15] D. Zeindler. Math322: Rings, fields and polynomials. Lancaster University Lecture Notes, October 2014.