Algebraic Foundations And Computational Attacks In Elliptic Curve Cryptography

av

Viggo Haeggström Grunewald

2025 - No K18

Algebraic Foundations And Computational Attacks In Elliptic Curve Cryptography

Viggo Haeggström Grunewald

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Jonas Bergström

2025

Abstract

Elliptic curve cryptography (ECC) is widely used in modern cryptographic systems and relies on the presumed hardness of the elliptic curve discrete logarithm problem (ECDLP), over suitably chosen curves. This thesis first develops the algebraic background including modular arithmetic, finite fields, the Weierstrass equation and the group law, before formally stating the ECDLP and examining potential attacks. We analyze the index-calculus method, a sub exponential algorithm, as an attack against the general discrete logarithm problem and discuss why it generally does not does not work for elliptic curves. After this we shift focus to Pollard's rho and lambda methods by analyzing how pseudo-random walks, cycle detection and the extraction of the secret k from colliding states works. To evaluate practical performance, we implement Pollard's rho algorithm in SageMath and test it in various scenarios. One test focuses on corroborating Washington's heuristic that about twenty partitions of the subgroup generated by P is near-optimal. To do this, 50 randomly generated curves over prime fields of order roughly 10^8 were generated, varying the walk-partition parameter s $\in \{2,\dots,40\}$. The experiment shows a steep runtime decline up to s ≈ 15 and only marginal gains beyond s ≈ 20. We then benchmark our implemented version of Pollard's rho against SageMath's built in version. This tests shows that SageMath's version is significantly more efficient, likely due to SageMath using cython. We then conclude by timing Pollard's rho against Pollard's kangaroo method using SageMath's built in versions. Which highlights how information about the range of the secret k can greatly improve performance. If it is known that $k \in (a,b)$ with $L = b-a$ and $|P| = n$, we observe that for $L = n/log(n)$ the runtime can be more than halved. Subsequent tests shows an even greater increase in speed for smaller intervals L.

Abstract

Elliptiskurvkryptografi (ECC) används världen över i moderna kryoptografiska system och förlitar sig på den förmodade svårigheten i det diskreta logaritmproblemet för elliptiska kurvor (ECDLP), över lämpligt valda kurvor. Detta examensarbete inleds med en redogörelse för den algebraiska bakgrunden nödvändig för att förstå strukturen kring elliptiska kurvor och attacker mot ECDLP. Vi går bland annat igenom modulräkning, ändliga kroppar, Weierstrass ekvationen och grupplagen som definierar hur additioner av punkter på kurvan sker. Vi analyserar index-calculus metoden, en algoritm som har snabbare tidskomplexitet än övriga undersökta, som en attack mot det generella diskreta logaritmproblemet (DLP) och förklarar varför den generellt sett inte är överförbar till elliptiska kurvor. Efter detta skiftar vi fokus till Pollard's rho och lambda metoder för att lösa ECDLP. Vi förklarar hur cykeldetektering fungerar och hur vi kan lösa ut skalären k ifrån detta. För att utvärdera den praktiska prestandan implementerar vi Pollard's rho i SageMath och utför två tester. Vi undersöker huruvida Washingtons tumregel att cirka tjugo partitioner av delgruppen genererad av P är optimal. För att göra detta genererade vi 50 kurvor över primkroppar med cirka 10^8 element och varierade antalet partitioner s från 2 till 40 stycken. Experimentet visar att tiden för algoritmen att lösa ECDLP minskar drastiskt för varje ökning av s fram till $s \approx 15$, med endast en marginell minskning efter $s \approx 20$. Sedan jämför vi tiden det tar för vår implementerade version av Pollards rho att lösa ECDLP mot SageMaths inbyggda. Detta test visar att SageMaths inbyggda version är betydligt snabbare, vilket troligtvis beror på att SageMath använder cython för beräkningar. Vi avslutar med att jämföra Pollards kangaroo metod mot Pollards rho via de inbyggda funktionerna i SageMath. Detta test visar hur information om att k ligger i ett givet

intervall kan drastiskt förkorta tiden att lösa ECDLP. Om vi vet att $k \in (a, b)$ med $L = b - a$ och $|P| = n$, så observerar vi att för $L = n/log(n)$ så kan tiden att lösa ECDLP halveras. Vidare tester visar en ännu större skillnad mellan funktionerna när storleken på L minskar.

Contents

1 Introduction **9**

2 Fields and Finite Fields **10**

2.1 Modular arithmetic . . . 11
2.2 Relations and cosets . . . 11
2.3 Primitive root . . . 14
2.4 Finite fields . . . 15

3 Elliptic curves **21**

3.1 Point addition on an elliptic curve . . . 23
3.2 The group structure of elliptic curves . . . 25
3.3 Elliptic curves over finite fields . . . 27

4 Discrete logarithm problem (DLP) **29**

4.1 Discrete logarithm problem for finite fields . . . 29
4.1.1 Index calculus method . . . 29
4.1.2 Why the Index calculus method (usually) does not work for elliptic curves . . . 31
4.2 Discrete logarithm problem for elliptic curves . . . 31
4.2.1 Implication of Lagrange's theorem . . . 31
4.2.2 Note . . . 31
4.3 Algorithms to solve ECDLP . . . 32
4.4 Pollard's rho . . . 32
4.4.1 Solving for k . . . 33
4.5 Pollard's lambda . . . 35
4.6 Pollard's kangaroo . . . 35

5 Empirical tests **37**

5.1 Selecting curves for testing the ECDLP . . . 37
5.2 Performance evaluation of Pollard's rho with different partition sizes . . . 37
5.2.1 Implementation in SageMath . . . 37
5.2.2 Experimental Methodology . . . 38
5.2.3 Results and observations . . . 38
5.3 Testing my implementation against SageMath's . . . 39
5.4 Empirical speed-up obtained by Pollard's kangaroo method . . . 39
5.4.1 Results . . . 40

1 Introduction

Elliptic curve cryptography (ECC) has rapidly developed during the last 30 years, becoming more and more widely adopted. It offers robust security with drastically smaller key sizes compared to classical methods such as RSA or Diffie-Hellman. As noted in [3][§6], Washington writes that "a key size of 4096 bits for RSA gives the same level of security as 313 bits in an elliptic curve system", showing the efficiency of ECC. The security of ECC relies on the presumed computational difficulty of the elliptic curve discrete logarithm problem (ECDLP), an adaptation of the classical discrete logarithm problem (DLP). The ECDLP asks us to find the integer k such that $Q = kP$ for given points P and Q on a specified elliptic curve. To get a deeper understanding on how elliptic curve cryptography works and known attacks on the ECDLP, we first need to develop an algebraic foundation.

This thesis begins by detailing the algebra necessary to define and understand elliptic curves and the algorithms that attack the ECDLP. We begin by looking at modular arithmetic, working our way through fields and finite fields until we define what an elliptic curve is in our setting. Defined through whats called the Weierstrass equation, an elliptic curve by our definition, has to fulfill that given two points and their secant line, the secant line intersects the elliptic curve in one unique additional point. This is what allows us to define a group structure from the points of the elliptic curve. We then revisit finite fields to discuss elliptic curves within this abstract but practical setting.

Once the theoretical background is established, we shift our attention towards the DLP and the index calculus method, a sub exponential algorithm for finite multiplicative groups. Here, we examine the structural differences between these groups and elliptic curves, explaining why adapting index calculus to elliptic curves generally fails due to the absence of a meaningful notion of prime factorization for points of the curve.

We then introduce the ECDLP and present some known attacks to break the ECDLP. Here, Pollard's methods; rho, lambda and kangaroo are central. They are all similar methods that are used for different scenarios. These methods all use pseudo-random walks within the elliptic curve group to find the discrete logarithm. Only keeping track of two elements at a time, allows them to keep storage at a minimal for a slight cost in computation. We explore how cycle detection works using Floyd's method and how to calculate the discrete logarithm from a cycle detection. This thesis concludes with three empirical analyses performed in SageMath, exploring practical aspects of Pollard's rho and kangaroo methods. We first test how different partition sizes of the cyclic group $\langle P\rangle$ affect the performance of Pollard's rho, then benchmark our implementation of Pollard's rho against SageMath's built in version and finally demonstrate how Pollard's kangaroo method can offer significant speed up with apriori knowledge about the range of k.

2 Fields and Finite Fields

Before we can explore elliptic curves, we review the notion of a *field*: every coordinate we encounter will lie in some field and every step of the elliptic curve group law relies on the structure of fields. This section follows [2, §22.3] and [1, §1.4 and §7.1]

Definition 2.0.1. A *field* $\mathbb{F}$, is a set that consists of two abelian groups under different operations, $(\mathbb{F},+)$, $(\mathbb{F}\setminus\{0\},\cdot)$ and satisfies the following properties called axioms.

Axioms of a Field

1. **Associativity:** $\forall a,b,c\in F$ it holds that $(a+b)+c=a+(b+c)$ and $(a\cdot b)\cdot c=a\cdot(b\cdot c)$.
2. **Commutativity:** $\forall a,b,c\in F$ it holds that $a+b=b+a$ and $a\cdot b=b\cdot a$.
3. **Distributivity:** $\forall a,b,c\in F$ it holds that $a\cdot(b+c)=ab+ac$ and $(a+b)\cdot c=ac+bc$
4. **Identity:** There exists an element $0\in F$ such that $a+0=a=0+a$ for all $a\in F$. There exists $1\in F$ such that $1\cdot a=a=a\cdot 1$ for all $a\in F$
5. **Inverse:** There exists for both operations a unique element for each $a\in F$ such that $a+(-a)=0$ and $a\cdot a^{-1}=1$. The latter if $a\neq 0$.

Example 2.0.2. The set of all rational numbers

$$\mathbb{Q}=\{\frac{m}{n}\mid m,n\in\mathbb{Z},\ n\neq 0\}$$

with the usual addition and multiplication all five axioms are satisfied. However, the ring of integers

$$\mathbb{Z}=\{\dots,-1,0,1,\dots\}$$

with the usual operations is *not* a field. It satisfies associativity, commutativity, and distributivity and has the additive identity 0 and the multiplicative identity 1. However, only the elements ± 1 have multiplicative inverses in $\mathbb{Z}$, thus it is not a field.

Definition 2.0.3. Let $\mathbb{F}$ be a field with additive identity 0, a non-zero element a in $\mathbb{F}$ is a *zero divisor* if there exists a non-zero element b in $\mathbb{F}$ such that

$$a\cdot b=0$$

Lemma 2.0.4. A field contains no zero-divisors.

Proof. Let F be a field and assume for contradiction, that there are a,b in $\mathbb{F}$ that satisfies

$$a\neq 0,\quad b\neq 0,\quad a\cdot b=0$$

Because $a\neq 0$, by the inverse axiom, a has an inverse a^{-1}. Multiply $a\cdot b=0$ on the left by a^{-1} and use associativity:

$$a^{-1}(a\cdot b)=(a^{-1}a)b=b$$

While the right side becomes:

$$a^{-1}\cdot 0=0$$

Hence $b=0$ which contradicts the assumption. Therefore no such non-zero pair can exist in a field, thus $\mathbb{F}$ has no zero-divisors. $\square$

2.1 Modular arithmetic

Modular arithmetic is a system of arithmetic for integers where numbers "wrap around" after reaching a certain modulus. This section follows [2, §13]

Definition 2.1.1. For an integer $n > 0$, we say that two integers a and b are congruent modulo n, written as

$$a \equiv b \pmod{n},$$

if n divides their difference, i.e., $n \mid (a - b)$.

Example 2.1.2. Consider integers, $\mathbb{Z}$ modulo 5. Then, $12 \equiv 2 \pmod{5}$, since 5 divides $12 - 2 = 10$.

2.2 Relations and cosets

To prepare us for Lagrange's theorem, we introduce the notion of relations on a set and cosets from group theory. This section follows [2, §7 and §12.2].

Definition 2.2.1 (Relation)**.** Let X be a set, a *relation* R on X is a subset

$$R \subseteq X \times X$$

If (x, y) is in R we write $x\,R\,y$.

Definition 2.2.2 (Potential properties of a relation)**.** A relation R on X can be

1. **reflexive** if $x\,R\,x$ for every x in X.
2. **symmetric** if $x\,R\,y$ then $y\,R\,x$ for all x, y in X.
3. **transitive** if $x\,R\,y$ and $y\,R\,z$ then $x\,R\,z$ for all x, y, z in X.

Definition 2.2.3 (Equivalence relation)**.** Let X be a set, an *equivalence relation* $\sim$, is a relation on X that is reflexive, symmetric and transitive. Given an equivalence relation $\sim$ on X and an element x in X, the equivalence class of x is:

$$[x] = \{y \in X : y \sim x\}$$

Lemma 2.2.4. Let X be a set with x, y in X and $\sim$ being an equivalence relation on X. Then the two equivalence classes $[x], [y]$ are either identical or disjoint.

Proof. Assume X is a non-empty set and x, y is in X. We prove the lemma by showing that if $[x]$ and $[y]$ share at least one element, they are identical. Suppose $[x] \cap [y] \neq \emptyset$, then take z in $[x] \cap [y]$, thus $z \sim x$ and $z \sim y$. By the symmetry property $x \sim z$, then the transitive property implies that $x \sim y$. Now take any w in $[x]$, since $w \sim x$ and $x \sim y$, transitivity implies that $w \sim y$, thus w is in $[y]$, so $[x] \subseteq [y]$. The reverse inclusion is proved similarly. This shows that two equivalence classes are either identical or disjoint. □

Theorem 2.2.5. Let X be a set and $\sim$, an equivalence relation on X. Then the distinct equivalence classes

$$[x] = \{y \in X \mid y \sim x\} \qquad x \in X$$

form a partition of X.

Proof. Assume that X is a set and $\sim$ is an equivalence relation on X. Then for any x in X, the reflexive property of the equivalence relation gives $x \sim x$, thus x is in $[x]$. This means that every element in X belongs to (at least) one equivalence class. By lemma 2.2.4, if two classes share at least one element they are equal. Therefore the union of distinct equivalence classes is pairwise disjoint, contains all elements of X and thus, cover all of X. □

Definition 2.2.6 (Cosets)**.** Let G be a group and $H \subseteq G$, then the left and right *coset* are the subsets formed by H multiplied by an element $g \in G$. That is, for the left coset

$$gH = \{x \in G \mid x = gh \quad \text{for some } h \in H\}$$

Example 2.2.7. Let $G = (\mathbb{Z}_6, +)$ and $H = \{0,\ 3\} \subset G$, then H is a subgroup since the identity, $0 \in H$ and the inverse of 3 which is 3 since $3+3 \equiv 0 \pmod 6$ is also in H. Then the left cosets of H are

Table 1: Left cosets of H

g	$g+H$
0	$\{0,3\}$ (same as H)
1	$\{1,4\}$
2	$\{2,5\}$
3	$\{3,0\}$ (same as H)
4	$\{4,1\}$ (same as $1+H$)
5	$\{5,2\}$ (same as $2+H$)

As we can see, not all subsets form a subgroup of G.

Theorem 2.2.8. Let G be a group and $H \subseteq G$. Define the relation $\sim$ on G by:

$$g \sim k \Leftrightarrow g^{-1}k \in H$$

Then $\sim$ is an equivalence relation and its equivalence classes are the left cosets gH.

Proof. Assume G is a group and $H \subseteq G$. Define the relation $\sim$ on G by:

$$g \sim k \Leftrightarrow g^{-1}k \in H$$

First we show that $\sim$ is an equivalence relation by checking the three properties:

- **Reflexive:** For any g in G, $g^{-1}g = e \in H$.
- **Symmetric:** Let g, k be in G, then if $g \sim k$ by definition $g^{-1}k$ is in H. Since subgroups are closed under inverses, $(g^{-1}k)^{-1}$ is also in H. But $(g^{-1}k)^{-1} = k^{-1}g$, which implies that $k \sim g$.
- **Transitive:** Let x, y, z in G with $x \sim y$ and $y \sim z$. This means that $x^{-1}y$ is in H and $y^{-1}z$ is in H. Since H is a subgroup we also have that $x^{-1}yy^{-1}z = x^{-1}z$ is in H, thus $x \sim z$.

Now we show that each equivalence class is a left coset. Fix a g in G. Then the equivalence class of g is

$$[g] = \{\, k \in G \mid g^{-1}k \in H \,\}$$

while the left coset determined by g is

$$gH = \{\, k \in G \mid k = gh \quad \text{for some } h \in H \,\}$$

Take k in $[g]$, then $g^{-1}k$ is in H, call this element h. Since

$$g^{-1}k = h \Leftrightarrow k = gh$$

we also have that k is in gH, thus $[g] \subseteq gH$. The reversed inclusion is proved similarly. Hence we have that

$$[g] = gH$$

□

Lemma 2.2.9. Let G be a group with $H \subseteq G$ a finite subgroup, then $|gH| = |Hg| = |H|$.

Proof. Let G be a group and $H \subseteq G$, then take any element $g \in G$ and form the left coset gH. If $|gH| \neq |H|$ then at least two elements in the coset would have to be equal. Assume for contradiction that

$$gh_1 = gh_2 \quad \text{for some} \quad h_1 \neq h_2 \quad \text{with} \quad h_1, h_2 \in H$$

Then since G is a group, g has an inverse which gives

$$g^{-1}gh_1 = g^{-1}gh_2 \quad \Longrightarrow \quad h_1 = h_2$$

disproving this statement. The same argument can be applied to right cosets. It follows that

$$|gH| = |Hg| = |H|$$

□

Theorem 2.2.10 (Lagrange's theorem)**.** Lagrange's theorem states that if G is a finite group of order $|G| = n$, then the order of any subgroup $H \subseteq G$ must divide n.

Proof. Let G be a group and $H \subseteq G$, with $|G| = n$ and $|H| = m$. Then by theorem 2.2.8 the distinct left cosets of H are equivalence classes and by theorem 2.2.5, they form a partition of G. Assume that we have k distinct left cosets, then:

$$G = g_1H \cup g_2H \cdots \cup g_kH$$

Now we can take the cardinality

$$|G| = |g_1H \cup g_2H \cdots \cup g_kH|$$

Since by assumption these cosets are all disjoint we get that

$$|G| = |g_1H| + |g_2H| \cdots + |g_kH|$$

By lemma 2.2.9, all left cosets of H have the same cardinality as H so

$$|G| = |H| + |H| \cdots + |H| = k \cdot m$$

Thus the order of H divides the order of G. □

2.3 Primitive root

We continue by introducing the notion of a *primitive root*. This section mainly builds towards the index calculus method, that will be discussed later. This section follows [1, §13.1]

Definition 2.3.1. Take any positive integer n and define $\mathbb{Z}_n^\times = \{a \mid 1 \leq a \leq n-1,\ gcd(a,n) = 1\}$ to be the group of co-primes to n (if n is prime then this includes all integers up until $n-1$). The set $\mathbb{Z}_n^\times$ together with multiplication forms a finite group and the size is given by $|\mathbb{Z}_n^\times| = \phi(n)$ (Euler's phi function).

Definition 2.3.2. Let $a \in \mathbb{Z}_n^\times$, then the *multiplicative order* of $a \pmod n$, denoted $ord_n(a)$, is the smallest positive integer $1 \leq k$ such that $a^k \equiv 1 \pmod n$.

Lemma 2.3.3. Let p be a prime number, then $ord_p(a)$ for any $a \in \mathbb{Z}_p^\times$ divides $p-1$.

Proof. A proof follows from Lagrange's theorem. Since p is prime $|\mathbb{Z}_p^\times| = p-1$, by Lagrange's theorem, take $a \in \mathbb{Z}_p^\times$, then if $|a| = m$ we must have that $m \mid p-1$. □

Definition 2.3.4. A *primitive root modulo p* (with p prime) for the group $\mathbb{Z}_p^\times$, is an integer $g \in \mathbb{Z}_p^\times$ such that $ord_p(g) = p-1$. Then g is a generator for the group $\mathbb{Z}_p^\times$, which means that the powers $g, g^2, \ldots, g^{p-1}$ run through all non-zero residues, so every integer is congruent to 0 or some power of g.

Example 2.3.5. Let p = 5, then $\mathbb{Z}_p^\times = \{1,2,3,4\}$ with 2 being a *primive root*. Since $2^4 = 16 \equiv 1 \pmod 5$, we can express each element as a power of 2

$$\begin{aligned} 2^1 &\equiv 2 \pmod 5 \\ 2^2 &\equiv 4 \pmod 5 \\ 2^3 &\equiv 3 \pmod 5 \\ 2^4 &\equiv 1 \pmod 5 \end{aligned}$$

Thus we clearly have:

$$\{2^1, 2^2, 2^3, 2^4\} \equiv \{2,4,3,1\} \pmod 5$$

Lemma 2.3.6. If p is a prime and g a primitive root for the multiplicative group $\mathbb{F}_p^\times$ Then:

$$g^{(p-1)/2} \equiv -1 \pmod p$$

Proof. We have that since g is a primitive root, g generates the finite group $\mathbb{F}_p^\times$ and $g^{p-1} \equiv 1 \pmod p$,then

$$g^{p-1} = (g^{\frac{p-1}{2}})^2 \equiv 1 \pmod p$$

Thus $g^{\frac{p-1}{2}}$ must be one of two square roots of 1 in the group, namely $+1$ or -1. If

$$g^{\frac{p-1}{2}} \equiv 1 \pmod p$$

then the order of g would divide $\frac{p-1}{2}$ contradicting that $ord_p(g) = p-1$. Hence the only possibility is that

$$\boxed{g^{\frac{p-1}{2}} \equiv -1 \pmod p}$$

□

2.4 Finite fields

Most familiar fields, $\mathbb{Q},\mathbb{R},\mathbb{C}$,are infinite, for cryptographic applications we instead work over finite fields that contain only finitely many elements. We now introduce *finite fields* and record some basic facts that will be needed later. This section follows [1, §9.2 and §13], [3, Appendix C] and [4, Chapter 1 and 2].

Definition 2.4.1. A *finite field* is a field $(\mathbb{F},+,\cdot)$ with an underlying set consisting of a finite number of elements. If $|\mathbb{F}| = q < \infty$ then $\mathbb{F}$ is called a finite field and is often denoted $\mathbb{F}_q$.

Finite fields can be categorized into two types:

- **Prime fields:** These occur when q is a prime number p. A prime field $\mathbb{F}_p$ consists of the integers modulo p. That is, the set $\{0,1,2,\dots,p-1\}$ with addition and multiplication performed modulo p.
- **Extension fields:** These occur when $q = p^n$ for some prime p and integer $n > 1$. These fields are built from prime fields by extending them using polynomials.

Example 2.4.2. (Finite field): The set $\mathbb{F}_3 = \{0,1,2\}$ with addition and multiplication taken modulo 3 is a finite field. Here 0 is the *additive identity* because $a+0 = 0+a = a$ for every $a \in \mathbb{F}_3$, and 1 is the *multiplicative identity* because $a\cdot 1 = 1\cdot a = a$ for every $a \in \mathbb{F}_3$. For instance,

$$2+2 = 4 \equiv 1 \pmod 3, \qquad 2\cdot 2 = 4 \equiv 1 \pmod 3.$$

Definition 2.4.3 (Polynomials in a field)**.** Given a field $\mathbb{F}$, a *polynomial* $f(x)$ is an expression of the form

$$f(x) = \sum_{i=0}^{n} a_i x^i \quad a_i \in \mathbb{F}, x \notin \mathbb{F} \quad \text{for some } n \in \mathbb{N}$$

With x being treated as a formal symbol, thus a polynomial as in the usual sense only with coefficients in $\mathbb{F}$. Operations are performed according to the usual definition for addition and multiplication.

Definition 2.4.4 (Polynomial ring)**.** The elements of a field $\mathbb{F}$ together with polynomials form a ring, denoted $\mathbb{F}[x]$ and is called the polynomial ring over $\mathbb{F}$. We will soon see how we can turn this into a field extention.

Definition 2.4.5 (Irreducible polynomial)**.** A polynomial $p(x)$ in a polynomial ring $\mathbb{F}[x]$ with $deg(p) \geq 1$ is *irreducible* over $\mathbb{F}$ if it does not factor over $\mathbb{F}[x]$. That is, $p(x)$ can **not** be written as

$$p(x) = a(x)\cdot b(x) \quad a(x), b(x) \in \mathbb{F}[x]$$

with $deg(a), deg(b) < deg(p)$ and neither $a(x)$ nor $b(x)$ being a non-zero constant.

Definition 2.4.6 (Residue class)**.** Given a polynomial ring $\mathbb{F}[x]$ and a polynomial $f(x)$ in $\mathbb{F}[x]$ with degree $n \geq 1$. The residue polynomial ring $\mathbb{F}[x]/f(x)$ is comprised of residue classes $[g]$ for polynomials $g(x)$ in $\mathbb{F}[x]$, where each residue class has a representative $r(x)$. That is the remainder of $g(x)$ divided by $f(x)$, which is simply saying that

$$g(x) \equiv r(x) \pmod{f(x)}$$

Theorem 2.4.7. Let $\mathbb{F}$ be a field and $f(x)$ in $\mathbb{F}[x]$ be a polynomial, then $\mathbb{F}[x]/f(x)$ is a field if and only if $f(x)$ is irreducible.

Proof. The proof of this theorem is showed in [4, Theorem 1.61] $\square$

Theorem 2.4.8. Let $\mathbb{F}_p$ be a finite prime field, then there exists a monic irreducible polynomial $f(x)$ in $\mathbb{F}_p[x]$ of degree $n > 1$. It follows that $\mathbb{F}_p[x]/f(x)$ is a field.

Proof. We will not prove this theorem here, if the reader is interested we refer to [4, Theorem 1.87] $\square$

Proposition 2.4.9. Let $f(x)$ in $\mathbb{F}_p[x]$ be a monic irreducible polynomial with degree $n > 1$ and let $\alpha := [x]$ be the residue class of x in $\mathbb{F}_p[x]/f(x)$. Then $\mathbb{F}_{p^n}$ is constructed by $\mathbb{F}_p[x]/f(x)$ and every element of $\mathbb{F}_{p^n}$ can be uniquely expressed as

$$\left\{a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{n-1}\alpha^{n-1} \mid a_i \in \mathbb{F}_p\right\}.$$

Hence

$$\boxed{\mathbb{F}_p[x]/f(x) \cong \mathbb{F}_{p^n}}$$

Proof. Let $\mathbb{F}_p$ be a finite prime field and $f(x)$ in $\mathbb{F}_p[x]$ a monic irreducible polynomial with degree $n > 1$. From theorem 2.4.7 we know that $\mathbb{F}_p[x]/f(x)$ is a field. Take any polynomial $g(x)$ in $\mathbb{F}_p[x]$ then $g(x)$ has a unique representative $r(x)$ with $deg(r(x)) < deg(f(x))$. If this representative was not unique, then two polynomials r_1, r_2 with degree less than n satisfies

$$r_1 \equiv r_2 \pmod{f(x)}$$

then

$$f(x) \mid (r_1 - r_2) \quad \text{but } deg(r_1 - r_2) < deg(f), \text{ so } r_1 = r_2$$

So each representative $r(x)$ is unique. Write

$$r(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}, \quad a_i \in \mathbb{F}_p$$

Since $\alpha = [x]$ in $\mathbb{F}_p[x]/f(x)$ we have that

$$[g(x)] = r(x) = a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1}$$

where any choice of a_i gives a polynomial of degree less than n. There are p possibilities for each coefficient a_i so the set above has p^n elements. The existence of this field is guaranteed by theorem 2.4.8 and later on theorem 2.4.16 tells us that this field is unique (up to isomorphism). Hence

$$\mathbb{F}_p[x]/f(x) \cong \mathbb{F}_{p^n}$$

$\square$

Example 2.4.10. (Extension field): The field $\mathbb{F}_{16}$ is not a prime field, it contains 16 elements so it must be an extension of $\mathbb{F}_2$. One classical way of constructing it is as the set

$$\mathbb{F}_{16} = \{0, 1, \alpha, 1+\alpha, \alpha^2, 1+\alpha^2, \alpha+\alpha^2, 1+\alpha+\alpha^2, \alpha^3, 1+\alpha^3, 1+\alpha^2+\alpha^3,$$
$$1+\alpha+\alpha^3, \alpha+\alpha^3, \alpha^2+\alpha^3, \alpha+\alpha^2+\alpha^3, 1+\alpha+\alpha^2+\alpha^3\}$$

Where α is a root of the irreducible polynomial x^4+x+1 over $\mathbb{F}_2$, thus

$$\mathbb{F}_{16} \cong \mathbb{F}_2[x]/(x^4+x+1)$$

When working in the multiplicative group $\mathbb{F}_{16}^{\times}$ we multiply the polynomials modulus x^4+x+1 (and reduce coefficients modulus 2). For example

$$\alpha^4 \equiv \alpha+1 \pmod{x^4+x+1},$$

so

$$\alpha^5 = \alpha\cdot\alpha^4 \equiv \alpha(1+\alpha) = \alpha+\alpha^2 \pmod{x^4+x+1}$$

and then

$$\alpha^{10} \equiv (\alpha+\alpha^2)^2 = \alpha^2+\alpha^4 \equiv 1+\alpha+\alpha^2 \pmod{x^4+x+1}$$

Addition is simply taking each coefficient modulo 2, for example

$$(\alpha+\alpha^2)+(\alpha^2+\alpha^3) = \alpha+2\alpha^2+\alpha^3 \equiv \alpha+\alpha^3 \pmod 2$$

Definition 2.4.11. Characteristic of a field: The *characteristic* of a field, denoted $char(\mathbb{F})$ is the smallest integer p such that

$$\underbrace{1+\cdots+1}_{p \text{ times}} = 0 \text{ in } \mathbb{F}$$

with 1 denoting the identity of $\mathbb{F}$, if such an integer exists, otherwise $\mathbb{F}$ is not finite and $char(\mathbb{F}) = 0$.

Example 2.4.12. The field $\mathbb{F}_3$ has *characteristic* 3, since 3 is a prime number. The field $\mathbb{F}_{16}$ has *characteristic* 2 since it is an extension of the prime field $\mathbb{F}_2$.

Lemma 2.4.13. If $\mathbb{F}_q$ is a finite field then $char(\mathbb{F}_q) = p$ for some prime p.

Proof. Since $\mathbb{F}_q$ is a finite field, $char(\mathbb{F}_q) \neq 0$. Assume for contradiction that $char(\mathbb{F}_q) = n$ with n not being prime, then n is the smallest positive integer such that

$$\underbrace{1+\cdots+1}_{n \text{ times}} = n\cdot 1 = 0 \text{ in } \mathbb{F}_q$$

Since n was assumed to be composite, let $n = a\cdot b$ with $1 < a, b < n$, then

$$0 = (n\cdot 1) = (a\cdot 1)(b\cdot 1)$$

We then have two non-zero elements whose product is zero, hence they are zero-divisors, something a field can not have by lemma 2.0.4. $\square$

Lemma 2.4.14. $\mathbb{Z}_p$ is a field $\iff$ p is a prime

Proof.

1. **p prime $\implies$ $\mathbb{Z}_p$ is a field.** It is tedious but easy to show that $(\mathbb{Z}_p, +)$ is an abelian group, so we only show that $(\mathbb{Z}_p^{\times}, \cdot)$ also is an abelian group. We show this by proving that every element has an inverse, since it is clear that $(\mathbb{Z}_p^{\times}, \cdot)$ is closed under multiplication and that the identity, distributivity, associativity and commutativity axioms hold. Fix an a in $(\mathbb{Z}_p^{\times}, \cdot)$ and define a map

$$\varphi : \mathbb{Z}_p^{\times} \longrightarrow \mathbb{Z}_p^{\times} \qquad \varphi(x) = a\cdot x \pmod p$$

This map is *injective* because if $\varphi(x_1) = \varphi(x_2)$ then $a \cdot x_1 \equiv a \cdot x_2 \pmod{p}$, so $p \mid a(x_1 - x_2)$, since p is a prime and since $a \not\equiv 0 \pmod{p}$ we obtain that $x_1 \equiv x_2 \pmod{p}$. A map between two finite sets of the same size that is injective must be a bijection. This means that there exists an element b in $\mathbb{Z}_p$ such that

$$b \cdot a \equiv 1 \pmod{p}$$

Thus b is the multiplicative inverse of a. This ensures that every non-zero element has a multiplicative inverse. So $(\mathbb{Z}_p^\times, \cdot)$ is indeed an abelian group which proves that $\mathbb{Z}_p$ is a field.

2. n **composite** $\Longrightarrow$ $\mathbb{Z}_n$ **is not a field.** Here we prove the converse of the implication and thus assume that n is not a prime. Then since n is a composite number we can express n as $n = m \cdot k$ with $1 < m, k < n$. Then in $\mathbb{Z}_n$ we have that

$$m \cdot k \equiv 0 \pmod{n} \quad \text{while } m \not\equiv 0 \text{ and } k \not\equiv 0$$

So m and k are non-zero elements whose product is zero, hence they are zero-divisors, something a field can not have by lemma 2.0.4. Thus

$$\boxed{\mathbb{Z}_p \text{ is a field} \quad \Longleftrightarrow \quad p \text{ is a prime}}$$

□

Theorem 2.4.15. Let $\mathbb{F}_q$ be a finite field, then

$$|\mathbb{F}_q| = q = p^n$$

Where $p = char(\mathbb{F}_q)$ and n is a positive integer.

Proof. Let $\mathbb{F}_q$ be a finite field. From lemma 2.4.13 we know that $char(\mathbb{F}_q) = p$ for some minimal prime p. Because of this we have that $1_{\mathbb{F}_q}$ added to itself p times is zero, while $1_{\mathbb{F}_q}$ added to itself less times, is not. So the set

$$S = \{0,\ 1,\ 1+1,\ 1+1+1,\ \ldots,\ (p-1)\cdot 1_{\mathbb{F}_q}\} \subset \mathbb{F}_q$$

has p distinct elements. Now define the map

$$\varphi : S \longrightarrow \mathbb{Z}_p \qquad \varphi\big(x \cdot 1_{\mathbb{F}_q}\big) = x \pmod{p}$$

We will show that φ is an isomorphism. To begin, we show that φ is a homomorphism. Take any $x \cdot 1_{\mathbb{F}_q}, y \cdot 1_{\mathbb{F}_q}$ in S Let

$$x \cdot 1_{\mathbb{F}_q} + y \cdot 1_{\mathbb{F}_q} = (x+y)\cdot 1_{\mathbb{F}_q} = s \cdot 1_{\mathbb{F}_q}$$

then if

$$x + y > p-1 \quad \text{then } x \cdot 1_{\mathbb{F}_q} + y \cdot 1_{\mathbb{F}_q} = (x+y-p)\cdot 1_{\mathbb{F}_q} \text{ in } S$$

This implies that

$$(x+y) \equiv s \pmod{p}$$

So then

$$\varphi(x \cdot 1_{\mathbb{F}_q} + y \cdot 1_{\mathbb{F}_q}) = \varphi(s \cdot 1_{\mathbb{F}_q}) = s \pmod{p}$$

shows that

$$\varphi(x \cdot 1_{\mathbb{F}_q} + y \cdot 1_{\mathbb{F}_q}) = s \equiv x + y = \varphi(x \cdot 1_{\mathbb{F}_q}) + \varphi(y \cdot 1_{\mathbb{F}_q}) \pmod{p}$$

Similarly for multiplication, for $0 \leq x, y \leq p - 1$ we have that

$$(x \cdot 1_{\mathbb{F}_q})(y \cdot 1_{\mathbb{F}_q}) = (xy) \cdot 1_{\mathbb{F}_q} = (xy \bmod p) \cdot 1_{\mathbb{F}_q} \equiv s \cdot 1_{\mathbb{F}_q} \text{ in } S$$

then

$$\varphi((x \cdot 1_{\mathbb{F}_q}) \cdot (y \cdot 1_{\mathbb{F}_q})) = \varphi(s \cdot 1_{\mathbb{F}_q}) = s$$

which implies that

$$\varphi((x \cdot 1_{\mathbb{F}_q}) \cdot (y \cdot 1_{\mathbb{F}_q})) = s \equiv x \cdot y = (x \cdot 1_{\mathbb{F}_q}) \cdot (y \cdot 1_{\mathbb{F}_q}) = \varphi(x \cdot 1_{\mathbb{F}_q}) \cdot \varphi(y \cdot 1_{\mathbb{F}_q}) \pmod{p}$$

So φ is a homomorphism. Now take any $x_1 \cdot 1_{\mathbb{F}_q}, x_2 \cdot 1_{\mathbb{F}_q}$ in S, if

$$\varphi(x_1 \cdot 1_{\mathbb{F}_q}) = \varphi(x_2 \cdot 1_{\mathbb{F}_q})$$

then that is equivalent to

$$x_1 = x_2$$

So φ is injective. Since

$$|S| = p = |\mathbb{Z}_p|,$$

the injective homomorphism φ must also be surjective, thus it is an isomorphism. Hence

$$(S, +, \cdot) \cong \mathbb{Z}_p$$

We know from lemma 2.4.14, that $\mathbb{Z}_p$ is a field for any prime p, hence every finite field $\mathbb{F}_q$ has $\mathbb{Z}_p$ as a subfield, for some prime p. Since $\mathbb{Z}_p$ is a field, we can scale any element of $\mathbb{F}$ by elements of $\mathbb{Z}_p$ and view $\mathbb{F}_q$ as a vector space over $\mathbb{Z}_p$. Since $\mathbb{F}_q$ is finite, this vector space must have finite dimension, say $\dim_{\mathbb{Z}_p} \mathbb{F}_q = n$. Now we can choose a basis of $\mathbb{F}_q$ over $\mathbb{Z}_p$, say $\alpha_1, \ldots, \alpha_n$. Then every element x in $\mathbb{F}_q$ can uniquely be expressed as a linear combination of this base:

$$x = b_1\alpha_1 + b_2\alpha_2 \cdots + b_n\alpha_n \quad \text{with } b_i \in \mathbb{Z}_p$$

Since the coefficients b_i belong to $\mathbb{Z}_p$, there are p possibilities for each b_i. This means that there are exactly p^n possible combinations and hence p^n distinct elements in $\mathbb{F}_q$. Thus

$$|\mathbb{F}_q| = q = p^n$$

□

Theorem 2.4.16. For every prime p and natural number n, there exists a unique (up to isomorphism) field $\mathbb{F}_{p^n}$ with $|\mathbb{F}_{p^n}| = p^n$.

Proof. We start by proving the case when $n = 1$. We know from lemma 2.4.14 that a field exists for every prime p, namely $\mathbb{Z}_p$. We must now show that it is unique (up to isomorphism). Let $\mathbb{F}_p$ be a field of order p and assume for contradiction that it is not isomorphic to $\mathbb{Z}_p$. Then from the proof of theorem 2.4.15 we know that

$$S = \{0, 1, \ldots, (p-1) \cdot 1_{\mathbb{F}_p}\} \subseteq \mathbb{F}_p$$

Since we also saw that

$$|S| = p = |\mathbb{F}_p| \quad \text{and that} \quad S \cong \mathbb{Z}_p$$

It follows that S must be the whole of $\mathbb{F}_p$ and that for every prime p

$$\mathbb{F}_p \cong \mathbb{Z}_p$$

For $n > 1$ it follows from theorem 2.4.8 that an irreducible monic polynomial $f(x)$ of degree $n > 1$ always exists and proposition 2.4.9 shows how $\mathbb{F}_{p^n}$ is constructed, however for the uniqueness we refer the reader to [4, Theorem 2.5] and its proof. $\square$

3 Elliptic curves

There are many elliptic curves, for cryptographic applications, we impose certain restrictions on the type of curves that we work over. Before defining what an elliptic curve in our application is, we need to introduce some standard results. This section mainly follows [3][§2.1].

Definition 3.0.1 (Weierstrass equation)**.** Fix a field $\mathbb{F}$ with $char(\mathbb{F}) \neq 2,3$, then the *Weierstrass equation* is a monic cubic polynomial in two variables defined as

$$y^2 = x^3 + Ax + B$$

where $A, B \in \mathbb{F}$. For this equation to be able to define an elliptic curve in our applications, we require that the polynomial does not have any repeated roots.

Definition 3.0.2 (Discriminant)**.** The *discriminant* for the Weierstrass equation is defined as

$$\Delta = -16(4A^3 + 27B^2)$$

Definition 3.0.3. If the discriminant $\Delta \neq 0$ then the Weierstrass curve defined over a field $\mathbb{F}$ is called *non-singular.*

Theorem 3.0.4. If the discriminant $\Delta \neq 0$ then the cubic polynomial $y^2 = x^3 + Ax + B$ only has distinct roots in $\mathbb{F}$

Proof. Let r, s, t be roots to $f(x) = x^3 + Ax + B$, then

$$f(x) = (x-r)(x-s)(x-t) = x^3 - (r+s+t)x^2 + (rs+rt+st)x - rst$$

Comparing coefficients with $x^3 + Ax + B$, whose x^2-coefficient is 0, yields three equations

$$\sigma_1 = r+s+t = 0, \quad \sigma_2 = rs+rt+st = A, \quad \sigma_3 = rst = -B \tag{1}$$

Since we want all roots to be unique, this is equivalent to

$$(r-s)^2(r-t)^2(s-t)^2 \neq 0 \tag{2}$$

Expanding this, we can combine from (2) the classical discriminant formula for a monic cubic

$$(r-s)^2(r-t)^2(s-t)^2 \;=\; \sigma_1^2\sigma_2^2 - 4\sigma_2^3 - 4\sigma_1^3\sigma_3 - 27\sigma_3^2 + 18\sigma_1\sigma_2\sigma_3 \tag{3}$$

Insert (1) into (3), since $\sigma_1 = 0$, all terms containing σ_1 vanish, leaving

$$(r-s)^2(r-t)^2(s-t)^2 = -4A^3 - 27B^2$$

Multiplying by the normalizing factor -16 gives

$$-16(r-s)^2(r-t)^2(s-t)^2 = -16(4A^3 + 27B^2) = \Delta \tag{4}$$

Because the left side of (4) is a product of squares, it vanishes *iff* at least two of the roots are equal, thus

$$\Delta \neq 0 \quad \Leftrightarrow \quad (r, s, t) \text{ are pairwise distinct}$$

□

Before defining an elliptic curve, we first need to introduce an extra element called the *point at infinity.*

Definition 3.0.5. Fix a field $\mathbb{F}$ with $char(\mathbb{F}) \neq 2,3$ and choose $A, B \in \mathbb{F}$ with non-zero discriminant $\Delta = -16(4A^3 + 27B^2) \neq 0$. The (affine) *Weierstrass curve* is

$$y^2 = x^3 + Ax + B$$

To make it into an elliptic curve and later to obtain a well-behaved group law, we adjoin one extra point, denoted ∞ and call it the *point at infinity.* Strictly speaking, ∞ originates from viewing the curve in *projective coordinates*, where every affine line acquires a "point at infinity" and all vertical lines meet in the single projective point $(0:1:0)$ [3, §2.3]. Expanding on theory behind projective mapping is not something we will do in this paper. Instead we treat ∞ as a formal symbol, obeying two simple rules:

- A line "passes through" ∞ exactly when it is vertical.
- In the group law on $E(\mathbb{F})$, the point ∞ acts as the identity.

Intuitively you may picture ∞ as a single point where the upper and lower ends of the y-axis are glued together. This ensures that every vertical line meets the curve at one additional point ∞, thus any two distinct vertical lines have exactly one common point on the curve.

Definition 3.0.6 (Elliptic curve)**.** Let $\mathbb{F}$ be a field with $char(\mathbb{F}) \neq 2,3$. Select coefficients A, B in $\mathbb{F}$ so that the discriminant

$$\Delta = -16(4A^3 + 27B^2) \neq 0$$

Then the *elliptic curve* over $\mathbb{F}$ is the set of points

$$E(\mathbb{F}) = \left\{(x, y) \in \mathbb{F} \times \mathbb{F} : y^2 = x^3 + Ax + B\right\} \cup \{\infty\}$$

Where ∞ is the *point of infinity.*

Example 3.0.7. Consider $E : y^2 = x^3 + 2x + 2$ over $\mathbb{F}_5$. Its discriminant is

$$\Delta = -16(4 \cdot 2^3 + 27 \cdot 2^2) = -16(32 + 108) = -2240 \equiv 0 \pmod 5$$

Since $\Delta \equiv 0 \pmod 5$ this curve is singular over $\mathbb{F}_5$ and is thus not an elliptic curve in our sense. However, slightly modifying the equation to $y^2 = x^3 + 2x + 1$ instead yields:

$$\Delta = -16(4 \cdot 2^3 + 27 \cdot 1^2) = -16(32 + 27) = -944 \equiv 1 \pmod 5$$

Since this discriminant is nonzero modulo 5, the modified curve is a non-singular elliptic curve.

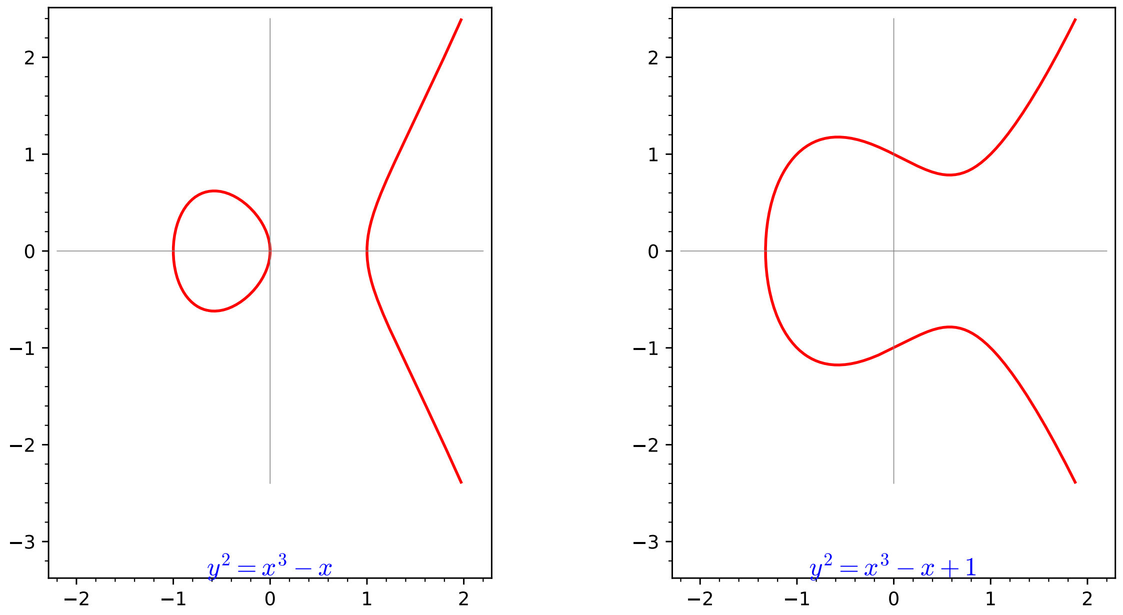

Figure 1: Graph of two non-singular elliptic curves. Left: $y^2 = x^3 - x$ (three real roots). Right: $y^2 = x^3 - x + 1$ (one real root).

3.1 Point addition on an elliptic curve

Fix a field $\mathbb{F}$, of characteristic $char(\mathbb{F}) \neq 2,3$ and a non-singular E in Weierstrass form. Then we can define an *addition law* on the points of the curve. This section follows [3, §2.2].

Definition 3.1.1. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on E, then the addition of points is defined as taking the line L between P_1 and P_2. As we will see this will intersect E in one additional point (x_3, y_3'). We then define y_3 to be the reflection of this point across the x-axis, that is taking $y_3 := -y_3'$. This results in several scenarios depending on the coordinates of P_1 and P_2:

1. **If** $x_1 \neq x_2$, take the secant line L through P_1 and P_2 with slope m defined as

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

The equation for the secant line is

$$L : y = m(x - x_1) - y_1$$

If we substitute y from L into the Weierstrass equation we get

$$(m(x - x_1) + y_1)^2 = x^3 + Ax + B$$

We can rearrange this to the monic cubic

$$g(x) = x^3 - m^2x^2 + (A + 2y_1m)x + \cdots = 0 \tag{1}$$

Since both (x_1, y_1) and (x_2, y_2) lie on E, x_1 and x_2 must be roots of g. Since we have a cubic there can only be one additional root x_3. For a monic cubic $x^3 +$

$a_2x^2 + a_1x + a_0$, the sum of its roots equals $-a_2$. In (1) the x^2-coefficient is $-m^2$, hence

$$x_1 + x_2 + x_3 = m^2 \implies \boxed{x_3 = m^2 - x_1 - x_2} \tag{2}$$

Now we can insert x_3 in the equation for L:

$$y_3' = m(x_3 - x_1) + y_1$$

Then since we defined y_3 to be reflected across the x-axis, we get

$$\boxed{y_3 = m(x_1 - x_3) - y_1}$$

An example of this can be seen in figure 2, picture 1, where P_3 is the reflection of the third point that the secant intersects E with.

2. **If** $x_1 = x_2$ **but** $y_1 \neq y_2$, then the line L through P_1 and P_2 is vertical and does not intersect E in a third point so we set

$$P_1 + P_2 = \infty$$

An example of this can be seen in figure 2, picture 2, where P_1 and P_2 have the same x-coordinate, thus the secant line is vertical.

3. **(Doubling) If** $P_1 = P_2$ **and** $y_1 \neq 0$, we are taking the tangent line to the point $P = P_1 = P_2$. Through implicit differentiation we can derive the slope as

$$m = \frac{3x_1^2 + A}{2y_1}$$

Similarly to case 1, we can then write the equation for L as

$$L: y = m(x - x_1) + y_1$$

With the same reasoning as above we get that

$$\boxed{x_3 = m^2 - 2x_1}$$

Inserting x_3 into the equation for L gives

$$y_3' = m(x_3 - x_1) + y_1$$

Thus

$$\boxed{y_3 = m(x_1 - x_3) - y_1}$$

An example of this can be seen in figure 2, picture 3, taking the tangent line to P_1 and reflecting the second intersection point to obtain P_2.

4. **If** $P_1 = P_2$ **and** $y_1 = 0$ then the tangent is vertical, thus giving us

$$2P = \infty$$

An example of this can be seen in figure 2, picture 4, where $P_1 = (x, 0)$ thus the tangent is vertical.

5. **If** $P_1 = (x, y)$ **and** $P_2 = \infty$ then the line is vertical and will intersect E in the reflection of P_1 across the x-axis. After reflecting across the x-axis again we end up back at P_1, thus

$$P + \infty = P$$

6. **If** $P_1 = P_2 = \infty$ then we define addition as $P_1 + P_2 = \infty + \infty = \infty$.

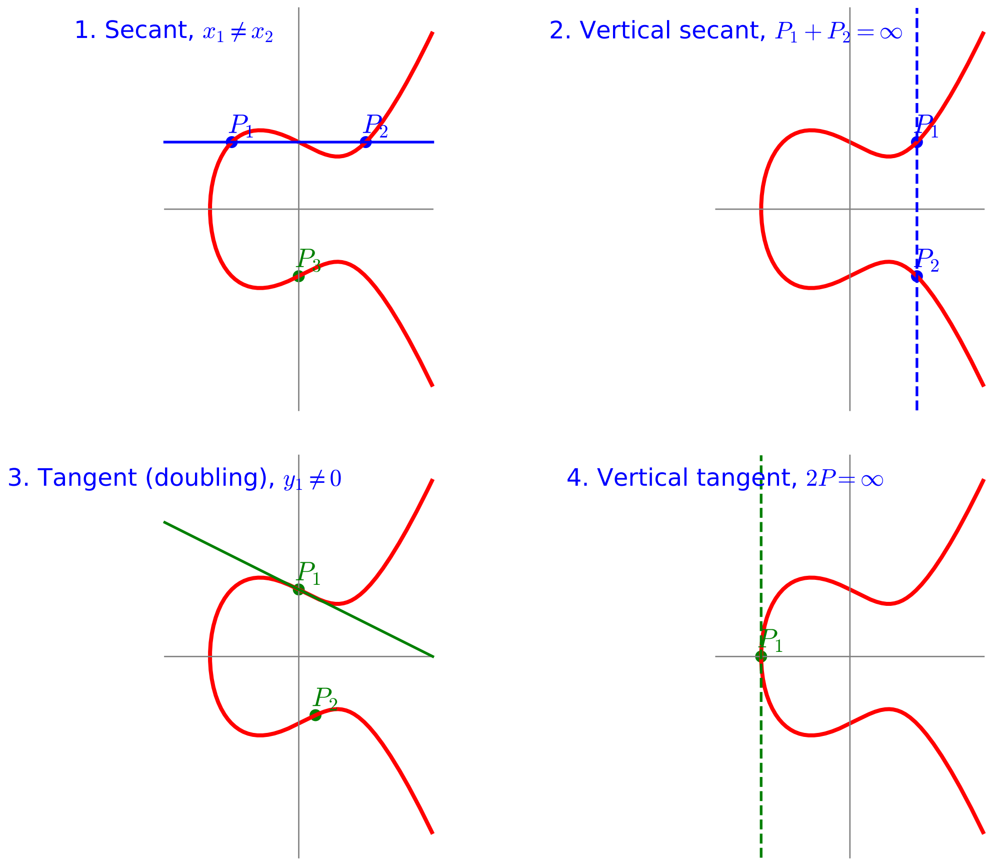
1. Secant, $x_1 \neq x_2$
P_1
P_2
P_3
2. Vertical secant, $P_1 + P_2 = \infty$
P_1
P_2
3. Tangent (doubling), $y_1 \neq 0$
P_1
P_2
4. Vertical tangent, $2P = \infty$
P_1

Figure 2: Examples of point addition on an elliptic curve

Example 3.1.2. On $E : y^2 = x^3 + x + 1$ over $\mathbb{F}_5$, add $P = (0, 1)$ and $Q = (2, 1)$. The slope is:

$$m = \frac{1-1}{2-0} = 0$$

Then:

$$x_3 = m^2 - x_P - x_Q = 0 - 0 - 2 \equiv 3 \pmod 5, \quad y_3 = m(x_P - x_3) - y_P = -1 \equiv 4 \pmod 5.$$

$$\text{Thus, } P + Q = (3, 4).$$

3.2 The group structure of elliptic curves

The previous subsection gave explicit addition formulas for pairs of points on a elliptic curve in Weierstrass form. An important result of this is that these rules satisfies all four axioms for an abelian group, turning the set of points on an elliptic curve into an abelian group [3, Theorem 2.1].

Theorem 3.2.1 (Group theorem elliptic curves)**.** Fix a field $\mathbb{F}$, of characteristic $char(\mathbb{F}) \neq 2,3$ and a non-singular elliptic curve, E, in Weierstrass form. Then the the addition of points, as previously described, on the elliptic curve $E(\mathbb{F})$ satisfies the following properties:

1. **commutative:** $P_1 + P_2 = P_2 + P_1 \quad \forall P_1, P_2 \in E$.
2. **Existence of identity:** $P + \infty = P \quad \forall P \in E$.
3. **Existence of inverses:** $\forall P \in E, \quad \exists P' \in E$ such that $P + P' = \infty$.
4. **Associativity:** $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \quad \forall P \in E$.

Thus the points on $E(\mathbb{F})$ form an abelian group with ∞ as the identity element.

Proof.

1. **Commutative:**

$$\text{Let } P = (x_1,\ y_1) \quad \text{and} \quad Q = (x_2,\ y_2) \quad \text{with} \quad P \neq \pm Q$$

then the slope m is symmetric since for

$$P + Q,\ m = \frac{y_2 - y_1}{x_2 - x_1} \text{ and for } Q + P,\ m = \frac{y_1 - y_2}{x_1 - x_2} = \frac{(-1)\cdot(y_2 - y_1)}{(-1)\cdot(x_2 - x_1)}$$

We get that $x_3 = m^2 - x_1 - x_2$ which is the same for $P + Q$ and $Q + P$ since m is the same. For $P + Q$ call the $y-$value $y_3$and for $Q + P$ call it y_3', then

$$y_3 = m(x_1 - x_3) - y_1 \text{ and } y_3' = m(x_2 - x_3) - y_2$$

their difference is

$$y_3' - y_3 = m(x_2 - x_3) - y_2 - (m(x_1 - x_3) - y_1) = m(x_2 - x_1) - (y_2 - y_1)$$

since by definition of m

$$y_2 - y_1 = m(x_2 - x_1)$$

we get that

$$y_3' - y_3 = m(x_2 - x_1) - m(x_2 - x_1) = 0 \implies y_3' = y_3$$

If $P = Q$ then commutativity obviously holds since $P + P = P + P$. If $P = (x,\ y)$ and $Q = -P$ then $Q = (x, -y)$ and $P + Q = Q + P = \infty$ by definition. Similarly if $P = (x, y)$ and $Q = \infty$ then by definition of ∞, $P + \infty = \infty + P = P$.

2. **Existence of identity:** Since by definition ∞ is the identity element, $P + \infty = P \quad \forall P \in E$.

3. **Existence of inverses:** If $P = (x, y)$ then we let P' be the reflection of P on the x-axis, thus $P' = (x, -y)$ giving us a tangent line that is vertical, thus $P + P' = \infty$.

4. **Associativity:** Proving associativity is substantially more involved than verifying the other axioms, one must track how three chords/tangents meet the curve in projective space and show that the two possible orders of addition gives the same fourth intersection point. The full argument runs several pages in [3, §2.4] and lies beyond the scope of this paper.

□

3.3 Elliptic curves over finite fields

We are now ready to define the type of elliptic curves that will be used throughout the rest of this paper. By restricting to curves defined over finite fields, we obtain a discrete group structure suitable for cryptographic applications such as the elliptic curve discrete logarithm problem that we discuss later on. We lose however, the geometric interpretation of point addition and viewing of the curve, as shown in Figure 3. This section follows [3, §4.1].

Definition 3.3.1. When an elliptic curve is defined over a finite field $\mathbb{F}_q$, with $char(\mathbb{F}_q) \neq 0$ and $\Delta \neq 0$, all coordinates belong to $\mathbb{F}_q$ and we write its set of $\mathbb{F}_q$-rational points as

$$E(\mathbb{F}_q) = \left\{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : y^2 = x^3 + Ax + B\right\} \cup \{\infty\}$$

Example 3.3.2. Consider $E : y^2 = x^3 + x + 1$ over $\mathbb{F}_5$. We find:

$$\begin{aligned}
x = 0 &: \quad y^2 = 1 \Rightarrow y_1 = 1,\ y_2 = 4 \\
x = 1 &: \quad y^2 = 3 \Rightarrow \text{no solutions in } \mathbb{F}_5 \\
x = 2 &: \quad y^2 = 1 \Rightarrow y_1 = 1,\ y_2 = 4 \\
x = 3 &: \quad y^2 = 1 \Rightarrow y_1 = 1,\ y_2 = 4 \\
x = 4 &: \quad y^2 = 4 \Rightarrow y_1 = 2,\ y_2 = 3
\end{aligned}$$

Thus, the points of $E(\mathbb{F}_5)$ are: $(0,1), (0,4), (2,1), (2,4), (3,1), (3,4), (4,2), (4,3)$ plus the point at infinity $\{\infty\}$.

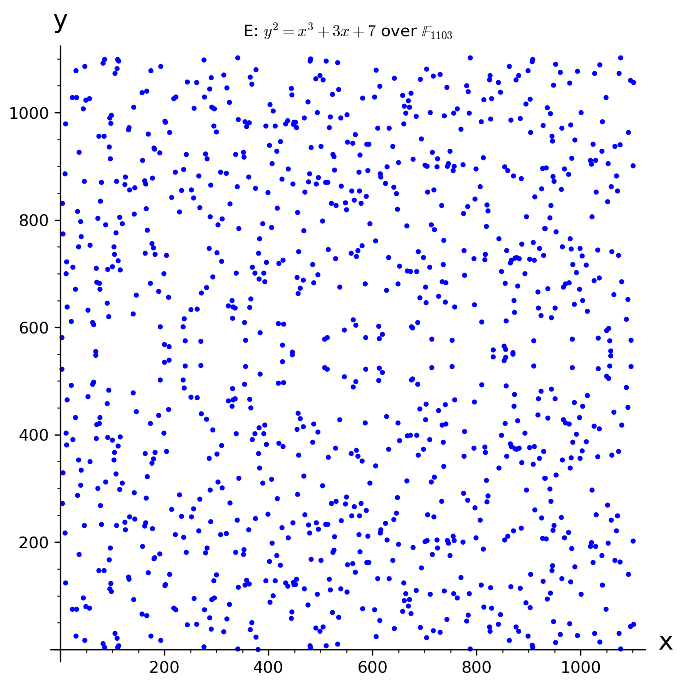

Figure 3: Example of an elliptic curve over a finite field

Theorem 3.3.3. Given a finite field $\mathbb{F}_q$ and taking an Elliptic curve E over $\mathbb{F}_q$ we can get an estimation for the number of points in E, denoted $\#E$ by

$$|\#E - (q-1)| \leq 2\sqrt{q}$$

Proof. The full proof can be found in [3][§4.2]. The main idea is to express the error term $a = q + 1 - \#E(\mathbb{F})$ as the trace that appears when one compares the q-power map $\phi : (x, y) \to (x^q, y^q)$ with the identity on the curve. It is shown that the quadratic $qx^2 - ax + 1$ is non-negative for every real x, this forces its discriminant to be non-positive, that is $a^2 \leq 4q$, which is exactly Hasse's bound. $\square$

4 Discrete logarithm problem (DLP)

Many cryptographic systems rely on the difficulty of solving the discrete logarithm problem and currently there are no known algorithms that run in polynomial time[8, §0]. We will be discussing one method of solving this problem for the multiplicative group for finite fields and after that move on to the DLP for elliptic curves. This introduction and index calculus mainly follows [3][§5.1].

4.1 Discrete logarithm problem for finite fields

Definition 4.1.1. Consider the multiplicative group of a finite field, $\mathbb{F}_q^\times$ with $char(\mathbb{F}_q) = p$. The classic discrete logarithm problem, in this group, requires us to find an exponent k from the congruence

$$g^k \equiv h \pmod{p} \quad \text{with } g, h, k \in \mathbb{F}_q^\times$$

Definition 4.1.2. Given a prime p, primitive root g and an integer k such that $g^k \equiv h \pmod{p}$ we denote the discrete logarithm of h with respect to k as:

$$L(h) = k \pmod{p-1}$$

That is, $L(h)$ is the power of g that gives h.

With this definition we can utilize logarithms laws since if

$$k = k_1 \cdot k_2 \text{ then } g^k = g^{k_1+k_2} \Rightarrow L(k) \equiv L(k_1) + L(k_2) \pmod{p-1}$$

Example 4.1.3. Let $p = 17$ then $\mathbb{F}_{17}^\times = \{1, 2, 3 \dots, 16\}$. Let $g = 3$, then g is a primitive root mod 17. We have that $3^{12} \equiv 4 \pmod{17}$, thus we have that $L(4) = 12$. Since $4 = 2 \cdot 2$ we can write

$$L(4) = L(2 \cdot 2) \equiv L(2) + L(2)$$

$$\text{Where} \quad L(2) = 14, \quad \text{therefore we have that } 14 + 14 = 28 \equiv 12 \pmod{16}$$

4.1.1 Index calculus method

The index calculus method takes advantage of being able to factor integers as product of primes, by calculating $L(k)$ for lots of small prime numbers k and then using these to build towards $L(h)$. First we decide a *smoothness* bound S, which is an integer that determines the *base* of primes called

$$B = \{k \in \mathbb{N} \mid k \text{ is prime and } k < S\}$$

From this base we then calculate relations of the form

$$g^x \equiv \pm k_i \cdot k_j \dots \pmod{p}$$

Thus we have found relations between $L(x)$ and sums of primes from our base B. We then compute $g^m \cdot h$ for several values of m until we find an integer $m = j$ so that

$$g^j \cdot h \equiv k_n \cdot k_p \cdot \dots \pmod{p} \qquad k_n, k_p \dots \in B$$

We can then calculate k from the relation

$$L(k) \equiv L(k_n) + L(k_p) + \dots - L(j) \pmod{p-1}$$

Example 4.1.4. Let $p = 877, g = 2, h = 15$ then we want to solve

$$2^k \equiv 15 \pmod{877}$$

First we set up our base, let $B = \{2, 3, 5, 7, 11\}$. We then look for exponents x with

$$2^x \equiv \pm \prod_{q \in B} q^i \pmod{877}$$

A search in SageMath, gives these five relations:

Table 2: Relations for index calculus

x	Congruence (mod 877)	Logarithm
10	$2^{10} \equiv 3 \cdot 7^2$	$L(3) + 2L(7)$
21	$2^{21} \equiv 5 \cdot 7^2$	$L(5) + 2L(7)$
1	$2^1 \equiv -5^3 \cdot 7$	$L(-1) + 3L(5) + L(7)$
43	$2^{43} \equiv -3^2 11$	$L(-1) + 2L(3) + L(11)$
54	$2^{54} \equiv -3 \cdot 5 \cdot 11$	$L(-1) + L(3) + L(5) + L(11)$

From lemma 2.3.6 we know that $L(-1) = 438$, we can thus put these calculated values into equations giving

$$L(3) + 2L(7) \equiv 10 \pmod{876} \tag{1}$$

$$L(5) + 2L(7) \equiv 21 \pmod{876} \tag{2}$$

$$438 + 3L(5) + L(7) \equiv 1 \pmod{876} \tag{3}$$

$$438 + 2L(3) + L(11) \equiv 43 \pmod{876} \tag{4}$$

$$438 + L(3) + L(5) + L(11) \equiv 54 \pmod{876} \tag{5}$$

Using (1),(2) and (3) we can back-substitute and find values for $L(3), L(5), L(7)$, then putting this together with (4) and (5) we get

$$L(2) = 1,\ L(3) = 686,\ L(5) = 697,\ L(7) = 100 \text{ and } L(11) = 861.$$

We have that

$$2^1 \cdot 15 = 30 = 2 \cdot 3 \cdot 5 \text{ so our } j \text{ is } 1$$

Hence, we get that

$$1 + L(15) \equiv L(2) + L(3) + L(5) \pmod{876} \quad \Rightarrow \quad L(15) = L(3) + L(5) = 1383 \equiv 507 \pmod{876}$$

A quick check shows that

$$2^{507} \equiv 15 \pmod{877}$$

4.1.2 Why the Index calculus method (usually) does not work for elliptic curves

The index calculus method relies on the ability to factor elements over a small base of generators, in the finite multiplicative group $\mathbb{F}_q^\times$ this translates to prime factorization. However, the group $E(\mathbb{F}_q)$ generally does not have anything equivalent to prime factorization. In particular, elliptic curve points do not decompose into "prime" or "irreducible" points in the same way integers do. As a result, there is no obvious choice of a factor base B, or an efficient method to determine whether a given point lies in the subgroup generated by a small set of such base points[8, §0 and §1]. Washington describes how one could adapt the index calculus method to specific types of curves, called hyper-elliptic curves, in [3][Chapter 13], this is however beyond the scope of this paper.

4.2 Discrete logarithm problem for elliptic curves

We are now ready to introduce the discrete logarithm problem for elliptic curves. Since we have seen that the points of an elliptic curve form an abelian group, we can adapt the classic discrete logarithm problem (DLP), usually posed in the multiplicative group $\mathbb{F}_p^\times$, to elliptic curves over finite fields. When working over elliptic curves, instead of asking for an exponent k, we ask for how many point additions k that links two points P and Q. The algorithms that we will mainly be looking at require roughly $\sqrt{\#E(\mathbb{F}_q)}$ group operations, so choosing q to be large makes the problem computationally exhaustive [3, Chapter 5].

Definition 4.2.1. ECDLP Let E be an elliptic curve defined by the Weierstrass equation $y^2 = x^3 + Ax + B$ over a finite field $\mathbb{F}_q$, with discriminant $\Delta \neq 0$. Denote by $E(\mathbb{F}_q)$ the finite set of $\mathbb{F}_q$-rational points on E, including the point of infinity $\{\infty\}$. The elliptic curve discrete logarithm problem can then be formulated as:

$$\text{Given points } P, Q \in E(\mathbb{F}_q) \text{ find the integer k such that}$$

$$Q = kP$$

That is the point P added to itself k times.

4.2.1 Implication of Lagrange's theorem

For elliptic curves, Lagrange's theorem implies that since the points form a group, the order of any point $P \in E(\mathbb{F}_q)$ must divide the total number of points $|E(\mathbb{F}_q)|$. Thus, solutions to equations of the form $Q = kP$, with $Q, P \in E(\mathbb{F}_q)$, are considered modulo the order of P.

4.2.2 Note

An important note is that if P and Q are arbitrary points in $E(\mathbb{F}_q)$, then it does not necessarily exist an k such that $Q = kP$. If P has order n then any point of the form xP lies in the cyclic subgroup $\langle P\rangle$ of order n. In this case the equation $Q = kP$ has a solution k mod n *if and only if* Q also lies in $\langle P\rangle$. In practice it is assumed that P has large prime order since if P is a composite, an attacker could break the problem one factor at a time (the Pohlig Hellman trick [3][§5.2.3]). Q is chosen specifically to be in the subgroup generated by P, so under these conditions one knows that Q is indeed a multiple of P so the *ECDLP* is well defined with k taken modulo the order of P.

4.3 Algorithms to solve ECDLP

Naively one could try to find k by computing multiples of P sequentially: $2P, 3P, 4P, \ldots$ until a match $kP = Q$ is found. However, this straightforward approach quickly becomes infeasible, as the computational time required grows exponentially with the size of the underlying field. Since an elliptic curve over $\mathbb{F}_q$ by theorem 3.3.3 has roughly q points, this could mean $\mathcal{O}(q)$ additions. If the field size is roughly $q \approx 2^m$, where m is the bit-length of the public key, the expected amount of additions is roughly 2^m operations, thus *exponential* in the key length. For a 256-bit field this would require on the order of 2^{256} additions, far beyond any realistic computing time. This section mainly follows [3, Chapter 5]

4.4 Pollard's rho

Pollard's ρ algorithm is a probabilistic algorithm designed to solve the ECDLP which if $|P| = n$, runs in $\mathcal{O}(\sqrt{n})$ time [3, §5.2.2]. Given an elliptic curve E taken over a finite field F, with known points P and Q in the group generated by $\langle P \rangle$, with the order of P denoted n. Pollard's ρ algorithm uses cycle detection in a randomly generated sequence of group elements. After choosing a function

$$f : \langle P \rangle \to \langle P \rangle$$

meant to pseudo-randomly shuffle the elements in $\langle P \rangle$ it chooses the next element to look at by recursion $P_{i+1} = f(P_i)$. Since we know that the subgroup $\langle P \rangle$ is finite, since $E(\mathbb{F}_q)$ is finite, the sequence generated by f inevitably enters a cycle after a finite number of steps. This produces indices $i_0 < j_0$ such that $P_{i_0} = P_{j_0}$. After this, all subsequent elements repeat periodically with a period equal to $j_0 - i_0$. Since we begin with a tail and then enter a cycle it forms the shape of a Greek 'rho' letter, where the name comes from. To increase efficiency, with a small increase to computational costs, Pollard's ρ method maintains only two elements generated by the sequence. These are called the tortoise and the hare, where one moves twice as fast as the other:

$$\text{Tortoise: } P_{i+1} = f(P_i),$$

$$\text{Hare } P_{2(i+1)} = f(f(P_{2i}))$$

When these two elements intersect, a cycle is detected and from there we can calculate k. This approach significantly reduces memory usage, as it requires storage only for the current pair of elements regardless of group size. The effectiveness of Pollard's ρ method can be greatly affected by the choice of the pseudo random function f. Next we define one version that is commonly used.

Definition 4.4.1 (Update function for Pollard's rho method)**.** Given an instance of the ECDLP, with the order of $|P| = n$. Start by fixing an integer $s \geq 2$, which will be the number of partitions of the group $\langle P \rangle$. Pick random coefficients

$$(a_i, b_i) \in \{0, \ldots, n-1\} \quad (0 \leq i \leq s)$$

and compute one jump for each residue i:

$$R_i = a_i P + b_i Q \in \langle P \rangle$$

During the walk we keep two triples of the form (X, a, b) where $X = aP + bQ$. To update we first compute

$$i = x(X) \pmod{s}$$

Where $x(X)$ is the affine x-coordinate of X viewed as an integer $0 \le x(X) < p$, where $p = char(\mathbb{F})$ and if X is the point of $\{\infty\}$ then i is taken to be 0. Then the function f is:

$$f(X, a, b) = (X + R_i,\ a + a_i,\ b + b_i)$$

Where $a + a_i,\ b + b_i$ are reduced modulo n while $X + R_i$ is a standard point addition.

4.4.1 Solving for k

To start we chose a random initial point by selecting $0 \le a_0, b_0 \le n - 1$ which gives us

$$(a_0 P + b_0 Q,\ a_0,\ b_0) = (X_0,\ a_0,\ b_0)$$

The algorithm then iterates until a collision is detected between the tortoise and the hare, with indices $i < j$ we have $X_i = X_j$. We can express these points as

$$X_i = a_i P + b_i Q \text{ and } X_j = a_j P + b_j Q$$

We thus have

$$a_i P + b_i Q = a_j P + b_j Q$$

which can rearrange according to group law as

$$(a_i - a_j)P = (b_j - b_i)Q$$

Given that $Q = kP$, we can derive that

$$(a_i - a_j) \equiv (b_j - b_i)k \bmod n$$

To solve for k we first calculate $d = \gcd(b_j - b_i,\ n)$. This gives way for three scenarios:

- **Case 1:** If $d = 1$ then $(b_j - b_i)$ is invertible $\bmod n$, and we can directly compute the value of k using the inversion:

$$k \equiv (b_j - b_i)^{-1}(a_i - a_j) \pmod{n}$$

- **Case 2:** If $d > 1$ and $d \mid (a_i - a_j)$ then the congruence is solvable but has exactly d solutions. This is because $\frac{(b_j - b_i)}{d}$ is invertible modulo $\frac{n}{d}$. Let

$$A = \frac{(a_i - a_j)}{d},\ B = \frac{(b_j - b_i)}{d} \text{ and } N = \frac{n}{d}$$

  then

$$k \equiv B^{-1}A + tN \quad (t = 0, 1, \ldots, d - 1)$$

  Each value can then individually be checked to see if it satisfies $Q = kP$.

- **Case 3:** If $d > 1$ but does not divide $(a_i - a_j)$ then there are no solutions and we have to restart the algorithm.

Example 4.4.2.

Let $E\colon y^2 = x^3 + 2x + 2$ over $\mathbb{F}_{17}$, $\quad P = (5,1), \quad n = |P| = 19, \quad Q = kP = (0,6), \quad s = 3$

we update the point $X = (x, y)$ by the function

$$(X, a, b) \longmapsto \big(X + R_i,\ a + a_i,\ b + b_i\big), \quad i = x(X) \bmod s$$

with partitions

$$(a_0, b_0) = (1,1), \quad (a_1, b_1) = (2,3), \quad (a_2, b_2) = (3,5), \quad R_i = a_i P + b_i Q$$

we can check that

$$R_0 = P + Q = (13,7), \quad R_1 = 2P + 3Q = (3,1), \quad R_2 = 3P + 5Q = \{\infty\}$$

We pick a starting state $(X, a, b) = (Q, 0, 1) = ((0,6), 0, 1)$ then

$$i \ = \ x(X) \bmod 3 \ = \ 0 \ \bmod 3 \ = \ 0.$$

So we update the tortoise by

$$((0,6),0,1) \to ((0,6) + R_0, 0 + a_0, 1 + b_0) = ((3,16),1,2)$$

Using the addition rules that were shown above. Then the hare that takes two jumps by

$$((0,6),0,1) \to ((3,16),1,2) \to ((3,16) + R_0, 1 + a_0, 2 + b_0) = ((3,1)2,3)$$

If we do this continuously we get this table:

Table 3: Floyd's cycle states for Pollard's ρ example

Step	Tortoise (X, a, b)	Hare (X, a, b)
1	$((3,16),1,2)$	$((3,1),2,3)$
2	$((3,1),2,3)$	$((5,1),4,5)$
3	$((0,11),3,4)$	$((5,1),10,15)$
4	$((5,1),4,5)$	$((5,1),16,6)$

Here we can see the collision at step 4, where we get a collision at (5.1). We can then calculate k from

$$aP + bQ = a'P + b'Q \quad \Rightarrow \quad (a - a')P = (b' - b)Q \quad \Leftrightarrow \quad (-12)P = 1 \cdot Q \quad \Rightarrow \quad -12 = 1 \cdot k$$

$$\text{Since } -12 \equiv 7 \pmod{19}$$

$$\text{we get } k = 7$$

A quick calculation can be done to check that it is indeed the case:

$$7 \cdot P = 2 \cdot (2 \cdot P) + 2 \cdot P + P = (3,1) + (6,3) + (5,1) = (0,6)$$

Remember that we are not adding coordinates, we are using the addition as defined in the group law.

4.5 Pollard's lambda

Pollard's λ method is a probabilistic algorithm designed to solve the ECDLP, extending the ideas of Pollard's ρ method to effectively leverage parallel computations. Instead of independently running multiple instances of the ρ algorithm, the λ method uses multiple concurrent instances that collaboratively report *distinguished* points to a central database. Washington claims that when running several parallel processes, Pollard's λ's expected runtime of $\mathcal{O}(\sqrt{n})$ can be "significantly improved" [3, 5.2.2]. On the other hand van Oorschot and Weiner claims, given m parallel processes the speedup is at most of the order $\mathcal{O}(\sqrt{m})$, which is notably not linear with the amount of processes[6, §3]. Given an instance of the ECDLP, we begin by defining a pseudo-random function similar to the ρ algorithm:

$$f : \langle P \rangle \to \langle P \rangle$$

which is used to generate sequences of points $P^j_{i+1} = f(P^j_i)$ from random chosen starting points $P_0 \dots P^r_o$ for each process. Then each sequence saves certain predetermined points, called distinguished, which are usually chosen by simple arithmetic criteria, such as points whose coordinates satisfy a particular condition (e.g. the last k bits being zero), to a central list. Due to the group $\langle P \rangle$ being finite, independent sequences are bound to eventually collide, producing matching distinguished points in the central list. When a second distinguished point is reported back, computations halt and similarly to the ρ algorithm, the discrete logarithm k can be computed from the collision points. Specifically, if two processes report the same distinguished points represented as

$$P = aP + bQ \quad \text{and} \quad P = a'P + b'Q$$

then we have:

$$(a - a')P = (b' - b)Q$$

and since $Q = kP$, we can derive the congruence:

$$(a - a') \equiv (b' - b)k \pmod{n}$$

which can be solved efficiently similar to in Pollard's ρ method. In practice, some collisions may not directly result in a solution similar to Pollard's ρ, requiring continued parallel computation until a useful collision emerges.

4.6 Pollard's kangaroo

Pollard's kangaroo method is closely related to the ρ and λ method and the two are often written about interchangeably. It is specifically useful when the discrete logarithm k is known to lie within a certain interval $[a, b]$ [6, §5.1]. Such prior knowledge about k's approximate range may arise from partial information leaks, previous computations or known structural constraints in cryptographic scenarios. The method (in a single processor case) involves two independent walks ("kangaroos") that make pseudo-random jumps in the group $\langle P \rangle$:

- **Tame kangaroo** starts at the upper endpoint of the interval, setting $T_0 = bP$ and performs jumps determined by a predefined pseudo-random function, recording its cumulative distance from bP.

- **Wild kangaroo** begins from the unknown point $W_0 = Q = kP$ and executes the same sequence of jumps as the tame kangaroo.

Both kangaroos follow the same deterministic rules for jumping, defined by a pseudo-random update function:

$$f : \langle P \rangle \to \langle P \rangle, \quad X_{i+1} = X_i + f(X_i)$$

Eventually, the wild kangaroo lands on a point previously visited by the tame kangaroo, at which point both kangaroos converge and continue identically after that. To recover the discrete logarithm k, suppose the tame kangaroo traveled a cumulative distance of d_t, ending at point $T_n = (b + d_t)P$. Similarly, the wild kangaroo traveled a distance d_w from Q, landing on the same point:

$$(k + d_w)P = (b + d_t)P$$

Solving for k gives:

$$k = b + d_t - d_w$$

Since the interval length is $L = b - a$, the expected runtime of Pollard's kangaroo method is $\mathcal{O}(\sqrt{L})$. Heuristically, because collisions inside the interval behave like a birthday paradox in a set of size L, we expect $\mathcal{O}(\sqrt{L})$ steps before a collision. A rigorous analysis is given in [6, §5.1]. This makes pollards kangaroo particularly efficient for intervals significantly smaller than the full group size. However, it should be noted that van Oorschot and Weiner shows that when the interval covers the whole group of size n, the kangaroo method requires around $2\sqrt{n}$ operations, while an optimized ρ-method roughly requires $\sqrt{\frac{\pi}{2}n}$ [6, §1 and (6)]. Thus the kangaroo method is expected to run 1.6 times slower on the full group.

5 Empirical tests

5.1 Selecting curves for testing the ECDLP

To be able to consistently select similar curves and points P for the following tests, we decided to standardize a way to select all variables. The goal was to obtain a point P with large prime order of approximately the same size across the "scale" set for each experiment. Since the difficulty of the ECDLP relies on the order of P we thus enforce some consistency across the curves within the same test.

- To start, we first pick a random prime p in the range $(10^{scale}, 10^{scale+1})$.
- Then we randomly select integers (A, B) in $\mathbb{F}_p$ such that $\Delta \neq 0$, thus giving us an elliptic curve $E(\mathbb{F}_p)$.
- We then factor $|E(\mathbb{F}_p)|$ and take the largest prime number q, if q is not in the range $\frac{p}{100} < q \leq |E(\mathbb{F}_p)|$ we start over.
- After finding a suitable curve we select a point P in $E(\mathbb{F}_p)$ such that $|P| = q$ (which we know exists from theorem 2.2.10).
- Finally we randomly draw k from $\{1, \ldots, q-1\}$ and calculate $Q = kP$.

5.2 Performance evaluation of Pollard's rho with different partition sizes

In [3][§5.2.2] Washington briefly suggests that dividing $\langle P \rangle$ into $s \approx 20$ subsets yields near-optimal running times for Pollard's rho algorithm. To test this claim, we implemented Pollard's rho algorithm with Floyd's cycle detection in SageMath with varying values for s.

5.2.1 Implementation in SageMath

The partition is set up for each s, an integer list is fixed $\{a_i, b_i\}_{i=0}^{s-1}$ where $a_i = i+1$, $b_i = 2i+1$ and pre-calculate points $R_i = a_i P + b_i Q \quad (0 \leq i < s)$. A state (X, a, b) with $X = aP + bQ$ updates by applying the precomputed jump by

$$(X, a, b) \longmapsto \big(X + R_i,\ a + a_i,\ b + b_i\big), \quad i = x(X) \bmod s$$

Where $x(X)$ denotes the affine x-coordinate of X (viewed as the integer $0 \leq x(X) < p$, where $p = char(\mathbb{F})$) and if X is the point of $\{\infty\}$ then i is taken to be 0. Thus at each step we reduce the current x-coordinate modulo s to get the partition i, then add the precomputed jump R_i and finally update $a \to a + a_i \pmod{n}$ and $b \to b + b_i \pmod{n}$. Cycle detection via the tortoise and hare method halts the process when two states share the same X. Upon collision $(X, a, b) = (X, a', b')$, we solve $(a - a')P = (b' - b)Q$ for the secret $k \equiv (a - a')\,(b' - b)^{-1} \pmod{n}$. IF $gcd(b' - b, n) > 1$ but remains below $n^{1/2}$, we test all candidate solutions in that small factor.

5.2.2 Experimental Methodology

- **Curve selection.** To be able to work efficiently with sufficiently large curves, 50 elliptic curves were randomly generated over prime fields of approximately 10^8 size using 5.1.
- **Trials per partition count.** For each curve and each $s \in \{2, 3, \ldots, 40\}$, 10 independent Pollard's rho trials were run, each with random starting states (a_0, b_0). If a run was not successful, it was restarted with a new starting state (up to 10 times).
- **Timing and averaging.** For each successful trial, the elapsed time was recorded and averaged out over the 10 runs, then averaged across all 100 curves to yield the plot below.

5.2.3 Results and observations

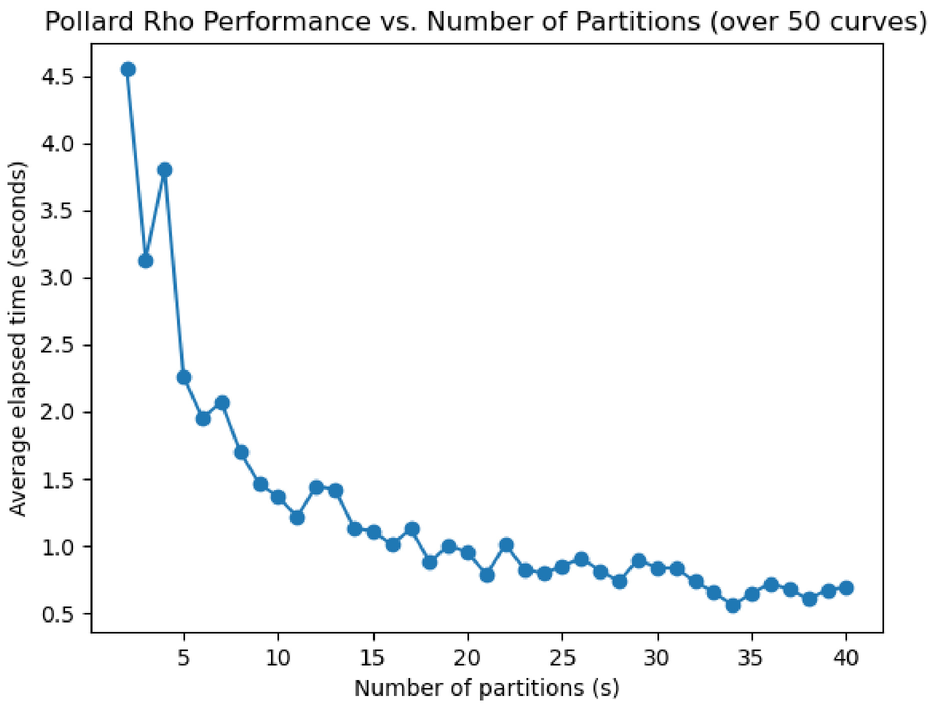

Figure 4: Performance evaluation of Pollard's rho method vs. number of partitions

The performance curve presented in Figure 4 clearly illustrates a sharp decrease in the average runtime of Pollard's rho method as the number of partitions s increases from 2 up to approximately 15. This trend indicates that the pseudo-random walk rapidly achieves a desirable level of mixing. However, beyond $s \approx 15$, additional increases in the partition count yield diminish returns, with only minor improvements observed after $s \approx 20$. There is a noticeable dip at $s = 34$, which could suggest that there are gains to be made by increasing s beyond 20. However, these findings are specific to our field sizes, curves, random choices of P, Q, starting points and implementation. Different parameter choices or moving to substantially larger groups, may shift the optimum. Thus, while our experiments support the $s = 20$ heuristic, they should not be considered as definite proof for all contexts.

5.3 Testing my implementation against SageMath's

To test the efficiency of our implementation of Pollard's rho, we compared its performance with the built-in Pollard's rho method provided by SageMath. Ten elliptic curves of scale 10^9 were randomly selected using 5.1 and the discrete logarithms were solved using both implementations. The results, as seen in Figure 5, indicate, that while both methods successfully computed the discrete logarithm problem, my implementation was consistently slower. This discrepancy is likely due to differences in the back-end, with the biggest culprit most likely being computational overhead. While my implementation relies on python-level operations, SageMath's implementation utilizes cython which allows a python program to interface with C code which can be much faster [7]. That is, SageMath's implementation most likely has a more optimized backend, which operates at lower levels and can compute much faster.

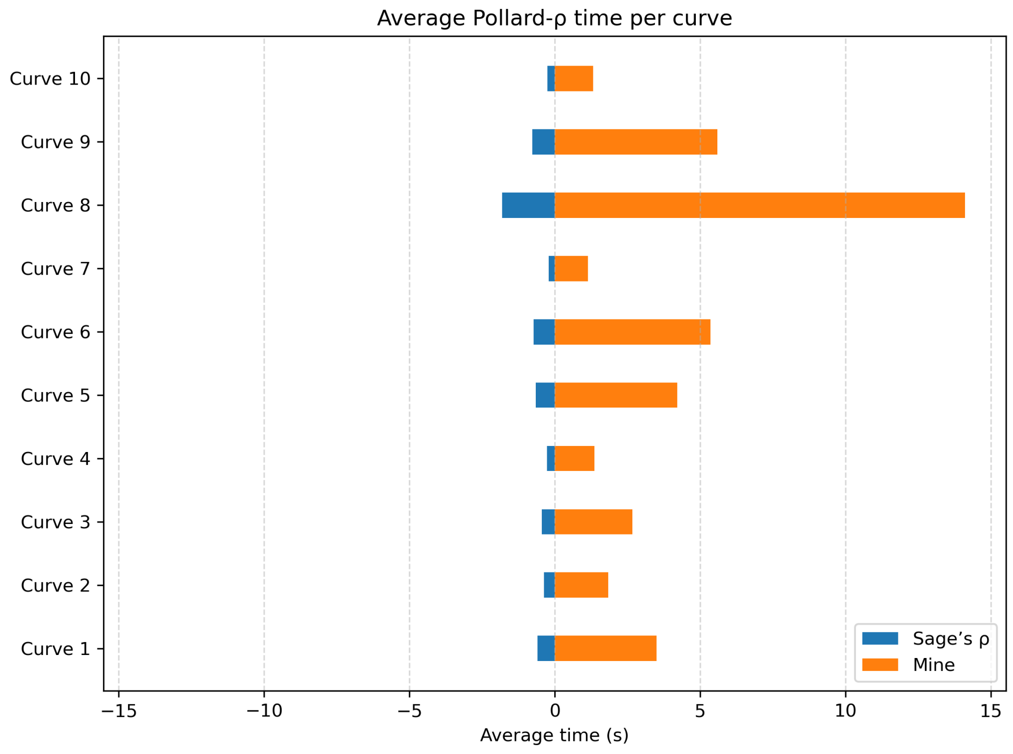

Figure 5: Performance evaluation of my implementation of Pollard's rho method vs. SageMath's built in.

5.4 Empirical speed-up obtained by Pollard's kangaroo method

Pollard's kangaroo algorithm exploits *apriori* knowledge that the discrete logarithm k is confined to an interval of $[a, b]$ of length $L = b - a$, giving an expected running time of $\mathcal{O}(\sqrt{L})$ instead of $\mathcal{O}(\sqrt{n})$ for Pollard's ρ walk on the whole group of order n. To see how that asymptotic advantage looks in practice, we bench-marked the two SageMath implementations `algorithm='rho'` and `algorithm='lambda'` on progressively smaller intervals. When running the 'lambda' algorithm we could pass on the interval L.

Experimental Methodology

- **Curve selection.** We chose a random curve using 5.1 of scale 12.

- **Intervals.** Then we fixed five intervals $\{\, n^{1/8},\ n^{1/4},\ n^{1/2},\ n/log(n),\ n/2,\ n\,\}$
- **Secret k.** For each interval we picked a random secret k within that interval, formed $Q = kP$ and timed
  - `discrete_log(Q,P, ord=n, algorithm='rho')`
  - `discrete_log(Q,P, ord=n, algorithm='lambda', bounds=(0,b))`

We repeated the whole procedure on 20 curves (`scale=12`, hence $p \approx 10^{12}$) and averaged the timings.

5.4.1 Results

The comparative performance of Pollard's kangaroo and Pollard's ρ methods can be seen in Figure 6. In the first figure, when the discrete logarithm k is restricted to intervals of size equal to n or $n/2$, the kangaroo method does not show any speed advantage, instead performing slower than Pollard's rho. This result aligns with the theoretical expectations, since the kangaroo method's performance advantage should only be seen when the interval size is substantially smaller than the entire group. The most interesting result is that when $L = n/log(n)$, we can see that in our tests, averaged out over 20 curves, that the kangaroo method performed more than twice as fast. The difference is as expected even larger when $L = \sqrt{n}$ and beyond, where the timing for the kangaroo method is less than $\frac{1}{14}$ that of Pollard's rho. This highlights that Pollard's kangaroo method is much more efficient as the search interval shrinks, showing its practical applicability primary in scenarios where some *apriori* information about k is available.

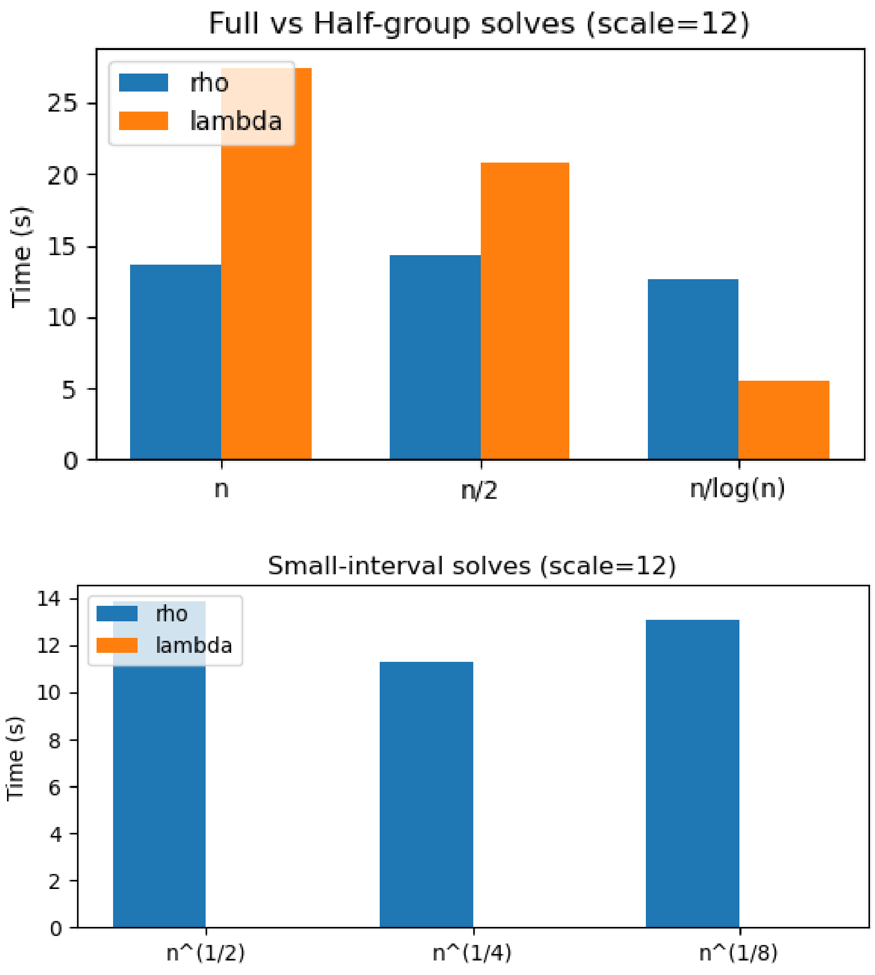

Figure 6: Performance evaluation of Pollard's rho method vs. Pollard's kangaroo method given that k is within a certain interval

References

[1] David S. Dummit, Abstract Algebra, third edition.

[2] Norman L. Biggs, Discrete Mathematics, second edition.

[3] Lawrence C. Washington, Elliptic Curves Number Theory And Cryptography second edition.

[4] Lidl R, Niederreiter H. Finite Fields. 2nd ed. Cambridge University Press; 1996.

[5] Diem C. On the discrete logarithm problem in elliptic curves. Compositio Mathematica. 2011;147(1):75-104. doi:10.1112/S0010437X10005075

[6] van Oorschot, P.C., Wiener, M.J. Parallel Collision Search with Cryptanalytic Applications. J. Cryptology 12, 1–28 (1999). `https://doi.org/10.1007/PL00003816`

[7] SageMath F.A.Q. answers how sage math is built: `https://doc.sagemath.org/html/en/faq/faq-general.html#is-sage-fast`

[8] Silverman, J.H., Suzuki, J. (1998). Elliptic Curve Discrete Logarithms and the Index Calculus. In: Ohta, K., Pei, D. (eds) Advances in Cryptology — ASIACRYPT'98. ASIACRYPT 1998. Lecture Notes in Computer Science, vol 1514. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/3-540-49649-1_10`