TEMPLE UNIVERSITY

Elliptic Curves and Cryptography

Tasnia Kader

College of Science and Technology (CST)

MATH 4096 : Senior Problem Solving

Dr. Matthew Stover

April 24, 2025

1 Introduction

Elliptic Curve Cryptography (ECC) is a form of public-key cryptography that uses the mathematical properties of elliptic curves. In general, strong cryptographic systems are trapdoor functions. These functions are easy to compute in one direction but extremely difficult to reverse without special information [5]. In encryption, this means it's simple to encode a message using the public key, but nearly impossible to decode it without the corresponding private key. ECC uses the Elliptic Curve Discrete Logarithm Problem (ECDLP) as a trapdoor function, where multiplying a point by a scalar is easy, but reversing that operation without the private key is extremely challenging.

Although ECC is relatively new compared to RSA, it offers stronger security with much shorter key lengths, making it more efficient in terms of speed, storage, and computational power. ECC is widely used in cryptocurrencies like Bitcoin, Ethereum, and Litecoin to generate secure public-private key pairs [5]. It also plays a key role in secure communication protocols, enabling encryption, digital signatures, and key exchange in systems like Transport Layer Security (TLS) for web browsing, Secure Shell (SSH) for remote access, and Virtual Private Networks (VPNs) for secure internet traffic [5].

Because ECC plays such a big role in modern cryptography, it's important to understand the mathematics behind elliptic curves. This paper aims to provide that understanding by first examining the structure of elliptic curves: their definition, geometric interpretation, and group structure. Next, it will explore the elliptic curve discrete logarithm problem (ECDLP), which is central to elliptic curve cryptography. Also, the paper will look at the Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol, which relies on the security of the ECDLP. Finally, it will analyze Pollard's Rho Algorithm, a well-known method used to try to crack the ECDLP.

1.1 Definition of the Elliptic Curve

An elliptic curve $E(K)$ over a field K is defined by the set of points $(x, y) \in K^2$ satisfying the equation

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

along with a point $\mathcal{O}$ called "the point at infinity"[6]. The coefficients a, b are in K. To ensure the curve has no repeated roots, the discriminant must satisfy the following condition: $4a^3 + 27b^2 \neq 0$. The field K can be $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{C}$, or $\mathbb{Z}/p\mathbb{Z}$.

2 Elliptic Curves Over Real Field

Although this paper mostly focuses on elliptic curves over finite fields, we first explore elliptic curves over the real field $\mathbb{R}$ to develop an intuitive understanding.

2.1 Geometric Addition on $E(\mathbb{R})$

We can define an addition operation $\oplus$ on the points of an elliptic curve geometrically.

Let P and Q be distinct points on $E(\mathbb{R})$. Then, we add P and Q using the following steps [6]:

1. Draw the straight line passing through P and Q.
2. This line intersects the curve at a third point, which we denote as $-R$.
3. Due to the symmetry of elliptic curves with respect to the x-axis, we can reflect $-R$ across the x-axis to obtain R.
4. We define the sum of P and Q as $P \oplus Q = R$.

This process is illustrated in the first graph of Figure 1.

On the other hand, if $P = Q$, we draw the tangent line at P and follow the same steps as shown in the second graph of Figure 1.

If two points P and Q lie on a vertical line, their sum is defined as the point at infinity $\mathcal{O}$. This is because the vertical line does not intersect the curve at a third point. Hence, we write $P \oplus Q = \mathcal{O}$. This is shown in the third graph of Figure 1.

Note that we define scalar multiplication kP as repeated addition $P \oplus P \oplus P \oplus ...$, where k is a positive integer.

3 Elliptic Curves Over Finite Fields

The definition of elliptic curves over finite fields mirrors the general form of elliptic curves over a field K, with K being $\mathbb{F}_p$ in this case.

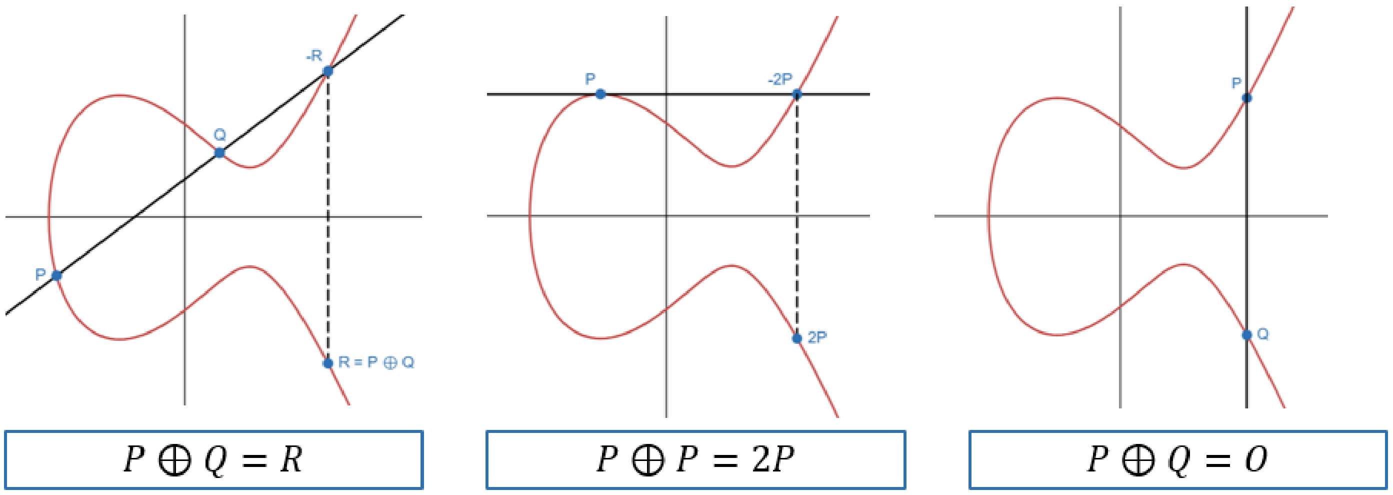


Figure 1: Geometric Addition of Points on an Elliptic Curve

Definition: An elliptic curve over $\mathbb{F}_p$, where p is a prime, is a set of points $(x, y) \in \mathbb{F}_p^2$ satisfying the equation

$$y^2 = x^3 + ax + b \pmod p,$$

with $a, b \in \mathbb{F}_p$ and an element $\mathcal{O}$ [6]. The coefficients a and b must satisfy the condition $4a^3 + 27b^2 \neq 0 \pmod p$ to ensure the curve has no repeated roots.

It is more challenging to geometrically visualize the addition of two points in a finite field because the graph appears as a scatter plot of points rather than a smooth, continuous curve. Figure 2 illustrates a graph of an elliptic curve over $\mathbb{F}_{17}$.

However, we can define an operation $\oplus$ on the elliptic curve $E(\mathbb{F}_p)$ using algebraic methods based on the geometric interpretation used in the real case. To add two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, we first calculate the slope λ of the line connecting them. If the two points are distinct, the slope is given by the standard formula $\lambda = \frac{y_2 - y_1}{x_2 - x_1} \pmod p$ where division is performed by multiplying by the modular inverse of $x_2 - x_1$ in $\mathbb{F}_p$. If the two points are identical, we instead use the slope of the tangent line at that point. The derivative of an elliptic curve is $2y \cdot \frac{dy}{dx} = 3x^2 + a$, so at (x_1, y_1) we would get $\lambda = \frac{dy}{dx} = \frac{3x_1^2 + a}{2y_1} \pmod p$. Once we have computed λ, we use

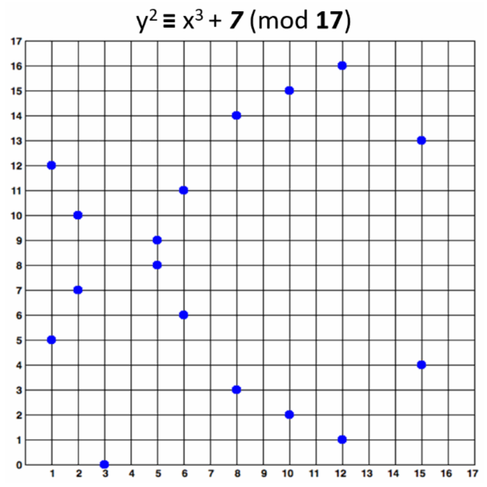


Figure 2: Elliptic Curve Over $\mathbb{F}_{17}$

it to find the third point of intersection $-R = (x_3, -y_3)$ of this line with the elliptic curve. The sum $P \oplus Q$ is then defined as the reflection of $-R$ across the $x-$axis: $P \oplus Q = R = (x_3, y_3) \pmod p$. This leads to the following coordinate formulas [3]:

$$x_3 = \lambda^2 - x_1 - x_2 \pmod p,$$

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

To summarize, the slope λ is given by

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \pmod p, & \text{if } P \neq Q \text{ and } x_1 \not\equiv x_2 \pmod p, \\ \frac{3x_1^2 + a}{2y_1} \pmod p, & \text{if } P = Q \text{ and } y_1 \not\equiv 0 \pmod p \end{cases}$$

Finally, note that if $Q = (x_1, -y_1 \mod p)$, then $P \oplus Q = \mathcal{O}$ since the line through them is vertical.

It is crucial to establish that $E(\mathbb{F}_p)$ forms a group under this operation because it provides the necessary properties for defining and performing cryptographic operations. Without these properties, the operations would not be well-defined, rendering the system both insecure and impractical. The group structure ensures that the system is predictable, secure, and efficient.

Theorem: The set $E(\mathbb{F}_p)$ forms an abelian group under $\oplus$.

Proof: Let $P, Q, S \in E(\mathbb{F}_p)$

1. **Commutative Law:** The line joining P and Q is determined uniquely by the two points, so it is the same as the line connecting Q and P, so it follows that $P \oplus Q = Q \oplus P$.

2. **Closure:** The addition operation on elliptic curves is defined such that the sum of any two points on the curve yields another point on the curve (including $\mathcal{O}$). Therefore, for any $P, Q \in E(\mathbb{F}_p)$, the point $P \oplus Q$ is also in $E(\mathbb{F}_p)$, and the set is closed under the operation $\oplus$.

3. **Associativity:** Verifying that the associative law $(P \oplus Q) \oplus S = P \oplus (Q \oplus S)$ holds is not trivial by any means. The proof can be approached either through lengthy computations with explicit formulas or by using more advanced analytic or algebraic techniques. In this paper, the proof is omitted; however, a verification using Mathematica, conducted by Kazuyuki Fujii and Hiroshi Oike, is provided in the bibliography [1].

4. **Identity Element:** To compute $P \oplus \mathcal{O}$, we draw a vertical line through P. This line will intersect the elliptic curve at a third point $-P$. By the definition of elliptic curve addition, we reflect this point $-P$, which will give us back the original point P. Hence, $P \oplus \mathcal{O} = P$. Similarly, by symmetry, $\mathcal{O} \oplus P = P$. Therefore, $\mathcal{O}$ is the identity element.

5. **Inverses:** A line through P and $-P$ intersects the curve at exactly those two points (since it will be a vertical line), so the result of adding them is the point at infinity, $\mathcal{O}$. Thus, $P \oplus (-P) = \mathcal{O}$, and $-P$ is the inverse of P. More specifically, if $P = (x, y)$, then $-P = (x, -y \bmod p)$ is the inverse of P.

Since all the group properties are satisfied, we can conclude that $E(\mathbb{F}_p)$ forms a finite abelian group under elliptic curve addition.

$\square$

It is worth mentioning that by the Fundamental Theorem of Finite Abelian Groups, the group $E(\mathbb{F}_p)$ is isomorphic to $\mathbb{Z}_m \times \mathbb{Z}_n$ where $m|n$. This structure is important because secure protocals rely on selecting a point from a large cyclic subgroup, which ensures the difficulty of the elliptic curve discrete logarithm problem.

3.1 Order of the Group $E(\mathbb{F}_p)$

In cryptography, the order of the elliptic curve group is crucial for security. A larger group order increases the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), which strengthens the system's resistance to attacks.

To find the number of points on an elliptic curve over a finite field $\mathbb{F}_p$, we can use the Legendre symbol, which is defined for any integer m as follows:

$$\left(\frac{m}{p}\right) = \begin{cases} 0 & \text{if } p \mid m, \\ 1 & \text{if } p \nmid m \text{ and } m \text{ is a square modulo } p, \\ -1 & \text{if } p \nmid m \text{ and } m \text{ is not a square modulo } p. \end{cases}$$

Proposition: Let $E : y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbb{F}_p$. Then the number of points on E is given by

$$\#E(\mathbb{F}_p) = 1 + p + \sum_{x \in \mathbb{F}_p} \left(\frac{x^3 + ax + b}{p}\right).$$

The following proof follows the steps discussed in Winkler's work on elliptic curves and cryptography [8].

Proof: Let $x \in \mathbb{F}_p$. For each x, define $f(x) = x^3 + ax + b$. Then, the number of solutions $y \in \mathbb{F}_p$ to the equation $y^2 = f(x)$ depends on whether $f(x)$ is a square modulo p. Hence, the total number of solutions for each x can be expressed as

$$\#\{y : y^2 = f(x)\} = \begin{cases} 1 & \text{if } \left(\frac{f(x)}{p}\right) = 0 \\ 2 & \text{if } \left(\frac{f(x)}{p}\right) = 1 \\ 0 & \text{if } \left(\frac{f(x)}{p}\right) = -1 \end{cases}$$

These values can be compactly expressed as

$$1 + \left(\frac{f(x)}{p}\right)$$

Since there are p elements in $\mathbb{F}_p$, the total number of points (excluding the identity $\mathcal{O}$) is

$$\sum_{x \in \mathbb{F}_p} \left(1 + \left(\frac{x^3 + ax + b}{p}\right)\right).$$

Now, adding 1 to this expression for the identity yields

$$\#E(\mathbb{F}_p) = 1 + \sum_{x\in\mathbb{F}_p} \left(1 + \left(\frac{x^3+ax+b}{p}\right)\right) = 1 + \sum_{x\in\mathbb{F}_p} 1 + \sum_{x\in\mathbb{F}_p} \left(\frac{x^3+ax+b}{p}\right)$$

Since the first sum simply counts the p elements in $\mathbb{F}_p$, we can write

$$\#E(\mathbb{F}_p) = 1 + p + \sum_{x\in\mathbb{F}_p} \left(\frac{x^3+ax+b}{p}\right).$$

□

This highlights how the number of points on the elliptic curve depends on the distribution of squares modulo p. While the specific values of a and b affect the shape of the curve, the total number of points doesn't vary wildly. In fact, the deviation of $\#E(\mathbb{F}_p)$ from the value of $p+1$ is tightly constrained by a classical result in number theory known as Hasse's Theorem.

Hasse's Theorem: The number of points in the group $E(\mathbb{F}_p)$ is bounded above and below. Specifically, the order of the group $E(\mathbb{F}_p)$ satisfies

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

Hasse's theorem states that the number of points on an elliptic curve $E(\mathbb{F}_p)$ is close to $p+1$ with a margin of error at most $2\sqrt{p}$. This result is important in elliptic curve cryptography (ECC) because, once again, the security of ECC depends on the size of the elliptic curve group. The theorem helps ensure that the group has a size large enough for strong cryptographic security.

To prove Hasse's Theorem, two concepts are introduced: the Frobenius endomorphism and an inequality involving a function that maps elements from an abelian group to integers. The proof presented here follows the approach outlined by J.H. Silverman in the graduate-level text *The Arithmetic of Elliptic Curves* [7].

Definition (Frobenius Endomorphism): Let E be an elliptic curve defined over the finite field $\mathbb{F}_p$. Then, the Frobenius endomorphism is the map

$$\phi : E \to E, \quad \phi(x,y) = (x^p, y^p).$$

Since we're working in the field $\mathbb{F}_p$, Fermat's Little Theorem tells us that $x^p \equiv x \mod p$, and similarly $y^p \equiv y \mod p$, so $\phi(x,y) = (x,y)$ for all points with coordinates in $\mathbb{F}_p$. Thus, P remains unchanged under this map: $\phi(P) = P$. For this, we can conclude

$$\phi(P) - P = \mathcal{O},$$

which means

$$(\phi - 1)(P) = \mathcal{O},$$

where 1 represents the identity function. Since $\mathcal{O}$ is the identity element in the elliptic curve group, this equation implies that P is in the kernel of the map $\phi - 1$. So we get

$$P \in \ker(\phi - 1).$$

Conversely, if $(\phi - 1)(P) = \mathcal{O}$, then $\phi(P) = P$, which means P is a point defined over $\mathbb{F}_p$. Therefore, the set of points on the elliptic curve is exactly the kernel of $\phi - 1$:

$$E(\mathbb{F}_p) = \ker(\phi - 1), \quad \text{so} \quad \#E(\mathbb{F}_p) = \# \ker(\phi - 1).$$

The map $\phi - 1$ is separable, meaning it does not collapse distinct points into one. Its kernel consists of the fixed points of the Frobenius endomorphism, which correspond to the elliptic curve's points. Since the degree of an endomorphism counts the number of points in the preimage of the identity element $\mathcal{O}$, we can conclude that the number of points on the elliptic curve is equal to the degree of $1 - \phi$

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

Lemma (Used in Proof): Let d be a positive definite quadratic form on an abelian group A, and let $\phi, \psi \in A$. Then for any two elements ϕ and ψ in A, we have

$$|d(\phi - \psi) - d(\phi) - d(\psi)| \leq 2\sqrt{d(\phi)d(\psi)}.$$

This inequality is analogous to the Cauchy-Schwarz inequality in vector spaces, but adapted for quadratic forms and abelian groups.

Proof of Hasse's Theorem: Apply the lemma with $\psi = 1$ and ϕ as the Frobenius endomorphism. Also, consider d to be the degree function. Then, we get

$$|\deg(1 - \phi) - \deg(1) - \deg(\phi)| \leq 2\sqrt{\deg(1)\deg(\phi)}.$$

We know that $\deg(1) = 1$, $\deg(\phi) = p$, and $\deg(1 - \phi) = \#E(\mathbb{F}_p)$.

Substituting these values, we get

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

which is the same as

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

□

3.2 Order of a Point

Suppose $P \in E(K)$. The order of the point P is the smallest positive integer k such that $kP = \mathcal{O}$. If no such k exists, then P is said to have infinite order. A point with finite order is called a torsion point, and if its order is exactly m, it is referred to as an m-torsion point.

In the case of elliptic curves over finite fields, all points are torsion points. This is because the group $E(\mathbb{F}_p)$ has a finite number of elements, and every element in a finite group must have finite order (result from group theory).

One way to find the order of a point P is to first determine the order of the group $\#E(\mathbb{F}_q)$. Then, using Lagrange's Theorem, which states that the order of any element in a finite group divides the order of the group, we can compute scalar multiples of P corresponding to each positive divisor of $\#E(\mathbb{F}_q)$. The smallest such value k for which $kP = \mathcal{O}$ will be the order of P. Because the order of the group can be quite large, it would be computationally expensive to check all scalar multiples of P sequentially. By using Lagrange's Theorem to narrow the search to only the divisors of the group order, we significantly reduce the number of candidates we need to test, making the process much more efficient.

Now that we have a solid understanding of the structure of elliptic curves and how to work with them, we can move on to exploring their applications in cryptography.

4 Elliptic Curve Discrete Logarithm Problem (ECDLP)

The Elliptic Curve Discrete Logarithm Problem (ECDLP) [4] is defined as follows: Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$. Suppose

$P, Q \in E(\mathbb{F}_p)$ such that point P is of finite prime order n and $Q \in \langle P \rangle$. Then, the goal is to determine an integer $k \in \mathbb{Z}_n$ such that

$$Q = kP.$$

Cryptographic protocols use this technique because it is hard to solve this problem. While computing the point multiplication kP is computationally efficient even with a large k, the inverse operation is computationally costly with current algorithms. Several algorithms exist to attempt to solve the ECDLP, which includes generic algorithms like Pollard's ρ method and index calculus algorithms. However, no known efficient algorithm exists to solve ECDLP and the computational cost grows exponentially with the size of the curve.

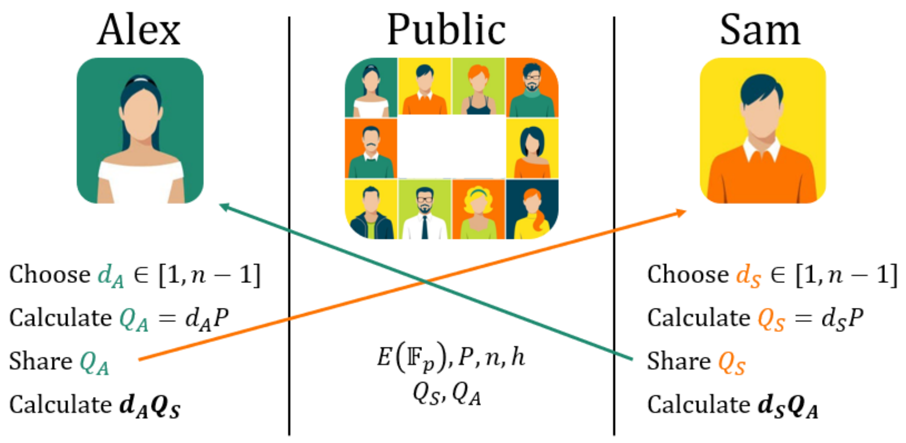
4.1 Elliptic Curve Diffie-Hellman (ECDH)

The Elliptic Curve Diffie-Hellman (ECDH) protocol allows two parties to securely exchange confidential information over an insecure channel [6]. It is based on the ECDLP.

Assume Alex and Sam want to exchange sensitive information over the internet. To begin, they agree on the following domain parameters:

- p : prime number that defines the finite field $\mathbb{F}_p$ over which the elliptic curve is defined
- a, b : coefficients that define the elliptic curve $E(\mathbb{F}_p)$: $y^2 = x^3 + ax + b \pmod{p}, 4a^3 + 27b^2 \neq 0 \pmod{p}$
- P : random point on E
- n : order of P; usually large prime to make the ECDLP harder to solve
- h: cofactor, defined as $h = \frac{\#E(\mathbb{F}_p)}{n}$ using Lagrange's Theorem; it measures how many distinct cosets of the subgroup $\langle P \rangle$ fit into the full group; typically, h is small (usually $h = 1$, or at most $h \leq 4$)

It is important to note that all of these domain parameters are publicly known [6]. Since deriving secure domain parameters can be time-consuming and computationally intensive, organizations like the Standards for Efficient Cryptography Group (SECG) publish recommended elliptic curves with precomputed, trusted parameters for participants to use.



Alex
Public
Sam
Choose $d_A \in [1, n-1]$
Calculate $Q_A = d_A P$
Share Q_A
Calculate $\boldsymbol{d_A Q_S}$
$E(\mathbb{F}_p), P, n, h$
Q_S, Q_A
Choose $d_S \in [1, n-1]$
Calculate $Q_S = d_S P$
Share Q_S
Calculate $\boldsymbol{d_S Q_A}$

Figure 3: ECDH Key Exchange

Next, each person (Alex and Sam) randomly selects a private key and then computes their respective public keys:

- Alex chooses integer $d_A \in [1, n-1]$ and then calculates $Q_A = d_A P$
- Sam chooses integer $d_S \in [1, n-1]$ and then calculates $Q_S = d_S P$

The private keys d_A and d_S are secret values known only to each person. However, the public keys Q_A and Q_S are shared publicly.

Lastly, they exchange each other's public key and compute the scalar multiplication between their own private key and the shared public key:

- Alex calculates $d_A Q_S$
- Sam calculates $d_S Q_A$

Both computations yield the same point on the elliptic curve:

$$d_A Q_S = d_A(d_S P) = d_A d_S P = d_S d_A P = d_S(d_A P) = d_S Q_A$$

The process is summarized in Figure 3. This common point becomes their shared secret. An eavesdropper who only knows the public values P, Q_A, and Q_S would need to solve for either d_A or d_S to compute the shared secret. This reduces to solving the ECDLP - for example, finding the integer d_A such that $Q_A = d_A P$, which as mentioned before is a computationally difficult task.

4.2 Naive Approach: Exhaustive Search

There are several algorithms designed to solve the Elliptic Curve Discrete Logarithm Problem. One of the simplest method is the naive approach, which computes the points $P, 2P, 3P, \ldots$ until it finds a point $kP = Q$ [2]. In the worst-case scenario, k could be close to n, meaning the algorithm would perform up to $O(n)$ elliptic curve point additions. It takes about $log_2(n)$ bits to represent a number n, so it will take $O((log_2(n))^2)$ time to do each point addition because of the cost of big integer multiplication and division. Therefore, the total time complexity of the naive method is

$$O(n \cdot (log_2(n))^2).$$

To understand how this scales with the input size, we define the input size x as the number of bits needed to represent n: $x = log_2(n)$. Then, $n = 2^x$ and the total runtime becomes

$$O(2^x \cdot x^2).$$

This shows that the naive method is exponential with input size x, which makes it extremely slow and inefficient for large values of n [2].

4.3 Pollard's Rho Algorithm

Pollard's Rho Algorithm is currently the most efficient known method for attacking the ECDLP, so the security of elliptic curve cryptography is closely tied to its performance. The algorithm generates a sequence of elliptic curve points using a pseudorandom process, and when two points in the sequence collide (when the same point is reached twice), this collision can be exploited to compute the discrete logarithm.

To understand the runtime of this algorithm, we analyze the number of steps required to find a collision. The number of ways to choose two distinct elements from a group of k elements is

$$\binom{k}{2} = \frac{k!}{2!(k-2)!} = \frac{k(k-1)}{2} \approx \frac{k^2}{2}.$$

If each pair of elements has an independent probability of $\frac{1}{n}$ of colliding, the expected number of collisions is

$$\frac{k^2}{2n}.$$

For at least one collision, we will have

$$\frac{k^2}{2n} \geq 1$$

$$k \geq \sqrt{2n}.$$

Therefore, we expect a collision (and thus a solution to the ECDLP) after about $\sqrt{n}$ steps. Each step involves point addition or doubling, which would take $O((log_2(n))^2)$ time per step. Hence, the total runtime is

$$O(\sqrt{n} \cdot (log_2(n))^2).$$

Now, if the input size is $x = log_2(n)$, then

$$O(\sqrt{n} \cdot log_2(n))^2) = O(\sqrt{2^x} \cdot (log_2(2^x))^2) = O(2^{\frac{x}{2}} \cdot x^2).$$

This shows that Pollard's algorithm also has exponential runtime [2]. However, it is important to note that Pollard's Rho grows slower than the naive method because of the base of the exponent. In Pollard's method, the base is $2^{\frac{1}{2}} = \sqrt{2}$ whereas in the naive method, the base is 2. Thus, Pollard's algorithm runs faster.

Pollard's Rho is the most efficient method for solving the ECDLP, but it's still exponential. This highlights how hard it is to crack the ECDLP and explains why agencies like the National Security Agency (NSA) have adopted ECC to protect classified information.

The procedure for Polard's rho algorithm is as follows [2]

- Firstly, we randomly pick two integers $a, b \in [0, n)$ where $n = \#E(\mathbb{F}_p)$.
- Then, we compute a point $R = aP \oplus bQ$, which will serve as the starting point for our random walk.
- We continue walking by iterating through a sequence of random operations until we land on a point R that we have visited before.
- If the point R is distinguished (we haven't seen it before), we store it along with the current a and b values.
- The next point R' is then computed according to the algorithm described later.

- When a collision occurs (when $R = R^*$ for some $R^* = cP \oplus dQ$), we retrieve c and d. We then solve the equation

  $$aP \oplus bQ = cP \oplus dQ$$

  $$(a - c)P = (d - b)Q$$

  Since $Q = kP$, we can substitute and solve for k

  $$(a - c)P = (d - b)kP$$

  $$k = \frac{a - c}{d - b} \pmod n$$

  (provided $b \not\equiv d \pmod n$)

To compute the next value of R in the walk, we use the following algorithm [2]. We define three equally sized disjoint sets S_1, S_2, S_3 to partition the elliptic curve group. These sets help determine the update rules for the current point in the random walk.

- If $R \in S_1$, then the new value of R is set to $R' = R \oplus Q$ and we increment b by 1.
- If $R \in S_2$, then the new value of R is set to $R' = 2R$ and both a and b are updated by doubling their values modulo n.
- If $R \in S_3$, then the new value of R is set to $R' = R \oplus P$ and a is incremented by 1.

Why are we defining three disjoint sets and applying different update rules for each? There are several reasons for this. First, the use of diverse update rules introduces randomness into the random walk, ensuring that the process does not follow predictable patterns. Additionally, partitioning the elliptic curve into equally sized sets helps distribute the search more evenly across the entire group, preventing any bias towards specific regions. These strategies collectively increase the probability of encountering previously visited points, thereby speeding up the collision process.

4.3.1 Why the Algorithm Works

We start by defining a base point $R = aP \oplus bQ$ for some random integers $a, b \in [0, n)$. Since $Q = kP$, we are essentially setting R to a scalar multiple of P:

$$R = aP \oplus bQ = aP \oplus b(kP) = (a + bk)P.$$

At each step of the algorithm, we update R using one of the following operations: $R \oplus P$, $2R$, or $R \oplus Q$. Each of these updates results in another scalar multiple of P.

Hence, every point generated in the sequence remains within the same cyclic subgroup. Because this subgroup has a limited number of elements, the sequence of R values must eventually repeat by the Pigeonhole Principle.

Pollard's Rho Algorithm can feel quite abstract in its theoretical form. To build intuition, we can illustrate the process with a concrete example. While the numbers in this example are deliberately small for clarity, it is important to note that practical implementations of the algorithm rely on much larger values to ensure cryptographic security.

4.3.2 Example

Let $E(\mathbb{F}_{11}) = \{(x, y) \in \mathbb{F}_{11}^2 : y^2 = x^3 + x + 3 \pmod{11}\} \cup \mathcal{O}$.

To find the points on this elliptic curve, we first list the integers $x \in [0, 11)$ in the first column of Table 1. For each x, we then compute $x^3 + x + 3$ mod 11, as shown in the second column. Next, we find the squares modulo 11 that are congruent to the values in the second column. The corresponding y values are determined based on these results, and the pairs of points (x, y) on this curve are presented in the last column.

Table 1: Points on $E(\mathbb{F}_{11}) : y^2 = x^3 + x + 3 \pmod{11}$

x	$x^3 + x + 3$	y	Points
0	3	$5, 6$	$(0, 5), (0, 6)$
1	5	$4, 7$	$(1, 4), (1, 7)$
2	$13 \equiv 2$	no solution	no solution
3	$33 \equiv 0$	0	$(3, 0)$
4	$71 \equiv 5$	$4, 7$	$(4, 4), (4, 7)$
5	$133 \equiv 1$	$1, 10$	$(5, 1), (5, 10)$
6	$225 \equiv 5$	$4, 7$	$(6, 4), (6, 7)$
7	$353 \equiv 1$	$1, 10$	$(7, 1), (7, 10)$
8	$523 \equiv 6$	no solution	no solution
9	$741 \equiv 4$	$2, 9$	$(9, 2), (9, 9)$
10	$1013 \equiv 1$	$1, 10$	$(10, 1), (10, 10)$

Thus, the points on the curve are

$\mathcal{O}$, (0,5), (0,6), (1,4), (1,7), (3,0), (4,4), (4,7), (5,1), (5,10), (6,4), (6,7), (7,1), (7,10), (9,2), (9,9), (10,1), (10,10)

Hence, $n = \#E(\mathbb{F}_{11}) = 18$.

Now, let's assume $P = (4, 4)$ and $Q = (1, 7)$. To find k such that $Q = kP$, we will use Pollard's Rho Algorithm. Note that the calculations here were done using python. The code for this is given in Appendix. We will keep track of the list of points that we have already visited using the set T, which is originally empty. We will divide our group into three disjoint sets given by

$$S_1 = \{(x, y) : x \equiv 0 \pmod 3\}$$
$$S_2 = \{(x, y) : x \equiv 1 \pmod 3\}$$
$$S_3 = \{(x, y) : x \equiv 2 \pmod 3\}$$

where x represents the x coordinate of our point.

Let's consider the following random values for a and b, $a = 2$ and $b = 3$. Then, $R = aP + bQ = 2(4, 4) \oplus 3(1, 7) = (7, 1)$.
Now, we add this point along with a and b to the set T

$$T = \{((7, 1), 2, 3)\}.$$

Since $7 \equiv 1 \pmod 3$, $(7, 1) \in S_2$. Thus, our update operation will be doubling this point: $2(7, 1) = (6, 7)$. We also double a, b: $a = 4, b = 6$. We have not yet visited $(6, 7)$, so we include these values in our set T

$$T = \{((7, 1), 2, 3), ((6, 7), 4, 6)\}$$

Since $6 \equiv 0 \pmod 3), (6, 7) \in S_1$. Hence, we add the point to Q: $(6, 7) \oplus (1, 7) = (4, 4)$. We also increment b by 1. We have not yet visited $(4, 4)$ so we add it to our set T

$$T = \{((7, 1), 2, 3), ((6, 7), 4, 6), ((4, 4), 4, 7)\}$$

As $4 \equiv 1 \pmod 3), 4 \in S_2$, so we update it by doubling it: $2(4, 4) = (7, 1)$. We also double a and b: $a = 8, b = 14$. However, already have the point $(7, 1)$ in T. This is a collision. The first time we saw $(7, 1)$, it was computed using $a = 2$ and $b = 3$. Now, we have a second set of coefficients, which we denote as $c = 8$ and $d = 14$. We use these to solve for k with the formula:

$$k = \frac{a - c}{d - b} \pmod n = \frac{2 - 8}{14 - 3} \pmod{18} = \frac{-6}{11} \pmod{18}$$

The modular inverse of $11 \mod 18$ is 5, so

$$k = (-6) \cdot 5 \pmod{18} = -30 \pmod{18} = 6$$

Finally, we have found it: $k = 6$! Therefore, we conclude that $(1, 7) = 6(4, 4)$. Using Python, when we compute $6(4, 4)$, we indeed get the point $(1, 7)$. This confirms that our implementation of Pollard's Rho algorithm successfully solves the elliptic curve discrete logarithm problem in this example.

5 Conclusion

As the use of elliptic curve cryptography continues to expand across various applications, it is important to understand the underlying mathematical framework that supports these systems. The geometric interpretation of elliptic curves provides valuable insights into how group operations are defined and how they can be manipulated. When elliptic curves are considered over finite fields, they form well-defined groups that make cryptographic operations efficient and reliable. The order of these groups plays a vital role in cryptography as selecting a large order ensures security. Additionally, tools such as the Legendre symbol and Hasse's theorem facilitate efficient computations and establish important bounds for group sizes. The order of a point is equally significant in cryptographic contexts, as we aim to select points that generate large cyclic groups, which are essential for the security of cryptographic protocols. The elliptic curve discrete logarithm problem (ECDLP) serves as the foundation for many cryptographic protocols due to its computational difficulty. The Elliptic Curve Diffie-Hellman (ECDH) protocol leverages this challenge to secure communications. While the naive method of solving the ECDLP by computing scalar multiples individually is inefficient, Pollard's Rho algorithm has emerged as the most effective known method to solve the problem, significantly improving the efficiency of ECC-based cryptographic systems. Therefore, a deep understanding of elliptic curves and the computational challenges of the ECDLP is essential for the ongoing development of secure cryptographic systems.

Appendices

```python
import random
import matplotlib.pyplot as plt
import numpy as np

# modular inverse of x in F_p
def inverse(x, p):
    # Fermat's little theorem:
    # prime p -> inverse(x) = x^(p - 2) mod p
    return pow(x, p-2, p)

def inv(x, n):
    try:
        return pow(x, -1, n)
    except ValueError:
        return None

def elliptic_add(P, Q, a, p):
    x1 = P[0]
    y1 = P[1]
    x2 = Q[0]
    y2 = Q[1]

    # (None, None) represents the identity element
    if (P == (None, None)):
        return Q
    if (Q == (None, None)):
        return P


    if(P == Q):
        m = ((3 * x1 ** 2 + a) * inverse(2 * y1, p)) % p
    else:
        # if P and Q are inverses, return identity
        if (x1 == x2):
            return (None, None)

        m = ((y2 - y1) * inverse(x2 - x1, p)) % p

    x3 = (m ** 2 - x1 - x2) % p
    y3 = (m * (x1 - x3) - y1) % p

    return (x3, y3)

def elliptic_scalar_mul(k, P, a, p):
    if(k == 0):
        return (None, None)

    if(k == 1):
        return P

    R = elliptic_add(P, P, a, p)

    for i in range(0, k - 2):
        R = elliptic_add(P, R, a, p)

    return R

def elliptic_lin_combo(k, P, l, Q, a, p):
    R = elliptic_scalar_mul(k, P, a, p)
    Q = elliptic_scalar_mul(l, Q, a, p)

    return elliptic_add(R, Q, a, p)


def plot_points(path):
    plt.figure(figsize = (8, 8))

    x = []
    y = []

    for point in path:
        x.append(point[0])
        y.append(point[1])

    plt.scatter(x, y, color="blue", zorder=5)

    for i in range(len(path) - 1):
        plt.plot([x[i], x[i + 1]], [y[i], y[i + 1]], color="red")

    plt.title("Walk Used by Pollard's Algorithm")
    plt.grid(True)
    plt.axhline(0, color='black',linewidth=1)
    plt.axvline(0, color='black',linewidth=1)

    plt.show()

def pollard_rho(P, Q, a, p, n):
    # pick two random integers k, l
    r = random.randint(0, n - 1)
    s = random.randint(0, n - 1)
    print(f"r = {r}, s = {s}")

    R = elliptic_lin_combo(r, P, s, Q, a, p)
    print(f"R = rP     sQ = {r}{P}     {s}{Q} = {R}\n")

    # T: set of visited points
    T = [[], [], []]

    while(R not in T[0]):
        if(R == (None, None)):
            return -1

        T[0].append(R)
        T[1].append(r)
        T[2].append(s)

        print(f"Add R{R} to T:")
        print(f"T = {T}")

        if(R[0] % 3 == 0):
            print(f"{R[0]} is 0 mod 3 -> add Q{Q} to {R}")
            R = elliptic_add(Q, R, a, p)
            s = (s + 1) % n
        elif(R[0] % 3 == 1):
            print(f"{R[0]} is 1 mod 3 -> double {R}")
            R = elliptic_scalar_mul(2, R, a, p)
            r = (2 * r) % n
            s = (2 * s) % n
        else:
            print(f"{R[0]} is 2 mod 3 -> add P{P} to {R}")
            R = elliptic_add(P, R, a, p)
            r = (r + 1) % n

        print(f"R = {R}, r = {r}, s = {s}\n")

        if(R in T[0]):
            print("Collision!\n")

    index = T[0].index(R)
    r1 = T[1][index]
    s1 = T[2][index]

    r2 = r
    s2 = s

    print(f"Q = kP, k = (({r1} - {r2}) / ({s2} - {s1})) mod {n}")

    num = (r1 - r2) % n
    den = (s2 - s1) % n

    if(inv(den, n) is None):
        return -1

    T[0].append(R)
    plot_points(T[0])

    return (num * inv(den, n)) % n

while(True):
    print("(1) P      Q")
    print("(2) kP")
    print("(3) kP      lQ")
    print("(4) Pollard's rho-algorithm")
    print("(5) Quit")
    option = int(input("Choose an option: "))

    if(option > 5 or option < 1):
        print("invalid input")
        break;
    if(option == 5):
        break;

    a = int(input("a: "))
    p = int(input("p: "))
    P_str = input("P: ")
```

```python
    P = tuple(map(int, P_str.split(',')))

    if(option == 1):
        Q_str = input("Q: ")
        Q = tuple(map(int, Q_str.split(',')))
        print(f"P     Q = {P}     {Q} = {elliptic_add(P, Q, a, p)}")
    elif(option == 2):
        k = int(input("k: "))
        print(f"kP = {k}{P} = {elliptic_scalar_mul(k, P, a, p)}")
    elif(option == 3):
        k = int(input("k: "))
        Q_str = input("Q: ")
        Q = tuple(map(int, Q_str.split(',')))
        l = int(input("l: "))
        print(f"kP     lQ = {k}{P}     {l}{Q} = {elliptic_lin_combo(k, P, l, Q, a, p)}")
    elif(option == 4):
        Q_str = input("Q: ")
        Q = tuple(map(int, Q_str.split(',')))
        n = int(input("n: "))
        print(f"k = {pollard_rho(P, Q, a, p, n)}")

    print()
```

Listing 1: Elliptic Curve Operations

References

[1] Fujii, K., and Oike, H. "An Algebraic Proof of the Associative Law of Elliptic Curves." *Advances in Pure Mathematics*, vol. 7, 2017, pp. 649-659. www.scirp.org/pdf/APM_2017121113570614.pdf.

[2] Gill, Puneet. *Solving Elliptic Curve Discrete Logarithm Problem Using Parallelized Pollard's Rho and Lambda Methods.* May 18, 2019. ece.uwaterloo.ca/~p24gill/Projects/Cryptography/Pollard's_Rho_and_Lambda/Project.pdf.

[3] Gruska, Jozef. "Elliptic Curves Cryptography and Factorization." *Department of Computer Science, Masaryk University*, 2011, www.fi.muni.cz/usr/gruska/crypto11/crypto_08.pdf.

[4] Menezes, Alfred. "Evaluation of Security Level of Cryptography: The Elliptic Curve Discrete Logarithm Problem (ECDLP)." University of Waterloo, 14 Dec. 2001. citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=352185778c4b6f553092025a18ea737c8d790c72.

[5] Rothaar, Teresa. "What Is Elliptic Curve Cryptography?" *Keeper Security Blog*, 7 June 2023. www.keepersecurity.com/blog/2023/06/07/what-is-elliptic-curve-cryptography/.

[6] Shevchuk, Olga. *Introduction to Elliptic Curve Cryptography.* 2020. Accessed August 16th, 2020. math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf.

[7] Silverman, Joseph H. *The Arithmetic of Elliptic Curves.* 2nd ed., vol. 106, Graduate Texts in Mathematics, Springer, 2009. *DOI*: doi.org/10.1007/978-0-387-09494-6.

[8] Winkler, Nolan. "The Discrete Log Problem and Elliptic Curve Cryptography." *University of Chicago*, 2014, math.uchicago.edu/~may/REU2014/REUPapers/Winkler.pdf.