1. ElGamal Cryptosystem

1.Public Parameters generation $PP = (p, g)$.
Generate strong prime number p: >> p=genstrongprime(28) % strong prime of 28 bit length
Number p is a **strong prime** if $p = 2*q + 1$, when q is also prime, e.g. $p = 11$, then $p = 2*5 + 1$, and 5 is also prime.

Strong prime number p: defines the set $Z_p^* = \{1, 2, 3, \ldots, p-1\}$, where multiplication operations **mod p** are defined.
Zp*is an algebraic group where division operations are defined as well.
Find a generator **g** in $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ using condition:
Strong prime $p=2q+1$, where q is prime, then g is a generator of Z_p^* iff
$g^q \neq 1 \bmod p$ and $g^2 \neq 1 \bmod p$.
Declare **Public Parameters** to the network $PP = (p, g)$; p = **268435019**; g = **2**;
$2^{28}-1 = 268435455$

```
>> 2^28-1
ans = 2.6844e+08
>> int64(2^28-1)
ans = 268435455
```

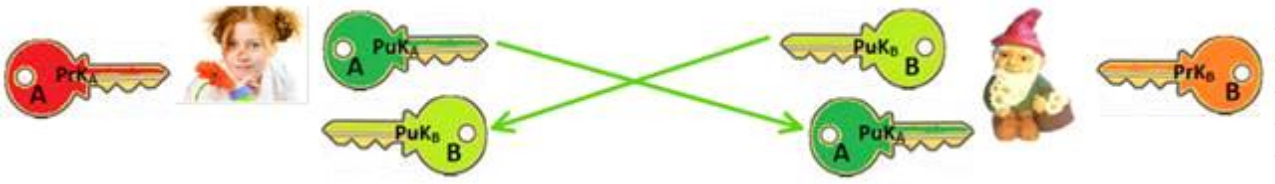
```
>> p=int64(268435019)
p = 268435019
>> q=(p-1)/2
q = 134217509
```

```
>> g=2
g = 2
>> mod_exp(g,q,p)
ans = 268435018     % ans =/= 1 mod p
```

Compatibility relations of modular arithmetic:
(a + b) mod p = (a mod p + b mod p) mod p. >> mod(a+b,p)
(a * b) mod p =((a mod p) * (b mod p)) mod p. >> mod(a*b,p)
$a^e \bmod p = (a \bmod p)^e \bmod p$. **>> mod_exp(a,e,p)**

Public Parameters $PP = (p, g)$.
Strong prime number p in real cryptography is of order : $p \sim 2^{2048} \sim 10^{700}$.
Strong prime number p in our examples is of order: $p \sim 2^{28}$
>> p=genstrongprime(28)
Key generation

- Randomly choose a private key x with $1 < x < p - 1$.
- The private key is **PrK = x = randi(p-1)** Compute $a = g^x \bmod p$.
- The public key is **PuK = $a = g^x \bmod p$**.

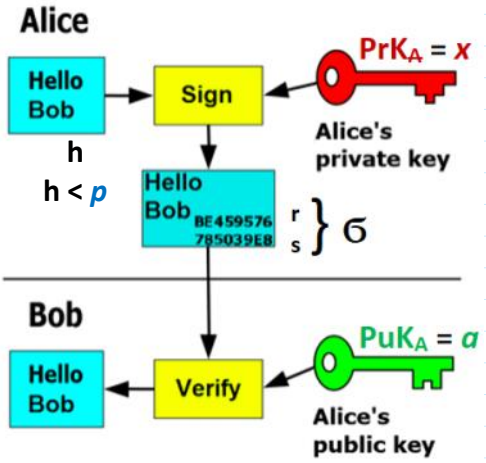


2. El-Gamal E-Signature

The **ElGamal signature scheme** is a digital signature scheme which is based on the difficulty of computing discrete logarithms.
It was invented by Taher ElGamal in 1984. The ElGamal signature algorithm is rarely used in practice.

A variant developed at NSA and known as the Digital Signature Algorithm is much more widely used.
The ElGamal signature scheme allows a third-party to confirm the authenticity of a message sent over an insecure channel.

EC Gamal sign. → Digital Signature Alg. (DSA) NSA
→ Elliptic Curve DSA – ECDSA Certicom

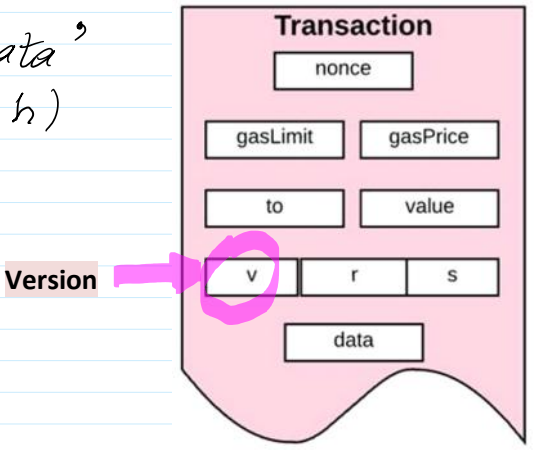


Authenticity

Signature creation for message ***M*** >> ***p***.

1. Compute decimal h-value ***h***=***H***(***M***); ***h***<***p***: SHA-256.
2. Generate >> ***i*** =int64(randi(***p***-1)) % such that **gcd**(***i***, ***p***-1)=**1**.
3. Compute i^{-1} mod (***p***-1). You can use the function
   >> gcd(i, p-1)
   ans = 1
   >> i_m1=mulinv(i, p-1);
1. Compute $r = g^i \bmod p$.
2. Compute $s = (h - xr)i^{-1} \bmod (p-1)$.
3. Signature on h-value ***h*** is σ = (***r***, ***s***)
   Sign(***x***, ***h***) = σ = (***r***, ***s***).

Ethereum



Tx = "nonce || gasLimit || gasPrice || to || value || data"
h = H(Tx) → σ = (r, s) = Sign(PrK, h)

GasLimit is evaluated by Wey.
1 Wey = 10^{-18} Eth --> 1 Eth = 10^{18} Wey.
Usually 1Gas is thousands of Wey's.

1.***Signature creation***
To sign any finite message ***M*** the signer performs the following steps using public parametres **PP** = (***p***, ***g***).

- Compute **h=H(*M*)**. E.g. **SHA-256:** |***h***| = 256 bits. If |***p***| = 2048 bits, then evidently |***h***| << |***p***|.
- Choose a random ***i*** such that **1** < ***i*** < ***p*** **− 1** and **gcd(*i*, *p* − 1) = 1**.
- Compute i^{-1} **mod (p-1)**: i^{-1} **mod (p-1) exists** if **gcd(*i*, *p* − 1) = 1**, i.e. ***i*** and **p-1** are relatively prime.

i^{-1} can be found using either Extended Euclidean algorithmt or Euler theorem or .....
>> i_m1=mulinv(i,p-1) % i^{-1}mod (p-1) computation.

- Compute $r=g^i \bmod p$
- Compute $s=(h-xr)i^{-1} \bmod (p-1)$ --> $h=xr+is \bmod (p-1)$

Signature **σ=(r, s)**

$s = (h - x \cdot r) \cdot i^{-1}$ / i
$s \cdot i = (h - x \cdot r) \cdot i^{-1} \cdot i$
$h - xr = s \cdot i$ → $h = x \cdot r + i \cdot s$ } mod p-1

2.***Signature Verification***

A signature $\sigma=(r, s)$ on a h-value ***h*** of message M is verified using Public Parameters **PP**=(**p**, **g**) and PuK_A=**a**.

1. Bob computes **h=H(M)**.
2. Bob verifies if $1<r<p-1$ and $1<s<p-1$.
3. Bob calculates $V1=g^h \bmod p$ and $V2=a^r r^s \bmod p$, and verifies if **V1=V2.**

The verifier Bob accepts a signature if all **conditions** are satisfied during the signature creation and rejects it otherwise.

3.**Correctness**

The algorithm is correct in the sense that a **signature generated with the signing algorithm will always be accepted by the verifier**.

The signature generation implies

$$h=xr+is \bmod (p-1)$$

Hence Fermat's little theorem implies that all operations in the exponent are computed **mod** (**p**-1)

$$\underbrace{g^h \bmod p}_{V1} = g^{(xr+is) \bmod (p-1)} \bmod p = g^{xr} g^{is} = (g^x)^r (g^i)^s = \underbrace{a^r r^s \bmod p}_{V2}$$

$$(a)^r\ (r)^s = V2$$

```
>> p= int64(268435019)
p = 268435019
>> g=2;
>> x =int64(randi(p-1))
x = 65770603
>> a=mod_exp(g,x,p)
a = 232311991
>> M='Hello Bob...'
M = Hello Bob...
>> h=hd28(M)
h = 150954921
```

```
>> i =int64(randi(p-1))
i = 201156232
>> gcd(i,p-1)
ans = 2
>> i =int64(randi(p-1))
i = 35395315
>> gcd(i,p-1)
ans = 1
>> i_m1=mulinv(i,p-1)
i_m1 = 192754179
>> mod(i*i_m1,p-1)
ans = 1
```

```
>> r=mod_exp(g,i,p)
r = 172536234
>> hmxr=mod(h-x*r,p-1)
hmxr = 20262153
>> s=mod(hmxr*i_m1,p-1)
s = 44575091
```

```
>> g_h=mod_exp(g,h,p)
g_h = 241198023
>> V1=g_h
V1 = 241198023

>> a_r=mod_exp(a,r,p)
a_r = 49998673
>> r_s=mod_exp(r,s,p)
r_s = 111993804
>> V2=mod(a_r*r_s,p)
V2 = 241198023
```

1-st Part: Elliptic Curve Cryptography - ECC

Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields.

δ

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.

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

Finite Field denoted by F_p (or rarely Z_p), when: p is prime.

F_p={0, 1, 2, 3, …, p-1}; $+_{\mathbf{mod}\ p}$, $-_{\mathbf{mod}\ p}$, $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$.

Cyclic Group: Z_p^* = {1, 2, 3, …, p-1}; $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$.

For example, if p=11, then one of the generetors is g=2.

p=11
x a=∈

The main function used in cryptography was Discrete Exponent Function - DEF:
DEF(x) = g^x mod p = a, e.g. when p = 11.

x	0	1	2	3	4	5	6	7	8	9	10
2^x mod p	1	2	4	8	5	10	9	7	3	6	1

Discrete Exponent Function - $DEF_g(x)=g^x$ mod p
x is in Z_{p-1}= Z_{10} = {0, 1, 2, …, 9};
DEF(x) is in Z_p^* =Z_{11}^* ={1, 2, 3, …, 10};
DEF: $Z_{p-1} \rightarrow Z_p^*$.
Fermat theorem: if p is prime, then for any z: z^{p-1}=**1** mod p.
If g is a generator in Z_p^* then DEF is 1-to-1 mapping.

$x \in Z_{10}$ $a \in Z_{11}^*$

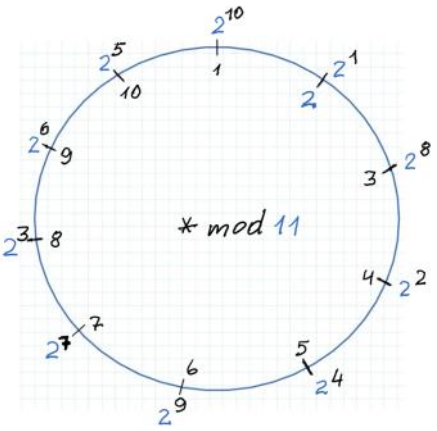
0				1
1				2
2				4
3				8
4				5
5				10
6				9
7				7
8				3
9				6

Multiplicative Group Z_p^*	**Additive Group Z_{p-1}^+**
Z_p^*={1, 2, 3, …, p-1}	Z_{p-1}^+={0, 1, 2, 3, …, p-2}
Operation: multiplication mod p	Operation: addition mod (p-1)
Neutral element is **1**.	Neutral element is **0**.
Generator g: Z_p^*={ g^i; i=0,1,2, …, p-2} Two criterions to find g when p is strong prime. $g^n \neq 1$ mod p if $n<p$.	Generator g: Z_{p-1}^+={$i \bullet g$; i=0,1, 2,…,p-2} E.g. g=1. (p-1)•g=0 mod (p-1) and $n \bullet g \neq 0$ mod (p-1) if 0<n<p-2.
Modular exponent: $t=g^k$ mod p t= $g \bullet g \bullet g \bullet$ …•g mod p; k–times.	Modular multiplication: $t=k \bullet g$ mod p-1 t= $g+g+g+$ …+g mod p-1; k–times.

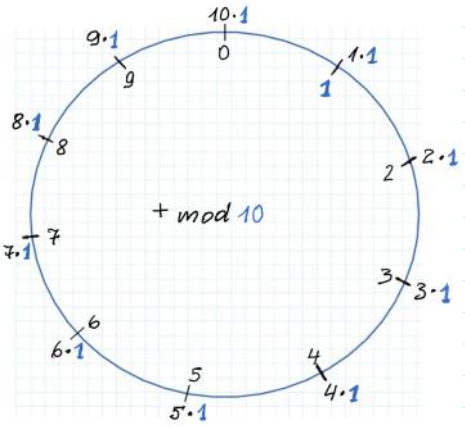
p = 11, p-1 = 10
$\bullet_{\mathbf{mod}\ p}$
Z_{11}^*={1, 2, …, 10}
$|Z_{11}^*|$=10, g=2.

p = 11, p-1 = 10
$+_{\mathbf{mod}\ (p-1)}$
Z_{10}^+={0, 1, 2, …, 9}
$|Z_{10}^+|$=10; g=1.

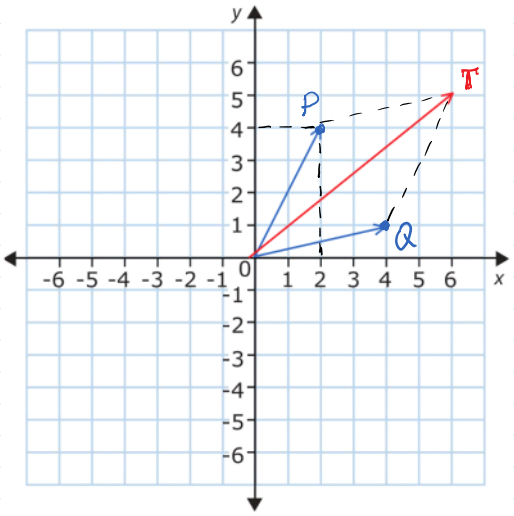
x		2^x
		mod 11
0	→	1
1	→	2
2	→	4
3	→	8
4	→	5
5	→	10
6	→	9
7	→	7
8	→	3
9	→	6
~~10~~	→	1



x		x•1
		mod 10
0	→	1
1	→	2
2	→	3
3	→	4
4	→	5
5	→	6
6	→	7
7	→	8
8	→	9
9	→	0
10	→	1



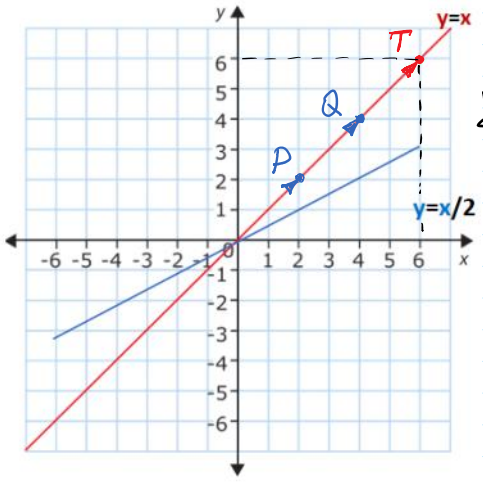
Coordinate systems XOY in subsequent examples are defined in the plane of real numbers.



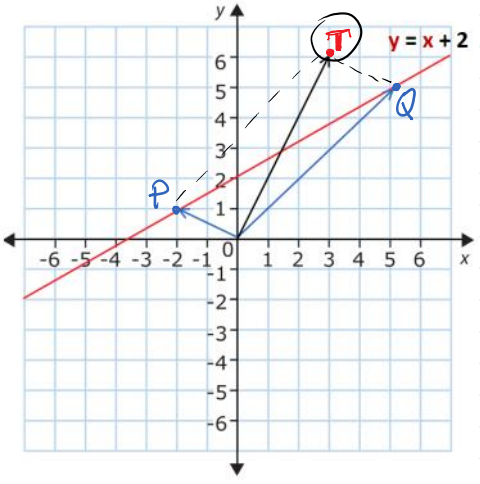
$P(x_P, y_P) = (2,4)$
$Q(x_Q, y_Q) = (4,1)$
$P+Q = (2+4, 4+1)$
$T = P+Q = (6,5)$

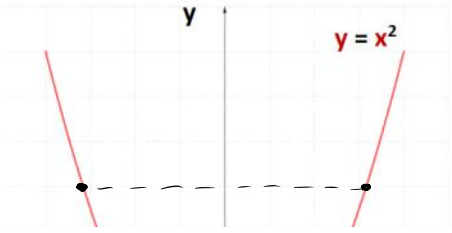
$T = P(x_P, y_P) \boxplus Q(x_Q, y_Q) =$
$= T(x_P + x_Q, y_P + y_Q) = T(x_T, y_T)$

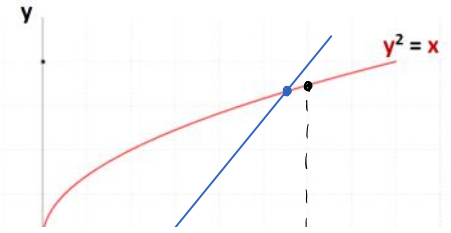
$x_T = x_P + x_Q$
$y_T = y_P + y_Q$
$T_2 = P+P = 2P =$

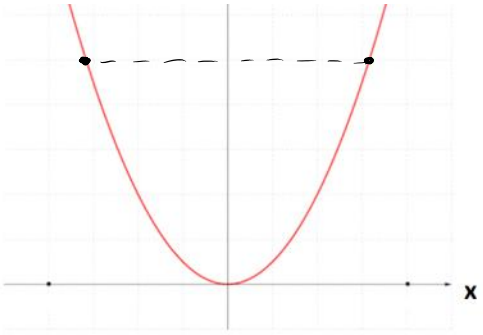


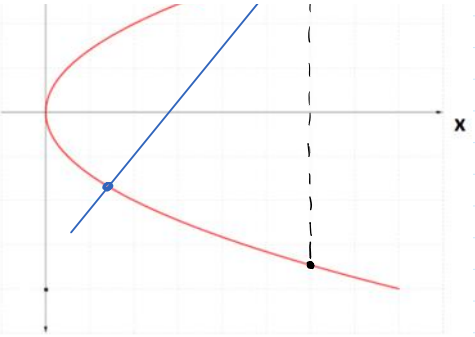
$x_T = 2+4 = 6$
$y_T = 2+4 = 6$
$|T| =$
$= \sqrt{6^2+6^2}$

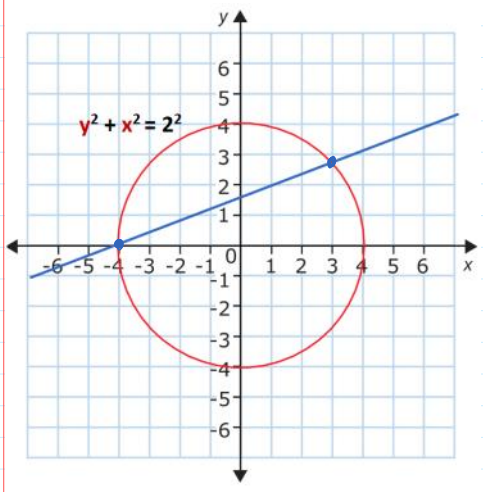


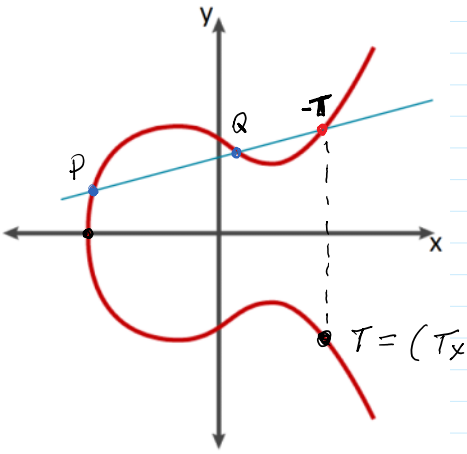












$y^2 = x^3 + ax + b$

Operations are performed in field F_p

$P \boxplus Q = T \in EC$

$T \boxplus (-T) = O$

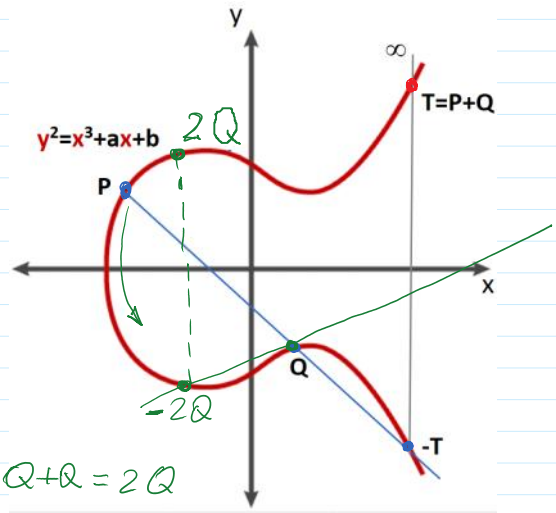
$\infty = \frac{5}{x_2 - x_1}$

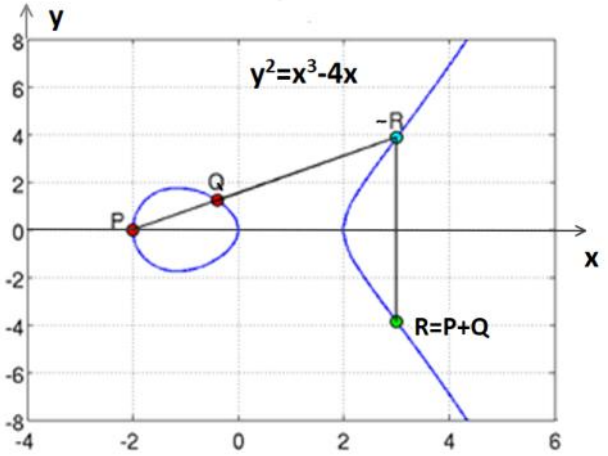
Elliptic curve ha a property that if line crosses two points, then there is a third crossing point in the curve.

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: $P \boxplus Q = T$

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





$5 - 5 \bmod 10 = 0$

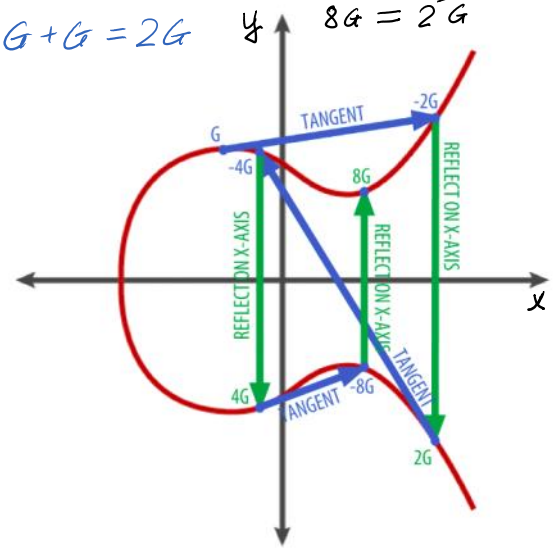
$7 + 0 \bmod 10 = 7$

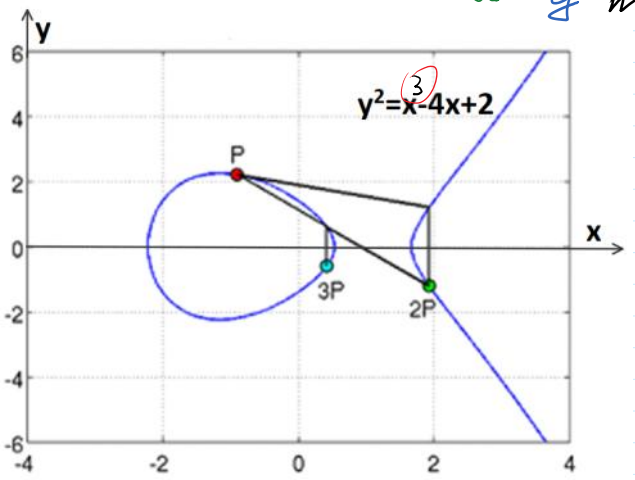
$T - T = „0" \longrightarrow T + (-T) = „0" \equiv \infty$

$T + \infty = T$

When z is large, $z \sim 2^{256} \rightarrow |z| = 256$ bits:
Doubling of points allows effectively compute point $A = zG$
$a = g^x \bmod p$

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





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, …, $\boldsymbol{p}$-1}, (rather than the real numbers) $\boldsymbol{p}$-is prime.
Which consists of the points satisfying the equation over $\boldsymbol{F_p}$

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

along with a distinguished point at infinity, denoted by $\boldsymbol{0}$ (∞).
Finite field is an algebraic structure, where 4 algebraic operations: $+_{\text{mod } p}$, $-_{\text{mod } p}$, $\cdot_{\text{mod } p}$, $:_{\text{mod } p}$ are defined except the division by 0 excluded.

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

EC Homomorphism

$ECHom(G, x) = x * G = G \boxplus G \boxplus G \boxplus \dots, \boxplus G$
x-times

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.

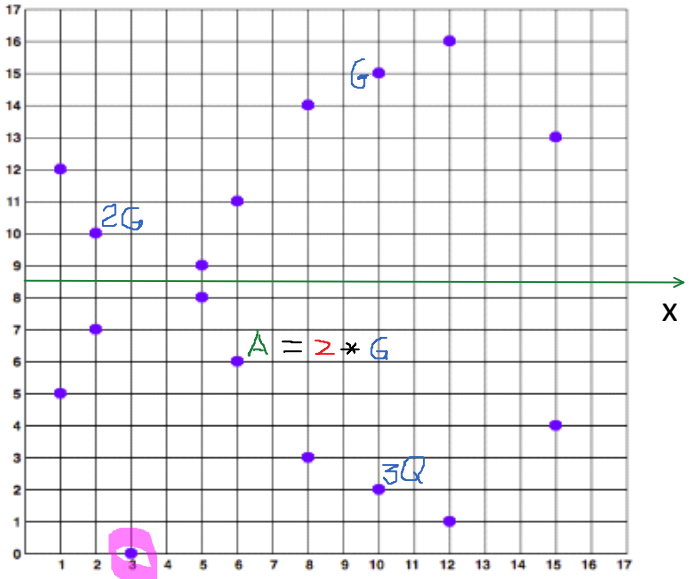


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

$DEF: g^x \bmod p = a;$ $DEF(x+y) \bmod p = DEF(x) \cdot DEF(y) \bmod p$

$g^{x+y} \bmod p = g^x \cdot g^y \bmod p$

$ECDEF: x * G = A = (x_A, y_A);$ (x+y)*G = x*G + y*G.

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 Elliptic Curve Cryptography - ECC

$ECDEF((\underbrace{x+y}_{z}) * G) = \underbrace{ECDEF(x*G)}_{P} \boxplus \underbrace{ECDEF(y*G)}_{Q} = T$

$P \boxplus Q = T$

ElGamal Cryptosystem (CS)	Elliptic Curve Cryptosystem (CS)
PP=(strongprime ***p***, generator ***g***); ***p***=255996887; ***g***=22;	**PP**=(EC **secp256k1**; BasePoint-Generator ***G***; prime ***p***; param. ***a***, ***b***); Parameters ***a***, ***b*** defines EC equation $y^2=x^3+ax+b \bmod p$ over F_p.
PrK=***x***; >> **x**=randi(***p***-1).	**PrK**$_{ECC}$=**z**; >> **z**=randi(***p***-1).
PuK=$a=g^x \bmod p$.	**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:

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

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

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

Then evidently

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

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

$-2 \bmod 11 = 9 \bmod 11$

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

2-nd Part: Elliptic Curve Digital Signature Algorithm - ECDSA

Signature is formed on the h-value *h* of Hash function H of message *M*

In Bitcoin H-function is SHA256 algorithm
In Ethereum H-function is keccak-256 algorithm it is an algorithm of the fakily of SHA-3 standard.

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