# The Math Behind Elliptic Curves

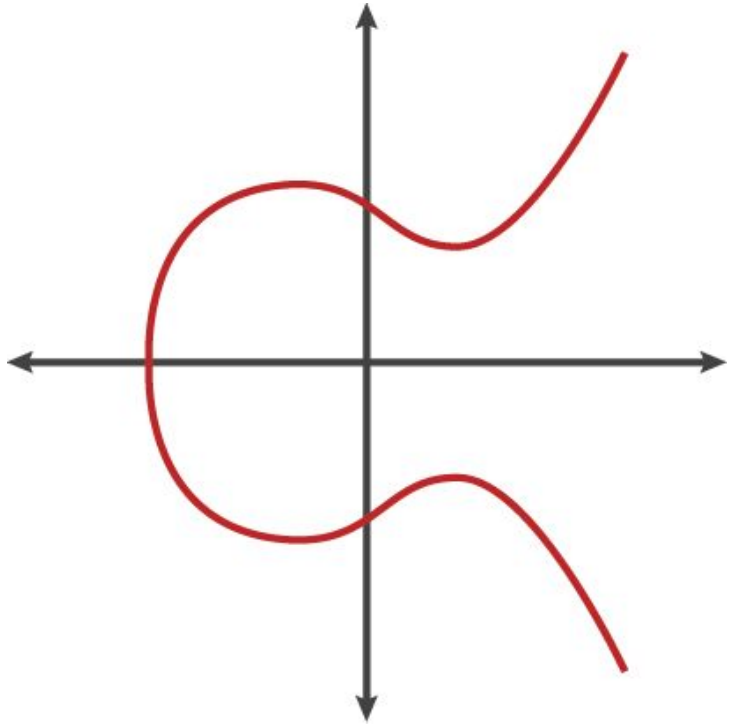
Blockchain Technology (CS467)

M.Tech. I, Semester II

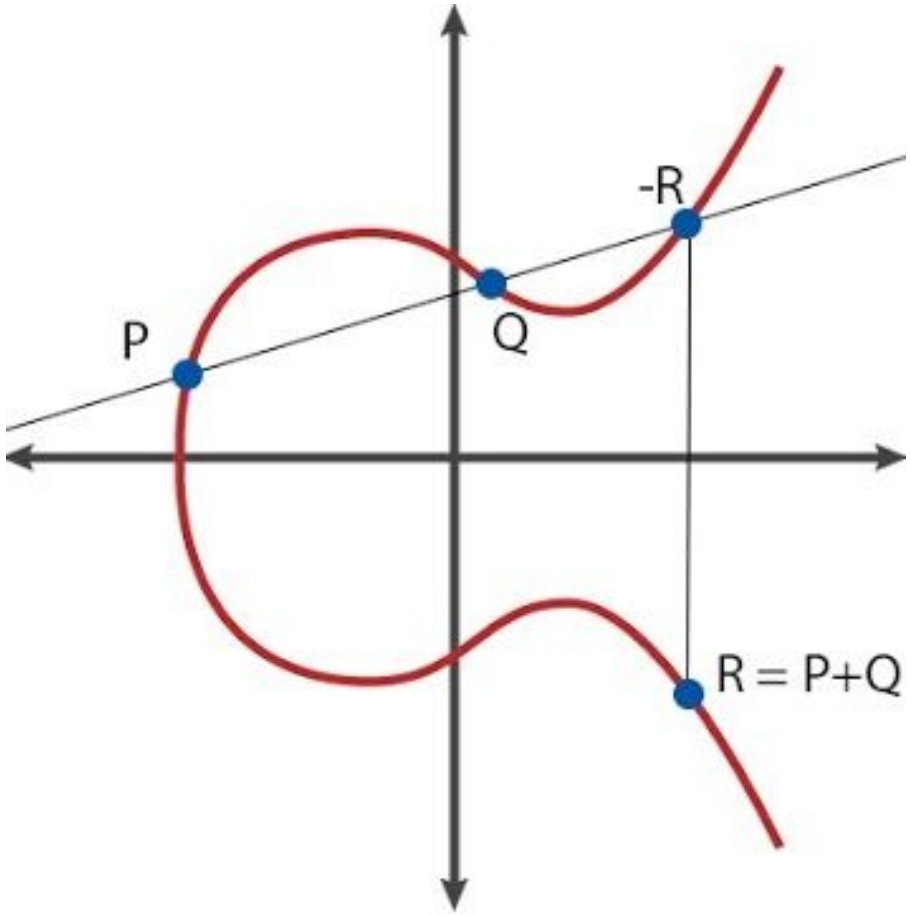

Department of Computer Science and Technology
S. V. National Institute of Technology-Surat

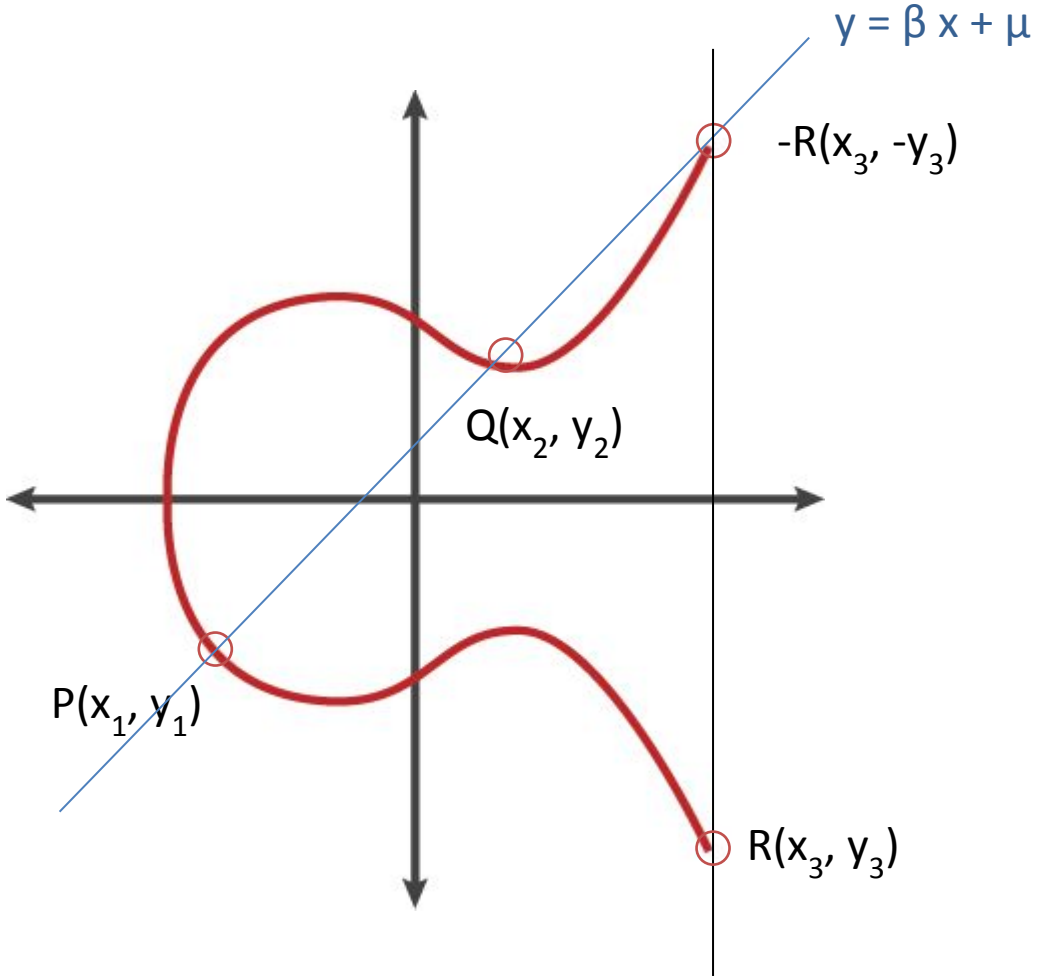
# Elliptic Curves over Prime Field – GF(p)

- $y^2 = x^3 + ax + b$ (weierstrass equation)



# Addition Law



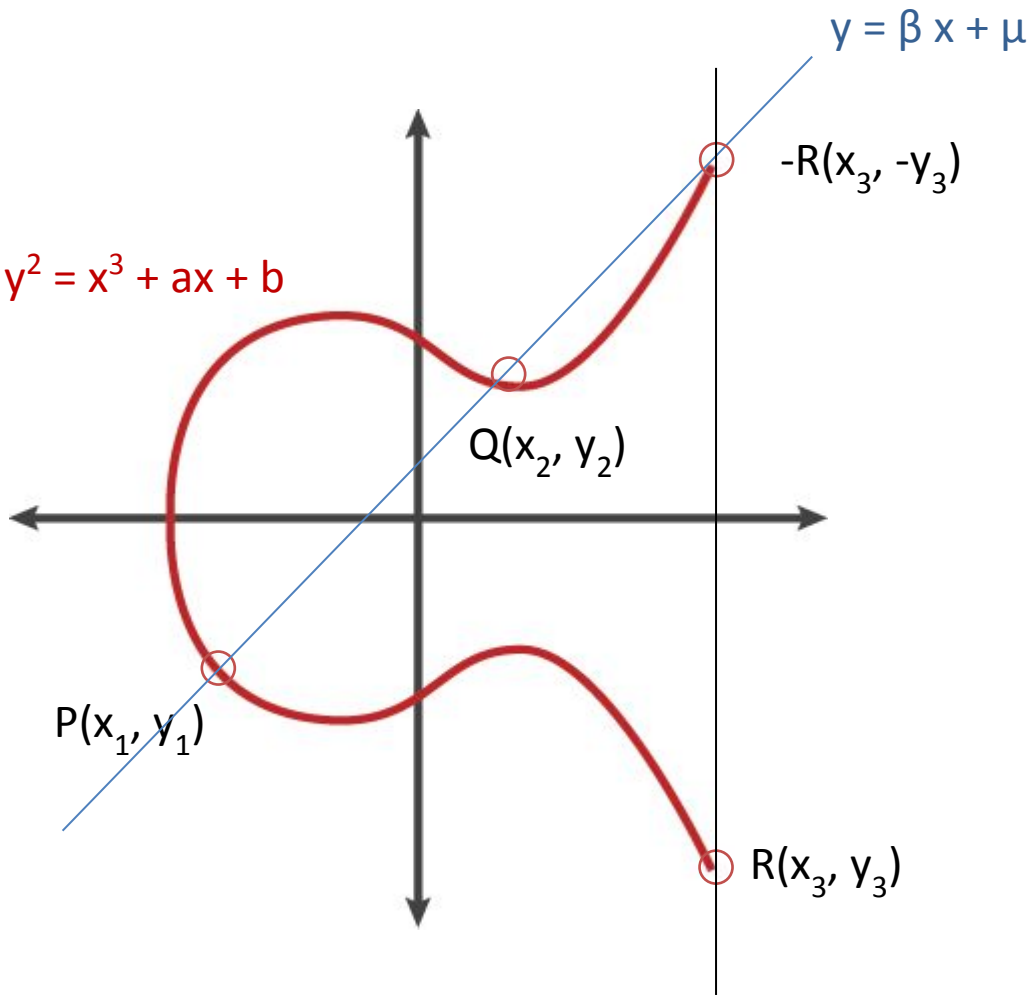



Points P and –R are on the blue line, they must satisfy the y = β x + μ equation

| P | -R |
|---|---|
| $y_1 = \beta.x_1 + \mu$ | $-y_3 = \beta.x_3 + \mu$ |
| $\mu = y_1 - \beta.x_1$ → | $-y_3 = \beta.x_3 + (y_1 - \beta.x_1)$ |
| | $y_3 = -\beta.x_3 - y_1 + \beta.x_1$ |
| | $y_3 = \beta.(x_1 - x_3) - y_1$ |

$$\mathbf{y_3 = \beta.(x_1 - x_3) - y_1}$$

β and $x_3$ are unknown




$y^2 = x^3 + ax + b \qquad y = \beta x + \mu$

$(\beta x + \mu)^2 = x^3 + ax + b$

$\beta^2 x^2 + \mu^2 + 2\beta\mu x = x^3 + ax + b$

$x^3 - \beta^2 x^2 + ax - 2\beta\mu x + b - \mu^2 = 0$

$x^3 - \beta^2 x^2 + (a - 2\beta\mu)x + (b - \mu^2) = 0$

# Polynomial Relation Rule

- Sum of the roots have to be equal to negative coefficient of $x^2$
- **Proof:**
- $y^2 = x^3 + ax + b$ has 3 distinct roots because it is cubic equation
- Let's say (a, b, c)

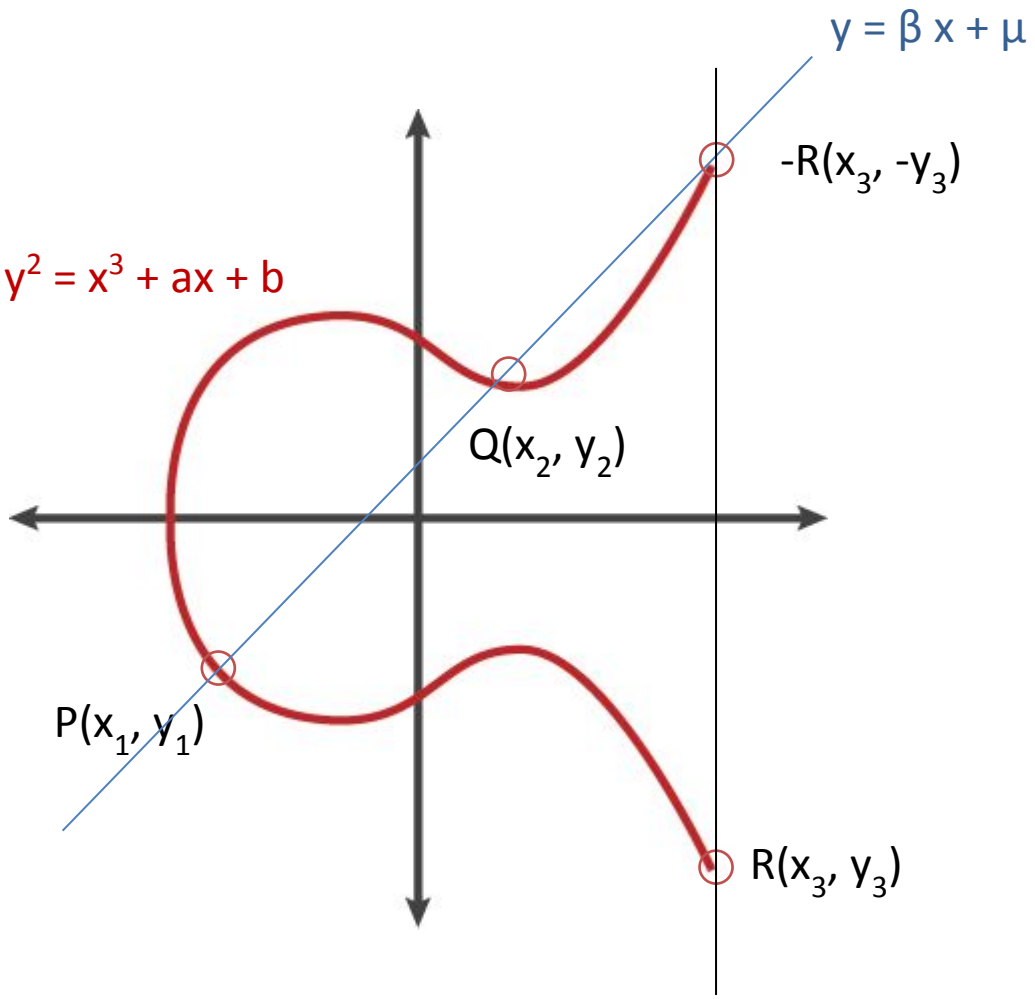
$(x - a).(x - b).(x - c) = 0$

$[(x - a).(x - b)].(x - c) = 0$

$[x^2 - b\,x - a\,x - a\,b].(x - c) = 0$

$x^3 - c\,x^2 - b\,x^2 + b\,c\,x - a\,x^2 + a\,c\,x - a\,b\,x + abc = 0$

$x^3 - (a + b + c)\,x^2 - (ab + ac + bc)x + abc = 0$




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

$$(\beta x + \mu)^2 = x^3 + ax + b$$

$$\beta^2 x^2 + \mu^2 + 2 \beta x \mu = x^3 + ax + b$$

$$x^3 - \beta^2 x^2 + ax - 2 \beta \mu x + b - \mu^2 = 0$$

$$x^3 - \beta^2 x^2 + (a - 2 \beta \mu)x + (b - \mu^2) = 0$$

---

**Polynomial relation rule** states sum of roots equal to negative coefficient of $x^2$

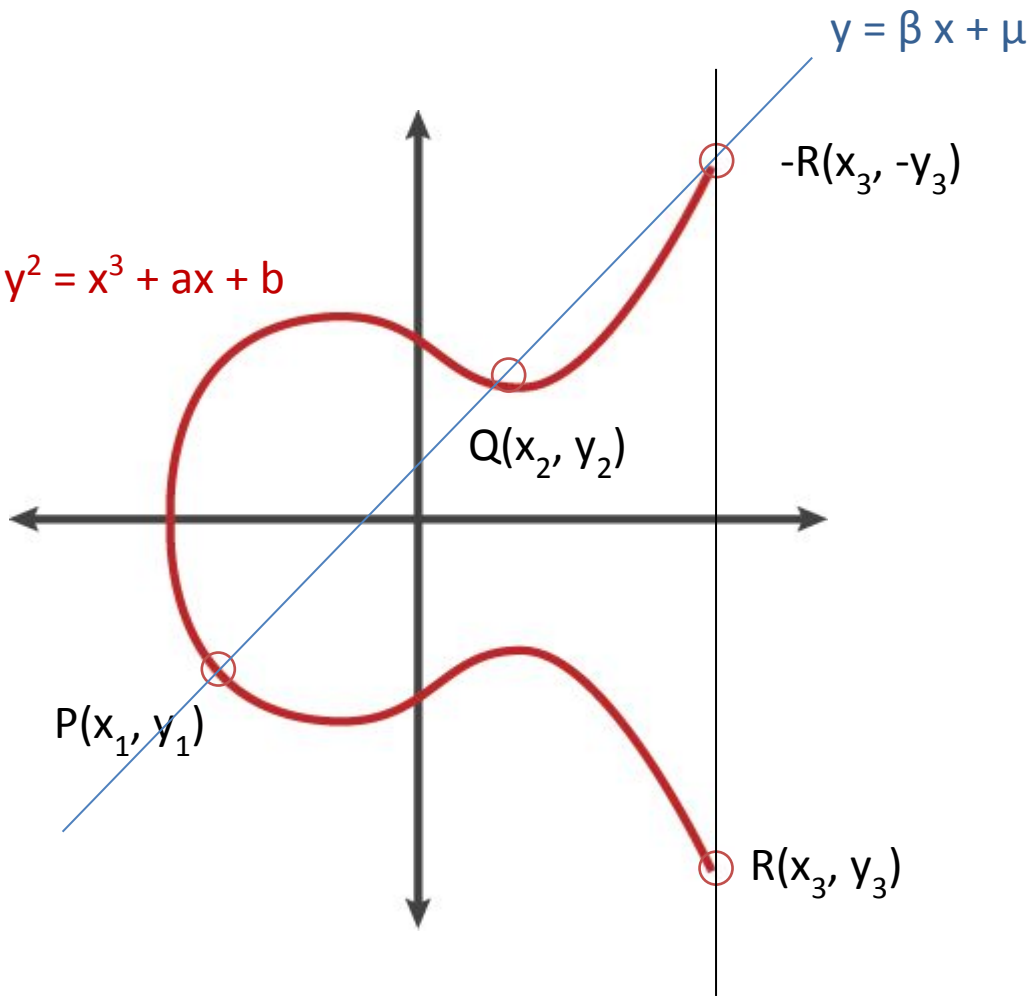
---

$$-( -\beta^2 ) = x_1 + x_2 + x_3$$

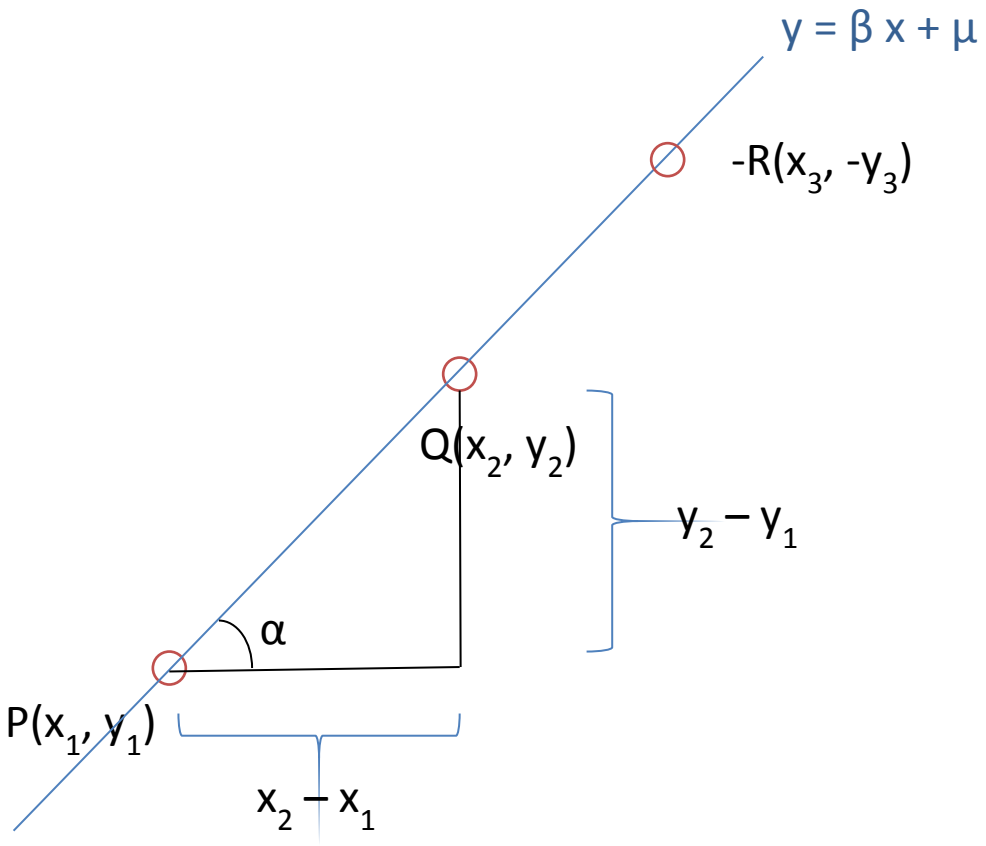
$$\beta^2 = x_1 + x_2 + x_3$$

$$\mathbf{x_3 = \beta^2 - x_1 - x_2}$$

β is still unknown

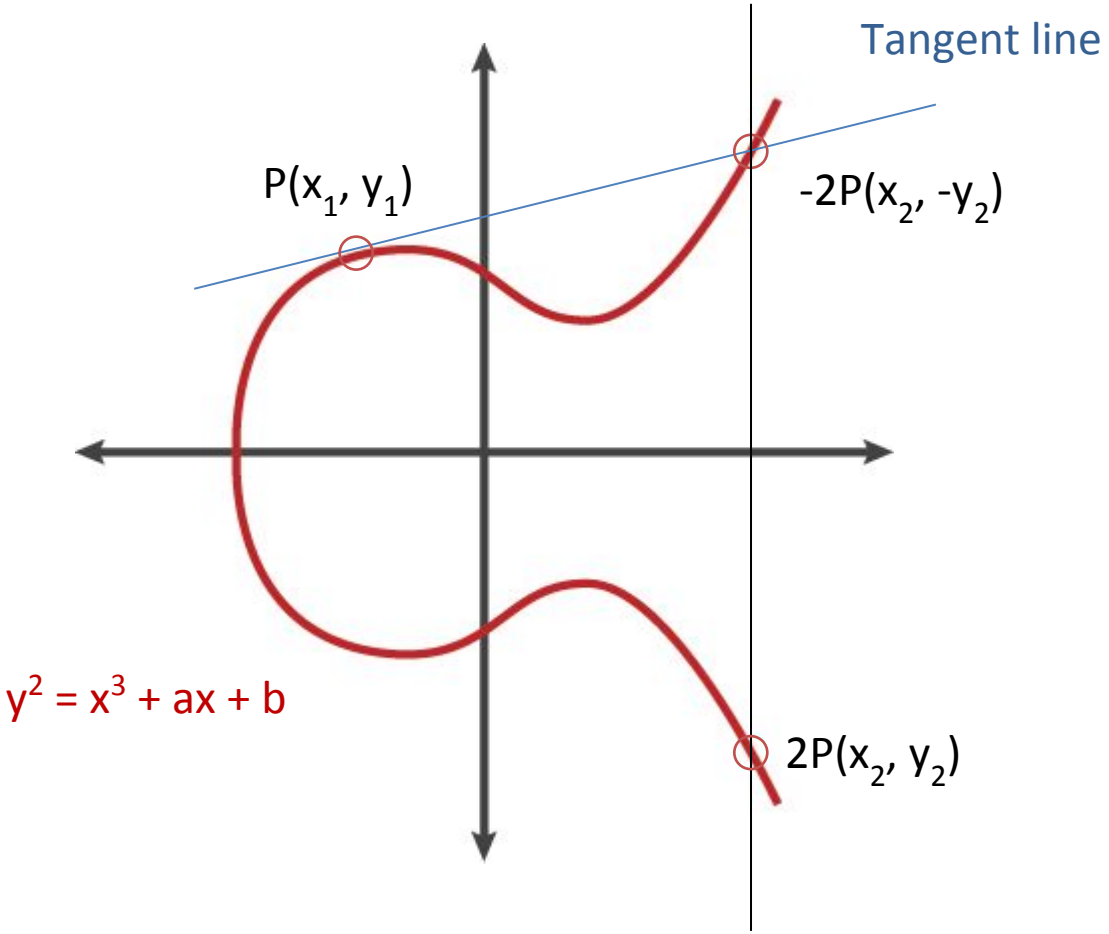

y = β x + μ
-R(x3, -y3)
y2 = x3 + ax + b
Q(x2, y2)
P(x1, y1)
R(x3, y3)

Just focus on the linear line




$$\beta = \tan\alpha = (y_2 - y_1)/(x_2 - x_1)$$

# Doubling a point




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

$2y.dy = (3x^2 + a).dx$

$dy/dx = (3x^2 + a)/2y$

$$\beta = (3x_1^{\,2} + a)/2y_1$$

$$x_3 = \beta^2 - x_1 - x_1$$

$$y_3 = \beta.(x_1 - x_3) - y_1$$

$$P(x_1, y_1) + P(x_1, y_1) = 2P(x_2, y_2)$$

# To sum up

$$P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3)$$

P != Q

$$\beta = (y_2 - y_1)/(x_2 - x_1)$$

P = Q

$$\beta = (3 x_1^{\,2} + a)/2 y_1$$

$$x_3 = \beta^2 - x_1 - x_2$$

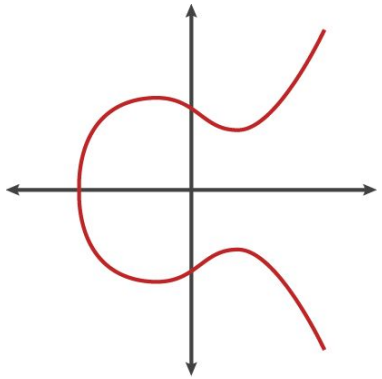
$$y_3 = \beta.(x_1 - x_3) - y_1$$

# Elliptic Curves

- Weierstrass form : $y^2 = x^3 + ax + b$

# Summary

Over Prime Field



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

Point addition

$$\beta = (y2 - y1)/(x2 - x1)$$
$$x3 = \beta^2 - x1 - x2$$
$$y3 = \beta(x1 - x3) - y1$$

Doubling

$$\beta = (3x1^2 + a)/2y1$$
$$x3 = \beta^2 - 2x1$$
$$y3 = \beta(x1 - x3) - y1$$

# Addition

- Group operation +

  Given $P, Q \in E, P = (x_1, y_1), Q = (x_2, y_2)$

  Compute $R = P + Q = (x_3, y_3)$

  - Addition $(P \neq Q)$

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

$$x_3 = \lambda^2 - x_1 - x_2$$

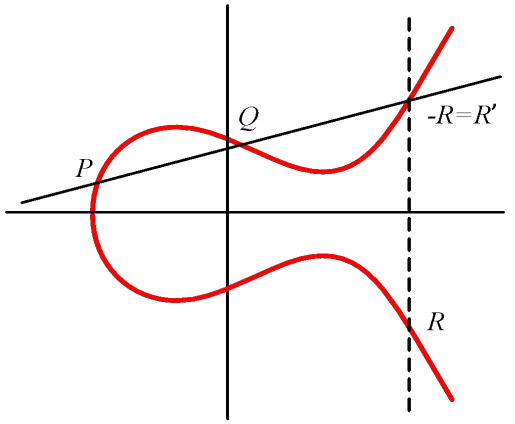
$$y_3 = (x_1 - x_3)\lambda - y_1$$

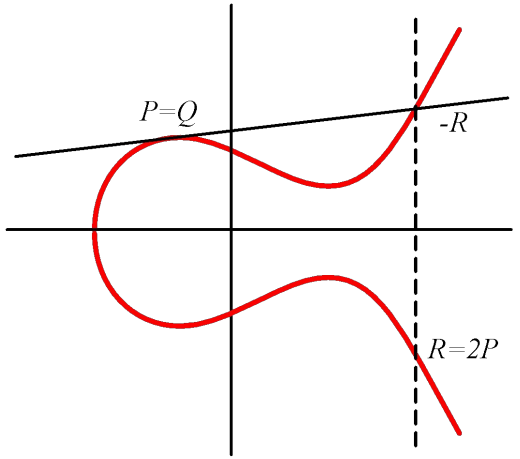
  - Doubling

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

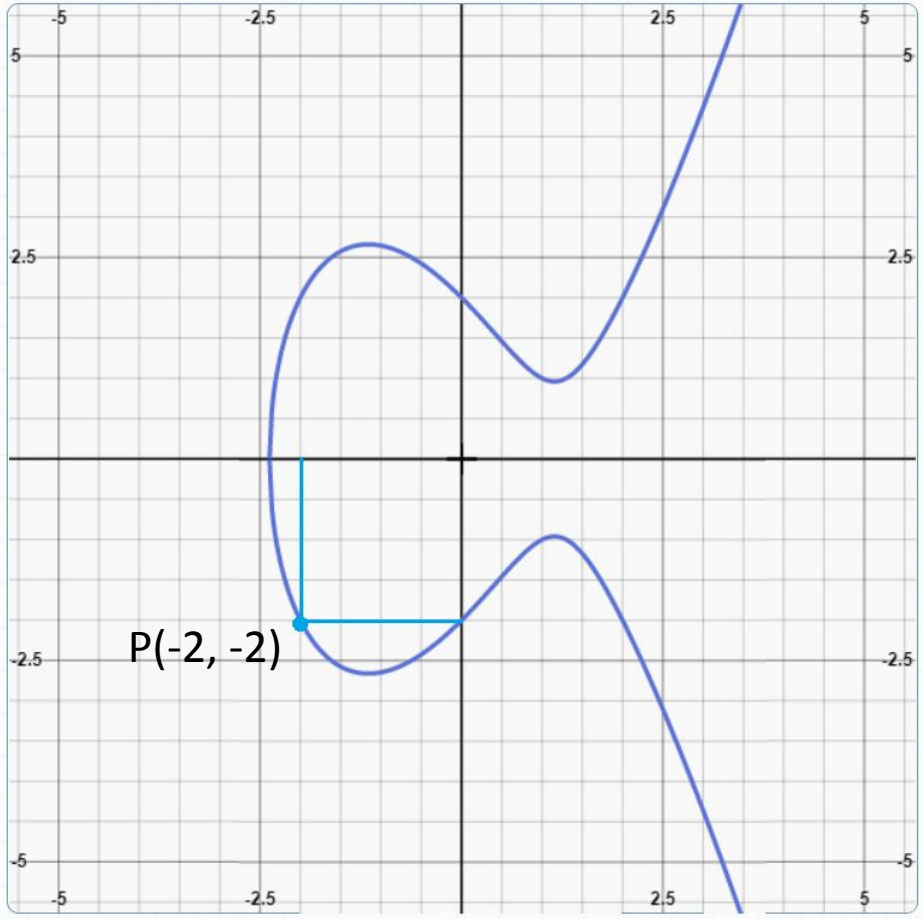
$$x_3 = \lambda^2 - 2x_1$$

$$y_3 = (x_1 - x_3)\lambda - y_1$$



# Calculation of kP (e.g. 1017P) – brute force




a = -4; b = 4
x1 = -2; y1 = -2

#Doubling (2P)
$\beta = (3x1^2 + a)/2y1$
$x2 = \beta^2 - 2x1$
$y2 = \beta(x1 - x2) - y1$

#Point addition
for i in range(1, 1017):
  $\beta = (y2 - y1)/(x2 - x1)$
  $x3 = \beta^2 - x1 - x2$
  $y3 = \beta(x1 - x3) - y1$

Requires 1017 steps. O(n) complexity is hard for really large integers.

# Example

- Example (addition):
  Given
  - $E : y^2 = x^3 - 25x$

$$P = (x_1, y_1) = (0,0),\ Q = (x_2, y_2) = (-5,0),\ P + Q = (x_3, y_3)$$

# Example

- Example (addition):
  Given
  - $E : y^2 = x^3 - 25x$

    $P = (x_1, y_1) = (0,0),\ Q = (x_2, y_2) = (-5,0),\ P + Q = (x_3, y_3)$

    $$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0 - 0}{-5 - 0} = 0$$

    $$x_3 = \lambda^2 - x_1 - x_2 = 0^2 - 0 - (-5) = 5$$

    $$y_3 = (x_1 - x_3)\lambda - y_1 = (0 - 5) \times 0 - 0 = 0$$

# Example

- Example (doubling):
  Given $E: y^2 = x^3 - 25x$

$$P = (x_1, y_1) = (-4,6),\ 2P = (x_2, y_2)$$

$$\lambda = \frac{3x_1^{\,2} + a}{2y_1} = \frac{3(-4)^2 - 25}{2 \times 6} = \frac{23}{12}$$

$$x_2 = \lambda^2 - 2x_1 = \left(\frac{23}{12}\right)^2 - 2 \times (-4) = \frac{1681}{144}$$

$$y_2 = (x_1 - x_2)\lambda - y_1 = \left(-4 - \frac{1681}{144}\right) \times \frac{23}{12} - 6 = -\frac{62279}{1728}$$

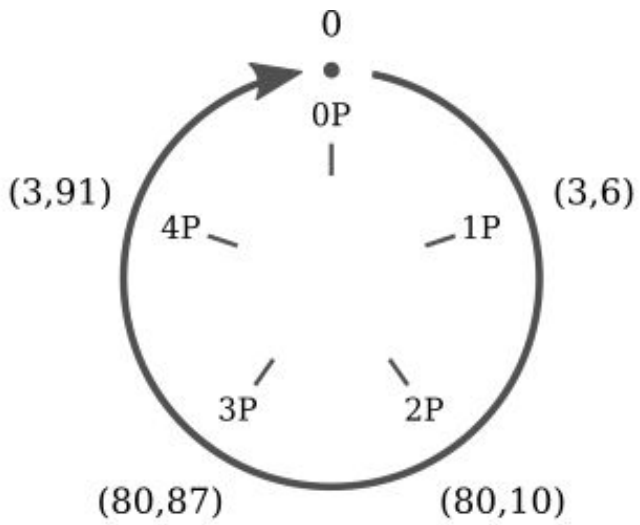
# ECC

- **Scalar multiplication and cyclic subgroups**
- multiplication can be defined as:
- nP = P + P + P+ ….. + P (n times)

# ECC

- multiplication over points for elliptic curves in has an interesting property. Take the curve $y^2 = x^3+2x+3 \pmod{97}$ and the point (3,6). Now calculate all the multiples of (3,6):
- The multiples of (3,6) are just five distinct points

  - $0P = 0$
  - $1P = (3,6)$
  - $2P = (80,10)$
  - $3P = (80,87)$
  - $4P = (3,91)$
  - $5P = 0$
  - $6P = (3,6)$
  - $7P = (80,10)$
  - $8P = (80,87)$
  - $9P = (3,91)$
  - ...



# Comparable Key Sizes for Equivalent Security

| Symmetric scheme (key size in bits) | ECC-based scheme (size of *n* in bits) | RSA/DSA (modulus size in bits) |
|---|---|---|
| 56 | 112 | 512 |
| 80 | 160 | 1024 |
| 112 | 224 | 2048 |
| 128 | 256 | 3072 |
| 192 | 384 | 7680 |
| 256 | 512 | 15360 |

# ECDSA

| Parameter | |
|---|---|
| CURVE | the elliptic curve field and equation used |
| *G* | elliptic curve base point, a point on the curve that generates a subgroup of large prime order n |
| *n* | integer order of *G*, means that n X G = O, where O is the identity element. |
| D | the private key (randomly selected) |
| Q | the public key D X G (calculated by elliptic curve) |
| *m* | the message to send |

# 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, $1 \leq k \leq n-1$ is chosen
2. $kG = (x_1, y_1)$ is computed. $x_1$ is converted to its corresponding integer $x_1'$
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.

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

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

This is nothing but $we + wrd = (u_1 + u_2d) \pmod n$

We have $u_1G + u_2Q = (u_1 + u_2d)G = kG$ which translates to $v = r$.

# Elliptic Curve Pintsov Vanstone Signature (ECPVS)

- Signature scheme using Elliptic Curves
- More efficient than RSA as overhead is extremely low

# ECPVS

- Signature Generation

The plaintext message is split into two parts: part C representing the data elements requiring confidentiality and part V representing the data elements presented in plaintext. Both the parts are signed. The signature is generated as follows:

1. A random number k, $1 \leq k \leq n-1$ is chosen.

2. Calculate the point R on the curve ($R = kG$).

3. Use point R and a symmetric encryption algorithm to get $e = T_R(C)$.

4. Calculate a variable d such that $d = HASH(e \,||\, I_A \,||\, V)$

where $I_A$ is the identity of the mailer terminal.

5. Now calculate the other part of the signature s as follows: $s = ad + k \pmod{n}$; where a is the private key of the sender.

The signature pair (s,e) is transmitted together with the portion V of the plaintext.

# ECPVS

- Signature Verification

1. Retrieve $Q_A$ ($Q_A$ is mailer A's public key)
2. Calculate the variable d = *HASH*(*e* || $I_A$ || *V*) using the same HASH algorithm as the one used for generating the signature.
3. Compute $U = sG - dQ_A$.
4. Recover $C = T_u^{-1}(e)$.
5. Run a redundancy test on C. If the test fails, discard the message. Else, the plaintext is recovered.

We have, $s = ad + k$. Multiply by base point G to obtain $sG = adG + kG$ which is $dQ_A + R$

Therefore, $R = sG - dQ_A$ which is U. Comparing the decrypted versions, m and m' obtained using U and R, we ascertain the validity of the signature

# Elliptic Curve Diffie-Hellman (ECDH)

- Elliptic curve variant of the key exchange Diffie-Hellman protocol.
- Decide on domain parameters and come up with a Public/Private key pair
- To obtain the private key, the attacker needs to solve the discrete log problem

# ECDH

- How the key exchange takes place:

1. Alice and Bob publicly agree on an elliptic curve E over a large finite field F and a point P on that curve.
2. Alice and Bob each privately choose large random integers, denoted a and b
3. Using elliptic curve point-addition, Alice computes aP on E and sends it to Bob. Bob computes bP on E and sends it to Alice.
4. Both Alice and Bob can now compute the point abP Alice by multiplying the received value of bP by her secret number a and Bob vice-versa.
5. Alice and Bob agree that the x coordinate of this point will be their shared secret value.

# Pros and Cons

- Pros
  - Shorter Key Length
    - Same level of security as RSA achieved at a much shorter key length
  - Better Security
    - Secure because of the ECDLP
    - Higher security per key-bit than RSA
  - Higher Performance
    - Shorter key-length ensures lesser power requirement – suitable in wireless sensor applications and low power devices
    - More computation per bit but overall lesser computational expense or complexity due to lesser number of key bits

# Pros and Cons

- Cons
  - Relatively newer field
    - Idea prevails that all the aspects of the topic may not have been explored yet – possibly unknown vulnerabilities
    - Doesn’t have widespread usage
  - Not perfect
    - Attacks still exist that can solve ECC (112 bit key length has been publicly broken)
    - Well known attacks are the Pollard’s Rho attack (complexity $O(\sqrt{n})$ ), Pohlig’s attack, Baby Step,Giant Step etc