Turinys

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:

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

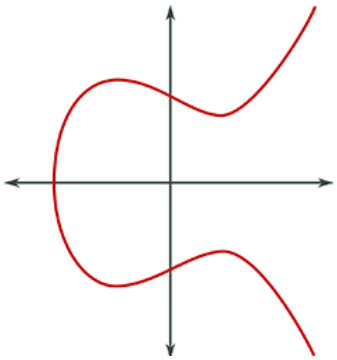
$$y^3 = \lambda(x_1 - x_3) - y_1 \bmod p;$$

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]:

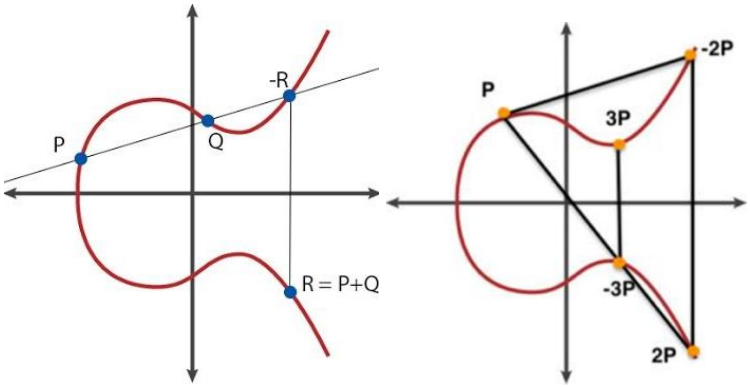
- Pirminis skaičius ***p***, kurio pagalba atliekama aritmetika tarp elipsinės kreivės taškų, ***p*** = 115792089210356248762697446949407573530086143415290314195533631308867097853951 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff;
- *a* ir *b* nurodo kreivės formą arba kitaip taškus 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 = 1145755640413222213188932311970740020298427894445730851180078265879514726267З5
py = 5437254857733224399794189769532681894240187992147038739664484809690481404388З
```

2. Neteisinga raktų pora,

```
px = 484395612939064517590525852527979142027629495260417479958440807170824046352Ю6
py = 361342509567497957985851279195878819566111066729850150718771982535684144051Ю9
```

3. Teisinga raktų pora,

```
px = 611633941399128854538430865215853813870349521620940686401070561515472175816З
py = 858849880781067627250490476224650119926380067431846252086142227094432395102Ю6
```

4. Neteisinga raktų pora,

```
px = 297707568019069957928547185775113530817207969612243441695541544160319639424Ю2
py = 776003316999216499651797683706308777107090698587459424146663485725284497945Ю5
```

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*):

$$y^2 = x^3 + ax + b \ \mathbf{mod}\ \boldsymbol{p},$$

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

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

Dėl to:

- suspaudus viešąjį raktą užtenka saugoti tik x koordinatę ir antrosios y koordinatės ženklą (tai yra, ar y yra lyginis ar nelyginis (modulinis atitikmuo $\boldsymbol{p} - y$));
- 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)
>>> print("y =", hex(y))
y = 0x6e926b0349950d6b84b62e0fb677bc240861553a16efe533f22df00b8536b9c2
```

Kai x = 0x03ce60af5fd2edef1944acb30186bc8583aedadcd4066b4808279247b5bb33dba8, o y lyginė (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)
>>> y = elk.field.p - y
>>> print("y =", hex(y))
y = 0xd2fd750cbeec2cad23e0b21d8adab801bc77c9a243a6c56fa4e64a01022b85dc
```

Kai x = 0x04ea4cc5461592b2833a2c5a0b768c27a4138e77b5af5cd5b3ccb45a601e0aed9ea51df9376d96cd07e74e0bbaab5086fba431893ddd809945f671291c01cea853 ir (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

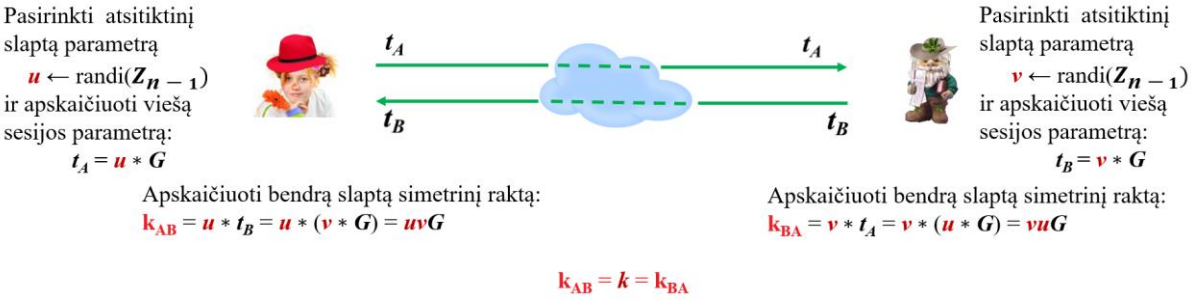
Aldona

```
>>> u = secrets.randbelow(elk.field.n)
>>> print("u=", u)
u =  1108739193027592831410497619167099219984720535801786232284120068431758067863 91
>>> 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.



$$k_{AB} = k = k_{BA}$$

4 pav. Aldona ir Bronius apskaičiuoja bendrą slaptą simetrinį raktą

Aldona

```
>>> kAB = u * tB
>>> print("kABx =", kAB.x, "\nkABy =", kAB.y)
kABx = 320422975719130429445637449879773471830808982909906973373271294676602050569 09
kABy = 595632894877329311145824123345881572639538818089609016756937948317044667852 23
```

Bronius

```
>>> kBA = v * tA
>>> print("kBAx =", kBA.x, "\nkBAy =", kBA.y)
kBAx = 320422975719130429445637449879773471830808982909906973373271294676602050569 09
kBAy = 595632894877329311145824123345881572639538818089609016756937948317044667852 23
>>> print("kBAx =", hex(kBA.x), "\nkBAy =", hex(kBA.y))
kBAx = 0x46d74c385be51f1e806f39b3178e648515a37f84ffb24663ec90089092c8638d
kBAy = 0x83afa08d3b120cb8b1b77702baf6dfd4a2e30bfc80f4f08dbdc245d91f8a03c7
```

$$k_{AB} = k = k_{BA}$$

```
kABx = 320422975719130429445637449879773471830808982909906973373271294676602050569 09 = kBAx
kABy = 595632894877329311145824123345881572639538818089609016756937948317044667852 23 = kBAy
Arba šešioliktaine forma:
kBAx = 0x46d74c385be51f1e806f39b3178e648515a37f84ffb24663ec90089092c8638d = kBAx
kBAy = 0x83afa08d3b120cb8b1b77702baf6dfd4a2e30bfc80f4f08dbdc245d91f8a03c7 = kBAy
```

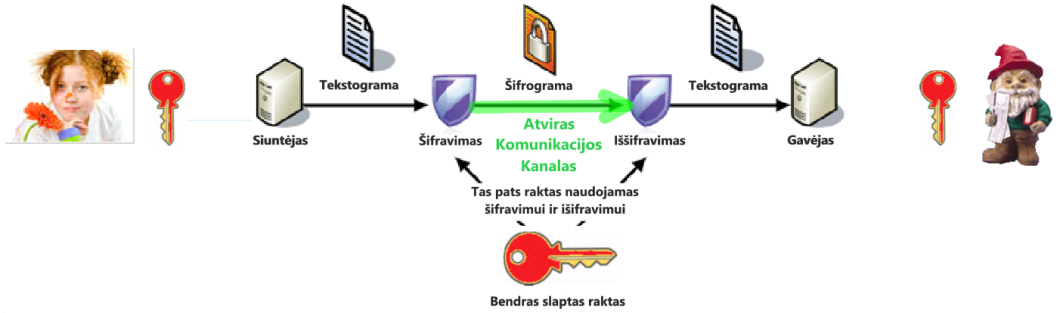
P.S

Kadangi buvote už Bronių ir Aldoną, kad patikrintumėte, ar teisingai apskaičiavote bendrą slaptą simetrinį raktą, įsitikinkite, kad yra tenkinama sąlyga $k_{AB} = k = k_{BA}$:

```
>>> 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*** pateikiama 5 pav.



5 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 tekstogramą/pranešimą

```
>>> k = kAB.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 = os.urandom(16)
>>> enkriptorius = Cipher(algorithms.AES(simetrinis_raktas), modes.GCM(iv)).encryptor()
>>> tekstograma = b"Labas Broniau!"
>>> sifrograma = enkriptorius.update(tekstograma) + enkriptorius.finalize()
>>> zyme = enkriptorius.tag

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

Aldona išsiunčia Broniui IV, žymę, šifrogramą.

Bronius iššifruoja tekstogramą/pranešimą

Bronius gavęs iš Aldonos IV, žymę, šifrogramą iššifruoja tekstogramą/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()
>>> tekstograma = dekriptorius.update(sifrograma) + dekriptorius.finalize()
>>> print("Gauta tekstograma =", tekstograma.decode('utf-8'))
Gauta tekstograma = Labas Broniau!
```

Užduotys Difio Helmano raktų apsikeitimo protokolui naudojant elipsines kreives.

Difio Helmano raktų apsikeitimo protokolo simuliacija naudojant elipsines kreives.