Diszkrét matematika
10. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2022, őszi félév

Miről volt szó az elmúlt előadáson?

- a prímszámtétel
- prímszámokkal kapcsolatos sejtések
- 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?

- hatványok, generátor elemek, biztonságos prímek,
- a diszkrét logaritmus (DL) probléma,
- a Diffie-Hellman kulcscsere,
- a kiterjesztett eukleidészi algoritmus,
- lineáris kongruenciák,
- moduláris inverz,
- az RSA rendszer, illteve a *baby*-RSA rendszer,

Hatványok és generátor elemek

- az $x^n \pmod m$ értékek meghatározásakor számos tulajdonság figyelhető meg,
- Példa: meghatározzuk $x^n \pmod{11}$, értékeit $x = 2, 3, \ldots 10$-re, illetve $n = 1, 2, \ldots 10$-re:

$2^1 = 2$	$3^1 = 3$	$4^1 = 4$	$5^1 = 5$	$6^1 = 6$	$7^1 = 7$	$8^1 = 8$	$9^1 = 9$	$10^1 = 10$
$2^2 = 4$	$3^2 = 9$	$4^2 = 5$	$5^2 = 3$	$6^2 = 3$	$7^2 = 5$	$8^2 = 9$	$9^2 = 4$	$10^2 = 1$
$2^3 = 8$	$3^3 = 5$	$4^3 = 9$	$5^3 = 4$	$6^3 = 7$	$7^3 = 2$	$8^3 = 6$	$9^3 = 3$	$10^3 = 10$
$2^4 = 5$	$3^4 = 4$	$4^4 = 3$	$5^4 = 9$	$6^4 = 9$	$7^4 = 3$	$8^4 = 4$	$9^4 = 5$	$10^4 = 1$
$2^5 = 10$	$3^5 = 1$	$4^5 = 1$	$5^5 = 1$	$6^5 = 10$	$7^5 = 10$	$8^5 = 10$	$9^5 = 1$	$10^5 = 10$
$2^6 = 9$	$3^6 = 3$	$4^6 = 4$	$5^6 = 5$	$6^6 = 5$	$7^6 = 4$	$8^6 = 3$	$9^6 = 9$	$10^6 = 1$
$2^7 = 7$	$3^7 = 9$	$4^7 = 5$	$5^7 = 3$	$6^7 = 8$	$7^7 = 6$	$8^7 = 2$	$9^7 = 4$	$10^7 = 10$
$2^8 = 3$	$3^8 = 5$	$4^8 = 9$	$5^8 = 4$	$6^8 = 4$	$7^8 = 9$	$8^8 = 5$	$9^8 = 3$	$10^8 = 1$
$2^9 = 6$	$3^9 = 4$	$4^9 = 3$	$5^9 = 9$	$6^9 = 2$	$7^9 = 8$	$8^9 = 7$	$9^9 = 5$	$10^9 = 10$
$2^{10} = 1$	$3^{10} = 1$	$4^{10} = 1$	$5^{10} = 1$	$6^{10} = 1$	$7^{10} = 1$	$8^{10} = 1$	$9^{10} = 1$	$10^{10} = 1$

1. értelmezés

x rendjén, $\pmod p$ szerint, azt a legkisebb k kitevőt értjük, amelyre fennáll: $x^k \equiv 1 \pmod p$.

Példa:

- $\pmod{11}$ szerint a 2-es szám rendje 10, a 3 szám rendje 5, míg a 10-es szám rendje 2.

Hatványok és generátor elemek

1. tétel

Legyen p egy prímszám, x, u, v egész számok, ahol $(x, p) = 1$ és legyen k az x rendje (mod p) *szerint. Fennáll hogy* $x^u \equiv x^v \pmod{p}$, *akkor és csakis akkor ha* $u \equiv v \pmod{k}$.

Példa:

- (mod 11) szerint a 4-es szám rendje 5. Ekkor fennáll
$$2 \equiv 12 \pmod{5} \iff 4^2 \equiv 4^{12} \pmod{11},$$
- (mod 101) szerint a 65-ös szám rendje 10. Ekkor fennáll
$$4 \equiv 54 \pmod{10} \iff 65^4 \equiv 65^{54} \pmod{65}.$$

2. tétel

Legyen p egy prímszám. Ekkor mindig létezik egy g egész szám, $g \in \{1, 2, \ldots, p-1\}$, amelynek hatványértékei (mod p) *szerint előállítják az $\{1, 2, \ldots, p-1\}$ halmaz elemeit egy tetszőleges sorrendben. Ezeket az elemeket primitív gyököknek, vagy generátor elemeknek hívjuk.*

Példa:

- (mod 11) szerint 2, 6, 7, 8 generátor elemek.

Biztonságos prímek és generátor elemek

Ha p egy prímszám, akkor

- egy generátor elem rendje $p-1$,
- a generátor elemek száma: $\phi(p-1)$,
- Példa: (mod 11) szerint a generátor elemek száma $\phi(10)=4$,
- fontos feladat, hogy egy adott prímszám esetében meghatározunk egy generátor elemet.

2. értelmezés

Biztonságos prímeknek (safe prime) hívjuk azokat a p prímszámokat, amelyek egyenlőek $2 \cdot q + 1$*-el, ahol q is prímszám.*

- egy biztonságos prím meghatározása jóval időigényesebb, mint egy "közönséges" prím kiválasztása,
- ha a p biztonságos prím, akkor a generátor elem kiválasztása nem számít nehéz feladatnak,

Példa:

- $11 = 2 \cdot 5 + 1, 47 = 2 \cdot 23 + 1$ biztonságos prímek,
- $3, 13, 17, 41$, *stb.* nem biztonságos prímek,

Biztonságos prímek és generátor elemek

3. tétel

Egy p biztonságos prímszám esetében a g szám generátor elem lesz (mod p) *szerint, ha fennáll:*

$$g \not\equiv \pm 1 \pmod{p} \text{ és } g^q \not\equiv 1 \pmod{p}.$$

A fenti értelmezés alapján, ha a p biztonságos prím, akkor a generátor elem kiválasztása nem számít nehéz feladatnak, egy hatványértéket kell megvizsgálni.
Példa:

- legyen $p = 47 = 2 \cdot 23 + 1$, $p - 1 = 46$,
  - $g = 13$ generátor elem?
  - Igen, mert $13^{23} = 46 \pmod{47}$.
  - $g = 3$ generátor elem?
  - Nem, mert $3^{23} = 1 \pmod{47}$.

Ha p biztonságos prím, akkor a generátor elemek száma $\phi(p-1) = \phi(2) \cdot \phi(q) = q - 1$.

Biztonságos prímek és generátor elemek

1. feladat

Írjunk Python függvényt, amely generál egy k bites biztonságos prímet és meghatározza a prím egy generátor elemét.

```
from random import getrandbits, randint
from eload9 import miller_rabinT
def safePrime(k):
    q = getrandbits(k)
    if not (q & 1): q += 1
    while True:
        p = 2 * q + 1
        if miller_rabinT(q, 10) and miller_rabinT(p, 10): break
        q = getrandbits(k)
        if not (q & 1): q += 1
    while True:
        g = randint(2, p-2)
        temp = pow(g, q, p)
        if temp != 1: break
    return p, g
```

```
>>> safePrime(32)
    (518189279, 466351168)
```

A diszkrét logaritmus (DL) probléma

- a diszkrét logaritmus problémának (DL probléma) több verziója is megadható,
- a $\pmod p$, illetve az elliptikus görbékre megadott értelmezések mindegyikét széles körben alkalmazzák az adatbiztonság területén,

A következő értelmezés a $\pmod p$ szerinti verzió:

3. értelmezés

Az egész számok $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ véges halmaza esetében, ahol p prímszám és g generátor elem a diszkrét logaritmusa probléma azt jelenti, hogy megkeressük azt az a pozitív egész számot, melyre fennáll:

$$g^a \equiv A \pmod p,$$

ahol $g, A \in \mathbb{Z}_p^*$.

- az a számot A g alapú diszkrét logaritmusának hívjuk,
- nagy számok esetében a DL problémára nem ismert hatékony algoritmus, jelenleg minimum 1024 bites prímszámot használnak,
- számos kriptorendszer biztonsága alapszik a DL problémán.

A diszkrét logaritmus (DL) probléma, brute force algoritmus

2. feladat

*Írjunk egy Python függvényt, amely meghatározza a g^0, g^1, g^2, ... értékeket, mindaddig, amíg nem talál egy **A**-val kongruens értéket, azaz határozza meg A g alapú diszkrét logaritmusát.*

```
def dLog(g, A, p):
    for i in range(1, p-1):
        if pow(g, i, p) == A:
            return i

>>> dLog(2, 10, 19)
    17

>>> dLog(1234, 10000, 530063)
    73132
```

- a p értékének a növelésével az algoritmus futási ideje elfogadhatatlanná válik,
- léteznek nem triviális algoritmusok a DL problémára:
  - a baby-step giant-step (Shanks) algoritmus,
  - a Pohlig-Hellman algoritmus,
  - az indexkalkulus algoritmus.

A Diffie-Hellman kulcscsere

- 1976-ban publikálták a szerzők, biztonsága a DL problémán alapszik,
- mérföldkőnek számít, amely alapjaiban meghatározta/meghatározza a számítógépes adatcsere biztonságát,
- két távoli egység (számítógép, mobileszköz, stb.) kulcscsere mechanizmusára adott elsőként megoldást: hogyan tud megosztani két távoli egység egy nyilvános csatornán egy egész számot (bájt szekvenciát), amelyet rajtuk kívül más egység nem tud meghatározni,
- megfigyelhető egy áttérés a $\pmod{p}$ aritmetikáról az elliptikus görbékre, azért hogy a matematikai műveleteket kisebb prímszámmal lehessen végezni,
- feltételezve, hogy a kommunikációban résztvevő két egység **A** és **B**, akkor a protokoll első lépéseként **A** és **B** egy hiteles szervertől lekér egy p prímszámot és a p-nek egy g generátor elemét,
- a p és g értékek nem titkosak bármilyen más egység ismerheti őket,
- az **A** és **B** között megosztott egész számot a szakirodalom mester kulcsnak (master key), titkos információnak nevez.

A Diffie-Hellman kulcscsere

- **A** a p, g ismeretében meghatározza az a, A értékeket, ahol:
  - $a \in \{2, \ldots, p-2\}$ véletlenszerűen kerül kiválasztásra,
  - $A = g^a \pmod{p}$,
  - az a értékét titokban tartja, A-t pedig elküldi **B**-nak.
- **B** a p, g ismeretében meghatározza a b, B értékeket, ahol:
  - $b \in \{2, \ldots, p-2\}$véletlenszerűen kerül kiválasztásra,
  - $B = g^b \pmod{p}$,
  - a b értékét titokban tartja, B-t pedig elküldi **A**-nak.
- **A** a közös K kulcsot a kövekezőképpen határozza meg:

$$K = B^a \pmod{p}.$$

- **B** a közös K kulcsot a kövekezőképpen határozza meg:

$$K = A^b \pmod{p}.$$

- Helyesség:

$$K = A^b = B^a = g^{ab} \pmod{p}.$$

A Diffie-Hellman kulcscsere, példa

- Legyen $p = 47, g = 13$, ahol $p = 2 \cdot 23 + 1$. Fennáll, hogy $13^{23} \equiv 46 \not\equiv 1 \pmod{47}$, azaz 13 generátor elem.
  - **A** számításai:
    - választ egy random a számot, legyen ez 12 és meghatározza $A = 13^{12} \equiv 9 \pmod{47}$,
    - elküldi **B**-nek az $A = 9$-es számot,
  - **B** számításai:
    - választ egy random b számot, legyen ez 34 és meghatározza $B = 13^{34} \equiv 21 \pmod{47}$,
    - elküldi **A**-nak a $B = 21$-es számot,
  - **A** számítása: $K = 21^{12} = 16 \pmod{47}$,
  - **B** számítása: $K = 9^{34} = 16 \pmod{47}$.
- a közös kulcs: $K = 16$.

A következő linken egy Diffie-Hellman kulcserén alapuló *kliens-szerver* Python program található.

A kiterjesztett eukleidészi algoritmus

- az a és b egész számok legnagyobb közös osztója, az a d legnagyobb pozitív egész szám, amely osztja az a-t és b-t is,
- a d legnagyobb közös osztó mindig felírható az a és b lineáris kombinációjaként: $d = x \cdot a + y \cdot b$, ahol x, y egész számok,
- Eukleidész algoritmusa meghatározza a d legnagyobb közös osztót, a kiterjesztett Eukleidészi algoritmus pedig az x, y egész számokat is,
- a kiterjesztett Eukleidészi algoritmus futási ideje logaritmikus éppen ezért fontos szerepet tölt be a számítógépes adatbiztonság területén:

4. tétel (Lamé tétele)

Az eukleidészi algoritmus során végzett osztások száma nem lesz nagyobb, mint ötször a kisebbik szám számjegyeinek száma

- fontos az eset, amikor az a és b egész számok relatív prímek, azaz a és b legnagyobb közös osztója 1.

A kiterjesztett eukleidészi algoritmus

Legyenek a, b pozitív egész számok, akkor felírható a következő összefüggés: $(a, b) = x_n \cdot a + y_n \cdot b$, $n \in \{0, 1, 2, \dots\}$, ahol x_n és y_n a következő számítási sorozat n-ik tagjai:

$$\begin{aligned} x_0 &= 1, \ y_0 = 0 \\ x_1 &= 0, \ y_1 = 1 \\ x_j &= x_{j-2} - q_{j-1} \cdot x_{j-1} \\ y_j &= y_{j-2} - q_{j-1} \cdot y_{j-1}, \end{aligned}$$

ahol q_j a megfelelő hányados, amelyet az eukleidészi algoritmus során számolunk ki és az iterációt addig végezzük, míg 0 maradékot nem kapunk.

A kiterjesztett eukleidészi algoritmus, példa

Határozzuk meg 76 és 32 legnagyobb közös osztóját, majd azokat az x és y egész számokat, melyekre fennáll a következő összefüggés: $76 \cdot x + 32 \cdot y = d$, ahol $d = (76,\ 32)$.

a	b	q	r	x_0	x_1	y_0	y_1
				1	0	0	1
76	32	2	12	0	1	1	-2
32	12	2	8	1	-2	-2	5
12	8	1	4	-2	3	5	-7
8	4	2	0				

Tehát a megoldás: $d = 4,\ x = 3,\ y = -7$, és fennáll a következő összefüggés: $76 \cdot 3 + 32 \cdot (-7) = 4$.

A kiterjesztett eukleidészi algoritmus, példa

Határozzuk meg 15 és 56 legnagyobb közös osztóját, majd azokat az x és y egész számokat, melyekre fennáll a következő összefüggés: $15 \cdot x + 56 \cdot y = d$, ahol $d = (15,\ 56)$.

a	b	q	r	x_0	x_1	y_0	y_1
				1	0	0	1
15	56	0	15	0	1	1	0
56	15	3	11	1	-3	0	1
15	11	1	4	-3	4	1	-1
11	4	2	3	4	-11	-1	3
4	3	1	1	-11	15	3	-4
3	1	3	0				

Tehát a megoldás: $d = 1,\ x = 15,\ y = -4$, és fennáll a következő összefüggés: $15 \cdot 15 + 56 \cdot (-4) = 1$.

A kiterjesztett eukleidészi algoritmus

3. feladat

Írjunk egy Python függvényt, amely Eukleidész kiterjesztett algoritmusával határozza meg az a és b legnagyobb közös osztóját, a d-t és azokat az x, y egész számokat amelyekre fennáll: $d = x \cdot a + y \cdot b$.

```
def extEuclid(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while True:
        q = a // b
        r = a - b * q
        if r == 0:
            return b, x1, y1
        x = x0 - q * x1
        y = y0 - q * y1
        x0, x1, y0, y1 = x1, x, y1, y
        a, b = b, r

>>> extEuclid(15, 56)
    (1, 15, -4)
```

A kiterjesztett eukleidészi algoritmus

Kiegészítjük az előző függvényt, hogy kiírassuk a részértékeket:

```
def extEuclid_(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    print("%4s%4s%4s%4s%4s%4s%4s%4s" % ("a","b","q","r","x0","x1","y0","y1"))
    print("%4s%4s%4s%4s%4i%4i%4i%4i" % ("","","","",x0,x1,y0,y1))
    while True:
        q = a // b
        r = a - b * q
        if r == 0:
            print("%4i%4i%4i%4i" % (a,b,q,r))
            return b, x1, y1
        x = x0 - q * x1
        y = y0 - q * y1
        x0, x1, y0, y1 = x1, x, y1, y
        print("%4i%4i%4i%4i%4i%4i%4i%4i" % (a,b,q,r,x0,x1,y0,y1))
        a, b = b, r

>>> extEuclid_(15, 56)
    a   b   q   r  x0  x1  y0  y1
                   1   0   0   1
    ...
```

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

5. 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 Eukleidészi 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 eukleidészi 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:
  - $9 \cdot 8 = 72 = 12 \pmod{15}$,
  - $9 \cdot 13 = 117 = 12 \pmod{15}$,
  - $9 \cdot 3 = 27 = 12 \pmod{15}$,

Moduláris inverz

4. értelmezés

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

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? 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? Mivel kell megszorozni 7-et, hogy 1-et kapjunk?
- Válasz: 9-cel, mert $7 \cdot 9 = 1 \pmod{31}$.

Moduláris inverz

Egy szám moduláris inverzét többféleképpen is meghatározhatjuk:

- brute-force módszerrel: az összes lehetséges értéket sorra próbáljuk
- a kiterjesztett Eukleidész algoritmussal
- 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: $25 \cdot 4 = 28 \pmod{36}$.

Moduláris inverz

4. feladat

A kiterjesztett Eukleidész algoritmussal határozzuk meg a inverzét $\pmod m$ szerint.

```
def inverz_(a, m):
    d, x, y = extEuclid(a, m)
    if d != 1:
        print ('nincs inverz')
        return -1
    return x % m

>>> inverz_(7, 31)
9
```

Megjegyzések:

- A kiterjesztett Eukleidész algoritmus az x és y értékekben negatív számot is meghatározhat.
- azt szeretnénk hogy a moduláris inverz, ha létezik, akkor az mindig pozitív érték legyen
- a Python % operátora a legkisebb pozitív maradékot határozza meg:

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

Moduláris inverz

A kiterjesztett Eukleidész algoritmusban az y változóval végzett műveleteket elhagyhatjuk, ha az algoritmust inverz számításra használjuk:

```
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
```

Az RSA rendszer

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

5. 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 információ cseréjére alkalmas,
- a gyakorlatban a titkosított információ egy szám, amelyet a szakirodalom kontextustól függően mesterkulcsnak, vagy nyílt szövegnek (plaintext) mond,
- a gyakorlatban az RSA-OAEP rendszert használják, amely az RSA egy módosított, biztonságos és standardizált változata,
- az RSA-OAEP-t Bellare és Rogaway dolgozták ki, és 1995-ben publikálták,
- az előadás keretén belül nem lesz szó az RSA-OAEP rendszerről, az 1977-es változat kerül bemutatásra, amelyet RSA-textbook, vagy baby-RSA néven említ a szakirodalom.

A baby-RSA rendszer

- a rendszerben meg kell adni három algoritmust: a kulcsgeneráló, a titkosító, és a visszafejtő algoritmusokat,
- a kulcsgeneráló algoritmus egy-egy értékpárt, pontosabban egy-egy kulcspárt határoz meg, ahol az egyik értékpárt publikus kulcsnak, a másik értékpárt privát kulcsnak hívjuk,
- 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,
- a rendszer helyessége az Euler-tétellel bizonyítható.

A baby-RSA rendszer

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,
- kiválasztja azt a legkisebb $1 < e < n$ számot, amelyre fennáll $(e, \phi(n)) = 1$,
- meghatározza e moduláris 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 titkosító algoritmus:

- bemeneti paramétere a publikus kulcs, és a K nyílt-szöveg (K egy egész szám, ahol $1 < K < n$),
- meghatározza a $cK = K^e \pmod{n}$ egész számot, a titkosított szöveget,

a visszafejtő algoritmus:

- bemeneti paramétere a privát kulcs és a titkosított szöveg,
- meghatározza $K = cK^d \pmod{n}$ nyílt-szöveget,

A baby-RSA rendszer, 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 $K = 2014$ mesterkulcs 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

5. 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)
    return (e, d, n, p, q)
```

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

Az RSA rendszer- *baby* változat

6. 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

- 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 eukleidészi algoritmussal azonban meghatározható,
- ha a d értéke kicsi, akkor Wiener-algoritmusa megtudja határozni a d értékét, anélkül, hogy ismerné a p, q értékeket.

A baby-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 eukleidészi 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.

A baby-RSA rendszer

Megjegyzések:

- a Diffie-Hellman és RSA rendszerek **nagy** 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!*