Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields.
ECC requires smaller keys compared to non-ECC cryptography to provide equivalent security.
For example, to achieve the same security ensured by ECC having private key of 256 bit length, it is required to use over 3000 bit private key length for RSA cryptosystem and others.

δ

Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks.
Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme.

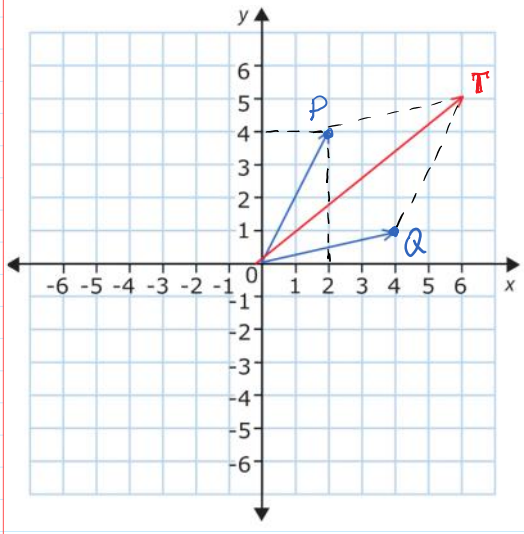
Elliptic Curve Digital Signature Algorithm - Bitcoin Wiki **(ECDSA)**
https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_AlgorithmFeb 10, 2015
Elliptic Curve Digital Signature Algorithm or **ECDSA** is a cryptographic algorithm used by Bitcoin, Ethereum and other blockchain methods to ensure that funds can only be spent by their owner.
https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

p=11
$xa=\in$

So far we exploited algebraic grup $\boldsymbol{Z_p^*}=\{1, 2, 3, \ldots, p\text{-}1\}$ with defined the following operations $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$.
Finite Field denoted by $\boldsymbol{F_p}$ (or rarely $\boldsymbol{Z_p}$), when: p is prime.

$\boldsymbol{F_p}=\{0, 1, 2, 3, \ldots, p\text{-}1\}$; $+_{\mathbf{mod}\ p}$, $-_{\mathbf{mod}\ p}$, $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$, (except division by 0).
Cyclic Group: $\boldsymbol{Z_p}^* = \{1, 2, 3, \ldots, p\text{-}1\}$; $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$.
For example, if p=11, then one of the generetors is g=2.

Coordinate systems XOY in subsequent examples are defined in the plane of real numbers.

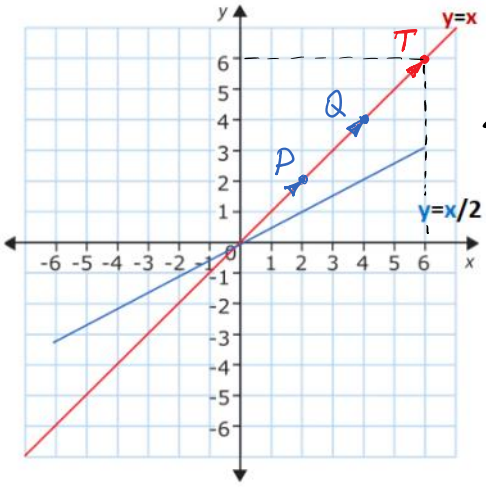


$P(x_P, y_P) = (2,4)$
$Q(x_Q, y_Q) = (4,1)$
$P+Q=(2+4,\ 4+1)$
$T = P+Q = (6,5)$

$T = P(x_P, y_P) \boxplus Q(x_Q, y_Q) =$
$= T(x_P + x_Q,\ y_P + y_Q) = T(x_T, y_T)$

$x_T = x_P + x_Q$
$y_T = y_P + y_Q$

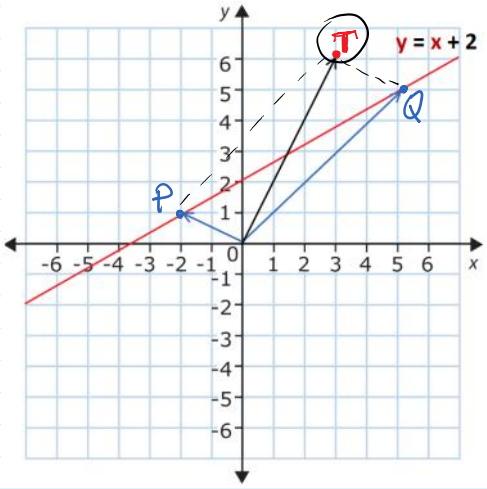
$T_2 = P + P = 2P =$

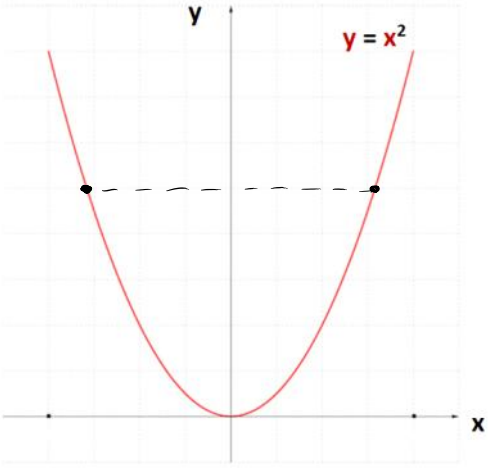


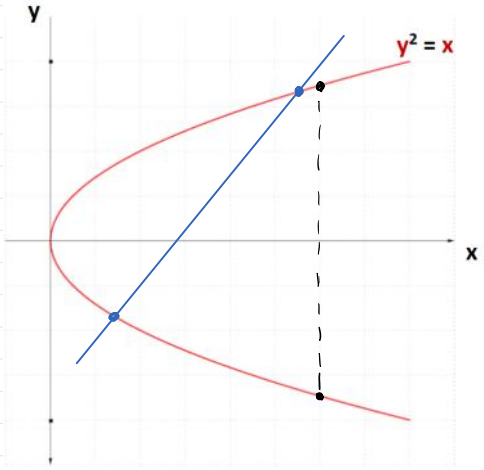
$x_T = 2+4 = 6$

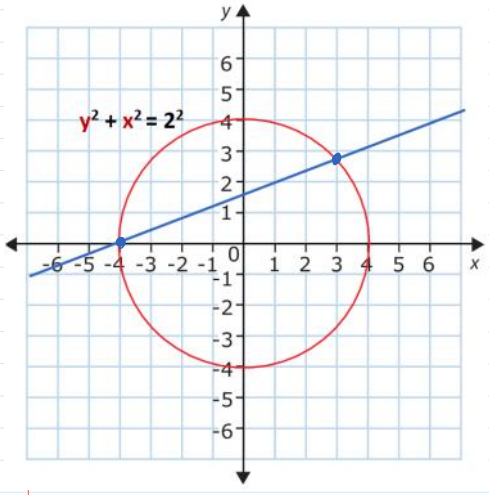
$y_T = 2+4 = 6$

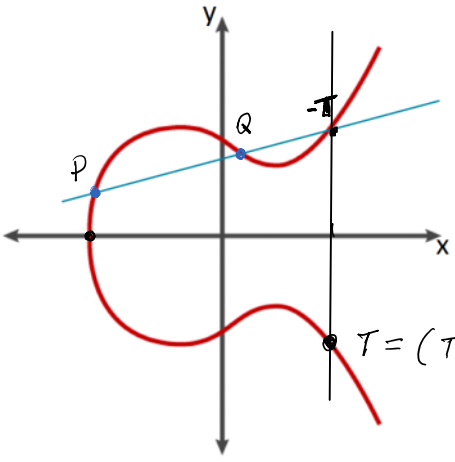
$|T| = \sqrt{6^2+6^2}$











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

Operations are performed in field F_p

$P \boxplus Q = T \in EC$

$T \boxplus (-T) = O$

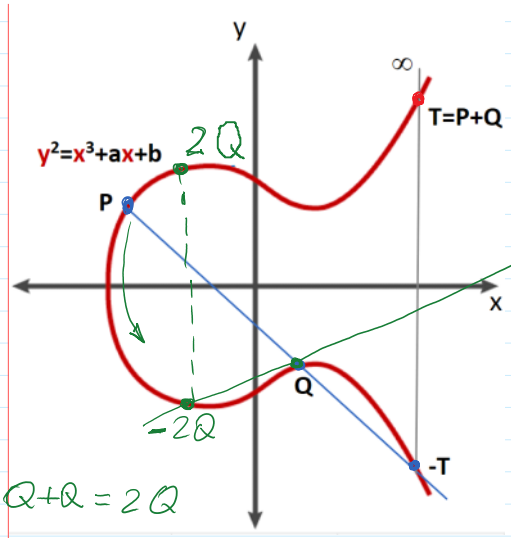
$a = \frac{\Delta}{x_2 - x_1}$

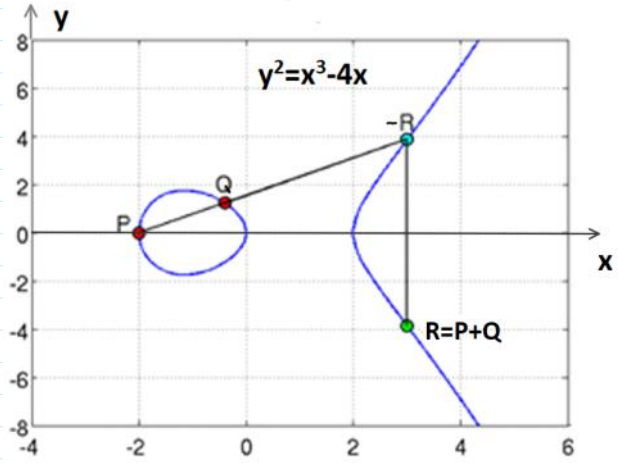
Elliptic curve ha a property that if line crosses two points, then there is a third crossing point in the curve.

Points in the plane or plane curve we denote by the capital letters, e.g. **A**, **G**, **P**, **Q**, etc.
Numbers-scalars we denote by the lowercase letters, e.g., **a**, **g**, **x**, **y**, **z**, etc.

Addition of points P and Q in EC: $\mathbf{P \boxplus Q = T}$

$\mathbf{P(x_P, y_P) + Q(x_Q, y_Q) = T(x_T, y_T)}$





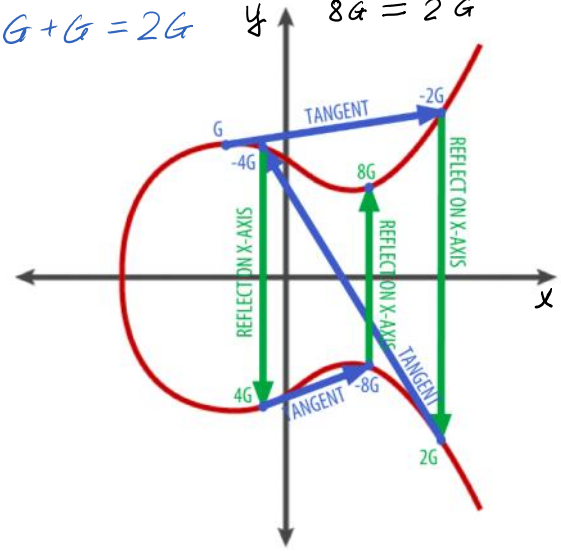
$5-5 \bmod 10 = 0$ $\quad T-T = „0" \rightarrow T+(-T) = „0" \equiv \infty$

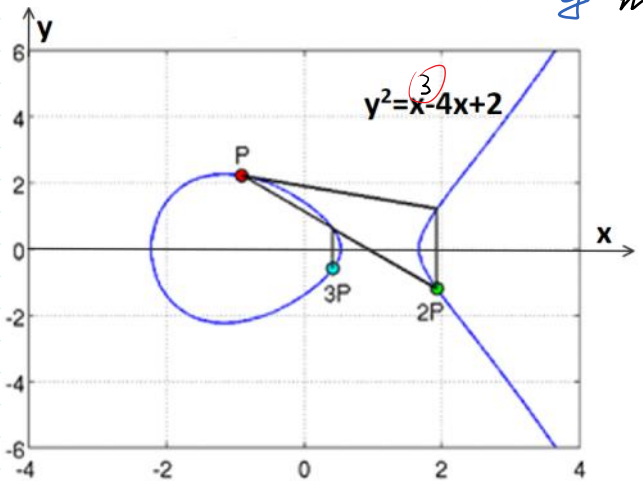
$7+0 \bmod 10 = 7$ $\quad T+\infty = T$

When z is large, $z \sim 2^{256} \rightarrow |z| = 256$ bits:

Doubling of points allows effectively compute point $A = zG$ $\quad a = g^x \bmod p$

$G+G=2G$ $\quad 8G = 2^3 G$





ECDSA animacija

Signing and Verifying Ethereum Signatures – Yos Riady · Software Craftsman

https://medium.com/coinmonks/elliptic-curve-cryptography-6de8fc748b8b

For current cryptographic purposes, an *elliptic curve* is a plane curve over a finite field $\boldsymbol{F_p}=\{0, 1, 2, 3, \ldots, \boldsymbol{p}\text{-}1\}$, (rather than the real numbers) $\boldsymbol{p}$-is prime.

Z_p^*

Which consists of the points satisfying the equation over $\boldsymbol{F_p}$

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

along with a distinguished point at infinity, denoted by $\boldsymbol{0}$ (∞).

Finite field is an algebraic structure, where 4 algebraic operations: $+_{\bmod p}$, $-_{\bmod p}$, $\cdot_{\bmod p}$, $:_{\bmod p}$ are defined except the division by 0 excluded.

Elliptic Curve Group (ECG)
Number of points **N** of Elliptic Curve (**EC**) with coordinates (***x***, ***y***) is an order of Elliptic Curve Group (**ECG**).
Addition operation ⊞ of points in **ECG**: let points $P(x_P, y_P)$ and $Q(x_Q, y_Q)$ are in EC with coordinates (x_P,y_P) and (x_Q,y_Q) then $P \boxplus Q = T$ with coordinates (x_T, y_T) in **EC**.
Neutral element is group zero ***0*** at the infinity (∞) of [XOY] plane.
Multiplication of any **EC** point ***G*** by scalar ***z***: $T=z*G$; $T=G \boxplus G \boxplus G \boxplus ... \boxplus G$; ***z***–times.
Generator or **Base Point** ***G***: ECG={ $i*G$; i=1,2,…,**N**}; $N*G=0$ and $q*G \neq 0$ if $q<N$.

Because this curve is defined over a finite field of prime order instead of over the real numbers, it looks like a pattern of dots scattered in two dimensions, which makes it difficult to visualize. However, the math is identical to that of an elliptic curve over real numbers. As an example, Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17 shows the same elliptic curve over a much smaller finite field of prime order 17, showing a pattern of dots on a grid. The secp256k1 bitcoin elliptic curve can be thought of as a much more complex pattern of dots on a unfathomably large grid.

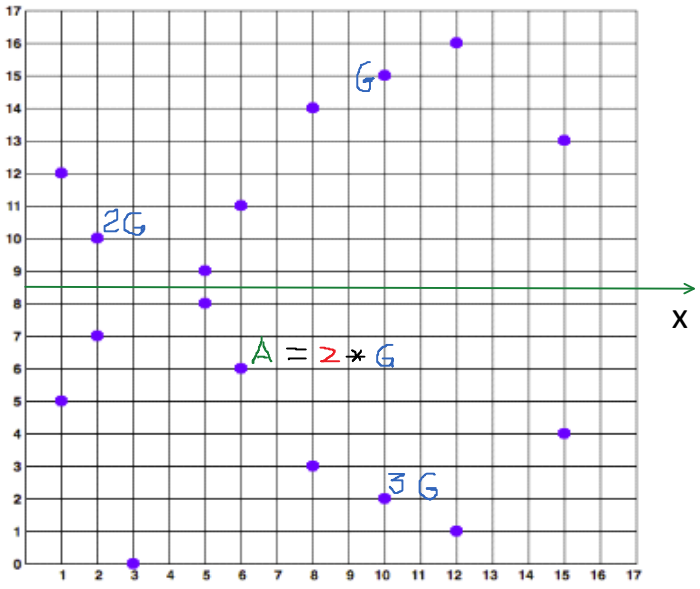


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

DEF: $g^x \bmod p = a$;

ECDEF: $z * G = A = (x_A, y_A)$;

z - is a random number of 256 bit length and is a private key **PrK** in Elliptic Curve Cryptography - **ECC**
A - is apoint of Elliptic Curve (**EC**) and is a public key **PuK** in **ECC**

Elliptic Curve Cryptosystem - ECC

ElGamal Cryptosystem (CS)	Elliptic Curve Cryptosystem (ECC)
PP=(strongprime ***p***, generator ***g***);	**PP**=(**EC secp256k1**; **BasePoint** or **Generator** ***G***; prime ***p***; param. ***a***, ***b***);

p=255996887; g=22; **Cyclic group**: Z_p*={1, 2, 3, …, p-1}; •mod p, :mod p.	Parameters a, b defines **EC** equation $y^2=x^3+ax+b$ **mod** p over **finite field** F_p={0, 1, 2, 3, …, p-1}; +mod p, −mod p, •mod p, :mod p, (except division by 0)
PrK=x; >> x=randi(p-1).	**PrK$_{ECC}$**=z; >> z=randi(p-1).
PuK=a=g^x mod p.	**PuK$_{ECC}$**=A=z*G.
Alice A: x=1975596; a=210649132;	**Alice** A: z=….; A=(x_A, y_A);

Let us consider abstract EC defined in XOY and expressed by the equation over the field F_p.
In axes X and Y are the points of field F_p.

$$y^2 = x^3 + ax + b \bmod p.$$

EC points are computed by choosing coordinate x and computing coordinate y^2.
To compute coordinate y it is needed to extract root square of y^2.

$$y = \pm\sqrt{y^2} \bmod p.$$

Notice that from y^2 we obtain 2 points in EC, namely y and $-y$ no matter computations are performed with integers **mod** p or with real numbers.
Notice also that since EC is symmetric with respect to x-axis, the points y and $-y$ are symmetric in EC.
Since all arithmetic operations are computed **mod** p then according to the definition of negative points in F_p points y and $-y$ must satisfy the condition

$$y + (-y) = 0 \bmod p.$$

Then evidently

$$y^2 = (-y)^2 \bmod p.$$

For example:
-2 mod 11 = 9
2^2 mod 11 = 4 & 9^2 mod 11 = 4
>> mod(9^2,11)
ans = 4

$-2 \bmod 11 = 9 \bmod 11$
$(+2 + (-2)) \bmod 11 = (2 + 9) \bmod 11 = 11 \bmod 11 = 0$

ECDSA: Signature creation for message M

Signature is formed on the h-value h of Hash function **H()** with input M.
In Bitcoin **H**-function is SHA256 algorithm
In Ethereum **H**-function is keccak-256 algorithm it is an algorithm of the family of SHA-3 standard.

1. h = **H**(M)=SHA256(M);
2. i <-- randi; $|i| \leq$ 256 bits; >> **gcd**(i, p)=1 --> Then there exists unique i^{-1} **mod** p such that $i \bullet i^{-1} = 1$ **mod** p.
3. R = i*G = $R(x_R, y_R)$;
4. r = x_R **mod** p;
5. s = (h + z • r) • i^{-1} mod p; $|s| \leq$ 256 bits; // Since i satisfies the condition that gcd(i,p)=1, then exists i^{-1} mod p.
   // >> i_m1=mulinv(i,p) % in Octave
6. **Sign**(**PrK$_{ECC}$**=z, PP, h) = σ = (r, s)

Signature vrification: Ver(PuK=A, σ, h)
1. Calculate u_1 = h • s^{-1} mod p and u_2 = r • s^{-1} mod p
2. Calculate the Elliptic Curve point V = u_1*G + u_2*A=$V(x_V, y_V)$
3. The signature is valid if R=V; r=x_V=x_R mod p.

ECDSA	ElGamal Signature	Schnorr Signature
h = H(m);	h = H(m);	h = H(m):

ECDSA	ElGamal Signature
h = H(m);	h = H(m);
i ←randi; Compute i^{-1} mod p	i ←randi; gcd(i, p-1)=1 Compute i^{-1} mod (p-1)
R = i*G = i*(x_G, y_G) = (x_R, y_R); r = x_R mod p; |i|≤ 256 bits;	r=g^i mod p;
s=(h+z•r)i^{-1} mod p; |s|≤ 256 bits;	s=(h-x•r)i^{-1} mod (p-1);
s^{-1}=(h+z•r)^{-1}i mod p;	h=xr+is mod (p-1).
Sign(PrK$_{ECC}$=z, h) = (r, s) = **ϭ**;	**Sign**(PrK=x, h) = (r, s) = **ϭ**;
ECDSA Verification	**ElGamal Signature Verification**
Compute u_1=h•s^{-1} mod p and u_2=r•s^{-1} mod p;	Compute: u_1= g^h **mod** p; and u_2= $a^r r^s$ **mod** p
Compute R = u_1*G + u_2*A = (x_R, y_R);	Signature is valid if: u_1= u_2
The signature is valid if r=x_R mod p.	

Schnorr Signature
h = H(m);
i ←randi;
r=g^i mod p;
s=(i+x•h) mod (p-1);
Sign(PrK=x, h) = (r, s) = **ϭ**;
Schnorr Signature Verification
Compute: u_1= g^s **mod** p. and u_2= ra^h **mod** p
Signature is valid if: u_1= u_2