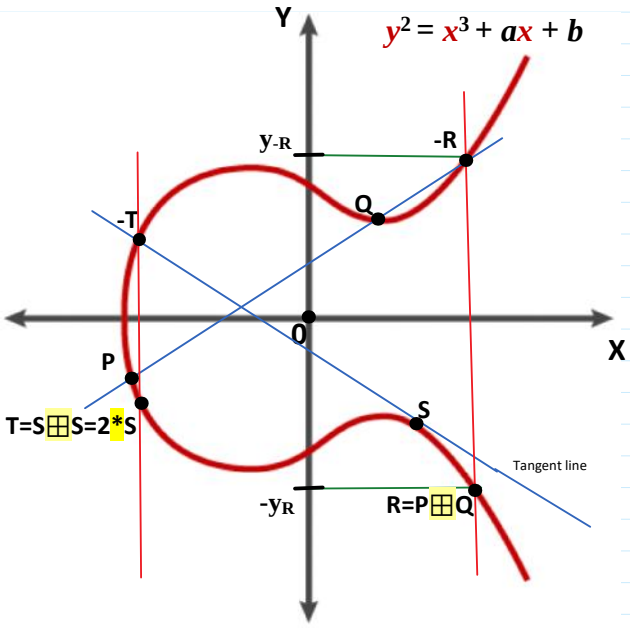
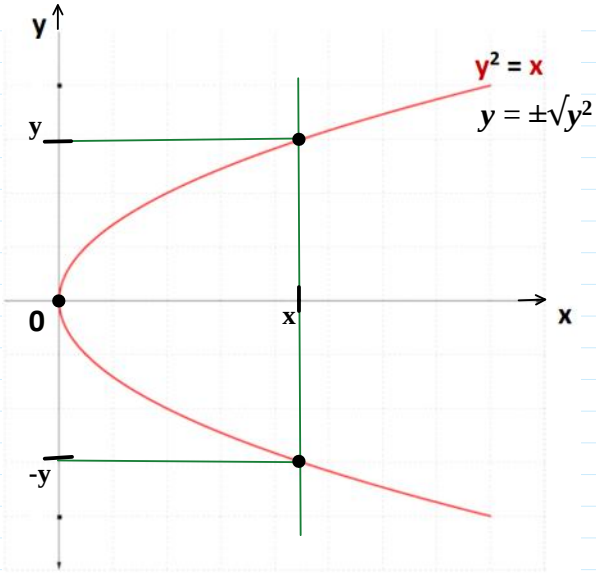




Elliptic Curve Cryptography keys generation

ECC Public Parameters: PP = (*EC*, *G*, *p*), *G*=(x_G, y_G); **ElGamal CS Public Parameters: PP = (*p*, *g*)**

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

G is a generator or base point of **EC**.

n - is an order (number of points) of **EC**, i.e. according to **secp256k1** standard is equal to ***p***: **n**=***p***; |**n**|=|***p***|=256 bits: 1<x_G<**n**, 1<y_G<**n**.

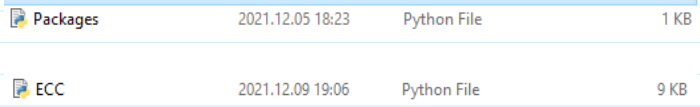
PrK$_A$=z <-- randi; **z**< **n**, max|**z**|<=256 bits.

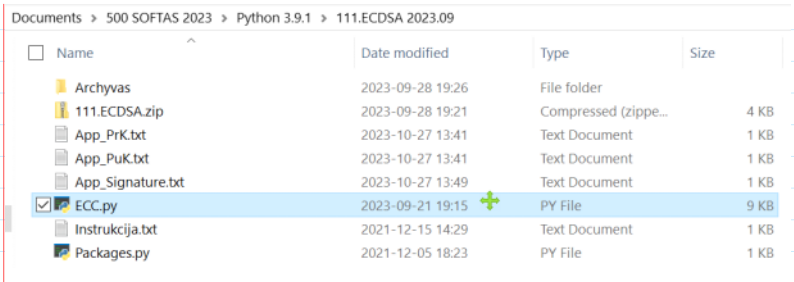
PuK$_A$ = **z***G = A(x_A, y_A); max|A|=2•256=512 bits.

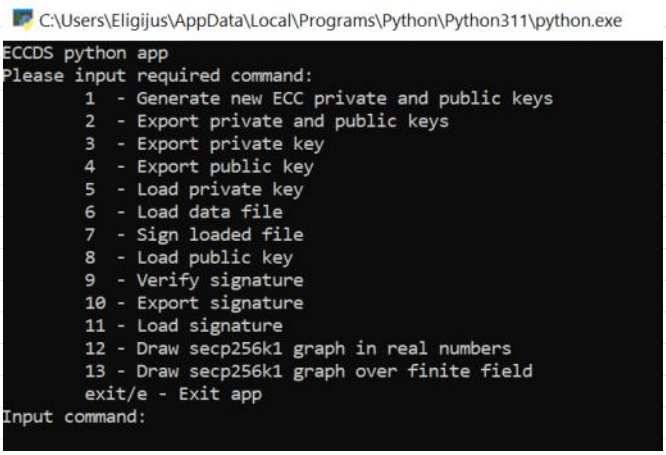
Security consideration

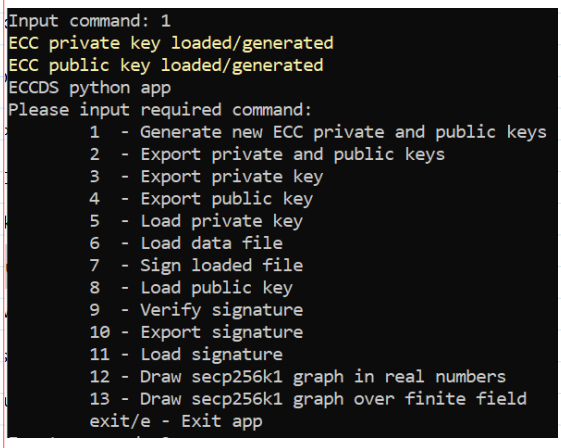
Let **PrK$_A$=z** and **PuK$_A$=z***G then it is infeasible to find **z** from the equation **PuK$_A$=z***G when **PuK$_A$** and G are given.

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.

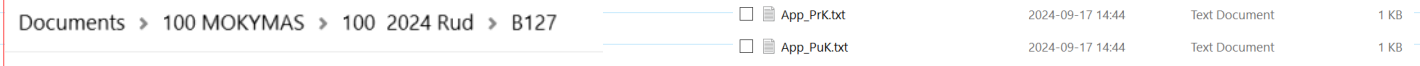








Remark: 2 - Export private and public keys -->
--> [Select folder] means that folder must be selected, not opened.



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

PrK = z It is a random generated number of 256 bit length less then p.
1099b9f87df15f7f27636629a863d2b0c327c50e18846f41d2bc06115ede8116

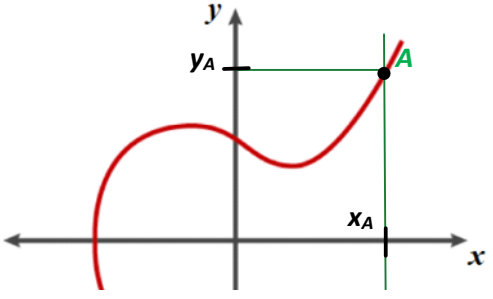
PuK = $A = z*G$ It is a an elliptic curve point with coordinates (x_A, y_A).
$A(x_A, y_A)$ is obtained by z-times adding generator (base point G).
71851cc3933a97ac8a4d5d2b893f6e1f10ad9c149bb34f3f2c00ca3c169f5b129 x_A
8d0140ec22f7f7b6fdc6b7bb825336294116dd4c192f48308e05152114837f y_A

15

256 bits length or a little less

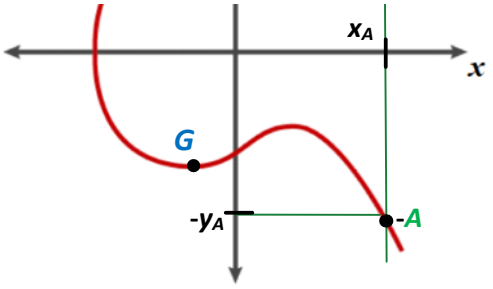
PuK = $A(x_A, y_A)$ has 512 bits length since it represents two coordinates (x_A, y_A) both having 256 bit length

Compressed form:
PuK = A have its symmetric point $-A$ with the same coordinate x_A.
Take in mind that EC coordinates are computed **mod p**.
If coordinate y_A is odd number, then coordinate is an even number.
And vice versa.
It can be seen from the example below when p=11.



y mod 11			(-y) mod 11
1	odd	even	-1=10

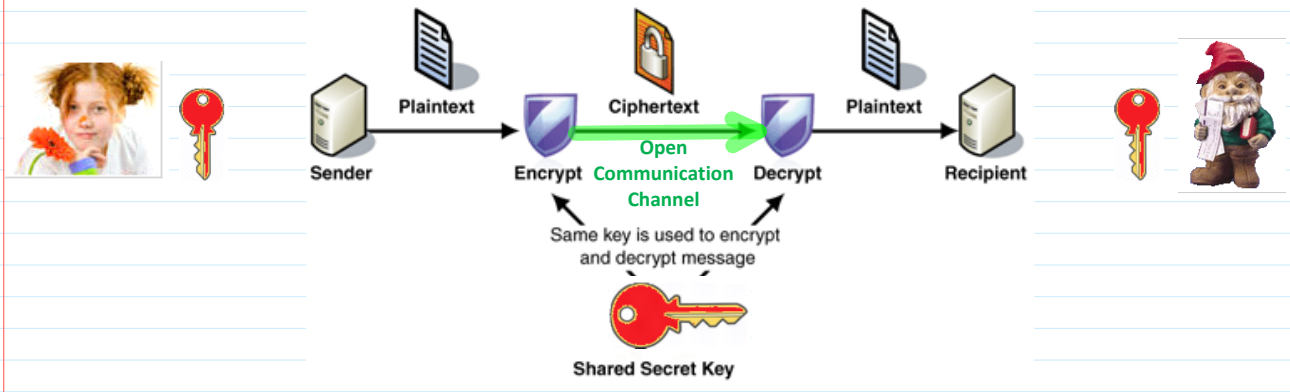
y mod 11			(-y) mod 11
1	odd	even	-1=10
2	even	odd	-2=9
3	odd	even	-3=8
4	even	odd	-4=7
5	odd	even	-5=6
6	even	odd	-6=5
7	odd	even	-7=4
8	even	odd	-8=3
9	odd	even	-9=2
10	even	odd	-10=1



It allows to reduce the **PuK** = ***A*** representation almost twice.
The even coordinate y_A is encoded by prefix 02.
The odd coordinate y_A is encoded by prefix 03.
If **PuK** is presented in uncompressed form than it is encoded by prefix 04.

We see that in example above coordinate y_A is odd
8d0140ec22f7f7b6fdc6b7bb825336294116dd4c192f48308e05152114837f
Then **PuK** is represented by coordinate x_A with prefix 03 in the following way:
0371851cc3933a97ac8a4d5d2b893f6e1f10ad9c149bb34f3f2c00ca3c169f5b129

To perform the computations in EC the algorithmmust performs the following steps:
1.Take x_A and put it in equation $y^2 = x^3 + ax + b \bmod p$ to obtain y^2
2.Extract square root from y^2 to obtain two coordinate values y_A and $-y_A$.
3.If Prefix is 02 then take an even coordinate, e.g. y_A, otherwice take an odd coordinate.



Diffie-Hellman Key Agreement Protocol (DH KAP)

Public Parameters PP=(p,g)

Animation

$u \leftarrow rand(Z_p^*)$

$g^u \bmod p = t_A$

t_B

t_A

$v \leftarrow rand(Z_p^*)$

$t_B = g^v \bmod p$

$k_{AB} = (t_B)^u \bmod p =$

$= (g^v)^u \bmod p = g^{vu} \bmod p$

$k_{BA} = (t_A)^v \bmod p =$

$= (g^u)^v \bmod = g^{uv} \bmod p$

$k_{AB} = k = k_{BA}$

EC-DH-Key Agreement Protocol EC-DH-KAP

$\mathcal{I}_p = \{0, 1, 2, \ldots, p-1\}$; $p \sim 2^{256} \approx 10^{80}$.

A: $u \leftarrow rand(\mathcal{I}_p)$

$u * G = T_A$

t_A →

← t_B

B: $v \leftarrow rand(\mathcal{I}_p)$

$T_B = v * G$

$K_{AB} = u * T_B = u * v * G$

$K_{BA} = v * T_A = v * u * G$

$(u \cdot v) * G = K_{AB} = K = K_{BA} = (v \cdot u) * G$

Signature creation for message ***M*** using **ECDSA**
Public Parameters: **PP** = (***EC***, ***G***, ***p***), ***G***=(x_G, y_G);

PrK$_A$=z <-- randi; **z**< **n**, max|**z**|<=256 bits.
PuK$_A$=z*G=***A**=(x_A, y_A); max|***A***|=2•256=512 bits.

Signature is formed on the h-value ***h*** of Hash function of ***M***.
Recommended to use SHA256 algorithm

1. ***h*** = H(***M***)=SHA256(***M***);
2. ***i*** <-- randi; |***i***|≤ 256 bits;
3. ***R*** = ***i****G = ***i***(x_G, y_G) = (x_R, y_R);
4. ***r*** = x_R mod ***p***;
5. ***s*** = (***h*** + ***z*** • ***r***) • ***i***$^{-1}$ mod ***p***; |***s***|≤ 256 bits; // Since ***p*** is prime, then exists ***i***$^{-1}$ mod ***p***.
   // >> s_m1=mulinv(s,p) % in Octave
6. **Sign**(**PrK$_{ECC}$=z**, ***h***) = $\mathfrak{G}$ = (***r***, ***s***)

EC-DH-Authenticated Key Agreement Protocol EC-DH-AKAP

A: must sign on session parameter T_A: $|T_A| > 256$ bits

Therefore A must compute h-value on T_A.

Realization: Let we have SHA256 H-function

SHA256: String of finite length → string of 256 bit length

>> h_A = sha256('T_A'); value h is an integer of 256 bit length.

Let ***Alice*** computed the following T_A value:
e0d473945a263cc22970731ba3070472358e514eff1f78464610ad07a952cece
6c08280f3559a79996ad2839143e252ef7b90da5e284cc73cf3d8922741baf91
Then
>> hA=sha256('e0d473945a263cc22970731ba3070472358e514eff1f78464610ad07a952cece6c08280f35
59a79996ad2839143e252ef7b90da5e284cc73cf3d8922741baf91')

hA = 38CC536D27E3890984BD3737B91429701A8463D5A28E2592F4B8B3FCDE5D3E5F
>> length(h)
ans = 64 % length of h-value is 64 hexadecimal numbers, i.e. 256 bits corresponding to function sha256

The signature ***Alice*** is placing on this h-value h_A.

Alice: $PrK_A = z$; $PuK_A = z*G = A$
$PuK_B = B$

Bob: $PrK_B = y$; $PuK_B = y*G = B$
$PuK_A = A$

$u \leftarrow rand(\mathcal{I}_p)$
$T_A = u * G$
$h_A = sha256('T_A')$
$Sign(z, h_A) = \sigma_A = (r_A, s_A)$

$T_A, \sigma_A = (r_A, s_A)$ →

Ver(A, σ_A) = True
$v \leftarrow rand(\mathcal{I}_p)$
$T_B = v * G$
$h_B = sha256('T')$
$Sign(y, h_B) = \sigma_B = (r_B, s_B)$

← $T_B, \sigma_B = (r_B, s_B)$

Ver(B, σ_B) = True

$K_{AB} = u * T_B = u*(v*G) =$
$= (u \cdot v)*G$ == K == $K_{BA} = v * T_A = v*(u*G) =$
$= (v \cdot u) * G$