Mathematics of Bitcoin: The ECDSA

by

Lewis Combes

MA4K8 Scholarly Report

Submitted to The University of Warwick

Mathematics Institute

April, 2018

Contents

1 Introduction

Bitcoin is a digital currency launched in 2009 by a person or persons assuming the name Satoshi Nakamoto [4]. Since its launch, Bitcoin has been steadily growing in value, and along with this has come an inevitable amount of media attention and a slew of rival digital coins (called *cryptocurrencies*). Bitcoin was the first of these cryptocurrencies, and was also the first digital currency to be totally decentralised. This means there is no presiding authority that verifies Bitcoin transactions, as with online banking. Instead, transactions are tracked via the *blockchain*, a copy of the list of who owns which coins kept by all users. When a user wants to transfer ownership of their coins to someone else, they send out a message to the userbase to update their copies of the blockchain, thus eliminating the need for a central authority to maintain the network.

The main issue in this situation is how exactly a user proves they own as many coins as they claim to. To prevent users from fraudulently claiming ownership over vast swathes of the currency, a *digital signature algorithm* is implemented in the transaction process. A digital signature algorithm (or DSA) is an example of public key cryptography, in which a user can trasmit private information using only publicly available information. In brief, a user has a *public key* and a *private key*—with the former being common knowledge (even to attackers), and the latter kept secret. From their public and private keys they can generate a signature, which is then checked by other users against their public key in order to verify it.

The hard part of this process, what makes it a mathematically interesting question to study, is how to design it in such a way that a user's private key cannot be reverse engineered from their public information. The solution designed for the original digital signature algorithm was to use arithmetic in the integers, modulo some prime p. In this setting, the *discrete logarithm problem* is difficult to solve, making it a useful basis for a public key encryption algorithm. The discrete logarithm problem for $(\mathbb{Z}/p\mathbb{Z})^\times$ is, given two integers x and y and p prime, to find $k \in \mathbb{Z}$ such that

$$x^k \equiv y \pmod{p} \tag{1}$$

The discrete log problem has the dual advantage of being hard to solve and easy to verify. That is, if you are given x and y and asked to find k, it is believed to be hard to find k quickly. Conversely, given x and k, it is very easy to check whether $x^k \equiv y \pmod{p}$. Note, we say the discrete log problem is *believed* to be difficult; currently there is no fast algorithm for computing them, but there is also no proof that there is no such algorithm.

Bitcoin uses a variant of the DSA to sign its transactions, called the elliptic curve digital signature algorithm (or ECDSA). The underlying concept is the same: instead of calculating discrete logs in the integers modulo p, it uses discrete logs in a finite group arising from an algebraic object called an elliptic curve.

The ECDSA has several advantages over the regular DSA, as well as several potential problems. For example, the ECDSA boasts a theoretical doubling of security over the regular DSA for private keys of the same length. This means the implementation can be made as efficient as the DSA but with much less memory usage. However, in order for the encryption to remain secure, an ECDSA deisgner must choose the elliptic curve over which the problem takes place very carefully. A poorly chosen curve can weaken the discrete logarithm problem, and in turn make the encryption much easier to break.

In this project I will discuss the mathematics that underpins the elliptic curve digital signature algorithm, and why it is important to understand it from a cryptographic perspective. Elliptic curve cryptography has something of "playing with fire" aspect to it: the technology can be hugely beneficial if used properly, but carelessness and ignorance can lead to disaster.

If the benefits of elliptic curve cryptography are to be reaped, then, it is essential for the implementor to understand the mathematics behind the technology. There are many standards used in elliptic curve cryptography, designed by various parties with various motivations, sometimes less-than wholesome ones. In order to know whether to trust these standards, it is important to understand how the elliptic curves they use are chosen, and what weaknesses they might have. To to this, one needs a decent understanding of what an elliptic curve actually is, and how it relates to cryptography.

2 Elliptic Curve Algebra

2.1 Projective Space

We start with a brief look at projective space, as it aids in understanding certain otherwise-baffling parts of the theory of elliptic curves.

Definition 2.1. *Let K be a field. We define an equivalence relation on the points $(x_0, x_1, ..., x_n) \in K^{n+1}$ by $(x_0, x_1, ..., x_n) \sim (y_0, y_1, ..., y_n)$ if and only if there is a non-zero number $\lambda \in K$ such that $(x_0, x_1, ..., x_n) = (\lambda y_0, \lambda y_1, ..., \lambda y_n)$. Then the n-dimensional projective space over K is*

$$\mathbb{P}^n(K) = (K^{n+1} \setminus \{(0, 0, ..., 0)\}) / \sim$$

Points in projective space are denoted $[x_0 : x_1 : ... : x_n]$.

Given any polynomial $f \in K[x_1, ..., x_n]$, we can obtain a homogeneous polynomial F over the associated projective space $\mathbb{P}^n(K)$ by setting

$$F(x_1, ...x_n, z) := z^{deg(f)} f\left(\frac{x_1}{z}, ..., \frac{x_n}{z}\right)$$

This process is called *homogenisation*, and can be reversed by setting $z = 1$.

Our main motivation for looking at polynomials over projective space will be to study their solution sets, i.e. those points where the polynomials vanishes. This, in turn, gives us information about the various polynomials in affine space we obtain by dehomogonising our projective curve at one of its variables.

The points on a projective cubic curve can be endowed with a group structure, but only when the curve is *smooth*. Informally, this just means the curve has no sharp turns or self-intersections. We formalise this notion using the partial derivatives of a polynomial.

Definition 2.2. *Given a polynomial F over $\mathbb{P}^n(K)$, its associated curve C given by the set $\{[x_0 : ... : x_n] \mid F([x_0 : ... : x_n]) = 0\}$ is called smooth if, for every point $[x_0 : x_1 : ... : x_n]$ that lies on C, we have that*

$$\frac{\partial F}{\partial x_i}([x_0 : x_1 : ... : x_n]) \neq 0$$

for each x_i. We also call the curve non-singular in this case. We call a point where all the partial derivatives of F vanish a singular point of C.

Note, for fields like $\mathbb{F}_p$ where the notion of limits doesn't make much sense, we just take the formal derivative of the polynomial F. With this definition in hand, we can define an elliptic curve.

Definition 2.3. *An elliptic curve E over a field K is a projective smooth curve given by the zero set of a polynomial*

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3$$

Equally, it can be thought of as all points $[X : Y : Z] \in \mathbb{P}^2(K)$ where

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

In either case, $a_1, a_2, a_3, a_4, a_6 \in K$, and this representation is called the long Weirstrass form of E.

This defines elliptic curves in their fullest generality. However, our purposes will be mostly cryptographic, and the vast majority of elliptic curves in use today are defined over a field whose characterisatic is neither 2 nor 3. This is helpful, because it allows us to simplify the representation of an elliptic curves a great deal.

Proposition 2.4. *Let E be an elliptic curve over a field K of characteristic neither 2 nor 3. Then E is the zero set of a polynomial*

$$F(X,Y,Z) = Y^2Z - X^3 - AXZ^2 - BZ^3$$

for some $A, B \in K$. This representation is called the short Weirstrass form of E.

Proof. Take a polynomial F as in Definition 2.3.

If our field does not have characteristic 2, we can complete the square with respect to Y, giving a polynomial of the form

$$F(X',Y',Z') = Y'^2Z' - X'^3 - b_2X'^2Z' - b_4X'Z'^2 - b_6Z'^3$$

Our field is not characteristic 3 either, so we can use the substitution $X' \mapsto X' - \frac{b_2Z}{3}$ to depress the polynomial, removing its X'^2Z' term. This leaves us with a polynomial of exactly the form required. $\square$

From here on, we will ignore curves that need to be defined in long Weirstrass form—that is, curves over fields of characteristic 2 or 3. While this makes our account a little less general, the details of how to extend the algebraic structure we will impose on E to curves over these fields are needless and will only slow us down, especially given their lack of use in real-world contexts. For a more complete account, see [1], page 47.

Once we have an elliptic curve given by $Y^2Z = X^3 + AXZ^2 + BZ^3$, it is only natural to ask what it looks like. For this, we have to dehomogenise the associated polynomial and look at it over the 2-dimensional affine space, K^2.

It is much easier to check whether a plane cubic curve is an elliptic curve or not when if we can write it in short Weirstrass form. If we have $y^2 = x^3 + Ax + B$ and we want to check whether it has any singular points, we only need that the discriminant of the polynomial $x^3 + Ax + B$ is non-zero. A zero discriminant means the derivative of $x^3 + Ax + B$ shares a root with $x^3 + Ax + B$ itself, so this root is a reapeated root of $x^3 + Ax + B$. Then at this point, all the partial derivatives will be zero due to the shared root.

Calculating the discriminant of $x^3 + Ax + B$, we see a curve in short Weirstrass form is an elliptic curve if and only if $4A^3 + 27B^2 \neq 0$. This is a much more useful criterion for testing whether we have an elliptic curve than calculating partial derivatives every time. It also extends to the homogenised version of the curve in projective space.

Example 1. *We take $F(X, Y, Z) = Y^2Z - X^3 + XZ^2 - Z^3$ and homogenise it by setting $Z = 1$. This gives the polynomial $f(x, y) = y^2 - x^3 + x - 1$, and the associated curve $C = \{(x, y) \in \mathbb{Q}^2 \colon y^2 = x^3 - x + 1\}$. Plotting these points on the x, y-plane gives:*

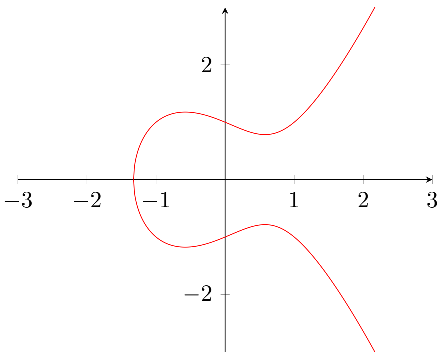

Figure 1: $y^2 = x^3 - x + 1$ over $\mathbb{R}$

This is as close as we can get to seeing the entire elliptic curve. This is not the whole picture, though. There *is* an inclusion of $\mathbb{Q}^2 \subset \mathbb{P}^2(\mathbb{Q})$ given by sending the point (x, y) to $[x : y : 1]$, but if we want to capture all the other points of an elliptic curve we will need to vary this final co-ordinate. If we replace it with any non-zero rational number z, we can see it will just be equivalent to the point $[\frac{x}{z} : \frac{y}{z} : 1]$, which we will have already captured by the inclusion above. So the only other points in $\mathbb{P}^2(\mathbb{Q})$ not given this way are those whose final co-ordinate is 0.

To find out how much we're missing, we simply set $Z = 0$ in $F(X, Y, Z) = Y^2Z - X^3 - AXZ^2 - BZ^3$, giving us $F(X, Y, 0) = -X^3$. If this point lies on the elliptic curve then F must vanish at this point, meaning $X = 0$. In projective space, we cannot have the point $[0 : 0 : 0]$, so the Y co-ordinate must be (up to scaling) 1.

This means when we look at the affine dehomogenisation of our curve, all we lose is the point $[0 : 1 : 0]$. This point is usually referred to as the *point at infinity* of the elliptic curve, a name it gets because of the way we obtain F from f. Recall, $F(X, Y, Z) = Z^3 f(\frac{X}{Z}, \frac{Y}{Z})$, and substituting $Z = 0$ would require us to evaluate f at infinity in both arguments. From here we will denote this point simply by ∞.

2.2 The Group Law

When we define the group structure on our elliptic curve, this point at infinity will serve as the identity element. But how do we get a group from the points of an elliptic curve? The naive method would be to simply define addition of points component-wise, that is to say for $P = (x_0, y_0)$ and $Q = (x_1, y_1)$, define $P + Q = (x_0 + x_1, y_0 + y_1)$. Unfortunately this does not work, as we can see in the above example. The point $(1, 1)$ lies on the curve $y^2 = x^3 - x + 1$ (or equivlanetly, the point $[1 : 1 : 1]$ lies on the curve $Y^2Z = X^3 - XZ^2 + Z^3$), but the point $(1, 1)``+"(1, 1) = (2, 2)$ does not (nor $[2 : 2 : 1]$ in the projective case).

The method for adding points on an elliptic curve is a little more complicated, and is best explained in the affine case (with a few appeals to the projective setting in the detailled parts). To wit, we define the group law as follows:

Definition 2.5. *For a field K and an elliptic curve E, the group law on the set $E(K)$ is given as follows: for two points P and Q in $E(K)$, the point $P \oplus Q$ is found by drawing the line connecting P and Q, finding the third point R of intersection of this line with E, then drawing the vertical line through the point R and seeing where that line intersects the curve again.*

Or, pictorally:

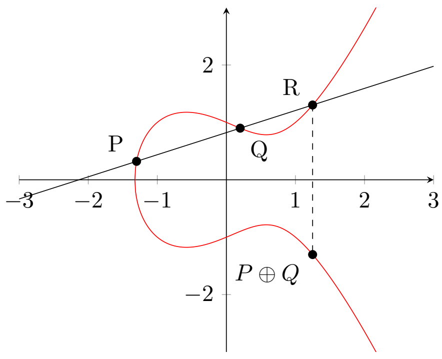

Figure 2: Elliptic Curve addition

There are a few things to note about this definition. First, the act of drawing the vertical line through R is actually masking the following process: transform the curve into its projective form and connect the image of R to the point at infinity $[0 : 1 : 0]$ by a straight line, find where this line connects the curve again, and dehomogenise. In most cases, this is simply achieved by reflecting through the x-axis, but it will become an important wrinkle when we come to prove this process indeed forms a group.

Second, "the line connecting P and Q" doesn't make much sense when $P = Q$ unless we allow for the drawing of the tangent at P in this case (this is in fact how we tackle this problem).

And third, if we draw a tangent at one of the points where our curve crosses the x-axis, we will get a vertical line. Once again, this is really the process of drawing the tangent at the image of P on the associated projective curve and finding where it again intersects the curve. This will be at the point at infinity only, giving $R = [0 : 1 : 0]$. Connecting R to the point at infinity gives a degenerate line: just the point $[0 : 1 : 0]$, which only intersects the curve again at itself. Thus these points P will all have order 2 in the group.

All that remains to show is that the group axioms actually hold.

Theorem 2.6. *The group law as described above endows $E(K)$ with the structure of an abelian group, with identity element* $[0 : 1 : 0]$.

Proof. For $E(K)$ to be an abelian group, it must satisfy the following for all $P, Q, R \in E(K)$:

1. (Closure) $P \oplus Q$ is in $E(K)$
2. (Commutativity) $P \oplus Q = Q \oplus P$
3. (Identity) $P \oplus [0 : 1 : 0] = P$
4. (Inverses) There is a $-P \in E(K)$ such that $P \oplus (-P) = [0 : 1 : 0]$
5. (Associativity) $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$

Closure: If we have points $P = (x_0, y_0) \neq Q = (x_1, y_1)$, we can find the point R as follows: writing the line between P and Q as $y = mx + c$ and substituting gives us that $(mx + c)^2 = x^3 + Ax + B$, or $x^3 - m^2x^2 + (A - 2cm)x + (B - c^2) = 0$. Normally factoring a cubic polynomial is non-trivial, but we already know two roots of this function, namely x_0 and x_1, so writing the final root as x_2 we have that

$$x^3 - m^2x^2 + (A - 2cm)x + (B - c^2) = (x - x_0)(x - x_1)(x - x_2)$$

Expanding the right hand side and equating coefficients gives that $m^2 = x_0 + x_1 + x_2$, and as m was obtained via field operations on elements in K, we get that m^2 is in K and so $x_2 \in K$, again by the closure of our field K. Then as $R = (x_2, y_2)$ lies on the line $y = mx + c$, we get that $y_2 = mx_2 + c$, so $y_2 \in K$. Thus R is in $E(K)$. In the case $P = Q$ we have the explanation as above that $P + P = [0 : 1 : 0]$ in projective space.

Commutativity: All we need is to notice is that the line connecting P and Q is exactly the same as the line connecting Q and P, so they give rise to the same R.

Identity: In projective space, this is $\infty = [0 : 1 : 0]$. If we imagine the line connecting P and ∞ intersecting E at a third point R, and then drawing the line connecting R and ∞, we will see that the two lines are the same, and so the line connecting R and ∞ intersects the curve again at P, giving $P \oplus \infty = P$.

Inverses: if $P = (x_0, y_0)$, we define $-P = (x_0, -y_0)$. The line connecting these two points is the vertical line through P (equivalently, the vertical line through $-P$), which as we have seen before intersects the curve again only in the projective case, at the point ∞. Connecting ∞ to itself once again yields the degenerate line that is simply the point ∞, which intersects E again only at itself.

Associativity: We omit this proof, owing to its cumbersome nature and irrelevance to understanding elliptic curve cryptography. See [1], pages 20-32 for a full proof. $\square$

From here on we will refer to the group $(E(K), \oplus)$ simply as $E(K)$ for the sake of brevity.

Calculating sums of points by finding secants every time can get a bit tiresome. Instead, by breaking it down into specific cases and running through the steps, we can write a fairly concise set of algebraic formulae. We take these from [1], page 14.

Group Law Formulae

Let E be an elliptic curve defined by $y^2 = x^3 + Ax + B$, and take two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on E. Then the following formula give the point $P_3 := P_1 \oplus P_2 = (x_3, y_3)$.

1. If $x_1 \neq x_2$ then

$$x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1, \qquad \text{where } m = \frac{y_2 - y_1}{x_2 - x_1}$$

2. If $x_1 = x_2$ but $y_1 \neq y_2$ then $P_1 \oplus P_2 = \infty$.
3. If $P_1 = P_2$ then

$$x_3 = m^2 - 2x_1, \qquad y_3 = m(x_1 - x_3) - y_1, \qquad \text{where } m = \frac{3x_1^2 + A}{2y_1}$$

4. If $P_1 = P_2$ and $y_1 = 0$ then $P_1 \oplus P_2 = \infty$.

An important operation on elliptic curves is taking a scalar multiple of a point P—that is, for a given integer n, we take the point P and add together n copies of P. We denote this nP. One simple method of calculating nP is the *double-and-add* method, which goes as follows:

We take our number n and write its binary expansion: $n = a_0 + 2a_1 + 2^2a_2 + ... + 2^n a_n$ with $a_i \in \{0, 1\}$. Then we use successive doubling to get $2P$, $4P$, ..., 2^nP, and then add up the appropriate terms. For small scalar multiplication, this doesn't save much on the computation of just adding up the point over and over again. However, the double-and-add method runs in time roughly log of the size of n, so for large enough n it becomes more efficient than straight repeated addition.

2.3 Endomorphisms and the Structure of $E(K)$

Now that we know $E(K)$ is a group, we can construct its endomorphism ring $\mathrm{End}(E)$. We define $(\Psi + \Phi)(P) = \Psi(P) \oplus \Phi(P)$ and $(\Psi \circ \Phi)(P) = \Psi(\Phi(P))$. Endomorphisms of elliptic curves rational functions that map the identity to itself: that is, $\Psi \in \mathrm{End}(E)$ acts on points $P = (x, y)$ by $\Psi(x, y) = (R_1(x, y), R_2(x, y))$ for some raional functions R_1 and R_2, and satisfies $\Psi(\infty) = \infty$.

As E comes equipped with the algebraic relation $y^2 = x^3 + Ax + B$, it is possible to eliminate every appearance of y^2 in both R_1 and R_2. This can also be used to remove every power of y in the denominators of these functions, meaning we can write both of them as $R_1(x, y) = r_1(x)+s_1(x)y$ and $R_2(x, y) = s_2(x)+r_2(x)y$ for some rational functions r_i, s_i.

Finally, as Ψ is a homomorphism, it preserves the group law on E. So,

$$(R_1(x, -y), R_2(x, -y)) = \Psi(x, -y) = -\Psi(x, y) = (R_1(x, y), -R_2(x, y))$$

This means that $r_1(x) - s_1(x)y = r_1(x) + s_1(x)y$, giving $s_1 \equiv 0$. Also, $s_2(x) + r_2(x)y = -s_2(x) + r_2(x)y$, giving $s_2 \equiv 0$. This means any rational function on E can be written as

$$\Psi(x, y) = (r_1(x), r_2(x)y)$$

for r_i rational functions of x alone. This argument closely follows [1], pages 50-51.

Every map in $\mathrm{End}(E)$ can be written as a pair of rational functions in this way. Conversely, a pair of rational functions of this form is an endomorphism precisely when it maps ∞ to ∞. This might seem a tough ask for functions defined over the affine plane, but homogenising them and looking to see where the point $[0 : 1 : 0]$ goes under the new map tests exactly this criterion.

Perhaps the most obvious example of an endomorphism of an elliptic curve is a scalar multiplication map. For an integer m we denote this map $[m]$. When $m > 0$, this is just the regular addition of m copies of P. When $m < 0$, we add up m copies of the point $-P$.

The endomorphisms $[m]$ are useful because they allow us to count the m-torsion points on our elliptic curve. This, in turn, tells us about the overall structure of the curve over a finite field. However, to do this we need the notion of the degree of an endomorphism:

Definition 2.7. *Let Ψ be an endomorphism of an elliptic curve E. Writing $\Psi(x,y) = (p(x)/q(x), r_2(x)y)$, we define the degree of Ψ (denoted $\deg(\Psi)$) to be the maximum of the degrees of $p(x)$ and $q(x)$ as polynomials.*

Theorem 2.8 (The Parallelogram Identity)**.** *Let E be an elliptic curve over a field K. Then the degree map $\deg : \mathrm{End}(E) \to \mathbb{N}$ satisfies the following for any $\Psi, \Phi \in \mathrm{End}(E)$:*

$$\deg(\Psi + \Phi) + \deg(\Psi - \Phi) = 2\deg(\Psi) + 2\deg(\Phi)$$

See [22], page 60, for a proof. Using this, we can prove the following:

Proposition 2.9. *Let E be an elliptic curve. Then $\deg([m]) = m^2$.*

Proof. (From [22], page 62)

We prove this by induction on m.

$m = 1$ is easy, as $[1]P = P = (x, y)$, which has degree 1 as $r_1(x) = x$.

We assume the result for $m > 1$, and proceed using the parallelogram law:

$$\begin{aligned}
\deg([m] + [1]) &= 2\deg([m]) + 2\deg([1]) - \deg([m] - [1]) \\
\deg([m + 1]) &= 2\deg([m]) + 2\deg([1]) - \deg([m - 1]) \\
&= 2m^2 + 2 - (m - 1)^2 \\
&= m^2 + 2m + 1 = (m + 1)^2
\end{aligned}$$

completing the induction. □

An important concept in the study of endomorphisms of elliptic curves is the notion of separability. We will see that we can relate the size of the kernel of a separable endomorphism to its degree:

Definition 2.10. *A polynomial $g(x) \in K[x]$ is called separable if its derivative is not identically zero. An endomorphism Ψ of an elliptic curve E over K is called separable if, writing $\Psi(x,y) = (p(x)/q(x), r_2(x)y)$, at least one of $p(x)$ and $q(x)$ is separable as a polynomial in $K[x]$.*

Theorem 2.11. *Let E be an elliptic curve over K, and let Ψ be an endomorphism of E. If Ψ is separable then $\#\ker(\Psi) = \deg(\Psi)$. Otherwise, $\#ker(\Psi) < \deg(\Psi)$.*

For a proof, see [1], page 54.

Over a field of characteristic zero, all endomorphisms are separable. Over a field of characteristic p, the inseparable endomorphisms are exactly with either the numerator or denominator of r_1 equal to a polynomial of the form $g(x^{p^k})$ for some $g \in K[x]$, $k \geq 1$. This is because these are the only non-constant polynomials whose derivative is identically zero.

Whether $[n]$ is separable or not, it's fairly easy to see that the number of n-torsion points on an elliptic curve is at most n^2. This is because the n-torsion points are exactly those $P \in \ker([n])$, and the size of this set is at most n^2 by the previous theorem.

Using this fact, we can derive the structure of the group $E(\mathbb{F}_q)$. Understanding the structure of this group is useful both in designing and trying to break elliptic curve cryptography, as all standards in the field use an elliptic curve defined over $\mathbb{F}_q$. Thus, it is useful to know the following theorem:

Theorem 2.12. *Let E be an elliptic curve over $\mathbb{F}_q$. Then*

$$E(\mathbb{F}_q) \cong (\mathbb{Z}/n_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z})$$

where $n_1 \mid n_2$.

Proof. There are only finitely many points in $\mathbb{F}_q^2$, and $E(\mathbb{F}_q) \subseteq \mathbb{F}_q^2$, so the group is finite. Then the Fundamental Theorem of Finitely Generated Abelian Groups gives

$$E(\mathbb{F}_q) \cong (\mathbb{Z}/n_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z}) \times ... \times (\mathbb{Z}/n_r\mathbb{Z})$$

with $n_1 \mid n_2$, $n_2 \mid n_3$, ..., $n_{r-1} \mid n_r$.

In $\mathbb{Z}/n_1\mathbb{Z}$, every element is n_1-torsion. As $n_1 \mid n_i$ for all $1 \leq i \leq r$, we also have n_1 points of n_1-torsion in each of the $\mathbb{Z}/n_i\mathbb{Z}$. Overall this gives n_1^r points of n_1-torsion.

But we have shown that there are at most n_1^2 points of n_1-torsion, so we get $n_1^r \leq n_2^2$, meaning $r \leq 2$. □

From the definition of addition and multiplication of endomorphisms, it's easy to see that the map $\mathbb{Z} \to \text{End}(E)$ given by $m \mapsto [m]$ is a ring homomorphism. Thus, we can think of the endomorphism ring of any elliptic curve as containing a copy of the integers. For some elliptic curves, this is the entire story. For others, though, we can find additional endomorphisms.

For elliptic curves over $\mathbb{F}_q$, we can define the map $\phi_q(x,y) := (x^q, y^q)$. By Lagrange's theorem, $k^{q-1} = 1$ for all $k \in \mathbb{F}_q^\times$, so $\phi_q(x,y) = (x,y)$ for all $(x,y) \in \mathbb{F}_q^2$. Thus ϕ_q is an endomorphism of any elliptic curve $E(\mathbb{F}_q)$, and we call it the Frobenius endomorphism.

The Frobenius endomorphism is inseparable, as the derivative of its x co-ordinate is $qx^{q-1} = 0$, since $q = 0$ in $\mathbb{F}_q$. From this it follows that ϕ_q^n is inseparable over $\mathbb{F}_q$ for all $n \geq 1$.

However, the endomophism $\phi_q^n - 1$ is always separable (for details see [1],[5]). This is useful, because it maps every point of $E(\mathbb{F}_{q^n}) \subset E(\overline{\mathbb{F}_q})$ to ∞ (and *only* the points of $E(\mathbb{F}_{q^n})$), so we get that $\#E(\mathbb{F}_{q^n}) = \ker(\phi_q^n - 1) = \deg(\phi_q^n - 1)$. Thus, knowing the degree of $\phi_q^n - 1$ allows us to count the number of points on $E(\mathbb{F}_q)$.

We denote by t the number $q + 1 - \#E(\mathbb{F}_q)$. It can be shown that the Frobenius endomorphism satisfies the polynomial equation $\phi^2 - t\phi + q = 0$—that is to say, the endomorphism obtained by composing and adding the endomorphisms ϕ, $[t]$ and $[q]$ in this way is the zero endomorphism.

Elliptic curves over finite fields always have an endomorphism ring larger than $\mathbb{Z}$. The two possible cases are (per [12]):

(i) $\text{End}(E) \cong \mathcal{O}_K$, where $K = \mathbb{Q}(\sqrt{D})$ for some $D < 0$.

(ii) $\text{End}(E) \cong \mathcal{O}$, where $\mathcal{O}$ is an order in a quaternion algebra. (For more information, see [1], page 318)

We call case (i) the complex-multiplication (or CM) case, and case (ii) the supersingular case. In the CM case, we can use group endomorphisms to speed up Pollard's Rho algorithm (see **§4.1**). We could theoretically do this in the supersingular case, but there are better attacks for curves of this type (see **§4.2.2**).

To find these endomorphisms in the CM case, the question becomes how to calculate D. We do this through the methods of algebraic number theory: $\mathcal{O}_{\mathbb{Q}(\sqrt{D})}$ is the ring of integral elements of the field $\mathbb{Q}(\sqrt{D})$—as such, every element in the ring satisfies a unique monic polynomial of degree at most 2. In particular, if $\text{End}(E) \cong \mathcal{O}_K$, ϕ_q satisfies $\phi^2 - t\phi + q = 0$ and so the image α of ϕ_q under this isomorphism satisfies $\alpha^2 - t\alpha + q = 0$. Solving this gives $\alpha = \frac{t \pm \sqrt{t^2 - 4q}}{2}$, so $\mathbb{Q}(\sqrt{t^2 - 4q}) = \mathbb{Q}(\alpha) \subseteq \mathbb{Q}(\sqrt{D})$. If $t^2 - 4q$ isn't a square of an integer (and in the CM case, it never is), we have a field extension of $\mathbb{Q}$ of degree 2, with a subfield also of degree 2. Thus the two fields must be equal, and so D is the squarefree part of $t^2 - 4q$.

This gives us a simple (though sometimes computationally taxing) way of calculating D. From there, it's a short calculation to find the ring of integers of $\mathbb{Q}(\sqrt{D})$ (see [24]):

$$\mathcal{O}_{\mathbb{Q}(\sqrt{D})} = \begin{cases} \mathbb{Z}[\sqrt{D}] & \text{if } D \not\equiv 1 \pmod 4 \\ \mathbb{Z}[\frac{1+\sqrt{D}}{2}] & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

One special type of endomorphism is an automorphism—an isomorphism of $E(K)$ to itself—which form a subgroup of $\mathrm{End}(E)$ denoted $\mathrm{Aut}(E)$. In order to find these, we need teh following:

Definition 2.13. *Let E be an elliptic curve over K in short Weirstrass form. We define the j-invariant $j(E)$ of E to be*

$$j(E) = 1728\frac{4A^3}{4A^3 + 27B^2}$$

Using this, we prove:

Theorem 2.14. *Let $E : y^2 = x^3 + Ax + B$ and $E' : y'^2 = x'^3 + A'x' + B'$ be elliptic curves over K. Then $E(\overline{K}) \cong E'(\overline{K})$ when $j(E) = j(E')$.*

Proof. (Proof adapted from [5], page 47)

First we assume $j(E) = j(E')$. Equating j's and rearranging gives $A^3B'^2 = A'^3B^2$. We look for maps of the form $(x, y) \mapsto (\mu^2 x, \mu^3 y)$.

When $A = 0$ we have $B \neq 0$ (as $\Delta_E \neq 0$). We also have $j(E') = j(E) = 0$, so $A' = 0$ also. This gives $B' \neq 0$, and we set $\tau(x, y) = (\mu^2 x, \mu^3 y)$, where $\mu = (B'/B)^{1/6}$.

When $B = 0$ we have $A \neq 0$. We also have $j(E) = j(E') = 1728$, giving $B' = 0$ also. Then $A' \neq 0$ and we set $\tau(x, y) = (\mu^2 x, \mu^3 y)$, where $\mu = (A'/A)^{1/4}$.

When $A, B \neq 0$ we have $A', B' \neq 0$ also. Here we take $\mu = (B'/B)^{1/6} = (A'/A)^{1/4}$.

Through simple (but tedious, and thus omitted) calculations, one can verify that $\tau(x, y) = (\mu^2 x, \mu^3 y)$ is a homomorphism of groups.

For injectivity, we note that τ acts on the projective version of the curve E by $\tau([X : Y : Z]) = [\mu^2 X : \mu^3 Y : Z]$. Then $\tau([X : Y : Z]) = [0 : 1 : 0]$ gives $X = Z = 0$ and $Y = 1/\mu^3$. So $\ker(\tau) = \{[0 : 1/\mu^3 : 0]\} = \{[0 : 1 : 0]\}$.

For surjectivity, we take (x', y') lying on E', i.e. $y'^2 = x'^3 + A'x' + B'$. Dividing by μ^6 we get $\left(\frac{y'}{\mu^3}\right)^2 = \left(\frac{x'}{\mu^2}\right)^3 + A\left(\frac{x'}{\mu^2}\right) + B$, meaning the point $\left(\frac{x'}{\mu^2}, \frac{y'}{\mu^3}\right)$ lies on E. Thus τ is surjective, and an isomorphism. □

This theorem holds in the other direction too: two ismorphic elliptic curves have the same j-invariant (see [5], page 45 for a proof). Then for a general elliptic curve, we have the following:

Proposition 2.15. *Let E be the elliptic curve $y^2 = x^3 + Ax + B$ over K. Then*

(i) If $A, B \neq 0$, we have $\mathrm{Aut}(E(K)) \cong \mathbb{Z}/2\mathbb{Z}$

(ii) If $A = 0$, we have $\mathrm{Aut}(E(\overline{K}) \cong \mathbb{Z}/6\mathbb{Z}$

(iii) If $B = 0$, we have $\mathrm{Aut}(E(\overline{K})) \cong \mathbb{Z}/4\mathbb{Z}$

Proof. An ismorphism of E to itself is given by a map $(x, y) \mapsto (\mu^2 x, \mu^3 y)$, where $A = \mu^4 A$ and $B = \mu^4 B$.

When $A, B \neq 0$, this reduces to $\mu^2 = 1$, giving two choices for $\mu \in \overline{K}$ (in fact, $\mu = \pm 1$). Then $|\mathrm{Aut}(E)| = 2$, and so is cyclic of order 2.

When $A = 0$, we must have $B \neq 0$, so all automorphisms of E are determined by $\mu^6 = 1$. This gives six choices, and choosing a $\mu \in \overline{K}$ such that $\mu^2, \mu^3 \neq 1$ we see that $\mathrm{Aut}(E)$ is cyclic of order 6.

When $B = 0$, we must have $A \neq 0$, so all automorphisms of E are determined by $\mu^4 = 1$. This gives four choices, and like before we can choose a $\mu \in \overline{K}$ such that $\mu^2 \neq 1$, giving us that $\mathrm{Aut}(E)$ is cyclic of order 4.

□

3 Elliptic Curve Cryptography

3.1 Discrete Logarithms in $E(K)$

Recall, the discrete logarithm problem for $\mathbb{Z}/p\mathbb{Z}$ is the task, given two integers x and y, of finding a $k \in \mathbb{Z}$ such that $x^k \equiv y \pmod{p}$. This problem can also be studied in the finite group $E(\mathbb{F}_q)$ for an elliptic curve E and finite field $\mathbb{F}_q$. Instead of powers of integers, we take multiples of points on the curve.

Like the discrete logarithm problem for integers, the elliptic curve discrete logarithm problem (or ECDLP) is assumed to be difficult to solve in general. Methods like Pollard's Rho algorithm (**§4.1**) can find discrete logs for any group G, but this runs in $O(\sqrt{\#G})$, which can be infeasible for large $\#G$. To date, nobody has found a sub-exponential time algorithm that solves this problem.

There are methods for breaking the discrete logarithm problem on some special elliptic curves, which take advantage of extra structure inherent in their associated groups. For example, curves defined over fields of characteristic 2 were considered a good cryptographic standard because their extra structure allowed for fast point arithmetic. However, the attack presented in [20] weakens the discrete logarithm problem on these curves significantly. Consequently, these "binary curves" have fallen out of favour in the cryptographic community, being mostly replaced by curves over fields of large prime order.

3.2 The ECDSA

The elliptic curve digital signatre algorithm (or ECDSA) is a cryptographic protocol that allows a user to digitally sign a file in such a way that it is easy to verify the signature but hard to reverse-engineer the information required to generate said signature. In this implementation, we will describe how one user Alice can use an elliptic curve over a prime field to provide a digital signature to another user, Bob. This version of the algorithm is adapted from [2], page 135.

The ECDSA

Suppose Alice wants to digitally sign a message m. She and Bob agree publicly on an elliptic curve E over a finite field and a base point P of order q. As the ECDSA is a form of public key encryption, Alice has to release public information. In this case, she chooses a random integer x in the interval $1 < x < q-1$ and releases the point $Q = xP$ pubicly.

To sign a message, Alice does the following:

1. She selects a random integer k in the interval $1 < k < q-1$.
2. She computes $kP = (x_1, y_1)$ and $r = x_1 \pmod{q}$. If $r = 0$, she returns to step 1.
3. She computes $s = k^{-1}(m + xr) \pmod{q}$. If $s = 0$, she returns to step 1.
4. Alice releases the pair (r, s) as the signature for the message m.

To verify a signed message, Bob does the following:

1. He computes $u_1 = ms^{-1} \pmod{q}$ and $u_2 = rs^{-1} \pmod{q}$.
2. He computes $u_1P + u_2Q = (x_0, y_0)$ and $v = x_0 \pmod{q}$.
3. Bob accepts the signature if $v = r$.

We now prove that the signature is valid, i.e. that someone with the knowledge of Alice's secret key, sent the message:

Proof. Using the process as above, Bob calculates

$$\begin{aligned} u_1P + u_2Q &= ms^{-1}P + rs^{-1}Q \\ &= m(k^{-1}(m+xr))^{-1}P + r(k^{-1}(m+xr))^{-1}Q \\ &= k(m+xr)^{-1}(mP + xrP) \\ &= k(m+xr)^{-1}(m+xr)P \\ &= kP \end{aligned}$$

And so the two points will have the same residue modulo q, namely r. □

In this proof, we've implicitly used various facts, such as $nP + mP = (n+m)P$ and $n(mP) = m(nP)$. These are all true, of course, and their proofs are simply routine checks using the fact that $E(\mathbb{F}_{p^n})$ is a finite abelian group.

These calculation show that a digital signature is verifiable, and given ECC takes place over fields for which q is extraordinarily large, the odds of a randomly chosen point also giving the same residue modulo q are miniscule.

Let's say we want to attack this encryption method to try and fake a signature from Alice. We obtain from Alice a signature pair (r, s) for a message m. Can we combine this with the public information (q, P and Q) to convincingly fake future signatures from Alice?

This is equivalent to asking if, for any message m, we can come up with a pair (r, s) that satisfies the calculation Bob will do. One method of doing this is to know Alice's private key, that is, the number x such that $Q = xP$, and calculating this is exactly the discrete logarithm problem for $E(\mathbb{F}_{p^n})$.

All other methods that might reverse the calculations done rely on knowing at least one of the random numbers k and x. If k is known, then kP can be calculated, and x can be calculated using the original message from Alice and the calculation in step 3. However, this again requires us to solve the discrete logarithm problem, this time for kP. Either way, faking signatures from Alice requires an attacker to either brute force their way to Alice's private key by checking nP for all n, or solving the discrete logarithm problem.

This is the crux of the encryption system the ECDSA puts in place. The discrete logarithm problem is assumed to be difficult to solve—though it is not known if it truly is. In this setting, Pollard's Rho algorithm runs in $\mathcal{O}(\sqrt{q})$, so all an implementor has to do to make this a useless avenue of attack is choose a large enough q.

Note, it is crucial that the value k is chosen randomly for each new signature. If an attacker knows that the same value of k has been used for two messages, they can extract Alice's private key as follows:

Suppose an attacker has two messages m_1 and m_2, with respective signatures (r_1, s_1) and (r_2, s_2). The s_i are calculated by $s_i = k^{-1}(m_i + xr_i) \pmod{q}$, so subtracting the two sequations gives

$$s_1 - s_2 = k^{-1}(m_1 - m_2 + x(r_1 - r_2)) \pmod{q}$$

However, as the r_i are calculated using the same k, they are the same, so we can rearrange to give

$$k = \frac{m_1 - m_2}{s_1 - s_2} \pmod{q}$$

Once k has been extracted, it is a short calculation to give $x = r_1^{-1}(ks_1 - m_1) \pmod{q}$. While this is only the residue of Alice's private key modulo q, it is enough for forging signatures from Alice as all the calculations in the algorithm are done modulo q anyway.

3.3 Secp256k1: The Bitcoin Curve

The elliptic curve used to sign Bitcoin transactions is called Secp256k1 (see [10]), and is defined as follows:

Definition 3.1. *The curve Secp256k1 is defined by the equation* $y^2 = x^3 + 7$ *over the field* $\mathbb{F}_p$*, where* $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

We note the base point of the curve only for the sake of completeness—it is rather long and doesn't do much for our edification:

(55066263022277343669578718895168534326250603453777594175500187360389116729240, 32670510020758816978083085130507043184471273380659243275938904335757337482424)

Over the real numbers, Secp256k1 looks like this:

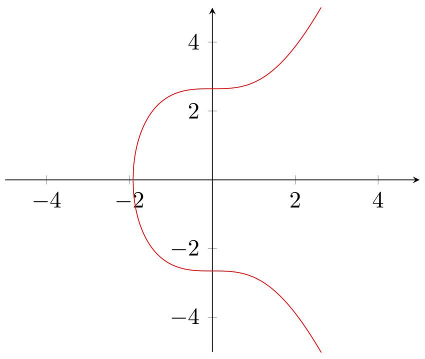

Figure 3: $y^2 = x^3 + 7$ over $\mathbb{R}$

Elliptic curves over finite fields look very different to their counterparts over $\mathbb{R}$. As the value of p used for Secp256k1 is rather too large for a plot on a piece of A4 paper, below is a plot of $y^2 = x^3 + 7$ over a smaller field:

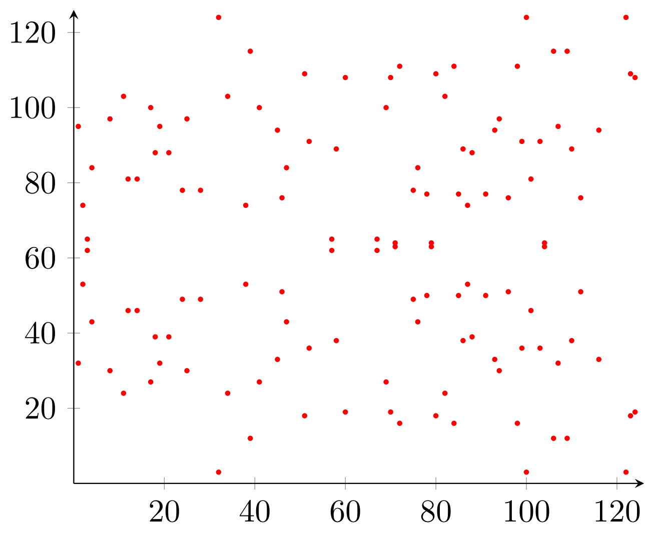

Figure 4: $y^2 = x^3 + 7$ over $\mathbb{F}_{127}$

We can still visualise the group law in this setting5. If we draw secant lines between points and allow ourselves to "wrap around" the sides of the $\mathbb{F}_p^2$ plane (in much the same way that residues of integers "wrap around" modulo n), we will find exactly one more point of intersection with the curve. In order to finish we need to reflect across the x-axis, but again imagining wrapping around, this is the same as reflecting through $y = p/2$.

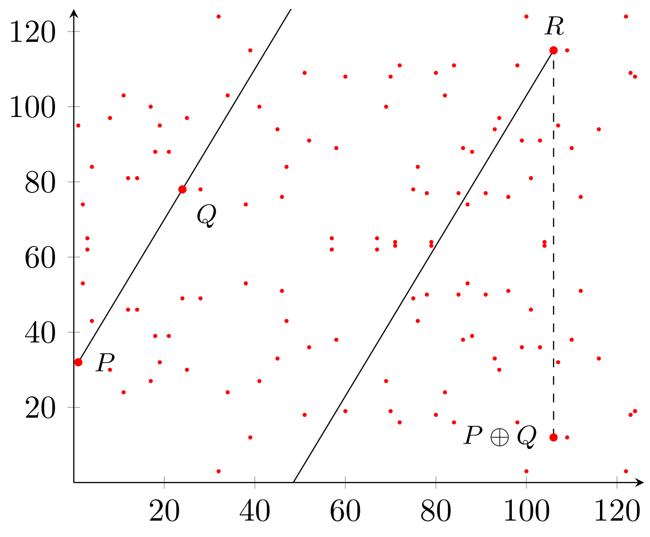

Figure 5: A visualisation of $(1, 32) \oplus (24, 78) = (106, 12)$, on $y^2 = x^3 + 7$ over $\mathbb{F}_{127}$.

Secp256k1 was designed by the United States Government's National Institute of Standards and Technology (or NIST). As such, the choices behind its various constants are not publicly known in great detail. The specific choice of the base point, for example, is not well-understood. However, the choice of prime is a fairly reasonable one.

The prime p used in Secp256k1 is what's called a *pseudo-Mersenne* prime, meaning it is composed of one large power of 2 and several small powers of 2. This is useful, because it can be leveraged when implementing Secp256k1 to calculate reduction modulo p much faster than the standard method of repeatedly peeling off copies of p until a number in the range $[0, p-1]$ is found. One algorithm that does this is the Solinas algorithm [21], which can be used since $2^{32} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 1$ is small relative to 2^{256}.

3.4 Other Cryptographic Curves

Secp256k1 is not the only curve widely used in current elliptic curve cryptography. The NIST detailled several recommended standard curves and associated fields in 1999 [17]. Ostensibly, these curves were chosen for their cryptographic security and computational efficiency. However, some cryptographic tools touted as secure standards by the NIST have been shown in recent years to have been weakened by the National Security Agenecy (NSA). In 2013, the New York Times reported that certain aspects of the Dual_EC_DRBG random number generator had been deliberately influenced to provide a backdoor to the NSA [18].

This random number generator was used to create private and public keys, so its security underlies the security of the encryption. A random number generator that is secretly deterministic in some way could be leveraged to provide a far smaller pool of potential private keys to an attacker, which could then be manually checked. Because of this backdoor, implementors of elliptic curve cryptography have become increasingly wary of cryptography standards put forward by the NIST.

This distrust has led to a rise in popularity of curves created by non-governmental organisations, and often even individuals. One such curve is Curve25519, published by Daniel J. Bernstein in [5]. The detail with which the various choices around the curve are explained has led many to trust that no backdoors have been inserted into it, as the various parameters as well as the structure of the curve were all chosen with an eye towards efficiency and security in practical implementations of the encryption protocol. This does not mean backdoors have not been inserted, rather it just makes it much less likely. If nothing else does, the detail present in [5] makes it clear that choosing a curve that provides secure ECC is a non-trivial task.

4 Implementing ECC

The main advantage of implementing the ECDSA over standard RSA-based digital signatures is that ECC offers approximately double the security of RSA methods with private keys of the same length. However, this theoretical increase in security requires that the implementor understand the mathematics of the elliptic curves well.

As well as allowing one to weaken the ECDLP, understanding the mathematics of elliptic curves can allow a designer to make the implementation of the encryptions run more efficiently than the naive approach.

4.1 Pollard's Rho Algorithm

Pollard's Rho algorithm [7] can be used to find discrete logarithms in the group $E(K)$ of an elliptic curve E. If we want to find a $k \in \mathbb{N}$ such that $kP = Q$, Pollard's Rho algorithm gives integers a, b, c, d such that

$$aP + bQ = cP + dQ$$

which is equivalent to saying

$$(d - b)Q = (a - c)P \quad \Rightarrow \quad Q = (a - c)(d - b)^{-1}P$$

where $(d - b)^{-1}$ is the inverse of $d - b$ modulo $\#E(K)$ (note, this may not always exist, see below).

The algorithm goes as follows (per [1], page 148): To set up, we divide $E(K)$ into subsets of roughly the same size, $S_1, S_2, ..., S_s$, and choose a pair of random integers a_i and b_i modulo $|E(K)|$ for each S_i. We calculate the points $M_i = a_iP + b_iQ$, and define the function $f(R) = R + M_i$ when $R \in S_i$.

To execute, we take our point P and calculate $P_0 = a_0P + b_oQ$, where again the a_0 and b_0 are random integers. Then we calculate $P_{k+1} := f(P_k)$. As G is finite, this process will eventually give integers $i_0 < j_0$ such that $P_{i_0} = P_{j_0}$. From here, the sequences $P_{i_0}, f(P_{i_0}), f(f(P_{i_0})), ...$ and $P_{j_0}, f(P_{j_0}), f(f(P_{j_0}))$ are the same, meaning the overall sequence is eventually periodic. This is where the algorithm gets its name, due to the resemblance of this process to the Greek letter ρ.

At each step, we keep a note of the expression $P_k = u_kP + v_kQ$ in the vector (u_k, v_k). If $P_k \in S_i$, we have $P_{k+1} = u_kP + v_kQ + a_iP + b_iQ = (u_k + a_i)P + (v_k + b_i)Q$, so $(u_{k+1}, v_{k+1}) = (u_k+a_i, v_k+b_i)$. Checking each new point against our list, we will eventually find a pair $(u_i, v_i) = (u_j, v_j)$. This corresponds to the relation $u_iP + v_iQ = u_jP + v_jQ$

which is exactly as we require. Inverting the value $u_i - u_j$ may not always be possible, as it could divide the order of the group $n = |E(K)|$. If it does, all the factors $p_i^{e_i}$ that $u_i - u_j$ shares with n must be divided out of n, giving the inverse modulo $d = p_1^{e_1} \cdot \ldots \cdot p_r^{e_r}$. From here, one simply checks each of the d possibilities for k modulo n.

Pollard showed in his original paper that the algorithm will run with time complexity $\sqrt{\pi p/2}$ on average, where p is the order of the base point of the curve. The algorithm is probabilistic, meaning it could actually take much longer than this.

The Rho algorithm can also be adapted to run over equivalence classes in the group, rather than the points themselves. The idea here is to speed up the algorithm by checking if a point in the walk is equal to any of the previous points, as well as points equivalent to them. This extra check takes slightly longer, but overall we can reduce the number of steps in our random walk. This provides an overall saving in the time complexity as checking if two points lie in the same class is much less computationally intensive than addition in the group $E(K)$.

If we can split $E(K)$ into m equivalence classes, the algorithm can be made to run with time complexity $\sqrt{\pi p/2m}$ (see [9] for details). One such equivalence relation that yields fast checking of equivlanece classes is defining $P \sim Q \Leftrightarrow \tau(P) = Q$, where τ is an automorphism of the group.

This can be applied to the curve Secp256k1. From Proposition 2.13, we know the curve has an automorphism group isomorphic to $\mathbb{Z}/6\mathbb{Z}$, generated by $\tau((x, y)) = (\zeta x, -y)$. In order to get the required speedup, we want $\zeta \in \mathbb{F}_p$. If it's not in $\mathbb{F}_p$, the best speedup we can get is by using the automorphism $(x, y) \mapsto (x, -y)$, as all other automorphisms will give points in the algebraic closure that won't be come across in the Rho algorithm.

We can check if there is such a $\zeta \in \mathbb{F}_p$, where $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, using SageMath. We define the field of size p, and ask for the roots of the polynomial $x^3 - 1$:

```
sage: K.<x> = GF(2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 )[]
sage: f = x^3-1
sage: f.roots()
```

which returns

```
[(60197513588986302554485582024885075108884032450952339817679072026166228089408, 1),
```

```
[(601975135889863025544855820248850751088840324509523398176790720261 6622\
8089408, 1),
(55594575648329892869085402983802832744385952214688224221778511981742606\
582254, 1), (1, 1)]
```

Sage returns the roots in the form (α, λ), where α is the root and λ is its multiplicity. In this case we have three roots each of multiplicity one.

Setting $\zeta_1 = 60197513588986302554485582024885075108884032450952339817679072026166 22$ and $\zeta_2 = 55594575648329892869085402983802832744385952214688224221778511981742606$, we have the two automorphisms of Secp256k1 given by

$$\tau_1((x,y)) = (\zeta_1 x, -y), \quad \tau_2((x,y)) = (\zeta_2 x, -y)$$

We could also use the automorphism for the third root, but since this third root is 1 we would only get $(x,y) \mapsto (x,-y)$, which has order 2 and therefore provides less of a speedup.

Comparing the speed of Rho over the group with the speed of Rho over the equivalence classes provided by one of the τ_i, we can see the algorithm is only faster by a factor of $\sqrt{6} \approx 2.45$. Given the private keys used in Secp256k1 are 256 bits long, this factor does not weaken the discrete log problem sufficiently to render the encryption useless. In the worst case scenario, it makes the security equivalent to keys of length 254 bits, which would still take an infeasible amount of time to break.

4.2 Other Attacks

Pollard's Rho algorithm is the best-general purpose attack on the ECDLP. However, there exist specialised attacks for curves with particular propoerties.

4.2.1 Pohlig-Hellman

Elliptic curve cryptography is done in the group $E(\mathbb{F}_q)$, but strictly it takes place in the group $\langle P \rangle$, with P is the base point of E specified in the encryption. Every implementation of ECC uses a point of prime order, but this isn't actually required for the algebra to still work. The reason a point of prime order is always required is because a non-prime order significantly weakens the discrete logarithm problem on the elliptic curve:

Setting $\mathrm{ord}(P) = n = p_1^{e_1} \cdot \ldots \cdot p_r^{e_r}$, $kP = Q$, and writing $k = \overline{k} + rp_i^{e_i}$ we can see that

$$\begin{aligned}
\overline{k}\frac{n}{p_i^{e_i}}P &= \frac{kn - rp_i^{e_i}n}{p_i^{e_i}}P \\
&= \frac{n}{p_i^{e_i}}kP - rnP \\
&= \frac{n}{p_i^{e_i}}Q
\end{aligned}$$

Thus $\overline{k} = k \pmod{p_i^{e_i}}$ can be calculated by solving a discrete log in a cyclic group of order $p_i^{e_i}$. When $e_i > 1$ the value of $\overline{k}$ can be found by first finding the discrete log in a cyclic group of order p_i, then lifting the value to $\pmod{p_i^{e_i}}$.

Finally, the values of $\overline{k}$ can be combined to give k modulo n using the Chinese Remainder Theorem. As $(k + rn)P = kP$, this gives the discrete logarithm.

This is called the Pohlig-Hellman attack (see [13]), and it reduces the problem of finding a discrete logarithm of a point to finding discrete logarithms modulo the prime factors of the group. This can be used in conjunction with Pollard's Rho algorithm to find discrete logs wth time complexity equal to the square-root of the largest prime factor of the base point's order.

In all cryptographic applications, then, it is highly recommended to choose a base point whose order is a prime. This prevents the Pohlig-Hellman algorithm from providing any speedups to Pollard's Rho, effectively thwarting this attack. Secp256k1 has a base point of prime order, rendering this attack useless for breaking the security of Bitcoin.

4.2.2 Transfer Attacks

Understanding the structure of the group $E(K)$ is useful when trying to solve the discrete logarithm problem. Most current instances of ECC use curves defined over a prime field, so we will focus on transfers of this kind.

A transfer is simply a map from the group $E(\mathbb{F}_p)$ to a group where solving the discrete logarithm is faster than using an algorithm like Pollard's Rho. If we can find such a map, it would make the discrete logarithm problem much easier to solve.

To know where we can map our group $E(\mathbb{F}_p)$, we first need to know its order. Calculating the trace t of the Frobenius endomorphism, we have that $\#E(\mathbb{F}_p) = p + 1 - t$. In the best case scenario (from an attacker's perspective), we have that $t = 1$ and so $\#E(\mathbb{F}_p) = p$, and we can perform an additive transfer into the group $(\mathbb{F}_p,+)$. This allows for the discrete logarithm problem to be solved in linear time. This was first proposed by Smart in [14].

If $\#E(\mathbb{F}_p) \neq p$, we find the smallest k such that $\#E(\mathbb{F}_p) \mid (p^k - 1)$. This k is called the embedding degree of the group $E(\mathbb{F}_p)$. From here, we can then map our group into the multiplicative group $(\mathbb{F}_{p^k})^\times$. Here the problem is solved in sub-exponential time using index calculus. This is called the MOV-attack, and was first proposed in [15].

The larger k is, the more cumbersome the process of finding a discrete logarithm in $(\mathbb{F}_{p^k})^\times$ becomes. There are various opinions on what constitutes a large enough embedding degree to render the discrete logarithm problem for a specific $E(\mathbb{F}_p)$. [10] states that k should be at least 20, whereas [11] puts the minimum value of k at $\frac{\#E(\mathbb{F}_p)-1}{100}$.

If $E(\mathbb{F}_p)$ is supersingular then $t = 0$ for $p \geq 5$ (see [1], page 135), and so the group order is $p+1$. This is a divisor of $p^2 - 1$, giving a multiplicative transfer of degree 2 into $(\mathbb{F}_{p^2})^\times$. This breaks the ECDLP much faster than Pollard's Rho algorithm.

But is Secp256k1 safe from a transfer attack? The smallest k such that $\#E(\mathbb{F}_p) \mid (p^k - 1)$ is exactly the multiplicative order of p modulo $\#E(\mathbb{F}_p)$. Using Sage, we can calclate this:

```
sage: p = 2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1
sage: E = EllipticCurve(GF(p),[0,7])
sage: n = E.order()
sage: mod(p,n).multiplicative_order()
```

which returns

```
192986815395526992372618308347813179754729273798458173971008605235863602\
49056
```

Certainly this is larger than 20, and in fact it is equal to $\frac{n-1}{6}$, so it satisfies the conditions of both [10] and [11]. To carry out a transfer attack against Secp256k1, one would have to work in a field of size approximately $10^{10^{78}}$, which seems somewhat infeasible.

4.3 Montgomery Curves

In 1987, Peter L. Montgomery propsoed implementing elliptic curve encryption over a type of curve now called a Montgomery curve in his honour. The main advantage of Montgomery curves is that they admit formulae for point addition that is generally much faster than those for curves in short Weirstrass form.

Definition 4.1. *A Montgomery curve E is a plane cubic curve given by*

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

with $B(A^2 - 4) \neq 0$.

The condition $B(A^2 - 4) \neq 0$ ensures this defines an elliptic curve, meaning the standard group law given by taking the sum of three co-linear points to be ∞ holds here too. Notice as well, there is no linear y term, meaning inverting a point is still just flipping it across the x-axis, as in the short Weirstrass case.

The formulae take a point $P = (x_1, y_1)$ and calculate $(n + m)P$. To do this, we write the point P in projective co-ordinates: $P = [X_1 : Y_1 : Z_1]$, with $x_1 = \frac{X_1}{Z_1}$ and $y_1 = \frac{Y_1}{Z_1}$. Then writing $tP = [X_t : Y_t : Z_t]$ and, we have the formulae [8]:

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

when $m \neq n$. When $m = n$, we have

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

Initially these don't seem that useful, as we need to know the difference of the two points at every step. Calculating this before calculating the sum of the two points negates any speedups we might hope to get from these formulae, but if we choose m and n differing by 1, we always get differences of X_1 and Z_1. Notice, these formulae don't calculate either of y_{2n} or Y_{2n} at any step—however, this is not needed for the ECDSA as only the x co-ordinate is used in the algorithm. For variants of ECC where the y co-ordinate is needed, there exist simple methods for recovering this information [19].

Performing divisions in $\mathbb{F}_p$ on a computer takes approximately four times longer than performing multiplications, and these formulae only use multiplications and additions (with additions being even faster than multiplications). The standard group law formulae use several divisions, meaning the Montgomery addition formulae should run much faster if correctly implemented.

This is done by chaining them together to form what's called the *Montgomery ladder*, which can be used to calculate nP. The algorithm for doing this goes as follows:

First, we write n in the form of a binary expansion. That is, $n = a_0 + 2a_1 + 2^2a_2 + ... + 2^ra_r$. Then we perform the following (from [3], page 287):

Algorithm 1 Montgomery's Scalar Multiplication Ladder

```
1: $P_1 \leftarrow P$ and $P_2 \leftarrow 2P$
2: for $i = l - 1$ down to $0$ do
3:     if $n_i = 0$ then
4:         $P_1 \leftarrow 2P_1$ and $P_2 \leftarrow P_1 \oplus P_2$
5:     else
6:         $P_1 \leftarrow P_1 \oplus P_2$ and $P_2 \leftarrow 2P_2$
7: return $P_1$
```

At each step, $P_2 - P_1$ is always P, up to sign. We can see this by checking at the start that $2P - P = P$, and then noticing that at every step we are replacing P_1 and P_2 with points whose difference is still P_1.

The Montgomery Ladder provides faster calculations when $n > 2$, as we only need to perform one division at the end, when we leave projective space. All the rest of the work is done with addition and multiplication.

The Montgomery ladder is incredibly useful if your curve has the shape $By^2 = x^3 + Ax^2 + x$, but many of the curves used in modern elliptic curve cryptography are not. Most of them, including Secp256k1, come in short Weirstrass form. Over fields not of characteristic 3, we can transform a Montgomery curve into a short Weirstrass curve by dividing by B and depressing the resulting monic cubic in x to get rid of the x^2 term. However, if we want to use these formulae to speed up calculations for a short Weirstrass curve, we want to be able to go in the other direction.

A curve $y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$ can be transformed into a Montgomery curve over $\mathbb{F}_p$ exactly when:

(i) $x^3 + Ax + B$ has at least one root α in $\mathbb{F}_p$

(ii) $3\alpha^2 + A$ is a square in $\mathbb{F}_p$

as per page 286 in [3].

This kind of equivalence is called *birational* equivlanece, and it doesn't always hold. We might hope, for example, that Secp256k1 can be transformed into a Montgomery curve in order to speed up the calculations used for signing Bitcoin transactions. We elaborate on the specific calculations in Appendix A, as they are rather cumbersome.

In essence, we first want to check if $x^3 + 7$ factors. We do this with the Sage commands in Appendix A, which show it does not. In order to turn Secp256k1 into a Montgomery curve, then, we would first need to work over the field $\mathbb{F}_{p^3}$, where $y^2 = x^3 + 7$ factors. If we wished to proceed from here, in spite of the size of the underlying field spiralling out of our control, we would then have to check whether each $3\alpha^2$ is a square in $\mathbb{F}_{p^3}$ (for α a cube root of -7 in $\mathbb{F}_{p^3}$).

Again appealing to Appendix A, we see that $3\alpha^2$ is not a square for each α a root of $x^3 + 7$ over $\mathbb{F}_{p^3}$, meaning to convert Secp256k1 into a Montgomery curve, we would have to work over the field $\mathbb{F}_{p^6}$. This field is of size approximately 10^{462}, and consists of degree 5 polynomials over $\mathbb{F}_p$, making the calculations much more cumbersome than any saving we might make from working with a Montgomery curve.

This seems unfortunate, and we might wonder how likely it is that a given elliptic curve over a finite field might be "Montgomerisable" in this way. To check this, I wrote a short program in Sage that returns the degree of the smallest field extension required to turn a Weirstrass curve into a Montgomery curve. This can be found in Appendix B, as well as code that checks this function for all possible elliptic curves over all prime fields up to a certain size.

Over the first 50 prime fields, I found the following:

n	1	2	3	6
Proportion	$\frac{125097}{379084}$	$\frac{253563}{758168}$	$\frac{108401}{758168}$	$\frac{73005}{379084}$
Approx. Percentage	32.99	33.44	14.29	19.25

where n is the degree of the smallest extension $\mathbb{F}_{p^n}$ such that a short Weirstrass curve over $\mathbb{F}_p$ is "Montgomerisable" over it. For the first 758168 elliptic curves, the proportion and corresponding approximate percentage of curves that require a degree n extension is given by the table. While the sample size is fairly small, these results still seem to suggest we were somewhat unlucky when it comes to the "Montgomery degree" of Secp256k1.

Given the curve's failure to be easily transformable into a Montgomery curve, we are forced to look elsewhere for ways of speeding up its calculations.

4.4 Scalar Multiplication with Endomorphisms

Following [16], let E be an elliptic curve over a prime field $\mathbb{F}_p$, and let $P \in E(\mathbb{F}_p)$ be a point of prime order n. Every endomorphism $\phi \in \text{End}(E)$ has a characteristic polynomial of degree 2, meaning $\phi^2 + \alpha\phi + \beta = 0$ for some $\alpha, \beta \in \text{End}(E)$. If this polynomial has the factor $(\phi - \lambda)$ modulo n for some $\lambda \in \mathbb{Z}$, then ϕ acts like $[\lambda]$ on the points $\langle P \rangle$.

If this endomorphism is easy enough to compute, it can be used to speed up scalar multiplication of points in $\langle P \rangle$. We do this as follows: to calculate kP, we find a representation $k = k_1 + k_2\lambda \pmod{n}$ with $k_1, k_2 \in [0, \sqrt{n}]$. This gives us

$$\begin{aligned} kP &= k_1P + k_2\lambda P \\ &= k_1P + k_2\phi(P) \end{aligned}$$

Provided finding this representation of k is not computationally taxing (which it tends not to be), we can calculate kP by doing two scalar multiplications of shorter bitlength than k and one evaluation of ϕ. A more thorough analysis of why this works, as well as an algorithm for computing k_1P and $k_2\phi(P)$ simultaneously, can be found in [16].

What does all this mean for Secp256k1? Secp256k1 has an endomorphism group isomorphic to $\mathbb{Z}\left[\frac{1+\sqrt{-3}}{2}\right]$, with the automorphisms τ_1 and τ_2 of order 6 (**§4.1**). These automorphisms are efficiently computable, as they use only two reductions modulo p. As per [16], these methods can speedup scalar multiplication by about 50% for fields of bitlength 160. This ratio gets better as the size of the field gets larger, and Secp256k1 being defined over a field of bitlength 256 means the time saved should be at least as much as 50%.

5 Conclusion

5.1 Is Secp256k1 a Good Choice?

Given the numerous possible attacks on ECC and their varied nature, it's clear that a curve and field chosen completely at random have little chance of providing secure encryption, and even if they do they stand little chance of having the required properties to provide faster computation methods on top of that. Then to get anything useful done with ECC, we have to rely on the standards set out by the experts.

The question for anyone attempting to implement ECC then becomes, how much should we trust the standard curves? Given the accusations by some that the NSA has intentionally weakened some of the NIST's cryptographic standards, taking it on blind faith that the encryption standards put forward by anyone can be trusted seems ludicrous. The budding cryptographer, then, has to take the time to understand the mathematics behind ECC and evaluate for themselves whether a given standard should be trusted.

From the mathematical perspective, there is nothing to suggest Secp256k1 is a bad choice for ECC. It resists Pollard's rho attack (even with speedups from its additional automorphisms), and has an embedding degree that would make a transfer attack take longer than the age of universe.

From the perspective of a paranoid cryptographer, though, it might not. If it, too, was weakened under influence from the NSA, it could provide the United States Government with a means of reading and forging private communications from people all over the globe. This is an especially troubling notion for Bitcoin, as it was designed to be free from influence and interference from any outside power.

Given that the NIST hasn't released detailled information about why the specific constants associated with the curve were chosen (and noting their previous association with weakening encryption standards for the NSA), many think curves like Secp256k1 may have also been tampered with, and that they should be avoided if possible.

Barring further Snowdon-eqsue leaking of classified government material, we will likely never know for sure whether this is true. In general, then, it seems a better idea to use curves like Curve25519, which have a detailled analysis of the decision-making process behind them publicly available. The odds are far slimmer that these curves have been designed maliciously.

The odds of getting enough of the Bitcoin userbase to switch, though, are equally slender. Given the number of people using the currency, it would likely take irrefutable proof of malfeasance to get enough users to switch from the system that has been working fine for them so far. This means anyone hoping to invest in Bitcoin more or less has to take it on faith that the security has not been tampered with in any way.

5.2 The Future of ECC (and Bitcoin)

The most dangeorus thing about ECC seems to be the variance in the dificulty of the discrete logarithm problem over all curves and fields. The level of mathematical knowledge required to verify the security of a randomly chosen curve over a randomly chosen field can be non-trivial, and with more and more money riding on the effectiveness of ECC, there will only be more attempts to undermine it in future. There is currently nothing to suggest ECC will become obselete overnight, but there is also nothing to suggest it won't. It is entirely conceivable someone is working on an attack that will undermine vast swathes of the current standards.

The job of designing an elliptic curve for cryptographic purposes is already a hard needle to thread; if new attacks are found that make this job even harder, we may see ECC wane in popularity. Its popularity is already starting to wane, due to its theoretical breakage under the power of quantum computing. Although quantum computers do not currently exist in a state capable of breaking ECC, it is conceivable that they will eventually, and so attention is increasingly turning to quantum-resistant encryption methods such as lattice-based methods and the Lamport scheme.

For now, though, the technology seems reasonably secure. If quantum computers arrived tomorrow, they would also break traditional RSA methods of encryption. Given this underpins much of the global banking and financial systems, we would have far bigger problems on our hands than Bitcoin transactions becoming untrustworthy.

Assuming no grand changes in the cryptographic landscape, the future of Bitcoin is still a mystery to more or less everyone. There are many who claim to know where the currency is headed, but for every expert touting one opinion as fact, there is another arguing for exactly the opposite view. Time and again Bitcoin has defied the expectations of many, so it seems Bitcoin will follow the golden rule of new technology: precious few people will predict its journey correctly, and those who claim to be able to likely have a vested interest in the outcome.

5.3 Final Thoughts

When most people hear about potential weaknesses in the encryption that governs their internet traffic, they get worried and start unplugging the wireless router. Similarly, possible weaknesses in the ECDSA for Secp256k1 could have serious ramifications for the future of Bitcoin. And while the standard seems safe enough for now, computers will grow in power. If Moore's Law of doubling computer power is to be trusted, eventually we will be able to solve the ECDLP even for Secp256k1. This is not news to anyone using Bitcoin, and there's a good few years before classical computers become powerful enough to do this.

So, essentially, Bitcoin is safe for now. The cultural inertia of its userbase may prove to be its fatal flaw, however, as recent news has shown the Bitcoin-owning population to be a hard beast to control. For evidence of this, one has to look no further than the littany of diverging Bitcoin standards (called “forks” in the parlance) that are cropping up with alarming frequency [23].

And while the failure of Secp256k1 would drastically affect the stability of Bitcoin's price, the reverse picture doesn't hold. That is to say, just because it's safe, it doesn't mean it's stable. In researching this project, I've come across a certain amount of ancillary material about the behavoir of the price of Bitcoin, and from it I've drawn the following conclusion: it is possible to get rich using Bitcoin, if you invested eight years ago.

Appendix A Attempting to “Montgomerise” Secp256k1

We take Secp256k1, the curve $y^2 = x^3 + 7$ defined over the field $\mathbb{F}_p$ (with $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$). We first show that -7 is not a cube in this field. We define the polynomial

```
sage: R.<x> = GF(2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 )[]
sage: f = x^3+7
sage: f.roots()
```

This returns the empty array, meaning $x^3 + 7$ doesn’t factor over this field.

This already looks pretty bad for “Montgomerising” Secp256k1, but it is possible we would have to work over an even bigger extension. To satisfy the second criterion, we need to check whether any of these roots correspond to squares in $\mathbb{F}_{p^3}$. So, we define the field $\mathbb{F}_p(\sqrt[3]{-7}) \cong \mathbb{F}_{p^3}$ in Sage, and ask for the roots again:

```
sage: p=2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1
sage: K.<crm7>=GF(p**3,modulus=(x^3+7))
sage: R.<t>=PolynomialRing(K)
sage: f=t^3+7
sage: f.roots()
```

This returns

```
[(601975135889863025544855820248850751088840324509523398176790720261662\
8089408*crm7, 1),
(555945756483298928690854029838028327443859522146882242217785119817426\
6582254*crm7, 1),
(crm7, 1)]
```

where “crm7” is $\sqrt[3]{-7}$, a root of $x^3 + 7$. Each of these is an α for which we need to check whether $3\alpha^2 + A$ is a square in $\mathbb{F}_{p^3}$. For Secp256k1, $A = 0$, so we only need check each $3\alpha^2$.

The relevant Sage commands are:

```
sage: g1=t^2-3*(6019751358898630255448558202488507510888403245095233981
679072026166228089408*crm7)^2
sage: g1.roots()
```

which returns no roots;

```
sage: g2=t^2-3*(5559457564832989286908540298380283274438595221468822422
1778511981742606582254*crm7)^2
sage: g2.roots()
```

which returns no roots, and

```
sage: g3=t^2-3*crm7^2
sage: g3.roots()
```

which also returns no roots.

None of these polynomials have roots, so each $3\alpha^2$ is not a square in $\mathbb{F}_{p^3}$. This means we would have to adjoin one of these square roots in order to fully "Montgomerise" Secp256k1.

Appendix B Weirstrass To Montgomery Sage Code

The following Sage code defines the function that determines the lowest degree field extension required to transform a Weirstrass curve f over the field of size p into a Montgomery curve:

```
def monty(f,p):

    A=f.list()[1]
    B=f.list()[0]
    R.<x>=GF(p)[]

    if factor_degree(f.roots())==1:
         g=x^2-(3*f.roots()[0][0]^2+A)
         if g.roots()!=[]:
             return 1
         else:
             return 2

     elif factor_degree(f.roots())==3:
         for i in range(len(f.roots())):
             g=x^2-(3*f.roots()[i][0]^2+A)
             if g.roots()!=[]:
                 return 1
             else:
                 return 2

     else:
         Field3.<alpha>=GF(p**3,modulus=(f))
         Ring3.<t>=PolynomialRing(Field3)
         h=t^3+A*t+B
         for i in range(len(h.roots())):
             g=t^2-(3*h.roots()[i][0]^2+A)
             if g.roots()!=[]:
                return 3

     return 6
```

This uses the function `factor_degree`, which counts the number of roots of the polynomial over the field $\mathbb{F}_p$ with multiplicity. This isn't hard to write; I include mine only for the sake of completeness:

```
def factor_degree(A):
    r=0
    for j in range(len(A)):
        r=r+A[j][1]
    return r
```

This code is written to be utilised as in the following example:

```
sage: R.<x>=GF(31)[]
sage: monty(x^3+2*x+5,31)
```

which returns a value of 2. In this case, the extension is $\mathbb{F}_{31}(\alpha) \cong \mathbb{F}_{31^2}$, where α is a root of $x^2 + 15x + 10$.

I also wrote a program that iterates over the function `monty(f,p)`, checking every possible elliptic curve over a finite field:

```
def monty_stats(prime):

    R.<x>=GF(prime)[]
    results=[0,0,0,0,0,0,0]
    baddies=0

    for i in range(0,prime):
        for j in range(0,prime):
            poly = x^3+i*x+j
                if prime.divides(4*i^3+27*j^2):
                    baddies=baddies+1
                else:
                    if monty(poly,prime)==1:
                        results[1]+=1
                    elif monty(poly,prime)==2:
                        results[2]+=1
                    elif monty(poly,prime)==3:
                        results[3]+=1
                    elif monty(poly,prime)==6:
                         results[6]+=1
                    else:
                            print "Error in executing monty(x^3+" +
                            str(i) + "*x+" + str(j) + "," + str(prime)
                            + ")---No possible extension found."

    return results
```

It seems for every prime p, the number of curves defined by $x^3 + ix + j$ that aren't an elliptic curve is exactly p. This is true at least for all primes up to $p_{50} = 229$, which is as far as my statistical analysis went (anything further might have melted my laptop!), so I assume it here.

Finally, the function `monty_stats(prime)` was iterated over for the first 50 primes using the following code:

```
n=50
A=[0,0,0,0,0,0,0]
d=0
for j in [1..n]:
    prime = nth_prime(j)
    stats=monty_stats(prime)
    for i in range(7):
        A[i]=A[i]+stats[i]
    d=d+prime^2-prime
B=[x/d for x in A]
print B
```

Which gave the final vector

$$B = [0, 125097/379084, 253563/758168, 108401/758168, 0, 0, 73005/379084]$$

as cited in the main body of the report.

Bibliography

[1] Washington L.C. (2008) *Elliptic Curves: Number Theory and Cryptography.* Chapman & Hall/CRC.

[2] Koblitz N. (1998) *Algebraic Aspects of Cryptography.* Springer, Berlin, London.

[3] Cohen H. and Frey G. (2006) *Handbook of Elliptic Curve and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC.

[4] Nakamoto S. (2009) *Bitcoin: A Peer-to-Peer Electronic Cash System.* `http://bitcoin.org/bitcoin.pdf`

[5] Silverman, J.H. (2016) *The Arithmetic of Elliptic Curves, 2^{nd} Edition*, Springer-Verlag New York.

[6] Bernstein D.J. (2006) *Curve25519: New Diffie-Hellman Speed Records.* Lecture Notes in Computer Science, **3958**. Springer, Berlin, Heidelberg.

[7] Pollard J. M. (1978) *Monte Carlo Methods for Index Computation* (mod p). Mathematics of Computation **32**. 918-924.

[8] Montgomery, P. L. (1987) *Speeding the Pollard and elliptic curve methods of factorization.* Mathematics of Computation **48**, 243-264.

[9] Duursma, I. and Gaudry, P. and Morain, F. (1999) *Speeding Up the Discrete Log Computation on Curves With Automorphisms.* Lecture Notes in Computer Science **1716**, Springer.

[10] Certicom Research. (2010) *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0.* `http://www.secg.org/sec2-v2.pdf`

[11] Bernstein, D.J. and Lange, T. (2013) *Safecurves: choosing safe curves for elliptic-curve cryptography.* `http://safecurves.cr.yp.to/`

[12] Sutherland, A. V. and Bisson, G. (2009) *Computing the Endomorphism Ring of an Ordinary Elliptic Curve*, `http://www-math.mit.edu/~drew/CCRTalk.pdf`

[13] Sommerseth, M.L. and Hoeiland, H. (2015) *Pohlig-Hellman Applied in Elliptic Curve Cryptography.*

[14] Smart, N.P. (1999) *The Discrete Logarithm Problem on Elliptic Curves of Trace One.* Journal of Cryptology **12**, 193-196.

[15] Menezes, A.J. and Okamoto, T. and Vanstone, S.A. (1993) *Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field.* IEEE Transactions on Information Theory **39**, 1639-1646.

[16] Gallant, R.P. and Lambert, R.J. and Vanstone, S.A. (2001) *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms.* Lecture Notes in Computer Science, **2139**, Springer, Berlin, Heidelberg.

[17] National Institute for Standards and Technology. (2000) *Digital Signature Standard.* Federal Information Processing Standards Publication **186-2**.

[18] Perlroth, N. (2013) *N.S.A. Able to Foil Basic Safeguards of Privacy on Web.* New York Times.
`https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html`

[19] Okeya, K. and Sakurai, K. (2001) *Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve.* Lecture Notes in Computer Science, **2162**. Springer, Berlin, Heidelberg.

[20] Wiener, M.J. and Zuccherato, R.J. (1998) *Faster Attacks on Elliptic Curve Cryptosystems.* Lecture Notes in Computer Science, **1556**. Springer, Berlin, Heidelberg.

[21] Solinas, J. (1999) *Generalized Mersenne Numbers.* Technical Report CORR 99-39, Centre for Applied Cryptographic Research, University of Waterloo.
`http://cacr.uwaterloo.ca/techreports/1999/corr99-39.ps`

[22] Anni, S. (2014) *MA426: Elliptic Curves*
`http://www.iwr.uni-heidelberg.de/groups/arith-geom/anni/MA426.pdf`

[23] Harper, C. (2018) *2018 Recent and Upcoming Bitcoin Hard Forks: What You Need to Know.* Coin Central.
`https://coincentral.com/the-upcoming-bitcoin-hard-forks-what-you-need-to-know/`

[24] Jarvis, F. (2014) *Algebraic Number Theory*, Springer, Cham.