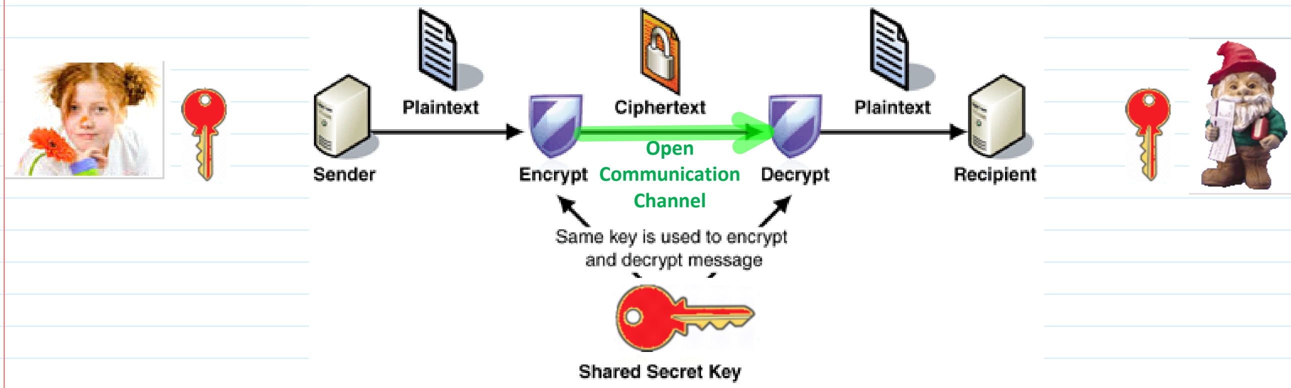
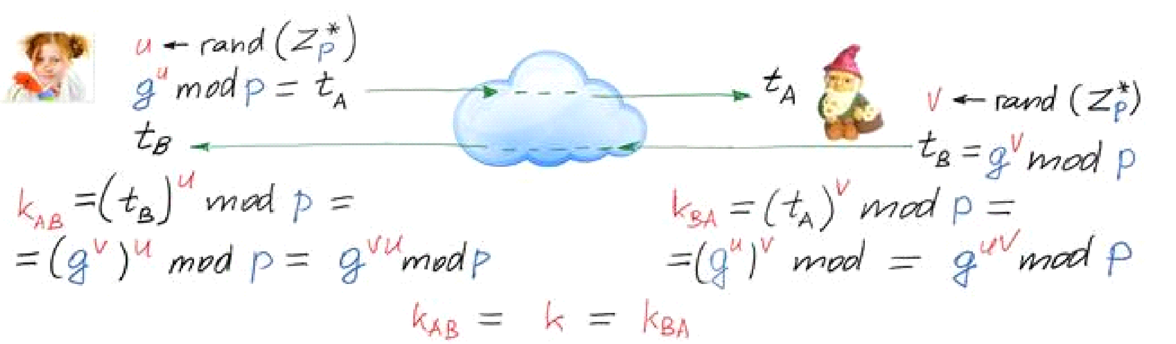


Diffie-Hellman Key Agreement Protocol (DH KAP)

Public Parameters PP=(p,g)

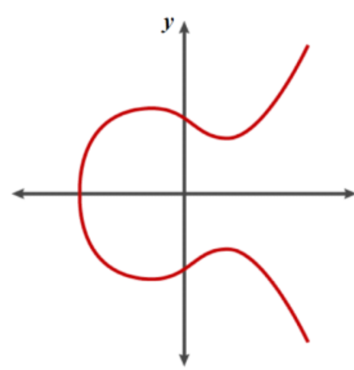


ElGamal Cryptosystem (CS)	Elliptic Curve Cryptosystem (CS)
PP=(strongprime p, generator g); p=255996887; g=22;	PP=(EC secp256k1; BasePoint-Generator G; prime p; param. a, b); Parameters a, b defines EC equation $y^2=x^3+ax+b$ **mod** p over F_p.
PrK=x; >> x=randi(p-1).	PrK_{ECC}=z; >> z=randi(p-1).
PuK=a=g^x mod p.	PuK_{ECC}=A=z∗G.
Alice A: x=1975596; a=210649132;	Alice A: z=…..; A=(x_A, y_A);

Signature creation for message M using **ECDSA**

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 s^{-1} mod p.
   // >> s_m1=mulinv(s,p) % in Octave
6. **Sign**(PrK_{ECC}=z, h) = б = (r, s)

Elliptic Curve - Diffie-Hellman Key Agreement Protocol EC-DH KAP

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.

A: $u \leftarrow randi(p)$

$K_A = u * G$ $\xrightarrow{K_A}$ B: $v \leftarrow randi(p)$

$\xleftarrow{K_B}$ $K_B = v * G$

$K_{AB} = u * (K_B) =$
$= (u \cdot v) * G.$
mod p

$K_{BA} = v * K_A =$
$= (v \cdot u) * G.$
mod p

$$K_{AB} = K = K_{BA}$$

ECC key gen

Authenticated KAP

h-value for **A** computation: $h_A = H(A)$; **A** = (x_A, y_A)

$PrK_B = y$; $PuK_B = B$

$Sign(PrK_A = z, A) = (r_A, s_A) = \sigma_A$

$u \leftarrow randi$

$K_A = u * G$

$Sign(PrK_A = z, K_A) =$
$(r_{KA}, s_{KA}) = \sigma_{KA}$

$\xrightarrow{A, \sigma_A, K_A, \sigma_{KA}}$

B: $Ver(PuK_A = A, \sigma_A, A) \rightarrow Yes$
$Ver(PuK_A = A, \sigma_{KA}, K_A) \rightarrow Yes$

$v \leftarrow randi$

$K_B = v * G$

$Sign(PrK_B = y, B) = (r_B, s_B) = \sigma_B$

$Sign(PrK_B = y, K_B) = (r_{KB}, s_{KB}) = \sigma_{KB}$

$\xleftarrow{B, \sigma_B, K_B, \sigma_{KB}}$

$Ver(PuK_B = y, \sigma_B, B) \rightarrow Yes$
$Ver(PuK_B = y, \sigma_{KB}, K_B) \rightarrow Yes$

$K_{AB} = u * (K_B) =$
$= (u \cdot v) * G.$

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

$$K_{AB} = K = K_{BA}$$

C:\Users\Eligijus\Documents\Zoom\2021-02-18 18.36.03 Eligijus Sakalauskas's Personal Meeting Room 9999112448

Key generation
1.Install Python 3.9.1.
2.Launch script Packages for joining a libraries.
3.Launch file ECC.

Packages	2021.12.05 18:23	Python File	1 KB
ECC	2021.12.09 19:06	Python File	9 KB

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

ECC	2021.12.09 19:06	Python File	9 KB

Documents > 500 SOFTAS 2023 > Python 3.9.1 > 111.ECDSA 2023.09

Name	Date modified	Type	Size
Archyvas	2023-09-28 19:26	File folder	
111.ECDSA.zip	2023-09-28 19:21	Compressed (zippe...	4 KB
App_PrK.txt	2023-10-27 13:41	Text Document	1 KB
App_PuK.txt	2023-10-27 13:41	Text Document	1 KB
App_Signature.txt	2023-10-27 13:49	Text Document	1 KB
ECC.py	2023-09-21 19:15	PY File	9 KB
Instrukcija.txt	2021-12-15 14:29	Text Document	1 KB
Packages.py	2021-12-05 18:23	PY File	1 KB

C:\Users\Eligijus\AppData\Local\Programs\Python\Python311\python.exe

```
ECCDS python app
Please input required command:
        1  - Generate new ECC private and public keys
        2  - Export private and public keys
        3  - Export private key
        4  - Export public key
        5  - Load private key
        6  - Load data file
        7  - Sign loaded file
        8  - Load public key
        9  - Verify signature
        10 - Export signature
        11 - Load signature
        12 - Draw secp256k1 graph in real numbers
        13 - Draw secp256k1 graph over finite field
        exit/e - Exit app
Input command:
```

```
Input command: 1
ECC private key loaded/generated
ECC public key loaded/generated
ECCDS python app
Please input required command:
        1  - Generate new ECC private and public keys
        2  - Export private and public keys
        3  - Export private key
        4  - Export public key
        5  - Load private key
        6  - Load data file
        7  - Sign loaded file
        8  - Load public key
        9  - Verify signature
        10 - Export signature
        11 - Load signature
        12 - Draw secp256k1 graph in real numbers
        13 - Draw secp256k1 graph over finite field
        exit/e - Exit app
```

```
Input command: 2
ECC private key loaded/generated
ECC public key loaded/generated
ECCDS python app
Please input required command:
        1  - Generate new ECC private and public keys
        2  - Export private and public keys
        3  - Export private key
        4  - Export public key
        5  - Load private key
        6  - Load data file
        7  - Sign loaded file
        8  - Load public key
        9  - Verify signature
        10 - Export signature
        11 - Load signature
        12 - Draw secp256k1 graph in real numbers
        13 - Draw secp256k1 graph over finite field
        exit/e - Exit app
```

Documents > 100 MOKYMAS > 100 2024 Rud > B127

App_PrK.txt	2024-09-17 14:44	Text Document	1 KB
App_PuK.txt	2024-09-17 14:44	Text Document	1 KB

PrK

0x1099b9f87df15f7f27636629a863d2b0c327c50e18846f41d2bc06115ede8116

256 bits length or a little less

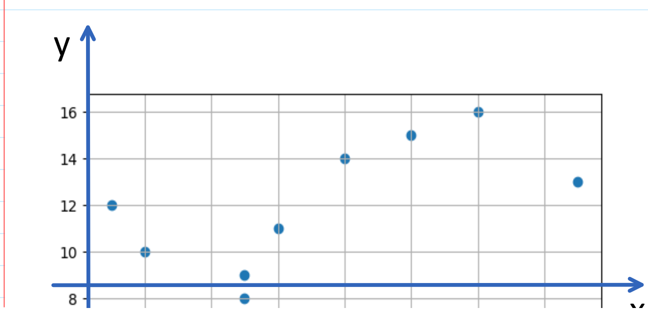
PuK

0x071851cc3933a97ac8a4d5d2b893f6e1f10ad9c149bb34f3f2c00ca3c169f5b1 x

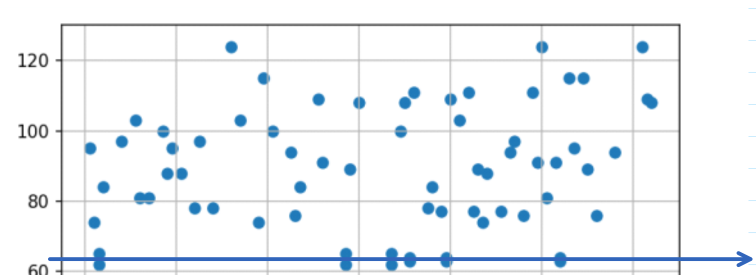
0x298d0140ec22f7f7b6fdc6b7bb825336294116dd4c192f48308e05152114837f y

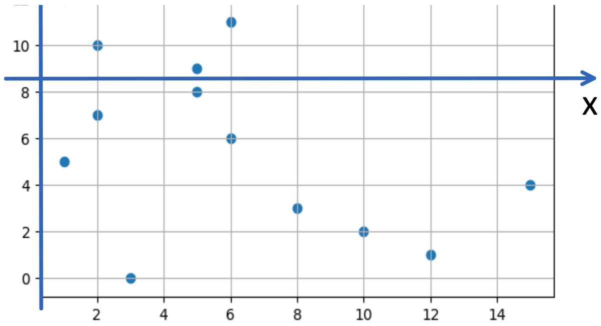
512 bits length or a little less

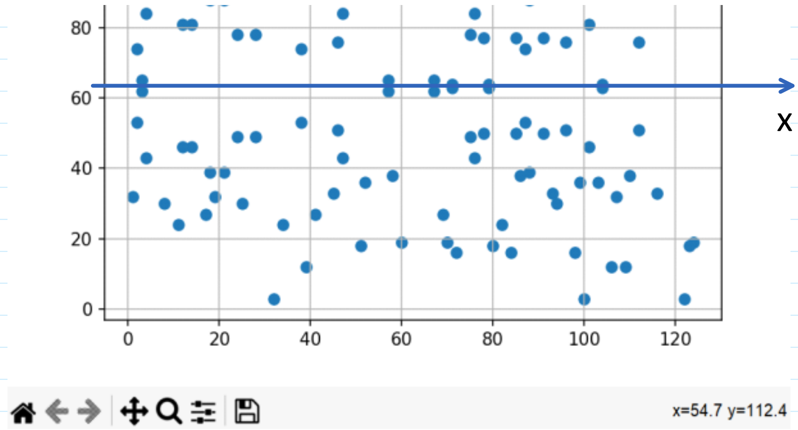
$y^2=x^3+ax+b$ **mod 17; F_p = {0, 1, 2, 3, …, 16}**



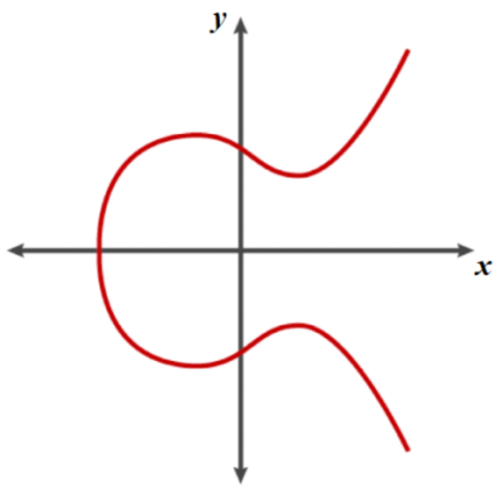
$y^2=x^3+ax+b$ **mod 127**







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 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

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

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

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.

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

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

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

Then evidently

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

For example:
-2 mod 11 = 9
2^2 mod 11 = 4 & 9^2 mod 11 = 4

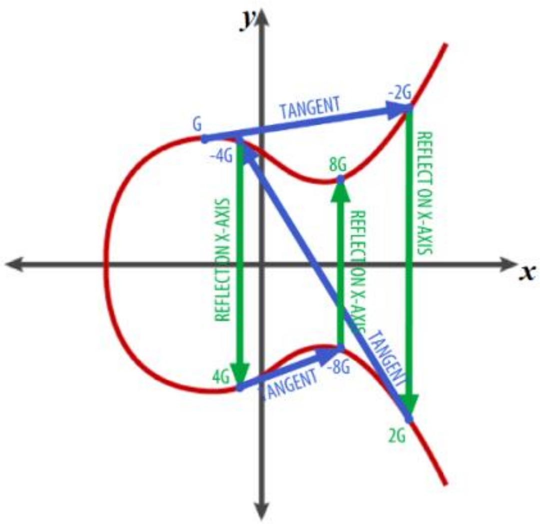
```
>> mod(9^2,11)
ans = 4
```

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 **PuK**$_{ECC}$=A=$z*G$=(x_A, y_A);
In normal representation of **PuK**$_{ECC}$ it is needed to store 2 coordinates (x_A, y_A) every of them having 256 bits.
For **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 **PuK**$_{ECC}$.
If **PuK**$_{ECC}$ is presented in uncompressed form than it is encoded by prefix 04.
Imagine, for example, that having generator G we are computing **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 **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.

EC: $y^2=x^3+ax+b$ **mod** ***p***
Let we computed **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 (x_A, y_A) and(x_A, -y_A).
According to the property of arithmetics of integers **mod** ***p***
either y_A is **even** and -y_A is **odd** or y_A is **odd** and -y_A is **even**.
The reason is that y_A+(-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 **PuK**$_{ECC}$ representation with 2 coordinates (x_A, y_A) by representing it with 1coordinate x_A and adding prefix either **02** if y_A is even or **03** if y_A is odd.

PrK = z:
0x1099b9f87df15f7f27636629a863d2b0c327c50e18846f41d2bc06115ede8116
PuK = A = (x_A, -y_A). Let -y_A is an even. Then coordinate -y_A of EC point A can be omited.
0x071851cc3933a97ac8a4d5d2b893f6e1f10ad9c149bb34f3f2c00ca3c169f5b1 x_A
0x298d0140ec22f7f7b6fdc6b7bb825336294116dd4c192f48308e05152114837f -y_A .

PrK = z:
0x1099b9f87df15f7f27636629a863d2b0c327c50e18846f41d2bc06115ede8116
PuK = A = (x_A, -y_A). Let -y_A is an even. Then coordinate -y_A of EC point A can be omited.
02071851cc3933a97ac8a4d5d2b893f6e1f10ad9c149bb34f3f2c00ca3c169f5b1 x_A



Effective summation of EC points.

For example coputation of **PuK$_{ECC}$**.

PrK = **z**.

>> **z**=randi(***p***-1)   % In real ECC |**z**| is of 256 bit length.

% This means that ~ 2^{256} ~ 10^{80}.

How to compute e.g. **PuK$_{ECC}$**=*A*?

PuK$_{ECC}$=*A*=**z**∗*G* =(x_A, y_A) when **z**.

The solution is points doubling algoriyhm:

For example: by doubling pouts we can 8 times to sum point G

By realizing only 3 doubling: 2^3=8.

In order to sum 4096=2^{12} points it is sufficient to sum $\log_2 2^{12}$ = 12 times.

In order to sum 2^{256} points it is sufficient to sum $\log_2 2^{256}$ = 256 times.