Elliptic Curve Cryptography

JU IT 4th year Project

Presented By : Group 27

Advisor : Utpal Kumar Roy,
Sujay Paul

Introduction to Elliptic Curve Cryptography

- **What is ECC?**
  Elliptic Curve Cryptography (ECC) is a form of public-key cryptography based on the algebraic properties of elliptic curves. Unlike traditional systems like RSA, which relies on large integer factorisation, ECC provides comparable security with smaller key sizes. This efficiency makes ECC ideal for systems with limited computing power and storage, such as mobile devices and IoT applications.

- **Key Benefits :**
  - **Smaller Key Sizes:** ECC can achieve the same level of security as RSA with much smaller keys. For example, a 256-bit ECC key offers similar security to a 3072-bit RSA key.
  - **Efficient Performance:** Smaller keys mean faster encryption/decryption and lower computational costs.
  - **High Security:** The discrete logarithm problem in the context of elliptic curves is difficult to solve, making ECC highly secure.

- **Applications of ECC:**
  - ECC is widely used in securing internet traffic (TLS/SSL), digital signatures (ECDSA), and key exchange protocols (ECDH), ensuring secure communication across networks.

Elliptic Curve Basics

- **Elliptic Curve Equation :**
  The elliptic curve equation for cryptographic purposes generally takes the form $y^2 = x^3 + ax + b$ (known as the Weierstrass form), with specific conditions on a and b to avoid singularities. In cryptographic use, calculations are done over a finite field, typically a prime field $GF(p)$

- **Parameters Explained :**
  - a and b: Constants defining the specific shape and properties of the curve. These values must satisfy $4a^3 + 27b^2 \neq 0$ to ensure the curve is non-singular.
  - p: Prime modulus, defining the finite field over which the curve operates. The field size impacts the curve's security strength.

- **Geometric Interpretation :**
  Points on an elliptic curve have a unique group structure: two points can be 'added' to produce a third point on the curve. This property enables ECC's cryptographic applications. The elliptic curve group's 'point at infinity' acts as the identity element for this addition operation.

Project Overview

- **Project Goals :**
  - Point Generation: Calculate all points on a specified elliptic curve to enable visualization.
  - Curve Order Calculation: Determine the total number of points on the curve, which is key for cryptographic strength.
  - Point Order Calculation: Find the order of specific points, relevant for protocols that rely on point multiplication, like ECDSA.

- **Project Components :**

Code Modules: Each module is designed for a particular function. For instance

  — elliptic_curve.py handles generating and plotting points.
  — find_curve_order.py calculates the overall order of the elliptic curve.
  — find_order_of_a_point.py finds the order of specific points on the curve.

Module 1: Point Generation (elliptic_curve.py)

- **Objective :**
  Generate all points on the elliptic curve by iterating through possible x-values and finding corresponding y-values that satisfy the elliptic curve equation.

- **Key Functions :**

  elliptic_curve_points() : Iterates through all integers x in the range [0, $p-1$], computes $x^3 + ax + b$, and finds all y-values that satisfy $y^2 \equiv x^3 + ax + b \ (mod\ p)$.

  find_square_root() : Determines the square root modulo p, using methods like Euler's Criterion or Tonelli-Shanks, ensuring only valid points are plotted.

  plot_elliptic_curve() : Generates a visual representation of the curve points, aiding in understanding the curve's structure.

- **Process :**
  This module uses brute-force methods to check every x-value modulo p, generating a plot to illustrate the elliptic curve's point distribution.

Module 2: Curve Order Calculation (find_curve_order.py)

- **Objective :**
  Calculate the number of points on the elliptic curve, known as the curve's order, including the point at infinity.

- **Key Functions :**
  check_prime() : Ensures p is a valid prime, ideally of the form $p \equiv 3 \pmod 4$ for efficient calculation. If p isn't prime, it increments until finding a valid prime, ensuring accuracy.
  find_points() : Calls *elliptic_curve_points()* to generate all valid points and then counts them to determine the curve order.

- **Workflow:**
  Input the curve parameters and validate p. The code calculates and counts the valid points to find the order, which is important for cryptographic calculations.

Module 3: Point Order Calculation (find_order_of_a_point.py)

- **Objective :**
  Calculate the order of a specific point on the curve, which is the smallest positive integer n such that nP = O (the point at infinity).

- **Key Functions :**
  add_point() : Adds two points P and Q on the elliptic curve using elliptic curve addition rules. Handles the cases of identical points (point doubling) and inverse points (resulting in the point at infinity).

  double_point() : Handles the addition of a point to itself, a critical operation in ECC key generation.

  find_order() : Repeatedly applies the *add_point()* function until it reaches the point at infinity, giving the order of the point.

- **Significance of Point Order :**
  Knowing the order is essential for ECC-based systems like the Elliptic Curve Digital Signature Algorithm (ECDSA), as it directly influences the security and efficiency of point multiplications.

Testing the Implementation (test_elliptic_curve.py)

- **Objective :**
  Validate each component's correctness, particularly the point generation and point order functions.

- **Test Cases :**

  — Testing is performed over various combinations of a, b, and p to ensure accuracy across different elliptic curves.

  — Known test cases with predictable outputs, such as specific curves with known point counts, help verify the implementation.

- **Testing Methodology :**
  The unit test framework automates these tests, helping catch any logical or implementation errors in point generation and calculations.

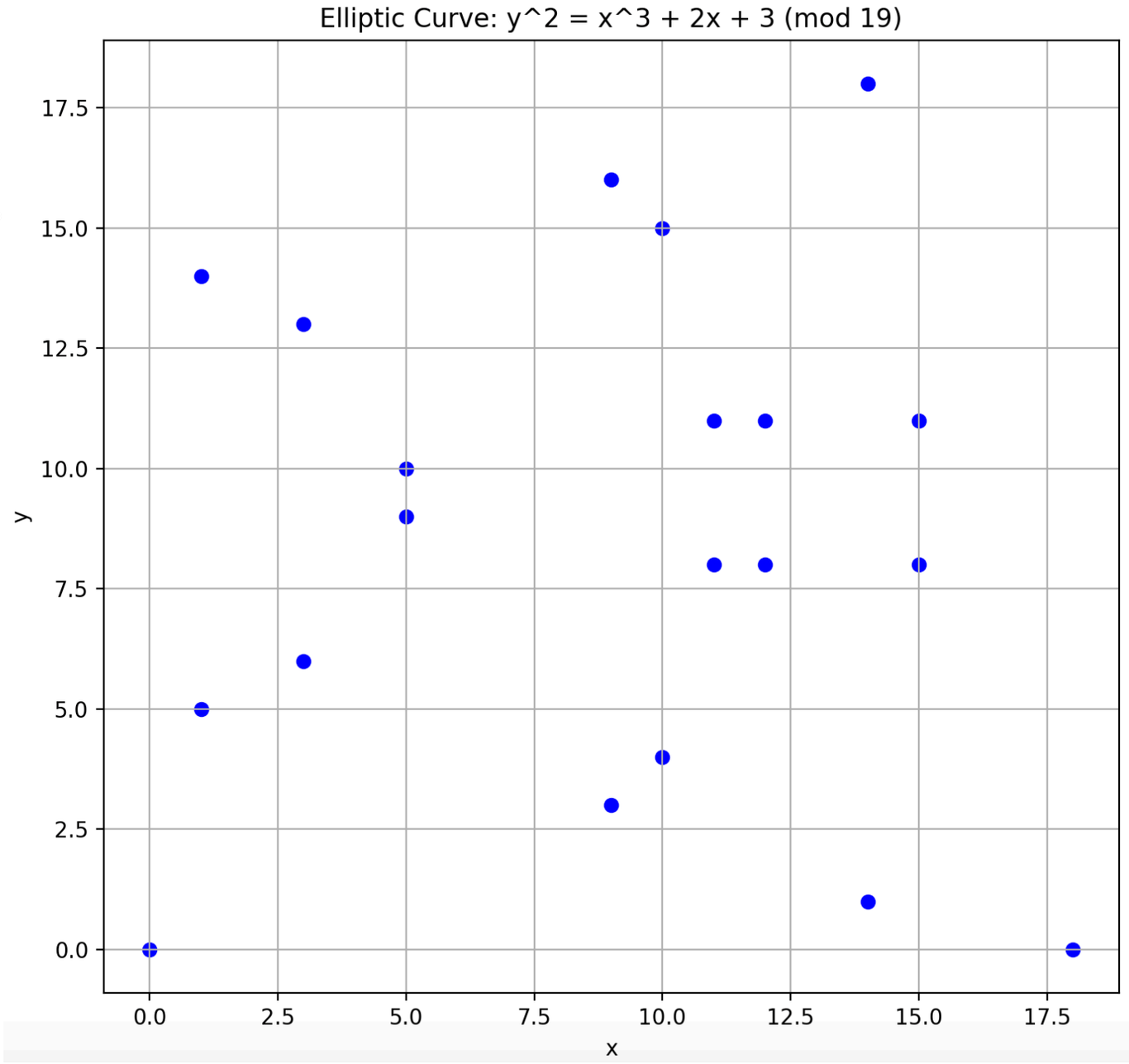
Example: Curve Visualization

- **Example Parameters :**
  Use specific parameters such as a = 2, b = 3, p = 19, and visualize the curve.

- **Visualization Purpose :**
  Seeing the points visually can help understand elliptic curve group behaviour, showing how points 'wrap' in a finite field, which underpins ECC's security.

```
Using Euler's formula for p = 19 (p ≡ 3 mod 4)
Using Euler's formula for p = 19 (p ≡ 3 mod 4)
Elliptic Curve Points (x, y):
(1, 5)
(1, 14)
(3, 6)
(3, 13)
(5, 9)
(5, 10)
(9, 16)
(9, 3)
(10, 4)
(10, 15)
(11, 11)
(11, 8)
(12, 11)
(12, 8)
(14, 1)
(14, 18)
(15, 11)
(15, 8)
(18, 0)
(0, 0)

Total number of points on the curve : 20
```



Results and Observations

- **Point Generation Results :**
  Confirmed that the code generates valid points for various configurations. Shows expected behaviour in terms of finite field properties.
- **Curve Order Findings :**
  Observed that different combinations of a, b, and p affect the order, which is critical for assessing cryptographic strength.
- **Point Order Calculation :**
  Accurate determination of point orders demonstrated reliability for potential ECC applications, such as secure key exchanges.

Conclusion and Future Scope

- **Project Summary :**

  Successfully demonstrated ECC's principles through code, including point generation, curve visualization, and order calculation, showing how ECC's mathematical foundation translates into secure cryptography.

- **Future Enhancements :**

  – ECDSA Implementation: Create digital signatures that authenticate data integrity and origin.

  – Elliptic Curves over Binary Fields: Extend ECC implementation to binary fields (used in some IoT devices and smart cards).

  – Advanced Visualization: Improve visual feedback for more complex field sizes, enhancing educational value.