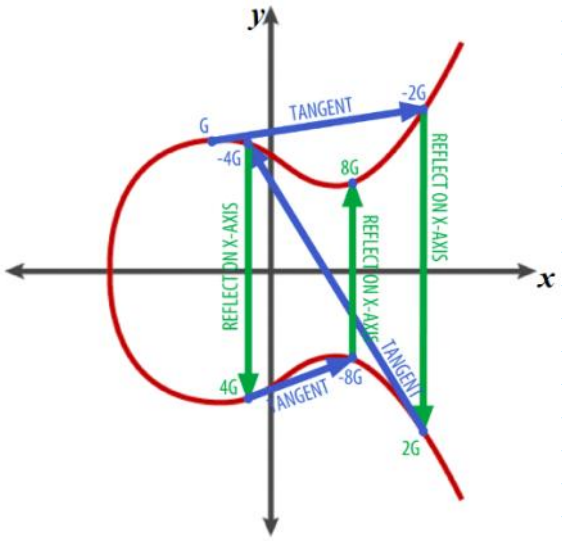


EC: $y^2=x^3+ax+b \mod p$
Let we computed $\mathbf{PuK_{ECC}}=A=(x_A, y_A)=8G$.
Then $(y_A)^2 = (x_A)^3+a(x_A)+b \mod p$ is computed.
By extracting square root from $(y_A)^2$ we obtain 2 points:
$8G$ and $-8G$ with coordinates $(\mathbf{x_A}, \mathbf{y_A})$ and($\mathbf{x_A}$, $-\mathbf{y_A}$).
According to the property of arithmetics of integers **mod** p either $\mathbf{y_A}$ is **even** and $-\mathbf{y_A}$ is **odd** or $\mathbf{y_A}$ is **odd** and $-\mathbf{y_A}$ is **even**.
The reason is that $\mathbf{y_A}+(-\mathbf{y_A})=0 \mod p$ as in the example above when p=11 and that there is a symmetry of EC with respect to x axis..
Then we can compress $\mathbf{PuK_{ECC}}$ representation with 2 coordinates (x_A, y_A) by representing it with 1 coordinate x_A and adding prefix either **02** if y_A is even or **03** if y_A is odd.

Let us consider abstract EC defined in XOY and expressed by the equation:

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

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.

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

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

$F_p = \{0, 1, 2, \ldots, p-1\}$
$* \bmod p$; $+ \bmod p$

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

Then evidently

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

For example:
$2 + (-2) \bmod 11 = 2 + 9 \bmod 11 = 11 \bmod 11 = 0$
$-2 \bmod 11 = 9$
$2^2 \bmod 11 = 4$ & $9^2 \bmod 11 = 4$
>> mod(9^2,11)
ans = 4

Elliptic Curve Group (ECG)
Number of points **N** of Elliptic Curve with coordinates (x, y) is an order of ECG.
Addition operation $\boxplus$ 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: $T=z*G$; $T=G \boxplus G \boxplus G \boxplus \ldots \boxplus G$; z–times.
Generator–Base Point or Generator G: ECG={ $i*G$; i=1,2,…,**N**}; $N*G$=0 and $q*G \neq 0$ if $q<N$.

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**=**.....**; $A=(x_A, y_A)$;

ECDSA:

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.

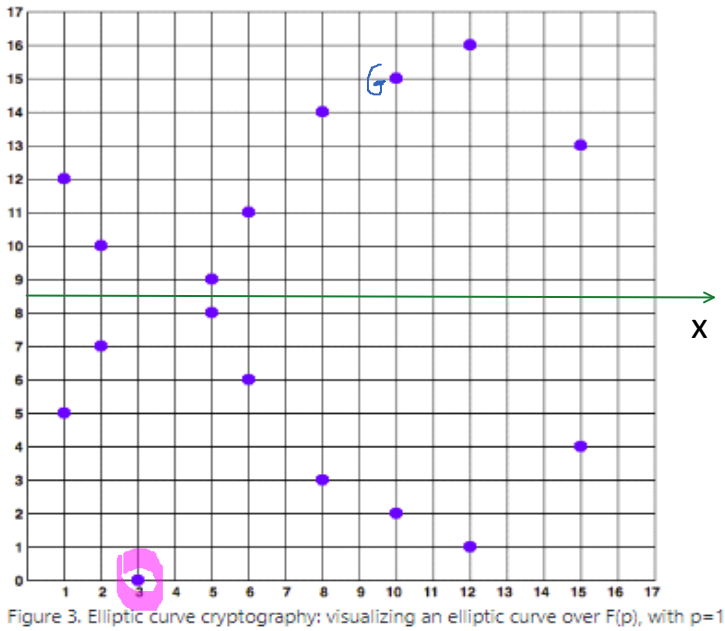


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

v, r, s Ethereum signature

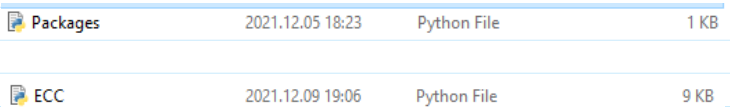
Key generation

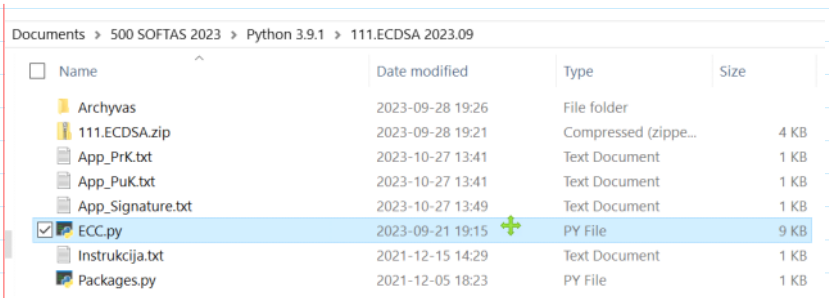
1.Install Python 3.9.1.

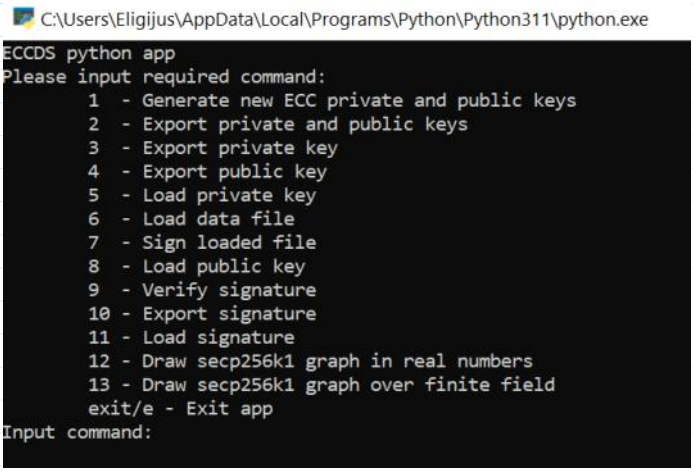
2.Launch script Packages for joining a libraries.

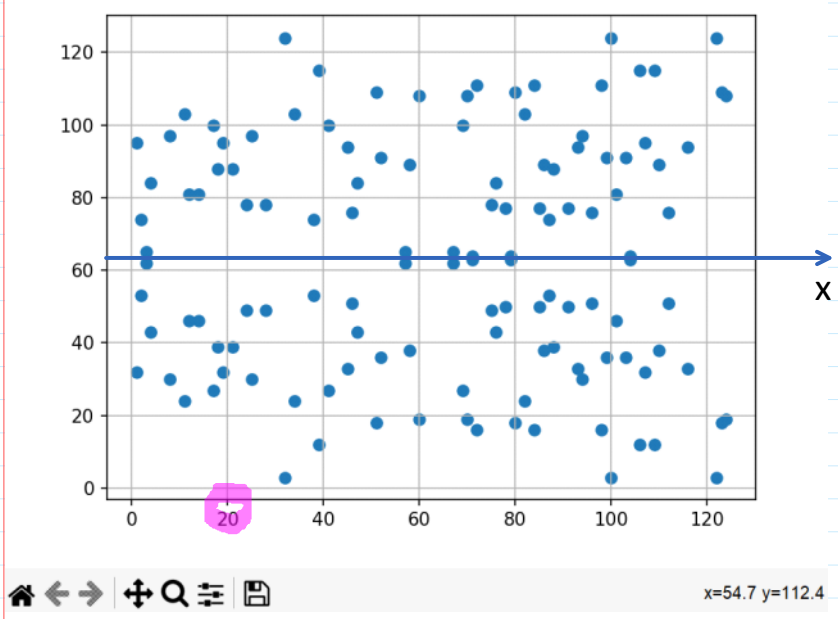
3.Launch file ECC.

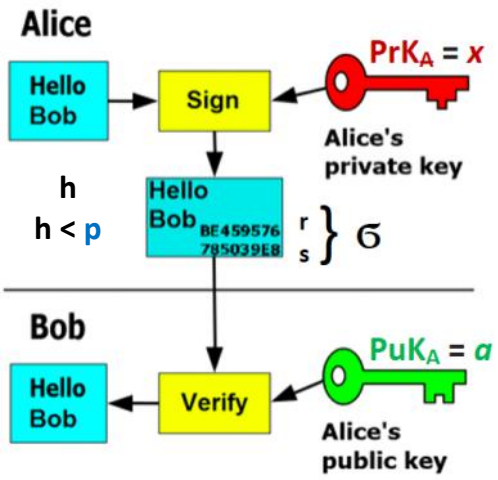
4.If window is escaping, then open hiden windows in icon near the Start icon.











ECDSA: Signature creation for message ***M***

Signature is formed on the h-value ***h*** of Hash function **H** of ***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. $h = H(M)$=SHA256(M);
2. i <-- randi; $|i| \leq 256$ bits; >> **gcd**(i,p)=1 --> Then there exists unique i^{-1} ***mod*** p such that $ii^{-1} = 1$ ***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; // Computed s should satisfy the condition that **gcd**(s,p)=1, then exists
   // unique s^{-1} **mod** p.
   // >> s_m1=mulinv(s,p) % in Octave
1. **Sign**(PrK_{ECC}=z, PP, h) = $\sigma = (r, s)$

Signature vrification: Ver(PuK=*A*, σ, *h*)

1. Calculate $u_1 = h \bullet s^{-1} \bmod p$ and $u_2 = r \bullet s^{-1} \bmod p$
2. Calculate the curve point $V = u_1*G + u_2*A = V(x_V, y_V)$

3. The signature is valid if $R=V$; $r=x_V=x_R \bmod p$.

ECDSA	ElGamal Signature	Schnorr Signature
$h = H(M)$;	$h = H(M)$;	$h = H(M)$;
$i \leftarrow$randi; Compute $i^{-1} \bmod p$	$i \leftarrow$randi; gcd(i, p-1)=1 Compute $i^{-1} \bmod (p-1)$	$i \leftarrow$randi;
$R = i*G = i*(x_G, y_G) = (x_R, y_R)$; $r = x_R \bmod p$; $\lvert i\rvert \leq 256$ bits;	$r=g^i \bmod p$;	$r=g^i \bmod p$;
$s=(h+z \bullet r)i^{-1} \bmod p$; $\lvert s\rvert \leq 256$ bits;	$s=(h-x \bullet r)i^{-1} \bmod (p-1)$;	$s=(i+x \bullet h) \bmod (p-1)$;
$s^{-1}=(h+z \bullet r)^{-1}i \bmod p$;	$h=xr+is \bmod (p-1)$.	
Sign($PrK_{ECC}=z$, h) = (r, s) = σ;	**Sign**($PrK=x$, h) = (r, s) = σ;	**Sign**($PrK=x$, h) = (r, s) = σ;
ECDSA Verification	**ElGamal Signature Verification**	**Schnorr Signature Verification**
Compute $u_1=h \bullet s^{-1} \bmod p$ and $u_2=r \bullet s^{-1} \bmod p$;	Compute: $u_1= g^h \bmod p$; and $u_2= a^r r^s \bmod p$	Compute: $u_1= g^s \bmod p$. and $u_2= ra^h \bmod p$
Compute $R = u_1*G + u_2*A = (x_R, y_R)$;	Signature is valid if: $u_1= u_2$	Signature is valid if: $u_1= u_2$
The signature is valid if $r=x_R \bmod p$.		

Let u, v are integers $< p$.
Property 1: $(u + v)*P = u*P \boxplus v*P$ replacement to --> $\underline{(u + v)P = uP + vP}$
Property 2: $(u)*(P \boxplus Q) = u*P \boxplus u*Q$ replacement to --> $u(P + Q) = uP + uQ$

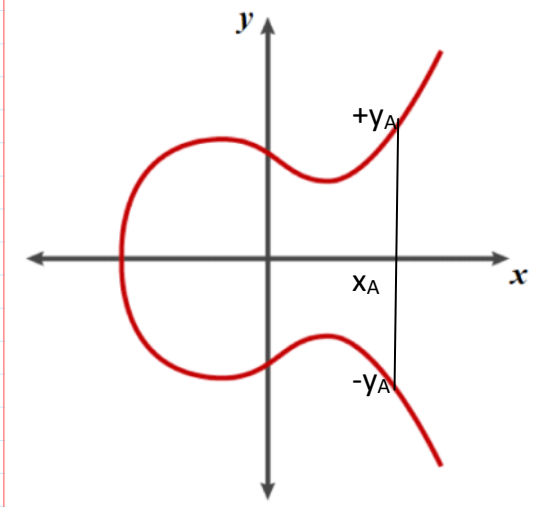
Important identity used e.g. in Ring Signature:
$(t-zc)*G+c*A = t*G-zc*G+c*A = t*G-c(z*G)+c*A = t*G-c*A+c*A = tG \bmod p$.

Correctness:
$R=u_1*G + u_2*A$
From the definition of the Public Key $A=z*G$ we have:
$R=u_1*G + (u_2 \bullet z)*G$
Because EC scalar multiplication distributes over addition we have:
$R=(u_1 + u_2 \bullet z)*G$
Expanding the definition of u_1 and u_2 from verification steps we have:
$R=(h \bullet s^{-1} + r \bullet s^{-1} \bullet z)*G$
Collecting the common term s^{-1} we have:
$R=[(h + r \bullet z) \bullet s^{-1}]*G$
Expanding the definition of s from signature creation we have:
$R=[(h + r \bullet z) \bullet (h + r \bullet z)^{-1} \bullet i]*G=i*G$.
Since the inverse of an inverse is the original element, and the product of an element's inverse and the element is the identity, we are left with $R = i*G = (x_R, y_R)$; $r=x_R$.

$PrK_{ECC}=z < n < 2^{256}$; $PuK_{ECC}=A=(a_x, a_y)$;
$\lvert PrK_{ECC}=z\rvert$=256 bits; $\lvert PuK_{ECC}=A\rvert$=512 bits.

The positive and negative coordinates y and $-y$ in EC in the real numbers plane XOY are presented in Fig.
The positive and negative numbers for p=11 are presented in table .

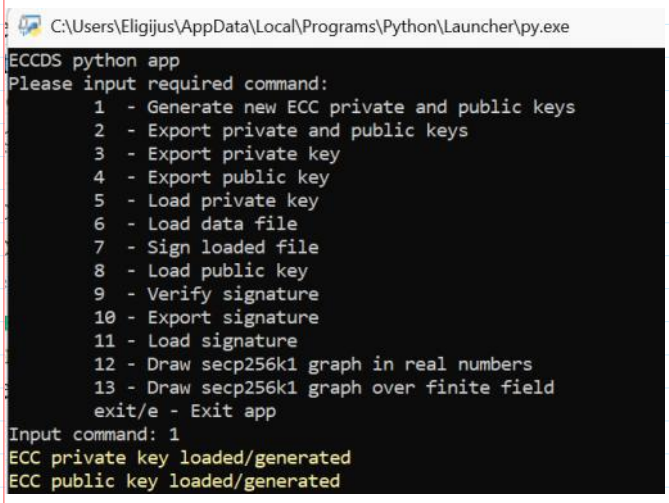
$y^2=x^3+ax+b \bmod p$ over F_p.

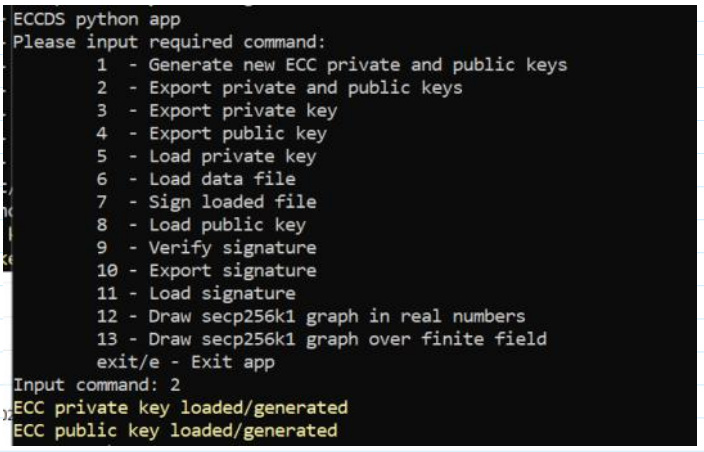


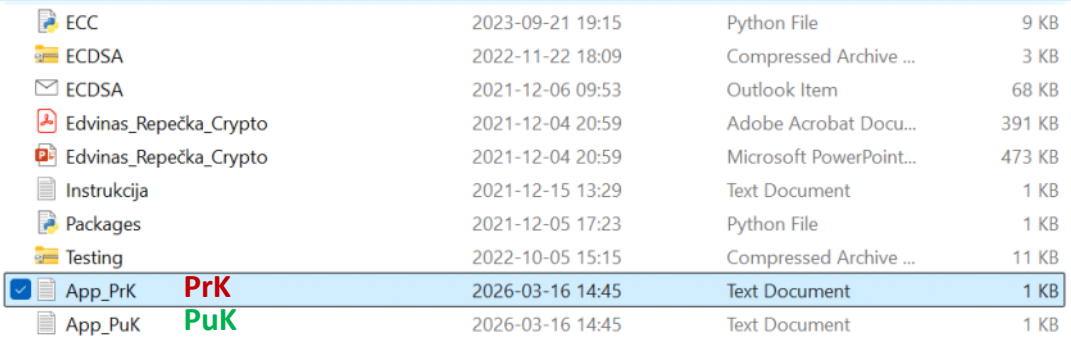
$(y^2)^{1/2} = +-y$

y mod 11			$(-y)$ mod 11
1	odd	even	-1=10
2	even	odd	-2=9
3	odd	even	-3=8
4	even	odd	-4=7
5	odd	even	-5=6
6	even	odd	-6=5
7	odd	even	-7=4
8	even	odd	-8=3
9	odd	even	-9=2
10	even	odd	-10=1

Notice that performing operations **mod** p if y is odd then $-y$ is even and vice versa.
This property allows us to reduce bit representation of $\mathbf{PuK_{ECC}}=A=z*G=(x_A, y_A)$;
In normal representation of $\mathbf{PuK_{ECC}}$ it is needed to store 2 coordinates (x_A, y_A) every of them having 256 bits. For $\mathbf{PuK_{ECC}}$ it is required to assign 512 bits in total.
Instead of that we can store only x_A coordinate with an additional information either coordinate y_A is odd or even.
The even coordinate y_A is encoded by prefix **02** and odd coordinate y_A is encoded by prefix **03**.
It is a compressed form of $\mathbf{PuK_{ECC}}$.
If $\mathbf{PuK_{ECC}}$ is presented in uncompressed form than it is encoded by prefix **04**.
Imagine, for example, that having generator G we are computing $\mathbf{PuK_{ECC}}=A=z*G=(x_A, y_A)$ when z=8.
Please ignore that after this explanation since it is crasy to use such a small z. It is a gift for adversary
To provide a search procedure.
Then $\mathbf{PuK_{ECC}}$ is represented by point $8G$ as depicted in Fig. So we obtain a concrete point in EC being either even or odd.
The coordinate y_A of this point can be computed by having only coordinate x_A using formulas presented above and having prefix either **02** or **03**.







PrK:
aa2d434a558a1b9d02b7494ca88bb8ff262138497fb0facbbb920f7b389a81c0

PuK:
PuK_x:
9154bfcc4aef7692b2ecd5ef1c5177826ff95cacce84b9cb1bd82fb9c1e07b6639
Puk_y:
733792bbd72655b6dd91a3add7e2edb069fec9c68150af9c5d1bb8d8d6de29

Prefix **03** to coordinate PuK_x is added since coordinate PuK_y is an odd numbey ending with 9

Therefore **Puk** can be represented by single coordinate PuK_x with prefix **03**:
039154bfcc4aef7692b2ecd5ef1c5177826ff95cacce84b9cb1bd82fb9c1e07b6639