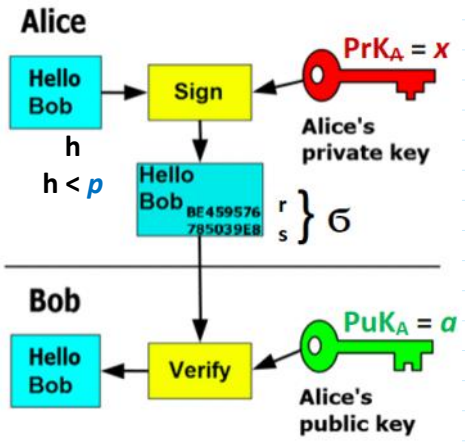
1. El-Gamal E-Signature

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 (DSA) Elliptic Curve Digital Signature Algorithm (ECDSA) is much more widely used.
ECDSA is very widely used in Blockchains.
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: Alfred Menezes et. al.

Private Key PrK: x = randi(p-1)
Public Key PuK: $a = g^x \bmod p$



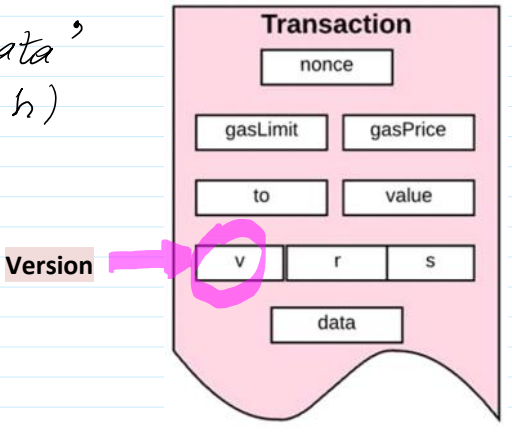
Signature creation for message M >> p.

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 $s = (h - xr)i^{-1} \bmod (p\text{-}1)$.
3. Signature on h-value h is $\sigma = (r, s)$
   Sign(x, h) = $\sigma = (r, s)$.

Authenticity

Ethereum

Tx = 'nonce || gasLimit || gasPrice || to || value || data'
$h = H(Tx)$ → $\sigma = (r, s) = Sign(PrK, h)$



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

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 mod p** and **V2=$a^r r^s$ mod 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.

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

Additive Elliptic Curve Group (ECG)
Number of points N of Elliptic Curve with coordinates (x, y) is an order of ECG: EC points are labeled by capital letters, and scalars with ..., e.g. z.
Addition operation ⊞ 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. EC point G is agenerator if suming G we can get all EC points.
Multiplication of any EC point, say generator G by scalar z: $A=z*G$; $A=G \boxplus G \boxplus G \boxplus \ldots \boxplus G$; z–times. In this case A can be a public key.
Generator–Base Point G: **ECG**={ $i*G$; i=1,2,… , N}; $N*G=0$ and $q*G \neq 0$ if $q<N$.

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.

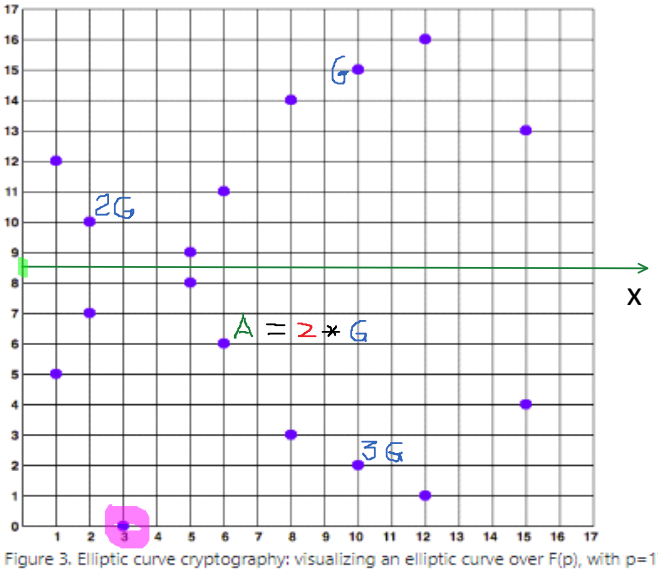


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

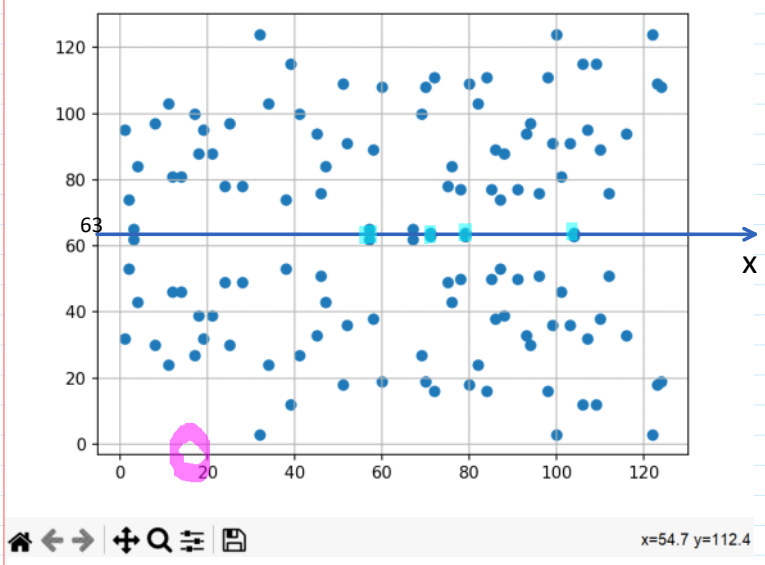
$F_p = Z_p = \{0, 1, 2, 3, \ldots, p-1\}$
When p = 17, then
$F_{17} = Z_{17} = \{0, 1, 2, 3, \ldots, 16\}$

In the case of secp256k1 the number
Of EC poits should be eqal to the number p.

Finite field of integers for EC secp256 creation:
$\mathbf{Z_p = \{0, 1, 2, 3, \ldots, p-1\}}$; p is prime, $p= 2^{255}-19$; $+_{\mathbf{mod}\ p}$, $-_{\mathbf{mod}\ p}$, $\bullet_{\mathbf{mod}\ p}$, $:_{\mathbf{mod}\ p}$ (except divsion by 0).
It is a finite field named also as Galois filed and alternatively denoted by $\mathbf{F_p}$.

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).

$PuK=a=g^x \bmod p$.	$PuK_{ECC}=A=z*G$.
Alice A: x=1975596; a=210649132;	Alice A: z=…..; $A=(x_A, y_A)$;

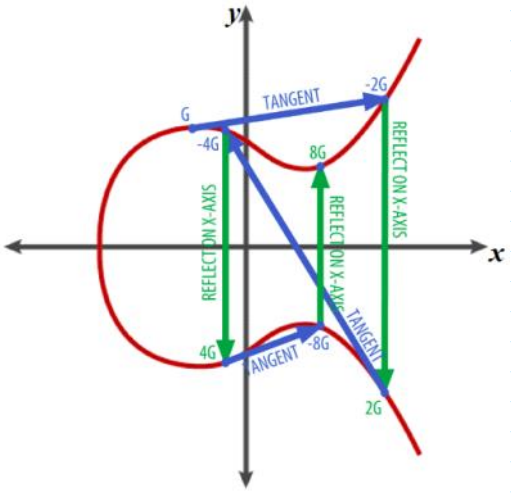


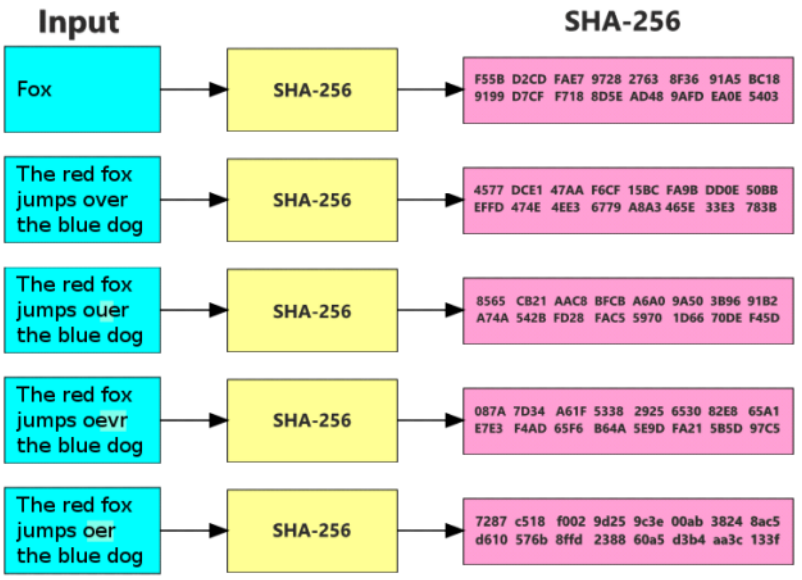
P = 127

$F_{127} = Z_{127}$ = {0, 1, 2, 3, …, p-1} = {0, 1, 2, 3, …, 126}

63 = 126/2

2. Elliptic Curves Digital Signature Algorithm - ECDSA





Bitcoin: H-function is **sha256**.
Ethereum: H-function is **keccak256**.

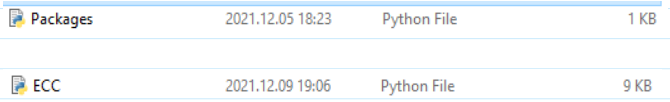
Signature σ creation for message M

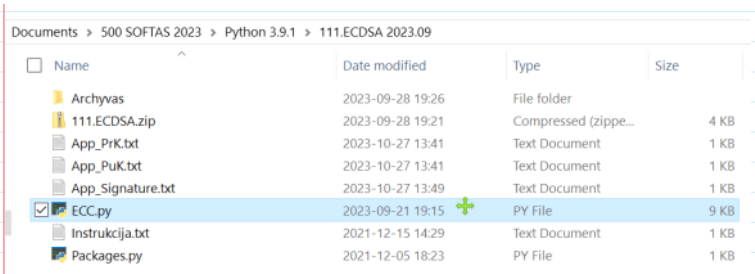
Signature is formed on the h-value h of H-function of M.
Recommended to use SHA256 algorithm which is used in Bitcoin.

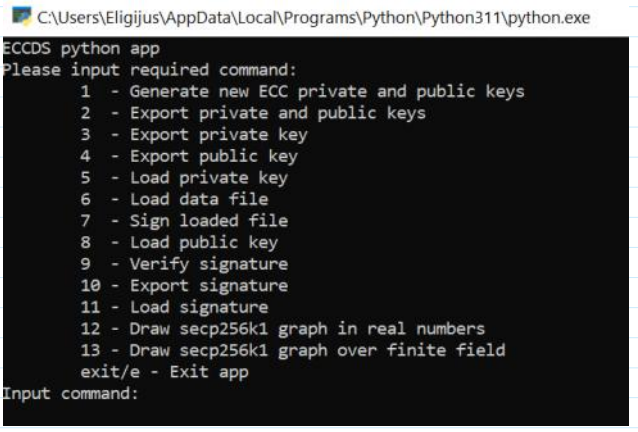
1. h = H(M)=SHA256(M);
2. i <-- randi; $|i| \leq$ 256 bits; // i^{-1} mod p exists since gcd(i, p)=1, since p is prime.
3. $R = i*G = i*(x_G, y_G) = (x_R, y_R)$;
4. $r = x_R \bmod p$;
5. $s = (h + z \bullet r) \bullet i^{-1} \bmod p$; $|s| \leq$ 256 bits; // **s** satisfies condition that gcd(s, p)=1,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

Signature vrification: Ver(PuK, σ, h)
1. Calculate $v_1 = h \bullet s^{-1} \bmod p$ and $v_2 = r \bullet s^{-1} \bmod p$
2. Calculate the curve point $V = v_1*G \boxplus v_2*A = V(x_V, y_V)$
3. The signature is valid if R=V: ==> r=x_V=x_R mod p.

ECDSA using **secp256k1** software:
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.

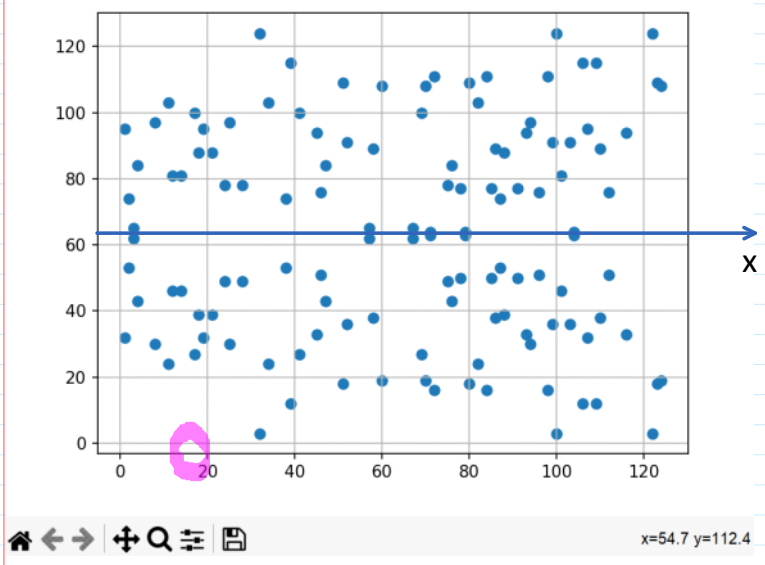






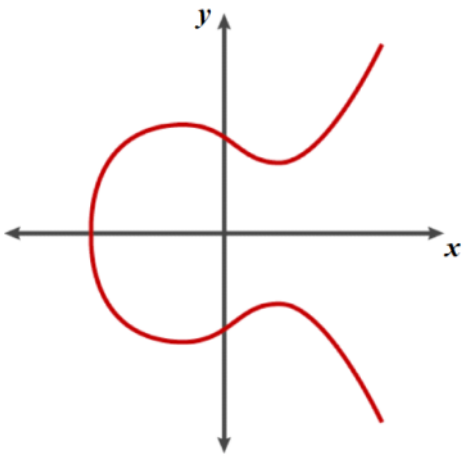
PrK: has 256 bit length or 64 hexadecimal digits:
0x4f30dcb4235b9d52d0cd9aef9eb11b9c75b07ef25fc2249491b071f48803e5d3

PuK: has 512 bit length or 128 hexadecimal digits:
0x0beffa8fb3b2665089333456bc98b3c003181d7b7c23750c1dde4fa070608442638d7391846bda26aad95accba5
810fc326611e804506f95e93ffa006d15b0f0



Till this place

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

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.