Diszkrét matematika
10. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév

Miről volt szó az elmúlt előadáson?

- kongruenciák
- moduláris hatványozás
- maradékosztályok, maradékrendszerek
- a kis Fermat tétel
- az Euler függvény, az Euler tétel
- az Euler függvényhez kapcsolódó összefüggések
- a Miller-Rabin prímteszt
- hatványok és generátor elemek

Miről lesz szó?

- lineáris kongruenciák,
- multiplikatív inverz,
- az RSA, illetve a *baby*-RSA rendszer,
- a kínai maradéktétel,
- az RSA és a kínai maradéktétel

Lineáris kongruenciák

Az $a \cdot x \equiv b \pmod{m}$ típusú egyenletet lineáris kongruenciának hívjuk, ahol x ismeretlen.

- ha x_0 megoldása a fenti kongruenciának és $x_1 \equiv x_0 \pmod{m}$, akkor x_1 is megoldás

1. tétel

Legyenek a, b, m egész számok, ahol $m > 0$ és $(a, m) = d$. Ha $d \nmid b$, akkor az $a \cdot x \equiv b \pmod{m}$ kongruenciának nincs megoldása. Ha $d \mid b$, akkor az $a \cdot x \equiv b \pmod{m}$ kongruenciának d inkongruens megoldása van szerint.

- Az $a \cdot x \equiv b \pmod{m}$ kongruencia ekvivalens az $a \cdot x - m \cdot y = b$ egyenlettel.
- kiterjesztett euklideszi algoritmussal meghatározzuk a $\hat{x}, \hat{y}$-t, úgy hogy teljesüljön: $a \cdot \hat{x} + m \cdot \hat{y} = d$.
- A kongruencia első megoldása $x_0 = \hat{x} \cdot (b/d)$ lesz.
- A kongruencia többi megoldásait az $n = 1, 2, ..., d - 1$ lehetséges értékek alapján, a következőképpen számoljuk ki: $x_i = x_0 + n \cdot (m/d)$.

Példák

Határozzuk meg a $9 \cdot x \equiv 12 \pmod{15}$ kongruencia megoldásait.

- $(9, 15) = 3$ és $3 \mid 12 \Rightarrow$ a kongruenciának 3 inkongruens megoldása van (mod 15) szerint.
- Kiterjesztett euklideszi algoritmussal kapjuk: $9 \cdot 2 + 15 \cdot (-1) = 3$, azaz $\hat{x} = 2$, $\hat{y} = -1$.
- A kongruencia első megoldása: $x_0 = 2 \cdot (12/3) = 8$.
- A többi megoldás:
  - $x_1 = x_0 + 1 \cdot (15/3) = 8 + 5 = 13 \pmod{15}$,
  - $x_2 = x_0 + 2 \cdot (15/3) = 8 + 10 = 18 \equiv 3 \pmod{15}$.
- fennáll:
  - $9 \cdot 8 = 72 = 12 \pmod{15}$,
  - $9 \cdot 13 = 117 = 12 \pmod{15}$,
  - $9 \cdot 3 = 27 = 12 \pmod{15}$,

Multiplikatív inverz

1. értelmezés

Az $a \cdot x \equiv 1 \pmod{m}$ kongruenciának az x megoldását, ahol $(a, m) = 1$ az a szám moduláris multiplikatív inverzének, vagy multiplikatív inverzének, vagy csak egyszerűen inverzének hívjuk.

Megjegyzés

- a fenti egyenlet egy sajátos esete az $a \cdot x \equiv b \pmod{m}$ egyenletnek: $b = 1$.

Példák:

- Mennyi lesz 4 inverze $\pmod 7$ szerint?, másképp fogalmazva: Mivel kell megszorozni 4-et, hogy 1-et kapjunk?
  - Válasz: 2-vel, mert $4 \cdot 2 = 1 \pmod 7$.
- Mennyi lesz 7 inverze $\pmod{31}$ szerint?, másképp fogalmazva: Mivel kell megszorozni 7-et, hogy 1-et kapjunk?
  - Válasz: 9-cel, mert $7 \cdot 9 = 1 \pmod{31}$.

Multiplikatív inverz

Egy szám multiplikatív inverzét többféleképpen is meghatározhatjuk:

- brute-force módszerrel: az összes lehetséges értéket sorra próbáljuk
- a kiterjesztett euklideszi algoritmussal: a gyakorlatban ezt használják
- az Euler tétel segítségével, ha $(a, m) = 1$ az $a \cdot x \equiv 1 \pmod{m}$ kongruenciának a megoldása a következőképpen adható meg:
  - $x = a^{\phi(m)-1} \pmod{m}$
  - ha m prímszám, akkor $x = a^{m-2} \pmod{m}$

Határozzuk meg a $13 \cdot x \equiv 4 \pmod{36}$ kongruencia egy megoldását, az Euler-tétel segítségével:

- $(13, 36) = 1 \Rightarrow$ a kongruenciának egy megoldása van, amely meghatározható az Euler tétel segítségével is,
- meghatározzuk:
  $\phi(36) = \phi(2^2 \cdot 3^2) = \phi(2^2) \cdot \phi(3^2) = (2^2 - 2^1) \cdot (3^2 - 3^1) = 2 \cdot 6 = 12$
- 13 inverze $\pmod{36}$ szerint: $13^{\phi(36)-1} = 13^{11} = 25 \pmod{36}$
- a kongruencia megoldása: $25 \cdot 4 = 28 \pmod{36}$.

Multiplikatív inverz

1. feladat

A kiterjesztett euklideszi algoritmussal határozzuk meg a inverzét (mod m) szerint.

```
def inverz(a, m):
    x0, x1 = 1, 0
    b = m
    while True:
        q = a // b
        r = a - b * q
        if r == 0:
            if b != 1: return -1
            else: return x1 % m
        x = x0 - q * x1
        x0, x1 = x1, x
        a, b = b, r

>>> inverz(13, 36)
    25
```

ellenőrzés:

```
>>> (13 * 25) % 36
    1
```

Multiplikatív inverz

Megjegyzések:

- az `inverz` függvényben egy kicsit másképp jár el, mint a kiterjesztett euklideszi algoritmus:
  - az *y* változóval végzett műveleteket elhagytuk, mert azok most fölöslegesek,
  - módosult a `return`-hoz tartozó műveletsor: csak egy értéket ad vissza a függvény, a meghatározott inverzet
- a kiterjesztett euklideszi algoritmus az `x` és `y` értékekben negatív számot is meghatározhat,
- azt szeretnénk hogy az inverz mindig pozitív érték legyen, ezért a `return`-ben a `x1 % m` műveletet alkalmaztuk,
- a Python % operátora mindig a legkisebb pozitív maradékot határozza meg:

```
>>> -5 % 7
    2
```

- az inverz meghatározása Python-ban, `-1`-es kitevőt kell használnunk:

```
>>> pow(13, -1, 36)
    25
```

Az RSA rendszer

- 1977-ban publikálták a szerzők: Rivest, Shamir és Adleman, biztonsága többek között a faktorizációs problémán alapszik,

2. értelmezés

Az egész számok $\{1, 2, \ldots, n\}$ véges halmaza esetében, ahol n összetett szám, pontosabban két prímszám szorzata a faktorizációs probléma azt jelenti, hogy megkeressük azt a két p és q prímszámot, amelyekre fennáll: $n = p \cdot q$.

- nagy számok esetében a faktorizációs problémára nem ismert hatékony algoritmus, jelenleg minimum 512 bites prímszámokat használnak,
- két távoli egység nyilvános csatornán történő kis méretű, titkosított információcseréjére (kulcscseréjére), illetve az egységektől származó adatok hitelesítésére alkalmas,
- a gyakorlatban a titkosított információ, a hitelesítésre szánt adat egy szám, amelyet a szakirodalom kontextustól függően szesszió kulcsnak (session key), vagy mesterkulcsnak master key, vagy szimmetrikus kulcsnak (symmetric key), vagy nyílt szövegnek (plaintext) mond,
- a TLS 1.3 protokollban digitális aláírás előállítására használják, 2018 előtt kulcscserére is használták, hasonlóan a Diffie-Hellman kulcscseréhez
- digitális aláírás: adott eszköztől származó adatok hitelesítésére alkalmas kriptográfiai eljárás

Az RSA rendszer

- a gyakorlatban az RSA-OAEP, vagy RSA-PSS rendszereket használják, amelyek az RSA módosított, biztonságos és standardizált változatai: PKCS#1 v2.1
- RSA-OAEP: kulcscserét, másképp fogalmazva aszimmetrikus titkosítást biztosít
- RSA-PSS: digitális aláírás (signature) használatát, másképp fogalmazva digitális tanusítványok (certificate) kibocsátását biztosítja,
- mindkettőt Bellare és Rogaway dolgozták ki az 1990-es évek közepén,
- az előadás keretén belül nem lesz szó az RSA-OAEP, RSA-PSS rendszerről, az 1977-es változat kerül bemutatásra, amelyet RSA-textbook, vagy baby-RSA néven említ a szakirodalom,
- a kulcsgeneráló algoritmus egy-egy értékpárt, pontosabban egy-egy kulcspárt határoz meg, ahol az egyik értékpárt publikus kulcsnak, a másik értékpárt privát kulcsnak hívjuk
- aszimmetrikus, publikus kulcsú kriptográfia: a kriptográfiának az a területe, amely a kulcscserék és digitális aláírások biztonságával foglalkozik

Publikus kulcsú titkosítás

- publikus kulcsú titkosítás:
  - feltételezve, hogy a kommunikációban résztvevő két egység **A** és **B**, akkor a protokoll első lépéseként **A**, vagy egy harmadik megbízható egység végrehajtja a kulcsgeneráló algoritmust, majd a publikus kulcsot egy nyilvános csatornán megosztja a **B** egységgel,
  - ha egy harmadik egység végzi a kulcsgenerálást, akkor a privát kulcsot egy titkos csatornán megosztja **A**-val,
  - **B** véletlenszerűen generál egy $1 < K < n$ egész számot, és a titkosító algoritmussal, illetve a publikus kulccsal meghatároz egy cK értéket, a cK-t elküldi egy nyilvános csatornán **A**-nak,
  - **A** a visszafejtő algoritmussal, illetve a privát kulccsal meghatározza a **K** értéket,
  - a megosztott titkos információ tehát a K lesz,
- az RSA esetében a rendszer helyessége az Euler-tétellel bizonyítható.

A baby-RSA rendszer - a titkosító változat

a kulcsgeneráló algoritmus:

- bemenete egy k biztonsági paraméter, amely a generált kulcsméretet jelenti,
- véletlenszerűen generál két k bites prímszámot, legyenek ezek p és q, és meghatározza az $n = p \cdot q$-t,
- kiszámolja: $\phi(n) = (p-1) \cdot (q-1)$,
- kiválasztja azt a legkisebb $1 < e < n$ számot, amelyre fennáll $(e, \phi(n)) = 1$,
- meghatározza e inverzét $\pmod{\phi(n)}$ szerint, legyen ez d, fennáll tehát $e \cdot d = 1 \pmod{\phi(n)}$,
- a publikus kulcs: (e, n), a privát kulcs: (d, n)
- a $p, q, \phi(n), d$ értékek szigorúan titkos adatok,

a titkosító algoritmus:

- bemeneti paramétere a publikus kulcs, és a K egész szám, ahol $1 < K < n$,
- meghatározza a $cK = K^e \pmod{n}$ titkosított értéket,

a visszafejtő algoritmus:

- bemeneti paramétere a privát kulcs és a titkosított érték,
- meghatározza $K = cK^d \pmod{n}$-t.

A baby-RSA rendszer, - a titkosító változat

Példa:

- Kulcsgenerálás
  - Legyen $p = 61,\ q = 97$ a két prímszám.
  - Meghatározzuk:
    - $n = 61 \cdot 97 = 5917$,
    - $\phi = (p-1) \cdot (q-1) = 60 \cdot 96 = 5760$.
  - Legyen $e = 7$, ahol $(7, \phi) = 1$.
  - Meghatározzuk e inverzét $\pmod{\phi}$ szerint, kapjuk: $d = 823$, mert $7 \cdot 823 = 1 \pmod{5760}$.
  - A publikus kulcs : $(7,\ 5917)$.
  - A privát kulcs : $(823,\ 5917)$.
- Titkosítás:
  - A $K = 2014$ meste értéket szeretnék titkosítani/megosztani. Ekkor a titkosított érték: $cK = 2014^7 \equiv 1526 \pmod{5917}$.
- Visszafejtés:
  - $K = 1526^{823} \equiv 2014 \pmod{5917}$.

A baby-RSA rendszer - a titkosító változat

2. feladat

Írjunk egy Python függvény, amely egy k egész szám bemeneti érték esetében a baby-RSA kulcsgeneráló algoritmusával meghatározza az e, d, n, p, q egész számokat.

```
from eload9 import primeGen
from math import gcd
def RSA_keyGen(k):
    p = primeGen(k, 10)
    q = primeGen(k, 10)
    n = p * q
    phi = (p-1) * (q-1)
    e = 3
    while True:
        if gcd(e, phi) == 1: break
        e += 2
    d = inverz(e, phi) #d = pow(e, -1, phi)
    return (e, d, n, p, q)
```

A `primGen` függvényt a 9. előadáson, míg az `inverz`-et egy pár oldallal előbbre adtuk meg.

A baby-RSA rendszer - a titkosító változat

3. feladat

Írjunk egy Python függvényt, amely `RSA_keyGen` függvény segítségével meghatároz egy publikus és privát kulcspárt, titkosít egy billentyűzetről beolvasott K értéket, majd a titkosított értéket visszafejti.

```
def RSA_fel():
    k = int(input('bit meret: '))
    e, d, n, p, q = RSA_keyGen(k)
    print ('publikus kulcs: ', e, n)
    print ('privat kulcs: ', d, n)
    print('kerek egy szamot, legyen kisebb mint: ', n)
    K = int(input())
    cK = pow(K, e, n)
    print ('titkositott ertek: ', cK)
    K = pow(cK, d, n)
    print ('visszafejtett ertek:', K)

>>> RSA_fel()
    bit meret: 128
    ...
```

A baby-RSA rendszer - a digitális aláírás változat

három algoritmust kell megadni:

- a kulcsgeneráló algoritmus: ugyanaz mint a titkosító változatnál
- az aláíró algoritmus:
  - bemeneti paramétere a privát kulcs, és a K egy egész szám, ahol $1 < K < n$,
  - nem az a szerepe, hogy titkosítsa a K értékét,
  - meghatározza a $sK = K^d \pmod{n}$ egész számot, azaz az aláírt értéket,
- az ellenőrző algoritmus:
  - bemeneti paramétere a publikus kulcs, a K egész szám, és az sK aláírt érték,
  - meghatározza $\hat{K} = sK^e \pmod{n}$,
  - ha $\hat{K} = K$-val akkor elfogadja az aláírást.

A baby-RSA rendszer, - a digitális aláírás változat

Példa:

- Kulcsgenerálás, ahogyan a titkosító változatnál tettük:
  - Legyen $p = 61,\ q = 97$ a két prímszám.
  - A publikus kulcs : $(7,\ 5917)$.
  - A privát kulcs : $(823,\ 5917)$.
- Aláírás előállítás:
  - A $K = 2023$ értéket szeretnék aláírni: $sK = 2023^{823} \equiv 3507 \pmod{5917}$.
- Aláírás ellenőrzés:
  - $\hat{K} = 3507^{7} \equiv 2023 \pmod{5917}$,
  - K egyenlő sK-val, tehát hiteles az aláírás.

Az RSA rendszer

- ha $n = p \cdot q$, akkor az Euler függvény: $\phi(n) = (p-1) \cdot (q-1)$,
- a generált p, q prímszámokat és a $\phi(n)$ értékét titokban kell tartani; a titkosító és visszafejtő algoritmusok nem használják őket
- az e értéket választhatjuk véletlenszerűen, a standard a 65537 konstans értékkel dolgozik,
- a standard előírja, hogy a p és a q minimum 512 bites prímszámok legyenek, ekkor az n 1024 bites lesz,
- ha d is megközelítőleg 1024 bites, akkor p és a q ismerete nélkül, egyelőre nincs algoritmus, amely meghatározná a d-t,
- a d értéke a p és a q ismeretében, a kiterjesztett euklideszi algoritmussal azonban meghatározható,
- ha a d értéke kicsi, akkor Wiener-algoritmusa megtudja határozni a d értékét, anélkül, hogy ismerné a p, q értékeket.

Az RSA rendszer

Miért alkalmazható a gyakorlatban a rendszer?

1024 bites kulcsok esetében is:

- hatékony algoritmussal meg lehet határozni a publikus és privát kulcsot: Miller-Rabin valószínűségi prímteszt, kiterjesztett euklideszi algoritmus,
- a publikus kulcs ismeretében hatékony algoritmussal meg lehet határozni a titkosított szöveget: moduláris hatványozó algoritmus,
- a privát kulcs ismeretében hatékony algoritmussal meg lehet határozni a nyílt szöveget: moduláris hatványozó algoritmus,
- a privát kulcs hiányában nem lehet meghatározni nyílt-szöveget.

Publikus kulcsú kriptográfia

Megjegyzések:

- a Diffie-Hellman és RSA rendszerek számok megosztására/titkosítására alkalmasak,
- a gyakorlatban bájt szekvenciák megosztására/titkosítására van szükség, amelyeket tehát át kell alakítani számmá,
- feltételezve, hogy t darab bájtot akarunk megosztani/titkosítani akkor ezeket a bájtokat 256-os számrendszerbeli számjegyeknek tekintve, ha átalakítjuk őket 256^t számrendszerbe, akkor egy nagy számot fogunk kapni,
- a kapott szám nem lehet nagyobb vagy egyenlő, mint n,
- a publikus és privát kulcsok külön vannak, állományokban eltárolva, általában base64 formában,
- a következő linken egy baby-RSA rendszeren alapuló *kliens-szerver* Python program található,
- a következő linken kriptográfia témakörben (Diffie-Hellman, RSA, stb.) találnak angol nyelvű bemutatóanyagokat: *videó!*

A kínai maradéktétel

- több kongruenciából álló egyismeretlenes szimultán kongruenciarendszer megoldását adja meg
- kínai matematikusok több mint 2000 éve ismerik a megoldást
- lehetővé teszi hogy a nagy számokkal szükséges számításokat kis számokkal végezhető műveletekre vezessünk vissza

Feladat: Ha egy tojásokkal teli kosárból kivesszük a tojásokat 2, 3, 4, 5, majd 6 -osával, akkor rendre 1, 2, 3, 4, 5 tojás marad mindig a kosárban. Ha 7-esével vesszük ki nem marad egy tojás sem. Hány tojás van a kosárban?

A feladat az alábbi kongruenciarendszerrel modellezhető:

$$\begin{aligned} x &\equiv 1 \pmod 2 \\ x &\equiv 2 \pmod 3 \\ x &\equiv 3 \pmod 4 \\ x &\equiv 4 \pmod 5 \\ x &\equiv 5 \pmod 6 \\ x &\equiv 0 \pmod 7 \end{aligned}$$

A kongruenciarendszer a kínai maradéktétellel oldható meg.

A kínai maradéktétel

2. tétel

Legyenek $m_1, m_2, ..., m_r$ pozitív, páronként relatív prímek. Ekkor az

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

kongruenciarendszernek $M = m_1 \cdot m_2 \cdot ... \cdot m_r$ modulus szerint egy megoldása van.

A megoldás meghatározásának menete:

- meghatározzuk: $M_k = M/m_k = m_1 \cdot m_2 \cdot ... \cdot m_{k-1} \cdot m_{k+1} \cdot ... \cdot m_r$,
- meghatározzuk az M_k értékek inverzét $\pmod{m_k}$ szerint, jelöljük ezeket $\hat{M}_k$-val,
- $x = a_1 \cdot M_1 \cdot \hat{M}_1 + a_2 \cdot M_2 \cdot \hat{M}_2 + ... + a_r \cdot M_r \cdot \hat{M}_r$ lesz a rendszer megoldása.

A kínai maradéktétel

A tojásos feladat az alábbi kongruenciarendszerre vezethető vissza:

$$\begin{aligned} x &\equiv 4 \pmod 5 \\ x &\equiv 11 \pmod{12} \\ x &\equiv 0 \pmod 7 \end{aligned}$$

mert

- az $x \equiv 3 \pmod 4$ megoldásai kielégítik az $x \equiv 1 \pmod 2$ megoldásait,
- az $x \equiv 5 \pmod 6$ megoldásai kielégítik az $x \equiv 2 \pmod 3$ megoldásait,
- az $x \equiv 11 \pmod{12}$ megoldásai kielégítik az $x \equiv 3 \pmod 4$ és $x \equiv 5 \pmod 6$ megoldásait.
- A megoldás menete:
  - $M = m_1 \cdot m_2 \cdot m_3 = 5 \cdot 12 \cdot 7 = 420$,
  - $M_1 = 84 \equiv 4 \pmod 5$ amelynek inverze 4, fennáll: $4 \cdot 4 = 1 \pmod 5$,
  - $M_2 = 35 \equiv 11 \pmod{12}$ amelynek inverze 11, fennáll: $11 \cdot 11 = 1 \pmod{12}$,
  - $M_3 = 60 \equiv 4 \pmod 7$ amelynek inverze 2, fennáll: $4 \cdot 2 = 1 \pmod 7$,
  - a rendszer megoldása: $x = 4 \cdot 84 \cdot 4 + 11 \cdot 35 \cdot 11 + 0 \cdot 60 \cdot 2 = 119 \pmod{420}$.

Az RSA, a kínai maradéktétel

- a kínai maradéktétel alkalmazható az RSA-nál, a visszafejtési folyamat gyorsítható vele, mert az n nagyságrendjével megegyező d hatványkitevő helyett két kisebb hatványkitevővel való hatványozást lehet végezni,
- mivel p, q prímszámok, fennáll a következő két összefüggés:

$$\begin{array}{lll} d \equiv dp \pmod{p-1} & \Leftrightarrow & K^d \equiv K^{dp} \pmod{p} \\ d \equiv dq \pmod{q-1} & \Leftrightarrow & K^d \equiv K^{dq} \pmod{q} \end{array}$$

- a következő egyenletrendszert megoldása, pedig a c^d értékét fogja adni, amelyet a kínai maradéktétellel oldhatunk meg:

$$\begin{array}{lll} x & \equiv & K^{dp} \pmod{p} \\ x & \equiv & K^{dq} \pmod{q} \end{array}$$

- meghatározzuk $dp, dq, \hat{M}q, \hat{M}p$ értékeket:

$$\begin{array}{ll} dp = d \pmod{p-1} & dq = d \pmod{q-1} \\ \hat{M}q = inverz(q, p) & \hat{M}p = inverz(p, q) \end{array}$$

- a $K^d \pmod{n}$ értéket megadja az x értéke, ahol

$$\begin{array}{l} x = (\hat{M}q \cdot q \cdot xp + \hat{M}p \cdot p \cdot xq) \pmod{n} \\ xp = K^{dp} \pmod{p} \\ xq = K^{dq} \pmod{q}. \end{array}$$

Az RSA, a kínai maradéktétel

Az alábbi Python függvény a korábban megadott `RSA_fel` függvényben, a visszafejtő `RSA_decrypt` függvény helyett a `RSA_decryptCR` függvényt hívja meg, paraméterként meg kell neki adni a p, q értékeket:

```
from eload10 import RSA_key_gen
def RSA_fel():
    k = int(input('bit meret: '))
    e, d, n, p, q = RSA_key_gen(k)
    print ('nyilvanos kulcs: ', e, n)
    print ('titkos kulcs: ', d, n)
    print ('titkos adatok: ', p, q)

    print('kerek egy szamot, kisebb legyen, mint ', n)
    K = int(input())
    cK = pow(K, e, n)
    print ('titkositott ertek: ', cK)
    K = RSA_decryptCR(cK, d, n, p, q)
    print ('visszafejtett ertek: ', K)
```

Az RSA, a kínai maradéktétel

A visszafejtő RSA_decryptCR függvény

```
def RSA_decryptCR(cK, d, n, p, q):
    dp = d % (p-1)
    dq = d % (q-1)
    Mq = inverz(q, p)
    Mp = inverz(p, q)
    cKp = pow(cK, dp, p)
    cKq = pow(cK, dq, q)
    K = (Mq * q * cKp + Mp * p * cKq) % n
    return K
```