Kriptográfia és Információbiztonság
10. előadás

MÁRTON Gyöngyvér

Sapientia Egyetem, Matematika-Informatika Tanszék
Marosvásárhely, Románia
mgyongyi@ms.sapientia.ro

2025

Miről volt szó?

- a Diffie-Hellman kulcscsere
- digitális aláírások:
  - a Schnorr digitális aláírás
  - az ElGamal digitális aláírás
  - a DSA (Digital Signature Algorithm) vagy DSS (Digital Signature Standard)

Miről lesz szó?

Elliptikus görbe kriptográfia

- ECC görbetípusok: Weierstrass, Montgomery, Edwards görbe
- alapműveletek
- ECC görbék: secp256r1, secp256k1, Curve25519
  - ECDH (kulcsmegosztás)
  - ECDSA (digitális aláírás)
  - ECC - biztonság

Elliptikus görbéken alapuló kriptográfia

- Diophantosz Aritmetika könyvsorozatában (13 kötet, ebből 6 maradt fenn) a következő probléma jelenik meg: határozzuk meg azokat az (x, y) racionális számpárokat, amelyek kielégítik az $y^2 = x^3 - x + 9$ egyenletet,
- egy elliptikus görbe általános alakja:

$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gy^2 + Hx + Iy + J = 0$$

- elliptikus görbékkel Koblitz és Miller az 1980-as évek végén kezdett el foglalkozni, eredményeiket az RSA-val ellentétben nem védték le,
- a kriptográfiában egyszerűbb alakkal dolgoznak, például a Weierstrass alak a következő:

$$y^2 = x^3 + ax + b,$$

- az első kripto standardok 2000-ben jelentek meg,
- a kis kulcsméret, a jó biztonság miatt nagy a népszerűségük, leggyakrabban 256 bites kulcsokkal dolgoznak.

Kriptográfiai kulcsok bitmérete

Table 10.3 Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Source: Certicom

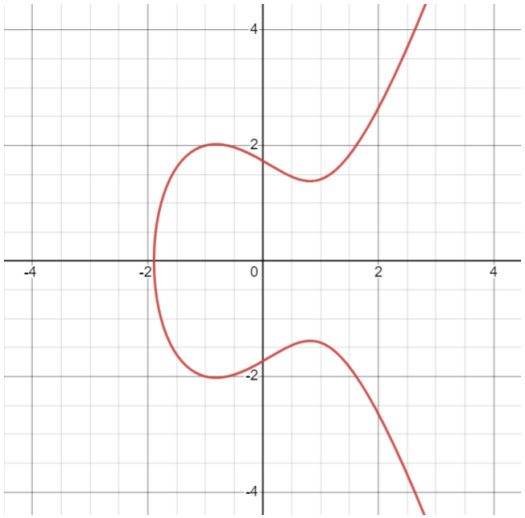
- a DL feltételezés az egész számok (mod p) szerinti multiplikatív csoportjában *már nem számít elég nehéznek*,
- 2019: egy 795 bites prímszám esetében megoldották a DL problémát, azóta legalább 2048 bites prímek használatát írják elő a DL feltételezésen alapuló protokollokban.

Elliptikus görbéken alapuló kriptográfia

- három típusú görbét szoktak használni,
- a valós számok felett értelmezett három típusú görbe grafikus alakja a következő:

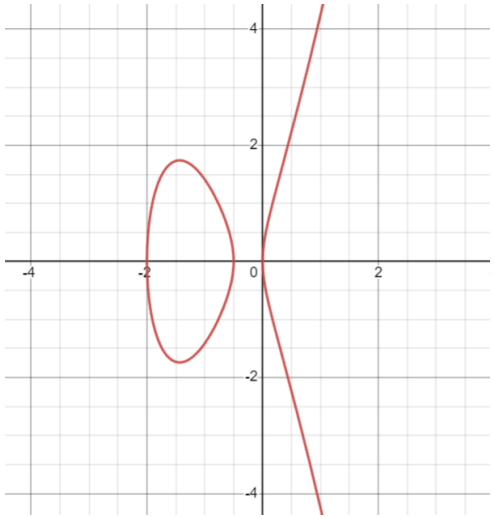
Weierstrass

$y^2 = x^3 - 2x + 3$



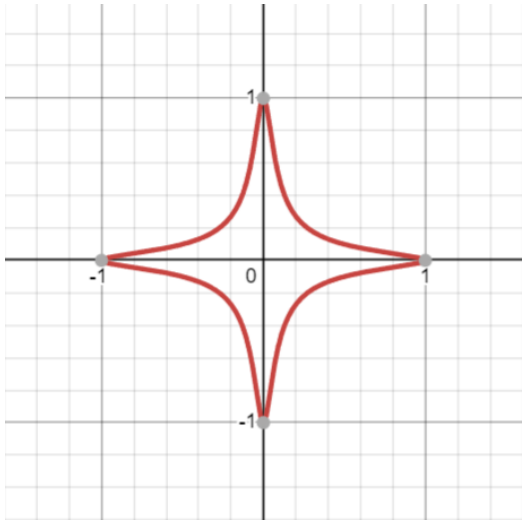
Montgomery

$\frac{1}{4}y^2 = x^3 + \frac{5}{2}x^2 + x$



Edwards

$y^2 + x^2 = 1 - 300x^2y^2$

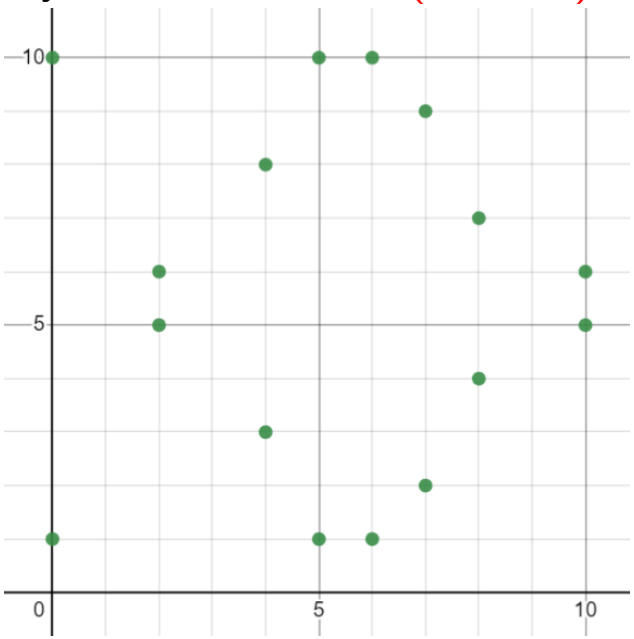


- különböző biztonságot nyújtanak,
- más hatékonyságot jelentenek,
- másképp van értelmezve az összeadás

Elliptikus görbéken alapuló kriptográfia

- egy véges test felett értelmezett görbe grafikus alakját pontok adják,
- a következő görbe esetén a görbe pontjainak száma: $16 + 1$.

$$y^2 \equiv x^3 - 3x + 1 \pmod{11}$$



$(0,1), (0,11-1)$	$1^2 \equiv 0^3 - 3 \cdot 0 + 1$
	$10^2 \equiv 0^3 - 3 \cdot 0 + 1$
$(2,5), (2,11-5)$	$5^2 \equiv 2^3 - 3 \cdot 2 + 1$
	$6^2 \equiv 2^3 - 3 \cdot 2 + 1$
$(4,3), (4,11-3)$	$3^2 \equiv 4^3 - 3 \cdot 4 + 1$
	$8^2 \equiv 4^3 - 3 \cdot 4 + 1$
$(5,1), (5,11-1)$	$1^2 \equiv 5^3 - 3 \cdot 5 + 1$
	$10^2 \equiv 5^3 - 3 \cdot 5 + 1$
$(6,1), (6,11-1)$	$1^2 \equiv 6^3 - 3 \cdot 6 + 1$
	$10^2 \equiv 6^3 - 3 \cdot 6 + 1$
$(7,9), (7,11-9)$	$9^2 \equiv 7^3 - 3 \cdot 7 + 1$
	$2^2 \equiv 7^3 - 3 \cdot 7 + 1$
$(8,4), (8,11-4)$	$4^2 \equiv 8^3 - 3 \cdot 8 + 1$
	$7^2 \equiv 8^3 - 3 \cdot 8 + 1$
$(10,5), (10,11-5)$	$5^2 \equiv 10^3 - 3 \cdot 10 + 1$
	$6^2 \equiv 10^3 - 3 \cdot 10 + 1$

Elliptikus görbéken alapuló kriptográfia

- kriptoban a számításokat valamely $\mathbb{F}_p$ véges test felett végzik, és az alkalmazott alapművelet két pont összege lesz
- például az

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

görbe esetén fenn kell álljon:

  - $a, b \in \mathbb{F}_p$, ahol $p > 3$ prímszám,
  - $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, amely biztosítja, hogy a görbe egyenletének nem lesz kétszeres gyöke,
  - $(x, y) \in \mathbb{F}_p$ a görbe egy pontja lesz, ha kielégíti a görbe egyenletét
- az elliptikus görbét $E/\mathbb{F}_p$-vel jelölik,
- ahhoz hogy az összeadást értelmezni lehessen a görbén található pontokhoz hozzá vesznek egy speciális pontot, a végtelen pontot, amelyet $\mathcal{O}$-val jelölnek,
- a görbe pontjai és az $\mathcal{O}$ pont által meghatározott halmazt $E(\mathbb{F}_p)$-vel jelölik:

$$E(\mathbb{F}_p) = \{(x, y) \mid y^2 \equiv x^3 + ax + b \pmod{p}\} \cup \{\mathcal{O}\}$$

- a számítások az $\mathbb{F}_{p^e}$, $e > 1$ bővített test felett is elvégezhetőek.

Elliptikus görbéken alapuló kriptográfia

- egy elliptikus görbe két pontja alapján könnyedén egy új pontot lehet meghatározni, amely szintén a görbén található,
- legyen P a görbe egy pontja, ekkor két pont összeadására nézve az $E(\mathbb{F}_p)$ halmaz véges, ciklikus Abel csoport lesz
- két pont összeadásának szabálya függ a választott görbétől,
- a $P + P$ műveletet $2P$-vel, a $P + P + P$ műveletet $3P$-vel jelöljük,
- az αP azt jelenti, hogy α-szor adtuk össze a P-t, ahol α tetszőleges pozitív egész szám,
- az αP hatékonyan meghatározható $2log_2\alpha$ csoport művelettel,
- a P pont rendje n, ha fennáll: $qP = \mathcal{O}$,
- G pont generátor elem lesz, ha többszörösei mind különböznek, és $E(\mathbb{F}_p)$ egyik pontját jelölik,
- az $E(\mathbb{F}_p)$ halmaznak mindig lesz egy generátor eleme, de nem minden pontja generátor elem,

Elliptikus görbéken alapuló kriptográfia

- jelölöjük N-el az $E(\mathbb{F}_{p^e})$ halmaz, azaz a görbe pontjainak elemszámát
- ha a P pont rendje n, akkor a P pont kofaktora h ahol $h = \frac{N}{n}$,
- ha d az N egy prímosztója, akkor az $E(\mathbb{F}_p)$-nek bármely P pontjára igaz, hogy az $\frac{N}{d} \cdot P$ pont vagy a semleges elem lesz, vagy egy olyan pont amelynek rendje d.
- a kriptográfiai protokollokban egy $E(\mathbb{F}_p)$ elliptikus görbének valamely n elemű ciklikus alcsoportját alkalmazzák, ahol n prímszám
- kiválasztásra kerül egy P alappont, amelynek rendje n, ezt a következőképpen végzik:
  - kiválasztják a görbe egy tetszőleges Q pontját, és meghatározzák a $P = h \cdot Q$ értéket,
  - ha $P \neq \mathcal{O}$, akkor P lesz az alappont, amelynek rendje tehát egyenlő n-val,
  - ha $P = \mathcal{O}$ akkor választanak egy új Q értéket.
- ha a kofaktor 1, akkor a teljes csoportban történnek a számítások.

Elliptikus görbéken alapuló kriptográfia

- az ECC rendszerek biztonsága az EC diszkrét logaritmus (ECDL) feltételezésen alapszik,
- ECDL probléma: legyen P egy pont az $E(\mathbb{F}_p)$ halmazban, amelynek rendje egyenlő a n egész számmal, ekkor a ECDL probléma azt jelenti, hogy ismerve az $P, \alpha P$ értékeket határozzuk meg az $\alpha \in \mathbb{Z}_n$ pozitív egész számot,
- ECDL feltételezés: megfelelően választott p és n értékek mellett nincs hatékony algoritmus, amely megoldaná az ECDL problémát,
- az ECDL problémát megoldó, leghatékonyabb algoritmus futási ideje, $O(\sqrt{n})$,
- gyakorlatban a p és n nagyságrendje 256 bit kell legyen,
- ha N minden prímosztója kisebb, mint 2^{80}, akkor az ECDL probléma könnyen megoldható, ezért a gyakorlatban olyan görbékkel dolgoznak, amelyek esetében N egyenlő $n, 4n$, vagy $8n$-val, ahol n prímszám,
- hogy az ECDL probléma ne legyen hatékonyan megoldható, előre kiválasztott görbével és alapponttal dolgoznak.

Elliptikus görbe típusok

Weierstrass görbe:

- az $E/\mathbb{F}_p$ elliptikus görbe egyenlete, ahol $p > 3$ prímszám a következő:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

és fennáll: $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$

- a $P = (x_1, y_1)$ és $Q = (x_2, y_2)$ pontok összege a következőképpen van értelmezve:
  - ha $x_1 = x_2$ és $y_1 = -y_2$, akkor $P + Q = \mathcal{O}$
  - másképp $P + Q = (x_3, y_3)$, ahol

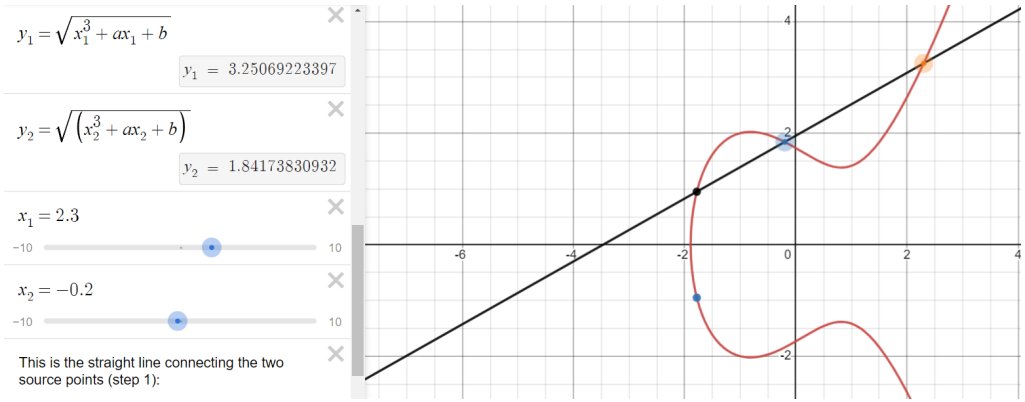
$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= \begin{cases} (y_1 - y_2) \cdot (x_1 - x_2)^{-1}, \text{ha } P \neq Q \\ (3x_1^2 + a) \cdot (2y_1)^{-1}, \text{ha } P = Q, y_1 \neq 0 \end{cases} \end{aligned}$$

- ha az egyik pont $\mathcal{O}$, akkor fennáll: $P + \mathcal{O} = \mathcal{O} + P = P$
- $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ feltétel biztosítja, hogy a görbe egyenletének nem lesz kétszeres gyöke.

Elliptikus görbe típusok

két pont összegét geometriailag a húrmódszer alapján adhatjuk meg:

- legyen P és Q két pont a görbén, ahol $P \neq Q$,
- húzzunk egy egyenest a P és Q pontokon keresztül, bebizonyítható, hogy az egyenes metszeni fogja a görbét egy harmadik pontban, amelynek az x tengelyre való szimmetrikusa lesz a P és Q összege:



https://www.desmos.com/calculator/ialhd71we3

Elliptikus görbe típusok

Montgomery görbe:

- az $E/\mathbb{F}_p$ elliptikus görbe egyenlete, ahol $p > 3$ prímszám a következő:

$$by^2 \equiv x^3 + ax^2 + x \pmod{p}$$

  és fennáll: $b(a^2 - 4) \not\equiv 0 \pmod{p}$
- átalakítható Weierstrass alakra: $x = bu - a/3, y = bv$, de a Weierstrass alak nem mindig hozható Montgomery alakra
- a számítások hatékonyabbak, mint a Weierstrass görbéken,
- a $P = (x_1, y_1)$ és $Q = (x_2, y_2)$ pontok összege a következőképpen van értelmezve:
  - ha $x_2 = x_1$ és $y_2 = -y_1$, akkor $P + Q = \mathcal{O}$
  - ha az egyik pont $\mathcal{O}$, akkor fennáll: $P + \mathcal{O} = \mathcal{O} + P = P$
  - másképp $P + Q = (x_3, y_3)$, ahol

$$\begin{aligned} x_3 &= b\lambda^2 - x_1 - x_2 - a \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= \begin{cases} (y_2 - y_1) \cdot (x_2 - x_1)^{-1}, \text{ha } P \neq Q \\ (3x_1^2 + 2ax_1 + 1) \cdot (2by_1)^{-1}, \text{ha } P = Q \end{cases} \end{aligned}$$

Elliptikus görbe típusok

Edwards görbe:

- az $E/\mathbb{F}_p$ elliptikus görbe egyenlete, ahol $p > 3$ prímszám a következő:

$$x^2 + y^2 \equiv 1 + dx^2y^2 \pmod{p}$$

  és fennáll: $d \in \mathbb{F}_p$ és $d \neq 0, 1$
- átalakítható Weierstrass alakra, de a Weierstrass alak nem mindig hozható Edwards alakra,
- az Edwards görbén is gyorsak a számítások
- az összeadás művelete egyszerűbb, a $P = (x_1, y_1)$ és $Q = (x_2, y_2)$ pontok összege a következőképpen van értelmezve:
  - ha $x_2 = x_1$ és $y_2 = -y_1$, akkor $P + Q = \mathcal{O}$
  - másképp $P + Q = (x_3, y_3)$, ahol

$$\begin{aligned} x_3 &= (x_1y_2 + x_2y_1)(1 + dx_1x_2y_1y_2)^{-1} \\ y_3 &= (y_1y_2 - x_1x_2)(1 - dx_1x_2y_1y_2)^{-1} \end{aligned}$$

Gyakran használt görbék

- 1999-ben a NIST 5 görbét adott meg, amelyek elsősorban hatékonyság és nem biztonsági szempontok alapján kerültek meghatározásra, (https://secg.org/sec2-v2.pdf)
- a NIST-prímek:

$$
\begin{aligned}
P192 &= 2^{192} - 2^{64} - 1 \\
P224 &= 2^{224} - 2^{96} + 1 \\
P256 &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
P384 &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
P521 &= 2^{521} - 1
\end{aligned}
$$

- észrevehető, hogy mindegyik prim felírható valamely 2 hatványösszege, vagy különbségeként
- a $P521$ prímet leszámítva, mindegyik kitevő 32 többszöröse, ami hatékony számításokat tesz lehetővé 32 bites architektúrán

A P256 görbe

- más néven secp256r1
- az egyik legnépszerűbb görbe
- az elnevezés eredete:
  - `sec`: Standards for Efficient Cryptography,
  - `p256`: a p prímszám 256 bites,
  - `r`: random görbe
- Weierstrass típusú görbe: $y^2 \equiv x^3 + ax + b \pmod{p}$, ahol p prímszám:

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$n = 115792089210356248762697446949407573529996955224135760342422259061068512044369$
$a = -3 = 115792089210356248762697446949407573530086143415290314195533631308867097853948$
$b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$
$x_1 = 48439561293906451759052585252797914202762949526041747995844080717082404635286$
$y_1 = 36134250956749795798585127919587881956611106672985015071877198253568414405109$

A P256 görbe

- $P(x_1, y_1)$ pont rendje az n prímszámmal egyenlő,
- a görbe pontjainak a száma megegyezik n-nel, a P pont tehát generátor elem,
- biztonsági szempontból random módon határozták meg a görbe paramétereit:
  - egy kezdeti *seed* alapján, az SHA-1 hash függvényt alkalmazva meghatározásra került egy r érték,
  - fenn kell álljon $r \neq 0$ és $4r + 27 \equiv 0 \pmod{p}$,
  - azok az a, b értékek lesznek megfelelők, amelyekre fennáll:

$$r \cdot b^2 \equiv a^3 \pmod{p},$$

  - az algoritmus adott, de a *seed* érték kiválasztására nincs magyarázat adva!!!
- azt feltételezik, hogy ebben a görbében az ECDL problémát 2^{128} csoportművelet elvégzésével lehet megoldani,
- Diffie-Helmman típusú kulcscsere során a `TLS 1.3` összes implementációja megköveteli ennek a görbének a támogatását,

A secp256k1 görbe

- Weierstrass típusú görbe: $y^2 \equiv x^3 + ax + b \pmod{p}$, ahol p prímszám:

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

- Koblitz görbének hívják: hatékonyan implementálható az αP művelet,
- a generátor $P(x_1, y_1)$ pont rendje az n prímszám,
- a görbe pontjainak a száma megegyezik n-nel,
- a bitcoin ezt a görbét használja, nem bízott meg a NIST `secp256r1` görbéjében,

$n = 115792089237316195423570985008687907852837564279074904382605163141518161494337$
$a = 0$
$b = 7$
$x_1 = 55066263022277343669578718895168534326250603453777594175500187360389116729240$
$y_1 = 32670510020758816978083085130507043184471273380659243275938904335757337482424$

A Curve25519 görbe

- Dan Bernstein tervezte,
- népszerűsége biztonsága és gyorsasága miatt növekvőben van,
- Montgomery típusú görbe: $y^2 \equiv x^3 + ax^2 + x \pmod{p}$, ahol

$$p = 2^{255} - 19$$

  az a legnagyobb prímszám, amely kisebb mint 2^{255},
- a következő Edwards görbévé alakítható át:

$$x^2 + y^2 = 1 + (121665/121666)x^2y^2$$

- az a érték kiválasztása: a lehető legkisebb érték kell legyen, amelyre az ECDL probléma még nehéz,
- a generátor $P(x_1, y_1)$ pont rendje az n prímszám,
- a görbe pontjainak száma: $8n$.

$a = 486662$
$n = 2^{252} + 27742317777372353535851937790883648493$
$x_1 = 9$
$y_1 = 14781619447589544791020593568409986887264606134616475288964881837755586237401$

EC - kulcscsere

- ECDH, Elliptic Curve Diffie–Hellman key exchange
- egy nyilvános csatornán keresztül két fél meg tud állapodni csak egy általuk ismert közös titokban,
- Diffie–Hellman típusú kulcscsere, biztonsága az ECDL feltételezésen alapszik,
- a közös titokból valamely szimmetrikus titkosító kulcsát, vagy egy MAC kulcsot lehet meghatározni,
- X25519:
  - Daniel J. Bernstein javasolta,
  - a CURVE25519 görbe alkalmazása ECDH-ban,
  - az egyik leggyorsabb protokoll, kulcsmérete 256 bit, nincs levédve,
  - a TLS 1.3 által támogatott.
- ECIES (Elliptic Curve Integrated Encryption Scheme):
  - a legszélesebb körben használt hibrid titkosítás,
  - alkalmazásával átmeneti (ephemeral) kulcsot generálnak,
  - a kulcscserét ECDH-val valósítják meg, amelyet aztán az AES-GCM kulcsaként használnak.

EC - publikus kulcsú titkosítás

- EC Public Key Encryption
- egy nyilvános csatornán keresztül, a küldő fél a fogadó publikus kulcsát használva titkosít egy általa kiválasztott random értéket, amelyet a fogadó fél a privát kulcsával vissza tud fejteni,
- a kiválasztott és biztonságosan megosztott random értékből valamely szimmetrikus titkosító kulcsát, vagy MAC kulcs értékét lehet meghatározni
- biztonsága az ECDL feltételezésen alapszik,
- EEECC (ElGamal Encryption Elliptic Curve Cryptography):
  - ElGamal típusú titkosító, ahol a műveletek valamely elliptikus görbe által meghatározott halmazban vannak értelmezve

EC - digitális aláírás

- EC Digital Signature
- egy nyilvános csatornán keresztül, az aláíró fél a privát kulcsával hitelesíteni tud egy üzenetet, amelynek hitelességét az aláíró publikus kulcsát használva bárki le tud ellenőrizni,
- az üzenet lehet valamelyik kommunikáló fél publikus kulcsa, lehet egy blokkláncra felkerülő tranzakció, stb,
- biztonsága az ECDL feltételezésen alapszik,
- ECDSA (Elliptic Curve Digital Signatures Standard):
  - számos szabványon keresztül lehet alkalmazni, ahol a szabványok nem mindig kompatibilisek egymással
  - az egyik leggyakrabban alkalmazott aláírási séma,
- EdDSA (twisted Edward Digital Signature Standard):
  - Daniel J. Bernstein tervezte 2011-ben
  - a Schnorr digitális aláírási sémán alapszik, miután ez 2010 után felhasználhatóvá vált, mert lejárt levédettségi ideje.

ECDH - Elliptic Curve Diffie–Hellman key exchange

- két távoli egység (számítógép, mobileszköz, stb.) kulcscsere mechanizmusára ad megoldást.
- feltételezve, hogy a két egység **A** és **B**, akkor a protokoll a következő:

1. **A** és **B** megegyeznek az E elliptikus görbében, a p prímszám, a görbe egy P pontjának, és a P pont n rendjének értékében
2. az **A** egység meghatározza:
   - $a \xleftarrow{R} \{2, \ldots, n-1\}$ privát kulcs (titokban tartja),
   - $A = aP = (x_A, y_A)$ publikus kulcs,
   - a A-t elküldi **B**-nek,
3. a **B** egység meghatározza:
   - $b \xleftarrow{R} \{2, \ldots, n-1\}$ privát kulcs (titokban tartja),
   - $B = bP = (x_B, y_B)$ publikus kulcs,
   - a B-t elküldi **A**-nak,
4. az **A** egység a közös $K = x_K$ kulcsot a következőképpen határozza meg:
$$aB = (x_K, y_K)$$
5. a **B** egység a közös $K = x_K$ kulcsot a következőképpen határozza meg:
$$bA = (x_K, \hat{y_K}).$$

ECDH - Elliptic Curve Diffie–Hellman key exchange

- helyesség:

$$aB = bA = abP.$$

- görbétől, illetve implementációtól függően elégséges ha
  - a 2. lépésben **A** csak az x_A-t küldi el **B**-nek,
  - a 3. lépésben **B** csak az x_B-t küldi el **A**-nak,
  - a E görbe egyenlete alapján meg lehet határozni a következő értékeket:

$$z_A = y_A^2, \; z_B = y_B^2$$

  - mivel a P256, illetve secp256k1 görbéknél használt p mindegyikére igaz, hogy $p+1$ néggyel való osztási maradéka nulla, azaz $p+1 \equiv 0 \pmod 4$, négyzetes maradékot lehet számolni:

$$y_A = z_A^{(p+1)/4} \pmod p, \; y_B = z_B^{(p+1)/4} \pmod p$$

- az y_K és $\hat{y_K}$ értékek lehet, hogy nem egyeznek meg, de ez nem jelent problémát, mert az x_K értékek mindig egyformák lesznek

ECDH - Elliptic Curve Diffie–Hellman key exchange

- Legyen $E : y^2 \equiv x^3 - 3x + 1 \pmod{11}$, $p = 11$, $P = (4, 8)$,
- a görbe pontjai:

$2 \cdot P = (7, 9)$	$3 \cdot P = (5, 10)$	$4 \cdot P = (6, 10)$	$5 \cdot P = (2, 5)$
$6 \cdot P = (10, 5)$	$7 \cdot P = (0, 1)$	$8 \cdot P = (8, 7)$	$9 \cdot P = (8, 4)$
$10 \cdot P = (0, 10)$	$11 \cdot P = (10, 6)$	$12 \cdot P = (2, 6)$	$13 \cdot P = (6, 1)$
$14 \cdot P = (5, 1)$	$15 \cdot P = (7, 2)$	$16 \cdot P = (4, 3)$	$17 \cdot P = \mathcal{O}$

- az **A** egység:
  - választja $a = 15$-t $\rightarrow A = a \cdot P = (7, 2)$,
  - elküldi **B**-nek az $x_A = 7$ értéket.
- a **B** egység:
  - választja $b = 6$-t $\rightarrow B = b \cdot P = (10, 5)$,
  - elküldi **A**-nak a $x_B = 10$ értéket.
- **A**:
  - $z_B = (10^3 - 3 \cdot 10 + 1) \pmod{11} = 3, y_B = 3^{(11+1)/4} = 5$
  - $aB = 15 \cdot (10, 5) = (2, 5)$,
- **B**:
  - $z_A = (7^3 - 3 \cdot 7 + 1) \pmod{11} = 4, y_A = 4^{(11+1)/4} = 9$
  - $bA = 6 \cdot (7, 9) = (2, 6)$,
- a közös kulcs: $K = 2$.

ECDSA - Elliptic Curve Digital Signature Algorithm

- a DSA változata, biztonsága nem bizonyított,
- a legszélesebb körben használt digitális aláírás,
- több standard specifikáció is létezik, amelyek nem mindegyike kompatibilis: ISO 14888-3, ANSI X9.62, NIST's FIPS 186-2, IEEE P1363, stb.
- `ANSI X9.62 ECDSA standard`
- a globális paraméterek:
  - a görbe egyenletének paraméterei,
  - a görbe egy alappontja, amely a görbének egy nagy elemszámú, ciklikus alcsoportját generálja: $P = (x_p, y_p)$,
  - a P pont rendje: n,
  - egy választott H hash függvény, pl. SHA-256.

EdDSA Edwards-curve Digital Signature Algorithm

- a Schnorr digitális aláírás változata
- Daniel J. Bernstein tervezte:
  - 2011-ben tette közzé, miután 2010-ban lejárt az oltalmi ideje a Schnorr aláírás szabadalmának,
  - nem találta elég biztonságosnak a NIST által javasolt görbéket.
- az alkalmazott elliptikus görbe: twisted Edwards curve
- az aláírás determinisztikus: a megfelelő biztonság így is megvalósítható
- az aktuális standard leírása az `RFC8032` dokumentumban található
- két különböző görbét lehet használni:
  - Edwards25519: a Daniel J. Bernstein Curve25529 változata, hatékonyabb a számítás
  - Edwards448: Mike Hamburg Ed448-Goldilocks görbéje
- nem egyetlen privát kulcsot generál az aláírás létrehozásához, hanem egy secret data (sk) alapján generál egy signing key (key_s)-t és egy nonce key (key_n)-t
- a kripto könyvtárcsomagok implementációi eltérnek: vannak amelyek az sk értékét, és vannak amelyek úgy a key_s-t, mint a key_n-t eltárolják.

EdDSA Edwards-curve Digital Signature Algorithm

a globális paraméterek ugyanazok, mint az ECDSA-nál: a görbe, P, n, H

Kulcs generálás:

- $key_s||key_n \leftarrow H(sk)$, ahol key_s, key_n, sk szigorúan titkosak,
- jelöljük a-val a key_s-t, ahol fenn kell álljon, hogy $a \in \{1, n-1\}$,
- $A = aP$,
- A: publikus kulcs, a: privát kulcs

Aláírás meghatározás: az $Aut((a, key_n), m)$ meghatározza az m üzenet (B, c) aláírását:

- legyen $b = H(key_n||m)$, a nonce érték, az üzenet értékétől függ
- $B = bP$, egy pont lesz a görbén,
- $h = H(B||A||m)$,
- $c = (b + a \cdot h) \pmod{n}$, egy egész szám lesz

Aláírás ellenőrzés: a $Ver((A), (m, B, c))$ bemenetre:

- legyen $h = H(B||A||m)$
- az aláírás helyes, ha fennáll:

$$B + hA == cP$$

Helyesség:

$$\begin{aligned} B + hA &= bP + haP \\ cP = (b + a \cdot h)P &= bP + ahP \end{aligned}$$

ECC - biztonság

- a standardként elfogadott görbék mindegyike esetében fennáll az ECDL feltételezés, de ez nem mindig elég az ECC biztonsághoz
- twist secure görbék:
  - minden elliptikus görbe esetében létezik egy *iker (twist)* görbe: az $E/\mathbb{F}_p : y^2 \equiv x^3 + ax + b$ görbének az $\hat{E}/\mathbb{F}_p : cy^2 \equiv x^3 + ax + b$ felel meg, ahol $c \in \mathbb{F}_p$ nem négyzetes maradék,
  - azért mert az ECDL feltételezés fennáll az $E/\mathbb{F}_p$-ben nem biztos hogy fennáll az $\hat{E}/\mathbb{F}_p$-ben is:
    - ha egy támadó **B**-nek elküldi a $\hat{P}$ pont $\hat{x}$ koordináta értékét, ahol $\hat{x} \in \mathbb{F}_p$ és $\hat{P} \in \hat{E}/\mathbb{F}_p$, és **B** visszaküldi az $\alpha\hat{P}$ értéket, és
    - ha az ECDL probléma könnyű a $\hat{E}/\mathbb{F}_p$-ben,
    - akkor a támadó meg tudja határozni α-t,
  - mindig le kell ellenőrizni, hogy a kézhez kapott pont EC-beli pont-e,
- a helytelen implementáció lehetővé tehet timing, side-channel típusú támadást, illetve eredményezhet nem biztonságos EC pontokat,
- habár elméletileg lehetséges gyakorlatban nagyon nehéz helyes ECC implementációt kivitelezni

ECC, Python - kulcs generálás

```
from Crypto.PublicKey import ECC
from Crypto.Signature import eddsa
from Crypto.Protocol.DH import key_agreement
from Crypto.Hash import SHAKE128, HMAC
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def keyECCWrite(fPub, fPriv, password):
    key = ECC.generate(curve='Ed25519')
    with open(fPub, 'wt') as f:
        f.write(key.public_key().export_key(format='PEM'))
    with open(fPriv, 'wt') as f:
         f.write(key.export_key(format='PEM',
                                passphrase=password,
                                protection='PBKDF2WithHMAC-SHA512AndAES256-CBC'))

password_CA = b'myPass10eload_CA'
keyECCWrite('ECC_keyPub_CA.pem', 'ECC_keyPriv_CA.pem', password_CA)

password_A = b'myPass10eload_peerA'
keyECCWrite('ECC_keyPub_peerA.pem', 'ECC_keyPriv_peerA.pem', password_A)

password_B = b'myPass10eload_peerB'
keyECCWrite('ECC_keyPub_peerB.pem', 'ECC_keyPriv_peerB.pem', password_B)
```

ECC, Python - kulcsok beolvasása

```
def privKeyRead(fPriv, password):
    with open(fPriv, 'rt') as f:
        temp = f.read()
    keyPriv = ECC.import_key(temp,
                             passphrase=password)
    return keyPriv

def pubKeyRead(fPub):
    with open(fPub, 'rt') as f:
        temp = f.read()
    keyPub = ECC.import_key(temp)
    return keyPub

password_CA = b'myPass10eload_CA'
privKey = privKeyRead('ECC_keyPriv_CA.pem', password_CA)
print(privKey.pointQ.x)
print(privKey.pointQ.y)
print(privKey.d)

pubKey = pubKeyRead('ECC_keyPub_CA.pem')
print(pubKey.pointQ.x)
print(pubKey.pointQ.y)
```

ECC, Python - digitális aláírás generálás/ellenőrzés

```
def signECC(mess, fPriv, password, fSign):
    privKey = privKeyRead(fPriv, password)
    signer = eddsa.new(privKey, 'rfc8032')
    signature = signer.sign(mess)
    with open(fSign, 'wt') as f:
        f.write(signature.hex())

def verifyECC(mess, fPub, fSign):
    pubKey = pubKeyRead(fPub)
    with open(fSign, 'rt') as f:
        temp = f.read()
    print('signature: ', fSign, temp)
    signature = bytes.fromhex(temp)
    verifier = eddsa.new(pubKey, 'rfc8032')
    try:
        verifier.verify(mess, signature)
        print('valid signature!')
    except:
        print('invalid signature!')

mess = b'message: EMTE Sapientia'
password_CA = b'myPass10eload_CA'
signECC(mess, 'ECC_keyPriv_CA.pem', password_CA, 'ECC_sign.txt')
verifyECC(mess, 'ECC_keyPub_CA.pem', 'ECC_sign.txt')
```

ECC, Python - publikus kulcsok aláírása

```
password_CA = b'myPass10eload_CA'

messA = pubKeyRead('ECC_keyPub_peerA.pem').export_key(format='raw')
signECC(messA, 'ECC_keyPriv_CA.pem', password_CA, 'signatureA.txt')
#verifyECC(messA, 'ECC_keyPub_CA.pem', 'signatureA.txt')

messB = pubKeyRead('ECC_keyPub_peerB.pem').export_key(format='raw')
signECC(messB, 'ECC_keyPriv_CA.pem', password_CA, 'signatureB.txt')
#verifyECC(messB, 'ECC_keyPub_CA.pem', 'signatureB.txt')
```

ECC, Python - nyílt szöveg titkosítása/visszafejtése

```
def encryptAES_GCM(plaintext, key, nonce):
    header = b'...'
    cipher = AES.new(key, AES.MODE_GCM, nonce)
    cipher.update(header)
    ciphertext, tag = cipher.encrypt_and_digest(plaintext)
    return tag, ciphertext

def decryptAES_GCM(ciphertext, key, tag, nonce):
    header = b'...'
    cipher = AES.new(key, AES.MODE_GCM, nonce)
    cipher.update(header)
    try:
        decryptedtext = cipher.decrypt_and_verify(ciphertext, tag)
        return decryptedtext
    except:
        print('crypted text auth failed!!')

def mykdf(x):
    return SHAKE128.new(x).read(32)
```

ECC, Python - a résztvevő felek műveletei

```
def peerA(plaintext, nonce, fPrivA, passwordA, fPubB, fPubCA, fSign):
    a = privKeyRead(fPrivA, passwordA)
    B = pubKeyRead(fPubB)
    mess = B.export_key(format='raw')
    print('pub key', fPubB, mess.hex())
    try:
        verifyECC(mess, fPubCA, fSign)
    except:
         print('pub key auth failed!')
         return
    K = key_agreement(static_priv=a, static_pub=B, kdf=mykdf)
    tag, cryptedtext = encryptAES_GCM(plaintext, K, nonce)
    return tag, cryptedtext

def peerB(cryptedtext, tag, nonce, fPrivB, passwordB, fPubA, fPubCA, fSign):
    b = privKeyRead(fPrivB, passwordB)
    A = pubKeyRead(fPubA)
    mess = A.export_key(format='raw')
    print('pub key', fPubA, mess.hex())
    try:
        verifyECC(mess, fPubCA, fSign)
    except:
       print('pub key auth failed!')
       return
    K = key_agreement(static_priv=b, static_pub=A, kdf=mykdf)
    decryptedtext = decryptAES_GCM(cryptedtext, K, tag, nonce)
    return decryptedtext
```

ECC, Python - hitelesítés/kulcscsere/titkosítás/visszafejtés

```
password_A = b'myPass10eload_peerA'
password_B = b'myPass10eload_peerB'

nonce = get_random_bytes(12)
plaintext = b'hello EMTE!'
tag, cText = peerA(plaintext, nonce,
                   'ECC_keyPriv_PeerA.pem', password_A,
                   'ECC_keyPub_PeerB.pem',
                   'ECC_keyPub_CA.pem',
                   'signatureB.txt')
print(cText.hex())

dText = peerB(cText, tag, nonce,
              'ECC_keyPriv_PeerB.pem', password_B,
              'ECC_keyPub_PeerA.pem',
              'ECC_keyPub_CA.pem',
              'signatureA.txt')
print(dText.decode())
```