Diszkrét matematika
13. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

Miről volt szó az elmúlt előadáson?

- a baby RSA rendszer: titkosító, digitális aláírás rendszer
- az RSA és a kínai maradéktétel
- összetett számok faktorizációja
  - a Fermat féle faktorizáció
  - a Pollard ρ féle faktorizáció

Miről lesz szó?

- Másodfokú kongruenciák, kvadratikus maradékok
- A Rabin digitális aláírás
- Elliptikus görbe kripto, Python függvények

Másodfokú kongruenciák, kvadratikus maradékok

Határozzuk meg $\pmod{11}$ szerint a számok négyzetét:

$$\begin{aligned} 1^2 &\equiv 1 & 6^2 &\equiv 3 \\ 2^2 &\equiv 4 & 7^2 &\equiv 5 \\ 3^2 &\equiv 9 & 8^2 &\equiv 9 \\ 4^2 &\equiv 5 & 9^2 &\equiv 4 \\ 5^2 &\equiv 3 & 10^2 &\equiv 1 \end{aligned}$$

Vegyük észre, hogy az alábbi kongruenciák megoldhatóak és minden esetben két megoldás van:

$$\begin{aligned} x^2 &\equiv 1 \pmod{11}, && \text{megoldások: } 1, 10 \\ x^2 &\equiv 4 \pmod{11}, && \text{megoldások: } 2, 9 \\ x^2 &\equiv 3 \pmod{11}, && \text{megoldások: } 5, 6 \\ x^2 &\equiv 5 \pmod{11}, && \text{megoldások: } 4, 7 \\ x^2 &\equiv 9 \pmod{11}, && \text{megoldások: } 3, 8 \end{aligned}$$

Az alábbi kongruenciák NEM oldhatóak meg:

$$\begin{aligned} x^2 &\equiv 2 \pmod{11} \\ x^2 &\equiv 6 \pmod{11} \\ x^2 &\equiv 7 \pmod{11} \\ x^2 &\equiv 8 \pmod{11} \\ x^2 &\equiv 10 \pmod{11} \end{aligned}$$

Másodfokú kongruenciák, kvadratikus maradékok

1. értelmezés

Az a számot kvadratikus maradéknak (négyzetes maradék) nevezzük, ha létezik olyan x amelyre az $x^2 \equiv a \pmod{n}$ kongruencia megoldható, ahol $\mathrm{lnko}(a, n) = 1$. Ebben az esetben x-et az a négyzetgyökének hívjuk. Az a számot, ha nem kvadratikus maradék, akkor kvadratikus nemmaradéknak hívjuk.

Az $1, 4, 3, 5, 9$ számok kvadratikus maradékok $\pmod{11}$ szerint, mert:

- 1 négyzetgyöke: $1, 10$,
- 4 négyzetgyöke: $2, 9$,
- 3 négyzetgyöke: $5, 6$,
- 5 négyzetgyöke: $4, 7$,
- 9 négyzetgyöke: $3, 8$.

A $2, 6, 7, 8, 10$, számok, pedig kvadratikus nemmaradékok.

Tehát 5 szám kvadratikus maradék, és szintén 5 szám kvadratikus nemmaradék $\pmod{11}$ szerint.

Másodfokú kongruenciák, kvadratikus maradékok

A (mod P) szerinti kongruenciákra kijelenthető, ahol P prímszám:

- ha egy a szám kvadratikus maradék, akkor az $x^2 \equiv a \pmod{P}$ kongruenciának két inkongruens megoldása van
- (mod P) szerint ugyanannyi szám kvadratikus maradék, mint amennyi kvadratikus nemmaradék, számuk: $\frac{P-1}{2}$.

Észrevehető a következő tulajdonság is:

$$
\begin{array}{lll}
1^5 \equiv 1 & 2^5 \equiv 10 \equiv (-1) & \pmod{11} \\
4^5 \equiv 1 & 6^5 \equiv 10 \equiv (-1) & \pmod{11} \\
3^5 \equiv 1 & 7^5 \equiv 10 \equiv (-1) & \pmod{11} \\
5^5 \equiv 1 & 8^5 \equiv 10 \equiv (-1) & \pmod{11} \\
9^5 \equiv 1 & 10^5 \equiv 10 \equiv (-1) & \pmod{11}
\end{array}
$$

Tehát:

- egy a szám akkor és csakis akkor kvadratikus maradék (mod P) szerint ha: $a^{(P-1)/2} \equiv 1 \pmod{P}$
- egy a szám akkor és csakis akkor kvadratikus nemmaradék (mod P) szerint ha: $a^{(P-1)/2} \equiv -1 \pmod{P}$

Másodfokú kongruenciák, kvadratikus maradékok

Határozzuk meg $\pmod{35 = 5 \cdot 7}$ szerint azon x számok négyzetét, amelyre fennáll $gcd(x, 35) = 1$:

$$\begin{array}{llll}
1^2 \equiv 1 & 9^2 \equiv 11 & 18^2 \equiv 9 & 27^2 \equiv 29 \\
2^2 \equiv 4 & 11^2 \equiv 16 & 19^2 \equiv 11 & 29^2 \equiv 1 \\
3^2 \equiv 9 & 12^2 \equiv 4 & 22^2 \equiv 29 & 31^2 \equiv 16 \\
4^2 \equiv 16 & 13^2 \equiv 29 & 23^2 \equiv 4 & 32^2 \equiv 9 \\
6^2 \equiv 1 & 16^2 \equiv 11 & 24^2 \equiv 16 & 33^2 \equiv 4 \\
8^2 \equiv 29 & 17^2 \equiv 9 & 26^2 \equiv 11 & 34^2 \equiv 1
\end{array}$$

Vegyük észre, hogy csak az alábbi kongruenciák oldhatóak meg és minden esetben négy megoldás van:

$$\begin{array}{ll}
x^2 \equiv 1 \pmod{35}, & \text{megoldások: } 1, 6, 29, 34 \\
x^2 \equiv 4 \pmod{35}, & \text{megoldások: } 2, 12, 23, 33 \\
x^2 \equiv 9 \pmod{35}, & \text{megoldások: } 3, 17, 18, 32 \\
x^2 \equiv 11 \pmod{35}, & \text{megoldások: } 4, 11, 24, 31 \\
x^2 \equiv 16 \pmod{35}, & \text{megoldások: } 8, 13, 22, 27 \\
x^2 \equiv 29 \pmod{35}, & \text{megoldások: } 9, 16, 19, 26
\end{array}$$

Másodfokú kongruenciák, kvadratikus maradékok

Írjunk egy Python kódot, amely meghatározza (mod n) szerint a kvadratikus maradékokat és a négyzetgyököket is.

```
import math
p, q = 7, 5
n = p * q
NegyzetL = set()
for x in range(n):
    if math.gcd(x, n) == 1:
        NegyzetL.add((x * x) % n)
negyzetGyD = {y : [] for y in NegyzetL}
print(NegyzetL)
for x in range(n):
        if math.gcd(x, n) == 1:
            negyzetGyD[(x * x) % n].append(x)
print(len(NegyzetL), (p-1) * (q-1)//4, n)
print(negyzetGyD)
```

Egy szám kvadratikus maradék vagy sem?

A $\pmod N$ szerinti kongruenciákra, ahol $N = P \cdot Q$ és P, Q különböző prímszámok kijelenthető:

- egy a szám akkor és csakis akkor kvadratikus maradék $\pmod N$ szerint ha a kvadratikus maradék $\pmod P$ szerint és kvadratikus maradék $\pmod Q$ szerint is,
- ha egy a szám kvadratikus maradék, akkor az $x^2 \equiv a \pmod N$ kongruenciának négy inkongruens megoldása van,
- $\pmod N$ szerint a kvadratikus maradékok száma: $\frac{(P-1)\cdot(Q-1)}{4}$.

A négyzetgyök meghatározása

- ha a modulus egy P prímszám, és $P \equiv 3 \pmod 4$, akkor az $x^2 \equiv a \pmod P$ kongruencia megoldása (2 megoldás lesz), azaz a négyzetgyökök meghatározása a következőképpen történik:

$$x \equiv \pm a^{(P+1)/4} \pmod P$$

- az egyik négyzetgyök páros lesz a másik páratlan
- Példa:
  - oldjuk meg az $x^2 = 5 \pmod{11}$ kongruenciát.
  - meghatározzuk $5^{(11+1)/4} \equiv 5^3 \equiv 4 \pmod{11}$
  - a kongruencia megoldása: $\pm 4 \pmod{11}$, azaz $4, 7$
  - ellenőrzés: $4^2 \equiv 5 \pmod{11}$ és $7^2 \equiv 5 \pmod{11}$.

A négyzetgyök meghatározása

Ha a modulus egy összetett N szám és tudjuk, hogy $N = P \cdot Q$ és $P \equiv Q \equiv 3 \pmod{4}$, akkor az $x^2 \equiv a \pmod{N}$, kongruencia megoldása (4 megoldás) a következőképpen történik:

- a 4 megoldás közül 2 páros és 2 páratlan lesz,
- először meghatározzuk a következő értékeket:

$$
\begin{aligned}
x_P &= a^{(P+1)/4} \pmod{P}, \\
x_Q &= a^{(Q+1)/4} \pmod{Q},
\end{aligned}
$$

- majd alkalmazzuk a Kínai maradéktételt, ahol $x_1, -x_1, x_2, -x_2$ lesz a 4 lehetséges megoldás, ahol:

$$
\begin{aligned}
x_1 &= (P^{-1} \cdot P \cdot x_Q + Q^{-1} \cdot Q \cdot x_P) \pmod{N}, \\
x_2 &= (P^{-1} \cdot P \cdot x_Q - Q^{-1} \cdot Q \cdot x_P) \pmod{N}
\end{aligned}
$$

- P^{-1} érték P inverze $\pmod{Q}$ szerint
- Q^{-1} érték Q inverze $\pmod{P}$ szerint

A négyzetgyök meghatározása

Példa:

- legyen $N = P \cdot Q = 2773$, ahol $P = 47$ és $Q = 59$
- oldjuk meg a $x^2 = 17 \pmod{2773}$ kongruenciát
- meghatározzuk P^{-1}, Q^{-1} értékeket:

$$\begin{array}{ll} P^{-1} \equiv 54 \pmod{59}, & \text{mert fennáll:} \quad 47 \cdot 54 \equiv 1 \pmod{59} \\ Q^{-1} \equiv 4 \pmod{47}, & \text{mert fennáll:} \quad 59 \cdot 4 \equiv 1 \pmod{47} \end{array}$$

- meghatározzuk:

$$\begin{aligned} x_P &= 17^{(P+1)/4} = 17^{12} \equiv 8 \pmod{47} \\ x_Q &= 17^{(Q+1)/4} = 17^{15} \equiv 28 \pmod{59} \\ x_1 &= 54 \cdot 47 \cdot 28 + 4 \cdot 59 \cdot 8 \pmod{2773} = 854 \\ x_2 &= 54 \cdot 47 \cdot 28 - 4 \cdot 59 \cdot 8 \pmod{2773} = 2624 \end{aligned}$$

- a négy megoldás:

$$\begin{array}{rlll} 854, & \text{ellenőrzés:} & 854^2 & \equiv 17 \pmod{2773} \\ -854 \equiv 1919 \pmod{2773}, & \text{ellenőrzés:} & 1919^2 & \equiv 17 \pmod{2773} \\ 2624, & \text{ellenőrzés:} & 2624^2 & \equiv 17 \pmod{2773} \\ -2624 \equiv 149 \pmod{2773} & \text{ellenőrzés:} & 149^2 & \equiv 17 \pmod{2773} \end{array}$$

A Rabin digitális aláírás

- 1979-ben publikálta Michael O. Rabin, az első probabilisztikus digitális aláírási séma,
- a titkosító változat az oktatásban jelenik meg, de csak később, ezzel Rabin nem is foglalkozott,
- a titkosító változat esetében a visszafejtés nem egyértelmű,
- 2001-ben jelenik meg az SAEP,
- a gyakorlatban se a digitális aláírást, se a titkosító változatot nem használják,
- az alkalmazott függvény a moduláris négyzetreemelés,
- bizonyítható, hogy biztonsága ekvivalens a faktorizációs feltételezéssel: a rendszer feltörhetősége (aláírás hamisítás) ugyanolyan nehéz, mint megoldani a faktorizációs problémát,
- A kvadratikus maradék feltételezés azt jelenti, hogy nincs hatékony algoritmus, amely egy n összetett egész szám esetében megállapítaná egy x számról, hogy az négyzetes maradék vagy sem. Egy x szám négyzetes maradék, ha létezik olyan y szám, amelyre fennáll:

$$x \equiv y^2 \pmod{n}.$$

- egy biztonságos hash függvény is alkalmazásra kerül: a bemenetet úgy alakítja át, hogy a kimenet alapján nem lehet meghatározni a bemeneti értéket.

A Rabin digitális aláírás

- a Rabin által alkalmazott függvény a következő :

$$F_{n,b}(x) \equiv x(x+b) \pmod{n},$$

ahol $n = p \cdot q$, p, q prímszámok, és b rögzített egész szám: $0 \leq b < n$,

- a legegyszerűbb eset ha $b = 0$: $F_n(x) = x^2 \pmod{n}$.
- az inverz függvény megadása nem olyan egyszerű, mint az RSA esetében, mert $(2, \phi) \neq 1$ (ϕ mindig páros), tehát 2-nek nem létezik multiplikatív inverze,
- az aláírás előállításához szükség van a p és q értékekre, másképp nem lehet meghatározni az x értékét.

a kulcsgeneráló algoritmus:

- bemenete egy k biztonsági paraméter, amely a generált kulcsméretet jelenti,
- véletlenszerűen generál két k bites prímszámot, legyenek ezek p és q, úgy hogy: $p \equiv q \equiv 3 \pmod{4}$, és meghatározza az $n = p \cdot q$-t,
- legyen b véletlenszerűen meghatározott egész szám: $1 < b < n$,
- publikus kulcs: $p_k = (n, b)$,
- privát kulcs: $s_k = (p, q)$

A Rabin digitális aláírás

az aláíró algoritmus: meghatározza az (U, x) aláírását a K-nak:

- bemeneti paramétere a privát kulcs, és a K egész szám, ahol $1 < K < n$,
- nem az a szerepe, hogy titkosítsa a K értékét,
- legyen H a hash függvény,
- legyen U egy véletlenszerűen meghatározott egész szám, és:

$$h = H(K||U) \pmod{n}$$

- meghatározzuk x-et, úgy hogy: $x^2 \equiv h \pmod{n}$,
- ha nem sikerül ilyen x értéket találni, akkor más U érték kerül kiválasztásra, a várható próbálgatások száma 4.

az ellenőrző algoritmus:

- meghatározza $y_1 \equiv x^2 \pmod{n}$, és $y_2 \equiv H(K||U) \pmod{n}$
- ha $y1$ egyenlő y_2-vel, akkor az aláírás hiteles.

A Rabin digitális aláírás

Az aláírás előállításakor olyan x-et kell meghatározni, amelyre fennáll, hogy

$$x^2 \equiv h \pmod{n},$$

azaz $h = H(K||U)$ kvadratikus maradék lesz. Ehhez a következőket kell meghatározni:

$$\begin{aligned}
&K_p \equiv h^{(p+1)/4} \pmod{p}\\
&K_q \equiv h^{(q+1)/4} \pmod{q}\\
&p^{-1} : p \cdot p^{-1} \equiv 1 \pmod{q}\\
&q^{-1} : q \cdot q^{-1} \equiv 1 \pmod{p}\\
&x_1 \equiv (p^{-1} \cdot p \cdot K_q + q^{-1} \cdot q \cdot K_p) \pmod{n}\\
&x_2 \equiv (p^{-1} \cdot p \cdot K_q - q^{-1} \cdot q \cdot K_p) \pmod{n}
\end{aligned}$$

- A következő 4 érték közül bármelyik betöltheti az x szerepét

$$x_1, x_2, x_3 = n - x_1, x_4 = n - x_2$$

- az aláírás: U, x lesz,

Az aláírás ellenőrzését a következőképpen végezzük:

- meghatározzuk $y_1 \equiv x^2 \pmod{n}$, és $y_2 \equiv H(K||U) \pmod{n}$
- ha $y1$ egyenlő y_2-vel, akkor az aláírás hiteles.

A Rabin digitális aláírás, példa

- legyen $p = 11$, és $q = 31 \Rightarrow n = 341$,
- legyen $K = 42$ az érték, amit alá szeretnénk írni és legyen $U = 285$
- az aláírás előállítása:
  - tegyük fel, hogy $h = H(K||U) = 169$, ekkor meghatározzuk:

$$169^{(11+1)/4} = 9 \pmod{11}$$
$$169^{(31+1)/4} = 18 \pmod{31}$$
$$31^{-1} = 5 \pmod{11}$$
$$11^{-1} = 17 \pmod{31}$$

  - a következő négy érték közül kiválasztunk egyet véletlenszerűen:

$$x_1 = (31 \cdot 31^{-1} \cdot 9 + 11 \cdot 11^{-1} \cdot 18) \equiv 328 \pmod{341},$$
$$x_2 = (31 \cdot 31^{-1} \cdot 9 - 11 \cdot 11^{-1} \cdot 18) \equiv 75 \pmod{341},$$
$$x_3 = 341 - x_1 = 13,$$
$$x_4 = 341 - x_2 = 266$$

  - az aláírás: $(285, 75)$
- az aláírás ellenőrzése:

$$x^2 = 75^2 \equiv 169 \pmod{341}$$
$$h = 169 \pmod{341}$$

Rabin, kulcsgenerálás

A hash függvényhez: PyCryptodome

```
from sympy import isprime
from random import getrandbits, randint, choice
from sympy.ntheory import is_quad_residue
from Crypto.Hash import SHA512

def rabinKeyGen(k):
    p = blumPrime(k)
    q = blumPrime(k)
    n = p * q
    return n, p, q

def blumPrime(k):
    while True:
        p = getrandbits(k)
        if p % 4 == 3 and isprime(p): break
    return p
```

Rabin, aláíráselőállítás

```
def rabinSign(bajtM, n, p, q):
    while True:
        U = randint(2, n)
        bitLen = U.bit_length()
        bajtLen = bitLen // 8 + 1
        bajtU = U.to_bytes(bajtLen, byteorder='big')
        bajtH = SHA512.new(bajtM + bajtU)
        h = int.from_bytes(bajtH.digest(), byteorder='big') % n
        if is_quad_residue(h, q) and is_quad_residue(h, p):
            p1 = pow(p % q, -1, q)
            q1 = pow(q % p, -1, p)
            xp = pow(h, (p + 1)//4, p)
            xq = pow(h, (q + 1)//4, q)
            x1 = (xp * q * q1 + xq * p * p1) % n
            x2 = (xp * q * q1 - xq * p * p1) % n
            x3 = n - x1
            x4 = n - x2
            x = choice([x1, x2, x3, x4])
            return U, x
```

Rabin, aláírásellenőrzés, meghívások

```
def rabinVerify(bajtM, U, x, n):
    bitLen = U.bit_length()
    bajtLen = bitLen // 8 + 1
    bajtU = U.to_bytes(bajtLen, byteorder='big')
    bajtH = SHA512.new(bajtM + bajtU)
    h = int.from_bytes(bajtH.digest(), byteorder='big') % n
    x = pow(x, 2, n)
    if h == x: print('OK')
    else: print('NOT OK')

n, p, q = rabinKeyGen(2048)
n = p * q
bajtM = b'Áldott Ünnepeket!'
U, x = rabinSign(bajtM, n, p, q)
print('x: ', x)
print('U: ', U)
rabinVerify(bajtM, U, x, n)
```

Véletlenszám generátorok

- véletlen, illetve álvéletlen módon generált számokra számos algoritmusban szükség van,
- véletlen módon generált számok (true random numbers): hardver eszközökkel,
- álvéletlen módon generált számok (pseudo random numbers): szoftver eszközökkel,
- a kriptográfiában komolyabb biztonsági kritériumnak eleget tevő álvéletlen számokat generáló algoritmusokat is használnak, ezek *erős* algoritmusok, ezek kulcs szerepet játszanak a rendszerek biztonságában,
- a gyakorlatban a Diffi-Hellman, az RSA, a Rabin rendszerek által kezelt üzeneteket, titkos információkat kiegészítik pszeudo random módon genereált bittekkel, ezek során *erős* algoritmusokat használnak,
- a publikus, a privát kulcsok generálása során nem szükséges *erős* algoritmusokat használni,
- álvéletlen számokat generáló algoritmusok:
  - a lineáris kongruencián alapuló generátor
  - a közép-négyzet módszer (middle-square)
  - a Blum-Blum-Shub generátor: *erős* álvéletlen számokat generál
- minden programozási nyelv rendelkezik álvéletlenszámokat generáló algoritmusokkal, amelyek nem alkalmasak kriptográfiai rendszerekben.

A Blum-Blum-Shub generátor

- alkalmas kriptográfiai rendszerekben, biztonságát a faktorizációs és kvadratikus maradék problémák nehézsége adja
- nagy számításigényű, ezért csak nagy biztonságot követelő rendszereknél alkalmazzák
- a moduláris négyzetreemelés az alkalmazott függvény, ahol az algoritmus a $(b_1, b_2, \dots b_n)$ bitsorozatot a következőképpen hozza létre:

$$b_i = u_i \pmod 2 \text{ és } u_i = u_{i-1}^2 \pmod N \text{, ahol}$$

  - u_0 a generátor egy kezdeti értéke, amelyet a következőképpen választunk ki: $u_0 = r^2 \pmod N$, ahol $r \xleftarrow{R} \{2, 3, \dots, N-1\}$ és $lnko(r, N) = 1$
  - $N = P \cdot Q$ és P, Q prímek, úgy hogy $P = 2 \cdot p + 1$, $Q = 2 \cdot q + 1$ és p, q is prímek
- a generált P, Q értékek tehát biztonságos prímek, amelyeket Blum egészeknek is hívnak, fennáll: $P \equiv Q \equiv 3 \pmod 4$.

A Blum-Blum-Shub generátor

Példa

- legyen $P = 23, Q = 59$ két biztonságos prím, $N = 1357$
- legyen $u_0 = 9$
- $u_i = u_{i-1}^2 \pmod N, b_i = u_i \% 2$

i	u_i	b_i
1	81	1
2	1133	1
3	1324	0
4	1089	1
5	1260	0
6	1267	1
7	1315	1
8	407	1

A generált bitsorozat: $1, 1, 0, 1, 0, 1, 1, 1$

A Blum-Blum-Shub generátor

1. feladat

Generáljunk l darab bitet a Blum-Blum-Shub generátorral, ahol az alkalmazott N legyen egy k bites szám.

```
from random import getrandbits, randint
def bbs(k, l):
    P = safePrime(k//2)
    Q = safePrime(k//2)
    N = P * Q
    r = randint(2, N - 1)
    u = (r * r) % N
    for i in range(0, l):
        u = (u * u) % N
        #print(u & 255, end = ' ')
        print(u & 1, end=' ')
```

```
def safePrime(k):
    while True:
        p = getrandbits(k)
        if not (p & 1): p += 1
        q = (p - 1) //2
        if isprime(q) and isprime(p): return p
```

```
>>> bbs(64, 16)
    0 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1
>>> bbs(512, 10)
    87 121 30 143 164 151 229 170 57 18
```

Elliptikus görbék

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

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

- ha az egyik pont $\mathcal{O}$, akkor fennáll: $P + \mathcal{O} = \mathcal{O} + P = P$

Műveletek Weierstrass típusú görbéken

A $P + P = 2P$ pont értékét meghatározó Python kód, a $y^2 \equiv x^3 + ax + b \pmod{p}$ görbe esetében:

```
def doublePoint(P, p, a):
    if P == (None, None): return P
    (x1, y1) = P
    if y1 == 0:
        return (None, None)
    lamb = ((3 * x1 * x1 + a) * pow(2 * y1, -1, p)) % p
    x3 = (lamb * lamb - 2 * x1) % p
    y3 = (lamb * (x1 - x3) - y1) % p
    return (x3, y3)
```

Műveletek Weierstrass típusú görbéken

A $P + Q$ pontok összegét meghatározó Python kód, a $y^2 \equiv x^3 + ax + b \pmod{p}$ görbe esetében:

```
def addTwoPoint(P, Q, p):
    if Q == (None, None):
        return P
    if P == (None, None):
        return Q
    (x1, y1) = P
    (x2, y2) = Q
    lamb = ((y2 - y1) * pow(x2 + p - x1, -1, p)) % p
    x3 = (lamb * lamb - x1 - x2) % p
    y3 = (lamb * (x1 - x3) - y1) % p
    return (x3, y3)
```

Műveletek Weierstrass típusú görbéken

Az αP értéket meghatározó Python kód a gyorshatványozáshoz hasonló módon jár el, csak a szorzás helyett az összeadás művelete kerül alkalmazásra:

```
def multPointNotConstTime(alpha, P, p, a):
    X = (None, None)
    while alpha > 0:
        if alpha & 1 == 1:
            X = addTwoPoint(X, P, p)
        P = doublePoint(P, p, a)
        alpha = alpha >> 1
    return X
```

Műveletek Weierstrass típusú görbéken

Az időzítés támadás (timing attack) elkerülése érdekében az $alpha \cdot P$ értékét Montgomery ladder technikával szokták meghatározni, ennek a Python kódja a következő:

```
def multPoint(alpha, P, p, a):
    X = doublePoint(P, p, a)
    binAlpha = bin(alpha)[2:]
    for bit in binAlpha[1:]:
        if bit == '0':
            X = addTwoPoint(X, P, p)
            P = doublePoint(P, p, a)
        else:
            P = addTwoPoint(P, X, p)
            X = doublePoint(X, p, a)
    return P
```

Műveletek Weierstrass típusú görbéken

A Diffie-Hellman típusú kulcscsere:

```
def weierstrassDH_noFlag(P, a, b, p, n):
    #A:
    privKeyA = random.randrange(2, n)
    pubKeyA = multPoint(privKeyA, P, p, a)

    #B:
    privKeyB = random.randrange(2, n)
    pubKeyB = multPoint(privKeyB, P, p, a)

    #A:
    xK, yK = multPoint(privKeyA, pubKeyB, p, a)
    print('A calc K: ', xK, yK)

    #B:
    xK, yK = multPoint(privKeyB, pubKeyA, p, a)
    print('B calc K: ', xK, yK)
```

A Diffie-Hellman típusú kulcscsere:

```
def eccMain():
    # p = 11
    # P, n = (4, 8), 17
    # a, b = -3, 1

    p = 3623
    P, n = (6, 730), 947
    a = 14
    b = 19

    # p = 2 ** 256 - 2 ** 224 + 2 ** 192 + 2 ** 96 - 1
    # x = 48439561293906451759052585252797914202762949526041747995844080717082404635286
    # y = 36134250956749795798585127919587881956611106672985015071877198253568414405109
    # P = (x, y)
    # a = -3
    # b = 41058363725152142129326129780047268409114441015993725554835256314039467401291
    # n = 115792089210356248762697446949407573529996955224135760342422259061068512044369
    #weierstrassDH_noFlag(P, a, b, p, n)
    weierstrassDH(P, a, b, p, n)
```

Műveletek Weierstrass típusú görbéken

```
def setParatlan(x, p):
    if x & 1 == 1: return x
    else: return p - x

def setParos(x, p):
    if x & 1 == 0: return x
    else: return p - x

def calcKey(privK, x, flagY, P, a, b, p, n):
    z = (x ** 3 + a * x + b) % p
    y = pow(z, (p+1) // 4, p)
    if flagY == 0: y = setParos(y, p)
    else: y = setParatlan(y, p)
    print('y:', y)
    xK, yK = multPoint(privK, (x, y), p, a)
    print('calc K: ', xK, yK, '\')
    return xK, yK
```

Műveletek Weierstrass típusú görbéken

```
def sendPub(privK, P, a, b, p, n):
    pubK = multPoint(privK, P, p, a)
    (x, y) = pubK
    if y & 1 == 0: flagY = 0
    else: flagY = 1
    print('Pub key: ', x, y)
    print('flag: ', flagY, '\')
    return x, flagY

def weierstrassDH(P, a, b, p, n):
    #A
    privKeyA = random.randrange(2, n)
    xA, flagA = sendPub(privKeyA, P, a, b, p, n)

    #B
    privKeyB = random.randrange(2, n)
    xB, flagB = sendPub(privKeyB, P, a, b, p, n)

    #A
    calcKey(privKeyA, xB, flagB, P, a, b, p, n)
    #B
    calcKey(privKeyB, xA, flagA , P, a, b, p, n)
```