Diszkrét matematika
12. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

Miről volt szó az elmúlt előadáson?

- A diszkrét logaritmus probléma
- A Diffie-Hellman kulcscsere
- Elliptikus görbék
- Elliptikus görbe Diffie-Hellman kulcscsere
- Lineáris kongruenciák
- Az RSA rendszer

Miről lesz szó?

- 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

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 ϕ-t: $\phi(n) = (p-1) \cdot (q-1)$,
- kiválasztja azt a legkisebb $1 < e < n$ számot, amelyre fennáll $(e, \phi) = 1$,
- meghatározza e inverzét $\pmod{\phi}$ szerint, legyen ez d, fennáll tehát $e \cdot d = 1 \pmod{\phi}$,
- a publikus kulcs: (e, n), a privát kulcs: (d, n)
- a p, q, ϕ, 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 $K = 2014$ 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

1. 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)
    q = primeGen(k)
    n = p * q
    phi = (p-1) * (q-1)
    e = 65537
    #e = 3
    #while True:
    #    if gcd(e, phi) == 1: break
    #    e += 2
    d = pow(e, -1, phi)
    return (e, d, n, p, q)
```

A primGen függvényt a 9. előadáson adtuk meg.

A baby-RSA rendszer - a titkosító változat

2. feladat

Írjunk egy Python függvényt, amely az `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
    ...
```

RSA, biztonsági problémák

- ha $n = p \cdot q$, akkor az Euler függvény: $\phi = (p-1) \cdot (q-1)$,
- a generált p, q prímszámokat és a ϕ értékét titokban kell tartani; a titkosító és visszafejtő algoritmusok nem használják őket
- az e értéket választhatjuk véletlenszerűen, a standard a 65537 konstans értékkel dolgozik,
- napjainkban a p és a q minimum 1024 bites prímszámok kell legyenek, ekkor az n 2048 bites lesz,
- ha d is megközelítőleg 2048 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.

RSA, a gyakorlatban

2048 bites kulcsok esetében is alkalmazható a gyakorlatban, mert:

- 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 biztonság miatt fontos a publikus kulcsok hitelesítése!!, ezt digitális aláírás alkalmazásával végzik
- baby-RSA rendszeren alapuló kliens-szerver Python program: *link*
- Diffie-Hellman, RSA, stb. videók: *link*

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.

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

1. 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:

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

- a következő egyenletrendszer 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:

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

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

Összetett számok faktorizációja

- az összetett számok faktorizációjával, azaz prímtényezők szotzatára való bontással az ókori görögök is intenzíven foglalkoztak,
- nagy összetett számok faktorizációjára nem ismert hatékony algoritmus,
- nagyobb számok faktorizációját a számítógépek megjelenése tette lehetővé,
- Peter Shor, 1997-ben bemutatott egy hatékony algoritmust, amely elméletben, kvantum számítógépek számításaira alapulva képes nagy számokat is faktorizálni, de 2001-ben még csak olyan kvantum gép létezett, amely a 15-ös számot volt képes faktorizálni,
- a mai kutatások olyan n összetett számokat próbálnak faktorizálni, ahol $n = p \cdot q$, és p, q prímszámok,
- RSA factoring Challenge: 1991-ben induló verseny, amely komoly pénzösszeggel jutalmazza azokat, akik bizonyos nagy számokat faktorizálnak:
  - 50,000$ kapott 2009-ben Thorsten Kleinjung és csapata a 768 bites RSA768 szám faktorizálásáért,
  - 100,000$ pénzjutalomat kap az, aki az 1024 bites RSA1024 számot faktorizálja, stb.

Összetett számok faktorizációja

A faktorizációs algoritmusok két csoportra oszthatók, a már bemutatott osztási próba és Eratosztenész szitája algoritmusok mellett:

- a speciális célú algoritmusok csak speciális alakú számok esetében alkalmazhatóak sikeresen, általános esetben nem működnek:
  - Fermat faktorizációs módszere: olyan összetett számok, ahol a p és q között kicsi a különbség
  - Pollard rho faktorizációs módszere: olyan összetett számok, ahol az osztók kis számok,
  - Pollard $p-1$ módszere: olyan összetett számok, ahol az osztók szomszédai kis számok,
  - Lenstra ECM módszere: elliptikus görbéken alapuló faktorizációs eljárás, leggyorsabb a speciális célú algoritmusok között, stb.
- az általános célú algoritmusok nagy memória és tár kapacitást igényelnek, ha a speciális célú algoritmusok nem adnak eredményt, akkor szokták őket használni:
  - faktorizálás lánctörtekkel,
  - a kvadratikus szita módszer,
  - az általánosított szám test szita (general number field sieve) módszer a leggyorsabb

Fermat faktorizációs módszere

Legyen $n = p \cdot q$, ahol feltételezzük, hogy a p és a q prímek közötti különbség kicsi:

- megpróbáljuk felírni n-et $n = a^2 - b^2$, alakba, ahol a, b tetszőleges egész számok, ekkor azonban $p = a - b$, $q = a + b$,
- meghatározzuk n négyzetgyökének felső egészrészét, legyen ez a
- $a, a+1, a+2, \ldots$ értékekre meghatározzuk a $b1 = a^2 - n$ értékeket, mindaddig amíg $b1$ nem lesz négyzetszám, ekkor megadható n két osztója: $a - \sqrt{b1}$ és $a + \sqrt{b1}$.

Példa. Határozzuk meg $n = 6283$ két prímosztóját, a Fermat faktorizációs módszerével:

$\lceil \sqrt{n} \rceil$	a	$b1 = a^2 - n$	négyzetszám-e?
80	80	117	nem
	81	278	nem
	82	441	igen

$b = \sqrt{b1} = \sqrt{441} = 21 \Rightarrow$

$$\begin{aligned} p &= 82 - 21 = 61, \\ q &= 82 + 21 = 103 \end{aligned}$$

Fermat faktorizációs módszere

3. feladat

Írjunk egy Python függvényt, amely Fermat faktorizációs módszerével meghatározza egy összetett szám prímtényezős felbontását.

```
from decimal import Decimal, getcontext
def fermatFaktorizacio(n):
    getcontext().prec = 400
    a = int(Decimal(n).sqrt()) + 1
    while True:
        b1 = a * a - n
        b = negyzetTeszt(b1)
        #print("%6i%6i%6i" % (a, b1, b))
        if b != -1:
            return (a - b, a + b)
        a += 1
```

```
def negyzetTeszt(x):
    i = int(Decimal(x).sqrt())
    if i*i == x: return i
    else: return -1
```

```
>>> fermatFaktorizacio(4668999961)
    (29033, 160817)
```

Fermat faktorizációs módszere

4. feladat

Írjunk egy Python függvényt, amely a compNr.txt állományban található számokat megpróbálja faktorizálni Fermat faktorizációs módszerével. Módosítsuk úgy a `fermatFaktorizacio` függvényt, hogy TIME LIMIT hibaüzenetet adjon, ha 10 másodperc alatt sem sikerül faktorizálni egy adott számot.

```
from time import time
from decimal import Decimal, getcontext
def fermatFaktorizacioTime(n):
    st = time()
    getcontext().prec = 400
    a = int(Decimal(n).sqrt()) + 1
    while True:
        b1 = a * a - n
        b = negyzetTeszt(b1)
        fs = time()
        if fs - st > 10: return -1
        if b != -1:
            return (a - b, a + b)
        a += 1
```

```
def fermatTime(nev = 'compNr.txt'):
    inf = open(nev, 'rt')
    temp = inf.read()
    L = temp.split('\n')
    inf.close()
    for elem in L:
        elem = int(elem)
        print(elem)
        res = fermatFaktorizacioTime(elem)
        if res == -1: print('TIME LIMIT')
        else: print(res)
        print()
```

Pollard ρ faktorizációs módszere

- John Pollard publikálta 1975-ben,
- az algoritmus keresi, azt a p számot, amely az n osztója lehet,
- ha a, b, két egész melyre: $0 < a, b < n$ és $a \neq b$, és $a = b \pmod{p}$, akkor $a - b$ a p egy többszöröse lesz,
- ekkor $p \leq gcd(a - b, n) < n$, azaz $a - b$ és n legnagyobb közös osztója egy nem triviális osztója lesz n-nek,
- az $f(x_i) = (x_{i-1}^2 + 1) \pmod{n}$ függvénnyel egy-egy számsorozatot generálunk, amelybe tulajdonképpen álvéletlen módon előállított számok kerülnek, ahol $x_0 = 1$,
- az előállított számsorozatban $a = x_i$ és $b = x_j$-re vizsgáljuk, hogy mikor lesz $gcd(x_i - x_j, n) \neq 1$,
- a hatékonyság miatt csak, ha $i = 2 \cdot j$, akkor számolunk legnagyobb közös osztót,
- a legnagyobb közös osztó meghatározását az eukleidészi algoritmussal végezzük.

Pollard ρ faktorizációs módszere

Határozzuk meg $n = 221$ két prímosztóját, a Pollard ρ féle faktorizációs módszerrel, ahol $x_0 = 1$:

a	b	$a - b$	$gcd(a - b, n)$
2	26	24	1
5	197	192	1
26	104	78	13
$\Rightarrow$			

$$p = 13,$$
$$q = n/13 = 17$$

Pollard ρ faktorizációs módszere

5. feladat

Határozzuk meg egy összetett szám prímtényezős felbontását a Pollard ρ faktorizációs algoritmussal.

```
from math import gcd
def pollard_rho(n):
    a = 1
    b = 2   #b = a * a + 1
    while True:
        a = (a * a + 1) % n
        b = (b * b + 1) % n
        b = (b * b + 1) % n
        d = gcd (a-b, n)
        print("%7i%7i%7i%7i" % (a, b, a-b, d))
        if 1 < d and d < n: return d
        if d == n: return n
```

Pollard ρ faktorizációs módszere

```
>>> pollard_rho(38989)
         2    26    -24     1
         5 29451 -29446     1
        26  5025  -4999     1
       677 10772 -10095     1
     29451 25123   4328     1
     12108  6581   5527     1
      5025 34775 -29750     1
     24743  8613  16130     1
     10772 16868  -6096   127
     (127, 307)

>>> pollard_rho(212612371982541690127801)
    (145812335489, 145812335609)

>>> pollard_rho(1490639506937854732063876 43)
    (10223, 14581233560968939959541)
```