The Elliptic Curves Discrete Logarithm Problem and an implementation of parallelized Pollard's ρ algorithm for ECDLP

Alberto Pizzirani

Università degli Studi di Napoli "Federico II"

Outline

Who wants to be a millionaire ?

- In a similar way of what was done by **RSA Security** in the past, the society **CERTICOM**, in 1997, published 'The Certicom Elliptic Curve Cryptography Challenge'.
- This is a list of instances of the discrete logarithm problem on elliptic curves and the resolution of each problem is awarded with a prize (ranging from a copy of the book *The Handbook of Applied Cryptography* and a copy of *Maple V software* to *$100.000*).
- The main reasons of this list are to enhance research on this topic and to test the security of the cryptographic systems based on elliptic curves.

Who wants to be a millionaire ?

- In a similar way of what was done by **RSA Security** in the past, the society **CERTICOM**, in 1997, published 'The Certicom Elliptic Curve Cryptography Challenge'.
- This is a list of instances of the discrete logarithm problem on elliptic curves and the resolution of each problem is awarded with a prize (ranging from a copy of the book *The Handbook of Applied Cryptography* and a copy of *Maple V software* to *$100.000*).
- The main reasons of this list are to enhance research on this topic and to test the security of the cryptographic systems based on elliptic curves.

Who wants to be a millionaire ?

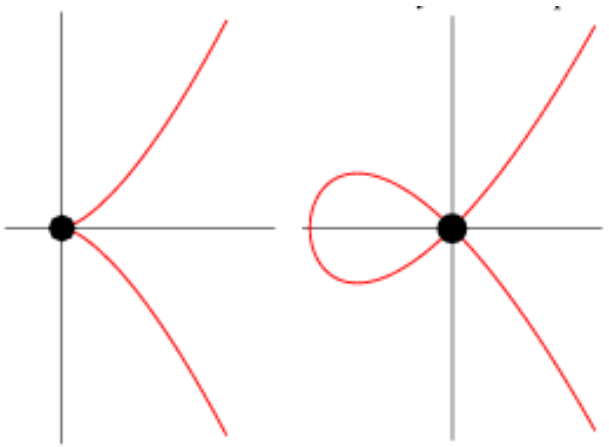
- In a similar way of what was done by **RSA Security** in the past, the society **CERTICOM**, in 1997, published 'The Certicom Elliptic Curve Cryptography Challenge'.
- This is a list of instances of the discrete logarithm problem on elliptic curves and the resolution of each problem is awarded with a prize (ranging from a copy of the book *The Handbook of Applied Cryptography* and a copy of *Maple V software* to *$100.000*).
- The main reasons of this list are to enhance research on this topic and to test the security of the cryptographic systems based on elliptic curves.

Elliptic curves over reals.

- $y^2 = x^3 + Ax + B$ with x, y, A and $B \in \mathbb{R}$
- without 'singular' points (cusps, knots)

Elliptic curves over reals.

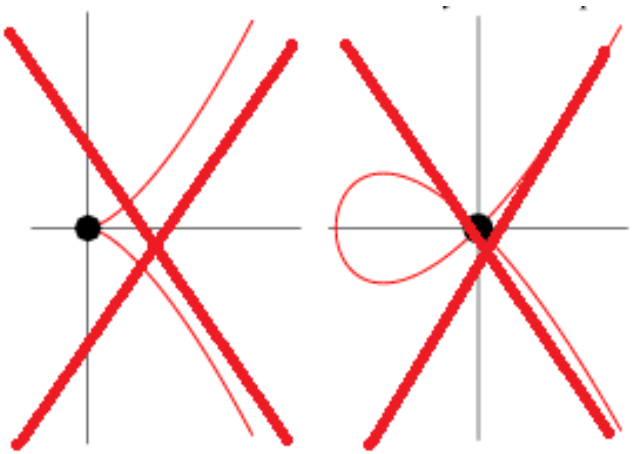
- $y^2 = x^3 + Ax + B$ with x, y, A and $B \in \mathbb{R}$
- without 'singular' points (cusps, knots)



Cusp Knot

Elliptic curves over reals.

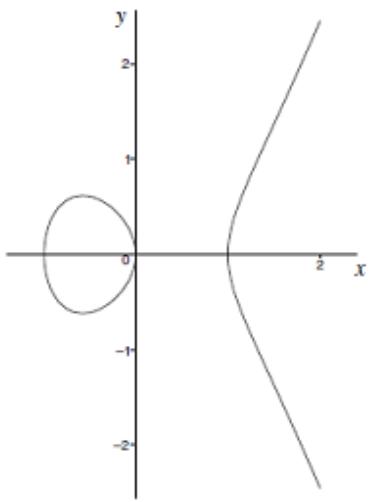
- $y^2 = x^3 + Ax + B$ with x, y, A and $B \in \mathbb{R}$
- without 'singular' points (cusps, knots)



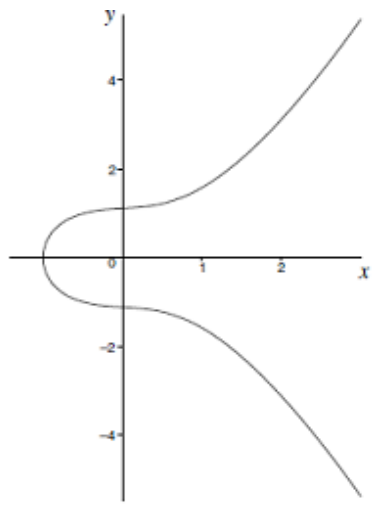
Cusp Knot

Elliptic curves over reals.

- $y^2 = x^3 + Ax + B$ with x, y, A and $B \in \mathbb{R}$
- without 'singular' points (cusps, knots)



$E_1 : y^2 = x^3 - x$



$E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

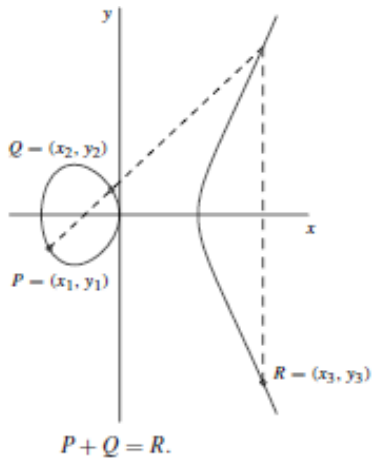
The elliptic group.

- Over the set $E(\mathbb{R}) := \{(x, y) : y^2 = x^3 + Ax + B\} \cup \{O_\infty\}$ can be defined an addition $+$ between points using the so-called 'chord-tangent' method.

- $(E(\mathbb{R}), +)$ is an abelian group, called **elliptic group over** $\mathbb{R}$.

The elliptic group.

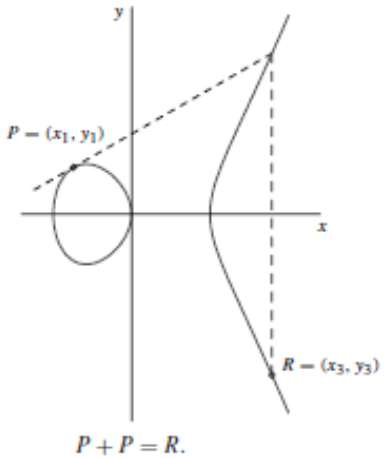
- Over the set $E(\mathbb{R}) := \{(x, y) : y^2 = x^3 + Ax + B\} \cup \{O_\infty\}$ can be defined an addition $+$ between points using the so-called 'chord-tangent' method.



- $(E(\mathbb{R}), +)$ is an abelian group, called **elliptic group over** $\mathbb{R}$.

The elliptic group.

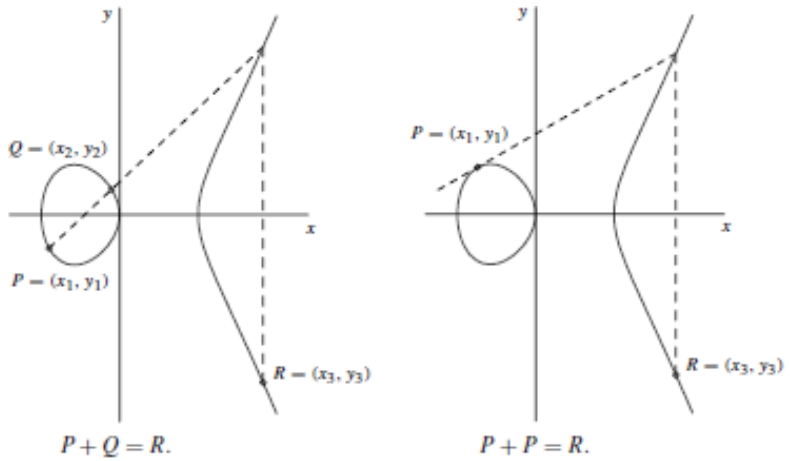
- Over the set $E(\mathbb{R}) := \{(x, y) : y^2 = x^3 + Ax + B\} \cup \{O_\infty\}$ can be defined an addition $+$ between points using the so-called 'chord-tangent' method.



- $(E(\mathbb{R}), +)$ is an abelian group, called **elliptic group over** $\mathbb{R}$.

The elliptic group.

- Over the set $E(\mathbb{R}) := \{(x, y) : y^2 = x^3 + Ax + B\} \cup \{O_\infty\}$ can be defined an addition $+$ between points using the so-called 'chord-tangent' method.



- $(E(\mathbb{R}), +)$ is an abelian group, called **elliptic group over** $\mathbb{R}$.

Explicit formulas for points addition.

- If $P = (x_1, y_1)$,$Q = (x_2, y_2)$ and $R = (x_3, y_3)$, then:
  - $P + Q = R$:
    $$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad , \quad y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$
  - $2P = R$:
    $$x_3 = \left(\frac{3x_1^2 + A}{2y_1}\right)^2 - 2x_1 \quad , \quad y_3 = -y_1 + \left(\frac{3x_1^2 + A}{2y_1}\right)(x_1 - x_3)$$
- These formulas, here defined for $\mathbb{R}$, can be redefined through the operations of a field $\mathbb{K}$ (for example finite fields $\mathbb{F}_p$ with $p > 3$ prime). In these cases we still have an elliptic group (that can be denoted with $E(\mathbb{F}_p)$ over finite fields).

Explicit formulas for points addition.

- If $P = (x_1, y_1)$,$Q = (x_2, y_2)$ and $R = (x_3, y_3)$, then:
  - $P + Q = R$:
    $$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad , \qquad y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$
  - $2P = R$:
    $$x_3 = \left(\frac{3x_1^2 + A}{2y_1}\right)^2 - 2x_1 \quad , \qquad y_3 = -y_1 + \left(\frac{3x_1^2 + A}{2y_1}\right)(x_1 - x_3)$$
- These formulas, here defined for $\mathbb{R}$, can be redefined through the operations of a field $\mathbb{K}$ (for example finite fields $\mathbb{F}_p$ with $p > 3$ prime). In these cases we still have an elliptic group (that can be denoted with $E(\mathbb{F}_p)$ over finite fields).

Explicit formulas for points addition.

- If $P = (x_1, y_1), Q = (x_2, y_2)$ and $R = (x_3, y_3)$, then:
  - $P + Q = R$:
    $$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad , \qquad y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$
  - $2P = R$:
    $$x_3 = \left(\frac{3x_1^2 + A}{2y_1}\right)^2 - 2x_1 \quad , \qquad y_3 = -y_1 + \left(\frac{3x_1^2 + A}{2y_1}\right)(x_1 - x_3)$$
- These formulas, here defined for $\mathbb{R}$, can be redefined through the operations of a field $\mathbb{K}$ (for example finite fields $\mathbb{F}_p$ with $p > 3$ prime). In these cases we still have an elliptic group (that can be denoted with $E(\mathbb{F}_p)$ over finite fields).

Explicit formulas for points addition.

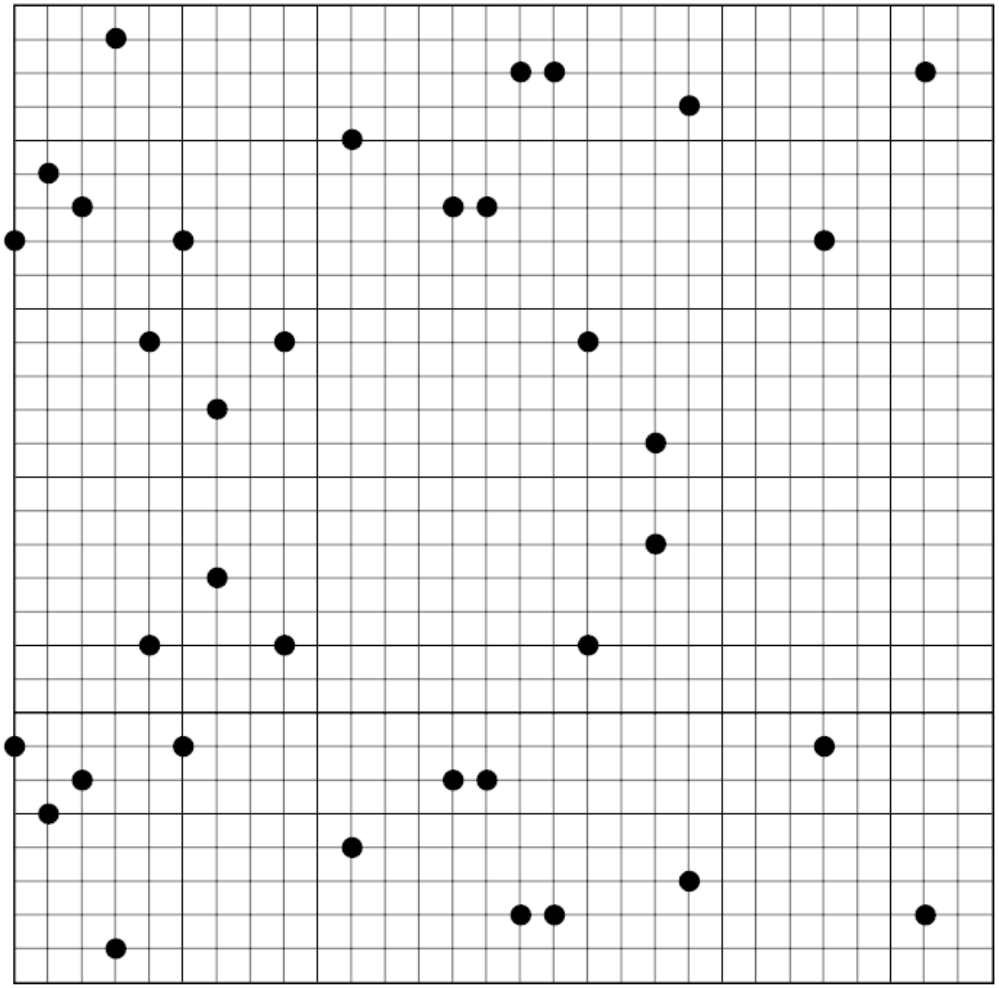
- If $P = (x_1, y_1)$,$Q = (x_2, y_2)$ and $R = (x_3, y_3)$, then:
  - $P + Q = R$:
    $$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad , \qquad y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$
  - $2P = R$:
    $$x_3 = \left(\frac{3x_1^2 + A}{2y_1}\right)^2 - 2x_1 \quad , \qquad y_3 = -y_1 + \left(\frac{3x_1^2 + A}{2y_1}\right)(x_1 - x_3)$$
- These formulas, here defined for $\mathbb{R}$, can be redefined through the operations of a field $\mathbb{K}$ (for example finite fields $\mathbb{F}_p$ with $p > 3$ prime). In these cases we still have an elliptic group (that can be denoted with $E(\mathbb{F}_p)$ over finite fields).

Elliptic Curves over finite fields (1/3).

This is an example of the elliptic curve $y^2 = x^3 + 4x + 20$ over field $\mathbb{F}_{29}$:

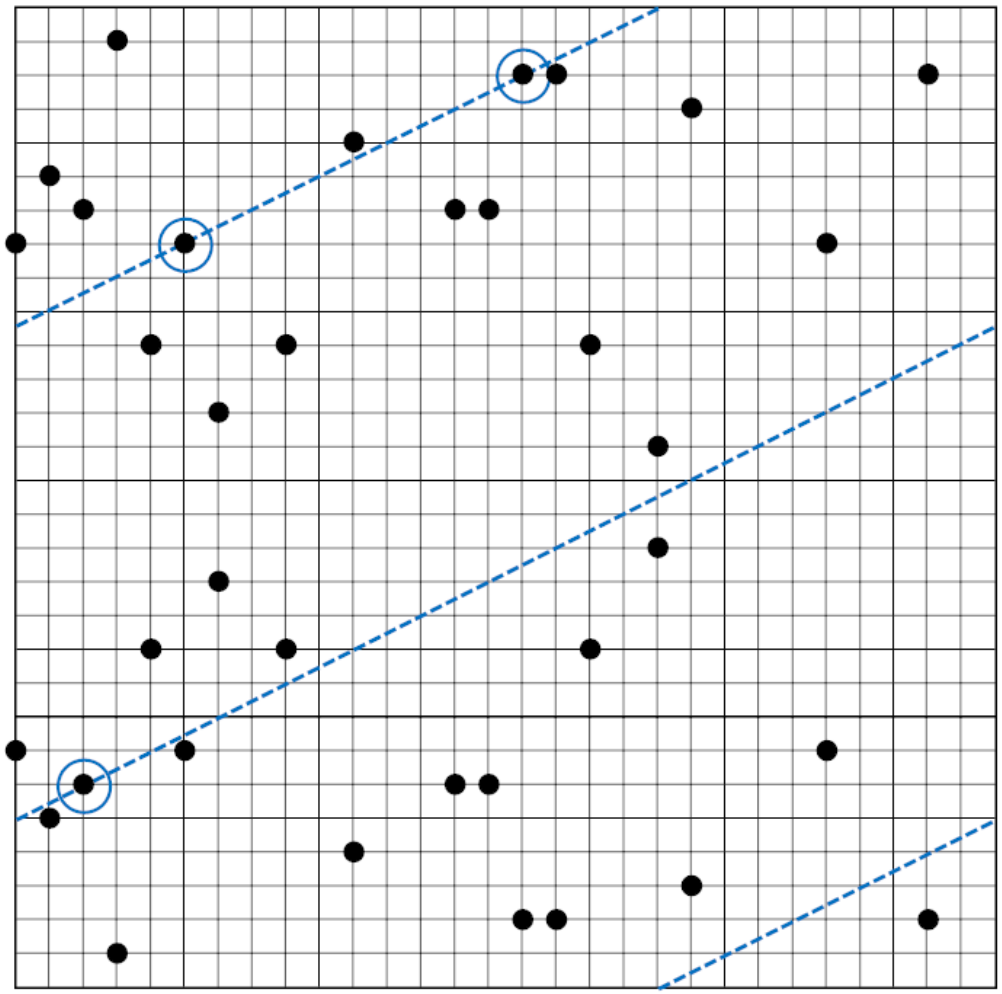
Elliptic Curves over finite fields (1/3).

This is an example of the elliptic curve $y^2 = x^3 + 4x + 20$ over field $\mathbb{F}_{29}$:



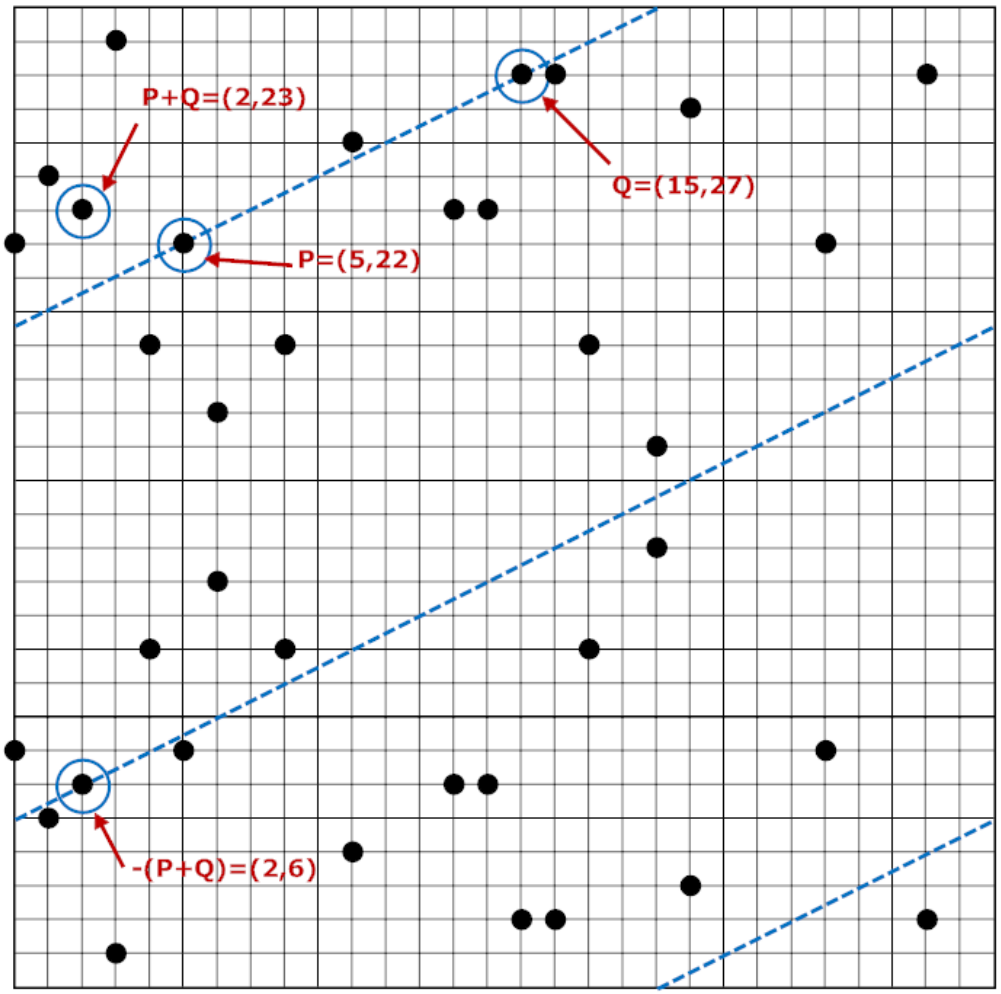
Elliptic Curves over finite fields (2/3).

The line through $P = (5, 22)$ and $Q = (15, 27)$ is:



Elliptic Curves over finite fields (3/3).

The sum of P and Q is:



Success of the Elliptic Curves Cryptography.

- In the middle of '80s, Neal Koblitz and Victor Miller (independently) proposed elliptic curves defined on finite fields as a base for a cryptosystem.
- Three main reasons caused an increasing interest in elliptic curve cryptography:
  - Large amount of elliptic groups for each finite field (Hasse's theorem and Deuring's theorem).
  - Subexponential time attacks for problems on which relie other cryptosystems (e.g. R.S.A.).
  - Fast arithmetic.

Success of the Elliptic Curves Cryptography.

- In the middle of '80s, Neal Koblitz and Victor Miller (independently) proposed elliptic curves defined on finite fields as a base for a cryptosystem.
- Three main reasons caused an increasing interest in elliptic curve cryptography:
  - Large amount of elliptic groups for each finite field (Hasse's theorem and Deuring's theorem).
  - Subexponential time attacks for problems on which relie other cryptosystems (e.g. R.S.A.).
  - Fast arithmetic.

Success of the Elliptic Curves Cryptography.

- In the middle of '80s, Neal Koblitz and Victor Miller (independently) proposed elliptic curves defined on finite fields as a base for a cryptosystem.
- Three main reasons caused an increasing interest in elliptic curve cryptography:
  - Large amount of elliptic groups for each finite field (Hasse's theorem and Deuring's theorem).
  - Subexponential time attacks for problems on which relie other cryptosystems (e.g. R.S.A.).
  - Fast arithmetic.

Success of the Elliptic Curves Cryptography.

- In the middle of '80s, Neal Koblitz and Victor Miller (independently) proposed elliptic curves defined on finite fields as a base for a cryptosystem.
- Three main reasons caused an increasing interest in elliptic curve cryptography:
  - Large amount of elliptic groups for each finite field (Hasse's theorem and Deuring's theorem).
  - Subexponential time attacks for problems on which relie other cryptosystems (e.g. R.S.A.).
  - Fast arithmetic.

Success of the Elliptic Curves Cryptography.

- In the middle of '80s, Neal Koblitz and Victor Miller (independently) proposed elliptic curves defined on finite fields as a base for a cryptosystem.
- Three main reasons caused an increasing interest in elliptic curve cryptography:
  - Large amount of elliptic groups for each finite field (Hasse's theorem and Deuring's theorem).
  - Subexponential time attacks for problems on which relie other cryptosystems (e.g. R.S.A.).
  - Fast arithmetic.

The discrete logarithm in elliptic groups.

The elliptic curve cryptosystems relie their security above all on the so-called **Elliptic Curve Discrete Logarithm Problem**:
Given two points P and Q belonging to a curve $E(\mathbb{K})$, find (if there's one) the integer k such that $Q = kP$. Such k is called (discrete) logarithm of Q in base P.

All the problems in the Certicom list are instances of ECDLP (Elliptic Curve Discrete Logarithm Problem).

The discrete logarithm in elliptic groups.

The elliptic curve cryptosystems relie their security above all on the so-called **Elliptic Curve Discrete Logarithm Problem**: *Given two points P and Q belonging to a curve $E(\mathbb{K})$, find (if there's one) the integer k such that $Q = kP$. Such k is called (discrete) logarithm of Q in base P.*

All the problems in the Certicom list are instances of ECDLP (Elliptic Curve Discrete Logarithm Problem).

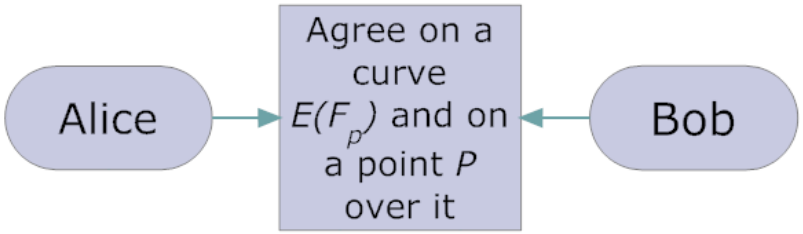
The discrete logarithm in elliptic groups.

The elliptic curve cryptosystems relie their security above all on the so-called **Elliptic Curve Discrete Logarithm Problem**: *Given two points P and Q belonging to a curve $E(\mathbb{K})$, find (if there's one) the integer k such that $Q = kP$. Such k is called (discrete) logarithm of Q in base P.*

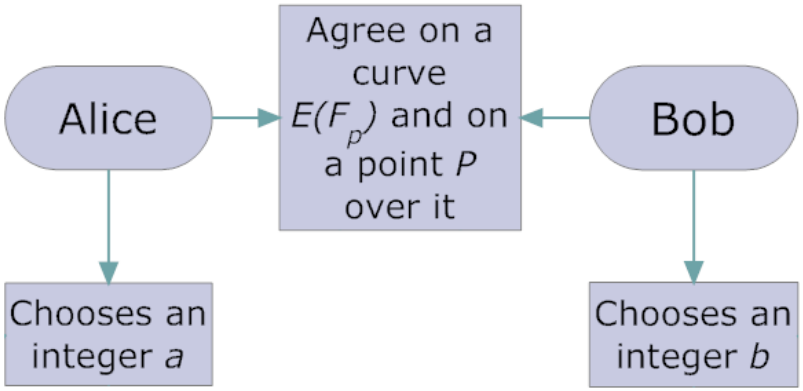
All the problems in the Certicom list are instances of ECDLP (Elliptic Curve Discrete Logarithm Problem).

Diffie-Hellman key exchange with elliptic curves.

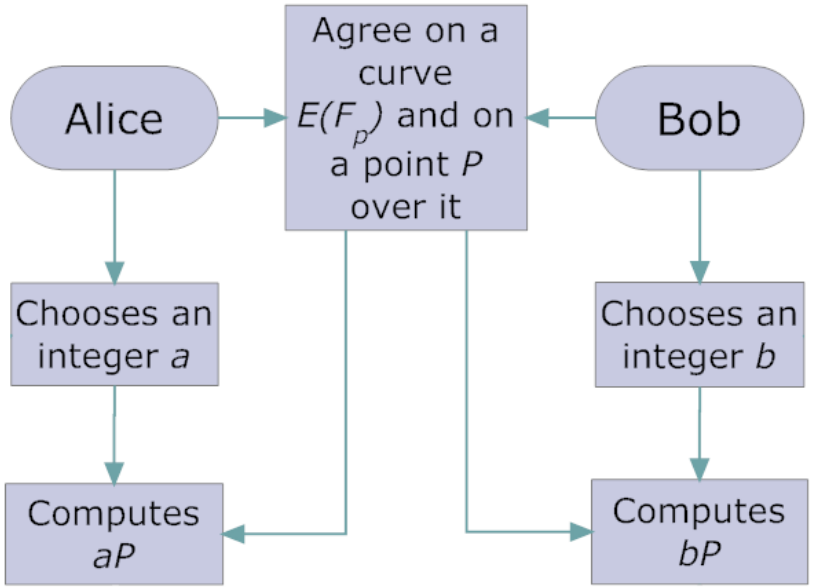
Diffie-Hellman key exchange with elliptic curves.



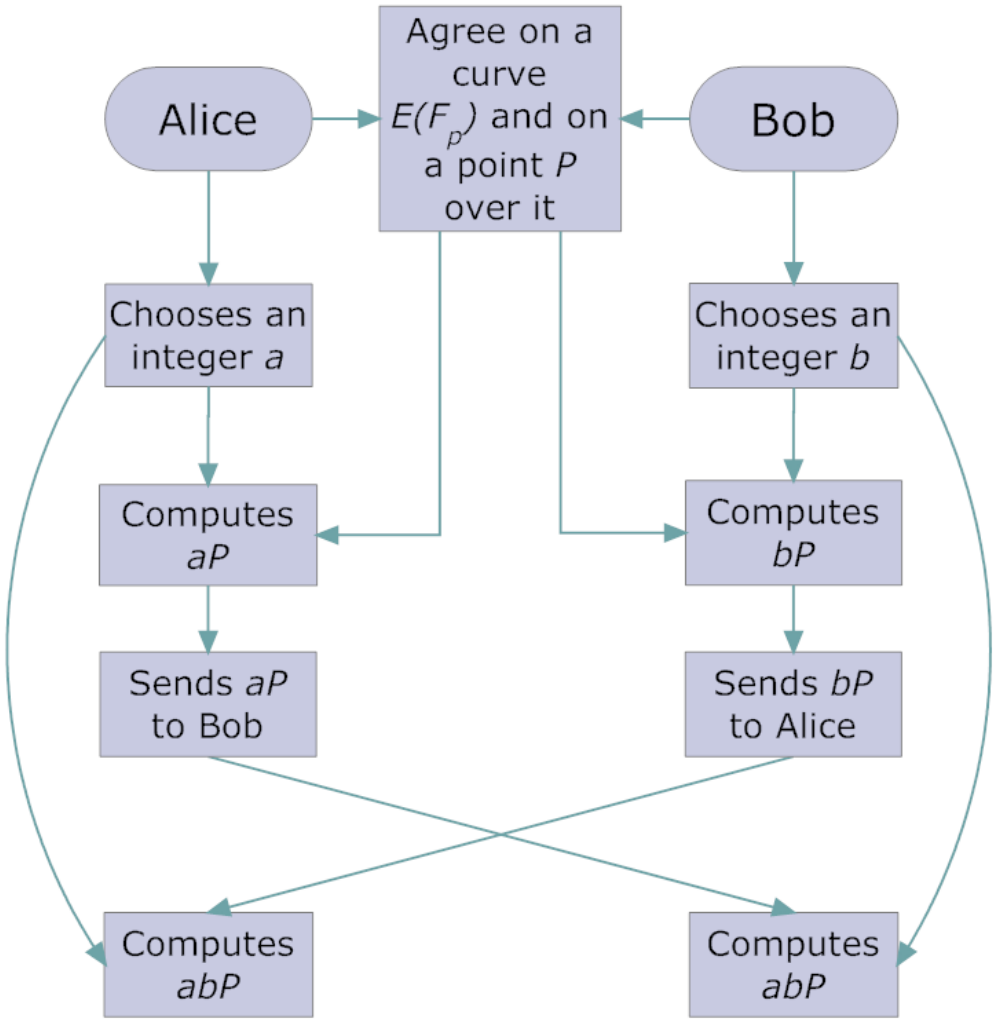
Diffie-Hellman key exchange with elliptic curves.



Diffie-Hellman key exchange with elliptic curves.

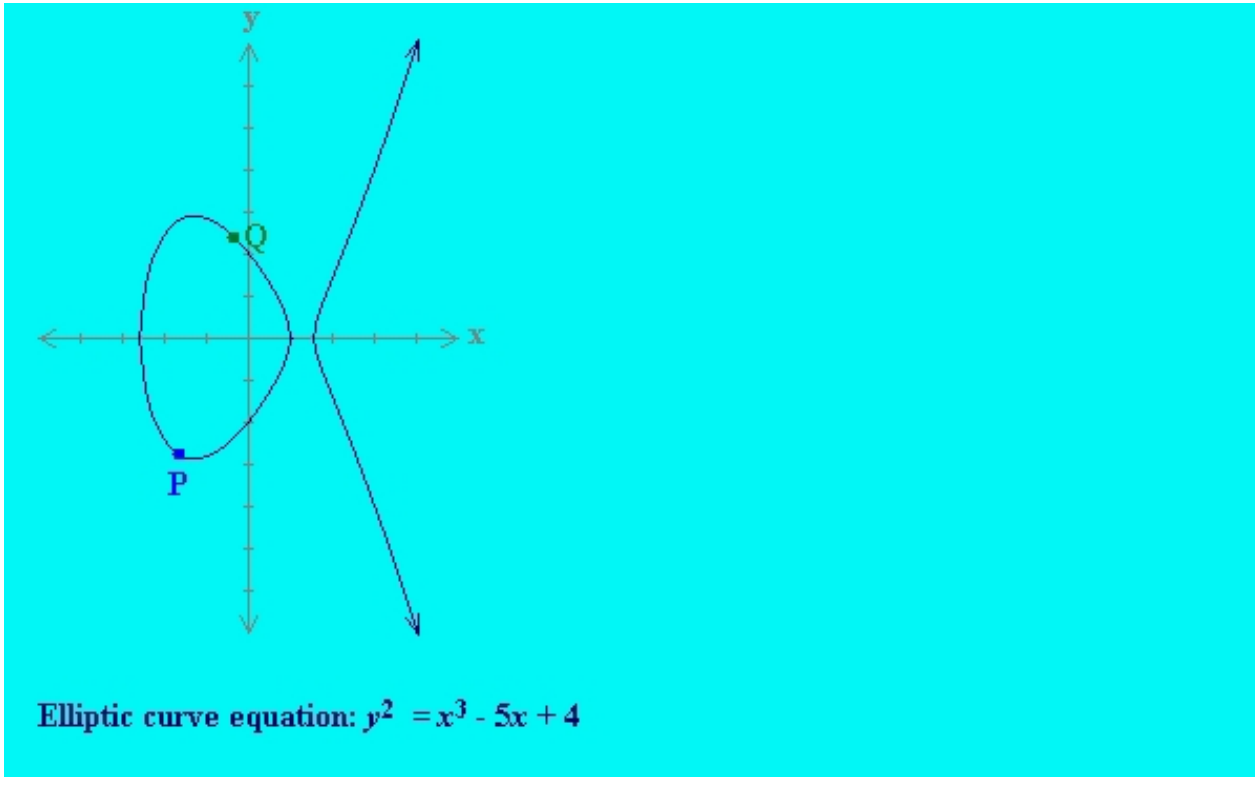


Diffie-Hellman key exchange with elliptic curves.



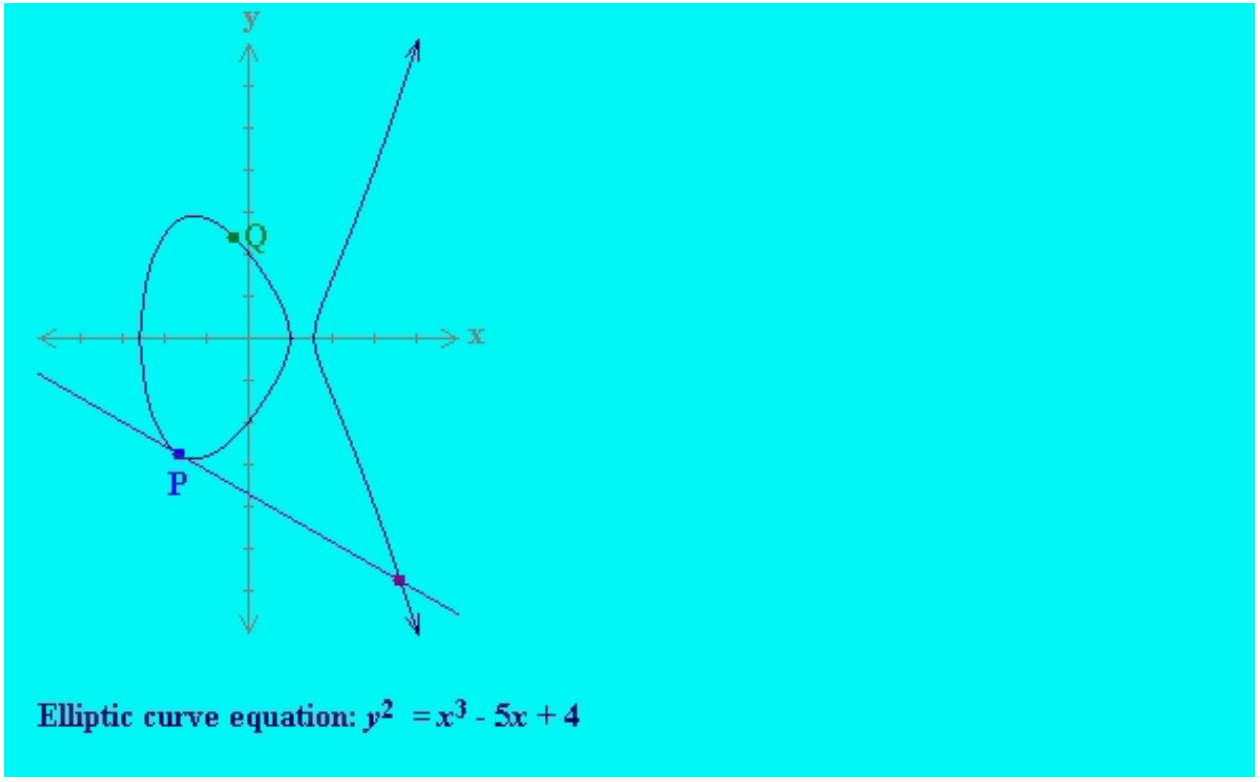
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



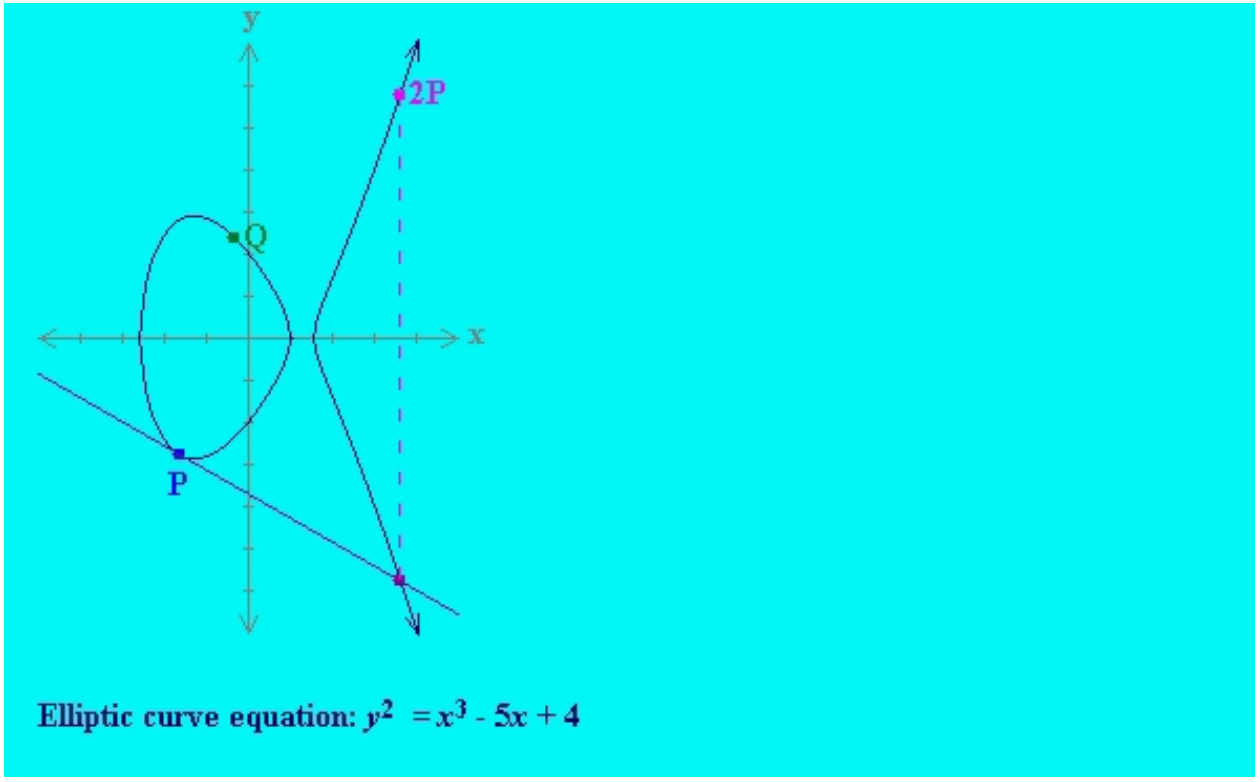
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



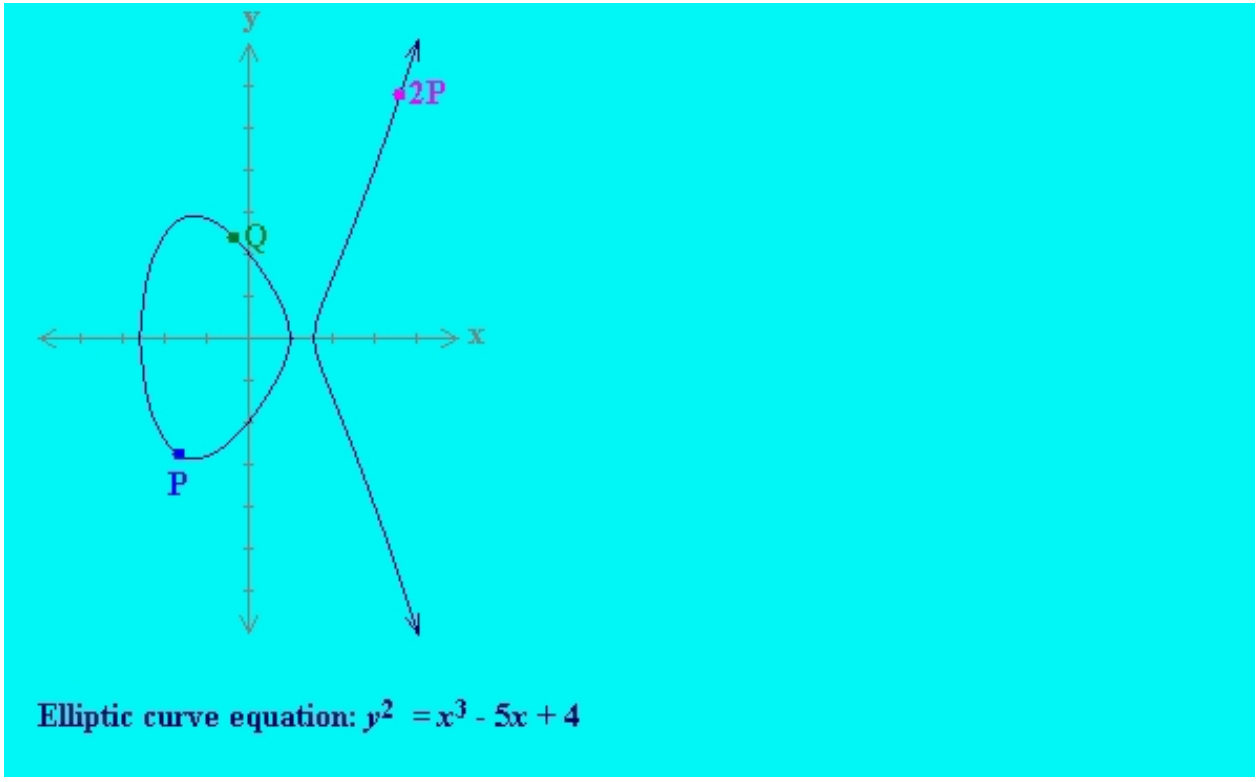
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



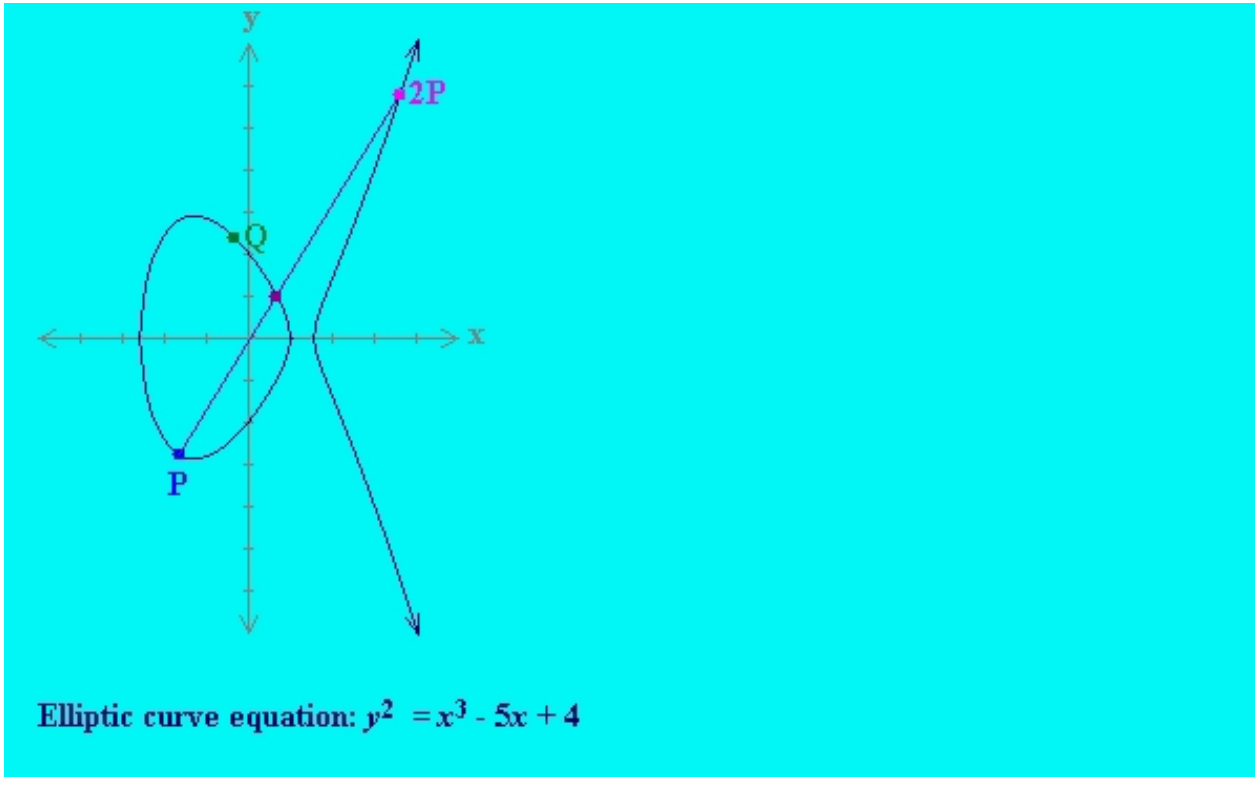
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



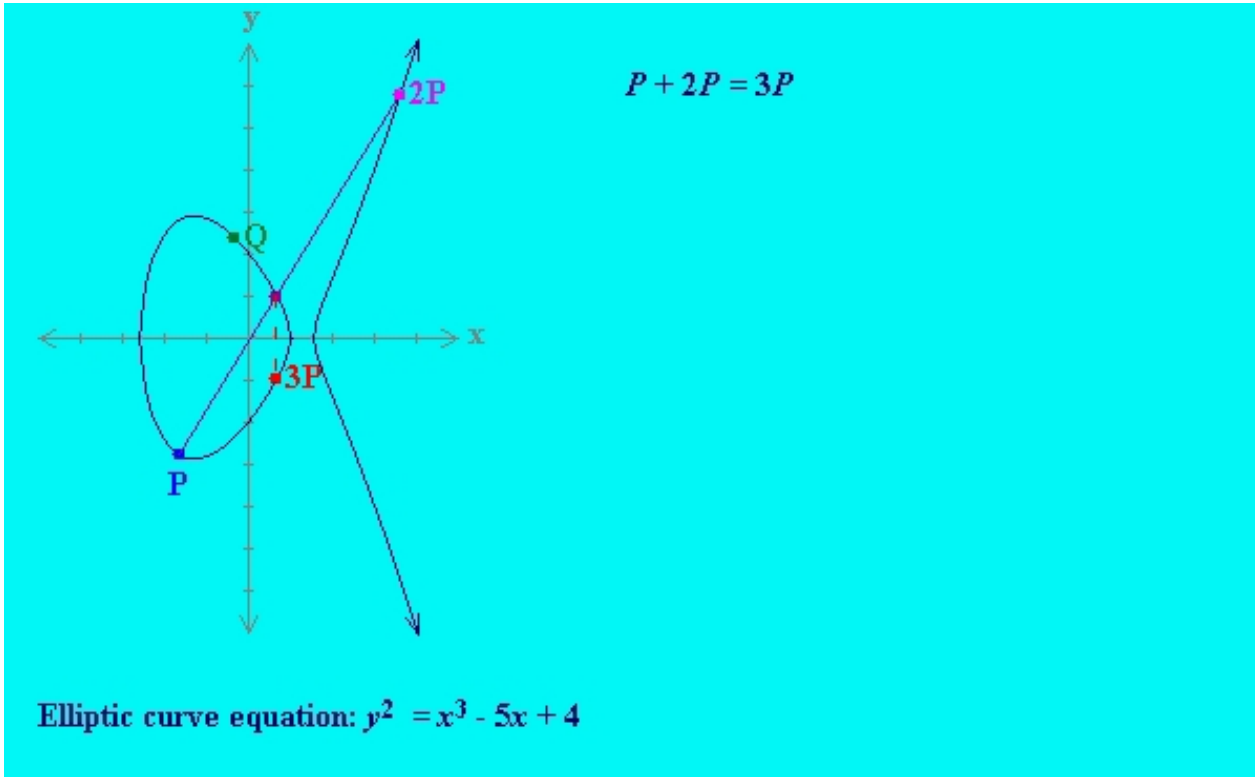
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



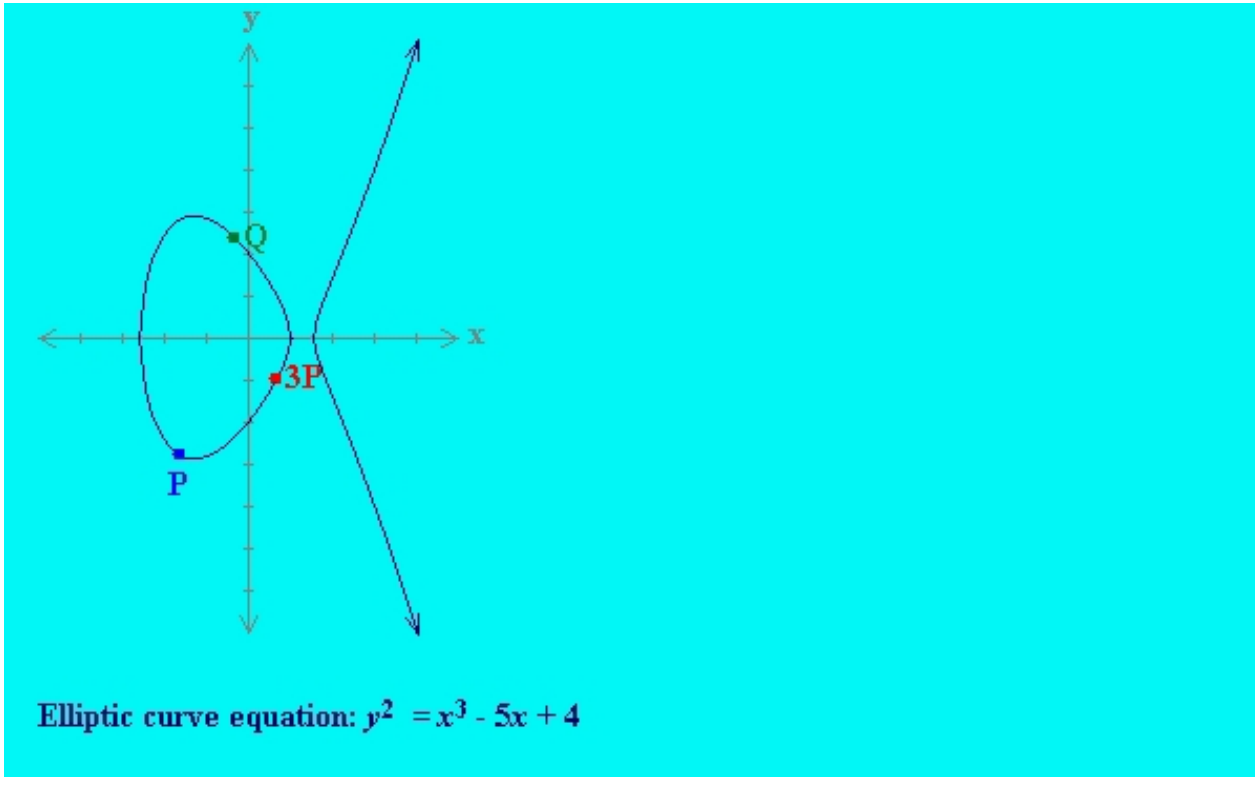
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



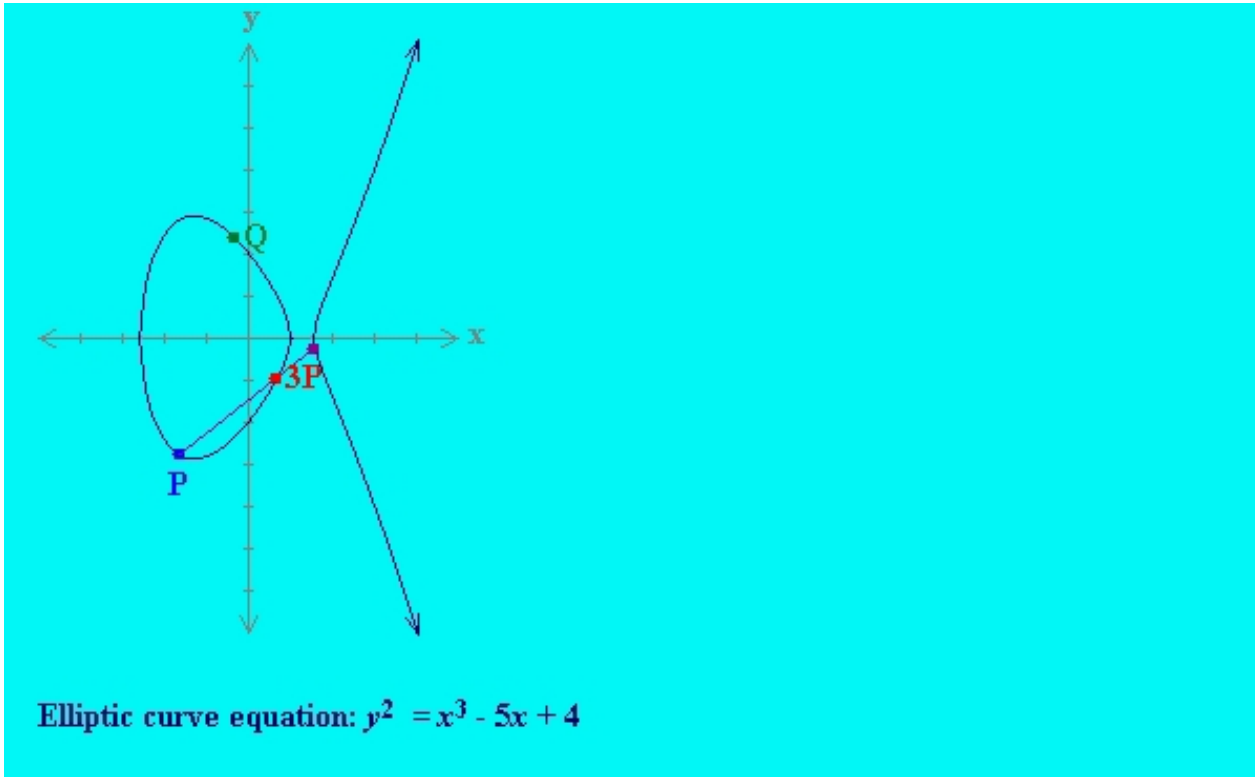
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



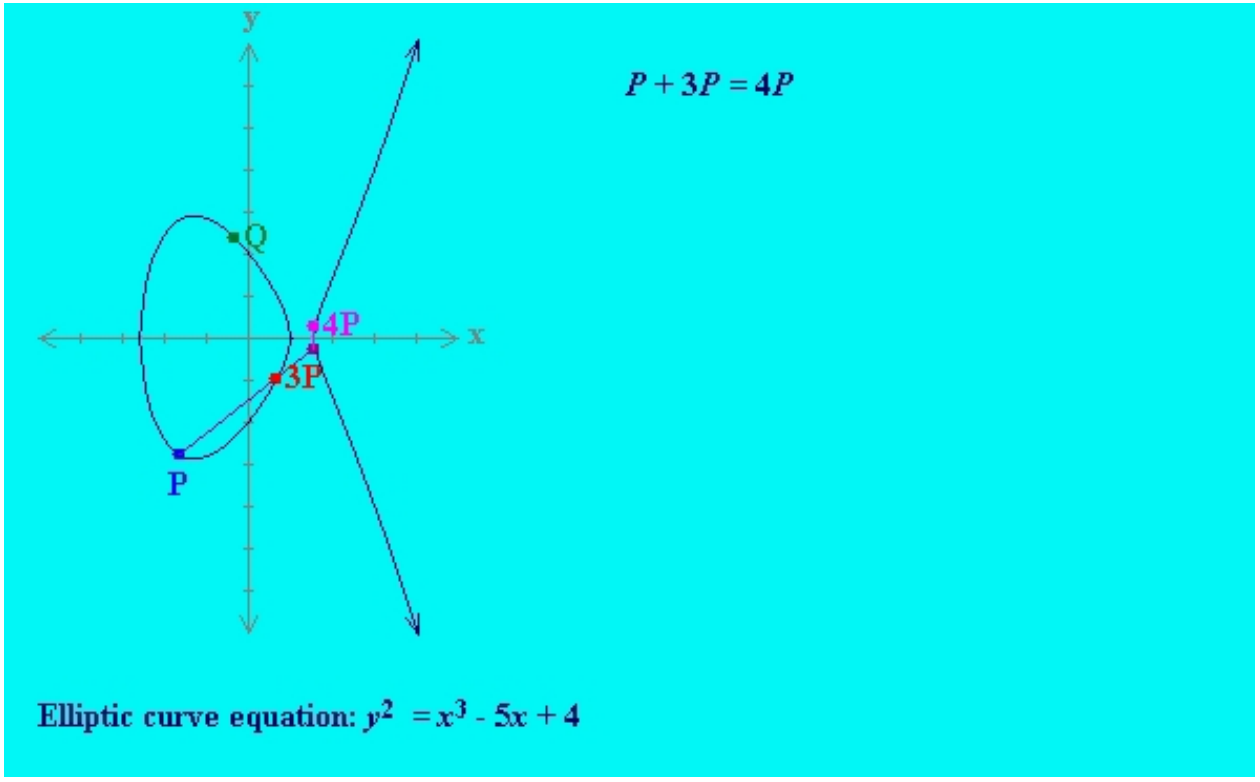
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



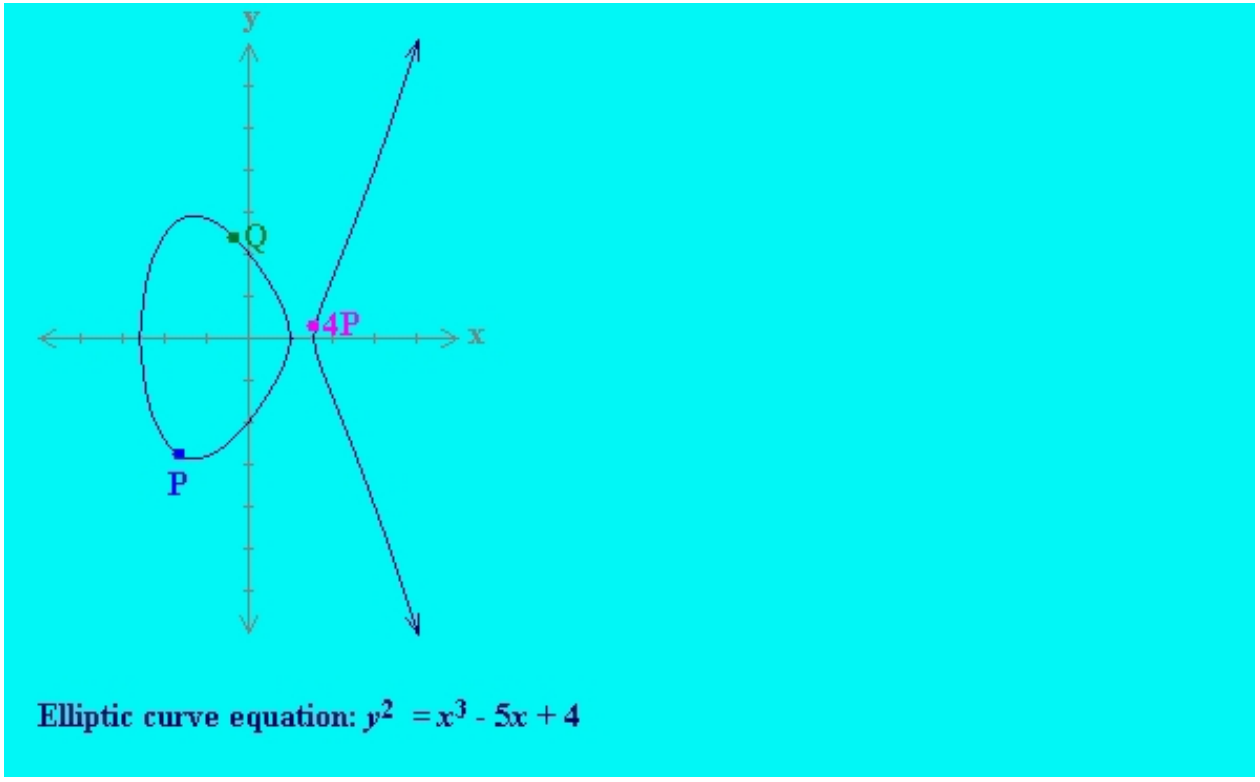
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



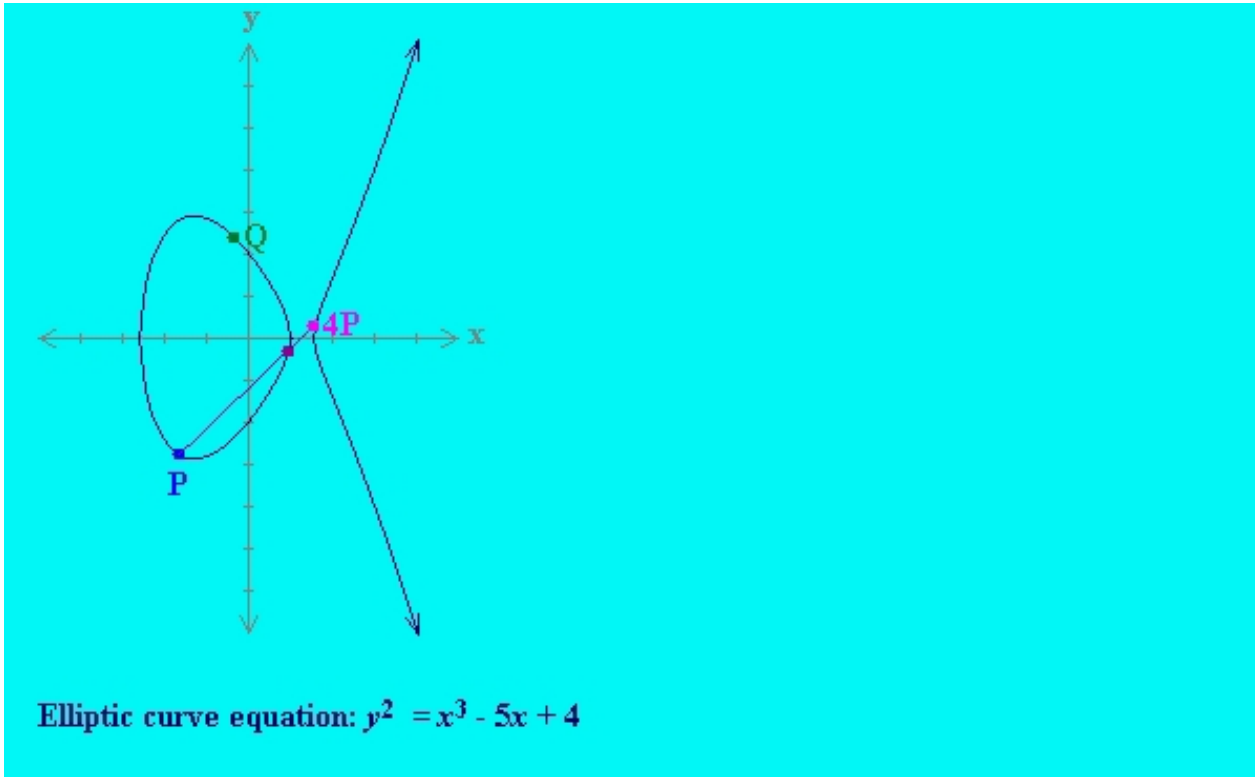
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



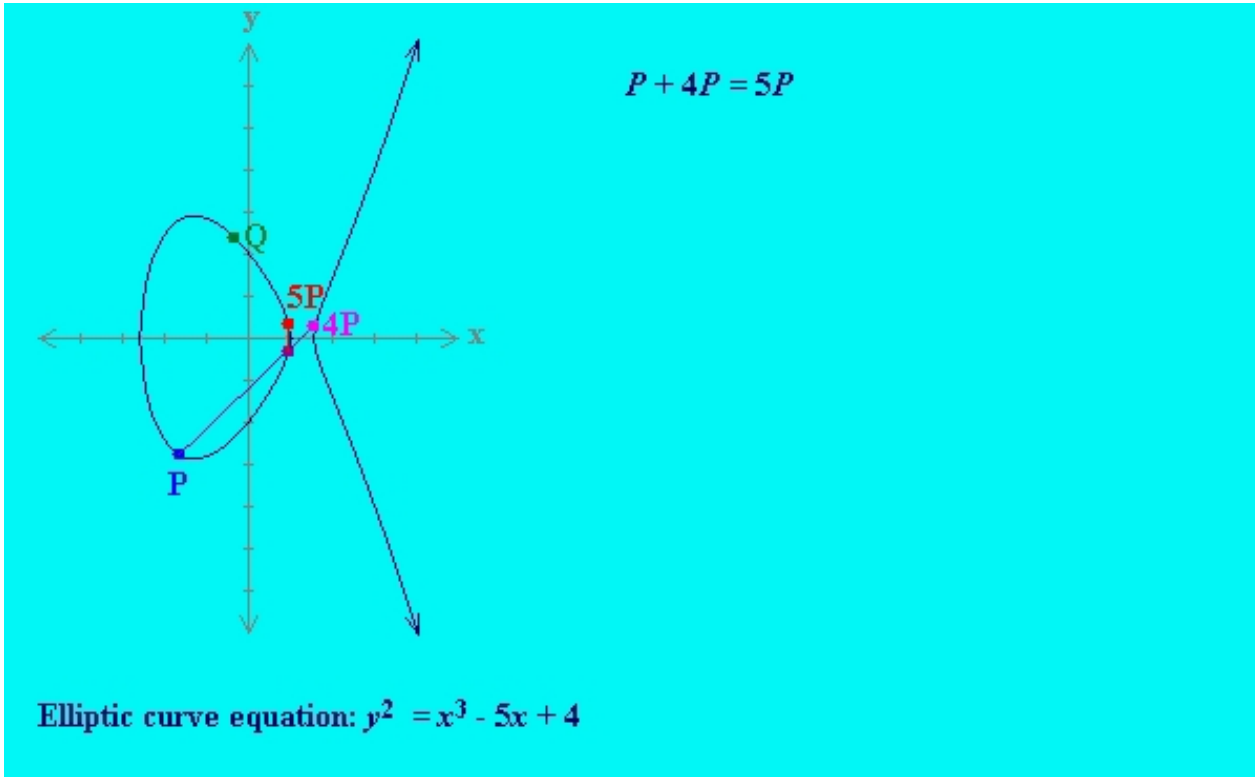
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



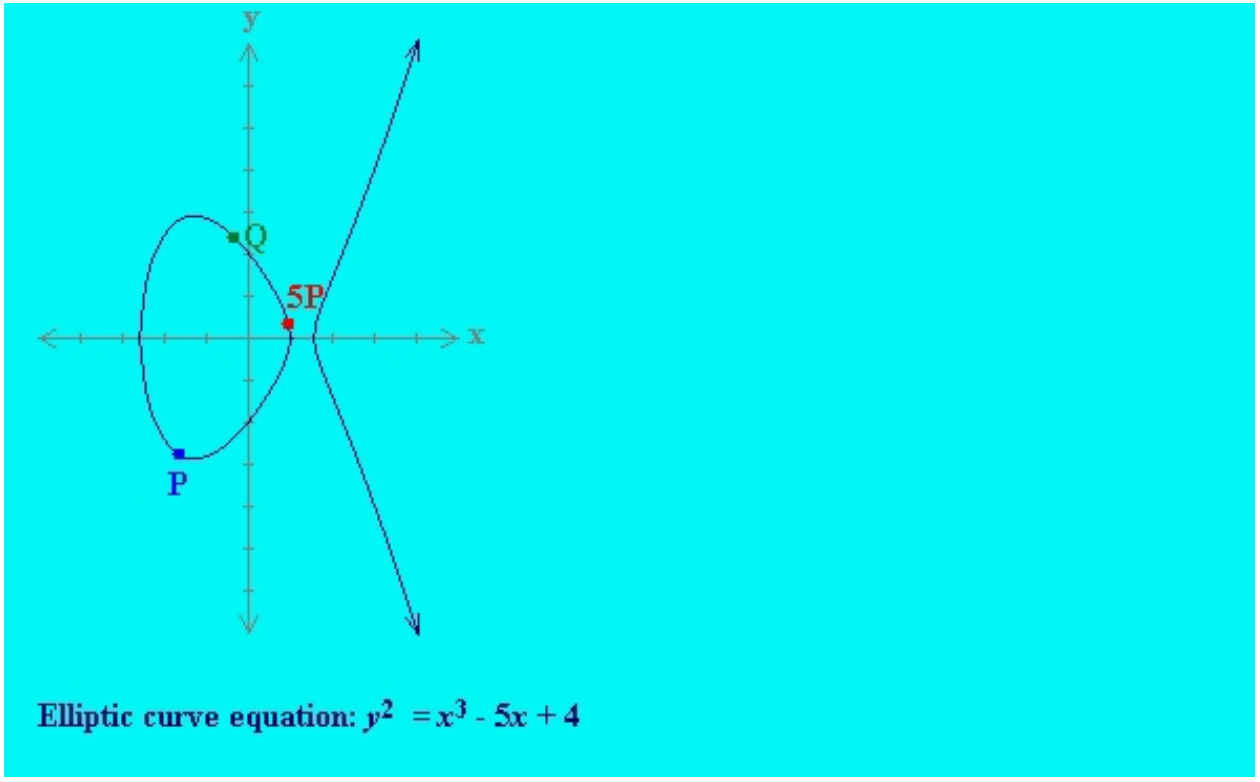
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



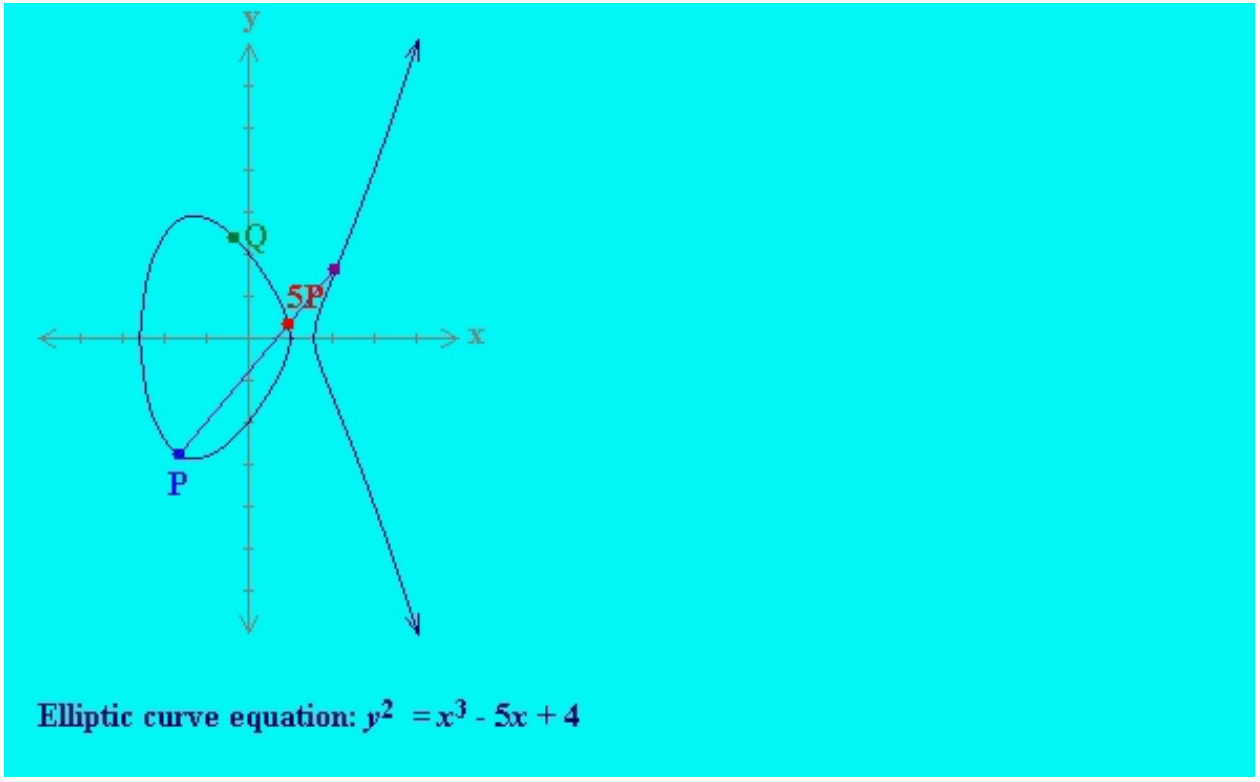
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



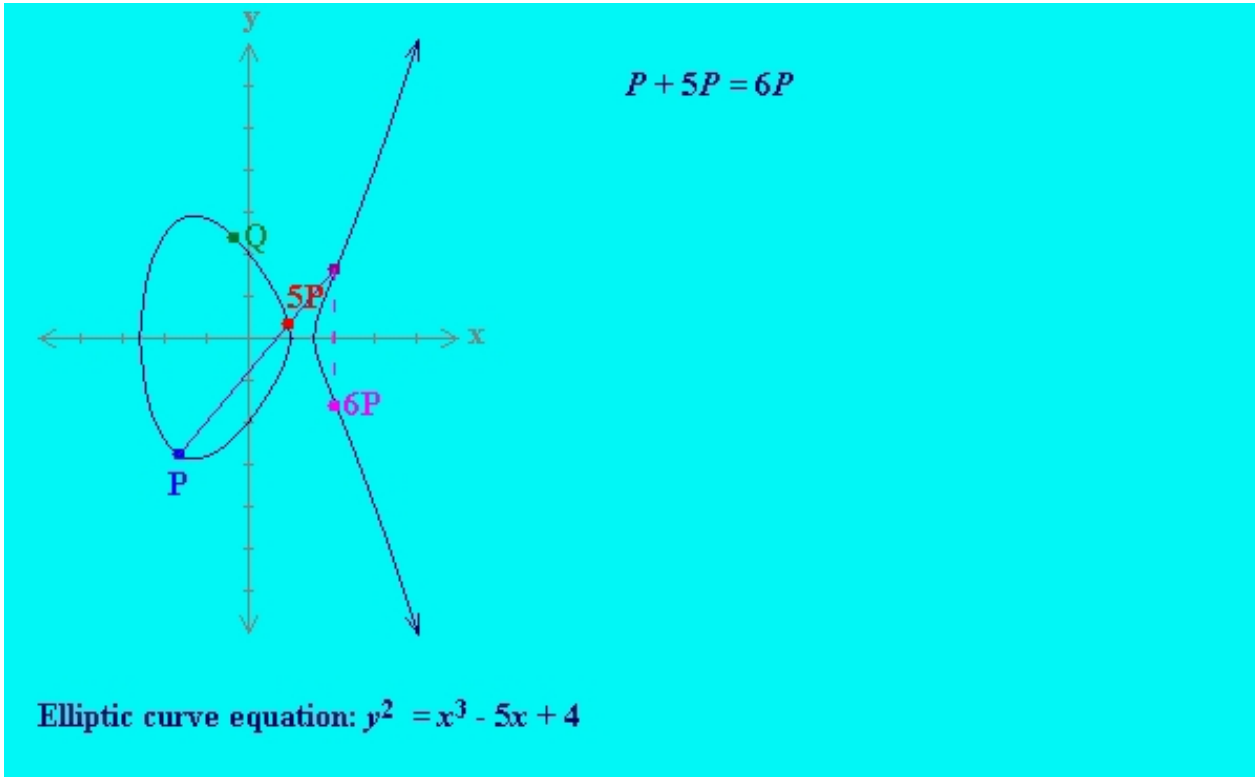
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



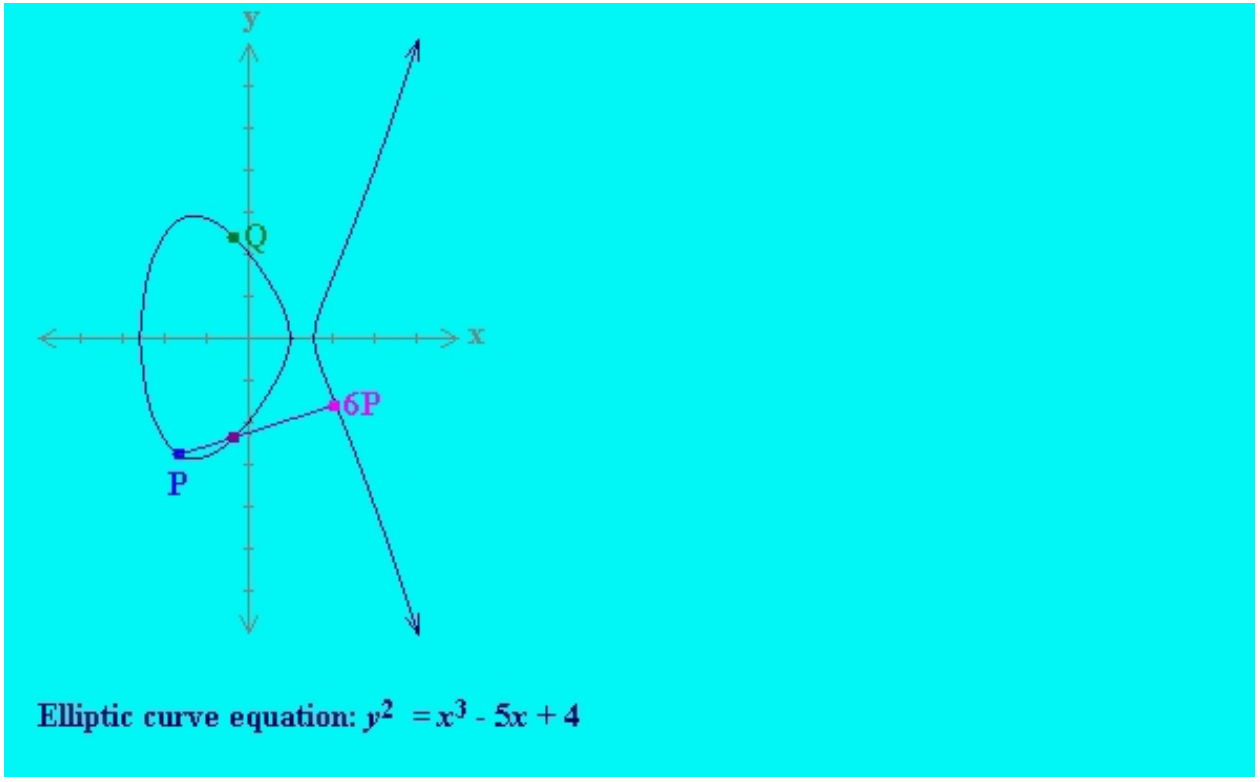
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



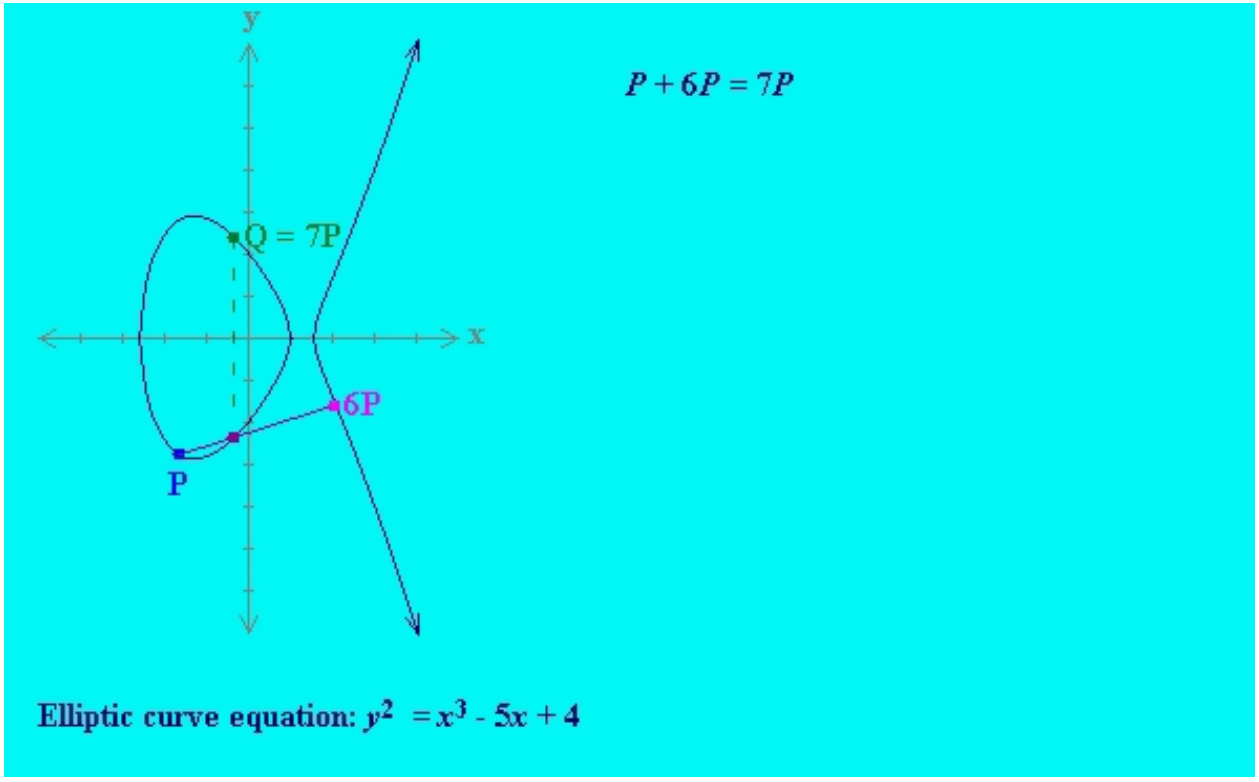
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



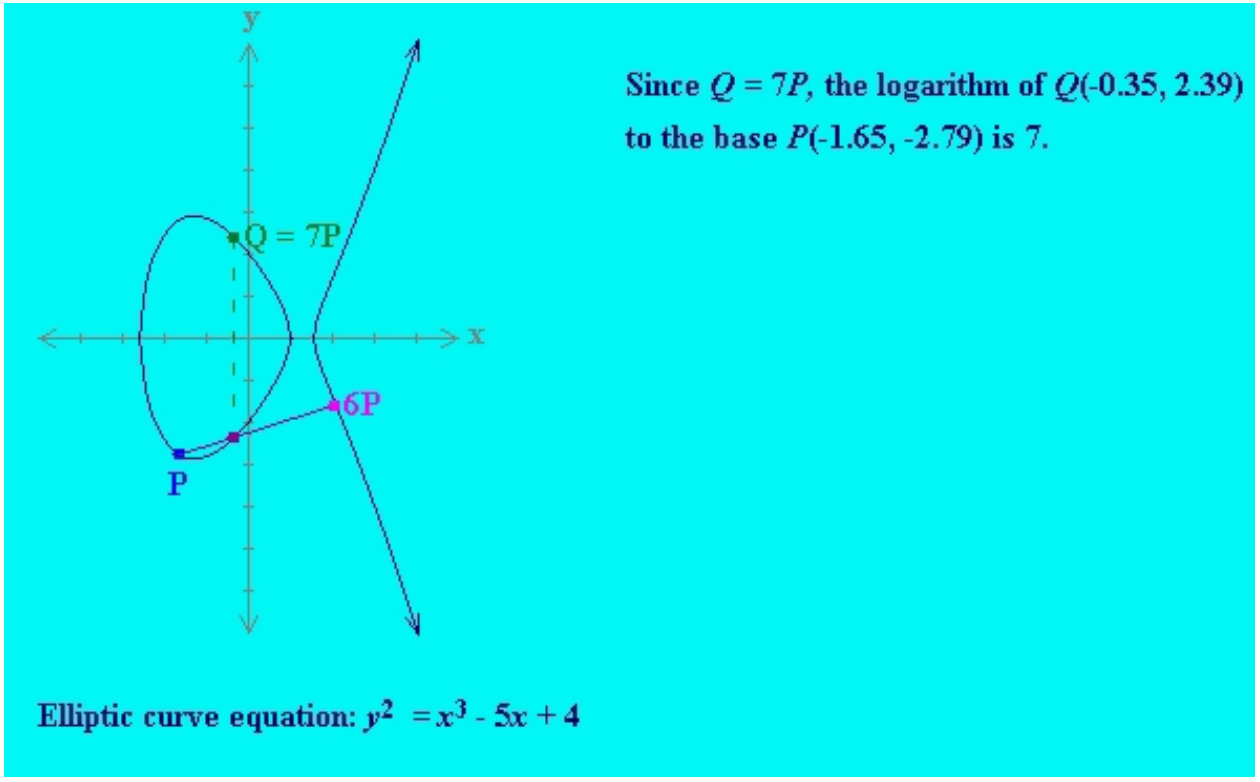
Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



Trivial method.

Given the curve $y^2 = x^3 - 5x + 4$ and points $P = (-1.65, 2.79)$ and $Q = (-0.35, 2.39)$, find the integer k such that $Q = kP$



ρ-Pollard (1/2)

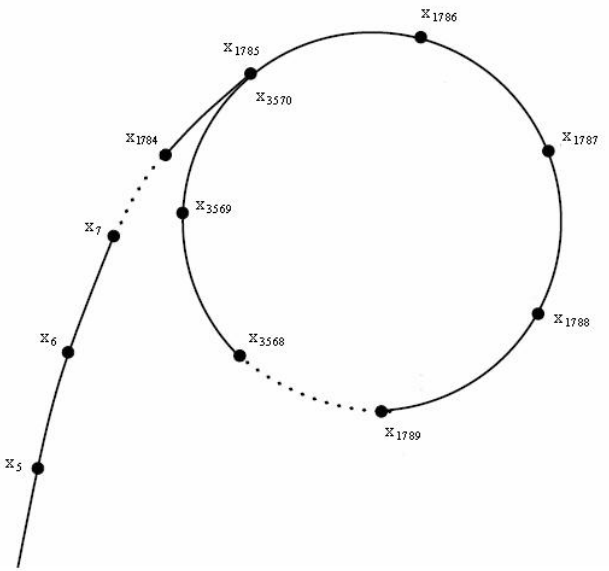
- All the problems of the Certicom list solved, until now, got a solution through ρ-Pollard method or some of its variants.
- This method iterates through the elements of the group until an element is discovered twice.
- The greek letter ρ in the name of the algorithm recalls the shape of the 'walk' of the iterations that closes over itself.

ρ-Pollard (1/2)

- All the problems of the Certicom list solved, until now, got a solution through ρ-Pollard method or some of its variants.
- This method iterates through the elements of the group until an element is discovered twice.
- The greek letter ρ in the name of the algorithm recalls the shape of the 'walk' of the iterations that closes over itself.

ρ-Pollard (1/2)

- All the problems of the Certicom list solved, until now, got a solution through ρ-Pollard method or some of its variants.
- This method iterates through the elements of the group until an element is discovered twice.
- The greek letter ρ in the name of the algorithm recalls the shape of the 'walk' of the iterations that closes over itself.



ρ-Pollard (2/2)

Given $P, Q \in E(\mathbb{F}_p)$, if we want to find the k such that $Q = kP$, the original version of ρ-Pollard works in this way:

- The group is partitioned into three disjoint subsets S_1, S_2 and S_3 of about the same size.
- Two integers a_0 and b_0 are choosen randomly with $0 \leq a_0, b_0 \leq n-1$, where n is the cardinality of $E(\mathbb{F}_p)$.
- Starting with the point $X_0 = a_0 P + b_0 Q$, a sequence of X_i is generated, defined for each $i \geq 1$, according following relation:

$$X_i = \begin{cases} P + X_{i-1} & \text{if} \quad X_{i-1} \in S_1 \\ 2X_{i-1} & \text{if} \quad X_{i-1} \in S_2 \\ Q + X_{i-1} & \text{if} \quad X_{i-1} \in S_3 \end{cases}$$

- If $X_i = X_j$ for some $i \neq j$ (*collision*), we can compute $a_i P + b_i Q = a_j P + b_j Q$ and then $(a_i - a_j)P = (b_j - b_i)Q$.

ρ-Pollard (2/2)

Given $P, Q \in E(\mathbb{F}_p)$, if we want to find the k such that $Q = kP$, the original version of ρ-Pollard works in this way:

- The group is partitioned into three disjoint subsets S_1,S_2 and S_3 of about the same size.
- Two integers a_0 and b_0 are choosen randomly with $0 \leq a_0, b_0 \leq n-1$, where n is the cardinality of $E(\mathbb{F}_p)$.
- Starting with the point $X_0 = a_0 P + b_0 Q$, a sequence of X_i is generated, defined for each $i \geq 1$, according following relation:

$$X_i = \begin{cases} P + X_{i-1} & \text{if} \quad X_{i-1} \in S_1 \\ 2X_{i-1} & \text{if} \quad X_{i-1} \in S_2 \\ Q + X_{i-1} & \text{if} \quad X_{i-1} \in S_3 \end{cases}$$

- If $X_i = X_j$ for some $i \neq j$ (*collision*), we can compute $a_i P + b_i Q = a_j P + b_j Q$ and then $(a_i - a_j)P = (b_j - b_i)Q$.

ρ-Pollard (2/2)

Given $P, Q \in E(\mathbb{F}_p)$, if we want to find the k such that $Q = kP$, the original version of ρ-Pollard works in this way:

- The group is partitioned into three disjoint subsets S_1,S_2 and S_3 of about the same size.
- Two integers a_0 and b_0 are choosen randomly with $0 \leq a_0, b_0 \leq n-1$, where n is the cardinality of $E(\mathbb{F}_p)$.
- Starting with the point $X_0 = a_0 P + b_0 Q$, a sequence of X_i is generated, defined for each $i \geq 1$, according following relation:

$$X_i = \begin{cases} P + X_{i-1} & \text{if} \quad X_{i-1} \in S_1 \\ 2X_{i-1} & \text{if} \quad X_{i-1} \in S_2 \\ Q + X_{i-1} & \text{if} \quad X_{i-1} \in S_3 \end{cases}$$

- If $X_i = X_j$ for some $i \neq j$ (*collision*), we can compute $a_i P + b_i Q = a_j P + b_j Q$ and then $(a_i - a_j)P = (b_j - b_i)Q$.

ρ-Pollard (2/2)

Given $P, Q \in E(\mathbb{F}_p)$, if we want to find the k such that $Q = kP$, the original version of ρ-Pollard works in this way:

- The group is partitioned into three disjoint subsets S_1,S_2 and S_3 of about the same size.
- Two integers a_0 and b_0 are choosen randomly with $0 \leq a_0, b_0 \leq n-1$, where n is the cardinality of $E(\mathbb{F}_p)$.
- Starting with the point $X_0 = a_0 P + b_0 Q$, a sequence of X_i is generated, defined for each $i \geq 1$, according following relation:

$$X_i = \begin{cases} P + X_{i-1} & \text{if} \quad X_{i-1} \in S_1 \\ 2X_{i-1} & \text{if} \quad X_{i-1} \in S_2 \\ Q + X_{i-1} & \text{if} \quad X_{i-1} \in S_3 \end{cases}$$

- If $X_i = X_j$ for some $i \neq j$ (*collision*), we can compute $a_i P + b_i Q = a_j P + b_j Q$ and then $(a_i - a_j)P = (b_j - b_i)Q$.

ρ-Pollard (2/2)

Given $P, Q \in E(\mathbb{F}_p)$, if we want to find the k such that $Q = kP$, the original version of ρ-Pollard works in this way:

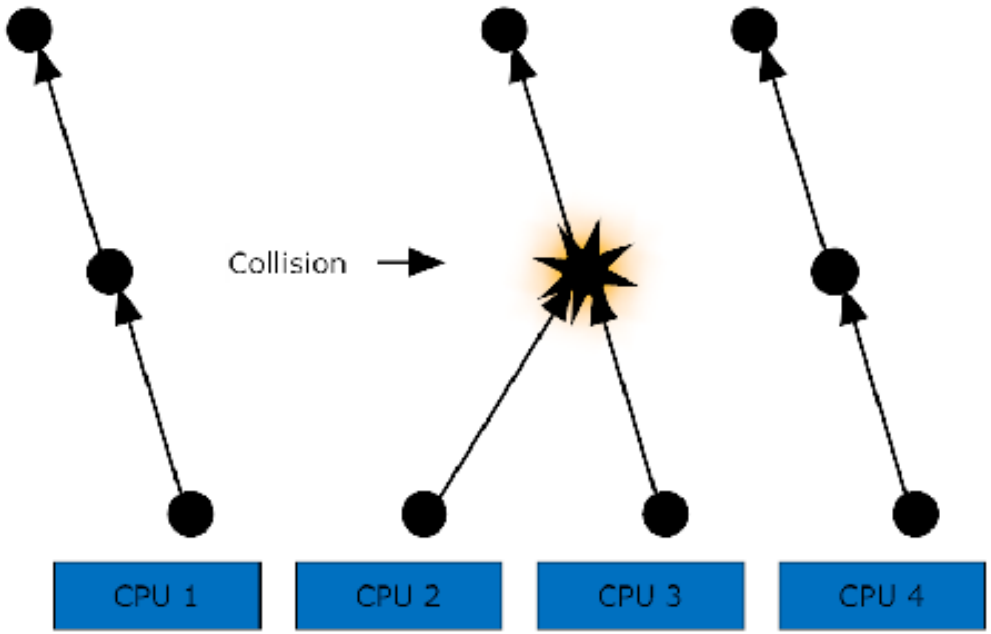
- The group is partitioned into three disjoint subsets S_1,S_2 and S_3 of about the same size.
- Two integers a_0 and b_0 are choosen randomly with $0 \leq a_0, b_0 \leq n-1$, where n is the cardinality of $E(\mathbb{F}_p)$.
- Starting with the point $X_0 = a_0 P + b_0 Q$, a sequence of X_i is generated, defined for each $i \geq 1$, according following relation:

$$X_i = \begin{cases} P + X_{i-1} & \text{if} \quad X_{i-1} \in S_1 \\ 2X_{i-1} & \text{if} \quad X_{i-1} \in S_2 \\ Q + X_{i-1} & \text{if} \quad X_{i-1} \in S_3 \end{cases}$$

- If $X_i = X_j$ for some $i \neq j$ (*collision*), we can compute $a_i P + b_i Q = a_j P + b_j Q$ and then $(a_i - a_j)P = (b_j - b_i)Q$.

λ-Pollard

It has been shown that ρ-Pollard algorithm can be efficiently parallelized over an arbitrary number of processors, reserving one of them for collisions search. Such method is called λ-Pollard where the greek letter λ recalls to mind the collision between two walks generated by two different processors.



Improving performances of Pollard's algorithms.

How can we improve performances of Pollard's algorithms ?

- Optimizing the iterating function.
- Using better all hardware resources available.

Improving performances of Pollard's algorithms.

How can we improve performances of Pollard's algorithms ?

- Optimizing the iterating function.
- Using better all hardware resources available.

Improving the iterating function.

- The complexity of the Pollard's algorithm is proportional to the square root of the order of the group where you want to solve the discrete logarithm problem.
- Such estimate is done under the hypotesis that the points generated through the method are generated randomly, so that they build a 'random walk' inside the group.
- Sadly, the original function proposed by Pollard has a behaviour that is really far from being random.
- It has been shown heuristically that increasing the number of partition S_i into which is split the group, cause a good gain of performances.

Improving the iterating function.

- The complexity of the Pollard's algorithm is proportional to the square root of the order of the group where you want to solve the discrete logarithm problem.
- Such estimate is done under the hypotesis that the points generated through the method are generated randomly, so that they build a 'random walk' inside the group.
- Sadly, the original function proposed by Pollard has a behaviour that is really far from being random.
- It has been shown heuristically that increasing the number of partition S_i into which is split the group, cause a good gain of performances.

Improving the iterating function.

- The complexity of the Pollard's algorithm is proportional to the square root of the order of the group where you want to solve the discrete logarithm problem.
- Such estimate is done under the hypotesis that the points generated through the method are generated randomly, so that they build a 'random walk' inside the group.
- Sadly, the original function proposed by Pollard has a behaviour that is really far from being random.
- It has been shown heuristically that increasing the number of partition S_i into which is split the group, cause a good gain of performances.

Improving the iterating function.

- The complexity of the Pollard's algorithm is proportional to the square root of the order of the group where you want to solve the discrete logarithm problem.
- Such estimate is done under the hypotesis that the points generated through the method are generated randomly, so that they build a 'random walk' inside the group.
- Sadly, the original function proposed by Pollard has a behaviour that is really far from being random.
- It has been shown heuristically that increasing the number of partition S_i into which is split the group, cause a good gain of performances.

How to improve the use of the hardware ?

- Taking advantage of computational power of other devices together with main cpu.
- Typical example: graphic processing unit of video cards in 3D graphics.
- Recently NVidia released some libraries that allow programmers to use video cards like general purpose processors.

How to improve the use of the hardware ?

- Taking advantage of computational power of other devices together with main cpu.
- Typical example: graphic processing unit of video cards in 3D graphics.
- Recently NVidia released some libraries that allow programmers to use video cards like general purpose processors.

How to improve the use of the hardware ?

- Taking advantage of computational power of other devices together with main cpu.
- Typical example: graphic processing unit of video cards in 3D graphics.
- Recently NVidia released some libraries that allow programmers to use video cards like general purpose processors.

CUDA

- Acronym of **Compute Unified Device Architecture**.
- Video card is viewed like a multiprocessor architecture (since, usually, there is a large number of 'cores' in the graphic processor).
- Library written in C language for device programming, but there are wrappers of third parts for other languages like Python, Fortran and Java.

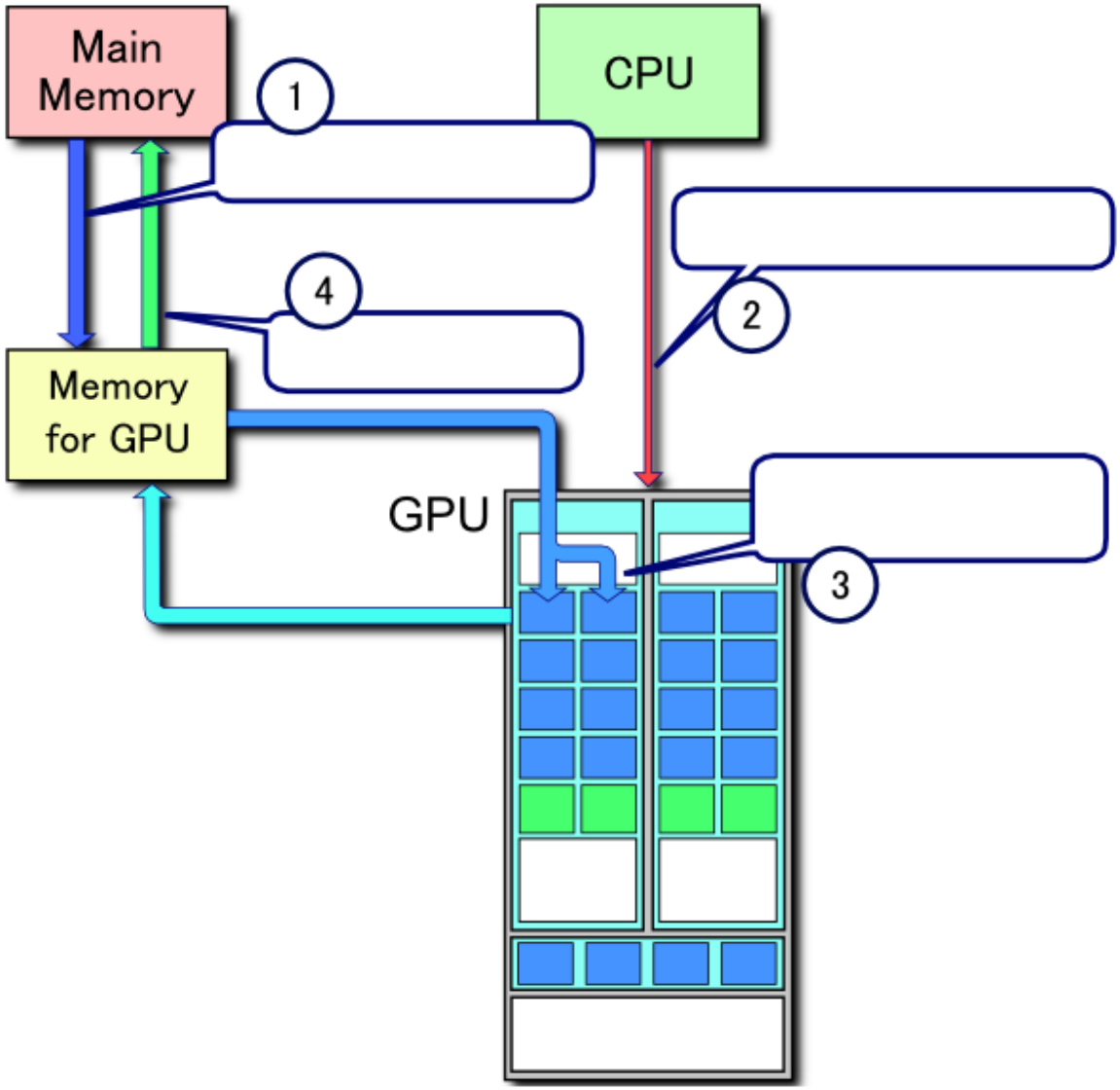
CUDA

- Acronym of **Compute Unified Device Architecture**.
- Video card is viewed like a multiprocessor architecture (since, usually, there is a large number of ‘cores’ in the graphic processor).
- Library written in C language for device programming, but there are wrappers of third parts for other languages like Python, Fortran and Java.

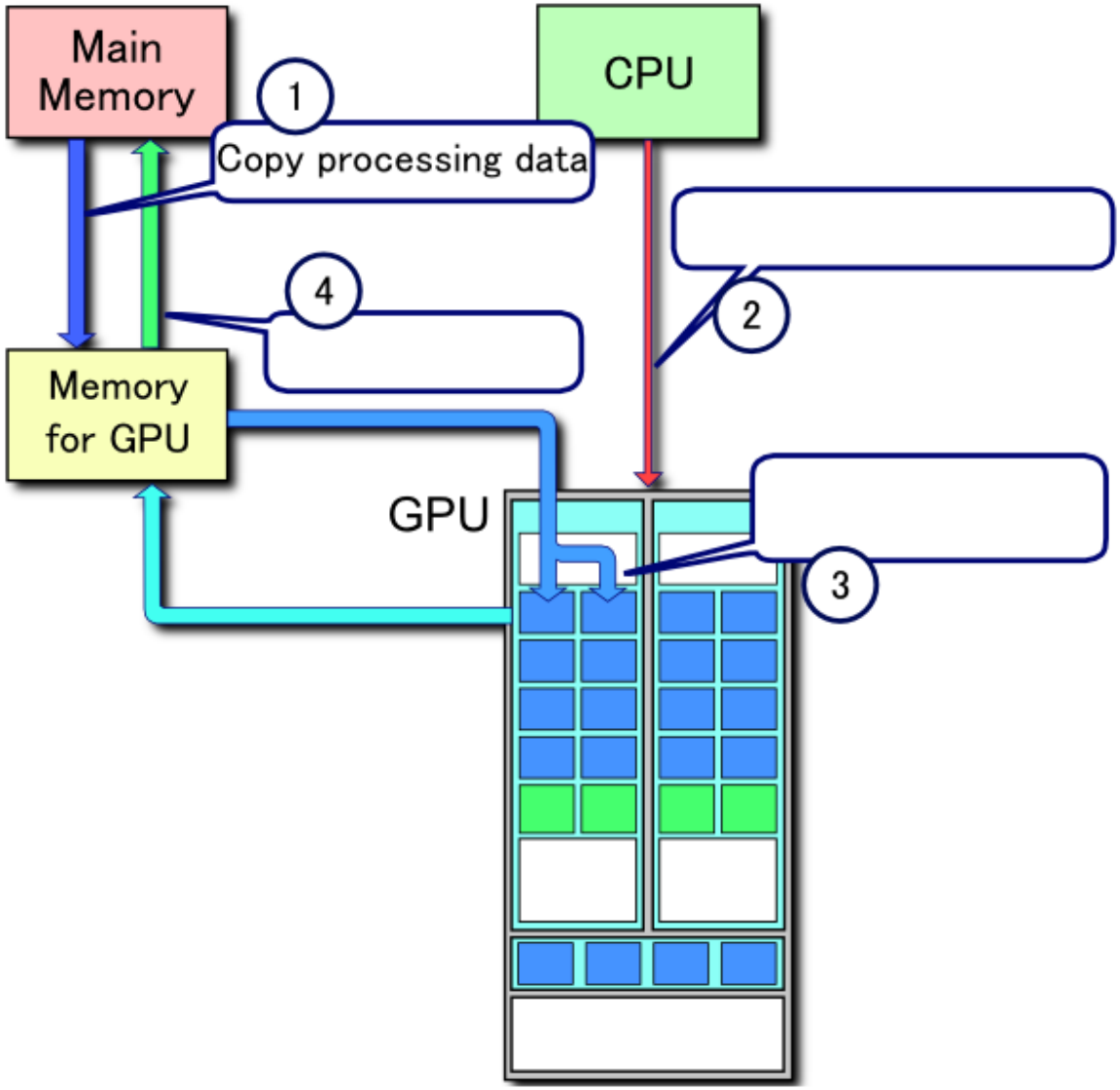
CUDA

- Acronym of **Compute Unified Device Architecture**.
- Video card is viewed like a multiprocessor architecture (since, usually, there is a large number of ‘cores’ in the graphic processor).
- Library written in C language for device programming, but there are wrappers of third parts for other languages like Python, Fortran and Java.

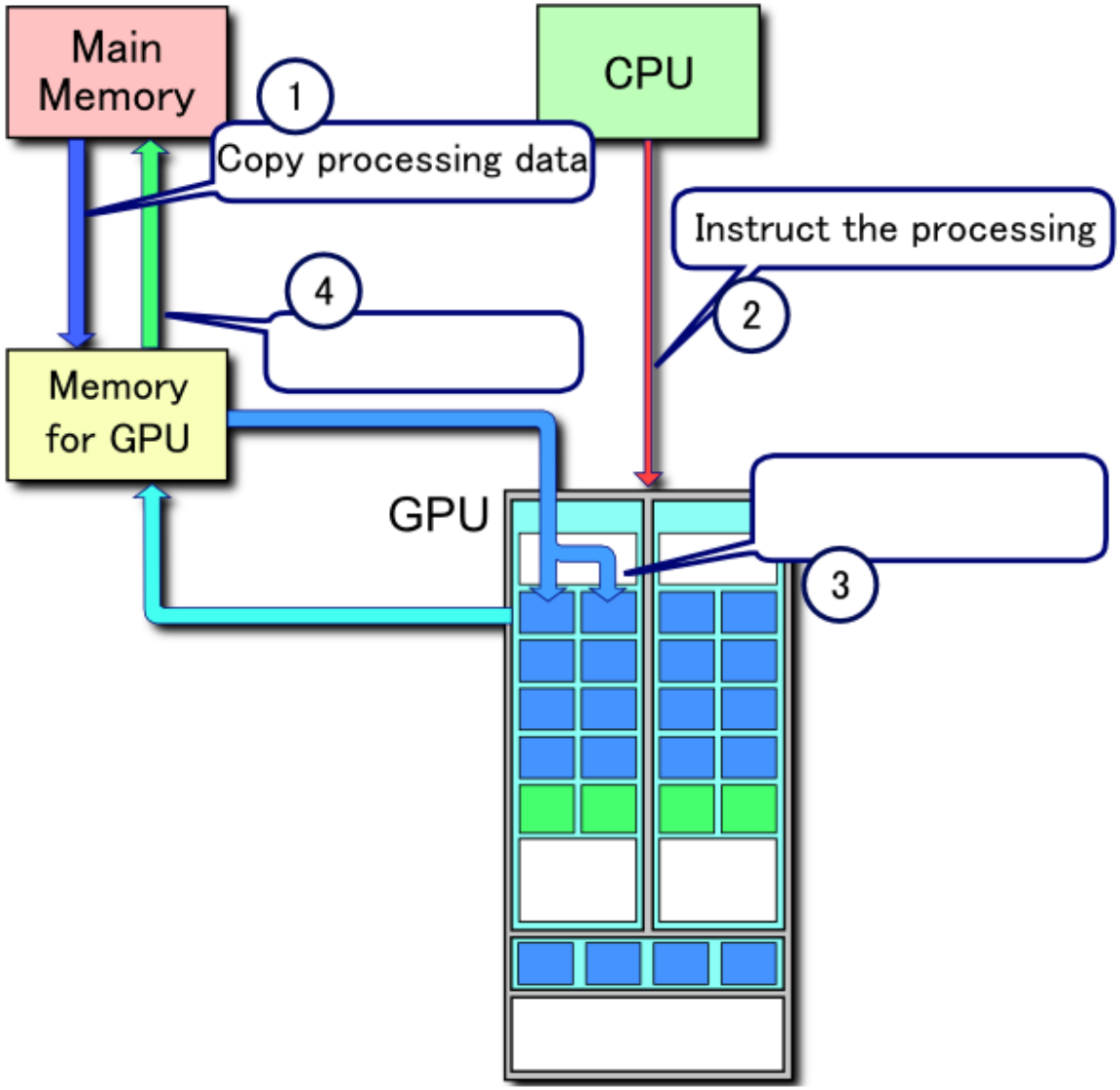
Code execution in CUDA.



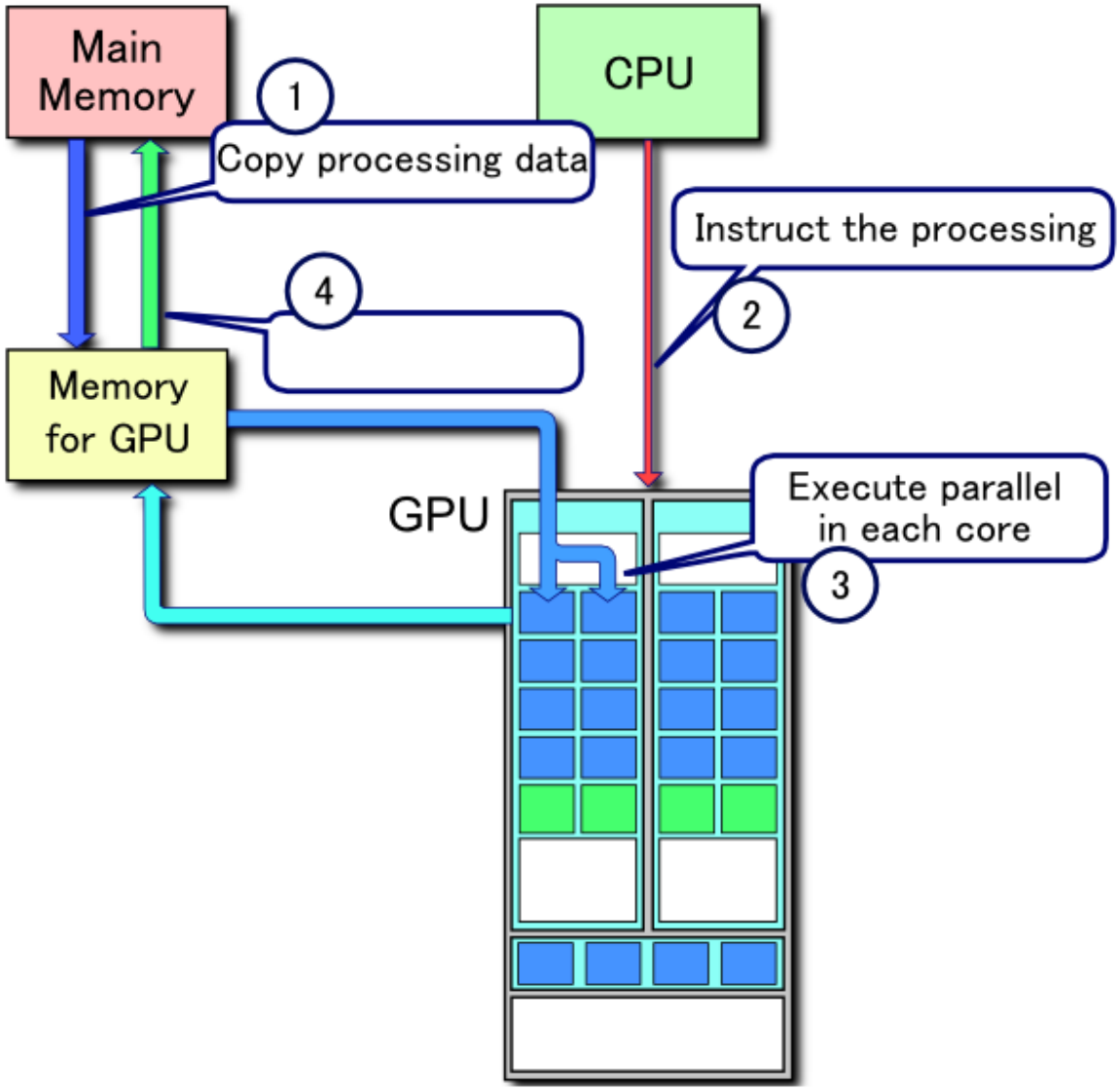
Code execution in CUDA.



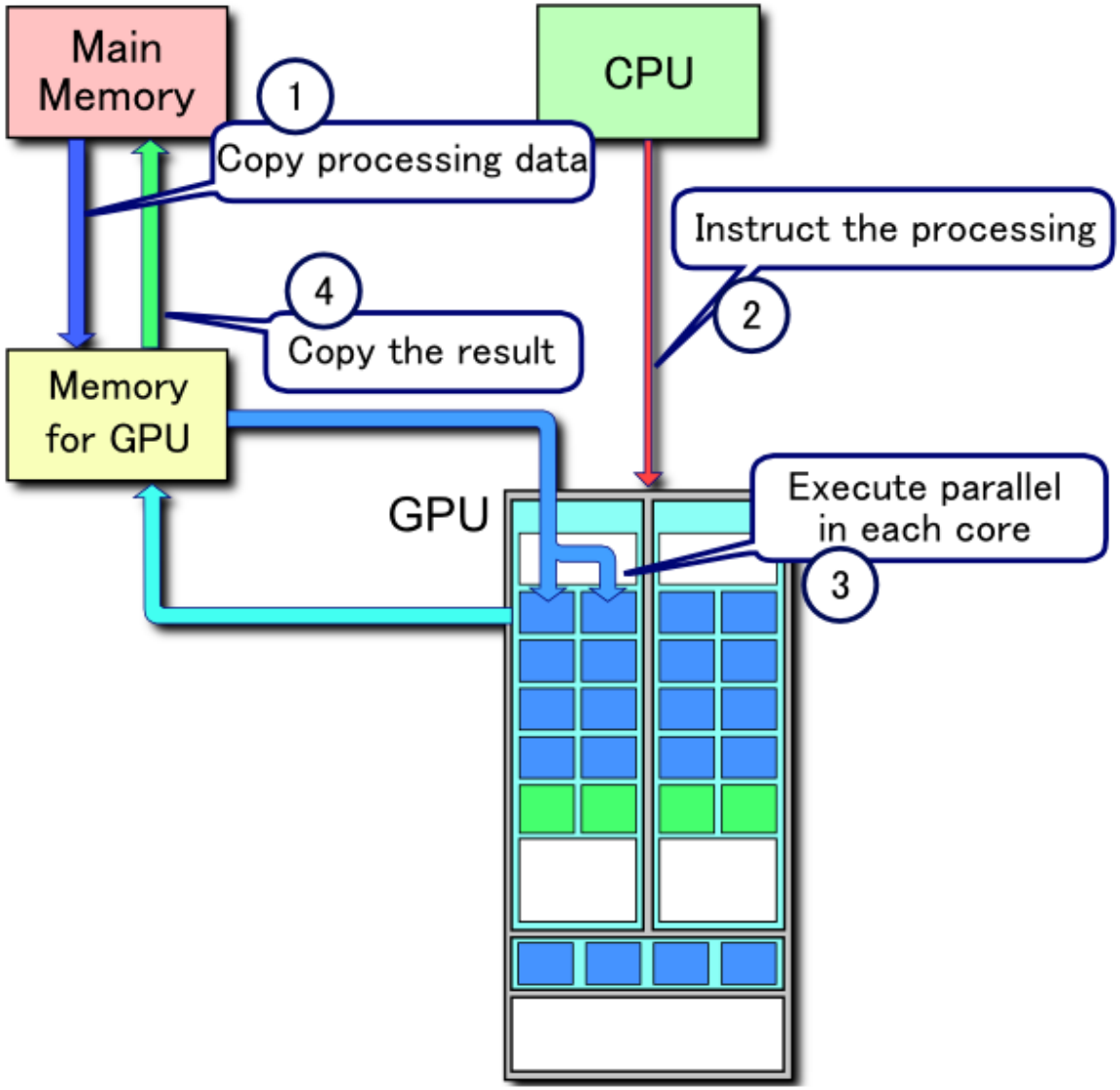
Code execution in CUDA.



Code execution in CUDA.



Code execution in CUDA.



CUDA limitations.

- Arithmetic divisions are really computationally expensive and should be avoided.
- Memory should be carefully handled to avoid latencies when loading data from it and writing data to it.
- Since graphics processor is a SIMD (Single Instruction Multiple Data) architecture, every branch dependent from data in the code execution could cause the so called 'divergent threads', furtherly reducing the speed of the algorithm.

CUDA limitations.

- Arithmetic divisions are really computationally expensive and should be avoided.
- Memory should be carefully handled to avoid latencies when loading data from it and writing data to it.
- Since graphics processor is a SIMD (Single Instruction Multiple Data) architecture, every branch dependent from data in the code execution could cause the so called 'divergent threads', furtherly reducing the speed of the algorithm.

CUDA limitations.

- Arithmetic divisions are really computationally expensive and should be avoided.
- Memory should be carefully handled to avoid latencies when loading data from it and writing data to it.
- Since graphics processor is a SIMD (Single Instruction Multiple Data) architecture, every branch dependent from data in the code execution could cause the so called ‘divergent threads’, furtherly reducing the speed of the algorithm.

Modular arithmetic with CUDA (1/7)

- In a code for ECDLP resolution, of course we're handling multi-words integers all of them belonging to the field $\mathbb{F}_p$ on which the elliptic curve is defined.
- Modular addition can be efficiently performed first adding two operands a and b. Then subtracting the modulus p to the sum previously computed. If this subtraction cause a positive result, then this is the sum reduced modulus p, otherwise the right sum modulus p is the one previously computed.
- In a similar way can be done the modular subtraction.

Modular arithmetic with CUDA (1/7)

- In a code for ECDLP resolution, of course we're handling multi-words integers all of them belonging to the field $\mathbb{F}_p$ on which the elliptic curve is defined.
- Modular addition can be efficiently performed first adding two operands a and b. Then subtracting the modulus p to the sum previously computed. If this subtraction cause a positive result, then this is the sum reduced modulus p, otherwise the right sum modulus p is the one previously computed.
- In a similar way can be done the modular subtraction.

Modular arithmetic with CUDA (1/7)

- In a code for ECDLP resolution, of course we're handling multi-words integers all of them belonging to the field $\mathbb{F}_p$ on which the elliptic curve is defined.
- Modular addition can be efficiently performed first adding two operands a and b. Then subtracting the modulus p to the sum previously computed. If this subtraction cause a positive result, then this is the sum reduced modulus p, otherwise the right sum modulus p is the one previously computed.
- In a similar way can be done the modular subtraction.

Modular arithmetic with CUDA (2/7)

- Modular multiplication is realized through the so called Montgomery product.
- If p is the modulus, we call k the integer such that $2^{k-1} \leq p < 2^k$ and r is 2^k. Given an integer $a < p$, we define *Montgomery representation* (or *p-residue*) with respect to r as

$$\overline{a} \equiv a \cdot r \pmod{p}$$

- Sum and difference of the Mongomery representations of two integers is Montgomery representation of their sum or difference.

Modular arithmetic with CUDA (2/7)

- Modular multiplication is realized through the so called Montgomery product.
- If p is the modulus, we call k the integer such that $2^{k-1} \leq p < 2^k$ and r is 2^k. Given an integer $a < p$, we define *Montgomery representation* (or *p-residue*) with respect to r as

$$\overline{a} \equiv a \cdot r(\mod p)$$

- Sum and difference of the Mongomery representations of two integers is Montgomery representation of their sum or difference.

Modular arithmetic with CUDA (2/7)

- Modular multiplication is realized through the so called Montgomery product.
- If p is the modulus, we call k the integer such that $2^{k-1} \leq p < 2^k$ and r is 2^k. Given an integer $a < p$, we define *Montgomery representation* (or *p-residue*) with respect to r as

$$\overline{a} \equiv a \cdot r(\mod p)$$

- Sum and difference of the Mongomery representations of two integers is Montgomery representation of their sum or difference.

Modular arithmetic with CUDA (3/7)

- Given two numbers a, b in their Montgomery representations ($\overline{a}, \overline{b}$ respectively) the Montgomery product is defined as:

$$\overline{u} \equiv \overline{a} \cdot \overline{b} \cdot r^{-1} (\mod p)$$

  where r^{-1} is the multiplicative inverse of r modulo p.

- The result of Montgomery product $\overline{u}$ is the p-residue of the product $u = a \cdot b (\mod p)$ since

$$\begin{aligned} \overline{u} &\equiv \overline{a} \cdot \overline{b} \cdot r^{-1} (\mod p) \\ &= (a \cdot r) \cdot (b \cdot r) \cdot r^{-1} (\mod p) \\ &= (a \cdot b) \cdot r (\mod p). \end{aligned}$$

Modular arithmetic with CUDA (3/7)

- Given two numbers a, b in their Montgomery representations ($\overline{a}, \overline{b}$ respectively) the Montgomery product is defined as:

$$\overline{u} \equiv \overline{a} \cdot \overline{b} \cdot r^{-1} (\mod p)$$

  where r^{-1} is the multiplicative inverse of r modulo p.
- The result of Montgomery product $\overline{u}$ is the p-residue of the product $u = a \cdot b (\mod p)$ since

$$\begin{aligned} \overline{u} &\equiv \overline{a} \cdot \overline{b} \cdot r^{-1} (\mod p) \\ &= (a \cdot r) \cdot (b \cdot r) \cdot r^{-1} (\mod p) \\ &= (a \cdot b) \cdot r (\mod p). \end{aligned}$$

Modular arithmetic with CUDA (4/7)

To describe Montgomery reduction algorithm, we need also the quantity p', that satisfies the property

$$r \cdot r^{-1} - p \cdot p' = 1$$

Both integers r^{-1} and p' can be easily computed through the extended Euclidean algorithm. And since in our algorithm the modulus is fixed, we can precompute these values and store them for all multiplications.

Modular arithmetic with CUDA (5/7)

- Given the integers $\overline{a}, \overline{b}$ the Montgomery product is computed by this algorithm:
  MonPro($\overline{a}, \overline{b}$)
  1. $t = \overline{a} \cdot \overline{b}$
  2. $m \equiv t \cdot p'(\mod r)$
  3. $\overline{u} = (t + m \cdot r)/r$
  4. if $\overline{u} \geq p$ then return $\overline{u} - p$ else return $\overline{u}$.
- The main feature of this product is that the operations involved are multiplications modulo r and division by r that can be efficiently implemented using bitwise operations since r is a power of 2.

Modular arithmetic with CUDA (5/7)

- Given the integers $\overline{a}, \overline{b}$ the Montgomery product is computed by this algorithm:
  MonPro($\overline{a}, \overline{b}$)
  1. $t = \overline{a} \cdot \overline{b}$
  2. $m \equiv t \cdot p'(\mod r)$
  3. $\overline{u} = (t + m \cdot r)/r$
  4. if $\overline{u} \geq p$ then return $\overline{u} - p$ else return $\overline{u}$.
- The main feature of this product is that the operations involved are multiplications modulo r and division by r that can be efficiently implemented using bitwise operations since r is a power of 2.

Modular arithmetic with CUDA (6/7)

- If p is odd, Montgomery product algorithm can be used to compute (normal) product $u \equiv a \cdot b(\mod p)$:
  ModMul(a, b)
  1. Compute p' using extended Euclidean algorithm
  2. $\overline{a} \equiv a \cdot r(\mod p)$
  3. $\overline{b} \equiv b \cdot r(\mod p)$
  4. $\overline{u} =$MonPro$(\overline{a}, \overline{b})$
  5. $u =$MonPro$(\overline{u}, 1)$
  6. return u .

Modular arithmetic with CUDA (7/7)

- A better algorithm is obtained observing that MonPro($\overline{a}, b$)=$(a \cdot r) \cdot b \cdot r^{-1} (\mod p) = a \cdot b (\mod p)$
- Thus we can modify the algorithm above: **ModMul(a, b)**
  1. Compute p' using extended Euclidean algorithm, and $r^2 (\mod p)$
  2. $\overline{a}$ =MonPro(a, r^2)
  3. u =MonPro($\overline{a}, b$)

Modular arithmetic with CUDA (7/7)

- A better algorithm is obtained observing that MonPro($\overline{a}, b$)=$(a \cdot r) \cdot b \cdot r^{-1} (\mod p) = a \cdot b (\mod p)$
- Thus we can modify the algorithm above: **ModMul(a, b)**
  1. Compute p' using extended Euclidean algorithm, and $r^2 (\mod p)$
  2. $\overline{a}$ =MonPro(a, r^2)
  3. u =MonPro($\overline{a}, b$)

ρ-Pollard with CUDA

- Further optimizations of the algorithm include:
  - Group has been split into 64 disjoint subsets (a 'good' value according to some recent heuristic results by E.Teske)
  - Jacobian coordinate system to memorize points given in input to the iterating function of the Pollard's method.
  - Affine coordinate system to memorize points that are added to input points in the iterating function.
  - These different coordinate systems allows the use of mixed affine-jacobian addition formulas that have better performances than pure affine addition formulas, while lower storage needing of affine coordinate allows to store thepoints of the iterating function in the memory region of the video card reserved to constants (a faster memory region in which all thread can access together without the introduction of any latency in the execution of code).

ρ-Pollard with CUDA

- Further optimizations of the algorithm include:
  - Group has been split into 64 disjoint subsets (a 'good' value according to some recent heuristic results by E.Teske)
  - Jacobian coordinate system to memorize points given in input to the iterating function of the Pollard's method.
  - Affine coordinate system to memorize points that are added to input points in the iterating function.
  - These different coordinate systems allows the use of mixed affine-jacobian addition formulas that have better performances than pure affine addition formulas, while lower storage needing of affine coordinate allows to store thepoints of the iterating function in the memory region of the video card reserved to constants (a faster memory region in which all thread can access together without the introduction of any latency in the execution of code).

ρ-Pollard with CUDA

- Further optimizations of the algorithm include:
  - Group has been split into 64 disjoint subsets (a 'good' value according to some recent heuristic results by E.Teske)
  - Jacobian coordinate system to memorize points given in input to the iterating function of the Pollard's method.
  - Affine coordinate system to memorize points that are added to input points in the iterating function.
  - These different coordinate systems allows the use of mixed affine-jacobian addition formulas that have better performances than pure affine addition formulas, while lower storage needing of affine coordinate allows to store thepoints of the iterating function in the memory region of the video card reserved to constants (a faster memory region in which all thread can access together without the introduction of any latency in the execution of code).

ρ-Pollard with CUDA

- Further optimizations of the algorithm include:
  - Group has been split into 64 disjoint subsets (a 'good' value according to some recent heuristic results by E.Teske)
  - Jacobian coordinate system to memorize points given in input to the iterating function of the Pollard's method.
  - Affine coordinate system to memorize points that are added to input points in the iterating function.
  - These different coordinate systems allows the use of mixed affine-jacobian addition formulas that have better performances than pure affine addition formulas, while lower storage needing of affine coordinate allows to store thepoints of the iterating function in the memory region of the video card reserved to constants (a faster memory region in which all thread can access together without the introduction of any latency in the execution of code).

Performances

Tabella: Performances (points generated per second) of various architectures over the curve $ECCp-97$ of the Certicom list.

Architecture	Points/second
NVidia 8800GTS g92	720.000
Alpha 22164	440.000
Pentium II 300Mhz	125.000
Core 2 Duo E8500	10.000

Conclusions

- Even if video cards CUDA enabled have some serious limitations, many problems, if carefully restated, can take advantage of their computational power.
- Actually a huge group of people (among them there are also Tanja Lange and Daniel J.Bernstein) is working on the problem of the CERTICOM list defined on the field $\mathbb{F}_{2^{130}}$ and they are using not only common cpus, but also gpus that supports CUDA and PlayStation 3. This project can also be followed on Twitter at this address: http://twitter.com/ECCchallenge .

Conclusions

- Even if video cards CUDA enabled have some serious limitations, many problems, if carefully restated, can take advantage of their computational power.
- Actually a huge group of people (among them there are also Tanja Lange and Daniel J.Bernstein) is working on the problem of the CERTICOM list defined on the field $\mathbb{F}_{2^{130}}$ and they are using not only common cpus, but also gpus that supports CUDA and PlayStation 3. This project can also be followed on Twitter at this address: http://twitter.com/ECCchallenge .

The end.