Computer Security

Elliptic Curve Crypto

Hyoungshick Kim
Department of Software
College of Software
Sungkyunkwan University

Elliptic Curve Crypto (ECC)

- “Elliptic curve” is **not** a cryptosystem
- We can construct Elliptic curve versions of DH, RSA, etc.
- Elliptic curves may be more efficient
  - Fewer bits needed for same security
  - But the operations are more complex

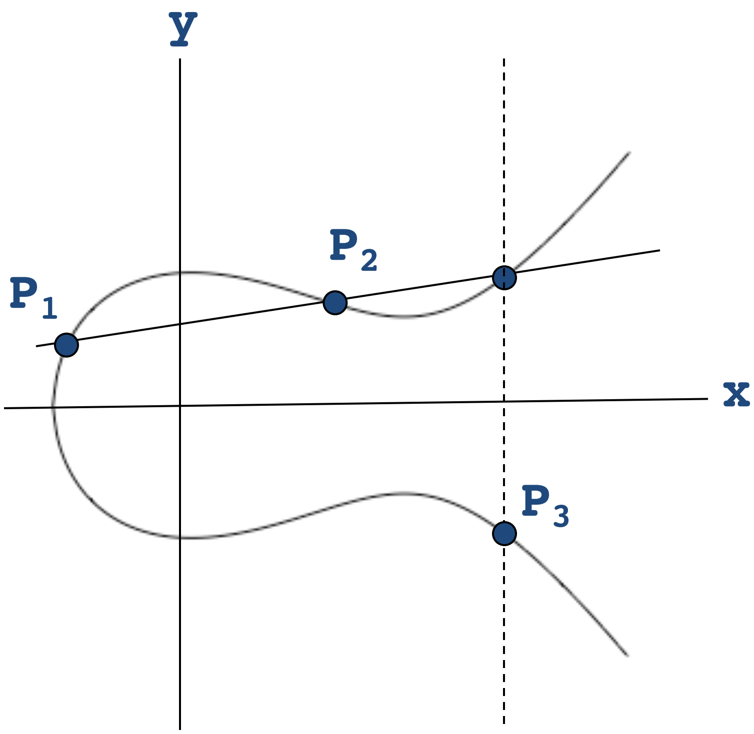
What is an Elliptic Curve?

- An elliptic curve E is the graph of an equation of the standard form

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

- Forms an Abelian group (commutative group)
- Symmetric about the x-axis
- Point at infinity acting as the identity element

Elliptic Curve picture



- Consider elliptic curve

  $E: y^2 = x^3 - x + 1$

- If P_1 and P_2 are on E, we can define

  $P_3 = P_1 + P_2$

  as shown in picture
- Addition is all we need

Points on Elliptic Curve

- Consider $y^2 = x^3 + 2x + 3 \pmod 5$

  $x = 0 \Rightarrow y^2 = 3 \Rightarrow$ no solution $\pmod 5$

  $x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1,4 \pmod 5$

  $x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0 \pmod 5$

  $x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1,4 \pmod 5$

  $x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0 \pmod 5$

- Then points on the elliptic curve are

  (1,1) (1,4) (2,0) (3,1) (3,4) (4,0)

  and the point at infinity: ∞

Elliptic Curve math

- Addition on: $y^2 = x^3 + ax + b \pmod{p}$

  $P_1=(x_1,y_1), P_2=(x_2,y_2)$

  $P_1 + P_2 = P_3 = (x_3,y_3)$ where

  $x_3 = m^2 - x_1 - x_2 \pmod{p}$

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

  and $m = (y_2-y_1)*(x_2-x_1)^{-1} \bmod p$, if $P_1 \neq P_2$

  $m = (3x_1^2+a)*(2y_1)^{-1} \bmod p$, if $P_1 = P_2$

  Special cases: If m is infinite, $P_3 = \infty$, and

  $\infty + P = P$ for all P

Elliptic Curve addition

- Consider $y^2 = x^3 + 2x + 3 \pmod 5$. Points on the curve are $(1,1)$ $(1,4)$ $(2,0)$ $(3,1)$ $(3,4)$ $(4,0)$ and ∞
- What is $(1,4) + (3,1) = P_3 = (x_3,y_3)$?

$$m = (1-4)*(3-1)^{-1} = -3*2^{-1}$$

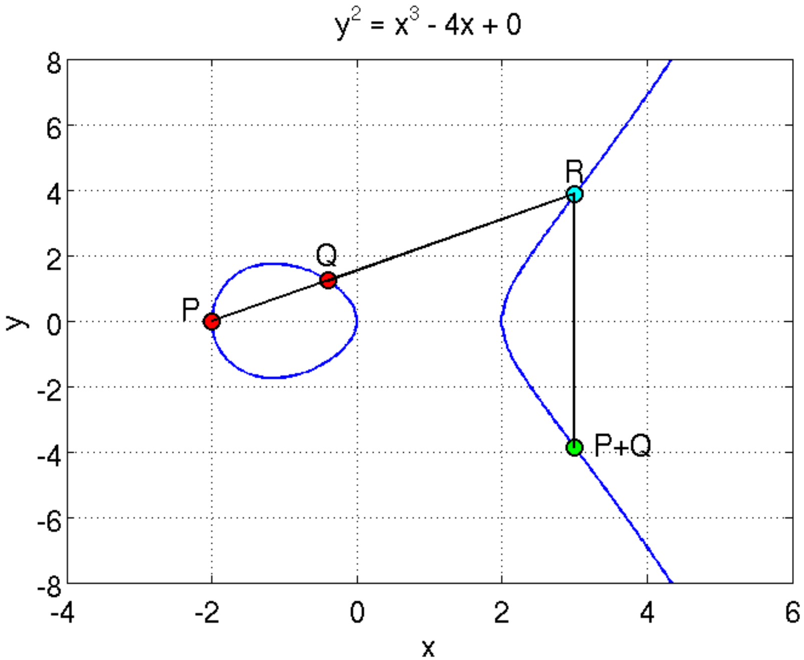
$$= 2(3) = 6 = 1 \pmod 5$$

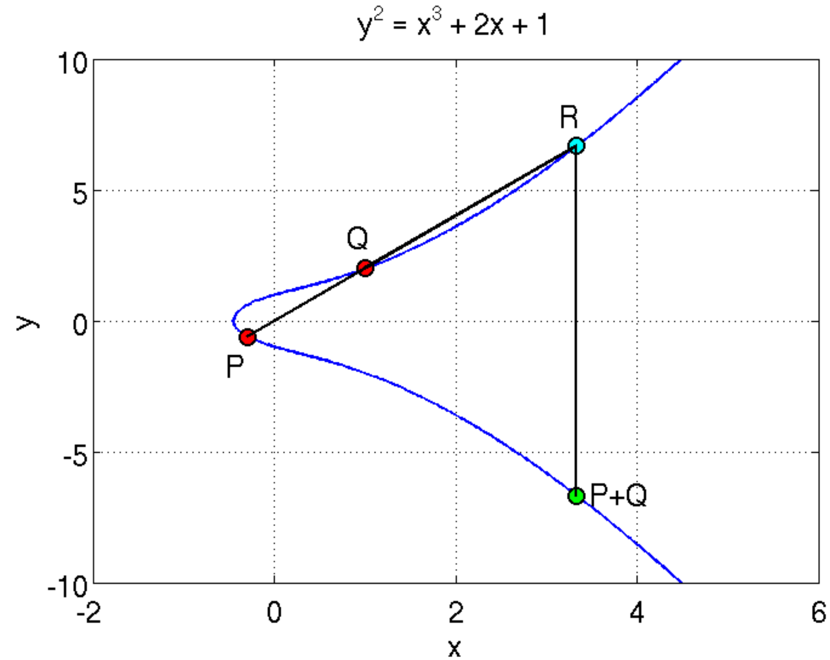
$$x_3 = 1 - 1 - 3 = 2 \pmod 5$$

$$y_3 = 1(1-2) - 4 = 0 \pmod 5$$

- On this curve, $(1,4) + (3,1) = (2,0)$

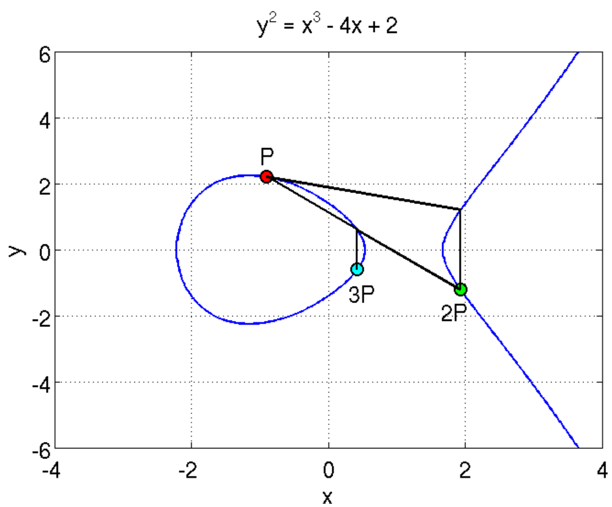
A visual look for P+Q

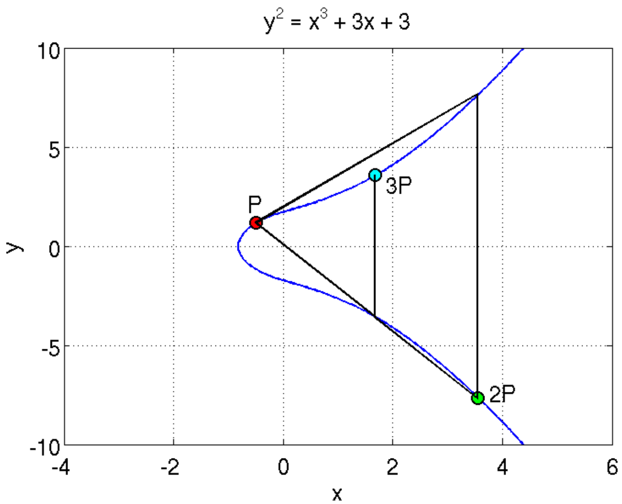




- Adding two points on the curve
- P and Q are added to obtain P+Q which is a reflection of R along the X axis

A visual look for kP





- A tangent at P is extended to cut the curve at a point; its reflection is 2P
- Adding P and 2P gives 3P
- Similarly, such operations can be performed as many times as desired to obtain Q = kP

Elliptic Curve Discrete Log Problem

- The security of ECC is due to the intractability or difficulty of solving the inverse operation of finding k given Q and P
- This is termed as the discrete log problem in ECC. The version applicable in ECC is called the Elliptic Curve Discrete Log Problem
- Methods to solve include brute force and Pollard's Rho attack both of which are computationally expensive or infeasible
- Exponential running time

How to use this

- Implementing group operations
  - Main operations - point addition and point multiplication
  - Adding two points that lie on an Elliptic Curve – results in a third point on the curve
  - Point multiplication is repeated addition
  - If P is a known point on the curve (aka Base point; part of domain parameters) and it is multiplied by a scalar k, Q=kP is the operation of adding P + P + P + P… + P (k times)
  - Q is the resulting public key and k is the private key in the public-private key pair

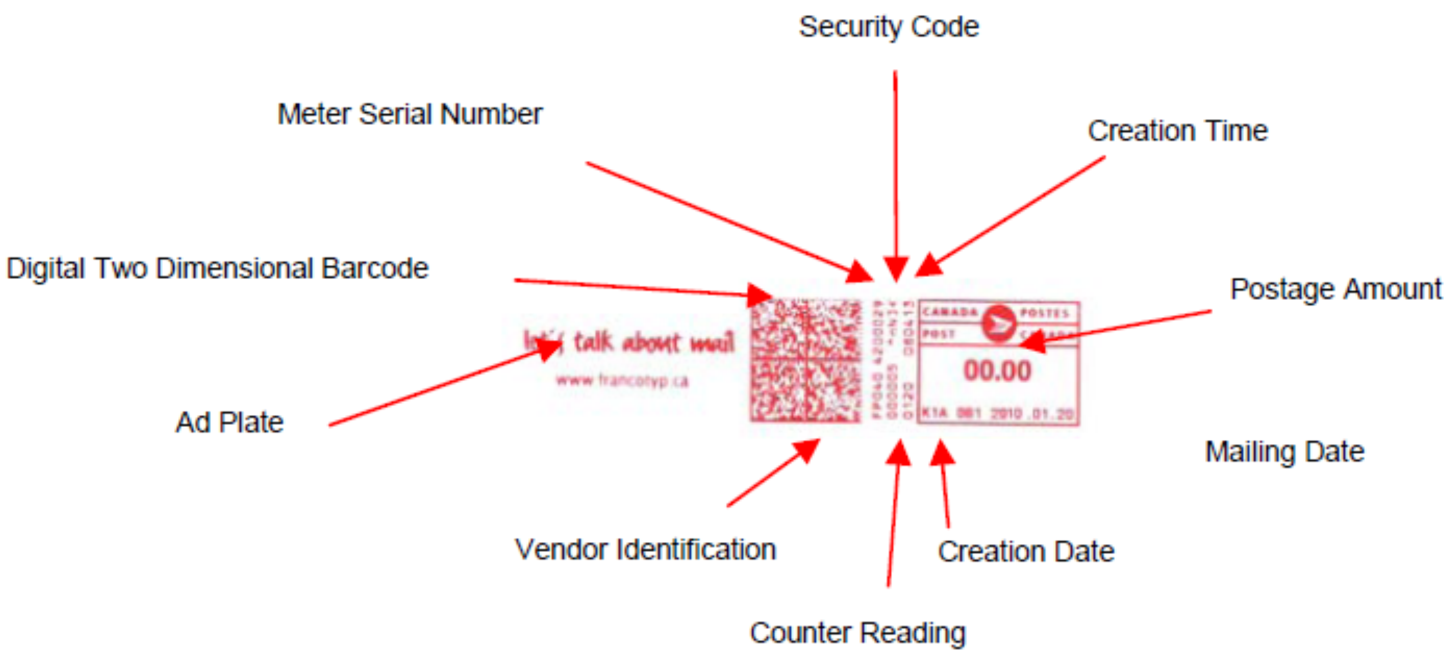
Why Elliptic Curve cryptography?

- Shorter key length
  - Same level of security as RSA achieved at a much shorter key length
- Lesser computational complexity
- Low power requirement
- Better secure
  - Secure because of the ECDLP
  - Higher security per key-bit than RSA

Comparable Key Sizes for Equivalent Security

Symmetric Encryption (Key Size in bits)	RSA and Diffie-Hellman (modulus size in bits)	ECC Key Size in bits
56	512	112
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Elliptic Curve Digital Signature Algorithm (ECDSA)



Canadian postage stamp that uses ECDSA

ECDSA : Key generation

- The private/public key-pair, (d,Q) is determine as follows by the Key-Generating-Center (KGC):

Private key

$$d \in_R \{1, \ldots, n-1\}$$

Public key

$$Q = dG$$

$$G$$

ECDSA - Signature generation

Once we have the domain parameters and have decided on the keys to be used, the signature is generated by the following steps.

1. A random number k, $1 \le k \le n-1$ is chosen
2. $kG = (x_1,y_1)$ is computed.
3. Next, $r = x_1 \bmod n$ is computed
4. We then compute $k^{-1} \bmod q$
5. $e = hash(m)$ where m is the message to be signed
6. $s = k^{-1}(e + dr) \bmod n$ where d is the private key of the sender.

We have the signature as (r,s)

ECDSA - Signature Verification

At the receiver's end the signature is verified as follows:

1. Verify whether r and s belong to the interval [1, n-1] for the signature to be valid.
2. Compute e = hash(m). The hash function should be the same as the one used for signature generation.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $(x_1,y_1) = u_1G + u_2Q$.
6. The signature is valid if $r = x_1 \bmod n$, invalid otherwise.

ECDSA - how the verification works properly

This is how we know that the verification works the way we want it to:

We have, $s = k^{-1}(e + dr) \bmod n$ which we can rearrange to obtain,

$k = s^{-1}(e + dr)$ which is

$$s^{-1}e + s^{-1}rd$$

That is, $k = s^{-1}e + s^{-1}rd = we + wrd = (u_1 + u_2 d) \pmod n$
where $w = s^{-1} \bmod n$

We have $u_1G + u_2Q = (u_1 + u_2 d)G = kG$ which translates to $x_1 = r$
where $Q = dG$.

Questions?


Sungkyunkwan
Campus
Social
Graduate
campus
Sciences
University
Seoul
Natural
Suwon
School
Korean
established
university
edit
English
constructed
Building
Global
Institute
Professors
Confucian
Humanities
schools
Science
Advanced
Education
Engineering
located
Joseon
Korea
lecturers
Administration
Academic
ASLE-Korea
Hall
two
four
also
city