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

EC secp256k1 → $x, y \in \mathcal{Z}_p = \{0, 1, 2, 3, \ldots, p-1\}$

$|x|_{max} = |y|_{max} \leq 256$ bits → $A = (x_A, y_A)$ → $|A| = 256+256 = 512$

Compressed form of PuK = A → $|A_c| = (512 + 2)$ bits.

Recovery from $A_c \rightarrow A$. $Rec(A_c) = A$.

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

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

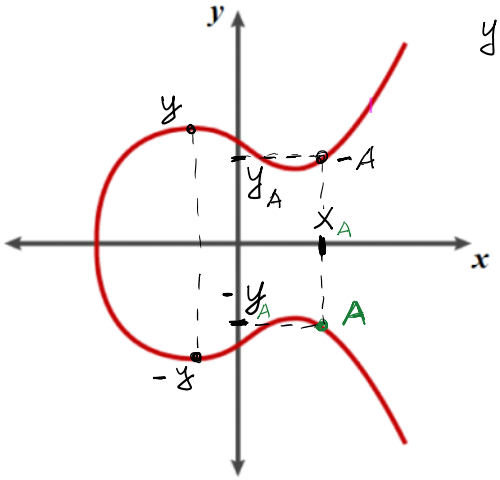
Then evidently

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

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

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

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 .

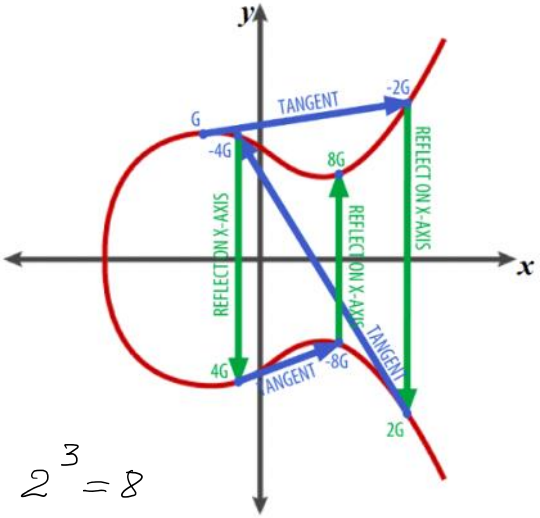


$y-y=0$

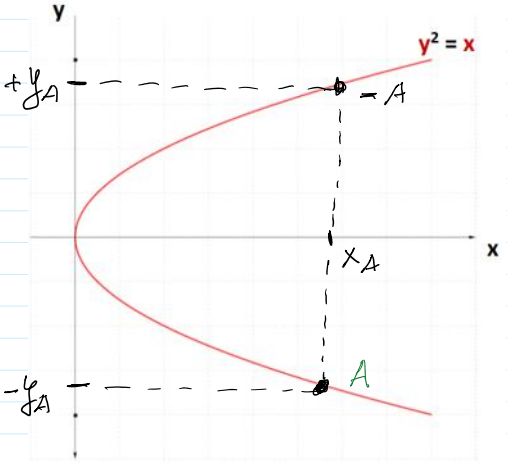
$1-1=0$

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 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. when $|p|=256$ bits.
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 encided 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.
If PuK is presented in then prefix is 04.

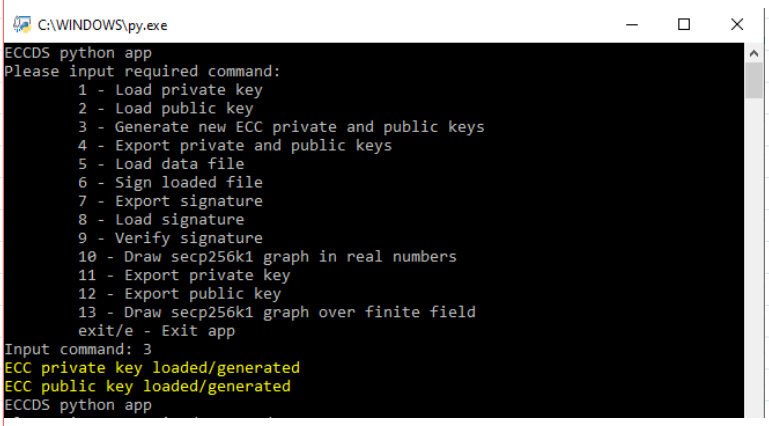


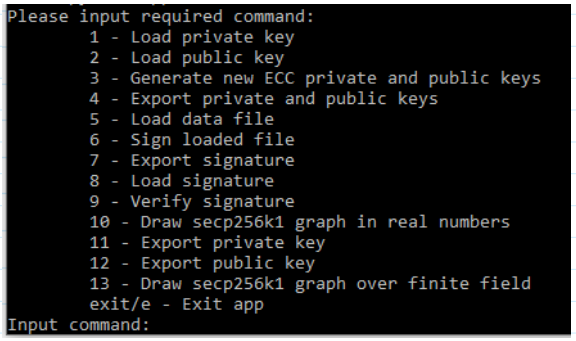
$2^3=8$

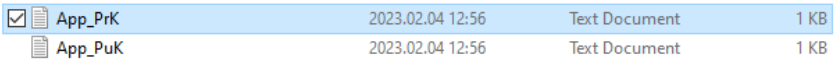


EC: $y^2=x^3+ax+b \mod p$
Let we computed $\mathbf{PuK_{ECC}}=A=(x_A, y_A)=8G$.

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 (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.
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.







$G+G=2G$
$2+G$

686a0f209c1bc05617f8b540afbd5a49651453ee48b472256e6537d2b9684513 PrK = z

fba40304078e170874e204afd87ff6d9f15f0c58b81a60e59e0a3104063c0667 $PuK_x = x_A$
58183c3388ddf828e81e23f1d3265ef69f52d39a837eed9d91e4c2d17d15f8c1 $PuK_y = y_A$

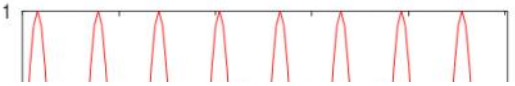
03 fba40304078e170874e204afd87ff6d9f15f0c58b81a60e59e0a3104063c0667

Private Key of EC Cryptosystem (ECC) is $PrK_{ECC}=z$, where z is secret integer generated at random, i.e. $z \leftarrow$ randi.
Public Key of ECC is $PuK_{ECC}=A=z*G=(x_A, y_A)$,
where * means generator G multiplication by integer z or this means z-times addition of point G in EC according to points addition rule defined above in Fig.

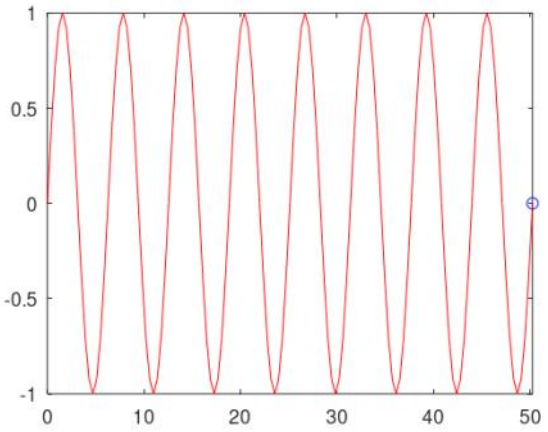
Let u, v are integers $< p$.
Property 1: $(u + v)*P = u*P \boxplus v*P$ replacement to --> $(u + v)P = uP + vP$
Property 2: $(u)*(P \boxplus Q) = u*P \boxplus u*Q$ replacement to --> $u(P + Q) = uP + uQ$

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.

```
>> pi
ans = 3.1416
```

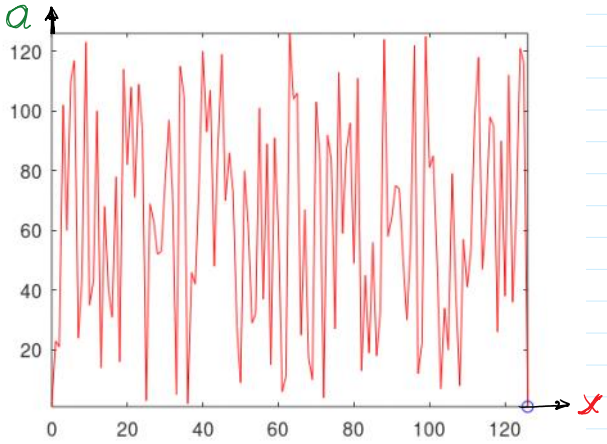


```
>> pi
ans = 3.1416
>> xrange=16*pi
xrange = 50.265
>> step=xrange/128
step = 0.3927
>> x=0:step:xrange;
>> y=sin(x);
>> comet(x,y)
```



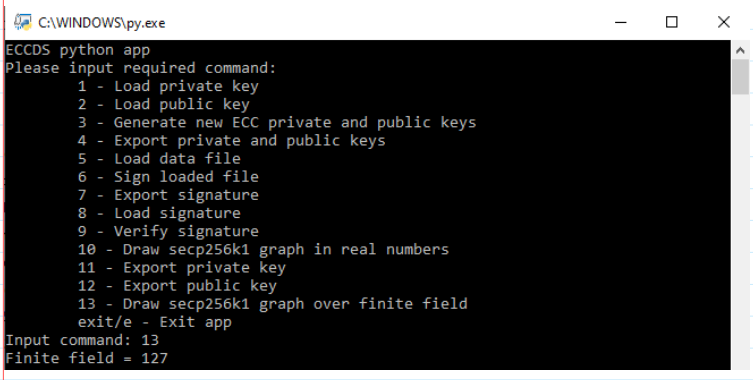

```
>> p=127
p = 127
>> g = 23
g = 23
>> x=0:p-1;
>> a=mod_expv(g,x,p)
>> comet(x,a)
```

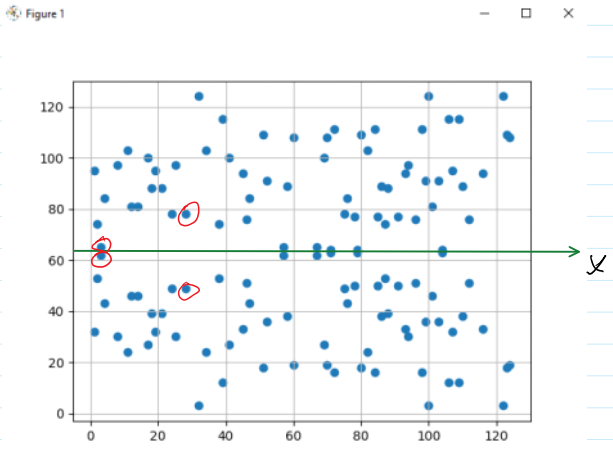
$a = g^{x} \bmod p$



111.ECDSA-Python Edvinas.Repecka.
Complement with:
1.More EC used in blockchain and others broadly used.
2.Introducing animation of EC points computing.
3.Compressed **PuK$_{ECC}$** reconstruction.

$p = 127 ; 126 : 2 = 63$





http://www.secg.org/sec2-v2.pdf

SEC 2: Recommended Elliptic Curve Domain Parameters
Certicom Research Contact: Daniel R. L. Brown (dbrown@certicom.com)
January 27, 2010 Version 2.0

Recommended standard **EC** for Bitcoin and other cryptocurrencies is **secp256k1**
The other standard **EC** is **secp256r1**.

Elliptic curve - EC: **domain parameters** over $F_p = Z_p = \{0, 1, 2, 3, \ldots, p-1\}$ are specified by the tuple $T = (p, a, b, G)$.

$GF_p = \{0, 1, 2, \ldots, p-1\}$

Fnite field F_p is defined by:

p = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF ==
== $2^{224}(2^{32}-1) + 2^{192} + 2^{96} - 1$ // p is prime

The elliptic curve **EC**: $y^2 = x^3 + ax + b \bmod p$ over F_p is defined by:

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC
b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

EC was chosen verifiably at random as specified in ANSI X9.62 [X9.62] from the seed:
S = C49D3608 86E70493 6A6678E1 139D26B7 819F7E90

The base point - generator G in compressed form is:
G = 03
66B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C29
G in uncompressed form is:
G = 04
66B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D98C296
4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

$G = (x_G, y_G)$ $|x_G|$=256 bits, $|y_G|$=256 bits --> $|G|$=512 bits.
The order N of G is:
N = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
$|N| = 8 \cdot 8 \cdot 4 = 256$ bits $N * G = 0$ // 0 - is a neutral element - point at infinity.

If p is prime number and is equal to the number of points of EC, then $p = N$.
It is a case if **EC** is **secp256k1**.
The cofactor is: h = 01

Till this place

EC: **secp256k1**.

Elliptic Curve Digital Signature Algorithm - Bitcoin Wiki
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 to ensure that funds can only be spent by their owner.
https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

ECDSA standards → Standards for Efficient Cryptography Group (SEC)
http://www.secg.org/
Bitcoin follows the **secp256k1** standard.
Public Parameters:
PP=(EC=**secp256k1**; BasePoint G).
secp256k1: $y^2 = x^3 + a \cdot x + b \bmod p$

Public Parameters:
PP=(EC=**secp256k1**; BasePoint ***G***).
secp256k1: $y^2=x^3+a \cdot x+b \mod p$

Compare with:
PP=(Generator ***g***; Prime ***p*** defining group $Z_p^* = \{1, 2, 3, \ldots, p-1\}$).

Addition operations of ***points*** in the Elliptic Curve (EC).
To perform these operations it is required to perform an arithmetic operations with variables
$x, y, a, b \in F_p = \{0, 1, 2, 3, \ldots, p-1\}$; $+_{\mod p}$, $-_{\mod p}$, $\bullet_{\mod p}$, $:_{\mod p}$.

BasePoint ***G*** is a generator of *additive* EC Group of points **secp256k1**.
Then number of points in EC Group defined by **secp256k1** standard is |EC Group|=***p***, where |***p***|=256.

The generation of domain parameters is not usually done by each participant because this involves computing the number of points on a curve which is time-consuming and troublesome to implement.
As a result, several standard bodies published domain parameters of elliptic curves for several common field sizes.
Such domain parameters are commonly known as "standard curves" or "named curves"; a named curve can be referenced either by name or by the unique object identifier defined in the standard documents:

- NIST, Recommended Elliptic Curves for Government Use
- SECG, SEC 2: Recommended Elliptic Curve Domain Parameters
- ECC Brainpool (RFC 5639), ECC Brainpool Standard Curves and Curve Generation

SECG test vectors are also available.[9]
NIST has approved many SECG curves, so there is a significant overlap between the specifications published by NIST and SECG.
EC domain parameters may be either specified by value or by name.
From <https://en.wikipedia.org/wiki/Elliptic-curve_cryptography>

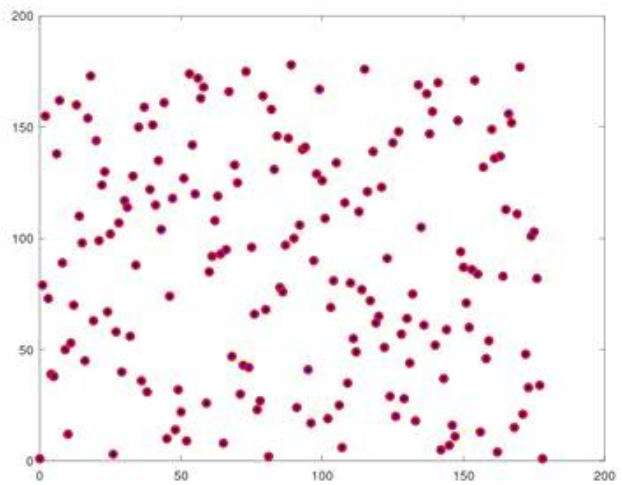
Suppose Alice wants to send a signed message to Bob. Initially, they must agree on the curve parameters which represents the Public Parameters **PP** = (**EC**, **G**, **p**):
EC - is Elliptic Curve tipe;
G - is a base point (generator) of EC;
p - is a number of points of Elliptic Curve, e.g. ***N***=***p*** and ***p*** is prime.

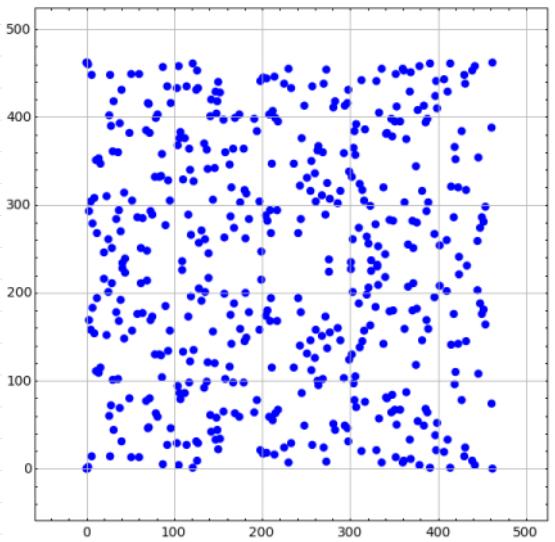
PP	**secp256k1**
EC	the elliptic curve field and equation used $y^2=x^3+a \cdot x+b \mod p$
G	elliptic curve base point, a generator of the elliptic curve with large prime order ***p***
p	Prime integer order of ***G***, means that ***p*******G***=**0**

SHA-256

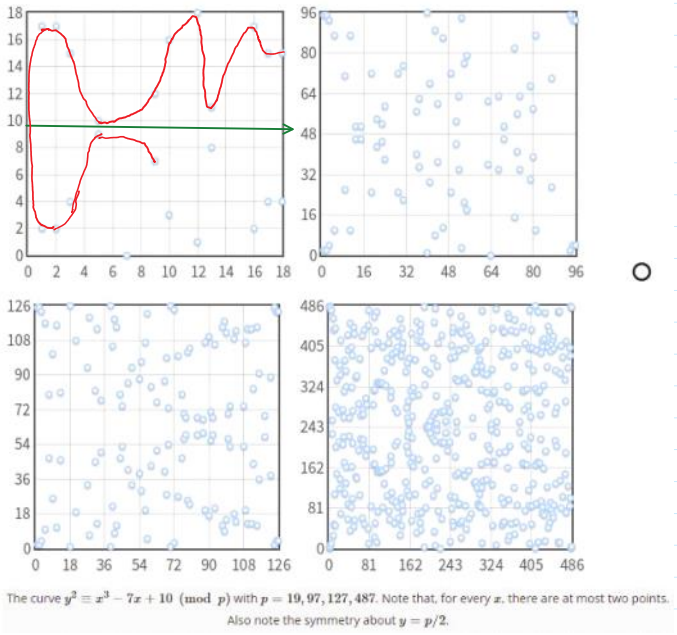
Public Parameters: **PP** = (***EC***, ***G***, ***p***), ***G***=(x_G, y_G)

https://doc.sagemath.org/html/en/thematic_tutorials/explicit_methods_in_number_theory/elliptic_curves.html





https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/



The curve $y^2 \equiv x^3 - 7x + 10 \pmod{p}$ with $p = 19, 97, 127, 487$. Note that, for every x, there are at most two points. Also note the symmetry about $y = p/2$.

The curve y2≡x3−7x+10(modp) with p=19,97,127,487. Note that, for every x, there are at most two points. Also note the symmetry about y=p/2.

From <https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/>

What is an elliptic curve cofactor? - Cryptography Stack Exchange

In cryptography, an elliptic curve is a group which has a given size n. We normally work in a subgroup of prime order r, where r divides n. The "cofactor" is h=n/r. For every point P on the curve, the point hP is either the "point at infinity", or it has order r; i.e., when taking a point, multiplying it by the cofactor necessarily yields a point in the subgroup of prime order r.

The cofactor matters inasmuch as it is not equal to 1:

- When the cofactor is 1, then the subgroup is the whole curve. Any non-zero point is a generator. Any incoming point (x,y) that fulfills the curve equation is part of the subgroup. Everything is fine.
- When the cofactor is *not* 1, then the subgroup of prime order is a strict subset of the curve. When considering a point, verifying that the curve coordinates match the curve equation is not sufficient to guarantee that the point is on the appropriate subgroup. Moreover, there will be points whose order is not a multiple of r. This is what happens, for instance, with Curve25519, which has a cofactor of 8. Such curves require some extra care in the protocol that uses them. For instance, when doing a Diffie-Hellman key exchange over Curve25519, the Diffie-Hellman private keys must be chosen as multiples of 8 (which is expressed as: "set the three least significant bits to zero"); this ensures that the points will be in the proper subgroup.

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

Doubling points in EC

$A=11*G$

$11= 1011_2 = 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 8 + 2 +1 = 11.$

$11= 1011_2 = 2\cdot2\cdot2 + 0\cdot2\cdot2 + 1\cdot2 + 1 = 2\cdot2\cdot2 + 2 + 1$ // $*G$

$A= 2*(2*(2*G)) \boxplus 0*G \boxplus 2*G \boxplus 1*G$

$A= (8*G) \boxplus 2*G \boxplus G.$

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.

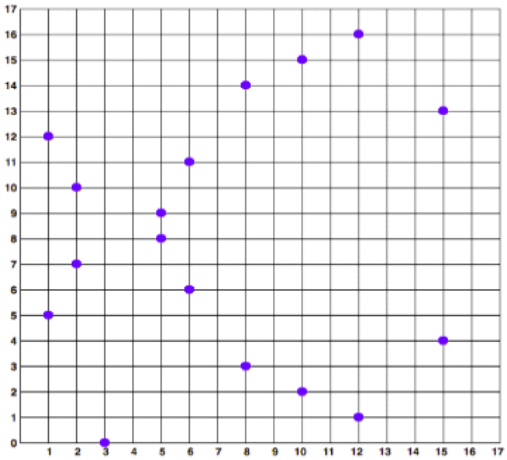


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17