Turinys

Sąvokos 2
Elipsinių kreivių kriptografija 3
Aplinkos pasiruošimas pratyboms 4
Daugybos ir sudėties operacijos elipsinėse kreivėse 5
Privataus ir viešo raktų poros apskaičiavimas 7
Elipsinių kreivių viešojo rakto suspaudimas 9
Difio Helmano raktų apsikeitimo protokolas naudojant elipsines kreives 12
Elipsinių kreivių elektroninio parašo algoritmas 14
Simetrinis šifravimas ir iššifravimas naudojant Difio Helmano autentifikuotą raktų apsikeitimo protokolą su Elipsinėmis kreivėmis ir AES-GCM 21
Autentifikuotas Difio Helmano raktų apsikeitimo protokolas su ECDSA parašais 21
Simetrinis šifravimas ir iššifravimas naudojant AES-GCM 27
Homomorfinė elipsinių kreivių savybė 29
Pedersono įsipareigojimas naudojant elipsines kreives 31
Įsipareigojimas 31
Maskavimas 31
Pedersono įsipareigojimas 32
Konfidencialių ir patikrintinų operacijų sistemos pavyzdys su Pedersono įsipareigojimais 32
Monero žiediniai parašai (LSAG) 46

Sąvokos

Kriptografijoje **Aldona** (angl. *Alice*) ir **Bronius** (angl. *Bob*) yra standartizuoti vardai, dažnai naudojami aprašant dviejų subjektų, dalyvaujančių komunikacijoje, veiksmus. **Aldona** paprastai veikia kaip informacijos siuntėja, o **Bronius** – kaip jos gavėjas. Šie vardai plačiai naudojami kriptografinių protokolų ir sistemų paaiškinimui, padedant vizualizuoti saugaus informacijos perdavimo scenarijus.

Papildomai, kriptografiniuose pavyzdžiuose dažnai naudojami ir kiti veikėjai (keletas iš jų):

- **Pasyvi Zosė (**angl. ***Eve*****)** – pasyvus užpuolikas arba stebėtojas, bandantis perimti ar iššifruoti **Aldonos** ir **Broniaus** komunikaciją.
- **Aktyvi Zosė (**angl. ***Mallory*****)** – aktyvus užpuolikas, kuris ne tik šnipinėja, bet ir bando pakeisti ar sugadinti komunikaciją.

Ši terminologija naudojama įvairiuose kriptografijos procesuose, pavyzdžiui, šifravime, skaitmeniniuose parašuose ir raktų mainuose, siekiant supaprastinti sudėtingas technines diskusijas.

Elipsinių kreivių kriptografija

Elipsinių kreivių kriptografija (angl. *Elliptic Curve Cryptography*, *ECC*) pirmą kartą buvo aptarta 1987 m. Nelo Koblitzo (angl. *Neal Koblitz*) straipsnyje *Elipsinių kreivių kriptosistema*. ECC yra viešojo rakto kriptografijos metodas, pagrįstas elipsinių kreivių algebrine struktūra baigtiniuose laukuose. ECC leidžia pasiekti **tokį patį saugumo lygį naudojant mažesnius raktus**, lyginant su kitomis sistemomis, tokiomis kaip **RSA** ar **ElGamalo kriptosistemos**, kurios naudoja modulines eksponentes Galois laukuose.

Elipsinės kreivės taikomos įvairiose srityse.

- **Raktų mainams** – naudojamos bendro slapto rakto apsikeitimui (angl. *Elliptic Curve Diffie-Hellman*, *ECDH*).
- **Elektroniniams parašams** – elipsinių kreivių skaitmeninis parašas **ECDSA** (angl. *Elliptic Curve Digital Signature Algorithm*).
- **Pseudoatsitiktinių skaičių generavimui** – naudojamos šifruotų duomenų apsaugai ir slaptoms reikšmėms kurti.
- **Šifravimui** – netiesiogiai naudojamos kartu su simetrine kriptografija, kaip dalis hibridinių šifravimo schemų.
- **Sveikųjų skaičių skaidymui** (angl. *factorization*) – elipsinės kreivės naudojamos Lenstros elipsinių kreivių skaidymo metode, kuris gali būti pritaikytas kriptografinių sistemų saugumui įvertinti.

Kriptografijoje naudojamos elipsinės kreivės (žr. 1 pav.), apibrėžtos baigtiniuose laukuose, t. y. taškų koordinatės yra baigtinio lauko elementai ir aritmetika tarp jų yra atliekama pagal baigtinio lauko taisykles.

Elipsinės kreivės taškų grupę baigtiniame lauke GF(p) nusako du tokie sveikieji neneigiami skaičiai a ir b, mažesni už p, kad

$$4a^3 + 27b^2 \bmod p \neq 0.$$

Tuomet elipsinės kreivės taškų grupę $E_p(a, b)$ sudaro aibė porų (x, y) kartu su be galo nutolusiu tašku O. Skaičiai x ir y yra sveikieji neneigiami, mažesni už p, ir tenkina lygybę (*kitoms elipsinėms kreivėms lygtis gali skirtis*):

$$y^2 = x^3 + ax + b \bmod p.$$

Elipsinės kreivės taškų grupė baigtiniame lauke taip pat yra baigtinė. Visi šios grupės taškai randami taip:

1. Apskaičiuojami visų sveikųjų teigiamų skaičių, mažesnių už p, kvadratai moduliu p, t. y. kiekvienam t ($0 \leq t < p$) apskaičiuojamas $t^2 \bmod p$;
2. Kiekvienam x ($0 \leq x < p$) apskaičiuojamas reiškinys $x^3 + ax + b \bmod p$. Jei gauta reikšmė yra ir baigtinio lauko elementų kvadratų lentelėje, tuomet taškas (-ai) (x, y) yra grupės elementas. Čia y atitinka kvadratinę šaknį (moduliu p) iš $x^3 + ax + b$;
3. Prie gautos taškų aibės prijungiamas be galo nutolęs taškas O.

Bendruoju atveju tiksliai nusakyti elipsinės kreivės grupės baigtiniame lauke eilę neįmanoma. Dėl to, naudojamas sąryšis:

$$p + 1 - 2\sqrt{p} \leq |E_p(a, b)| \leq p + 1 + 2\sqrt{p}.$$

Konkrečios elipsinės kreivės $E_p(a, b)$ atveju dviejų taškų $P = (x_1, y_1)$ ir $Q = (x_2, y_2)$ suma $R = (x_3, y_3)$ apibrėžiama formulėmis:

$$x^3 = \lambda^2 - x_1 - x_2 \bmod p,$$

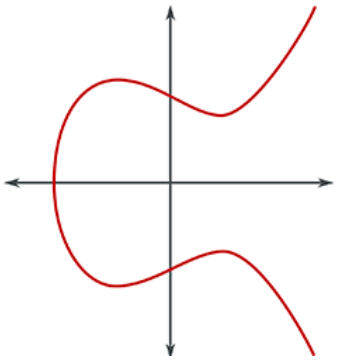
$$y^3 = \lambda(x_1 - x_3) - y_1 \bmod p;$$

čia

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} \bmod p, & \text{kai } P \neq Q, \\ (3x_1^2 + a)(2y_1)^{-1}, & \text{kai } P = Q. \end{cases}$$

Analogiškai įvedama ir elemento daugybos iš skaičiaus operacija, pvz., $4P = P \boxplus P \boxplus P \boxplus P$. Sumos ir daugybos iš skaičiaus operacijos tenkina distributyvumo dėsnį, t. y. bet kuriems dviem taškams P ir Q iš elipsinės kreivės $E_p(a, b)$ grupės ir bet kuriam sveikajam skaičiui k yra teisinga lygybė:

$$k(P \boxplus Q) = kP \boxplus kQ.$$



1 pav. elipsinės kreivės pavyzdys

Aplinkos pasiruošimas pratyboms

Pitono įdiegimo instrukcija (papildoma įdiegimo instrukcija, jeigu reikia rankiniu būdu pridėti pitono kelią į *windows* sistemą).

Įdiegus Pitono interpretatorių *Windows* komandinėje eilutėje (angl. *cmd*) įveskite šias komandas (reikiamų Pitono bibliotekų įdiegimas ir Pitono komandinės eilutės paleidimas – paskutinė komanda):

```
pip install tinyec
pip install ecdsa
pip install cryptography
python
```

Pratyboms su elipsinėmis kreivėmis naudosime šias bibliotekas:

tinyec – skirta elipsinių kreivių kriptografijai;
secrets – skirta kriptografiškai saugiems atsitiktiniams skaičiams generuoti. Ji naudojama vietoje atsitiktinių skaičių generavimo (angl. *random*), nes:
🔒 užtikrina aukštesnį atsitiktinumo lygį (svarbu generuojant raktus);
🔒 nėra nuspėjama (tinkama slaptažodžiams, raktams).
hashlib – skirta saugioms santraukos funkcijoms generuoti.
ecdsa – skirta elipsinių kreivių skaitmeninio parašo algoritmams ir elipsinių kreivių operacijų atlikimui.

cryptography – skirta saugiems ir moderniems kriptografiniams algoritmams, t.y. skirta šifravimui, skaitmeniniams parašams, raktų išvedimui ir duomenų autentiškumo užtikrinimui.

```
>>> import tinyec.ec as kreive
>>> import tinyec.registry as registras
>>> import secrets
>>> import hashlib as santrauka
>>> from ecdsa.numbertheory import square_root_mod_prime
# cryptography biblioteką naudosime vėliau
```

Toliau naudosime, plačiausiai naudojamą, *secp256r1* (P-256) elipsinę kreivę (***tinyec*** palaikomos elipsinės kreivės)[1]:

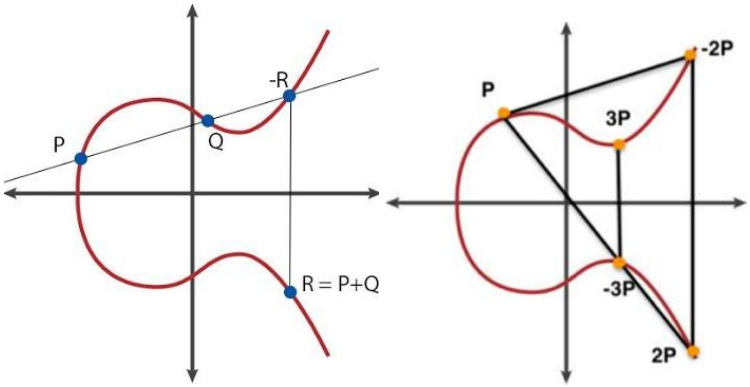
- Pirminis skaičius ***p***, kurio pagalba atliekama aritmetika tarp elipsinės kreivės taškų, ***p*** = 115792089210356248762697446949407573530086143415290314195533631308867097853951 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff;
- *a* ir *b* nurodo kreivės formą arba kitaip taškus kurie priklauso kreivei, *a* = 115792089210356248762697446949407573530086143415290314195533631308867097853948 0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc, *b* = 41058363725152142129326129780047268409114441015993725554835256314039467401291 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b;
- Bazinis taškas arba grupės generatorius, *G* = (48439561293906451759052585252797914202762949526041747995844080717082404635286; 36134250956749795798585127919587881956611106672985015071877198253568414405109) (0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296, 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5);
- Elipsinės kreivės taškų skaičius, *n* = 115792089210356248762697446949407573529996955224135760342422259061068512044369 0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551.

```
>>> elk = registras.get_curve('secp256r1')
```

Daugybos ir sudėties operacijos elipsinėse kreivėse

Šiame poskyryje praktiškai pasibandysite elipsinių kreivių sudėties ir daugybos operacijas tarp taškų, vizualiai pateikiamas 1 pav.

[1] Šaltinis platesniam susipažinimui su standartizuotomis elipsinėmis kreivėmis.



1 pav. Supaprastinta elipsinių kreivių sudėties ir daugybos operacijų vizualizacija

Pasirinkite atsitiktinį skaičių pvz. $a = 5$ ir apskaičiuokite $P = a * \boldsymbol{G}$:

```
>>> a = 5
>>> P = a * elk.g
>>> print("Px =", P.x, "\nPy =", P.y)
Px = 367946693408968830121014734395389297591523964766486925917953181940545801553 73
Py = 101659946828913883886577915207667153874746613498030835602133042203824767462820
```

Pasirinkite atsitiktinį skaičių pvz. $b = 5$ ir apskaičiuokite $Q = b * \boldsymbol{G}$:

```
>>> b = 7
>>> Q = b * elk.g
>>> print("Qx =", Q.x, "\nQy =", Q.y)
Qx = 643754830177177113486348896017938363299664479635106486816256812113489438767 71
Qy = 524313919169835044232176278490209167296019694090539011925613228059625775433 48
```

Patikrinkite ar lygybė galioja $P \boxplus Q = a * \boldsymbol{G} \boxplus b * \boldsymbol{G}$:

```
>>> PQ = P + Q
>>> print("PQx =", PQ.x, "\nPQy =", PQ.y)
PQx = 525210041856415366272666005368048169315353291333555399620209801938023830578 60
PQy = 336532199988672138926993714427671109158562719686560581596931230187280744494 7
>>> R = a * elk.g + b * elk.g
>>> print("Rx =", R.x, "\nRy =", R.y)
Rx = 525210041856415366272666005368048169315353291333555399620209801938023830578 60
Ry = 336532199988672138926993714427671109158562719686560581596931230187280744494 7
>>> PQ == R
True
```

Apskaičiuokite $R_2 = (a + b) * \boldsymbol{G}$ ir patikrinkite ar galioja distributyvumo tapatybė $a * \boldsymbol{G} \boxplus b * \boldsymbol{G} = (a + b) * \boldsymbol{G}$:

```
>>> R2 = (a + b) * elk.g
>>> print("R2x =", R2.x, "\nR2y =", R2.y)
R2x = 525210041856415366272666005368048169315353291333555399620209801938023830578 60
R2y = 336532199988672138926993714427671109158562719686560581596931230187280744494 7
>>> R == R2
True
```

Patikrinkite ar galioja distributyvumo tapatybė $a * (P \boxplus Q) = a * P \boxplus a * Q$:

```
>>> L = a * (P + Q)
>>> print("Lx =", L.x, "\nLy =", L.y)
Lx = 26483433592348094702300487170013003498033287656110843234796600681993023839 71
Ly = 84596004577236158890436657491599029909992396403682976655398612345353730736932
>>> L2 = a * P + a * Q
>>> print("L2x =", L2.x, "\nL2y =", L2.y)
L2x = 26483433592348094702300487170013003498033287656110843234796600681993023839 71
L2y = 84596004577236158890436657491599029909992396403682976655398612345353730736932
>>> L == L2
True
```

Privataus ir viešo raktų poros apskaičiavimas

Raktų generavimas susideda iš šių žingsnių:

1. Sugeneruoti privatų raktą (PR) x, pasirenkant atsitiktinį skaičių $x \leftarrow \text{randi}(Z_{n-1})$, ir patikrinti ar tenkinama sąlyga $1 < x < n-1$:

   ```
   >>> x = secrets.randbelow(elk.field.n)
   >>> 1 < x < elk.field.n-1
   True
   >>> print("x =", x)
   x=  25169523658291793410518420305092143189435459919080404098218684431955762012 85
   ```

2. Apskaičiuoti viešą raktą (VR) $P = xG$:

   ```
   >>> P = x * elk.g
   ```

3. Konkretaus subjekto privatus raktas PR = x, viešas raktas VR = P:

   ```
   >>> print("Privatus raktas:", x)
   Privatus raktas:
   25169523658291793410518420305092143189435459919080404098218684431955762012 85

   >>> print("Viešas raktas:", "\npx =", P.x, "\npy =", P.y)
   Viešas raktas:
   px = 81546113548787720434932183500374788786801752272667736874724408264738276539053
   py = 26792708957152783007630244784371544686776714940212812297244812825216824385319
   ```

Toliau asimetrinei kriptografijai naudosime subjektui **Aldona** raktų porą (x, P):

```
>>> x = 25169523658291793410518420305092143189435459919080404098218684431955762012 85
>>> px = 81546113548787720434932183500374788786801752272667736874724408264738276539053
>>> py = 26792708957152783007630244784371544686776714940212812297244812825216824385319
>>> P = kreive.Point(elk, px, py)
```

Užduotys privataus ir viešo raktų poros apskaičiavimui.

Turėdami atsitiktinį skaičių (privatų raktą) ***x***, apskaičiuokite viešą raktą ***P*** ir nustatykite, kuri privataus ir viešo raktų pora iš toliau pateiktų yra teisinga:

```
1. x = 55778771337930686293732138345419960686487998206752105799643652053942981798999
2. x = 11579208921035624876269744694940757352999695522413576034242225906106851204437O
3. x = 25525682626444154399436457467899112721549351175926204253077569160274550555162
4. x = 45815502821744898809709883507454195561921754072307425411412530419400383287131Z
```

Raktų poros teisingumas ir viešieji raktai (**VR**) ***P***:

1. Teisinga raktų pora,

```
px = 1145755640413222213188932311970740020298427894445730851180078265879514726267 35
py = 5437254857733224399794189769532681894240187992147038739664484809690481404388 5
```

2. Neteisinga raktų pora,

```
px = 4843956129390645175905258525279791420276294952604174799584408071708240463528 6
py = 3613425095674979579858512791958788195661110667298501507187719825356841440510 9
```

3. Teisinga raktų pora,

```
px = 6116339413991288545384308652158538138703495216209406864010705615154721758164
py = 8588498807810676272504904762246501199263800674318462520861422270944323951027 6
```

4. Neteisinga raktų pora,

```
px = 2977075680190699579285471857751135308172079696122434416955415441603196394249 2
py = 7760033169992164996517976837063087771070906985874594241466634857252844979455 5
```

Elipsinių kreivių viešojo rakto suspaudimas

Elipsinių kreivių viešojo rakto suspaudimas yra procesas, kurio metu viešasis raktas (taškas elipsinėje kreivėje) yra užkoduojamas į trumpesnį formatą, kad užimtų mažiau vietos. Tai ypač naudinga, kai reikia išsaugoti ar perduoti viešąjį raktą sutaupant vietos (atminties), pavyzdžiui, komunikacijos protokoluose ar saugyklose.

Viešasis elipsinių kreivių raktas yra taškas $P = (x, y)$ elipsinėje kreivėje. Tačiau, kadangi elipsinės kreivės tenkina lygtį (*kitoms elipsinėms kreivėms lygtis gali skirtis*):

$$y^2 = x^3 + ax + b \bmod p,$$

todėl jei žinoma x koordinatė, kreivės *secp256r1* y (lyginė ar nelyginė) koordinatė gali būti apskaičiuota taip:

$$y = \pm\sqrt{x^3 + ax + b} \bmod p = \pm(x^3 + ax + b)^{\frac{p+1}{4}} \bmod p.$$

Dėl to:

- suspaudus viešąjį raktą užtenka saugoti tik x koordinatę ir antrosios y koordinatės ženklą (tai yra, ar y yra lyginis ar nelyginis;
- nesuspaustas viešasis raktas saugo abi koordinates (x, y).

Suspaustas viešasis raktas paprastai užima *secp256r1* (P-256) kreivei:

- 0x02 – y yra lyginis ir toliau pateikiama x koordinatė (1 + 32 baitai):

```
>>> px = 0xc1de1a4b20f1c438eabe03b44cf0ad9999f5eb69ae11ccb45b146a23d6d41c7a
>>> py = 0x957495ac057341e89f89e981c54a6a70d7f5f2783e8d3519c4dc6118cc9bcb8a
>>> P  = kreive.Point(elk, px, py)
>>> P.y % 2 == 0
True
>>> print("x =", "0x02"+str(hex(P.x)[2:]))
x = 0x02c1de1a4b20f1c438eabe03b44cf0ad9999f5eb69ae11ccb45b146a23d6d41c7a
```

- 0x03 – y yra nelyginis ir toliau pateikiama x koordinatė (1 + 32 baitai):

```
>>> px = 0xe202a12b83b478052f34247ba7a401e30ee9caa136979f013fd2f1b546d2a23d
>>> py = 0xc9ebb19f8dca6300b2d28391edba94f807edf5c6397b059d44391065f1bf7fbf
>>> P  = kreive.Point(elk, px, py)
>>> P.y % 2 == 0
False
>>> print("x =", "0x03"+str(hex(P.x)[2:]))
x = 0x03e202a12b83b478052f34247ba7a401e30ee9caa136979f013fd2f1b546d2a23d
```

- 0x04 – nesuspaustas formatas – su toliau pateikiamomis x ir y koordinatės (1 + 32 +32 baitai):

```
>>> px = 0x5a50b6c334505bf198e12c65d8b7dd46e8339cc976a14fe187747590730f52bf
>>> py = 0x8d08ad0b92350b99282fc5e9b49aa35acabc5dcbbee08d8a6dda7952b89c12c9
>>> P  = kreive.Point(elk, px, py)
>>> print("xy =", "0x04" + str(hex(P.x)[2:]) + str(hex(P.y)[2:]))
xy = 0x045a50b6c334505bf198e12c65d8b7dd46e8339cc976a14fe187747590730f52bf8d08ad0b9235
0b99282fc5e9b49aa35acabc5dcbbee08d8a6dda7952b89c12c9
```

Suspaustas formatas yra plačiai naudojamas Bitkoine ir kitose kriptovaliutose.

Kai x = 0x022f842276a29f567b83153d47b0ad878344ffd8734407505606ea691c544031e7, o y lyginė (pašalinus 02) $y = \sqrt{x^3 + ax + b}$ **mod** p:

```
>>> xh = "0x022f842276a29f567b83153d47b0ad878344ffd8734407505606ea691c544031e7"[4:]
>>> x = int(xh, 16)
>>> y_vid = (x**3 + elk.a * x + elk.b) % elk.field.p
>>> y = square_root_mod_prime(y_vid, elk.field.p)
>>> y % 2 == 0 #tikriname ar šaknis grąžino lyginę y reikšmę
>>> True
>>> #y = elk.field.p - y #kai ištraukus šaknį gautas y yra nelyginis, y paverčiame į lyginį
>>> print("y =", hex(y))
y = 0x6e926b0349950d6b84b62e0fb677bc240861553a16efe533f22df00b8536b9c2
```

Kai x = 0x03ce60af5fd2edef1944acb30186bc8583aedadcd4066b4808279247b5bb33dba8, o y nelyginė (pašalinus 03) $y = -\sqrt{x^3 + ax + b}$ **mod** p:

```
>>> xh = "0x03ce60af5fd2edef1944acb30186bc8583aedadcd4066b4808279247b5bb33dba8"[4:]
>>> x = int(xh, 16)
>>> y_vid = (x**3 + elk.a * x + elk.b) % elk.field.p
>>> y = square_root_mod_prime(y_vid, elk.field.p)
>>> print("y =", hex(y))
>>> y % 2 == 0 #tikriname ar šaknis grąžino nelyginę y reikšmę
>>> True
>>> y = elk.field.p - y #kai ištraukus šaknį gautas y yra lyginis, y paverčiame į nelyginį
>>> print("y =", hex(y))
y = 0x2d028af24113d353dc1f4de2752547fe4388365ebc593a905b19b5fefdd47a23
```

Jei ištraukus kvadratinę šaknį y gaunama priešinga koordinatė (nelyginė vietoje lyginės arba atvirkščiai), tuomet y pakeičiama į $y = p - y$.

Kai x = 0x04ea4cc5461592b2833a2c5a0b768c27a4138e77b5af5cd5b3ccb45a601e0aed9ea51df9376d96cd07e74e0bbaab5086fba431893ddd809945f671291c01cea853 (pašalinus 04):

```
>>> xy = "0x04ea4cc5461592b2833a2c5a0b768c27a4138e77b5af5cd5b3ccb45a601e0aed9ea51df9376d96cd
07e74e0bbaab5086fba431893ddd809945f671291c01cea853"[4:]
>>> x = int(xy[:64], 16)
>>> y = int(xy[64:], 16)
>>> print("x =", hex(x), "\ny =", hex(y))
x = 0xea4cc5461592b2833a2c5a0b768c27a4138e77b5af5cd5b3ccb45a601e0aed9e
y = 0xa51df9376d96cd07e74e0bbaab5086fba431893ddd809945f671291c01cea853
```

Užduotys Elipsinių kreivių viešojo rakto suspaudimui.

1. Turėdami elipsinės kreivės viešojo rakto tašką ***P***, suspauskite jį, naudodami šias reikšmes:
   1.
```
px = 0x4fa3e3c8b85359d3f99bdea76b52dc044277185194a7f455e186d7285b985f8f
py = 0x8b76674bded88e5e44f3bafc7f84bcff442b0dda82560e4055ebbb4f8829b1f2
P  = kreive.Point(elk, px, py)
```
   2.
```
px = 0x9a7c944feb51349cd7bbfe10e12bba5320d5bc6eb46643e6a0b2f2dbe63e94e6
py = 0x590b5b481d28168d11d1f3cf85cacb2d40bc86987fd44935c65b726546a9b011
P  = kreive.Point(elk, px, py)
```
   3.
```
px = 0x604e644d4acdefdf48c8c66c3b507f21d1c1e9b784e83dfc9bb4bb900c74cd75
py = 0x99f17f45a3e29fd1bc2a21a214f1338a40da591e4fe6bbbeb4e41200dd0df2fb
P  = kreive.Point(elk, px, py)
```
   4.
```
px = 0x14ab5b94837dddd28cb349e67c436f7cae605c5edf264ad3d606a9deefc893ed
py = 0xfb11bfa9904aba0adc8d96eeab6242475f45af0be30e9860c4a1e37d1c7ea8d0
P  = kreive.Point(elk, px, py)
```
   Du viešieji raktai turi lyginę *y* koordinatę, o du – nelyginę.

2. Turėdami suspausto elipsinės kreivės viešojo rakto x koordinatę, atkurkite y koordinatę, naudodami šias reikšmes:
```
1. xh = "0x03a76767da98fb483d8d4ae2115ddf4e7a9ac444ea5638128b2d5b0a6064a56ec9"
2. xh = "0x02e59dcf1751c27e66c53e6e9d4cc6985809452587386684d5eb2beebde3799f86"
3. xh = "0x032526a25ea69294969132af8bb7ebe36dabfa281500992eabdb693e5283fce9ce"
4. xh = "0x02b180ef501f3bc450aebf5bc68c6161564d3627aace42761f0567595b0affb58a"
```
   Atkurtos *y* koordinatės:
```
1. y = 0x21c79bbe96d4d8828232b43ca64a737ed0f3805ce12794727b3850d0247d1e48
2. y = 0xa6cd181abf04cbc388df1ac48ad1bf7178db6fab2e20d94d0bba76da550aeba9
3. y = 0xfc52a90c65b9a4bf77b2e6ef1b69b9a84c39ad4f9b7e364f9fb8005cbaf88bda
4. y = 0x2f5331d348b1471be49b9b7042a6947b0a29dd39b2de6050928b96f23b087aef
```

Difio Helmano raktų apsikeitimo protokolas naudojant elipsines kreives

(angl. *Elliptic-curve Diffie–Hellman key agreement protocol*, *ECDH KAP*)

Difio Helmano raktų apsikeitimo protokolas naudojant elipsines kreives (angl. trump. *ECDH KAP*) yra Difio Helmano protokolo variantas, paremtas elipsinių kreivių kriptografija, kuris pirmą kartą buvo aptartas 1987 m. Nelo Koblitzo (angl. *Neal Koblitz*) straipsnyje *Elipsinių kreivių kriptosistema*.

Difio Helmano raktų apsikeitimas (angl. trump. *DH KAP*) – tai matematinis metodas saugiai keistis kriptografiniais raktais vykdant komunikaciją viešaisiais kanalais. Be to, tai vienas pirmųjų viešojo rakto protokolų, kurį sugalvojo Ralfas Merkle (angl. *Ralph Merkle*) ir pavadino Vitfildo Difio (angl. *Whitfield Diffie*) ir Martino Helmano (angl. *Martin E. Hellman*) vardu. DH yra vienas pirmųjų praktinių viešojo rakto apsikeitimo pavyzdžių kriptografijos srityje. 1976 m. paskelbtame Difio ir Helmano darbe anksčiausiai iš viešai žinomų darbų pasiūlyta privataus rakto ir atitinkamo viešojo rakto idėja.

Aldona ir Bronius pasirinkę slaptus atsitiktinius parametrus $u \leftarrow \text{randi}(Z_{n-1})$, $v \leftarrow \text{randi}(Z_{n-1})$, apskaičiuoja viešus sesijos parametrus $T_A = u * G$, $T_B = v * G$, kuriuos siunčia per tinklą (žr. 3 pav.) vienas kitam.

3 pav. Aldona ir Bronius apsikeičia viešais sesijos parametrais

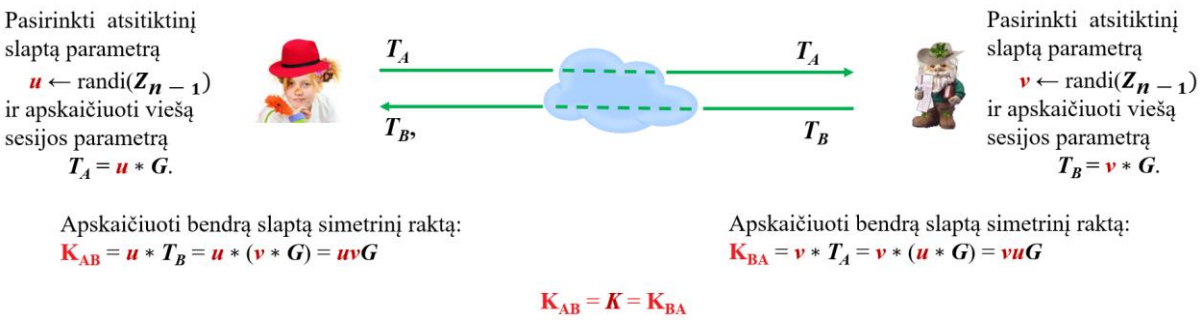
Aldona

```
>>> u = secrets.randbelow(elk.field.n)
>>> print("u=", u)
u =  110873919302759283141049761916709921998472053580178623228412006843175806786391
>>> TA = u*elk.g
>>> print("TAx =", TA.x, "\nTAy =", TA.y)
TAx = 587175392140014013953719842133569471522424343118516818046974704431534376724 6
TAy = 821650785273339280106945479083010634063274964658621220652976137791319329680 46
```

Bronius

```
>>> v = secrets.randbelow(elk.field.n)
>>> print("v=", v)
v = 592475841431904024440778186552691175123418925883505122997761472309832848697 62
>>> TB = v * elk.g
>>> print("TBx =", TB.x, "\nTBy =", TB.y)
TBx = 806029291363199246284865661568349831887552350774806809866445750442579038165 85
TBy = 274828765741211407748617985720532242287742278258939778219307009252775229675 79
```

Aldona ir Bronius gavę vienas kito viešus sesijos parametrus apskaičiuoja bendrą slaptą simetrinį raktą K apskaičiuodami $K_{AB} = u * T_B$ ir $K_{BA} = v * T_A$ **mod** p, platesni skaičiavimai pateikiami 4 pav.



4 pav. Aldona ir Bronius apskaičiuoja bendrą slaptą simetrinį raktą

Aldona apskaičiuoja bendrą slaptą simetrinį raktą

```
>>> KAB = u * TB
>>> print("KABx =", KAB.x, "\nKABy =", KAB.y)
KABx = 32042297571913042944563744987977347183080898290990697337327129467660205056909
KABy = 59563289487732931114582412334588157263953881808960901675693794831704466785223

>>> print("KABx =", hex(KAB.x), "\nKABy =", hex(KAB.y))
KABx = 0x46d74c385be51f1e806f39b3178e648515a37f84ffb24663ec90089092c8638d
KABy = 0x83afa08d3b120cb8b1b77702baf6dfd4a2e30bfc80f4f08dbdc245d91f8a03c7
```

Bronius apskaičiuoja bendrą slaptą simetrinį raktą

```
>>> KBA = v * TA
>>> print("KBAx =", KBA.x, "\nKBAy =", KBA.y)
kBAx = 32042297571913042944563744987977347183080898290990697337327129467660205056909
kBAy = 59563289487732931114582412334588157263953881808960901675693794831704466785223

>>> print("KBAx =", hex(KBA.x), "\nKBAy =", hex(KBA.y))
KBAx = 0x46d74c385be51f1e806f39b3178e648515a37f84ffb24663ec90089092c8638d
KBAy = 0x83afa08d3b120cb8b1b77702baf6dfd4a2e30bfc80f4f08dbdc245d91f8a03c7
```

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

```
KABx = 32042297571913042944563744987977347183080898290990697337327129467660205056909 = KBAx
KABy = 59563289487732931114582412334588157263953881808960901675693794831704466785223 = KBAy
Arba šešioliktaine forma:
KBAx = 0x46d74c385be51f1e806f39b3178e648515a37f84ffb24663ec90089092c8638d = KBAx
KBAy = 0x83afa08d3b120cb8b1b77702baf6dfd4a2e30bfc80f4f08dbdc245d91f8a03c7 = KBAy
```

P.S

Kadangi buvote už Bronių ir Aldoną, kad patikrintumėte, ar teisingai apskaičiavote bendrą slaptą simetrinį raktą, įsitikinkite, ar galioja lygybė $K_{AB} = K = K_{BA}$:

```
>>> KAB == KBA
True
```

Elipsinių kreivių elektroninio parašo algoritmas

Elipsinių kreivių elektroninio parašo algoritmas (angl. *Elliptic Curve Digital Signature Algorithm*, *ECDSA*) pirmą kartą buvo aptartas 1992 m. Ronaldo Rivesto (angl. *Ronal Rivest*) ir kt. straipsnyje *Atsakymai į NIST pasiūlymą*. ECDSA yra elipsinių kreivių pagrindu sukurtas skaitmeninio parašo algoritmas, kuris yra alternatyva klasikiniam skaitmeninio parašo algoritmui (angl. *Digital Signature Algorithm*, *DSA*). ECDSA užtikrina tokį patį saugumą tik su mažesniais raktų dydžiais.

Raktų ir parašų dydžiai

Kaip ir kitose elipsinių kreivių sistemose, privatusis raktas ECDSA algoritme paprastai yra apie du kartus didesnis už reikiamą saugumo lygį, bitais. Pavyzdžiui, 80 bitų saugumo lygiui reikia 160 bitų ilgio privataus rakto.

Parašo dydis tiek DSA, tiek ECDSA atveju yra maždaug **4t bitai**, kai **t** yra eksponentė iš formulės **2^t**. Tai reiškia, kad esant **80 bitų saugumo lygiui**, parašo dydis bus apie **320 bitų**.

Elipsinių kreivių palyginimas su RSA (plačiausiai naudojama kriptosistema).

RSA algoritme saugumo lygis priklauso nuo didelių pirminių skaičių ***p*** ir ***q*** daugybos sudėtingumo. Siekiant to paties saugumo lygio, RSA reikalauja daug ilgesnių raktų nei ECC. Pavyzdžiui, 80 bitų saugumo lygiui reikia apie 1024 bitų ilgio RSA rakto, tuo tarpu ECC pakanka 160 bitų privataus rakto.

1 lentelė. Elipsinių kreivių ir RSA saugumo palyginimas

Saugumas bitais	**Elipsinių kreivių rakto ilgis bitais**	**RSA rakto ilgis bitais**
80	160-223	1024
112	224-255	2048
128	256-383	3072
192	384-511	7680
256	512+	15360

Toliau naudosime subjektui **Aldona** raktų porą.

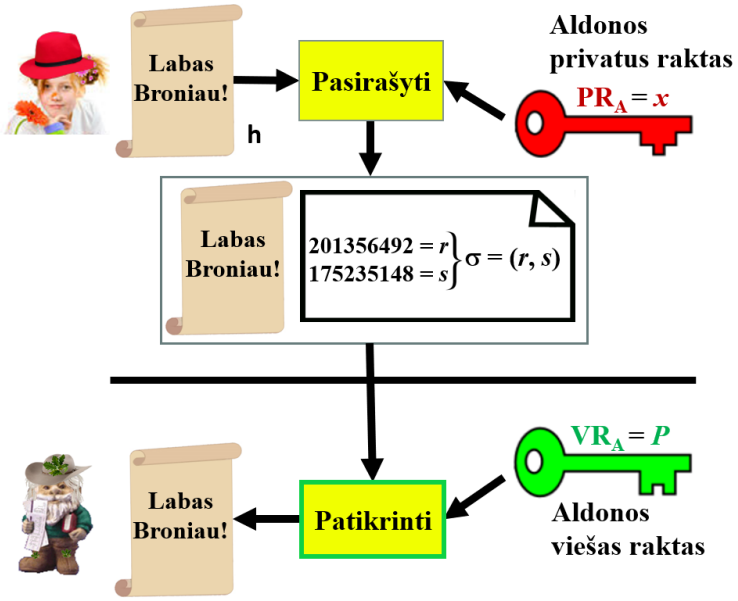
Funkcija, supaprastinanti *sha256* funkcijos naudojimą iš *hashlib* bibliotekos:

```
>>> def sha256(pranesimas):return int(santrauka.sha256(pranesimas.encode()).hexdigest(), 16)
# Po funkcijos įvedimo vieną kartą paspauskite enter
```

Aldonos pranešimas ***t***:

```
>>> t = "Labas Broniau!"
>>> print("t =", t)
t = Labas Broniau
```

Supaprastinta parašo formavimo ir tikrinimo schema pateikiama 5 pav.
Pasirašyti(PR_A, h) = ɕ = (*r*, *s*)
V = Patikrinti(VR_A, *h*, ɕ), P∈{*True*, *False*} ≡ {1, 0}



5 pav. Parašo formavimo ir jo patikrinimo schema

Aldona formuoja parašą **σ** pranešimui ***t*** su savo privačiuoju raktu (**PR**$_A$) ***x***:

1. Sugeneruokite atsitiktinį skaičių ***k*** ir patikrinkite ar tenkinama sąlyga $1 \le k \le n\text{-}1$:

```
>>> k = secrets.randbelow(elk.field.n)
>>> print("k =", k)
k = 826651727594245011212729137849243689407611700044907819857356595040888299416 11
>>> 1 < k < elk.field.n-1
True
```

2. Apskaičiuokite $kG = (x_1, y_1)$ ir pirmąją parašo komponentę $r = x_1$ **mod** n (jei $r = 0$, e. parašas formuojamas iš naujo):

```
>>> kG = k * elk.g
>>> print("kGx1 =", kG.x, "\nkGy1 =", kG.y)
kGx1 = 77755821034407097748113048338853386176238942289921860450924367662103140337209
kGy1 = 91779879744577979899327466736295161557251488791994954994055455211198592123 12
>>> r = kG.x % elk.field.n
>>> r != 0
True
>>> print("r =", r)
r = 77755821034407097748113048338853386176238942289921860450924367662103140337209
```

3. Apskaičiuokite k^{-1} **mod** n:

```
>>> km1=pow(k, -1, elk.field.n)
>>> print("km1 =", km1)
km1 = 599357062779510019757256588790331247402939981750525220352104952775950613668 02
```

4. Apskaičiuokite pranešimo ***t*** santrauką ***h*** = **H**(***t***):

```
>>> h = sha256(t)
>>> print("h =", h)
h = 33428293288344138697462033928697126385497612430434714066603703596314523426753
```

5. Apskaičiuokite antrąją parašo komponentę $\mathbf{s} = \boldsymbol{k}^{-1}(h(\mathrm{t}) + xr)$ **mod** n (jei s = 0, e. parašas formuojamas iš naujo):

```
>>> s = (km1 * (h + x * r)) % elk.field.n
>>> s != 0
True
>>> print("s =", s)
s = 158694602002908358712154400669772602898894998643125281916362673303999931563038
```

6. Parašas $\boldsymbol{\sigma} = (\boldsymbol{r},\boldsymbol{s})$ santraukai $\boldsymbol{h}$ yra
Pasirašyti($\boldsymbol{x}$, $\boldsymbol{h}$) = $\boldsymbol{\sigma} = (\boldsymbol{r},\boldsymbol{s})$ =

```
>>> print("r =", r, "\ns =", s)
r = 77755821034407097748113048338853386176238942289921860450924367662103140337209
s = 158694602002908358712154400669772602898894998643125281916362673303999931563038
```

7. **Aldona** siunčia **Broniui** parašą $\boldsymbol{\sigma}$, pranešimą $\boldsymbol{t}$.

Bronius tikrina parašą $\boldsymbol{\sigma}$ pranešimui $\boldsymbol{t}$. Parašas $\boldsymbol{\sigma}=(\boldsymbol{r},\boldsymbol{s})$ pranešimui $\boldsymbol{t}$ yra patikrinamas naudojant **Aldonos** viešą raktą ($\mathrm{VR_A}$) $\boldsymbol{P}$:

1. Patikrinkite ar r ir s yra iš intervalo $[1, n-1]$ (jei ne parašas atmetamas kaip suklastotas):

```
>>> 1 <= r <= elk.field.n-1
True
>>> 1 <= s <= elk.field.n-1
True
```

2. Apskaičiuokite pranešimo $\boldsymbol{t}$ santrauką $\boldsymbol{h'} = \mathbf{H}(\boldsymbol{t})$:

```
>>> h = sha256(t)
>>> print("h =", h)
h = 33428293288344138697462033928697126385497612430434714066603703596314523426753
```

3. Apskaičiuokite $w = s^{-1}$ **mod** n:

```
>>> w = pow(s, -1, elk.field.n)
>>> print("w =", w)
w = 79687189133701683644029171187239501296850607898355787167675319307929668862 55
```

4. Apskaičiuokite $u_1 = h(\boldsymbol{t})w$ **mod** n ir $u_2 = rw$ **mod** n:

```
>>> u1 = (h * w) % elk.field.n
>>> u2 = (r * w) % elk.field.n
>>> print("u1 =", u1, "\nu2 =", u2)
u1 = 689226494927535338426576131765681791488009270950200558053371971052174721111 64
u2 = 172715108617348180428001811039803802579377039283153501399177846171254709555 0
```

5. Apskaičiuokite $\mathrm{X} = u_1\mathrm{G} + u_2\mathrm{P} = (x_2, y_2)$ ir patikrinkite ar galioja lygybė $y^2 = x^3 + ax + b$ **mod** $\boldsymbol{p}$. (jei $X = O$, t. y. X yra be galo nutolęs taškas, tai e. parašas – suklastotas. Jei $X \neq O$, apskaičiuojamas $v = x_2$ **mod** n):

```
>>> X = u1 * elk.g + u2 * P
>>> print("x2 =", X.x, "\ny2 =", X.y)
x2 = 77755821034407097748113048338853386176238942289921860450924367662103140337209
y2 = 91779879744577979899327466736295161557251488791994954994055455211198592123 12

>>> y_2 = (X.y ** 2) % elk.field.p
```

```
>>> x3_ax_b = (X.x ** 3 + elk.a * X.x + elk.b) % elk.field.p
>>> y_2 == x3_ax_b
True

>>> v = X.x % elk.field.n
>>> print("v =", v)
v = 77755821034407097748113048338853386176238942289921860450924367662103140337209
```

6. E. parašas yra tikras tada ir tik tada, kai $v = r$:

```
>>> v == r
True ← jeigu True parašas tikras
```

Patikrinti(P, **σ**, h') = $P \in \{True, False\} \equiv \{1, 0\}$.

Tikrintojas **Bronius** priima parašą, jeigu tenkinamos visos aukščiau pateiktos sąlygos, kitais atvejais parašą atmeta.

Užduotys Elipsinių kreivių elektroniniam parašui.

Toliau naudosime subjektui **Aldona** raktų porą.

1. Turėdami privatų raktą (**PR**) ***x***, atsitiktinį skaičių ***k***, pranešimą ***t***, nustatykite, kuriam iš pokalbio metu tarp **Aldonos** ir **Broniaus** toliau pateiktų parašų **σ = (*r*,*s*)** formavimui buvo panaudotos šios reikšmės:

   1.

```
x  = 339535404239503717041302854308640398634224185791802266146066232008051972600 52
k  = 212117571210824577870701433928586987795005858138607342616543030730183342065 34
t  = "Labas Broniau!"
```

   2.

```
x  = 622501924131245107523401169213961669645948345444707666427675925974199268587 48
k  = 103362166037498859475146270258536529587612490403856080069080535408473303922521
t  = "Labas Aldona!"
```

   3.

```
x  = 101298727404956896140883411101971900898630369338194700342362607834155193235526
k  = 824340081587767200480496807573284557440183932677874820439000942619692345513 1
t  = "Kada galėtume susitikti."
```

   4.

```
x = 210766868055858811662668373914233838645590662488133243902437225292592590272 96
k = 169496356680232031012361545804864446548675753141633731265740128388424454484 40
t = "Susitikime vakare."
```

   Parašai **σ = (*r*,*s*)**:

   1.

```
r  = 984422122011680524644783829397664625155892521383477235227101132636567855262 53
s  = 106675467894595800635116525069247039178056439152888264198127370744038715961 41
```

   2.

```
r  = 603347013929797241452933136968374286735422938486856413816457205525556428823 42
s  = 106675467894595800635116525069247039178056439152888264198127370744038715961 41
```

   3.

```
r  = 727216519755373952882642633039447985907301570308870065611905510715627730057 54
s  = 509005338818078274148141681040512372135144999314204128163404633414680579930 33
```

   4.

```
r  = 739904697863576295131645239975493981024570917793568935501546200594143883269 63
s  = 752817449113901849842219360670245044303385346843638944331618103392528522787 69
```

   5.

```
r  = 984422122011680524644783829397664625155892521383477235227101132636567855262 53
s  = 509005338818078274148141681040512372135144999314204128163404633414680579930 33
```

   6.

```
r  = 603347013929797241452933136968374286735422938486856413816457205525556428823 42
s  = 689932059098656113227482887753089417973769172084858991725592665375256064514 11
```

   7.

```
r  = 727216519755373952882642633039447985907301570308870065611905510715627730057 54
s  = 689932059098656113227482887753089417973769172084858991725592665375256064514 11
```

   8.

```
r  = 739904697863576295131645239975493981024570917793568935501546200594143883269 63
s  = 752817449113901849842219360670245044303385346843638944331618103392528522787 69
```

2. Turėdami viešą raktą (**VR**) ***P***, pranešimą ***t***, parašą **σ = (*r*,*s*)**, nustatykite, kuriems **Aldonos** ir **Broniaus** pranešimams suformuoti parašai yra galiojantys, panaudojant šias reikšmes:

1.

```
px = 1155426930795598228320440965985000256559943870236426142713409652637167964591 69
py = 2122822287049023902130882210356215191211823229506550761449441228316446678334 6
P  = kreive.Point(elk, px, py)
t  = "Šalia seno ąžuolo."
r  = 879485266249216570695056849301699738933529860351777191617548532625843034105 57
s  = 573144322537528293875101957038985952803590465238186358935379063211430223203199
```

2.

```
px = 613168308002434370278896922857798411254190207914487783217443960361181176205 36
py = 998478005062512588868307252313696808266549317949550500290271452606303950936 18
P  = kreive.Point(elk, px, py)
t  = "Šalia didelio kelmo."
r  = 596941634401934107130065535300979459160728948790095788103729463757945267939 93
s  = 276846563699983375258029686777702699877005185058715979998300649307202229734 09
```

3.

```
px = 186720427264994038722921189484640416257120582913382729886336092958207443951 25
py = 320478541619563813615558905129090349794413700526201753438350435206826460849 02
P  = kreive.Point(elk, px, py)
t  = "Iki greito."
r  = 106551258383783590879379856650915420894087639883558844906778695031044525785735
s  = 106483267458862694725267867545266964959623870581276421885145377225480376529658
```

4.

```
px = 404505841653051744398834444657403241737562278279765043238933955280670422024 46
py = 410046253585991098868134249466752338410138409003542805367122833030715632132 03
P  = kreive.Point(elk, px, py)
t  = "Iki pasimatymo."
r  = 113655098800224831507701417363809798142881788147751429845922836631785724516399
s  = 725020411948233253973394105983937676450415846613001048957478130730395260297 24
```

Tik du parašai **σ = (*r*,*s*)** galioja.

3. Turėdami viešą raktą (**VR**) ***P*** ir parašą **σ = (*r*,*s*)**, nustatykite, kuriems **Aldonos** ir **Broniaus** pranešimams ***t*** buvo suformuotas parašas, panaudojant pateiktas reikšmes:

1.

```
px = 40450584165305174439883444465740324173756227827976504323893395528067042202446
py = 41004625358599109886813424946675233841013840900354280536712283303071563213203
P  = kreive.Point(elk, px, py)
r  = 66150586982165713844094658100013425958446156880262390299613109427504652060172
s  = 49300567341085075449096676023700387451725824974855234317394859706519230426093
```

2.

```
px = 18672042726499403872292118948464041625712058291338272988633609295820744395125
py = 32047854161956381361555890512909034979441370052620175343835043520682646084902
P  = kreive.Point(elk, px, py)
r  = 90883113098721183787270231874178240167477118056311160800532864565001248024795
s  = 37041248185411831820636440813538985514507235842145535237887070075773705411574
```

3.

```
px = 61316830800243437027889692285779841125419020791448778321744396036118117620536
py = 99847800506251258886830725231369680826654931794955050029027145260630395093618
P  = kreive.Point(elk, px, py)
r  = 75650549904985110955262282860290120955451785344100212159191916401538343646232
s  = 48823711485289235488634170928019046529150575028893409257138689980439061484 79
```

4.

```
px = 11554269307955982283204409659850002565599438702364261427134096526371679645916 9
py = 21228222870490239021308822103562151912118232295065507614494412283164466783346
P  = kreive.Point(elk, px, py)
r  = 79517987049720958287638294459614952088547107991316178271706277803240917192 77
s  = 36403805725235902238467565097140625544997019862435997734689037402506632825599
```

Pranešimai ***t***:

1. t_1 = "Kelintą valandą vakare.";
2. t_2 = "19 valandą.";
3. t_3 = "Kurioje vietoje.";
4. t_4 = "Jaukioje parko kavinėje.".

Simetrinis šifravimas ir iššifravimas naudojant Difio Helmano autentifikuotą raktų apsikeitimo protokolą su Elipsinėmis kreivėmis ir AES-GCM

Šiame skyriuje aprašoma, kaip naudoti autentifikuotą Difio-Helmano raktų apsikeitimo protokolą (angl. trump. *ADH KAP*), kad būtų sugeneruotas bendras slaptas raktas, skirtas AES-GCM šifravimui. Naudojant ADH KAP, šalys apsikeičia viešaisiais parametrais, apskaičiuoja bendrą simetrinį raktą ir jį naudoja Vernamo šifravimui, užtikrinant saugų žinučių užšifravimą ir iššifravimą.

Autentifikuotas Difio Helmano raktų apsikeitimo protokolas su ECDSA parašais

(angl. *Authenticated Elliptic-curve Diffie–Hellman key agreement protocol, AECDH KAP*)

Autentifikuotas Difio-Helmano raktų apsikeitimas (angl. *Authenticated Diffie-Hellman Key Agreement Protocol*, *ADH KAP*) – tai matematinis metodas, leidžiantis dviem ar daugiau šalių saugiai susitarti dėl bendro kriptografinio rakto, net ir vykdant komunikaciją per nesaugius viešuosius kanalus. Šis protokolas yra patobulinta Difio-Helmano raktų apsikeitimo protokolo versija, kuri įtraukia autentifikaciją, kad būtų užkirstas kelias atakoms, tokioms kaip *Žmogaus viduryje ataka* (angl. *Man-in-the-Middle Attack*). Autentifikacija užtikrina, kad abi šalys tikrai komunikuoja viena su kita, o ne su priešu, naudodamos skaitmeninius parašus ar kitus patikimumo mechanizmus.

Originalus Difio-Helmano protokolas buvo pasiūlytas 1976 m. ir leido saugiai keistis raktais, tačiau neturėjo autentifikacijos, todėl buvo pažeidžiamas atakoms. 1992 m. Vitfildas Difis ir kt. pasiūlė *Station-to-Station* (angl. trump. STS) protokolą – pirmąjį autentifikuotą raktų apsikeitimo metodą, kuris išsprendė šią problemą. Šis protokolas tapo pagrindu daugeliui šiuolaikinių saugių komunikacijų sistemų, tokių kaip IPsec, RSA pagrįstas TLS/SSL, Elipsinėmis kreivėmis pagrįstas ECMQV protokolas ir kt.

Aldona ir **Bronius** kartu su elipsinės kreivės parametrais iš anksto turi žinoti vienas kito ECC viešą raktą. **Aldonos** (x_A ir P_A) ir **Broniaus** (x_B ir P_B) ECC raktų poros:

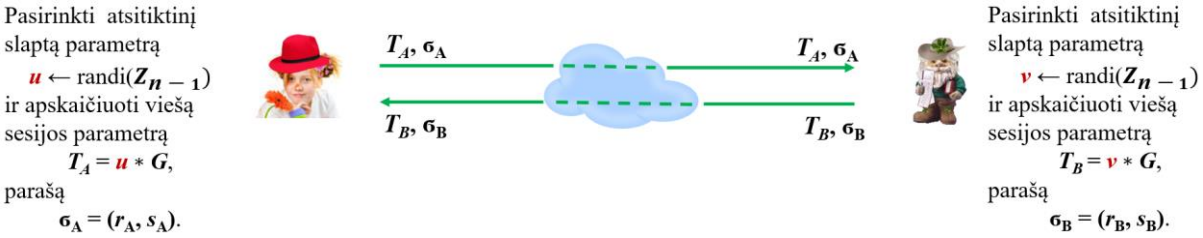
```
>>> xA = 708910953140955399377323767517550948424616764664562570378630643408108547999 68
>>> pxA = 691884626773785407224693737275472825530296847072804051685832362329392696106 84
>>> pyA = 700108848936096844749222546753474371187818396314914869294417369338814403436 79
>>> PA = kreive.Point(elk, pxA, pyA)

>>> xB = 534316707661617913285650807009530528045567153295580019452294418261311205595 0
>>> pxB = 101420005963710390919450344978684598693346911361452098437102584203374475353886
>>> pyB = 606236652113001661418136718788022118260252995550417448832510603657481846074 3
>>> PB = kreive.Point(elk, pxB, pyB)
```

Funkcija, supaprastinanti *sha256* funkcijos naudojimą iš *hashlib* bibliotekos:

```
>>> def sha256(pranesimas):return int(santrauka.sha256(pranesimas.encode()).hexdigest(), 16)
# Po funkcijos įvedimo vieną kartą paspauskite enter
```

Bendro slapto simetrinio rakto apskaičiavimas

Aldona ir **Bronius** pasirinkę slaptus atsitiktinius parametrus u, v, apskaičiuoja viešus sesijos parametrus $T_A = g^u \bmod p$, $T_B = g^v \bmod p$, kuriems suformuoja parašus $\sigma_A = (r_A, s_A)$ ir $\sigma_B = (r_B, s_B)$ ir vienas kitam išsiunčia viešus sesijos parametrus kartu su parašais per tinklą, kaip vaizduojama 6 pav.



6 pav. Aldona ir Bronius apsikeičia viešais sesijos parametrais ir parašais

Aldona

```
>>> u = secrets.randbelow(elk.field.n)
>>> print("u =", u)
u = 67010935565937631520176879619233976066320168844821675348793659081239711877 00
>>> TA = u * elk.g
>>> print("TAx =", TA.x, "\nTAy =", TA.y)
TAx = 49368944904280402688519412730786139167150269052610524661589295448338945213091
TAy = 92962741727809227162590303304420413776195981840907342107262981923106237142 14
```

Bronius

```
>>> v = secrets.randbelow(elk.field.n)
>>> print("v =", v)
v = 53904538474129125776310327915718345606668085273837472612533047417204003761381
>>> TB = v * elk.g
>>> print("TBx =", TB.x, "\nTBy =", TB.y)
TBx = 17393253845469536218770873392800522821428717858565058972897209647031598580947
TBy = 54200035079596416301913865445777252516851734492872136869541263770084096121367
```

Aldona ir **Bronius** formuoja parašus σ_A ir σ_B viešiems sesijos parametrams T_A ir T_B su savo privačiais raktais x_A ir x_B:

1. Sugeneruokite atsitiktinius skaičius k_A ir k_B ir patikrinkite ar tenkinama sąlyga $1 \le k_A, k_B \le n\text{-}1$:

```
>>> kA = secrets.randbelow(elk.field.n)
>>> print("kA =", kA)
kA = 80150522175081402715690497818169343253570724065609507009066335474247906079332
>>> 1 < kA < elk.field.n-1
True

>>> kB = secrets.randbelow(elk.field.n)
>>> print("kB =", kB)
kB = 10018094357816081110595661043677473889041527866864927633858183601365681634 0019
>>> 1 < kB < elk.field.n-1
True
```

2. Apskaičiuokite $k_A G = (x_{A1}, y_{A1})$ ir $k_B G = (x_{B1}, y_{B1})$ ir pirmąsias parašo komponentes $r_A = x_{A1} \bmod n$ ir $r_B = x_{B1} \bmod n$ (jei $r_A = 0$ ar $r_B = 0$, e. parašas formuojamas iš naujo):

```
>>> kAG = kA * elk.g
>>> print("kAGx1 =", kAG.x, "\nkAGy1 =", kAG.y)
kAGx1 = 533754767452409236599369865482667929342308226357922969192583731183388780219999
kAGy1 = 96388243651634776422054491221120360820753198267631562363948045090764466200 32

>>> kBG = kB * elk.g
>>> print("kBGx1 =", kBG.x, "\nkBGy1 =", kBG.y)
kBGx1 = 430008856070997154790255550703015152158588425073542500825822541301331779136 14
kBGy1 = 100838061317515552520534038529718919293859826939388315078667350199078650699 7764

>>> rA = kAG.x % elk.field.n
>>> rA != 0
True
>>> print("rA =", rA)
rA = 533754767452409236599369865482667929342308226357922969192583731183388780219999

>>> rB = kBG.x % elk.field.n
>>> rB != 0
True
>>> print("rB =", rB)
rB = 430008856070997154790255550703015152158588425073542500825822541301331779136 14
```

3. Apskaičiuokite $k_A^{-1} \bmod n$ ir $k_B^{-1} \bmod n$:

```
>>> kAm1 = pow(kA, -1, elk.field.n)
>>> print("kAm1 =", kAm1)
kAm1 = 715264645720271159929374225463428843077259735521343822845522526685053038965 13

>>> kBm1 = pow(kB, -1, elk.field.n)
>>> print("kBm1 =", kBm1)
kBm1 = 106722255999046400830317853550158621978433462896027507287785744013231791507740
```

4. Apskaičiuokite viešų sesijos parametrų T_A ir T_B santraukas $h_A = H(T_A)$ ir $h_B = H(T_B)$:

```
>>> hA = sha256(str(TA))
>>> print("hA =", hA)
hA = 458977178214344557700669008211778260998404410051479263246766008976569019216 20

>>> hB = sha256(str(TB))
>>> print("hB =", hB)
hB = 395253112662842559810042414866225184279131039712437898533417368363368829026 12
```

5. Apskaičiuokite antrąsias parašo komponentes $s_A = k_A^{-1} (h(t_A) + x_A r_A) \bmod n$ ir $s_B = k_B^{-1} (h(t_B) + x_B r_B) \bmod n$ (jei $s_A = 0$ ar $s_B = 0$, e. parašas formuojamas iš naujo):

```
>>> sA = (kAm1 * (hA + xA * rA)) % elk.field.n
>>> sA != 0
True
```

```
>>> print("sA =", sA)
sA = 92505444051882421911199601047667226146508091237094489001874347469262593668452

>>> sB = (kBm1 * (hB + xB * rB)) % elk.field.n
>>> sB != 0
True
>>> print("sB =", sB)
sB = 79243937009549914038841460043507203476789621095239657862373835390956390788905
```

6. Parašas $\sigma_A = (r_A, s_A)$ santraukai h_A yra
   Pasirašyti $(x_A, h_A) = \sigma_A = (r_A, s_A) =$

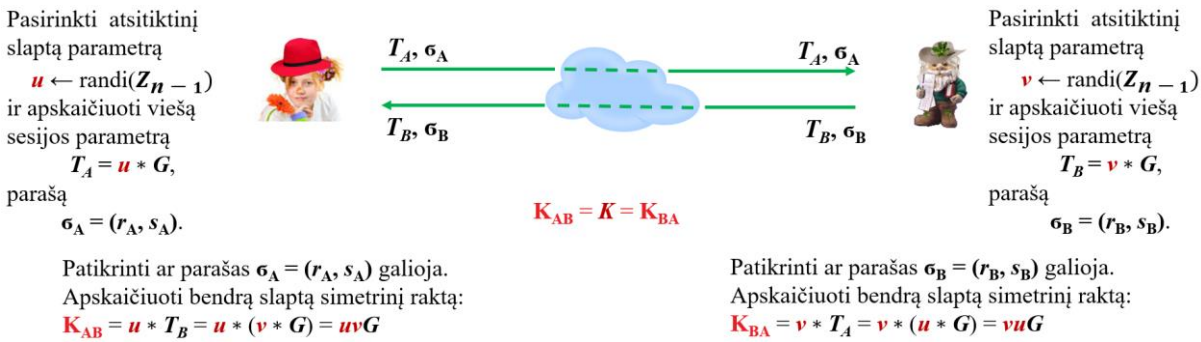
```
>>> print("rA =", rA, "\nsA =", sA)
rA = 533754767452409236599369865482667929342308226357922969192583731183388780219 99
sA = 92505444051882421911199601047667226146508091237094489001874347469262593668452
```

   Parašas $\sigma_B = (r_B, s_A)$ santraukai h_B yra
   Pasirašyti $(x_A, h_A) = \sigma_B = (r_B, s_B) =$

```
>>> print("rB =", rB, "\nsB =", sB)
rB = 43000885607099715479025555070301515215858842507354250082582254130133177913614
sB = 79243937009549914038841460043507203476789621095239657862373835390956390788905
```

7. **Aldona** siunčia **Broniui** viešą sesijos parametrą T_A ir parašą σ_A, o **Bronius** siunčia **Aldonai** viešą sesijos parametrą T_B ir parašą σ_B.

Aldona ir **Bronius** gavę vienas kito viešus sesijos parametrus T_A, T_B ir parašus σ_A, σ_B, pirmiausia patikrina ar parašai galioja, ir parašams galiojant apskaičiuoja bendrą slaptą simetrinį raktą K, apskaičiuodami $K_{AB} = u * T_B$ ir $K_{BA} = v * T_A$, platesni skaičiavimai pateikiami 7 pav.



7 pav. Aldona ir Bronius patikrina parašus ir apskaičiuoja bendrą slaptą simetrinį raktą

Bronius ir **Aldona** tikrina parašus σ_A ir σ_B viešiems sesijos raktams T_A ir T_B. Parašai σ_A ir σ_B patikrinami naudojant **Aldonos ir Broniaus** viešąjį raktą P_A ir P_B:

1. Patikrinkite ar r_A ir s_A, r_B ir s_B yra iš intervalo $[1, n - 1]$ (jei ne parašas atmetamas kaip suklastotas):

```
>>> 1 <= rA <= elk.field.n-1
True
>>> 1 <= sA <= elk.field.n-1
True
>>> 1 <= rB <= elk.field.n-1
True
>>> 1 <= sB <= elk.field.n-1
True
```

2. Apskaičiuokite viešų sesijos parametrų T_A ir T_B santraukas $h_A = H(T_A)$ ir $h_B = H(T_B)$:

```
>>> hA = sha256(str(TA))
>>> print("hA =", hA)
hA = 458977178214344557700669008211778260998404410051479263246766008976569019216 20

>>> hB = sha256(str(TB))
>>> print("hB =", hB)
hB = 39525311266284255981004241486622518427913103971243789853341736836336882902612
```

3. Apskaičiuokite $w_A = s_A^{-1} \bmod n$ ir $w_B = s_A^{-1} \bmod n$:

```
>>> wA = pow(sA, -1, elk.field.n)
>>> print("wA =", wA)
wA = 807169872101368911206057819107456845404607967021814665042682894535130836403 55

>>> wB = pow(sB, -1, elk.field.n)
>>> print("wB =", wB)
wB = 419184942850951811983170424949598089575633233272991894459716291686582312837 24
```

4. Apskaičiuokite $u_{A1} = h(t_A)w_A \bmod n$ ir $u_{A2} = r_A w_A \bmod n$ ir $u_{B1} = h(t_B)w_B \bmod n$ ir $u_{B2} = r_B w_B \bmod n$:

```
>>> uA1 = (hA * wA) % elk.field.n
>>> uA2 = (rA * wA) % elk.field.n
>>> print("uA1 =", uA1, "\nuA2 =", uA2)
uA1 = 85329190872389148662999885562940160381458750092583700910188095953373089559454
uA2 = 40259114571468227518729674868786480794425313865145203584352974517122695289178

>>> uB1 = (hB * wB) % elk.field.n
>>> uB2 = (rB * wB) % elk.field.n
>>> print("uB1 =", uB1, "\nuB2 =", uB2)
uB1 = 80893921733572551367380194159741575987119379892400821981706112930015516086668
uB2 = 38117694283336313534886754711356820401506647373840984661415430454372176352
```

5. Apskaičiuokite $X_A = u_{A1}G + u_{A2}P = (x_{A2}, y_{A2})$, $X_B = u_{B1}G + u_{B2}P = (x_{B2}, y_{B2})$ ir patikrinkite ar tenkinama lygybė $y^2 = x^3 + ax + b \bmod p$. (jei $X_A = O$ ar $X_B = O$, t. y. X_A ar X_B yra be galo nutolęs taškas, tai e. parašas – suklastotas. Jei $X_A \neq O$ ar $X_B \neq O$, apskaičiuojamas $v_A = x_{A2} \bmod n$, $v_B = x_{B2} \bmod n$):

```
>>> XA = uA1 * elk.g + uA2 * PA
>>> print("xA2 =", XA.x, "\nyA2 =", XA.y)
xA2 = 53375476745240923659936986548266792934230822635792296919258373118338878021999
yA2 = 96388243651634776422054491221120360820753198267631562363948045090764466200 32

>>> XB = uB1 * elk.g + uB2 * PB
>>> print("xB2 =", XB.x, "\nyB2 =", XB.y)
xB2 = 43000885607099715479025555070301515215858842507354250082582254130133177913614
yB2 = 10083806131751552520534038529718919293859826939388315078667350199078650699 7764
```

```
>>> yA_2 = (XA.y ** 2) % elk.field.p
>>> xA3_axA_b = (XA.x ** 3 + elk.a * XA.x + elk.b) % elk.field.p
>>> yA_2 == xA3_axA_b
True

>>> yB_2 = (XB.y ** 2) % elk.field.p
>>> xB3_axB_b = (XB.x ** 3 + elk.a * XB.x + elk.b) % elk.field.p
>>> yB_2 == xB3_axB_b
True

>>> vA = XA.x % elk.field.n
>>> print("vA =", vA)
vA = 533754767452409236599369865482667929342308226357922969192583731183388780219999

>>> vB = XB.x % elk.field.n
>>> print("vB =", vB)
vB = 430008856070997154790255550703015152158588425073542500825822541301331779136l4
```

6. E. parašas yra tikras tada ir tik tada, kai $v_A = r_A$, $v_B = r_B$:

```
>>> vA == rA
True ← jeigu True Aldonos parašas tikras

>>> vB == rB
True ← jeigu True Broniaus parašas tikras
```

Patikrinti(*P*, **σ**, ***h′***) = *P*∈{*True*, *False*}≡{1, 0}.

Tikrintoja **Aldona** ir tikrintojas **Bronius** priima parašą, jeigu tenkinamos visos aukščiau pateiktos sąlygos, kitais atvejais parašą atmeta.

Aldona apskaičiuoja bendrą slaptą simetrinį raktą

```
>>> KAB = u * TB
>>> print("KABx =", KAB.x, "\nKABy =", KAB.y)
KABx = 654334007434251643115835720932456132232266199094730473089017085619708827729 7
KABy = 107329128405710264079035690258463770806401434513355898882494920675794215884917

>>> print("KABx =", hex(KAB.x), "\nKABy =", hex(KAB.y))
KABx = 0xe7766347f76ddda35596bcbb3c5716bbd74acf831681c2410fe92f20506f331
KABy = 0xed4a22484dc631a048e2542ae109e42ced0c5214e74b73ab81fd969a83c1d875
```

Bronius apskaičiuoja bendrą slaptą simetrinį raktą

```
>>> KBA = v * TA
>>> print("KBAx =", KBA.x, "\nKBAy =", KBA.y)
KBAx = 654334007434251643115835720932456132232266199094730473089017085619708827729 7
KBAy = 107329128405710264079035690258463770806401434513355898882494920675794215884917
```

```
>>> print("KBAx =", hex(KBA.x), "\nKBAy =", hex(KBA.y))
KBAx = 0xe7766347f76ddda35596bcbb3c5716bbd74acf831681c2410fe92f20506f331
KBAy = 0xed4a22484dc631a048e2542ae109e42ced0c5214e74b73ab81fd969a83c1d875
```

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

```
KABx = 654334007434251643115835720932456132232266199094730473089017085619708827729 7   = KBAx
KABy = 107329128405710264079035690258463770806401434513355898882494920675794215884917 = KBAy
Arba šešioliktaine forma:
KBAx = 0xe7766347f76ddda35596bcbb3c5716bbd74acf831681c2410fe92f20506f331  = KBAx
KBAy = 0xed4a22484dc631a048e2542ae109e42ced0c5214e74b73ab81fd969a83c1d875 = KBAy
```

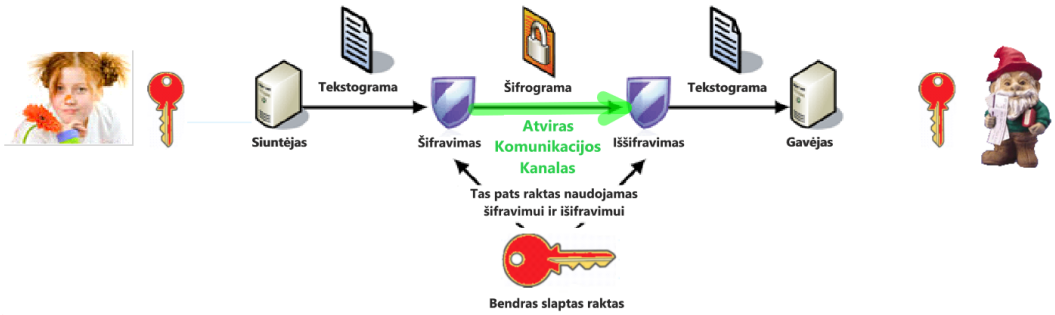
P.S

Kadangi buvote už Bronių ir Aldoną, kad patikrintumėte, ar teisingai apskaičiavote bendrą slaptą simetrinį raktą, įsitikinkite, kad galioja lygybė $K_{AB} = K = K_{BA}$:

```
>>> KAB == KBA
True
```

Simetrinis šifravimas ir iššifravimas naudojant AES-GCM

Aldona užšifruoja pranešimą *Labas Broniau!* ir siunčia šifrogramą *Ch* Broniui. Bronius gavęs šifrogramą ją iššifruoja ir perskaito pranešimą. Komunikacijos schema naudojant bendrą slaptą simetrinį raktą ***k*** (gali būti ECC *x* arba *y* koordinatė arba jų tam tikras derinys) pateikiama 8 pav.



8 pav. Aldonos ir Broniaus komunikacija naudojant bendrą slaptą simetrinį raktą

Toliau naudosime AES-GCM (angl. *Advanced Encryption Standard - Galois/Counter Mode*) – simetrinio rakto šifravimo algoritmas, kuris užtikrina duomenų konfidencialumą ir vientisumo patikrinimą (autentiškumą). Šis algoritmas derina AES (blokinį šifrą) su GCM režimu, kuris naudoja IV (inicijavimo vektorių), kad būtų sugeneruojami (naudojant pastovų bendrą slaptą simetrinį raktą) unikalūs šifravimo raktai kiekvienam šifruojamam duomenų blokui. Taip pat, naudojama autentifikacijos žyma (angl. *tag*), užtikrinant, kad duomenys nebuvo pakeisti ar sugadinti perduodant.

```
>>> from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
>>> from cryptography.hazmat.primitives.kdf.hkdf import HKDF
>>> from cryptography.hazmat.primitives import hashes
>>> import os
```

Aldona šifruoja pranešimą

```
>>> k = KAB.x.to_bytes(32, "big") #konvertuoti į big-endian formatą
>>> pranesimas = b"Labas Broniau!"

>>> simetrinis_raktas = HKDF(algorithm=hashes.SHA256(), length=32, salt=None,
info=b"encryption_key").derive(k)
>>> iv = os.urandom(16)
>>> enkriptorius = Cipher(algorithms.AES(simetrinis_raktas), modes.GCM(iv)).encryptor()
>>> sifrograma = enkriptorius.update(pranesimas) + enkriptorius.finalize()
>>> zyme = enkriptorius.tag

>>> print("Simetrinis raktas =", simetrinis_raktas.hex())
Simetrinis raktas = 953ec9c874fad129f8c865b6f48421e3896dc1f559fb3b3c36989c03728d51da
>>> print("IV =", iv.hex(), "\nŽymė =", zyme.hex(), "\nŠifrograma =", sifrograma.hex())
IV = 15322dc7d1ecaeafcf4d792a778879fe
Žymė = 7ea3661a30f411e26f3ad0d82b4b5ef6
Šifrograma = 8185ea469ab9c06bade026a9d649
```

Aldona išsiunčia Broniui IV, žymę, šifrogramą.

Bronius iššifruoja šifrogramą

Bronius gavęs iš Aldonos IV, žymę, šifrogramą iššifruoja ir perskaito pranešimą.

```
>>> k = KBA.x.to_bytes(32, "big") #konvertuoti į big-endian formatą
>>> simetrinis_raktas = HKDF(algorithm=hashes.SHA256(), length=32, salt=None,
info=b"encryption_key").derive(k)
>>> iv = bytes.fromhex("15322dc7d1ecaeafcf4d792a778879fe")
>>> zyme = bytes.fromhex("7ea3661a30f411e26f3ad0d82b4b5ef6")
>>> sifrograma = bytes.fromhex("8185ea469ab9c06bade026a9d649")
>>> dekriptorius = Cipher(algorithms.AES(simetrinis_raktas), modes.GCM(iv,zyme)).decryptor()
>>> pranesimas = dekriptorius.update(sifrograma) + dekriptorius.finalize()
>>> print("Gautas pranesimas =", pranesimas.decode('utf-8'))
Gautas pranesimas = Labas Broniau!
```

Užduotys Autentifikuotam Difio Helmano raktų apsikeitimo protokolui naudojant elipsines kreives.

Autentifikuoto Difio Helmano raktų apsikeitimo protokolo simuliacija naudojant elipsines kreives.

Homomorfinė elipsinių kreivių savybė

Homomorfiškumas yra matematinių struktūrų savybė, leidžianti atlikti operacijas su užšifruotais duomenimis taip, kad šios operacijos atitinka tas pačias operacijas, kaip ir su neapdorotais duomenimis pvz. multiplikatyvus homomorfiškumas 2 • 3 užšifravus $\mathbf{c_1}$ ir $\mathbf{c_2} \rightarrow \mathbf{c_{12}} = \mathbf{c_1} \bullet \mathbf{c_2} \rightarrow \mathbf{c_{12}}$ iššifravus = 6 (*ElGamalio* ir kt. šifrai); adityvus homomorfiškumas 1 + 2 užšifravus $\mathbf{c_1}$ ir $\mathbf{c_2} \rightarrow \mathbf{c_{12}} = \mathbf{c_1} \bullet \mathbf{c_2} \rightarrow \mathbf{c_{12}}$ iššifravus = 3 (Pailer ir kt. šifrai), 2 + 3 užšifravus $\mathbf{c_1}$ ir $\mathbf{c_2} \rightarrow \mathbf{c_{12}} = \mathbf{c_1} + \mathbf{c_2} \rightarrow \mathbf{c_{12}}$ iššifravus = 5 (Elipsinės kreivės ir kt. šifrai). Kitaip tariant, jei turime dvi struktūras (pvz., grupes, žiedus), homomorfiškumas leidžia perkelti operacijas iš vienos struktūros į kitą, išlaikant jų prasmę.

Elipsinės kreivės, pasižymi adityviai homomorfine savybe, kuri leidžia atlikti operacijas su užšifruotais duomenimis taip, kad iššifravus rezultatą gaunamas teisingas atsakymas, tarsi skaičiavimai būtų atlikti tiesiogiai su pradiniais, nešifruotais duomenimis. Naudojantis Elipsinių kreivių adityviu homomorfiškumu, sumą $1 + 2 + \cdots + n$ galima įgyvendinti sudedant užšifruotų skaičių šifrogramas, o iššifravus rezultatą gaunama teisinga suma pvz. 2 + 3 užšifravus gaunama $\mathbf{c_1}$ ir $\mathbf{c_2} \rightarrow \mathbf{c_1} + \mathbf{c_2} = \mathbf{c_{12}}$ ir yra $\mathbf{c_{12}}$, o iššifravus = 5.

Supaprastintas elipsinių kreivių homomorfiškumo pavyzdys su imituojamais atskleistais, ankščiau paslėptais duomenimis $2G + 3G = 5G$:

```
>>> C1 = 3 * elk.g
>>> print("C1x =", C1.x, "\nC1y =", C1.y)
C1x = 428776569712758113102625648944902100247592871821771961624253491316759467124 28
C1y = 611548011120142145041782814619925700172471720047042770416810939275696037765 62
>>> C2 = 2 * elk.g
>>> print("C2x =", C2.x, "\nC2y =", C2.y)
C2x = 565152197906911714131090579040116886954248101558029299735264813213098562420 40
C2y = 337703184371225825922371145149145259808867551975154856711245809463549758356 9

>>> C12_ = 5 * elk.g
>>> print("C12_x =", C12_.x, "\nC12_y =", C12_.y)
C12_x = 367946693408968830121014734395389297591523964766486925917953181940545801553 73
C12_y = 101659946828913883886577915207667153874746613498030835602133042203824767462 820

>>> C12 = C1 + C2
>>> print("C12x =", C12.x, "\nC12y =", C12.y)
C12x = 367946693408968830121014734395389297591523964766486925917953181940545801553 73
C12y = 101659946828913883886577915207667153874746613498030835602133042203824767462 820

>>> C12 == C12_
True
```

Homomorfinis šifravimo principas naudojamas elektroniniame balsavime, duomenų debesijos privačiuose skaičiavimuose, finansinėse operacijose ir auditui, medicininiuose tyrimuose su jautriais duomenimis.

Užduotys Elipsinių kreivių homomorfiškumui.

1. Turėdami atskleistas vertes, kurios buvo panaudotos skaičiuojant $\mathbf{c_1}$, ..., $\mathbf{c_n}$, remdamiesi elipsinių kreivių homomorfiškumo savybe patikrinkite ar gautas taškas $\mathbf{c_{1n}}$ yra teisingas, panaudojant pateiktas reikšmes:

1.

```
C1 = 5 * elk.g
C2 = 4 * elk.g
C3 = 6 * elk.g
C4 = 3 * elk.g

C14_x = 525210041856415366272666005368048169315353291333555399620209801938023830578600
C14_y = 336532199988672138926993714427671109158562719686560581596931230187280744494700
C14_ = kreive.Point(elk, C14_x, C14_y)
```

2.

```
C1 = 3 * elk.g
C2 = 5 * elk.g
C3 = 8 * elk.g

C13_x = 536720452480578891154429906509742469561114335099492278171544474989885067582200
C13_y = 766776177343635092490148195912266214048004711225470846035251986738294174722720
C13_ = kreive.Point(elk, C13_x, C13_y)
```

3.

```
C1 = 2 * elk.g
C2 = 8 * elk.g

C12_x = 872347893016426931444867660906786790918255107500695658815443688554339609136590
C12_y = 590382227813318047306555869794241520053473980091565205888955376074028363889720
C12_ = kreive.Point(elk, C12_x, C12_y)
```

4.

```
C1 = 4 * elk.g
C2 = 9 * elk.g
C3 = 3 * elk.g
C4 = 5 * elk.g
C5 = 6 * elk.g

C15_x = 109968170958995759751103689520893366273090918277189655778866193017531169842730
C15_y = 871381237950746173010695135222729066077220015696852083709482480147801686954070
C15_ = kreive.Point(elk, C15_x, C15_y)
```

Tik du $\mathbf{c_{1n}}$ taškai apskaičiuoti teisingai.

Pedersono įsipareigojimas naudojant elipsines kreives

Siekiant tiksliau aptarti pedersono įsipareigojimus pasiremsime *MimbleWimble* konfidencialių ir patikrintinų operacijų sistemos dokumentacija.

Įsipareigojimas

Įsipareigojimo schema (angl. *commitment scheme*) yra kriptografinis primityvas, leidžiantis įsipareigoti pasirinktai vertei, išlaikant ją paslaptyje nuo kitų, su galimybe vėliau atskleisti įsipareigotą vertę.

Savybės:

- φ slaptumas (angl. *hiding*) – niekas, išskyrus įsipareigojusį, negali matyti ar nustatyti tikrosios įsipareigotos vertės;
- φ pririšimas (angl. *binding*) – įsipareigojantysis negali pakeisti vertės po to, kai įsipareigojimas yra paskelbtas.

ECC galima naudoti įsipareigojimui sukurti. Tarkime, norime įsipareigoti vertei 8

$$\textit{įsipareigoti}\ (8) \rightarrow 8 * G.$$

Visiems kitiems mūsų įsipareigojimas $8 * G$ atrodo kaip atsitiktinis taškas, ir mes jį paskelbiame. Po kurio laiko atskleidžiame savo vertę 8.

Dabar bet kuris stebėtojas gali padauginti mūsų paskelbtą reikšmę 8 iš viešo taško G ir patikrinti, ar rezultatas yra lygus anksčiau paskelbtam įsipareigojimui.

$$\textit{patikrinti}\ (8, \text{įsipareigojimas}) == 8 * G\ ? \Rightarrow \text{True/False}$$

Tačiau yra didelė problema, jei vertė yra iš mažo intervalo, kas dažniausiai ir būna, dėl to bet kas gali lengvai sužinoti, kokiai vertei įsipareigojome, net jei jos neatskleidžiame. Tiesiog išbandydami skirtingas reikšmes (angl. *brute-force* metodą), randant tą vienintelę reikšmę, kuri, padauginta iš G, sutampa su pradiniu įsipareigojimu.

Tarkime, statome už tai, kiek įvarčių komanda įmuš iki metų pabaigos. Mūsų spėjimas yra 23, ir mes įsipareigojame tam paskelbdami įsipareigojimą $23 * G$. Problema ta, kad bet kas gali lengvai atskleisti mūsų spėjimą tiesiog bandydamas įsipareigoti reikšmėms 1, 2, 3, 4 ir t. t., kol ras rezultatą, kuris sutampa su mūsų įsipareigojimu. Šiuo atveju vertė būtų atskleista jau po 23 paprastų žingsnių.

Koks galėtų būti sprendimas?

Maskavimas

Ši problema išsprendžiama naudojant maskavimo reikšmę ***r*** (paprastai panašaus dydžio, kaip ir privatus raktas). Ši reikšmė naudojama tam, kad tikroji vertė būtų užmaskuota ir jos nebūtų galima atspėti ar atskleisti.

Galėtume pabandyti įsipareigoti naudodami $(8 + \boldsymbol{r}) * G$ ir tada atskleisti 8 bei r.

Tačiau tai sulaužo pririšimo (angl. *binding*) savybę, nes vietoje vertės 8 ir užmaskavimo veiksnio ***r*** galėtume atskleisti, pavyzdžiui, 7 ir $\boldsymbol{r} + 1$, 6 ir $\boldsymbol{r} + 2$ arba bet kokią kitą reikšmių kombinaciją.

Todėl reikalingas kitoks ***r*** įtraukimo metodas.

Pedersono įsipareigojimas

Pederseno įsipareigojimas elipsinių kreivių pagrindu leidžia paslėpti vertę taip, kad ją būtų galima vėliau atskleisti, tačiau iki tol niekas negali jos sužinoti ar pakeisti.
Pedersono įsipareigojimas apskaičiuojamas taip:

$$\mathbf{C} = \boldsymbol{r}G + \boldsymbol{v}H$$

kuriame:

- G ir H – viešai žinomi skirtingi elipsinės kreivės generatoriai;
- $\boldsymbol{r}$ – atsitiktinis maskavimo veiksnys, užtikrinantis slaptumą;
- $\boldsymbol{v}$ – įsipareigojama vertė;
- $\boldsymbol{C}$ – įsipareigojimo taškas elipsinėje kreivėje.

Savybės:

- tobulas slaptumas – be $\boldsymbol{r}$ neįmanoma nustatyti $\boldsymbol{v}$;
- saistymas – neįmanoma pakeisti $\boldsymbol{r}$ ar $\boldsymbol{v}$ be didelių skaičiavimo išteklių (diskretinio logaritmo problema);
- homomorfiškumas – dviejų įsipareigojimų (jų gali būti ir daugiau) suma sukuria naują įsipareigojimą jų sumai:
  $\mathbf{C}_1 + \mathbf{C}_2 = (\boldsymbol{r}_1G + \boldsymbol{v}_1H) + (\boldsymbol{r}_2G + \boldsymbol{v}_2H) = (\boldsymbol{r}_1 + \boldsymbol{r}_2)G + (\boldsymbol{v}_1 + \boldsymbol{v}_2)H$.

Naudojimas:

- konfidencialiose operacijose (pvz., *Monero* kriptovaliutoje, konfidencialių operacijų sistemoje *Mimblewimble*) – slepia sumas, bet leidžia patikrinti balansą;
- nulinio atskleidimo įrodymuose (pvz., neinteraktyviuose intervalų įrodymuose (angl. *Bulletproofs*)) – įrodo, kad įsipareigojimas atitinka tam tikras sąlygas be vertės atskleidimo (pvz. tam tikra suma yra reikiamose ribose);
- anoniminiame balsavime – balsai lieka paslėpti, bet gali būti patvirtinti.

Konfidencialių ir patikrintinų operacijų sistemos pavyzdys su Pedersono įsipareigojimais

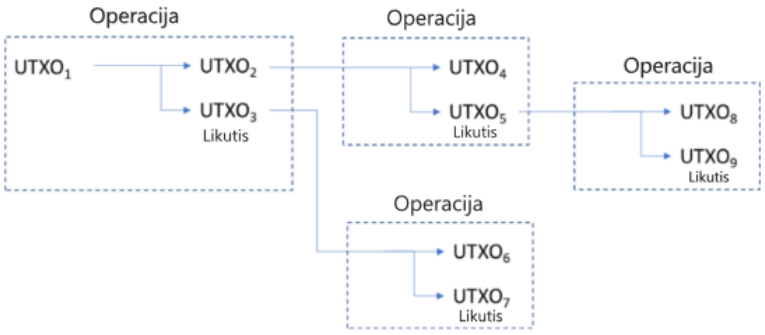
Susipažinti, kaip gali praktiškai atrodyti kriptografinių metodų taikymas, kuriant sistemas per supaprastintą konfidencialių ir patikrintinų operacijų sistemos pavyzdį, naudojant Pedersono įsipareigojimus. Bendrai ši schema yra konfidencialių ir patikrintinų operacijų sistema, paremta Pedersono įsipareigojimais ir elipsinių kreivių homomorfiškumu. Sistemoje kiekviena pinigų suma užšifruojama pagal formulę

$$\mathbf{C} = \boldsymbol{r}G + \boldsymbol{m}H.$$

Tokiu būdu tikrosios vertės lieka paslėptos, tačiau naudojant *Pedersono* įsipareigojimų adityvų homomorfiškumą galima matematiškai patikrinti, kad įėjimų ir išėjimų suma yra lygi (pvz., nėra sukuriama naujų lėšų ir išleidžiama tiek kiek galima, t.y ne daugiau ir ne mažiau negu sąskaitos balansas leidžia).

Operacijų įrašai saugomi blokų grandinėje pagal neišleistų operacijų principą (angl. trump. *UTxO*) (žr. 9 pav.), o tinklo mazgai ar kiti tinklo dalyviai patikrina operacijų teisingumą naudojant adityvų homomorfizmą, užtikrinant, kad net jei konkrečios vertės nėra atskleistos, balansai yra teisingi. Taip užtikrinamas saugus, patikimas ir konfidencialus pinigų pervedimas, išlaikant duomenų privatumą ir sąžiningą balansų valdymą.

Kadangi šios dalies tikslas yra pasibandyti Pedersono įsipareigojimus, neatliksime perduodamų verčių intervalų patikrinimo (angl. *range proofs*), **ypač nuo neigiamų reikšmių Tinklo mazge gavimo.**



9 pav. Neišleistų operacijų sistemos principo pavyzdys (naudotojų siunčiamų sumų ir likučių vizualizacija)

Apskaičiuojamas papildomas taškas H (mažesnis už G), apskaičiuojant atsitiktinę reikšmę $\boldsymbol{f} \leftarrow \text{randi}(\boldsymbol{Z_n})$ ir $H = \boldsymbol{f} * G$:

```
>>> f = secrets.randbelow(elk.field.n)
>>> print("f =", f)
f = 106987865628957589483119723362919795984757049124772000948300205763051126433849
>>> H = f * elk.g
>>> print("Hx =", H.x, "\nHy =", H.y)
Hx = 10361306947960057353533073333411852583019829742785524422091058095083184961054 9
Hy = 82300962205787621101914762994825633823729807459504303153582105487611986218442
```

Vos atidarius **Aldonai** sąskaitą, jos balansas $\boldsymbol{m} = 0$, **Tinklo mazgas** (angl. *node*) sugeneruoja maskavimo skaičių $\boldsymbol{r} \leftarrow \text{randi}(\boldsymbol{Z_n})$, apskaičiuoja įsipareigojimą $\mathbf{C_A} = \boldsymbol{r}G + \boldsymbol{m}H$ ir kurį **Tinklo mazgas** išsaugo neišleistų operacijų sąraše (angl. trump. *UTxO*):

```
>>> m = 0
>>> r = secrets.randbelow(elk.field.n)
>>> print("r =", r)
r = 878237705126478597202802482970947666627155987131451887259840926259017856865 50
>>> CA = r * elk.g + m * H
>>> print("CAx =", CA.x, "\nCAy =", CA.y)
CAx = 37096019144651982898252676325092880486234205847053931485378920751465044580019
CAy = 18261556284567679245756420538640179887391435062708622984352645564847233209554
```

Būtina pastebėti, kad sąskaitos gyvavimo eigoje, t.y. gaunant ir išleidžiant lėšas, $\boldsymbol{m}$, $\boldsymbol{r}$, $\mathbf{C}_?$ reikšmės būna kitokios, tačiau toliau pateikiami principai išlieka.

Tinklo mazgas iš **Broniaus** ir **Lino** gauna **Aldonai** persiunčiamų $\boldsymbol{m}_1$ ir $\boldsymbol{m}_2$ sumų įsipareigojimus $\mathbf{C_{1BA}}$ ir $\mathbf{C_{2LA}}$, kuriuos įsirašo į neišleistų operacijų sąrašą susiedamas su **Aldonos** pseudotapatybe (šiame žingsnyje praleisime **Broniaus** ir **Lino** balansų likučių palyginimą, tai vėliau atliksime su **Aldonos** persiunčiama pinigų suma **Emai**):

```
>>> #Tinklo mazgas gavo iš Broniaus ir Lino pervedamų pinigų sumų įsipareigojimus
>>> C1BAx = 709707548251521947807822231826109588313478616473137208790371849227796636 98376
>>> C1BAy = 311945945567082102335168444562985300769262652676785378201959389986264642 82032
>>> C1BA  = kreive.Point(elk, C1BAx, C1BAy)
>>> C2LAx = 309834104349894804495173933352014206289169997653259126594862457738042306 86659
>>> C2LAy = 514429032792395042996007816591172710256853516121954614014426101159941745 81081
>>> C2LA  = kreive.Point(elk, C2LAx, C2LAy)
```

Aldona iš **Broniaus** gauna užmaskavimo vertę r_1 ir siunčiamą piniginę vertę m_1 (šios dvi vertės **Aldonai** perduodamos šifruotu kanalu ir likusiam tinklui nežinomos išskyrus **Aldoną** ir **Bronių**) ir įsipareigojimą $\mathbf{C_{1BA}}$, kurį jai perduoda **Tinklo mazgas**, o iš **Lino** gauna užšifruotą užmaskavimo vertę r_2 ir piniginę vertę m_2 (šios dvi vertės **Aldonai** perduodamos šifruotu kanalu ir likusiam tinklui nežinomos išskyrus **Aldoną** ir **Liną**) ir įsipareigojimą $\mathbf{C_{2LA}}$, kurį perduoda **Tinklo mazgas**.

```
>>> #Aldona gavo iš Broniaus ir Lino pervedamų pinigų sumų duomenis
>>> m1 = 50
>>> m2 = 22
>>> r1 = 11544285064075519851616848280434778192249154104697750185887059811554942952922 4
>>> r2 = 74078960972376396783501760669364274515317583567259914967147323054756883907 7

>>> #Aldona gavo iš Tinklo mazgo Broniaus ir Lino pervedamų pinigų sumų įsipareigojimus
>>> C1BAx = 709707548251521947807822231826109588313478616473137208790371849227796636983 76
>>> C1BAy = 311945945567082102335168444562985300769262652676785378201959389986264642820 32
>>> C1BA  = kreive.Point(elk, C1BAx, C1BAy)
>>> C2LAx = 309834104349894804495173933352014206289169997653259126594862457738042306866 59
>>> C2LAy = 514429032792395042996007816591172710256853516121954614014426101159941745810 81
>>> C2LA  = kreive.Point(elk, C2LAx, C2LAy)
```

Aldona nutaria išsiųsti **Emai** pinigų sumą $m_3 = 34$ (pagal neišleistų operacijų principą (angl. trump. *UTxO*), operacija susideda iš pervedamos sumos ir likučio įsipareigojimų, likutis tai atgal pervedama **Aldonai** likusi piniginė suma atmetus išleidžiamą sumą iš bendro balanso):

1. Patikrinkite ar **Bronius** ir **Linas** atsiuntė teisingas r_1, r_2, m_1, m_2 vertes, perskaičiuodami iš **Audito** atsiųstus **Broniaus** ir **Lino** Pedersono įsipareigojimus $\mathbf{C_1} = r_1G + m_1H$ ir $\mathbf{C_2} = r_2G + m_2H$ ir patikrindami ar galioja sąlygos $\mathbf{C_1} = \mathbf{C_{1BA}}$ ir $\mathbf{C_2} = \mathbf{C_{2LA}}$, kad vėliau nebūtume jų suklaidinti ir neapskaičiuotume klaidingo **Aldonos** balanso (klaidingų verčių r_1, r_2, m_1, m_2 pateikimo atveju **Aldona** informuoja **Tinklo mazgą**, kad tam tikri įsipareigojimai būtų atsieti nuo jos sąskaitos, arba laikomi nuošalyje vykdant balanso patikrinimus):

```
>>> C1 = r1 * elk.g + m1 * H
>>> print("C1x =", C1.x, "\nC1y =", C1.y)
C1x = 709707548251521947807822231826109588313478616473137208790371849227796636983 76
C1y = 311945945567082102335168444562985300769262652676785378201959389986264642820 32
>>> C1 == C1BA
True

>>> C2 = r2 * elk.g + m2 * H
>>> print("C2x =", C2.x, "\nC2y =", C2.y)
C2x = 309834104349894804495173933352014206289169997653259126594862457738042306866 59
C2y = 514429032792395042996007816591172710256853516121954614014426101159941745810 81
>>> C2 == C2LA
True
```

2. Apskaičiuokite **Aldonos** balansą $\boldsymbol{m} = \boldsymbol{m} + m_1 + m_2$ ir reikšmę $\boldsymbol{r} = (\boldsymbol{r} + r_1 + r_2)$ **mod** n (modulis gali būti atliekamas ir po kiekvienos $\boldsymbol{r}$ reikšmių sumos):

```
>>> m = m + m1 + m2
>>> print("m =", m)
m = 72
>>> r = (r + r1 + r2) % elk.field.n
```

```
>>> print("r =", r)
r = 882153215527705734415863017587286178003633603716595293921039049109302720104 82
```

3. Apskaičiuokite **Aldonos** balanso likutį m_L (tinklui patvirtinus **Aldonos** pinigų pervedimo operaciją $m = m_{AL}$) nutarus persiųsti **Emai** pinigų sumą $m_3 = 34$:

```
>>> m3 = 34
>>> mAL = m - m3
>>> print("mAL =", mAL)
mAL = 38
```

4. Apskaičiuokite atsitiktinę **Emai** persiunčiamų pinigų maskavimo reikšmę $r_3 \leftarrow$ randi(Z_r) ir balanso likučio maskavimo reikšmę $r_{AL} = r - r_3$ (tinklui patvirtinus **Aldonos** pinigų pervedimo operaciją $r = r_{AL}$):

```
>>> r3 = secrets.randbelow(r)
>>> print("r3 =", r3)
r3 = 720997351733458957765064130551532277147616157209711373915097159593035609869 60
>>> rAL = (r - r3) % elk.field.n
>>> print("rAL =", rAL)
rAL = 161155863794246776650798887035753900856017446506883920005941889516267110235 22
```

5. Apskaičiuokite **Emai** ir balanso likučiui skirtus įsipareigojimus $C_{3AE} = r_3G + m_3H$ ir $C_{4AL} = r_{AL}G + m_{AL}H$:

```
>>> C3AE = r3 * elk.g + m3 * H
>>> print("C3AEx =", C3AE.x, "\nC3AEy =", C3AE.y)
C3AEx = 83927416677290355648673249959414697609437465746803547955109956680435123280863
C3AEy = 76683850573863126497620821039097366214996341885131303129635710977669810082067

>>> C4AL = rAL * elk.g + mAL * H
>>> print("C4ALx =", C4AL.x, "\nC4ALy =", C4AL.y)
C4ALx = 27486892065802354995895015165480628375764477138449072485869584597435053568485
C4ALy = 45529844945133525947513114943089536222597127130610437925376942250055175257166
```

6. **Aldona** išsiunčia **Emai** šifruotu kanalu r_3 ir m_3 vertes.
7. **Aldona** išsiunčia **Tinklo mazgui Emai** pervedamos sumos ir likučio įsipareigojimus C_{3AE} ir C_{4AL}.

Tinklo mazgas iš **Aldonos** gauna **Emai** persiųstos sumos m_3 įsipareigojimą C_3 ir balanso likučio m_4 įsipareigojimą C_4 ir turėdamas iš ankščiau siųstus **Broniaus** ir **Lino** įsipareigojimus C_{1B} ir C_{2L} **patvirtina arba atmeta atliekamą pinigų pervedimo operaciją**:

1. Apskaičiuokite **Aldonos** balanso ($m = 0$) įsipareigojimo, anksčiau **Broniaus** ir **Lino** siųstų (**Aldonai** skirtų pajamų $m_1 + m_2$) įsipareigojimų sumą $C_{12BLA} = C_A + C_{1BA} + C_{2LA}$:

```
>>> C12BLA = CA + C1BA + C2LA
>>> print("C12BLAx =", C12BLA.x, "\nC12BLAy =", C12BLA.y)
C12BLAx = 549552496600988365730937964226529916528340850905163236066125738301370125057 39
C12BLAy = 370946773650298744251101982183950855729470617299327007563216845176903400841 9
```

2. Turėdami pajamų ir išlaidų įsipareigojimus, patikrinkite ar **Aldonos** balansas yra pakankamas atlikti pinigų pervedimo operaciją **Emai**, patikrindami ar tenkinama balanso sąlyga, kartu

įsitikinant, kad nėra sukuriama naujų pinigų. Tai galima atlikti dviem dažniausiai taikomais metodais $\mathbf{C_{12BLA}} = \mathbf{C_{3AE}} + \mathbf{C_{4AL}}$ arba $0G = \mathbf{C_{12BLA}} - \mathbf{C_{3AE}} - \mathbf{C_{4AL}}$:

```
>>> CAEL = C3AE + C4AL
>>> print("CAELx =", CAEL.x, "\nCAELy =", CAEL.y)
CAELx = 549552496600988365730937964226529916528340850905163236066125738301370125057 39
CAELy = 370946773650298744251101982183950855729470617299327007563216845176903400841 9
>>> C12BLA == CAEL
True

#Arba
>>> 0*elk.g == C12BLA - C3AE - C4AL
True
```

3. Esant pakankamam **Aldonos** balansui **Tinklo mazgas** patvirtina pinigų pervedimo operaciją **Emai**, o esant nepakankamam atmeta. Kai balansas pakankamas, **Tinklo mazgas** neišleistų operacijų sąraše pašalina **Broniaus** ir **Lino** įsipareigojimus $\mathbf{C_{1BA}}$ ir $\mathbf{C_{2LA}}$ skirtus **Aldonai** į **Aldonos** balanso Likučio įsipareigojimą $\mathbf{C_{4AL}}$, o **Emai** priskiria ir išsaugo **Aldonos** siųstą įsipareigojimą $\mathbf{C_{3AE}}$.

Kai **Tinklo mazgas** patvirtina pinigų pervedimo operaciją ir iš ankščiau žinant, kad Bronius ir Linas patvirtinimo **Aldona** atsinaujina savo balansą $\boldsymbol{m} = \boldsymbol{m}_{\mathbf{AL}}$ ir išsaugo $\boldsymbol{r} = \boldsymbol{r}_{\mathbf{AL}}$:

```
>>> m = mAL
>>> print("m =", m)
m = 38
>>> r = rAL
>>> print("r =", r)
r = 161155863794246776650798887035753900856017446506883920005941889516267110235 22
```

Toliau **Emos** operacijoms atliekami analogiški veiksmai, priklausomai nuo gautų pajamų.

Taip pat, kai **Tinklo mazgas** patvirtina pinigų pervedimo operaciją, įėjimo ir išėjimo įsipareigojimai greta gavėjų pseudotapatybės ir kitų reikiamų duomenų yra įrašomi į blokų grandinę, kas leidžia visiems kitiems likusiems tinklo dalyviams savarankiškai patikrinti atliktos operacijos sąžiningumą, nežinant tikrųjų siuntėjų ir gavėjų tapatybių, bei pervedamų sumų.

Užduotys konfidencialių ir patikrintinų operacijų sistemos pavyzdžiui naudojant Pedersono įsipareigojimus.

Toliau naudosime subjektui **Aldona** papildomą tašką *H*.

1. Būdami už **Aldoną** ir turėdami sąskaitoje m pinigų sumą ir maskavimo reikšmę r, bei įsipareigojimą C_A ir turėdami **Broniaus**, **Lino, Tomo** (priklausomai nuo varianto) atsiųstas maskavimo $r_1, r_2, ..., r_n$ ir pinigines $m_1, m_2, ..., m_n$ vertes ir Pedersono įsipareigojimus C_{1BA}, C_{2LA}, C_{3T}. Jeigu **Aldona** gauna pajamų, patikrinkite ar tiek **Broniaus**, tiek **Lino**, tiek **Tomo** atsiųstos vertės yra teisingos ir su teisingomis vertėmis atnaujinę sąskaitos balanso piniginę vertę m ir maskavimo vertę r, atlikite pinigų pervedimą **Emai** suformuodami pervedamos sumos C_{AE} ir likučio C_{AL} įsipareigojimus, panaudojant pateiktas reikšmes:

1.

```
#Aldonos sąskaitos balanso duomenys
m = 5
r = 24998193511304764017003884053525726949660637215612655750834316096693995294086

CAx = 60691148363097389013627378896610532503205813113926334721044253596861985745127
CAy = 78218632407424074955805434441474353769535508308152238047600279506373195152 23
CA  = kreive.Point(elk, CAx, CAy)

#Aldona gavo iš Broniaus pervedamos pinigų sumos duomenis
m1 = 17
r1 = 10011107475068482542389517313725710879555342950285504361089769524743715783644 3

#Aldona gavo iš Tinklo mazgo Broniaus pervedamos pinigų sumos įsipareigojimą
C1BAx = 81328245871462294105501060876540518231871736502358267488988923773651180004506
C1BAy = 97210086494735083254508519027894948233122371531554048568015350643687797005895
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

#Aldonos Emai pervedamos pinigų sumos duomenys
m2 = 12
r2 = 25595919292475774045376330972499612877335415510994960099307846716516090173715
```

2.

```
#Aldonos sąskaitos balanso duomenys
m =  15
r = 587519829104051499272705203404829112516135722493967668515142221885332212084 52

CAx = 450803368712693302151971648847040191481868086655252352881463408445318517222 51
CAy = 154692817960379551087588620536843739359333568707043658647374026015658877616 98
CA  = kreive.Point(elk, CAx, CAy)

#Aldona gavo iš Broniaus, Lino ir Tomo pervedamų pinigų sumų duomenis
m1 = 17
m2 = 25
m3 = 28
r1 = 25802857290268420746519622728098695428745781065604716820712355236644913300669
r2 = 63523266590345457754842101915436862609028179490725628121794339547677757096268
r3 = 11313328054153752611312398857813568079524935393524385259991824337158890487847 7
```

```
#Aldona gavo iš Tinklo mazgo Broniaus, Lino ir Tomo pervedamų pinigų sumų įsipareigojimus
C1BAx = 834389365938758079706134011961028346315820469780197387268013042647120973846 45
C1BAy = 892096116727668824056257387449346087336069489487861747891292403832029755836 37
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

C2LAx = 706899321406996919036529074373872413460904203420011951300976774753936493975 27
C2LAy = 859084110440733223061659380698123225504461470013106490292411765605477981153 34
C2LA  = kreive.Point(elk, C2LAx, C2LAy)

C3TAx = 281571268125164031186912018065758752665069920576191982108564225783494897326 77
C3TAy = 450525130854624765448241102927529792465166732627271417219574788671888938519 69
C3TA  = kreive.Point(elk, C3TAx, C3TAy)

#Aldonos Emai pervedamos pinigų sumos duomenys
m4 = 24
r4 = 838750313748722946290398524079990069742102001321367273552152013218172432729 78
```

3.

```
#Aldonos sąskaitos balanso duomenys
m =  11
r = 577399494093257430543690550360059043812518156732773985669679038012250996913 74

CAx = 881408174026983595042070591920044967825189469078667970726859099427423403701 94
CAy = 478557100513872761573576466477815901695855009130562876437670671694511124666 0
CA  = kreive.Point(elk, CAx, CAy)

#Aldona gavo iš Broniaus ir Lino pervedamų pinigų sumų duomenis
m1 = 8
m2 = 22
r1 = 402764725204252781229871008878322145684046788313673446536893092580428136178 24
r2 = 893585835567916061165947055724055268916010420771575805420542900886103000500 32

#Aldona gavo iš Tinklo mazgo Broniaus ir Lino pervedamų pinigų sumų įsipareigojimus
C1BAx = 327767724996752557132914405166341097076880691893758636419302728548819270951 24
C1BAy = 554957235750365653574832986122724123410685339020531194320434519812194810181 42
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

C2LAx = 654043927527014015215513986766598587049329670297410237582602052194296788881 97
C2LAy = 424895120611129352235371594762332934066810323050169712208515911496199097174 26
C2LA  = kreive.Point(elk, C2LAx, C2LAy)

#Aldonos Emai pervedamos pinigų sumos duomenys
m3 = 15
r3 = 834959454195581626569999794421223276204848016193886943002756908555501983898 81
```

4.

```
#Aldonos sąskaitos balanso duomenys
m =  29
r = 80172079000133744013133465712252404202265438170039763584557276099262284714156

CAx = 456562810035012592081648105315965560577792731410281983146051625105036687886 52
CAy = 216964546030859086687785444699843554472698444545150623239680175618294009372 77
CA  = kreive.Point(elk, CAx, CAy)

# Aldona negavo pajamų

#Aldonos Emai pervedamos pinigų sumos duomenys
m1 = 26
r1 = 10375674455149627110895871047447897923670920182173075365426116669205594816516 7
```

5.

```
#Aldonos sąskaitos balanso duomenys
m = 35
r = 6365009058398926071653822710035985838257310471951804621544944904523412980038 1

CAx = 5689423070347114547314481785433059873154115430568148922671141285519369298507 2
CAy = 6639439869282921635646034888873363228653816507688407617121605027023824415634 5
CA  = kreive.Point(elk, CAx, CAy)

#Aldona gavo iš Broniaus, Lino ir Tomo pervedamų pinigų sumų duomenis
m1 = 56
m2 = 66
m3 = 77
r1 = 4615715537071691491637543466464186738619249801641051242721798703099130898352 1
r2 = 1077055498797939032842170075731811577141782569452780944276187288256992418639 8
r3 = 8462490619514714119582700857815633810653454502400443596304817502239219008095 9

#Aldona gavo iš Tinklo mazgo Broniaus, Lino ir Tomo pervedamų pinigų sumų įsipareigojimus
C1BAx = 4505753565307851676504150333389084224098115038573616167676475972731111775544 6
C1BAy = 8472011508019657312441686558964962670337465311074113360043950653849459226793 2
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

C2LAx = 4283583610080766827658645097306952756566233873654873548603976538218296612048
C2LAy = 2200471434632999271909015122272356357672192653124776435589756967365834625048 1
C2LA  = kreive.Point(elk, C2LAx, C2LAy)

C3TAx = 4880789075158462049586180260062641584718255433637145665228617152718606026465 4
C3TAy = 8857780243557449846412773995218131056862850742771685650555040317228618421742
C3TA  = kreive.Point(elk, C3TAx, C3TAy)

#Aldonos Emai pervedamos pinigų sumos duomenys
m4 = 44
r4 = 7739674269334639910466585677273817601056434512371146686433596423974749595942 2
```

6.

```
#Aldonos sąskaitos balanso duomenys
m = 60
r = 5103325300640923065679520776940904988228997572605229192395395668689610407391 2

CAx = 90561323442387904329938729012196336286607122197109568962895829424379145385959
CAy = 43196604475859512705690989394228979416096965316426011371257376928809569537333
CA  = kreive.Point(elk, CAx, CAy)

#Aldona gavo iš Broniaus pervedamos pinigų sumos duomenis
m1 = 75
r1 = 92093132937775698239124508472708765116615764495373928672031851517230124141330

#Aldona gavo iš Tinklo mazgo Broniaus pervedamos pinigų sumos įsipareigojimą
C1BAx = 48777871445882044448152704219608818399931353727028120682180347581492830407730
C1BAy = 11345522561318354182645639732795151756145244660691370349958269740064020482531 8
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

#Aldonos Emai pervedamos pinigų sumos duomenys
m2 = 55
r2 = 28350353516572721848152683708407733071082023345171165041065987604064064526149
```

Pervedamos sumos $\mathbf{C_{AE}}$ ir likučio $\mathbf{C_{AL}}$ Pedersono įsipareigojimai:

1.

```
CAEx = 232277149163042205559230863757168679339846258853295475692182297178364523391 01
CAEy = 223441220529004629437377715381300754575256166031580185722808908853030062414 55
CALx = 539856590255324434220285323126249063743381133017760317936943202851822909691 8
CALy = 570572354732085993852501588659589223867630029161990684968573052248301428782 02
```

2. Gautas vienas klaidingas Pedersono įsipareigojimas pervedamai pinigų sumai.

```
CAEx = 287291064455421927887006119434864153995926271887165535060047844539214387076 94
CAEy = 225251868106748288986710209899701166001197410467995764749904849644348317083 98
CALx = 861539969335285797256458655825253336793311392476440444044042037978966182917 473
CALy = 110638747723266437426014374164214230971745294936144428536176022765159897661 358
```

3.

```
CAEx = 991806954645607879325395504768401096510917622096538762388469210856920814001 54
CAEy = 673530930135076698082810788857781703091047006076317652283744859595261793909 62
CALx = 300387326626682753082905981288971950630374205844482620665138123402378944349 79
CALy = 175139113788942971961146352566002212878346141734156683817518840515130825180 48
```

4. Gauti du klaidingi Pedersono įsipareigojimai pervedamai pinigų sumai.

```
CAEx = 685166088431185817733486947410345978353522730170009715419617244354295458519 15
CAEy = 487564955623132443198183915381359243960984033950731886684410320925537328494 12
CALx = 110188435311105507044827480629883243283349787033441496400365250263671875000 445
CALy = 768778867387485218940604167474091493530919536160297389606512851974942696193 47
```

5.

```
CAEx = 357459426223679798858312121100927522125897859297414009574364512933191151502 02
CAEy = 186608232480566211735437851086400720820689301187185790226759987414052954286 91
CALx = 114561620610559581632521429760977632600652180050017458098181435660960080397 408
CALy = 420537583278939341526110566354988157161012439710681757404616148663982008865 75
```

6.

```
CAEx = 1005699495832588788905692466721725444015918870150655807114206573016649093986 82
CAEy = 6405361461078750142078107650971906580767275518745037341795000495603068444909
CALx = 65300226873126023493821083055504009829591364919198576585591965325918036797049
CALy = 17604933944695627075143908052077750587741142260571349351197143884498823324360
```

2. Būdami už **Tinklo mazgą** ir turėdami **Aldonos** balanso įsipareigojimą **C** ir **Aldonai** priskirtus pajamų įsipareigojimus $\mathbf{C_1}$, $\mathbf{C_2}$, ..., $\mathbf{C_n}$ (**Broniaus** ir kt. asmenų atsiųsti įsipareigojimai, jų gali ir nebūti) ir **Aldonos Emai** pervedamos sumos ir **Aldonos** likučio įsipareigojimus $\mathbf{C_{AE}}$ ir $\mathbf{C_{AL}}$, patikrinkite ar galioja balanso sąlyga (ar **Aldona** nepateikė klaidingų duomenų **Tinklo mazgui**) ir patvirtinkite, arba atmeskite pinigų pervedimo operaciją:

1.

```
#Aldonos sąskaitos balanso įsipareigojimas
Cx = 718111654744391312755678916589956549291098283589155534694286562456445157954
Cy = 623589262646491263792659751176261607579220331193471534817375870978333259958 18
C  = kreive.Point(elk, Cx, Cy)

#Aldona negavo pajamų

#Aldonos išlaidų įsipareigojimas
CAEx = 13568212920185986332405797494522334090121668823986084292372097841695027292605
CAEy = 45272880658061745756515776273438421266422055518427463813892560563102453169819
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 10719440758157849229726902798831675899730403080089574867416286479790670005 1141
CALy = 74086604767383823134107197144374777778263227794813712553342336256478598909433
CAL  = kreive.Point(elk, CALx, CALy)
```

2.

```
#Aldonos sąskaitos balanso įsipareigojimas
Cx = 754511615008651063014847449738792831495007829245674609412409122681400716857 86
Cy = 67460196057913147439800876576563848633860518575367320879618632166496204861507
C  = kreive.Point(elk, Cx, Cy)

#Aldonos gautų pajamų iš Broniaus ir Lino įsipareigojimai
C1BAx = 655798546175150363431026117535868540326766442648072638729976759753504006352 15
C1BAy = 426572133800483322874598602213533241613845423517991211877192192886613459472 68
C1BA  = kreive.Point(elk, C1BAx, C1BAy)
C2LAx = 106754635721945994767034337586005131522057179341183035916035547319311422071730
C2LAy = 20650208863817229805457440108395885534511157129589069707986585398739615443 98
C2LA  = kreive.Point(elk, C2LAx, C2LAy)

#Aldonos išlaidų įsipareigojimas
CAEx = 61387822558328005091946065009871699516999009374679119446565389041218905682129
CAEy = 30205202694991777202286302766660968468938665333402735434349278904932603358469
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 63293252195993862698149245946529101320782737352094346912033897342890810817116
CALy = 44279171280460406003310574124281299724986312217402801105060134076791624880238
CAL  = kreive.Point(elk, CALx, CALy)
```

3.

```
#Aldonos sąskaitos balanso įsipareigojimas
Cx = 786844280384302246662804753474142525210608808044563475864992520832892024613330
Cy = 897853357639741892069698652886532473161120295901266238367392392403560134621420
C  = kreive.Point(elk, Cx, Cy)

#Aldonos gautų pajamų iš Broniaus įsipareigojimas
C1BAx = 10609980707099704912919597869487664584097550757938954490304343698912832011742800
C1BAy = 54050464503361113375213396632821628305202431631198674555401191205430865251630
C1BA  = kreive.Point(elk, C1BAx, C1BAy)

#Aldonos išlaidų įsipareigojimas
CAEx = 35929713784280900920835125053650326470507673573739115810173754929989300277256
CAEy = 49452157986864871887637741930982026500844101293918875831497613873496342739440
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 994541658615604908317961767200305660905350172461059235508530352300316789513250
CALy = 70725369033407744419236414422570654247296955410865482759828404951060329835517
CAL  = kreive.Point(elk, CALx, CALy)
```

4.

```
#Aldonos sąskaitos balanso įsipareigojimas
CAx = 10891827402429284238550562156039375732899686837498877499775691600964774746242100
CAy = 10425109325787900871874665641960400336797612747173865674291932738927519433524800
CA  = kreive.Point(elk, CAx, CAy)

#Aldonos gautų pajamų iš Broniaus ir Lino įsipareigojimai
C1BAx = 35192638168227845885254588762663033667858765711808128551356379905155540492029
C1BAy = 47324504357801702820971604691814931775216177846735575396046019372113820548895
C1BA  = kreive.Point(elk, C1BAx, C1BAy)
C2LAx = 94617135887168317264597820346310853138219528017679486518011272003093574680803
C2LAy = 43454550864414584923644952165827096026244363925055748148264075627394330033040
C2LA  = kreive.Point(elk, C2LAx, C2LAy)

#Aldonos išlaidų įsipareigojimas
CAEx = 96910485441436966465559113844063431139916278202234388581015318983729268635297
CAEy = 11501674668154004706707797527343279804799979105976514688839726016479249484807500
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 113999156702459149212847853226118479466416766488951580818630713204018390559977
CALy = 10718152201523354156309955406802600934226886658433095658611744458719184552909800
CAL  = kreive.Point(elk, CALx, CALy)
```

5.

```
#Aldonos sąskaitos balanso įsipareigojimas
CAx = 3019550067236931833025999429478416786758627247049532105979847970081463850568O
CAy = 1009277242172845189839290467484315986425325302990749214662463108877018262912222
CA  = kreive.Point(elk, CAx, CAy)

#Aldonos gautų pajamų iš Broniaus, Lino ir Tomo įsipareigojimai
C1BAx = 401641327398220266848689903718232049023554945512399898907280150901030927705990
C1BAy = 804357573694201005769404515482233681971924732106067095517030253241993170006220
C1BA  = kreive.Point(elk, C1BAx, C1BAy)
C2LAx = 254653307839609256869820290364562418079531144325533611103709375677236373167430
C2LAy = 372493051091244819809583681469161288745495445893022196667183651315149994393
C2LA  = kreive.Point(elk, C2LAx, C2LAy)
C3TAx = 808128569495642690541473255040856905048241367588631433172927998812688200742440
C3TAy = 170127283064053499884590444275665824552872420161238892907776662385106782416470
C3TA  = kreive.Point(elk, C3TAx, C3TAy)

#Aldonos išlaidų įsipareigojimas
CAEx = 135682129201859863324057974945223340901216688239860842923720978416950272926050
CAEy = 452728806580617457565157762734384212664220555184274638138925605631024531698190
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 3547347994057907815521508865320978239015780139387386605586660849819931192552700
CALy = 432923184339314694076156479066992191893611898832221399341829243692340259895200
CAL  = kreive.Point(elk, CALx, CALy)
```

6.

```
#Aldonos sąskaitos balanso įsipareigojimas
CAx = 1089182740242928423855056215603937573289968683749887749977569160096477474624210
CAy = 1042510932578790087187466564196040033679761274717386567429193273892751943352480
CA  = kreive.Point(elk, CAx, CAy)

#Aldonos gautų pajamų iš Broniaus, Lino, Tomo ir Justo įsipareigojimai
C1BAx = 2971061138892251729479980163895636734457709211258640195752259932912703524794200
C1BAy = 6888677586197114249716177487243733867530716562470313984853551145505888254200300
C1BA  = kreive.Point(elk, C1BAx, C1BAy)
C2LAx = 2460532656871930530291375900268658883426591829916356226399752940533287629658500
C2LAy = 7026129623650970673455467201796039210587064710247533502825261820028821385789400
C2LA  = kreive.Point(elk, C2LAx, C2LAy)
C3TAx = 1555157872561959291133519757304930252753705659346699860127241092730807043226700
C3TAy = 1617874363805212854505678713776391564236551974267653154096733677009206453346800
C3TA  = kreive.Point(elk, C3TAx, C3TAy)
C4JAx = 9759765765668505604337859118111011528873441817351755827307430752441383596226700
C4JAy = 3996808658041652540295159244339560963023431811479236946047766248262483943775600
C4JA  = kreive.Point(elk, C4JAx, C4JAy)

#Aldonos išlaidų įsipareigojimas
CAEx = 439175048079693579126054409096669708090368220758101487615171025836012376735770
```

```
CAEy = 610817231544941219489954646270656817923792663827625448901253059163814037842 74
CAE  = kreive.Point(elk, CAEx, CAEy)

#Aldonos likučio įsipareigojimas
CALx = 37065873051800576512233633378322764464483969019490525748703917414628772188080
CALy = 72776676752978270167068421237525023857929002440273744182057772606327518254552
CAL  = kreive.Point(elk, CALx, CALy)
```

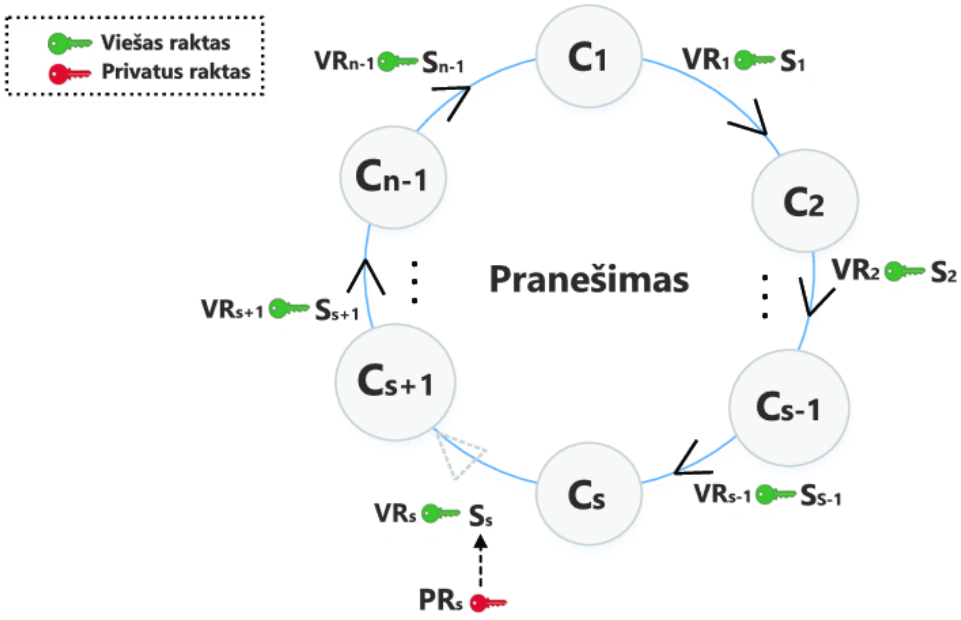
Dviejų operacijų **Tinklo mazgas** nepatvirtino.

Monero žiediniai parašai (LSAG)

Siekiant tiksliau aptarti *Monero* žiedinius parašus pasiremsime straipsniu *Privatumas Monero Blokų grandinėje*[2].

Žiediniai parašai – tai parašai, sukurti naudojant vieną privatų raktą ir nesusijusių viešųjų raktų rinkinį (angl. *set*). Visas viešųjų raktų rinkinys, įskaitant ir tą, kuris atitinka turimą privatųjį raktą, paprastai vadinamas *žiedu*. Žiedas suformuojamas (žr. 10 pav.) ne mažiau negu iš dviejų viešųjų raktų ir taip, kad kas nors, tikrinantis parašą, negalėtų pasakyti, kieno privatus raktas iš žiedo buvo panaudotas parašui sukurti.

Žiediniai parašai iš pradžių buvo vadinami grupiniais parašais, nes jie buvo laikomi būdu įrodyti, kad pasirašiusysis priklauso grupei, nebūtinai identifikuojant konkretų asmenį. Kreipiant žvilgsnį į *Monero* sandorius, jie padeda užtikrinti, kad valiutos srautų nebūtų galima atsekti.



10 pav. Žiedinio parašo vizualizacija

Žiedinio parašo schemos gali pasižymėti tam tikromis savybėmis, kurios naudingos konfidencialiems sandoriams:

Anonimiškumas. Stebėtojas neturėtų galėti nustatyti tikrojo pasirašančiojo tapatybės-pranešimo autoriaus. Gali nustatyti tik tai, kad naudojamas privatus raktas atitinka vieną iš žiedo viešųjų raktų.

Susiejamumas. Jei privatus raktas naudojamas dviem skirtingiems pranešimams pasirašyti, pranešimai bus susieti ir bus atskleistas jų dubliavimasis. Monero atveju ši savybė padeda išvengti dvigubo išleidimo atakų.

Išvengiamumas. Žiedo narys, kurio viešasis raktas buvo du kartus panaudotas dviejuose žiediniuose parašuose, bet nėra tikrasis abiejų raktų pasirašytojas, nebus susietas.

[2] Žr. straipsnio 3 skyrių, 11 psl.

Vaizdo įrašas apie Monero naudojamus žiedinius parašus ir protokolą *Ring CT*.

Funkcija, supaprastinanti *sha256* funkcijos naudojimą iš *hashlib* bibliotekos:

```
>>> def sha256(pranesimas):return int(santrauka.sha256(pranesimas.encode()).hexdigest(), 16)
# Po funkcijos įvedimo vieną kartą paspauskite enter
```

Žemiau pateikiami Monero **Susietų spontaniškų anoniminių grupės parašų**[3] (angl. trump. *LSAG*) pavyzdžiai. Supaprastinimui naudosime pranešimų santraukas $h_t = \mathbf{H}(t)$.

1 pavyzdys.

Aldona pasirenka *k* kitų naudotojų viešųjų raktų, pvz., 1:

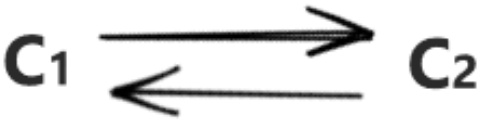
```
>>> px1 = 798726556201716497336522035909086153798511964645877913723216328435613983171111
>>> py1 = 465000062528052932431269605013099904696085409909664307161547434375709208999940
>>> P1 = kreive.Point(elk, px1, py1)
```

Aldona savo viešąjį raktą įterpia savo nuožiūra arba atsitiktinai į viešųjų raktų rinkinį, pvz., į 2-ąją poziciją. Toliau naudosime subjektui **Aldona** raktų porą (*z*, *P*):

```
>>> z = 434196856235539588612242849836326614904811237557557881183805632605342055793331
>>> px2 = 627685151695298722975103679584133413954202161994963086337694556855629061711149
>>> py2 = 139356695635205035617203294356522884030158520728201784908330873494255855314721
>>> P2 = kreive.Point(elk, px2, py2)
```

Aldonos pranešimas ***t*** ir apskaičiuota santrauka $h_t = \mathbf{H}(t)$:

```
>>> t = "Labas Broniau!"
>>> ht = sha256(t)
>>> print("ht =", ht)
ht = 33428293288344138697462033928697126385497612430434714066603703596314523426753
```



11 pav. Supaprastinta **Aldonos** žiedinio parašo žiedo vizualizacija

[3] Monero žiedinis parašo LSAG matematinis aprašymas pateikiamas 12 psl.

Aldona formuoja parašą **σ** pranešimui ***t*** su savo privačiuoju raktu (PR_A) z ir jos sudarytu viešųjų raktų rinkiniu P_1 ir P_2:

1. Sujunkite viešuosius raktus $R=P_1 \| P_2$:

```
>>> u1 = str(P1.x) + str(P1.y)
>>> u2 = str(P2.x) + str(P2.y)
>>> R = u1 + u2
>>> print("R =", R)
R = 798726556201716497336522035909086153 79851196...207282017849083308734942558553147 2
```

2. Apskaičiuokite tašką $HR = \mathbf{H(R)} * G$ ant elipsinės kreivės:

```
>>> hr = sha256(R)
>>> print("hr =", hr)
hr = 64909361798535688151340705868781725567630600761874113470948993316373955814742

>>> HR = hr * elk.g
>>> print("HRx =", HR.x, "\nHRy =", HR.y)
HRx = 10797626100652160887185703739165088640676537082425747838793487318509676481282 8
HRy = 98394613010839762582815157242702207771400610789167201046054869515353726028599
```

3. Apskaičiuokite bendrą anonimizuotą viešą raktą (angl. *Key Image*) $P = z * HR$:

```
>>> P = z * HR
>>> print("Px =", P.x, "\nPy =", P.y)
Px = 331967459880782000031256789781467480334963873823361378012286834448655970368 43
Py = 114617704495852430921225477332867507823764983836248950983276549267191546814829
```

4. Apskaičiuokite **rpht** = $\mathbf{R} \| P \| h_t$:

```
>>> rpht = R + str(P.x) + str(P.y) + str(ht)
>>> print("rpht =", rpht)
rpht = 798726556201716497336522035909086153...976124304347140666037035963145234267 53
```

5. Sugeneruokite atsitiktinį skaičių α ir patikrinkite ar galioja sąlyga $1 \leq \alpha \leq n\text{-}1$:

```
>>> al = secrets.randbelow(elk.field.n)
>>> print("al =", al)
al = 767311425974613145825476322975327901671698982764989150131217427629420206577 93
>>> 1 <= al <= elk.field.n-1
True
```

6. Apskaičiuokite $c_1 = \mathbf{H(rpht} \| \alpha * G \| \alpha * HR\mathbf{)}$, pradėdami nuo sekančio žiedo nario:

```
>>> alG = al * elk.g
>>> alHR = al * HR
>>> c1 = sha256(rpht + str(alG.x) + str(alG.y) + str(alHR.x) + str(alHR.y))
>>> print("c1 =", c1)
c1 = 599273019825712715726254163644862813786996178809327706385855808506612834792 43
```

7. Sugeneruokite atsitiktinį skaičių $1 \leq r_1 \leq n-1$ likusiam žiedo nariui (**Aldonos** pozicija 2 praleiskime, nes vėliau bus apskaičiuota reikšmė r_2):

```
>>> r1 = secrets.randbelow(elk.field.n)
>>> print("r1 =", r1)
r1 = 25770246921138737637294902308593062172333517625926759797237808495776606871198
>>> 1 <= r1 <= elk.field.n-1
True
```

8. Apskaičiuokite $c_2 = \mathbf{H}(\mathbf{rpht} \| r_1 * G + c_1 * P_1 \| r_1 * HR + c_1 * P)$:

```
>>> q11 = r1 * elk.g + c1 * P1
>>> q12 = r1 * HR + c1 * P
>>> c2 = sha256(rpht + str(q11.x) + str(q11.y) + str(q12.x) + str(q12.y))
>>> print("c2 =", c2)
c2 = 100629887388077574123456695917904793886473757662310976012272471638669595530 91
```

9. Pririškite **Aldonos** privatų raktą z, apskaičiuodami $r_2 = \alpha - z \bullet c_2 \ \mathbf{mod}\ n$:

```
>>> r2 = (al - z * c2) % elk.field.n
>>> print("r2 =", r2)
r2 = 90306192913599892990701596560230501034388808992097743575094385917092314029506
```

10. Žiedinis parašas $\sigma = (c_1, r_1, r_2, P)$ santraukai h_m yra
Pasirašyti$(x, h) = \sigma = (c_1, r_1, r_2, P) =$

```
>>> print("c1 =", c1, "\nr1 =", r1, "\nr2 =", r2, "\nPx =", P.x, "\nPy =", P.y)
c1 = 59927301982571271572625416364486281378699617880932770638585580850661283479243
r1 = 25770246921138737637294902308593062172333517625926759797237808495776606871198
r2 = 90306192913599892990701596560230501034388808992097743575094385917092314029506
Px = 331967459880782000031256789781467480334963873823361378012286834448655970368 43
Py = 114617704495852430921225477332867507823764983836248950983276549267191546814829
```

11. **Aldona** anoniminiu kanalu siunčia **Broniui** viešųjų raktų rinkinį P_1, P_2, pranešimą t, parašą σ.

Bronius gauna viešųjų raktų rinkinį P_1, P_2, žiedinį parašą σ pranešimui t. Žiedinis parašas $\sigma = (c_1, r_1, r_2, P)$ pranešimui t yra patikrinamas naudojant anonimizuotą viešą raktą (VR) P ir su juo susietus viešuosius raktus P_1, P_2:

1. Apskaičiuokite pranešimo santrauką $h_t = \mathbf{H}(t)$:

```
>>> ht = sha256(t)
>>> print("ht =", ht)
ht = 33428293288344138697462033928697126385497612430434714066603703596314523426753
```

2. Sujunkite viešuosius raktus $R = P_1 \| P_2$:

```
>>> u1 = str(P1.x) + str(P1.y)
>>> u2 = str(P2.x) + str(P2.y)

>>> R = u1 + u2
>>> print("R =", R)
R = 798726556201716497336522035909086153798511 96...207282017849083308734942558553147 2
```

3. Apskaičiuokite tašką $HR = \mathbf{H(R)} * \boldsymbol{G}$ ant elipsinės kreivės:

```
>>> hr = sha256(R)
>>> print("hr =", hr)
hr = 64909361798535688151340705868781725567630600761874113470948993316373955814742

>>> HR = hr * elk.g
>>> print("HRx =", HR.x, "\nHRy =", HR.y)
HRx = 107976261006521608871857037391650886406765370824257478387934873185096764812828
HRy = 983946130108397625828151572427022077714006107891672010460548695153537260285 99
```

4. Apskaičiuokite **rpht** = **R**‖P‖h_t:

```
>>> rpht = R + str(P.x) + str(ht)
>>> print("rpht =", rpht)
rpht = 798726556201716497336522035909086153...976124304347140666037035963145234267 53
```

5. Apskaičiuokite $c'_2 = \mathbf{H(rpht}\|z'_1\|z''_1\mathbf{)}$, kai $z'_1 = \boldsymbol{r_1} * G + c_1 * P_1$ ir $z''_1 = \boldsymbol{r_1} * HR + c_1 * P$:

```
>>> z11 = r1 * elk.g + c1 * P1
>>> z12 = r1 * HR + c1 * P
>>> c2s = sha256(rpht + str(z11.x) + str(z11.y) + str(z12.x) + str(z12.y))
>>> print("c2s =", c2s)
c2s = 100629887388077574123456695917904793886473757662310976012272471638669595530 91
```

6. Apskaičiuokite $c'_1 = \mathbf{H(rpht}\|z'_2\|z''_2\mathbf{)}$, kai $z'_2 = \boldsymbol{r_2} * G + c_2 * P_2$ ir $z''_2 = \boldsymbol{r_2} * HR + c_2 * P$:

```
>>> z21 = r2 * elk.g + c2s * P2
>>> z22 = r2 * HR + c2s * P
>>> c1s = sha256(rpht + str(z21.x) + str(z21.y) + str(z22.x) + str(z22.y))
>>> print("c1s =", c1s)
c1s = 599273019825712715726254163644862813786996178809327706385855808506612834792 43
```

7. Patikrinkite ar žiedinis parašas yra tikras, t.y. ar galioja lygybė $c'_1 = c_1$:

```
>>> c1s == c1
True
```

Patikrinti(P_1, ..., P_n, **σ**, h') = $P \in \{True, False\} \equiv \{1, 0\}$.

Tikrintojas **Bronius** priima parašą, jeigu tenkinamos visos aukščiau pateiktos sąlygos, kitais atvejais parašą atmeta.

2 pavyzdys.

Bronius pasirenka *k* kitų naudotojų viešųjų raktų, pvz., 4:

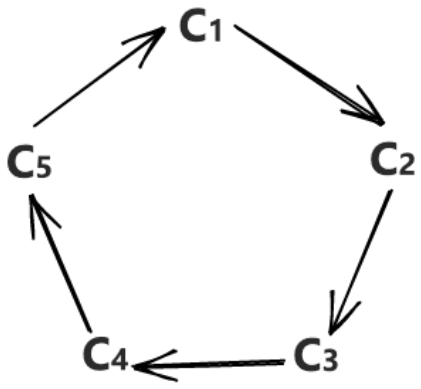
```
>>> px1 = 25188394464646109948510273140890967300336480323249509563736890716485663003998
>>> py1 = 52282638831520867937251910496569429050172994974736170142589370715506453984990
>>> P1 = kreive.Point(elk, px1, py1)

>>> px2 = 11141992422541985370375341632082759953336624834413464061786546691140921198184
>>> py2 = 43265706988717716161675945387856481761664467552925908377907353758987668001233
>>> P2 = kreive.Point(elk, px2, py2)

>>> px3 = 12464916109265137941454695641342886159807701409210595851795130643628881675435
>>> py3 = 11042397349299068392662351488486307711344817425218002643019252375791237993000l
>>> P3 = kreive.Point(elk, px3, py3)

>>> px5 = 24309571957507296510824039000827700039932388769329954644451836668304586629820
>>> py5 = 23686607278159503604469419581577981516062184158831410838875870589229645464794
>>> P5 = kreive.Point(elk, px5, py5)
```

Bronius savo viešąjį raktą įterpia savo nuožiūra arba atsitiktinai į viešųjų raktų rinkinį, pvz., į 4-ąją poziciją. Toliau naudosime subjektui **Bronius** raktų porą (*z*, *P*):

```
>>> z = 67924357263940576683310086449679921151167932554419082853763348853948356882665
>>> px4 = 97569817079350850888698776003651547474639826590171615614759618418511727946339
>>> py4 = 10859199313565609751572522720638709518262836109642447593919597497427229630014 9
>>> P4 = kreive.Point(elk, px4, py4)
```

Broniaus pranešimas ***t*** ir apskaičiuota santrauka $h_t = \mathbf{H}(t)$:

```
>>> t = "Labas Aldona!"
>>> ht = sha256(t)
>>> print("ht =", ht)
ht = 90393738114187435939744967269771007382521481775584304776635022915877227838477
```



12 pav. Supaprastinta **Broniaus** žiedinio parašo žiedo vizualizacija

Bronius formuoja parašą **σ** pranešimui ***t*** su savo privačiuoju raktu (PR_B) z ir jo sudarytu viešųjų raktų rinkiniu P_1, P_2, P_3, P_4, P_5:

1. Sujunkite viešuosius raktus $R=P_1\|P_2\|\ldots\|P_5$:

```
>>> u1 = str(P1.x) + str(P1.y)
>>> u2 = str(P2.x) + str(P2.y)
>>> u3 = str(P3.x) + str(P3.y)
>>> u4 = str(P4.x) + str(P4.y)
>>> u5 = str(P5.x) + str(P5.y)

>>> R = u1 + u2 + u3 + u4 + u5
>>> print("R =", R)
R = '25188394464646109948510273140890967300336 48...588314108388758705892296454647 94'
```

2. Apskaičiuokite tašką $HR = \mathbf{H(R)} * \boldsymbol{G}$ ant elipsinės kreivės:

```
>>> hr = sha256(R)
>>> print("hr =", hr)
hr = 84941730785316339722767219760397580979360503914824390585682219327499416806469

>>> HR = hr * elk.g
>>> print("HRx =", HR.x, "\nHRy =", HR.y)
HRx = 76162088898217352844297536538528767347935336655319599773880587453101591677 76
HRy = 85073994337600418464883237435400493368838911679594649846848555615346112537 32
```

3. Apskaičiuokite bendrą anonimizuotą viešą raktą (angl. *Key Image*) $P = z * HR$:

```
>>> P = z * HR
>>> print("Px =", P.x, "\nPy =", P.y)
Px = 10513184720889542565419624019707313887818203688001878858665334245221348562239 3
Py = 45292101864647635482725187791814301645688625584810879748709223325651448958113
```

4. Apskaičiuokite **rpht** = $\mathbf{R}\|P\|h_t$:

```
>>> rpht = R + str(P.x) + str(P.y) + str(ht)
>>> print("rpht =", rpht)
rpht = 25188394464646109948510273140890967...252148177558430477663502291587722783847 7
```

5. Sugeneruokite atsitiktinį skaičių α ir patikrinkite ar galioja sąlyga $1 \leq \alpha \leq n$-1:

```
>>> al = secrets.randbelow(elk.field.n)
>>> print("al =", al)
al = 10464238866091601065486222748610588282226109911282268219612680278046728451777 9
>>> 1 <= al <= elk.field.n-1
True
```

6. Apskaičiuokite $c_5 = \mathbf{H(rpht}\|\alpha * G\|\alpha * HR\mathbf{)}$, pradėdami nuo sekančio žiedo nario:

```
>>> alG = al * elk.g
>>> alHR = al * HR
>>> c5 = sha256(rpht + str(alG.x) + str(alG.y) + str(alHR.x) + str(alHR.y))
>>> print("c5 =", c5)
c5 = 34418213609155717679975855096028241352903734715564490092533102403205075064035
```

7. Sugeneruokite atsitiktinius skaičius $1 \leq r_1, r_2, r_3, r_5 \leq n\text{-}1$ likusiems žiedo nariams (**Aldonos** pozicija 4 praleiskime, nes vėliau bus apskaičiuota reikšmė r_4):

```
>>> r1 = secrets.randbelow(elk.field.n)
>>> print("r1 =", r1)
r1 = 11716589737894886957591048189071887367127599026994585198020320236300811940423
>>> 1 <= r1 <= elk.field.n-1
True

>>> r2 = secrets.randbelow(elk.field.n)
>>> print("r2 =", r2)
r2 = 641718319398514180184848689860409318950227884334469865180875542630077902791 9
>>> 1 <= r2 <= elk.field.n-1
True

>>> r3 = secrets.randbelow(elk.field.n)
>>> print("r3 =", r3)
r3 = 248133337816634443761190332467325428862285963910611750794826950814010559172 77
>>> 1 <= r3 <= elk.field.n-1
True

>>> r5 = secrets.randbelow(elk.field.n)
>>> print("r5 =", r5)
r5 = 19797215335872796306208142349719933375144280213751938419759417017092580125474
>>> 1 <= r5 <= elk.field.n-1
True
```

8. Apskaičiuokite $c_1 = \mathbf{H(rpht} \| r_5 * G + c_5 * P_5 \| r_5 * HR + c_5 * P\mathbf{)}$:

```
>>> q51 = r5 * elk.g + c5 * P5
>>> q52 = r5 * HR + c5 * P
>>> c1 = sha256(rpht + str(q51.x) + str(q51.y) + str(q52.x) + str(q52.y))
>>> print("c1 =", c1)
c1 = 95361518101455791571482154725259170590535185524868458924885977690582591214 42
```

9. Apskaičiuokite $c_2 = \mathbf{H(rpht} \| r_1 * G + c_1 * P_1 \| r_1 * HR + c_1 * P\mathbf{)}$:

```
>>> q11 = r1 * elk.g + c1 * P1
>>> q12 = r1 * HR + c1 * P
>>> c2 = sha256(rpht + str(q11.x) + str(q11.y) + str(q12.x) + str(q12.y))
>>> print("c2 =", c2)
c2 = 81676929804124314910891448186987418799011665304311005399749805678257640031543
```

10. Apskaičiuokite $c_3 = \mathbf{H(rpht} \| r_2 * G + c_2 * P_2 \| r_2 * HR + c_2 * P\mathbf{)}$:

```
>>> q21 = r2 * elk.g + c2 * P2
>>> q22 = r2 * HR + c2 * P
>>> c3 = sha256(rpht + str(q21.x) + str(q21.y) + str(q22.x) + str(q22.y))
>>> print("c3 =", c3)
c3 = 53758181727606796397216995224252910463335121719221402688643298263334089320351
```

11. Apskaičiuokite $c_4 = \mathbf{H(rpht} \| r_3 * G + c_3 * P_3 \| r_3 * HR + c_3 * P\mathbf{)}$:

```
>>> q31 = r3 * elk.g + c3 * P3
>>> q32 = r3 * HR + c3 * P
>>> c4 = sha256(rpht + str(q31.x) + str(q31.y) + str(q32.x) + str(q32.y))
>>> print("c4 =", c4)
c4 = 321096852440882399570010505606616428085451647405253977887500700875429882699 81
```

12. Pririškite **Aldonos** privatų raktą z, apskaičiuodami $r_4 = \alpha - z \bullet c_2 \bmod n$:

```
>>> r4 = (al - z * c4) % elk.field.n
>>> print("r4 =", r4)
r4 = 413051726765775397648204706448602747416812100769687026879674833877365449643 29
```

13. Žiedinis parašas $\sigma = (c_1, r_1, r_2, r_3, r_4, r_5, P)$ santraukai h_m yra
Pasirašyti$(x, h) = \sigma = (c_1, r_1, r_2, r_3, r_4, r_5, P) =$

```
>>> print("c1 =", c1, "\nr1 =", r1, "\nr2 =", r2, "\nr3 =", r3, "\nr4 =", r4, "\nr5
=", r5, "\nPx =", P.x, "\nPy =", P.y)
c1 = 95361518101455791571482154725259170590535185524868458924885977690582591214 42
r1 = 11716589737894886957591048189071887367127599026994585198020320236300811940423
r2 = 64171831939851418018484868986040931895022788433446986518087554263007790279 19
r3 = 24813333781663444376119033246732542886228596391061175079482695081401055917277
r4 = 64652333962940150717421592960048474154230044006800558915274038269928469379174
r5 = 413051726765775397648204706448602747416812100769687026879674833877365449643 29
Px = 10513184720889542565419624019707313887818203688001878858665334245221348562239 3
Py = 45292101864647635482725187791814301645688625584810879748709223325651448958113
```

14. Bronius anoniminiu kanalu siunčia **Aldonai** viešųjų raktų rinkinį P_1, P_2, P_3, P_4, P_5, pranešimą t, žiedinį parašą σ.

Aldona gauna viešųjų raktų rinkinį P_1, P_2, P_3, P_4, P_5, žiedinį parašą σ pranešimui t. Žiedinis parašas $\sigma = (c_1, r_1, r_2, r_3, r_4, r_5, P)$ pranešimui t yra patikrinamas naudojant anonimizuotą viešąjį raktą (**VR**) P ir su juo susietus viešuosius raktus P_1, P_2, P_3, P_4, P_5:

1. Apskaičiuokite pranešimo santrauką $h_t = \mathbf{H}(t)$:

```
>>> ht = sha256(t)
>>> print("ht =", ht)
ht = 90393738114187435939744967269771007382521481775584304776635022915877227838477
```

2. Sujunkite viešuosius raktus $R = P_1 \| P_2 \| \ldots \| P_5$:

```
>>> u1 = str(P1.x) + str(P1.y)
>>> u2 = str(P2.x) + str(P2.y)
>>> u3 = str(P3.x) + str(P3.y)
>>> u4 = str(P4.x) + str(P4.y)
>>> u5 = str(P5.x) + str(P5.y)

>>> R = u1 + u2 + u3 + u4 + u5
>>> print("R =", R)
R = '25188394464646109948510273140890967300336 48...588314108388758705892296454647 94'
```

3. Apskaičiuokite tašką $HR = \mathbf{H(R)} * \boldsymbol{G}$ ant elipsinės kreivės:

```
>>> hr = sha256(R)
>>> print("hr =", hr)
hr = 84941730785316339722767219760397580979360503914824390585682219327499416806469

>>> HR = hr * elk.g
>>> print("HRx =", HR.x, "\nHRy =", HR.y)
HRx = 761620888982173528442975365385287673479353366553195997738805874531015916777 6
HRy = 850739943376004184648832374354004933688389116795946498468485556153461125373 2
```

4. Apskaičiuokite $\mathbf{rpht} = \mathbf{R}\|P\|\boldsymbol{h}_t$:

```
>>> rpht = R + str(P.x) + str(P.y) + str(ht)
>>> print("rpht =", rpht)
rpht = 251883944646461099485102731408909 67...8252148177558430477663502291587722783847
```

5. Apskaičiuokite $c'_2 = \mathbf{H(rpht}\|z'_1\|z''_1\mathbf{)}$, kai $z'_1 = \boldsymbol{r_1} * G + c_1 * \boldsymbol{P_1}$ ir $z''_2 = \boldsymbol{r_1} * HR + c_1 * \boldsymbol{P}$:

```
>>> z11 = r1 * elk.g + c1 * P1
>>> z12 = r1 * HR + c1 * P
>>> c2s = sha256(rpht + str(z11.x) + str(z11.y) + str(z12.x) + str(z12.y))
>>> print("c2s =", c2s)
c2s = 816769298041243149108914481869874187990116653043110053997498056782576400315 43
```

6. Apskaičiuokite $c'_3 = \mathbf{H(rpht}\|z'_2\|z''_2\mathbf{)}$, kai $z'_2 = \boldsymbol{r_2} * G + c_2 * \boldsymbol{P_2}$ ir $z''_2 = \boldsymbol{r_2} * HR + c_2 * \boldsymbol{P}$:

```
>>> z21 = r2 * elk.g + c2s * P2
>>> z22 = r2 * HR + c2s * P
>>> c3s = sha256(rpht + str(z21.x) + str(z21.y) + str(z22.x) + str(z22.y))
>>> print("c3s =", c3s)
c3s = 537581817276067963972169952242529104633351217192214026886432982633340893203 51
```

7. Apskaičiuokite $c'_4 = \mathbf{H(rpht}\|z'_3\|z''_3\mathbf{)}$, kai $z'_3 = \boldsymbol{r_3} * G + c_3 * \boldsymbol{P_3}$ ir $z''_3 = \boldsymbol{r_3} * HR + c_3 * \boldsymbol{P}$:

```
>>> z31 = r3 * elk.g + c3s * P3
>>> z32 = r3 * HR + c3s * P
>>> c4s = sha256(rpht + str(z31.x) + str(z31.y) + str(z32.x) + str(z32.y))
>>> print("c4s =", c4s)
c4s = 321096852440882399570010505606616428085451647405253977887500700875429882699 81
```

8. Apskaičiuokite $c'_5 = \mathbf{H(rpht}\|z'_4\|z''_4\mathbf{)}$, kai $z'_4 = \boldsymbol{r_4} * G + c_4 * \boldsymbol{P_4}$ ir $z''_4 = \boldsymbol{r_4} * HR + c_4 * \boldsymbol{P}$:

```
>>> z41 = r4 * elk.g + c4s * P4
>>> z42 = r4 * HR + c4s * P
>>> c5s = sha256(rpht + str(z41.x) + str(z41.y) + str(z42.x) + str(z42.y))
>>> print("c5s =", c5s)
c5s = 344182136091557176799758550960282413529037347155644900925331024032050750640 35
```

9. Apskaičiuokite $c'_1 = \mathbf{H(rpht}\|z'_5\|z''_5\mathbf{)}$, kai $z'_5 = \boldsymbol{r_5} * G + c_5 * \boldsymbol{P_5}$ ir $z''_5 = \boldsymbol{r_5} * HR + c_5 * \boldsymbol{P}$:

```
>>> z51 = r5 * elk.g + c5s * P5
>>> z52 = r5 * HR + c5s * P
>>> c1s = sha256(rpht + str(z51.x) + str(z51.y) + str(z52.x) + str(z52.y))
```

```
>>> print("c1s =", c1s)
c1s = 95361518101455791571482154725259170590535185524868458924885977690582591214442
```

10. Patikrinkite ar žiedinis parašas yra tikras, t.y. ar galioja lygybė $c'_1 = c_1$:

```
>>> c1s == c1
True
```

Patikrinti(P_1, ..., P_n, **σ**, h') = $P \in \{True, False\} \equiv \{1, 0\}$.

Tikrintoja **Aldona** priima parašą, jeigu tenkinamos visos aukščiau pateiktos sąlygos, kitais atvejais parašą atmeta.

Užduotys Monero žiediniams parašams.

1. Turėdami viešųjų raktų rinkinį P_1, ..., P_n su į jį įterptu viešuoju raktu $P_?$ ir pasirašančiajam priklausančiu privačiu raktu z, pranešimą t, atsitiktinį skaičių α ir atsitiktines r_1, ..., r_n vertes, nustatykite, kuriam iš pokalbio metu tarp **Aldonos** ir **Broniaus** toliau pateiktų, pagal poziciją įterpiant viešąjį raktą, žiedinio parašo $r_?$ komponentės apskaičiavimui buvo panaudotos šios reikšmės:

1.

```
#Dviejų viešųjų raktų rinkinys, kai pasirašančio viešas raktas įterptas į 1 poziciją
z = 832872413285084623106331029503542048232438227174327070999895626816247210196 6
px1 = 4400172112073334995437365267625415159043810695195809343695031908879654332227 2
py1 = 2695857889288733023754623028861226380488852839653918203485935419774332408489
P1  = kreive.Point(elk, px1, py1)

px2 = 5643038696547405296436166003918424541693393313368654040090566753434162127881 1
py2 = 2249842545543175775692748866608253568084150564979260996396688071207261781550 0
P2  = kreive.Point(elk, px2, py2)

#Pranešimas
t = "Labas Broniau!"

#Atsitiktiniai skaičiai
al = 810998342772794717663955398859044130962004537668843813677958150468951735702 19
r2 = 573392315116403178154178203920632048485611300349459028220279924047659243259 24
```

2.

```
#Trijų viešųjų raktų rinkinys, kai pasirašančio viešas raktas įterptas į 3 poziciją
px1 = 587287034109581022426117368816121378577845297713070967669612637301089178971 36
py1 = 788465395076667843479964962035264167764528485322060464330449360097890548008 25
P1  = kreive.Point(elk, px1, py1)

px2 = 332362282349194695464949336486132544287047642072285628697876770563121203542 49
py2 = 987739665935022816220163039933195809288649063187657354891341949849236614609 70
P2  = kreive.Point(elk, px2, py2)

z = 883899380536903189818478937161608672652774948094937070350779304855008697692 56
px3 = 961149288233974440265395364996539933337854924839348781230927713675759307879 09
py3 = 797851647256951530643771298272068400440953998824733626082070376254391997998 60
P3  = kreive.Point(elk, px3, py3)

#Pranešimas
t = "Labas Aldona!"

#Atsitiktiniai skaičiai
al = 114315250671031963612243681194046708128105492652265948740693308149414598857721
r1 = 584006497711195523146164216763273341755024779339785818165232919366739334941 41
r2 = 932959106229753916419651776333869961802390663912366726350148505322875655749 19
```

3.

```
#Keturių viešųjų raktų rinkinys, kai pasirašančio viešas raktas įterptas į 3 poziciją
px1 = 718213327215827001764194434392937515955207935113288751561992956736255793859 24
py1 = 110496601853806985083519490759082387560226318451394388493469040967605801828729
P1  = kreive.Point(elk, px1, py1)

px2 = 441571791735912589589932430357388946652522986751000144176644384710157857280 49
py2 = 136822898936247629176027983154026352435119131681937358932008047054990149166 92
P2  = kreive.Point(elk, px2, py2)

z = 339248542502830241727125796536079741982709976096260873018012269797806619327 22
px3 = 695312993493433728058421428211914741683878327338509267643546429656716448457 34
py3 = 677423848931607683761288508763281086335577200620005182828441851954504973732 00
P3  = kreive.Point(elk, px3, py3)

px4 = 125286546348110907582163353759304443287181301876679227005073098286081833882 51
py4 = 396902296896113081491581681572030260936410855401342775652500103206327608988 93
P4  = kreive.Point(elk, px4, py4)

#Pranešimas
t = "Kada galėtume susitikti."

#Atsitiktiniai skaičiai
al = 956801003183555243408468773242688569543205167423357877095492412634913983492 81
r1 = 239373421937447849733253818421469575456573527901019765154015034775924904946 89
r2 = 542061841276017905178884031141525024389056340969876975857521447802986801483 47
r4 = 370209829062389990296364151873029631325815128145076461304488364727337259019 35
```

4.

```
#Penkių viešųjų raktų rinkinys, kai pasirašančio viešas raktas įterptas į 2 poziciją
px1 = 64466237149283990662131326379832342708311844148659743658961252302225550399979
py1 = 55611335126490961186310287641259312249410746660829948471469165977852807921311
P1  = kreive.Point(elk, px1, py1)

z = 53284654007069018082193003883376488566104204190265017343711131860615837 3971
px2 = 45035044523474950884222253255184748855702046059252744264261238265117374337284
py2 = 16902167268153629113081940549465392858043540364598003035256393448087818713763
P2  = kreive.Point(elk, px2, py2)

px3 = 42079581408390993340591950901606868409733921997425770488877204330681196611587
py3 = 63774140256676857577476429538065036037231077480581677331949550978975137574196
P3  = kreive.Point(elk, px3, py3)

px4 = 60997315653220357422075298945245954946243594195536248729056059519730776652508
py4 = 52804400541928918693402410318540056478877285150525750660626128486348174494977
P4 = kreive.Point(elk, px4, py4)

px5 = 10804819573553617853262896014283744868405070157434319290729494075394538149001 1
py5 = 41619732710768511520304494690557424467809398983882486380838831140763344648149
P5  = kreive.Point(elk, px5, py5)

#Pranešimas
t = "Susitikime vakare."

#Atsitiktinės skaičiai
al = 38138109069112283298470891680917550412226490358293624451540158117526128005 43
r1 = 77497605548678247654476547421444598523731525808975865370681022435497920760585
r3 = 41397480192552498686490706901599130188105949182575779353853457629106219486237
r4 = 72747403843258892448414397653086421047988411278967878051104485642730079921184
r5 = 50273494184434155295886966281928444701218269121878784993282903251427589632096
```

Žiedinio parašo komponentė $r_? = r$:

```
1. r = 84383186469669092640887892307277498051689806790580255246955304187238586782373
2. r = 16306017504825298874913804376809414464971441059381260060480744708665276318927
3. r = 70575836451675031974552946945789976679175701798310614457392448188300535545139
4. r = 19245285633335607755987940256310688198352703451950447067066672963696044559930
```

2. Turėdami viešųjų raktų rinkinį P_1, ..., P_n, pranešimą t, parašą $\sigma = (c_1, r_1, ..., r_n, P)$, nustatykite, kurie **Aldonos** ir **Broniaus** pranešimams suformuoti parašai yra galiojantys, panaudojant šias reikšmes:

1.

```
#Dviejų viešųjų raktų rinkinys
px1 = 928655551830725815982349283291336664074325044511670983489000864589630489750 93
py1 = 297426910323799262962253408170629459924988875948142095010232402096086185558 43
P1  = kreive.Point(elk, px1, py1)

px2 = 901545698325655466651125221278714906245836454473637782890059874700430623194 44
py2 = 100063128047863047373258827246752183787829240159032693047670673743702892594630
P2  = kreive.Point(elk, px2, py2)

#Pranešimas
t  = " Šalia akmenuoto kelio."

#Parašas
c1 = 107308791570533023720696024963598246666838818018984387035559892089459536720882
r1 = 333700194216288298642829737974500830904050756053772416211515623340534567493 94
r2 = 738756945015181138553746008934204734982992171502466362541145403971125328007 20
px = 102506266228483850598533298747190269910839040082450076210330481815264834027791
py = 956682055644646189847790626686229987307705023986162917593552941199392356418 05
P  = kreive.Point(elk, px, py)
```

2.

```
#Dviejų viešųjų raktų rinkinys
px1 = 708797070128116742409268784381774842245435147481667493001867464604345683383 49
py1 = 533183786767008417520706761530625497497204832184098969880165912802072448903 69
P1  = kreive.Point(elk, px1, py1)

px2 = 960478706509826615143195342825643047355488326568907950191494641603015880034 16
py2 = 350615777115191290452225960751153741918839488282901393069529433973240983342 18
P2  = kreive.Point(elk, px2, py2)

#Pranešimas
t  = " Šalia seno ąžuolo."

#Parašas
c1 = 349727592949918630531169681805804685174597522173721580342282271810132840463 94
r1 = 714096296748370223633733810059678944278153460703972129830789046846214849948 60
r2 = 750501046744913327899681450629562532287150094911151116128592137297244183760 05
px = 587539863079743988556992173523028808542507217727924607977258426388178992185 97
py = 181365175422518979858748608641180187420906860190583058112157465270574097528 45
P  = kreive.Point(elk, px, py)
```

3.

```
#Trijų viešųjų raktų rinkinys
px1 = 537139034060105102927015035510787456771318188850128830788085827476005255595550
py1 = 184576965255382585770327622697490205360412648965178611273298424648978061829 52
P1  = kreive.Point(elk, px1, py1)

px2 = 10336022474444309825181747812651186221539805399674711391670179372377272899937 2
py2 = 41151292748499649090025899398708305288892896276838521168147370981874049731695
P2  = kreive.Point(elk, px2, py2)

px3 = 97924519194668135706827317700984392713331129354602027057383183648363462083123
py3 = 38372554197529940073052154270669384513072743779311091013482966880693670504434
P3  = kreive.Point(elk, px3, py3)

#Pranešimas
t  = "Šalia didelio kelmo."

#Parašas
c1 = 24039816543716924780084382467333767859141312669271529158802290856181627356345
r1 = 48885999825653889454667650757897318676200944615982924567838951767687262225049
r2 = 57622523114287648248812763048045802543870561729823264539104120722223622507961
r3 = 81576677071780635287406083776108302529522984940862181169217262645872383147694
px = 40163787708410509093240711475795024864656284959299889508571982196043813278699
py = 50398970068325316931052171435882094355494659809088680730788726369844400103272
P  = kreive.Point(elk, px, py)
```

4.

```
#Keturių viešųjų raktų rinkinys
px1 = 97924519194668135706827317700984392713331129354602027057383183648363462083123
py1 = 38372554197529940073052154270669384513072743779311091013482966880693670504434
P1  = kreive.Point(elk, px1, py1)

px2 = 76124843048995436811290843479035371926908451866680686343235588028842483341690
py2 = 70797959199909951903051764628970933277916813606970380040202051985351171127566
P2  = kreive.Point(elk, px2, py2)

px3 = 61085166652789954796215222519362128464175290709611008541334040640784959744061
py3 = 97814021737289204197690650961937031622496173685737257854105342163378703043070
P3  = kreive.Point(elk, px3, py3)

px4 = 10613400432505849299643010801565479084166532949142757290132467610824943191118 7
py4 = 18883702573475720328368091265366408486338995370674918656864839296322478763293
P4  = kreive.Point(elk, px4, py4)

#Pranešimas
t  = "Iki greito."
```

```
#Parašas
c1 = 945348812078832991880458966624503580858960476466981903437619840793710516898 09
r1 = 555568294863905796401680525722634272852739243727008181016223154062405343447 68
r2 = 322849455753224318680826990327447167110422793743520727030782977766812292539 6
r3 = 915293602237630402050450651329728486666098691511774018405067360243247971222 43
r4 = 104068157001129309025064675526031542207954866926574617440898734276688705112680
px = 112554138946071335271639189562371871395087493898761112689251882684334466654366
py = 707024434941219558883286699082113858041362325479726341615642356524348125097 60
P  = kreive.Point(elk, px, py)
```

5.

```
#Penkių viešųjų raktų rinkinys
px1 = 110070367996902822946447538907631777520921370709586263297324111579168758861 22
py1 = 245317635483308082706776269459803666915017128480929271309770357926697208603 57
P1  = kreive.Point(elk, px1, py1)

px2 = 813511096621696156534039696719151798761029421403171089678250756755207765960 36
py2 = 105097308365999837760452587556438885744777995763916077469373176550421422823963
P2  = kreive.Point(elk, px2, py2)

px3 = 773548020014308875989826871394183683621404677034033458429086574237030326297 01
py3 = 328582292346394900477327942511199942012187836115197799638151031771055936650 35
P3  = kreive.Point(elk, px3, py3)

px4 = 775247666802316943433965271606579823202855079690678634116518335946382670558 23
py4 = 200694456459004209565631852270401973076600860047624698219008788770454227285 3
P4  = kreive.Point(elk, px4, py4)

px5 = 538382256339269549773458382393660887729745531692518943293052982945446610120 23
py5 = 173483284103747269839029350868903546865219228642776269217045167776748872686 43
P5  = kreive.Point(elk, px5, py5)

#Pranešimas
t  = "Iki pasimatymo."

#Parašas
c1 = 711347494218281848058455249713898425033867554465495734198519519373562148552 63
r1 = 435286134318140813359320085514882972720130597404943847063067749137566731748 85
r2 = 110820237801856040610156692615898720526133214526771871702153862447982904519025
r3 = 793625463232918561390427438326445531970241828587998571883052176130955955225 41
r4 = 875661751879793815022266915547162673215549317992941092010166650070298499307 21
r5 = 593797234161605323552975332083804303102015698491044780202329115974557738109 64
px = 430922457570602618316987126094520283661704318717197215688677686048244564865 66
py = 698003258120678273407619630770789822262297030208827739013753626570628686098 54
P  = kreive.Point(elk, px, py)
```

Tik trys žiediniai parašai $\mathbf{\sigma} = (c_1, r_1, \ldots r_n, P)$ galioja.

3. Turėdami viešųjų raktų rinkinį P_1, ..., P_n, parašą $\sigma = (c_1, r_1, ..., r_n, P)$, nustatykite, kuriems **Aldonos** ir **Broniaus** pranešimams t buvo suformuotas parašas, panaudojant pateiktas reikšmes:

1.

```
#Dviejų viešųjų raktų rinkinys
px1 = 900987527425650068956098029873500109566712120845766633488710264904263624725 25
py1 = 867984674416515736997152785834664015866618516828361835177455838502387478655 14
P1  = kreive.Point(elk, px1, py1)

px2 = 697788443959638067730189758785554790036503654774027402661747207278506155582 90
py2 = 108576521991799194792357547108275040716786710135938326103512731947348758836929
P2  = kreive.Point(elk, px2, py2)

#Parašas
c1 = 578024490268927432240141393116495082439925788048352214733011269497223502679 95
r1 = 988177363979527416664458159420669226088708014137547739990085268020242045973 54
r2 = 103973983920667243848925484782223532541524622037958902018713778075941324563599
px = 378451393586732025390645162147101155418690229893318998706071635772205054944 87
py = 101500848491014777002070989122875505479454023071564293786061153047512131533179
P  = kreive.Point(elk, px, py)
```

2.

```
#Trijų viešųjų raktų sąrašas
px1 = 569041374044218508149536773854310288912078990867818157541870117640544238500 31
py1 = 427744913509150569431020419284399954689936295479000718238815689943701694948 58
P1  = kreive.Point(elk, px1, py1)

px2 = 945969338267156147518200052543705815276737595452434842039124113928289578631 30
py2 = 518608898326037866458975437889855461491350890599305631142158956912002493373 16
P2  = kreive.Point(elk, px2, py2)

px3 = 637287376019656536815734292536590147009714619386793434398037744757557781046 76
py3 = 701273757353166223931372482149905170792945358738948768652372717490512661102 86
P3  = kreive.Point(elk, px3, py3)

#Parašas
c1 = 296885806868391678301148060952106097113194151986854851583983844722740005178 68
r1 = 706401290110133167255592708543266607733146932859921078811743468553844540768 61
r2 = 723727790587833811204213036832272172039370499572610193894763502915038057625 9
r3 = 106469142046452120717912973772553017377705779248341119359804700376404484366375
px = 533505865677300835779610316422206823594552571349537036285732246333592685238 85
py = 249848681080684349839552946804805242629612328860675289555556931586624740176 70
P  = kreive.Point(elk, px, py)
```

3.

```
#Keturių viešųjų raktų rinkinys
px1 = 330262892871000790558763569480634825328407267977857639998608703783325812051870
py1 = 633904743797671354812451539543508558931859629907870689952198657073616793784 7
P1  = kreive.Point(elk, px1, py1)

px2 = 210166864473733032806768997346667310861322714173121477620615841336046705644260
py2 = 332233364569169183313226427127246769586982577742787858862956091660371399845200
P2  = kreive.Point(elk, px2, py2)

px3 = 476596747909372712519412353403130865605165187655997065549250878532334726739240
py3 = 597154574797719388255103687713896889881832704734867977504061433235752068114960
P3  = kreive.Point(elk, px3, py3)

px4 = 912534127964305284499105028922159735947620538045804251937812007007383288235880
py4 = 739669475919672297266299687814047855401422410723568529503822085784014418267 4
P4  = kreive.Point(elk, px4, py4)

#Parašas
c1 = 113812129712904984490415058480063636297526383060653716463823471633059656421365
r1 = 536369257962233791095053377130753089828311788066465816728477196498107109366
r2 = 348170459375610530186153183688590273924672416138257564206665151377304559279 19
r3 = 108149390578143980541071043253975166764350034531987303714989307272434525407710
r4 = 378314280178754646118233491057611761435526344424895493485437070977504958501 39
px = 621566715474667448786129724887840914817278165841774174537482115693718381690870
py = 113428260509699121360534502410504728011716488848828939034357383250926285434745
P  = kreive.Point(elk, px, py)
```

4.

```
#Penkių viešųjų raktų rinkinys
px1 = 110875635346607494356312476340428433614982908876903534498879259846471433334027
py1 = 114619855239918945458148108865638849417250562053681520351419780755737253314020
P1  = kreive.Point(elk, px1, py1)

px2 = 533056677525005610950845271606060036187229751523606339629090774337966398560 06
py2 = 415216572590395579763719080756314054252980307769009416056834866362157720385 77
P2  = kreive.Point(elk, px2, py2)

px3 = 707443699848041656288168193506272518613329433729779530248808669909167705735 44
py3 = 969239676965399639628319391840539379939599281354674768978711468765574597263 76
P3  = kreive.Point(elk, px3, py3)

px4 = 873772111946412065366290298168509502846886437358666530584770002434996213593 37
py4 = 211770376163132490717193467025060711998351586829204946447834225366447005938 76
P4  = kreive.Point(elk, px4, py4)

px5 = 759085660636607369665568989987992767947636286840300304974713603484437986143 1
py5 = 340009132661846866455887234693247302456916691752573309727801210635510393308 35
P5  = kreive.Point(elk, px5, py5)
```

```
#Parašas
c1 = 996639291885688578471329456331201998490334481332358341236691023312093830375 44
r1 = 10689626767780970038708141960631196318761280807821478858944901653392228202240 1
r2 = 70090694151785131465431792990854963337885330968085688092225960991116857707472
r3 = 25992466816936634414357646682808383216460850442521473076807591445929159774575
r4 = 96785410113506663229017850476106621149715595273184144234713965286330083002011
r5 = 16530330109186769696039111810180774152116616862352196882949642965736753875 98
px = 15914241532839462955968286834444759774554164282740591799608605309394410255298
py = 79761042312450325686593731916841998693498939775202239919261677413588111716425
P  = kreive.Point(elk, px, py)
```

Pranešimai ***t***:

1. t_1 = "Kelintą valandą vakare.";
2. t_2 = "19 valandą.";
3. t_3 = "Kurioje vietoje.";
4. t_4 = "Jaukioje parko kavinėje.".