Mid Term Exam (**MTE**) will be held in Thursday, 5-th of November, at 17:30 distantly, on-line.
Please connect to Teams:
https://teams.microsoft.com/meet/356624603496709?p=J4owseOGUXMfnrnqWX

During the MTE you must solve 2 problems:
1. Diffie-Hellman Key Agreement Protocol - DH KAP.
2. Man-in-the-Middle Attack (MiMA) for Diffie-Hellman Key Agreement Protocol - DH KAP.
The problems are presented in the site:
https://imimsociety.net/en/14-cryptography

Please register to the site and after that you receive 10 Eur virtual money to purchase the problems.
For registration you should input the first 2 letters of your Surname and full Name, e.g. John Smith
Should register as **Sm John**.

Please purchase the only one problem at a time.

If the solution is successful then you are invited to press the green button [Get reward].
No any other declaration about the solution results from your side is required.
If the solution failed, then you must press the button [Return] in the top on the left side.

More info is presented in my lecture 111_006_
You can solve the problems as many times as you wish before the MTE for the better preparation.

You can solve the problems in imimsociety as many times as you wish to better prepare for MTE.

I advise you to try at first to solve the problem in 'Intellect' section to exercise the brains.
It is named as 'WOLF, GOAT AND CABBAGE TRANSFER ACROSS THE RIVER ALGORITHM'.
< https://imimsociety.net/en/home/15-wolf-goat-and-cabbage-transfer-across-the-river-algorithm.html>

The questions concerning the MTE you can ask at the end of the lectures.

Poster Report (**PR**) topics are presented in the Google drive:
https://docs.google.com/document/d/1UjYY4fG0z0wevZtO4sMnxPP1O983uxqd/edit?usp=sharing&ouid=111502255533491874828&rtpof=true&sd=true

The requirements of PR are presented in in the Google drive:
https://docs.google.com/document/d/1IPzwEVVmvnObQoJPRP9GDUiHccEGWgMw/edit?usp=sharing&ouid=111502255533491874828&rtpof=true&sd=true

Main identities:
Let $x*P = Q$, $y*P = R$ and $Q \boxplus R = T$, then
$(x + y)*P = x*P \boxplus y*P = Q \boxplus R = T$.

The last identity is used when $P = G$ and G is a generator.

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.

Let us consider abstract EC defined in XOY plane 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 for p=11:
-2 mod 11 = 9
2^2 mod 11 = 4 & 9^2 mod 11 = 4
>> mod(9^2,11)
ans = 4

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=…..; $A=(x_A, y_A)$;

ECDSA: Signature creation for message M

Signature is formed on the h-value h of Hash function **H** of M.
In Bitcoin H-function is SHA256 algorithm
In Ethereum H-function is keccak-256 algorithm it is an algorithm of the family of SHA-3 standard.

1. h = **H**(M)=SHA256(M);
2. i <-- randi; $|i| \leq$ 256 bits; >> **gcd**(i,p)=1 --> Then there exists unique i^{-1} **mod** p such that $ii^{-1} = 1$ **mod** p.
3. R = $i*G = R(x_R, y_R)$;
4. $r = x_R \bmod p$;
5. $s = (h + z \bullet r) \bullet i^{-1} \bmod p$; $|s| \leq$ 256 bits; // Parameter s must satisfy the condition that **gcd**(s, p)=1, then exists $s^{-1} \bmod p$.
   // >> i_m1=mulinv(i,p) % in Octave
6. **Sign**(PrK_{ECC}=z, PP, h) = **ϭ** = (r, s)

ECDSA: Signature vrification: Ver(PuK=A, σ, h)

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

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 $R = i * G = (x_R, y_R)$; $r = x_R$.

$\mathbf{PrK}_{ECC} = z < 2^{256}$ is generated at random;

$\mathbf{PuK\ ECC} = A = z * G = (a_x, a_y)$; $|\mathbf{PuK}_{ECC} = A| = 512$ bits.

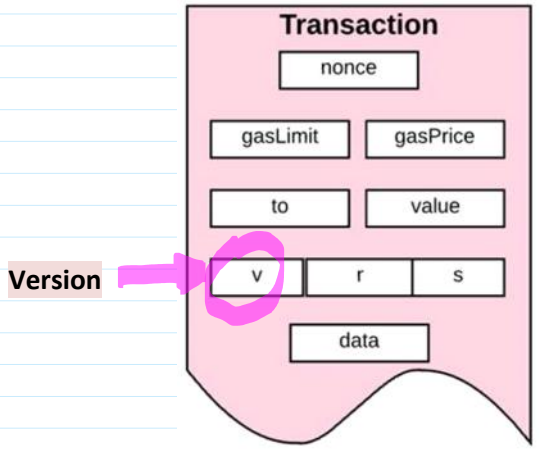
$G = (x_G, y_G)$

ECDSA	ElGamal Signature
$h = H(m)$;	$h = H(m)$;
$i \leftarrow$randi; Compute $i^{-1} \bmod p$	$i \leftarrow$randi; gcd(i, p-1)=1 Compute $i^{-1} \bmod (p-1)$
$R = i * G = i * = (x_R, y_R)$; $r = x_R \bmod p$; $\lvert i \rvert \leq 256$ bits;	$r = g^i \bmod p$;
$s = (h + z \bullet r) i^{-1} \bmod p$; $\lvert s \rvert \leq 256$ bits;	$s = (h - x \bullet r) i^{-1} \bmod (p-1)$;
$s^{-1} = (h + z \bullet r)^{-1} i \bmod p$;	$h = xr + is \bmod (p-1)$.
Sign($\mathbf{PrK}_{ECC}$=z, h) = (r, s) = σ;	**Sign**(PrK=x, h) = (r, s) = σ;
ECDSA Verification	**ElGamal Signature Verification**
Compute $u_1 = h \bullet s^{-1} \bmod p$ and $u_2 = r \bullet s^{-1} \bmod p$;	Compute: $u_1 = g^h \bmod p$; and $u_2 = a^r r^s \bmod p$
Compute $R = u_1 * G + u_2 * A = (x_R, y_R)$;	Signature is valid if: $u_1 = u_2$
The signature is valid if $r = x_V \bmod p$.	

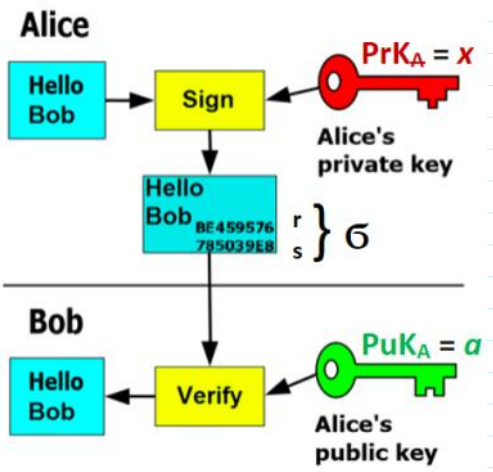
Schnorr Signature
$h = H(m)$;
$i \leftarrow$randi;
$r = g^i \bmod p$;
$s = (i + x \bullet h) \bmod (p-1)$;
Sign(PrK=x, h) = (r, s) = σ;
Schnorr Signature Verification
Compute: $u_1 = g^s \bmod p$. and $u_2 = r a^h \bmod p$
Signature is valid if: $u_1 = u_2$

Ethereum

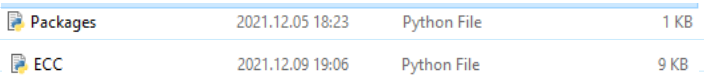
GasLimit is evaluated by Wey.
1 Wey = 10^{-18} Eth --> 1 Eth = 10^{18} Wey.
Usually 1Gas is dozens of thousands of Wey's.
Parameters (***v***, ***r***, ***s***) is an Ethereum signature:
v - version; but not a **PuK** for signature verification.
PuK - is placed in Ethereu eWallet named MetaMask.
r - first component of signature;
s - second component of signature.



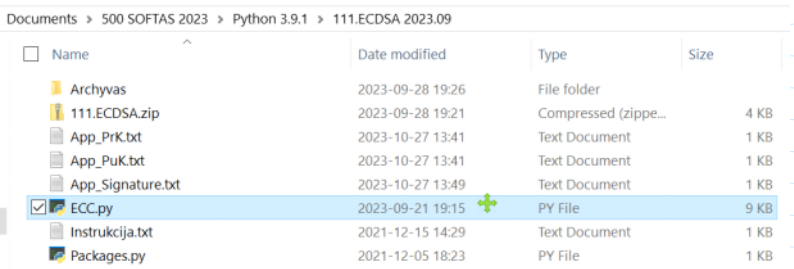
Tx = 'nonce||gasLimit||gasPrice||to||value||v||r||s||data'
h = H(Tx) ---> Sign(**PrK**, h) = $\sigma = (r, s)$

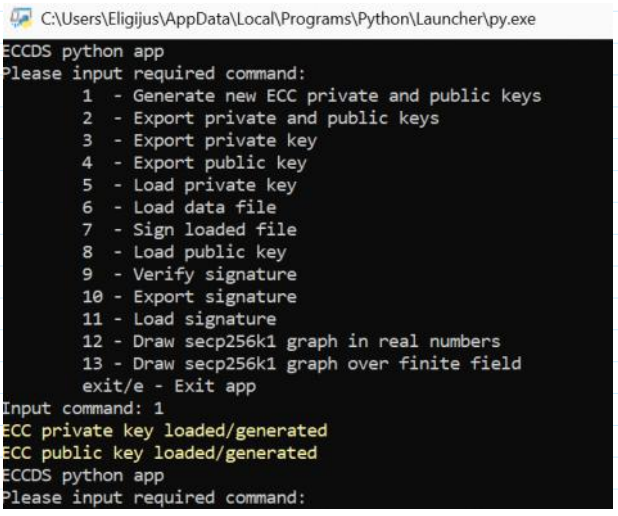


ECDSA using **secp256k1** software:
1.Install Python 3.9.1.
2.Launch script Packages for joining a libraries.
3.Launch file ECC.
4.If window is escaping, then open hiden windows in icon near the Start icon.



Software can be found in the site
https://crypto.fmf.ktu.lt/xdownload/

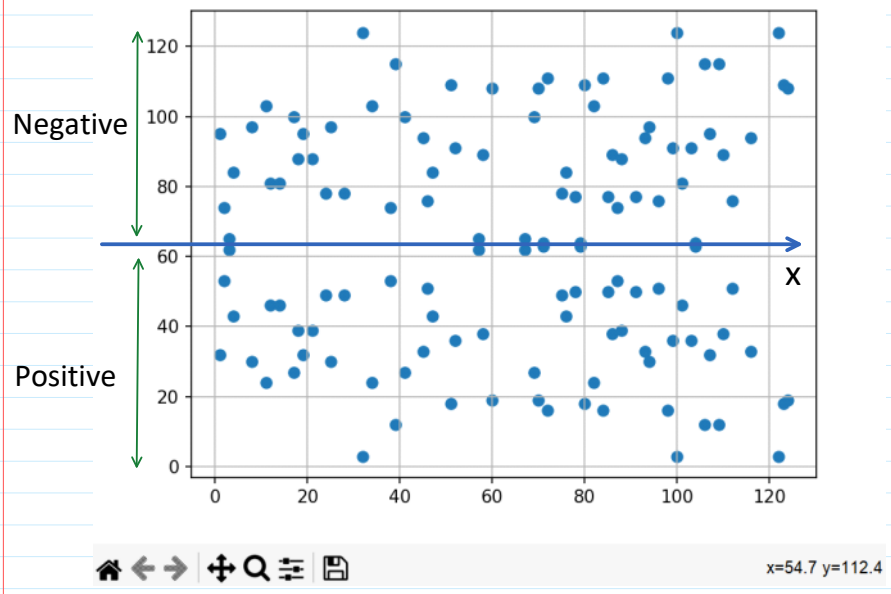




PrK:
B64e8c0c98b2aaaf36caea44a452cb1c7a22b88859cf08a5b4f7d45b165ef82d

PuK:
x = 424da40e7cfe5cccc2204aa1785bced9aef8e2845843479dbc1f128c8ae08810
y = ed83908cb8352ce098598416be3957f45e8f24553dc7adec3cd523a7ffff2873

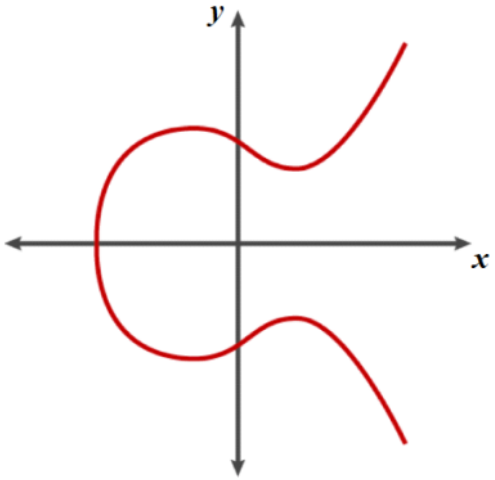
The signature created by **PrK** on **PuK** in file App_Signature (self-signing example) is represented in the following way:
r = d0fee51d4ebbfaa03fe9b37d6cd6aa6242260e6f9070c3b81131f71c00ad9502
s = b77b56f39b251c11660dd06f808e4d3fdb9052582427d63130e4969780e3f8f8



F_p = {0, 1, 2, 3, …, p-1}
Operations are mod p
p = 127; p - is prime
F_{127} = (0, 1, 2, 3, …, 126)
Operations are mod 127

y = 35 mod 127 = 35
How to find -y ?
y + (-y) = 0 mod 127
127 mod 127 = 0
y + (-y) mod 127 = y + (0-y) mod 127 =
= y + (127-y) mod 127 = y - y + 127 mod 127 =
= 0 + 127 mod 127 = 127 mod 127 = 0
-y = 127-y = 127 - 35 = 92

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

$PuK_{ECC}=A=z*G=(x_A, y_A)$ of EC we can compress to 256 bits;
If y_A is even, then we add prefix 02x to PuK_{ECC} coordinate x

If y_A is odd, then we add a prefix 03x to $\mathbf{PuK_{ECC}}$ coordinate x

Let **PuK**:
x = 424da40e7cfe5cccc2204aa1785bced9aef8e2845843479dbc1f128c8ae08810
y = ed83908cb8352ce098598416be3957f45e8f24553dc7adec3cd523a7ffff2873

Then for **PuK** representation we can use only coordinate x represented in the form:
x =03x424da40e7cfe5cccc2204aa1785bced9aef8e2845843479dbc1f128c8ae08810

Then using EC equation ove F_p: $y^2=x^3+ax+b \bmod p$,
we can compute y^2 by putting coordinate x:
424da40e7cfe5cccc2204aa1785bced9aef8e2845843479dbc1f128c8ae08810
and then extract square root from y^2 finding y and -y and choose odd value
according to prefix 03x.

Till this place

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.