http://crypto.fmf.ktu.lt/telekonf/archyvas/B127%20DuomenuSauga/B127%20DataSecurity%202024/

Elliptic Curve Cryptosystem - ECC

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

$F_p = \{0, 1, 2, ..., p-1\}$
$*$ mod p

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

Then evidently

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

For example:
-2 mod 11 = 9

2 + (-2) mod 11 = 2 + 9 mod 11 = 11 mod 11 = 0

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

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.

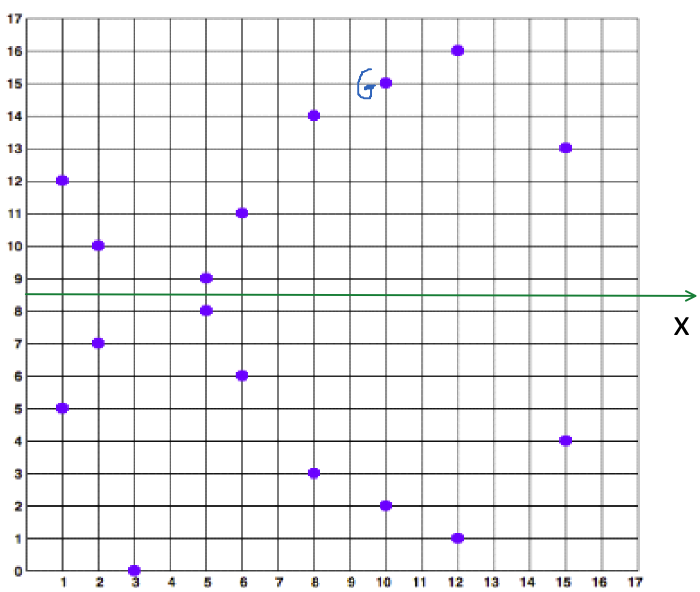


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over F(p), with p=17

v, r, s, Ethereum signature

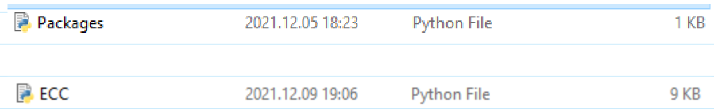
Key generation

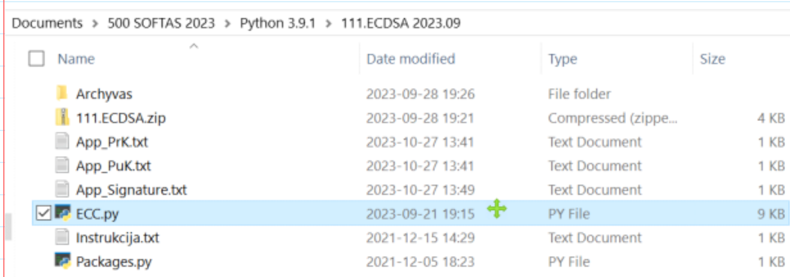
1.Install Python 3.9.1.

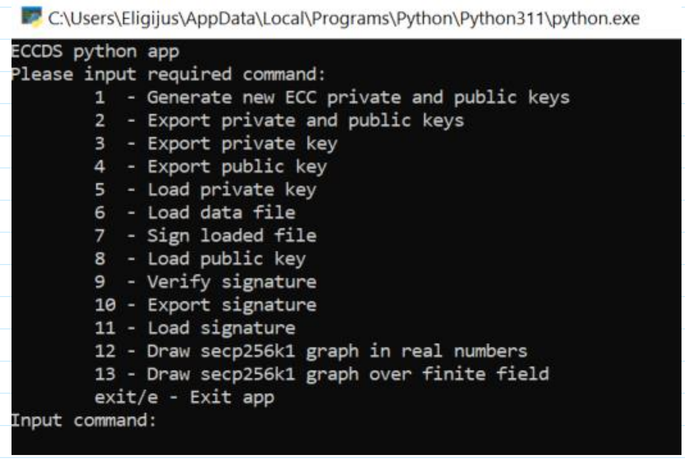
2.Launch script Packages for joining a libraries.

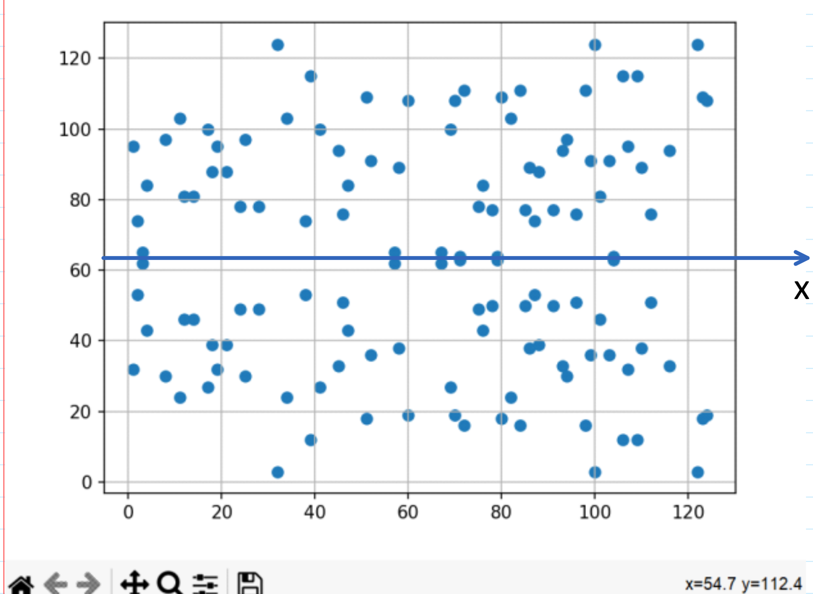
3.Launch file ECC.

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









Elliptic Curve Digital Signature Algorithm - ECDSA

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

$1<x_G<\mathbf{n}$, $1<y_G<\mathbf{n}$.

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.

$PrK_A=z$ <-- randi; $z<\mathbf{n}$, max|**z**|<=256 bits.

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

Signature creation for message ***M***

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; $|i| \leq$ 256 bits; >> gcd(i,p)=1 --> ∃ ! such that i^{-1} exists.
3. R = $i*G = i*(x_G, y_G) = (x_R, y_R)$;
4. $r = x_R$ mod p;
5. $s = (h + z \bullet r) \bullet i^{-1}$ mod p; $|s| \leq$ 256 bits; // Since i p satisfies the condition that gcd(i,p)=1, then exists i^{-1} mod p.
   // >> i_m1=mulinv(i,p)   % in Octave   σ
6. **Sign**(**PrK**$_{ECC}$=**z**, ***PP***, h) = σ = (r, s)

Signature vrification: Ver(PuK, σ, h)

1. Calculate $u_1 = h \bullet s^{-1}$ mod p and $u_2 = r \bullet s^{-1}$ mod p
2. Calculate the curve point $V = u_1*G + u_2*A = V(x_V, y_V)$
3. The signature is valid if $R=V$; $r=x_V=x_R$ mod p.

ECDSA	ElGamal Signature	Schnorr Signature
h = H(m);	h = H(m);	h = H(m);
i ←randi; Compute i^{-1} mod p	i ←randi; gcd(i, p-1)=1 Compute i^{-1} mod (p-1)	i ←randi;
$R = i*G = i*(x_G, y_G) = (x_R, y_R)$; $r = x_R$ mod p; $\|i\| \leq$ 256 bits;	$r=g^i$ mod p;	$r=g^i$ mod p;
$s=(h+z \bullet r)i^{-1}$ mod p; $\|s\| \leq$ 256 bits;	$s=(h-x \bullet r)i^{-1}$ mod (p-1);	$s=(i+x \bullet h)$ mod (p-1);
$s^{-1}=(h+z \bullet r)^{-1} i$ mod p;	$h=xr+is$ mod (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 $u_1=h \bullet s^{-1}$ mod p and $u_2=r \bullet s^{-1}$ mod p;	Compute: $u_1= g^h$ **mod** p; and $u_2= a^r r^s$ **mod** p	Compute: $u_1= g^s$ **mod** p. and $u_2= r a^h$ **mod** 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 $r=x_R$ mod p.		

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 mod p.

$u + v * P$ $\sqrt{a+x} - \sqrt{a} = \sqrt{x}$

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** $_{ECC}$=$A=(a_x, a_y)$;

|**PrK** $_{ECC}$=**z**|=256 bits; |**PuK** $_{ECC}$=**A**|=512 bits.

2^{256} 2^{40} – 1T ~ 10^{12}

Doubling points in EC

$A=11*G$

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

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

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

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

[Ethereum signatures uses ECDSA and secp256k1 constants to define the elliptic curve.](https://www.bing.com/search?q=ethereum+signature&PC=U316&FORM=CHROMN)

From <https://www.bing.com/search?q=ethereum+signature&PC=U316&FORM=CHROMN>

Ethereum for signing transactions is using **secp256k1** EC together with **keccak256** H-function.

secp256k1 has co-factor=1. When the cofactor is 1, everything is fine.

The signature of transaction in **Ethereum** is placed in the varaibles **v**, **r**, **s**.

Variable **v** represents the version of signature and (**r**, **s**)=**σ**.

Public-key cryptography is based on the intractability of certain mathematical problems.
Early public-key systems are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors.
For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point (generator) is infeasible: this is the "elliptic curve discrete logarithm problem" (ECDLP).
The security of elliptic curve cryptography depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points.
The size of the elliptic curve determines the difficulty of the problem.
The primary benefit promised by elliptic curve cryptography is a smaller key size, reducing storage and transmission requirements, i.e. that an elliptic curve group could provide the same level of security afforded by an RSA-based system with a large modulus and correspondingly larger key: for example, a 256-bit elliptic curve public key should provide comparable security to a 3072-bit RSA public key.
The U.S. National Institute of Standards and Technology (NIST) has endorsed elliptic curve cryptography in its Suite B set of recommended algorithms, specifically elliptic curve Diffie–Hellman (ECDH) for key exchange and Elliptic Curve Digital Signature

IBM
Peter Shoor
433 qbits
quantum
entalgement
1KB = 1024
1024^2
2^{1024}

Algorithm (ECDSA) for digital signature.
The U.S. National Security Agency (NSA) allows their use for protecting information classified up to top secret with 384-bit keys.[2]
However, in August 2015, the NSA announced that it plans to replace Suite B with a new cipher suite due to concerns about quantum computing attacks on ECC.[3]

https://en.wikipedia.org/wiki/SHA-2

SHA-2 (**Secure Hash Algorithm 2**) is a set of cryptographic hash functions designed by the United States National Security Agency(NSA).[3] Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed "hash" (the output from execution of the algorithm) to a known and expected hash value, a person can determine the data's integrity.

SHA-160

$2^{\sqrt{160}} = 2^{80}$

2^{70}

SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**.

2^{128} birthday paradox $2^{256} = \sqrt{2^{512}}$

2^{112} secure against brute force attack