MÁRTON GYÖNGYVÉR

KRIPTOGRÁFIAI ALAPISMERETEK

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MŰSZAKI ÉS HUMÁNTUDOMÁNYOK KAR
MATEMATIKA–INFORMATIKA TANSZÉK

MÁRTON GYÖNGYVÉR

KRIPTOGRÁFIAI ALAPISMERETEK

Scientia Kiadó
Kolozsvár·2008

A kiadvány megjelenését támogatta:

Lektor:
Ködmön József (Nyíregyháza)

Sorozatborító:
Miklósi Dénes

Descrierea CIP a Bibliotecii Naţionale a României
MÁRTON GYÖNGYVÉR
Kriptográfiai alapismeretek / Márton Gyöngyvér. – Cluj-Napoca: Scientia, 2008.
Bibliogr.
ISBN 978-973-1970-00-4

TARTALOM

CONTENTS

CUPRINS

Jelölések

A könyvben használt jelölésrendszer nem követ semmiféle nemzetközi standard rendszert, ezért az alábbiakban feltüntetjük ezeket.

Jelölés	*Elnevezés*
$gcd(x,\ y)$	x és y legnagyobb közös osztója
$x <> y$	x különbözik y-tól
$x = y$	x egyenlő y-nal
$x := y$	x felveszi y értékét
$x * y$	x y-nal való szorzata
$x \vert y$	x osztója y-nak
$x\ mod\ y$	x y-nal való osztási maradéka
$x\ div\ y$	x y-nal való osztási egész része
x^ y	x az y hatványon
[]	üres lista
$[x]$	az x-et tartalmazó egyelemű lista
$x + y$	listák esetében az x lista után fűzzük az y lista elemeit
$x + [k]$	listák esetében az x listához hozzáadjuk a k elemet
$x[i]$	listák esetében az x lista i-dik eleme
$x[0]$	listák esetében nincs értelmezve
$x[1]$	listák esetében a lista első eleme
$x[i,\ j]$	az x két dimenziós tömb i sorának j oszlopbeli eleme
$length(x)$	a x lista elemeinek száma
$sqrt(x)$	az x négyzetgyökét meghatározó függvény
$floor(x)$	az x alsó egészrészét meghatározó függvény
$\lfloor x \rfloor$	az x alsó egészrésze
$ceil(x)$	az x felső egészrészét meghatározó függvény
$\lceil x \rceil$	az x felső egészrésze
$rand(x_1,\ x_2)$	x_1 és x_2 között véletlenszámot generáló függvény
log	kettes alapú logaritmus
ln	természetes alapú logaritmus
ϕ	Euler phi-függvény
$det\ A$	az A mátrix determinánsa
$adj\ A$	az A adjungált mátrix
$\odot$	szorzás mod $2^n + 1$
$\oplus$	moduláris összeadás
$\boxplus$	összeadás mod 2^n

1. FEJEZET

BEVEZETŐ

Jelen egyetemi jegyzet célja, hogy azon kriptográfiai alapismeretekkel megismertesse az olvasót, melyekre elengedhetetlenül szükség van az informatikai, számítástechnikai világban. A jegyzet kifejezetten főiskolás diákoknak készült, és azokat a kérdésköröket ismerteti, melyeket a kriptográfiához kapcsolódó elméleti és gyakorlati órákon el kell sajátítaniuk. Ezért feltételezi, hogy a diákok rendelkeznek programozói, illetve minimális számelméleti ismeretekkel.

A kriptográfia matematikai hátterének nincs külön fejezet szentelve, a megfelelő fogalmakat ott ismertetjük, ahol szükség van rájuk. Ez elsősorban azért van így, mert a jegyzet nem annyira elméleti, mint inkább gyakorlati szempontból vezeti be az olvasót a digitális adatbiztonsághoz kapcsolódó ismeretekbe. Ha adott helyen olyan fogalmat, kijelentést használunk, amelyet korábban adtunk meg, akkor vagy a megadott hivatkozási pontot megkeresve, vagy a jegyzet végén található tárgymutató segítségével könnyedén rákereshetünk arra a részre, ahol a keresett kifejezés, értelmezés található.

A jegyzet első részében azoknak az algoritmusoknak a leírása és pszeudokódbeli implementálása van megadva, amelyeket a későbbi fejezetekben bemutatásra kerülő kriptográfiai rendszereknél, protokolloknál használunk. Az algoritmusok pszeudokódjainál a megszokott feltételes és ciklikus utasításokat használjuk, adatszerkezetként legtöbb helyen listát alkalmazunk, az ezekkel kapcsolatos műveleteket pedig a *Jelölések* fejezetben adtuk meg, úgyszintén itt tüntetjük fel a további sajátos jelöléseket. Az algoritmusok mindegyikéhez hosszabb vagy rövidebb magyarázatot írtunk, illetve számpéldákon keresztül próbáltuk a könnyebb megértést biztosítani. Az algoritmusok bonyolultságára e jegyzet keretén belül nem térünk ki, erre vonatkozó szakirodalmat a [19]-ben találunk.

A jegyzet második része a titkos kulcsú kriptográfiai rendszereket mutatja be, a harmadik rész pedig a nyilvános kulcsú kriptográfiát. A nyilvános kulcsú kriptográfia keretén belül az ilyen kriptográfiai rendszerekről, a hash függvényekről és a digitális aláírásokról beszélünk.

Az utolsó részben gyakorlatban alkalmazott kriptográfiai protokollokat ismertetünk.

2. FEJEZET

ALAPALGORITMUSOK

2.1. Számrendszerek

Mennyiségek értékeinek a kifejezésére a legáltalánosabban elfogadott jelölésrendszer a 10-es számrendszerbeli számjegyek használata. Ennek a jelölésmódnak a használata semmi mással nem magyarázható, mint hogy az embereknek 10 ujjuk van. Léteztek azonban olyan civilizációk, melyek ettől eltérő alapú számrendszereket használtak. Napjainkban is számos, a 10-es számrendszertől eltérő alapú számrendszert használnak. Például a számítógép 2-es alapú, a számítástechnikában pedig akár 2^k alapú számrendszert is használhatunk, ahol k egy tetszőleges pozitív egész szám. A következő algoritmusok éppen ezért a fontosabb számrendszerek közötti átalakításokat mutatják be, részletesebben lásd a [24] könyvben.

Matematikailag bebizonyítható, hogy bármely 1-nél nagyobb szám alkalmas arra, hogy számrendszerként használják: azaz bármely $1 \leq sz$ szám egyértelműen felírható $1 \leq p$ alapú számrendszerben a következőképpen:

$$sz = a_0 + a_1 \cdot + \ldots + a_{n-1} \cdot p^{n-1},$$

ahol n nem negatív egész szám és a_j-re fennáll, hogy:

$$0 \leq a_j \leq p - 1,\ j = 0,\ 1,\ \ldots,\ n.$$

Ha a számjegyek 0 és 9 közötti értékek, akkor 10-es alapú számrendszerbeli számjegyeket jelölnek, ha a 0 és az 1 értékek, akkor biteknek nevezzük őket.

A következő leírásra kerülő algoritmus egy 10-es alapú számrendszerben megadott számot, jelölje ezt a *szam* változó, átalakít p alapú számrendszerbeli számmá, ahol az eredmény egy lista, az új számrendszerbeli számjegyekkel. Az új számrendszerbeli számjegyeket megkapjuk, ha meghatározzuk a *szam* p-vel való osztási maradékát, ismételve ezt az elgondolást úgy, hogy a *szam*-nak megfeleltetjük a *szam* p-vel való osztási egészrészét, folytatva az eljárást, míg a $szam \neq 0$.

2.1. algoritmus.
Bemenet: a *szam* és a p pozitív egész számok.
Kimenet: egy lista, az *eredlist*, mely tartalmazza a *szam* p alapú számrendszerbeli számjegyeit.

```
atalakitas1 := proc(szam, p)
    eredlista := [];
    while (szam <> 0) do
        eredlista := eredlista + [szam mod p];
        szam := szam div p;
    end do;
    return eredlista;
end proc;
```

2.1. megjegyzés. Az *eredlista*-ba fordított sorrendbe kerülnek az új számrendszerbeli szám számjegyei. Ezt a tárolási formát a továbbiakban is követjük.

2.1. példa. Határozzuk meg a 256, 10-es alapú számrendszerbeli szám 2-es alapú számrendszerbeli alakját. Az algoritmus eredményként a $[0, 0, 0, 0, 0, 0, 0, 0, 1]$ listát adja.

2.2. példa. Határozzuk meg a 783 759, 10-es alapú számrendszerbeli szám 16-os alapú számrendszerbeli alakját. Az algoritmus eredményként a $[15, 8, 5, 15, 11]$ listát adja.

2.2. megjegyzés. Egy 16-os alapú számrendszerben 16 különböző jegyünk van, ezeket jelölhetjük rendre

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A,\ B,\ C,\ D,\ E,\ F\text{-fel},$$

illetve

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15\text{-tel}.$$

A jegyzetben ez utóbbi jelölést használjuk.

A következő leírásra kerülő algoritmus egy p alapú számrendszerben megadott számot, ahol a számjegyek listabeli elemekként vannak meghatározva, átalakít 10-es alapú számrendszerbe. Feltételezve, hogy a szám n számjegyből áll és a számjegyek $a_0,\ a_1,\ \ldots,\ a_{n-1}$, akkor az eljárás a következő összegzést végzi el:

$$a_0 + a_1 \cdot p + \ldots + a_{n-1} \cdot p^{n-1}.$$

2.2. algoritmus.
Bemenet: a *lista* és a p pozitív egész szám, ahol a *lista* a p alapú számrendszerben megadott számjegyeket tartalmazza.
Kimenet: az *ered* 10-es alapú számrendszerbeli szám.

```
atalakitas2:= proc(lista, p)
    ered := 0;
    hatv := 1;
```

```
    for i from 1 to n length(lista) do
        ered := ered + lista[i] * hatv;
        hatv := hatv * p;
    end do;
    return ered;
end proc;
```

2.3. példa. Határozzuk meg a $[0, 0, 0, 0, 0, 0, 0, 0, 1]$ szám, mint 2-es alapú számrendszerbeli számjegyek, 10-es alapú számrendszerbeli alakját. Az algoritmus eredményként a 256 számot adja.

2.4. példa. Határozzuk meg a $[15, 8, 5, 15, 11]$ szám, mint 16-os alapú számrendszerbeli számjegyek, 10-es alapú számrendszerbeli alakját. Az algoritmus eredményként 783 759 számot adja, ahol $783\,759 = 11 \cdot 16^4 + 15 \cdot 16^3 + 5 \cdot 16^2 + 8 \cdot 16^1 + 15 \cdot 16^0$.

2.3. megjegyzés. A *lista* változóba fordított sorrendbe kell megadnunk a szám számjegyeit.

Ahhoz, hogy egy p_1 alapú számrendszerből p_2 alapú számrendszerbe tudjunk átalakítani, ahol p_1 és p_2 tetszőleges pozitív egész számok, először alkalmazzuk az *atalakitas*2 algoritmust átalakítva a számot p_1 alapú számrendszerből 10-esbe, majd meghívjuk az *atalakitas*1 algoritmust, átalakítva a kapott 10-es alapú számrendszerbeli számot p_2 alapú számrendszerbe.

A p és p^k alapú, ahol p, k tetszőleges pozitív egész számok, számrendszerek közötti átalakítás a 10-es alapú számrendszer elkerülésével is megvalósítható, ezért ezen algoritmusok pszeudokódját külön is megadjuk, lásd [24].

A következő algoritmus egy p alapú számrendszerben megadott számot, ahol a számjegyek listabeli elemek, átalakít p^k alapú számrendszerbe, az eredmény egy lista lesz. Az elgondolás a következő: csoportosítjuk a p alapú számrendszerben megadott szám számjegyeit k-asával, jelölve őket $a_0,\ a_1,\ \ldots,\ a_{k-1}$-el, majd mindegyik csoport esetében elvégezzük a következő összegzést:

$$a_0 + a_1 \cdot p + \ldots + a_{k-1} \cdot p^{k-1}.$$

Elvégezve csoportonként az összegzéseket, megkapjuk az új alapú számrendszerbeli szám számjegyeit.

2.3. algoritmus.
Bemenet: a *lista*, p, k pozitív egész számok, a fenti leírásnak megfelelően.
Kimenet: az *eredlista*, a p^k alapú számrendszerben levő számjegyekkel.

```
atalakitas3 := proc(lista, p, k)
   ered := 0;
```

```
   hatv := 1;
   eredlista := [];
   for i from 1 to length(lista) do
       ered := ered + lista[i] * hatv;
       hatv := hatv * p;
       if (i mod k = 0) then
           eredlista := eredlista + [ered];
           hatv := 1;
           ered := 0;
       end if;
   end do;
   if (ered <> 0) then
       eredlista := eredlista + [ered];
   end if;
   return eredlista;
end proc;
```

2.5. példa. Határozzuk meg a $[1, 1, 0, 1, 0, 1, 1, 1, 0, 1]$ 2-es alapú számrendszerben írt szám, $2^4 = 16$-os alapú számrendszerbeli alakját. Az algoritmus meghívása a következőképpen történik:

$$atalakitas3\ ([1, 1, 0, 1, 0, 1, 1, 1, 0, 1],\ 2,\ 4).$$

Az eredmény a $[11, 14, 2]$ 16-os alapú számrendszerbeli számokat tartalmazó lista lesz.

A következő algoritmus egy p^k alapú számrendszerben megadott számot, ahol a számjegyek listabeli elemekként vannak megadva, átalakít p alapú számrendszerbe. Az eredmény egy lista lesz, a megfelelő számjegyekkel. Mindegyik p^k alapú számrendszerbeli számjegyre elvégezzük a következőket: jelöljük a *szam* változóval egy p^k alapú számrendszerbeli számjegyet, ekkor meghatározzuk a *szam* p-vel való osztási maradékát, ismételve ezt az elgondolást úgy, hogy a *szam*-nak megfeleltetjük a *szam* p-vel való osztási egészrészét, folytatva az eljárást, míg a $szam \neq 0$.

2.4. algoritmus.
Bemenet: a *lista*, p, k pozitív egész számok, a fenti leírásnak megfelelően.
Kimenet: az *eredlista*, a p alapú számrendszerben levő számjegyekkel.

```
atalakitas4 := proc(lista, p, k)
    eredlista := [];
    for i from 1 to length(lista) do
        szam := lista[i];
        for j from 1 to k do
            eredlista := eredlista + [szam mod p];
            szam := szam div p;
        end do;
```

```
    end do;
    return eredlista;
end proc;
```

2.6. példa. Határozzuk meg a $[11, 14, 2]$ 16-os alapú számrendszerben írt szám 2-es alapú számrendszerbeli alakját. Az algoritmus meghívása a következőképpen történik:

$$atalakitas4\ ([11, 14, 2],\ 2,\ 4).$$

Az eredmény az $[1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0]$ 2-es alapú számrendszerbeli számokat tartalmazó lista lesz.

2.2. Az euklidészi algoritmus és alkalmazásai

2.2.1. Az euklidészi algoritmus

Az algoritmus, mely egyike a legrégebbi, napjainkban is alkalmazott eljárásoknak, meghatározza két egész szám, legyenek ezek a és b, legnagyobb közös osztóját, jelöljük ezt $gcd(a,\ b)$-vel. Az algoritmus kidolgozása Eukleidész ókori görög matematikusnak köszönhető és egyik legfontosabb érdeme az, hogy anélkül határozza meg a két szám legnagyobb közös osztóját, hogy a számokat prímtényezőkre bontaná, leírását lásd [3]. Az eljárás során meghatározzuk a, b-vel való osztási maradékát, majd miután az a-t egyenlővé tesszük b-vel és b-t egyenlővé tesszük a kapott maradékkal, megismételjük a fenti számítási sorozatot addig, amíg 0-ás maradékot nem kapunk. A legnagyobb osztó ekkor meghatározható, egyenlő lesz az utolsó nem nulla maradékkal.

2.5. algoritmus.
Bemenet: az a és b pozitív egész számok.
Kimenet: az a és b számok legnagyobb közös osztója.

```
euklid := proc(a, b)
    while (b <> 0) do
        r := a mod b;
        a := b;
        b := r;
    end do;
    return a;
end proc;
```

2.7. példa. Határozzuk meg 84 és 35 legnagyobb közös osztóját.

r	a	b	Magyarázat
	84	35	
14	35	14	$84 \equiv 14 \pmod{35}$
7	14	7	$35 \equiv 7 \pmod{14}$
0	**7**	0	$14 \equiv 0 \pmod{7}$

Tehát a megoldás: $gcd(a,\ b) = 7$.

2.2.2. A kiterjesztett euklidészi algoritmus

Az algoritmus az előző részben megadott euklidészi algoritmus kiegészített változata, mert a legnagyobb közös osztó mellett meghatározza azon x és y egész számokat, melyekre fennáll a következő összefüggés:

$$d = a \cdot x + b \cdot y,$$

ahol $d = gcd(a,\ b)$. Az algoritmus az x és y értékeket iteratívan határozza meg a következő összefüggések alapján:

$$\begin{aligned} x_{k+1} &= q_k \cdot x_k + x_{k-1}, \\ y_{k+1} &= q_k \cdot y_k + y_{k-1}, \end{aligned}$$

ahol a kezdeti értékek:

$$\begin{aligned} x_0 &= 1, \quad y_0 = 0, \\ x_1 &= 0, \quad y_1 = 1, \end{aligned}$$

és q_k-val jelöltük az aktuális körben az a, b-vel való osztási egészrészét. A keresett x és y értékek azzal az x_k, illetve y_k értékekkel lesznek egyenlőek, amelyeket abban a körben határoztunk meg, amikor a, b-vel való osztási maradéka nulla lett. A meghatározott x és y előjele eltér, a kezdeti x_0 előjele pozitív, az y-é negatív, amit minden egyes körben megcserélünk.

Az eljárás számos kriptorendszer alapját képezi, ugyanakkor felhasználható, ahogy a továbbiakban látni fogjuk, több számelméleti feladat megoldásánál. Leírását lásd a [3] könyvben.

2.6. algoritmus.
Bemenet: az a és b pozitív egész számok.
Kimenet: az a és b számok legnagyobb közös osztója: d és az x és y számok, a következő tulajdonságokkal: $d = a \cdot x + b \cdot y$.

```
exteuklid := proc(a, b)
    x0 := 1;
    x1 := 0;
    y0 := 0;
    y1 := 1;
    sign := 1;
    while (b <> 0) do
```

```
        r := a mod b;
        q := a div b;
        a := b;
        b := r;
        xx := x1; yy := y1;
        x1 := q * x1 + x0;
        y1 := q * y1 + y0;
        x0 := xx; y0 := yy;
        sign := -sign;
    end do;
    x0 := sign * x0;
    y0 := -sign * y0;
    return (a, x0, y0);
end proc;
```

2.8. példa. Határozzuk meg 84 és 35 legnagyobb közös osztóját, majd azokat az x és y egész számokat, melyekre fennáll a következő összefüggés: $84 \cdot x + 35 \cdot y = d$, ahol $d = gcd(84,\ 35)$.

r	q	a	b	x_1	x_0	y_1	y_0	$sign$
		84	35	0	1	1	0	+
14	2	35	14	1	0	2	1	−
7	2	14	7	2	1	5	2	+
0	2	**7**	0	5	**2**	12	**5**	−

Tehát a megoldás: $d = 7,\ x = -2,\ y = 5$, és fennáll a következő összefüggés: $84 \cdot (-2) + 35 \cdot 5 = 7$.

2.2.3. Multiplikatív inverz algoritmus

Az algoritmus meghatározza egy adott a egész szám $\pmod n$ szerinti inverzét, azaz azon x egész számot, melyre fennáll a következő lineáris kongruencia:

$$a \cdot x \equiv 1 \pmod n.$$

A kongruencia megoldhatósági feltétele az, hogy $gcd(a,\ n) = 1$. Az ezzel a tulajdonsággal rendelkező számot multiplikatív inverznek hívják, erről részletes matematikai leírást találunk a [11] könyvben. Mivel a kongruencia átírható

$$a \cdot x + n \cdot y = 1$$

alakba, ezért az eljárás a kiterjesztett euklidészi algoritmus (lásd 2.6. algoritmus) alapján meghatározza azon x és y egész számokat, melyekre $a \cdot x + n \cdot y = d$. Ha a $d \neq 1$, akkor ez azt jelenti, hogy nincs multiplikatív inverz, ellenkező esetben az x lesz a keresett inverz.

2.7. algoritmus.

Bemenet: az a és n pozitív egész számok.
Kimenet: az x szám a következő tulajdonsággal $a \cdot x \equiv 1 \pmod{n}$.

```
inverz := proc(a, n)
    (d, x, y) := exteuklid (a, n);
    if (d = 1) then return x;
    else return 0;
    end if;
end proc;
```

2.9. példa. Határozzuk meg $12 \cdot x \equiv 1 \pmod{5}$ kongruencia esetében az x értékét.

Az algoritmus a kiterjesztett euklidészi algoritmussal (lásd 2.6. algoritmus) meghatározza $d = 1,\ x = -2,\ y = 5$ értékeket. Tehát a megoldás $x = -2$, ahol $-2 \equiv 3 \pmod{5}$, azaz $12 \cdot 3 \equiv 26 \equiv 1 \pmod{5}$.

2.10. példa. Határozzuk meg $17 \cdot x \equiv 1 \pmod{27}$ kongruencia esetében az x értékét.

Az algoritmus a kiterjesztett euklidészi algoritmussal (lásd 2.6. algoritmus) meghatározza $d = 1,\ x = 8,\ y = -5$ értékeket. Tehát a megoldás $x = 8$.

2.2.4. Lineáris kongruencia algoritmus

Az algoritmus meghatározza azon x egész számot, melyre fennáll a következő lineáris kongruencia:

$$a \cdot x \equiv b \pmod{n}.$$

A kongruencia megoldhatósági feltétele az, hogy $d = gcd(a,\ n)$ osztója legyen b-nek, és ebben az esetben a megoldások száma, melyek n szerint nem kongruensek, d lesz. Ha $gcd(a,\ n) = 1$, akkor a kongruenciának egy megoldása van. Részletes matematikai leírást találunk a [24] könyvben.

Mivel a kongruencia átírható

$$a \cdot x + n \cdot y = b$$

alakba, ezért az eljárás a kiterjesztett euklidészi algoritmussal (lásd 2.6. algoritmus) meghatározza azon x és y egész számokat, melyekre $a \cdot x + n \cdot y = d$. A d értékétől függően a következő eseteket különböztetjük meg:

1. Ha a kapott d nem lesz b-nek osztója, akkor a kongruenciának nincs megoldása.
2. Ha a kapott $d = 1$, akkor a kongruenciának egy megoldása lesz, mégpedig: $b \cdot x \pmod{n}$.
3. Ha az előző két eset nem áll fenn, akkor a kongruenciának d darab megoldása van.

A d megoldás meghatározása végett a következőképpen járunk el:

– végigosztjuk d-vel a kongruenciát, egy új kongruenciát kapva, legyen ez:

$$a_1 \cdot y \equiv b_1 \pmod{n_1},$$

– meghatározzuk a_1 multiplikatív inverzét, legyen ez: a_1^{-1},
– az eredeti kongruencia megoldásait $x_0,\ x_1,\ \ldots,\ x_{d-1}$-el jelölve, a következőképpen határozzuk meg:

$$\begin{aligned} x_0 &\equiv (b_1 \cdot a_1^{-1}) \pmod{n1}, \\ x_1 &\equiv (x_0 + 1 \cdot n_1) \pmod{n}, \\ &\vdots \\ x_{d-1} &\equiv (x_0 + (d-1)) \cdot n_1 \pmod{n}. \end{aligned}$$

2.8. algoritmus.
Bemenet: az a, b és n pozitív egész számok.
Kimenet: az x szám a következő tulajdonsággal $a \cdot x \equiv b \pmod{n}$.

```
kongruencia := proc(a, b, n)
    (d, x, y) := exteuklid (a, n);
    if (d = 1) then return (b * x) mod n;
    end if;
    if (b mod d <> 0) then return 0;
    else
        a := a div d;
        b := b div d;
        n := n div d;
        a1 := inverz (a, n);
        if (a1 <> 0) then
            k := [];
            k := k  + [(b * a1) mod n];
            for i from 1 to (d-1) do
                k := k + [k[1] + n * i];
            end do:
            return k;
        else return 0;
        end if;
    end if;
end proc;
```

2.11. példa. Határozzuk meg $84 \cdot x \equiv 3 \pmod{35}$ kongruencia esetében az x értékét.

Az eljárás a kiterjesztett euklidészi algoritmussal előbb meghatározza azon $d,\ x,\ y$ értékeket, melyekre fennáll:

$$84 \cdot x + 35 \cdot y = d.$$

Ennek a megoldása: $d = 7,\ x = -2,\ y = 5$. Mivel a $b = 3$ nem osztható $d = 7$-tel, a kongruenciának nincs megoldása, az algoritmus visszatérítési értéke 0.

2.12. példa. Határozzuk meg $17 \cdot x \equiv 25 \pmod{27}$ kongruencia esetében az x értékét.

Az eljárás a kiterjesztett euklidészi algoritmussal előbb meghatározza azon $d,\ x,\ y$ értékeket, melyekre fennáll:

$$17 \cdot x + 27 \cdot y = d.$$

Ennek a megoldása: $d = 1,\ x = 8,\ y = -5$. Az algoritmus tehát $x \equiv 25 \cdot 8 \equiv 200 \equiv 11 \pmod{27}$ visszatérítési értékkel leáll.

2.13. példa. Határozzuk meg a $60 \cdot x \equiv 16 \pmod{64}$ kongruencia esetében az x értékét.

A kiterjesztett euklidészi algoritmus eredményeként a

$$60 \cdot x + 64 \cdot y = d$$

egyenlet megoldása: $d = 4,\ x = -1,\ y = 1$.

A kongruenciának ebben az esetben több megoldása is lesz, szám szerint $d = 4$. Ezeknek a meghatározása végett végigosztjuk a kongruenciát $d = 4$-gyel, majd megoldjuk a $15 \cdot x \equiv 1 \pmod{16}$ kongruenciát. Előbb tehát meghatározzuk 15 multiplikatív inverzét $\pmod{16}$ szerint, mely -1 lesz.

A kongruencia megoldásait pedig a következő egyenletek adják:

$$\begin{aligned} &-1 \cdot 4 \equiv 12 \pmod{16},\\ &12 + 1 \cdot 16 \equiv 28 \pmod{64},\\ &12 + 2 \cdot 16 \equiv 44 \pmod{64},\\ &12 + 3 \cdot 16 \equiv 60 \pmod{64}. \end{aligned}$$

Azaz a megoldások $12,\ 28,\ 44,\ 60$ lesznek.

2.3. A kínai maradéktétel

A kínai maradéktétel algoritmusának eredeti változata a harmadik századból ered, egy kínai matematikustól, leírását számos számelméleti könyvben megtaláljuk, többek között lásd a [24] könyvet.

Az algoritmus az $a_1,\ a_2,\ \ldots,\ a_n$ tetszőleges egész számok esetében meghatározza azon x egész számot, mely eleget tesz a következő kongruenciarendszernek. A rendszernek egyetlen megoldása lesz $\pmod{m_1 \cdot m_2 \cdot \ldots \cdot m_n}$ szerint:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n}, \end{aligned}$$

ahol $m_1,\ m_2,\ \ldots,\ m_n > 0$ páronként relatív prímek.

2.4. megjegyzés. A fenti kongruenciarendszer a megadott feltételek mellett bármilyen $a_1,\ a_2,\ \ldots,\ a_n$ egész számok esetén megoldható.

2.9. algoritmus.
Bemenet: egy n pozitív egész szám, az $a = [a_1,\ a_2,\ \ldots,\ a_n]$, illetve az $m = [m_1,\ m_2,\ \ldots,\ m_n]$ pozitív egész számokat tartalmazó listák.
Kimenet: az x szám, mely megoldása a fenti kongruenciarendszernek.

```
kinaimt := proc(a, m, n)
    modulus := 1;
    for i from 1 to n do modulus := modulus * m[i];
    end do;
    eredmeny := 0;
    for i from 1 to n do
        M := modulus div m[i];
        inv := inverz(M, m[i]);
        eredmeny := (eredmeny + inv * M * a[i]) mod modulus;
    end do;
    return eredmeny;
end proc;
```

2.14. példa. Határozzuk meg azon x egész számot, mely eleget tesz a következő kongruenciarendszernek:

$$\begin{aligned} x &\equiv 1 \pmod{3} \\ x &\equiv 4 \pmod{5} \\ x &\equiv 2 \pmod{11} \\ x &\equiv 3 \pmod{13}. \end{aligned}$$

Mivel 3, 5, 11, 13 páronként relatív prímszámok, a kongruenciának egy megoldása van $\pmod{3 \cdot 5 \cdot 11 \cdot 13} \equiv \pmod{2145}$ szerint. Az algoritmus sorra meghatározza

$$\begin{aligned} M_1 &= 2145/3 = 715, \\ M_2 &= 2145/5 = 429, \\ M_3 &= 2145/11 = 195, \\ M_4 &= 2145/13 = 165 \end{aligned}$$

értékeket, majd

$$\begin{array}{llll} inv_1 = 1, & \text{ahol} & 715 \cdot inv_1 \equiv 1 & (\text{mod } 3), \\ inv_2 = -1, & \text{ahol} & 429 \cdot inv_2 \equiv 1 & (\text{mod } 5), \\ inv_3 = -4, & \text{ahol} & 195 \cdot inv_3 \equiv 1 & (\text{mod } 11), \\ inv_4 = 3, & \text{ahol} & 165 \cdot inv_4 \equiv 1 & (\text{mod } 13). \end{array}$$

Tehát a megoldás:
$x \equiv 1 \cdot 715 \cdot 1 + (-1) \cdot 429 \cdot 4 + (-4) \cdot 195 \cdot 2 + 3 \cdot 165 \cdot 3 \equiv 1069 \pmod{2145}$.

2.4. Moduláris hatványozás

A számítástechnika különböző területein gyors eljárásra van szükség ahhoz, hogy meghatározzuk az

$$a^e \pmod m$$

értéket, ahol a, e nem negatív egész számok, n pedig egy pozitív egész szám.

A bemutatásra kerülő algoritmus ezt a hatványértéket a következő elgondolás alapján határozza meg: egy while ciklus keretén belül csökkentjük a hatványkitevő értékét, mely egyenlő lesz a régi, 2-vel való osztási egészrészével, mindaddig, amíg ez nagyobb mint 0. Ugyancsak a ciklus keretén belül, ha a hatványkitevő páratlan, meghatározzuk az

$$r_k = a_{k-1} \cdot r_{k-1}$$

szorzatot, majd a kitevő értékétől függetlenül az

$$a_k = a_{k-1} \cdot a_{k-1}$$

négyzetre emelést, ahol r_0 kezdeti értéke 1 és $a_0 = a$. A szorzás és négyzetre emelés eredményeként kapott értékeket minden alkalommal $(\text{mod } m)$ szerint vesszük. Az eljárás leírását megtalálhatjuk az [5] könyvben.

2.10. algoritmus.
Bemenet: az a, e, m pozitív egész számok.
Kimenet: $a^e \pmod m$.

```
modhatv := proc(a, e, m)
    r := 1;
    while (e > 0) do
        if (e mod 2 <> 0) then
            r := (r * a) mod m;
        end if;
        a := (a * a) mod m;
        e := e div 2;
    end do;
return r;
end proc;
```

2.15. példa. Határozzuk meg a $7^{36} \pmod{89}$ értéket.

r	a	e	Magyarázat
1	7	36	
1	49	18	$7 \cdot 7 \equiv 49 \pmod{89}$
1	87	**9**	$49 \cdot 49 \equiv 87 \pmod{89}$
87	4	4	$1 \cdot 87 \equiv 87 \pmod{89}$
			$87 \cdot 87 \equiv 4 \pmod{89}$
87	16	2	$4 \cdot 4 \equiv 16 \pmod{89}$
87	78	**1**	$16 \cdot 16 \equiv 78 \pmod{89}$
22	32	0	$87 \cdot 78 \equiv 22 \pmod{89}$
			$78 \cdot 78 \equiv 32 \pmod{89}$

Tehát az eredmény 22.

2.16. példa. Határozzuk meg az $5^{9} \pmod{61}$ értéket.

r	a	e	Magyarázat
1	5	**9**	
5	25	4	$1 \cdot 5 \equiv 5 \pmod{61}$
			$5 \cdot 5 \equiv 25 \pmod{61}$
5	15	2	$25 \cdot 25 \equiv 15 \pmod{61}$
5	42	**1**	$15 \cdot 15 \equiv 42 \pmod{61}$
27	56	0	$5 \cdot 42 \equiv 27 \pmod{61}$
			$42 \cdot 42 \equiv 56 \pmod{61}$

Tehát az eredmény 27.

2.5. Lineáris kongruencián alapuló véletlenszám-generálás

Véletlenszerűen generált számokra a számítástechnika számos területén szükség van, többek között a kriptográfiában is. Éppen ezért a legtöbb programozási nyelvnek megvan a saját beépített véletlenszám-generáló algoritmusa. Az egyik legrégibb és legegyszerűbb álvéletlenszámokat generáló algoritmus lineáris kongruenciát használ.

A bemutatásra kerülő algoritmus d darab számot generál véletlenszerűen, a k, a, c, m konstansok, előre megadott értékei alapján, részletes leírást a [14] könyvben találunk. Az algoritmusban használt k, a, c, m konstans értékek szokásos megnevezése: k kezdeti érték, a szorzó, c növelő és m modulusz.

A véletlenszám-generáló algoritmusok minőségi tényezője, hogy a generált számok ismétlődése minél később következzen be, azaz a generált számsor periódusa minél nagyobb legyen. A lineáris kongruencián alapuló véletlenszám-generáló algoritmus minősége a fent értelmezett konstansok

megválasztásától függ. A kriptográfiában ennél az eljárásnál sokkal biztonságosabb algoritmusokat alkalmaznak, de mivel implementálása egyszerű és gyors, pedagógiai szempontból ennek a leírását adjuk meg.

D. H. Lehmer 1951-ben az a, c, m konstansokra a következő értéket adta meg, lásd [17]:

$$\begin{aligned} a &= 23, \\ c &= 0, \\ m &= 10^{81} + 1. \end{aligned}$$

1988-ban a Park és Miller által javasolt konstansok értéke, melyeket az IBM360 számítógépcsalád is alkalmaz, a következőek:

$$\begin{aligned} a &= 16\,807, \\ c &= 0, \\ m &= 2^{31} - 1. \end{aligned}$$

Az alábbi algoritmus a paraméterként megadott k, a, c, m értékek alapján, felhasználva az

$$x_n \equiv (x_{n-1} \cdot a + c) \pmod{m}$$

lineáris kongruenciát, előállít egy d elemű listát, ahol $x_0 = k$. A lista elemei közül bármelyiket felhasználhatjuk mint véletlenül előállított értéket. A k értékét igazíthatjuk a számítógép órájához.

2.11. algoritmus.
Bemenet: a d, k, a, c, m pozitív egész számok.
Kimenet: a d elemszámú *lista*.

```
veletlen := proc(d, k, a, c, m)
    lista := [];
    x := k;
    for i from 1 to d do
        lista := lista + [x];
        x := (x * a + c) mod m;
    end do;
    return lista;
end proc;
```

A gyakorlatban az algoritmusnak számos gyenge pontja van, többek között:

- Ha ismertek az a, c, m értékek, és ismert továbbá egyetlen véletlenszerűen generált szám, akkor bárki meghatározhatja az ezután következő bármelyik másik számot.
- Ha nem ismertek az a, c, m értékek, viszont ismert négy véletlenszerűen generált szám, legyenek ezek x_0, x_1, x_2, x_3, akkor nincs más

feladat, mint a következő rendszert $a,\ c,\ m$ ismeretlenek függvényében megoldani:

$$\begin{aligned} x_1 &\equiv (a \cdot x_0 + c) \pmod{m_1} \\ x_2 &\equiv (a \cdot x_1 + c) \pmod{m_2} \\ x_3 &\equiv (a \cdot x_2 + c) \pmod{m_3}. \end{aligned}$$

2.6. Teljes hatvány ellenőrzése

A leírásra kerülő algoritmus egy n pozitív egész szám esetében megvizsgálja, hogy a szám teljes hatvány-e vagy sem, azaz felírható-e a következő alakba $n = x^a$, tetszőleges $x,\ a$ pozitív számok esetében. Számos algoritmus létezik, különböző komplexitással, arra vonatkozóan, hogy egy számról megállapítsuk, teljes hatvány-e vagy sem. Ehhez kapcsolódó részletes leírás a [4]-ben található.

A leírásra kerülő algoritmus minden $p \leq \log_2(n)$ számra, ahol p prímszám, meghatározza az $n^{1/p}$ egy közelítő értékét. Ez utóbbi értéket a *gyok* algoritmus a következő iterációs képlet alapján számolja ki, mely a Newton-féle numerikus módszer egy sajátos esete:

$$\begin{aligned} x_0 &= \sqrt{n} \\ x_{k+1} &= x_k - \tfrac{x_k^p - n}{p \cdot x_k^{p-1}}. \end{aligned}$$

Először tehát a *gyok* algoritmus pszeudokódja következik, ahol meghívásra kerül a *ceil* a felső egészrész könyvtárfüggvény, és a *hatv* függvény, amelyik a *modhatv* függvényhez hasonló módon (lásd 2.10. algoritmus) meghatározza x^p értékét anélkül, hogy a szorzatok és négyzetre emelések során kapott értékeknek meghatározná az osztási maradékát, valamely egész szám szerint. A *hatv* függvény implementálását az olvasóra bízzuk.

2.12. algoritmus.
Bemenet: egy $n \geq 2$ egész szám és egy p prímszám.
Kimenet: meghatározza azt az x egész számot, melyre $x^p \leq n \leq (x+1)^p$.

```
gyok := proc(n, p)
    x := n;
    f := hatv(x, p) - n;
    while (0 < f) do
        x := ceil(x - f/(p * hatv(x, p-1)));
        f := hatv(x, p)- n;
    end do;
    return x;
end proc;
```

2.17. példa. Határozzuk meg a *gyok* algoritmus által meghatározott értékeket, $n = 243$ és $p = 3$ esetében.

x	f
15	3132
10	757
7	100
6	-27

Az algoritmus $x = 6$-ot ad eredményül, azaz $6^3 = 216 \leq 243 \leq 7^3 = 343$.

A *hatvany* algoritmusban meghívásra kerül a *ceil*, felső egészrészt meghatározó könyvtárfüggvény, továbbá a *prim* függvény, mely megvizsgálja a paraméterként megadott számról, hogy prím-e. Ez utóbbit többféle algoritmussal is megvizsgálhatjuk, a következő fejezetben erre számos eljárást mutatunk.

2.13. algoritmus.
Bemenet: Egy $n \geq 2$ egész szám.
Kimenet: 1, ha az n szám teljes hatvány, 0, ha nem az.

```
hatvany := proc(n)
    for i from 2 to ceil(log[2](n)) do
    if (prim(i) = 1) then
        h := gyok(n, i);
        if (hatv(h, i) = n) then return 1;
        end if;
    end if;
    end do;
    return 0;
end proc;
```

2.18. példa. Vizsgáljuk meg, hogy 243 teljes hatvány-e. Az algoritmus által meghatározott értékek a következő táblázatból olvashatóak le.

i	h	h^i
2	15	225
3	6	216
5	3	243

Az algoritmus visszatérítési értéke 1, ami azt jelzi, hogy 243 teljes hatvány.

2.19. példa. Vizsgáljuk meg, hogy 1215 teljes hatvány-e. Az algoritmus által meghatározott értékek a következő táblázatból olvashatóak le.

i	h	h^i
2	34	1156
3	10	1000
5	4	1024
7	2	128

Az algoritmus az összes lehetséges i-re elvégzi a megfelelő számítási sorozatot, egy értékre sem találja igaznak a $hatv(h,\ i) = n$ egyenlőséget, tehát visszatérítési értéke 0, ami azt jelzi, hogy $1215 = 3^5 \cdot 5$ nem teljes hatvány.

2.7. Prímtesztek és nagy prímek generálása

Prímteszt alatt olyan eljárást értünk, mely során egy n pozitív egész számról el tudjuk dönteni, hogy prímszám-e. Fontos megjegyezni, hogy míg erre a problémára léteznek hatékony algoritmusok, addig egy szám prímtényezőinek a meghatározása „nehéz” feladat, nem ismert rá hatékony algoritmus. Ez utóbbit a prímfaktorizációs algoritmusok végzik. Algoritmusok bonyolultságára vonatkozó leírást, mivel jelen jegyzet keretén belül erre nem térünk ki, a [19] könyvben találunk.

A gyakorlatban alkalmazott prímteszt algoritmusok legtöbbje valószínűségi (probabilisztikus) prímteszt, ami alatt azt értjük, hogy az n vizsgálandó számról, akkor jelentjük ki egy bizonyos valószínűséggel hogy prím, ha az többször is átmegy a kiválasztott tesztelési eljáráson.

2.7.1. Osztási próba

Ha egy n pozitív egész számról el szeretnénk dönteni, hogy prímszám-e vagy összetett, akkor a legegyszerűbb algoritmus, ami szerint ezt megoldhatjuk, a következő kijelentésen alapszik: ha n egy pozitív összetett szám, akkor n legkisebb prímosztója nem lehet nagyobb $\sqrt{n}$-nél, lásd [2]. Azt az eljárást, mely $\sqrt{n}$-ig sorra megvizsgálja az összes páratlan számmal való oszthatóságot, osztási próbának (trial division) nevezik.

Az algoritmust a gyakorlatban 10^6-nál nagyobb számok prímtesztelésére már nem szokták használni, mert nem hatékony.

2.14. algoritmus.
Bemenet: egy n pozitív egész szám.
Kimenet: 1, ha az n szám prím, 0, ha összetett.

```
prim := proc(n)
    if (n = 2) then return 1;
    end if;
```

```
    if (n mod 2 = 0) then return 0;
    end if;
    t := 3; s := 4;
    while (s <= n) do
        if (n mod t = 0) then return 0;
        else
            t := t + 2;
            s := s + 2 * t - 1;
            print(t, s);
        end if;
    end do;
    return 1;
end proc;
```

2.7.2. Fermat-teszt

A Fermat-teszt, melynek leírását megtaláljuk a [20] könyvben, egy valószínűségi prímteszt, mely a kis Fermat-tételen alapszik, mely szerint:

2.1. tétel. Ha az n szám prím, akkor minden olyan a szám esetében, ahol $1 \leq a \leq (n-1)$, fennáll a következő összefüggés: $a^{n-1} \equiv 1 \pmod{n}$.

A fentiek értelmében, ha egy n számra nem áll fenn a fenti összefüggés, akkor az n egyértelműen összetett. Abban az esetben viszont, mikor az összefüggés fennáll, nem lehetünk biztosak abban, hogy a szám prím, mert az összefüggés néha összetett számokra is fennáll.

2.1. értelmezés. Az olyan összetett n számokat, melyek esetében létezik olyan a, $1 \leq a \leq (n-1)$ szám, melyre fennáll az $a^{n-1} \equiv 1 \pmod{n}$ összefüggés, a alapú Fermat-féle álprímeknek hívjuk.

Sőt mi több, vannak olyan n összetett számok, melyek esetében az összes a, $1 \leq a \leq (n-1)$, ahol $gcd(a,\ n) = 1$ számra fennáll az összefüggés, azaz $a^{n-1} \equiv 1 \pmod{n}$. Ezeket Carmichael-számoknak hívjuk, számuk végtelen, viszont nagyon „ritkák” az egész számok között, pontosabban 105 212 Carmichael-szám van, mely $\leq 10^{15}$.

2.20. példa. Egy 2-es alapú Fermat-féle álprím a $341 = 11 \cdot 31$, mert $2^{340} \equiv 1 \pmod{341}$.

2.21. példa. A legkisebb Carmichael-szám az $561 = 3 \cdot 11 \cdot 17$.

A bemutatásra kerülő algoritmus t darab véletlenszerűen generált szám esetében vizsgálja, hogy fennáll-e a kis Fermat-tétel. Ha mind a t esetben azt

állapítja meg, hogy fennáll a tételbeli kongruencia, akkor azzal a visszatérítési értékkel áll le, hogy a szám prím, ha egyetlen egy esetet is talál, mikor nem áll fenn a kongruencia, akkor kijelenti, hogy a szám nem prím. Tehát ha a vizsgálandó számról azt mondja, hogy prím, azt csak egy bizonyos valószínűséggel állítja, míg az összetettséget teljes bizonyossággal meg tudja állapítani. Mivel az algoritmus a Carmichael-számok esetében tévedhet, nem annyira gyakorlati, mint inkább elméleti szempontból jelentős.

2.5. megjegyzés. Az algoritmus keretében meghívásra kerül a már korábban bemutatott *modhatv* algoritmus, lásd 2.10. algoritmus.

2.15. algoritmus.
Bemenet: egy $n \geq 3$ páratlan szám és egy $t \geq 1$ biztonsági paraméter.
Kimenet: 1, ha az n szám prím, 0, ha összetett.

```
fermat := proc(n, t)
    for i from 1 to t do
        a := rand(2, (n-2));
        r := modhatv(a, n-1, n);
        if (r <> 1) then return 0;
        end if;
    end do;
return 1;
end proc;
```

2.22. példa. Vizsgáljuk meg a Fermat-teszt algoritmussal, hogy $n = 97$ prímszám-e, $t = 5$ véletlenszerűen generált a értékre. Az algoritmus által meghatározott értékek az alábbi táblázatból olvashatóak le.

a	r	i	Magyarázat
41	1	1	$41^{96} \equiv 1 \pmod{97}$
32	1	2	$32^{96} \equiv 1 \pmod{97}$
94	1	3	$94^{96} \equiv 1 \pmod{97}$
17	1	4	$17^{96} \equiv 1 \pmod{97}$
73	1	5	$73^{96} \equiv 1 \pmod{97}$

Az algoritmus az összes véletlenszerűen generált a érték esetében igaznak találja az $a^{n-1} \equiv 1 \pmod{n}$ kifejezést, ami azt jelzi, hogy a 97 prímszám, tehát 1 visszatérítési értékkel leáll.

2.23. példa. Vizsgáljuk meg a Fermat-teszt algoritmussal, hogy $n = 143 = 11 \cdot 13$ prímszám-e, $t = 5$ véletlenszerűen generált a érték esetében. Az algoritmus által meghatározott értékek az alábbi táblázatból olvashatóak le.

a	r	i	Magyarázat
124	108	1	$124^{142} \equiv 108 \pmod{143}$

Az algoritmus már az első véletlenszerűen generált a érték esetében nem találja igaznak az $a^{n-1} \equiv 1 \pmod{n}$ kifejezést, ami azt jelzi, hogy a 143 összetett szám, tehát 0 visszatérítési értékkel leáll.

2.7.3. Solovay–Strassen-prímteszt

A Solovay–Strassen-prímteszt egy valószínűségi prímteszt, napjainkban azonban kevesebb a gyakorlati haszna. Leginkább az RSA kriptorendszer gyakorlati alkalmazhatóságának bizonyításánál használták fel. Elsősorban azért nem alkalmazzák a gyakorlatban, mert nem annyira hatékony, másodsorban mert tévedési aránya is nagyobb, mint más ilyen jellegű algoritmusoké.

Az algoritmus ismertetése előtt definiáljuk a szükséges matematikai fogalmakat, részletesebb ismertetésüket lásd a [11] könyvben:

2.2. értelmezés. A Legendre-szimbólumot, melyet $\left(\frac{a}{n}\right)$-el jelölünk, tetszőleges a egész és p páratlan prímszám esetében, a következőképpen értelmezzük:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{ha } a \equiv 0 \pmod{p} \\ 1, & \text{ha } a \not\equiv 0 \pmod{p} \text{ és } \exists\, x, \text{ melyre } a \equiv x^2 \pmod{p} \\ -1, & \text{ha } a \not\equiv 0 \pmod{p} \text{ és } \nexists \text{ ilyen } x \end{cases}$$

Ha $\left(\frac{a}{p}\right) = 1$, akkor a-t kvadratikus maradéknak hívjuk $\pmod{p}$ szerint.

2.3. értelmezés. A Jacobi-szimbólumot, mely a Legendre-szimbólum általánosítása, hasonló módon jelöljük és a következőképpen definiáljuk: legyen $3 \leq n$ tetszőleges egész szám, ahol ha n prímtényezős felbontása:

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}, \text{ akkor}$$

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \left(\frac{a}{p_2}\right)^{e_2} \cdot \ldots \cdot \left(\frac{a}{p_k}\right)^{e_k}.$$

2.4. értelmezés. Az Euler-féle kritérium, szerint ha az n szám prím, akkor minden olyan a szám esetében, ahol $gcd(a,\ n) = 1$, fennáll a következő összefüggés:

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}.$$

2.5. értelmezés. Az Euler phi-függvény, melyet az $1 \leq n$ szám esetében értelmezünk, $\phi(n)$-nel jelölünk, megadja az $\{1,\ \ldots,\ n\}$ halmazból azon számok számát, melyek relatív prímek n-nel.

A Solovay–Strassen-algoritmus az Euler-féle kritériumon alapszik, melyet azonban csak bizonyos körülmények között használhatunk prímtesztelésre. Vannak ugyanis olyan összetett számok, melyekre létezik olyan a szám, melyre $gcd(a,\ n) = 1$ és

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}.$$

Az ilyen tulajdonsággal rendelkező számokat a alapú Euler-féle álprímeknek nevezzük.

2.24. példa. Egy 47-es alapú Euler-féle álprím a $221 = 13 \cdot 17$, mert

$$47^{110} \equiv -1 \pmod{221},$$

$$\left(\frac{47}{221}\right) \equiv -1 \pmod{221}.$$

Hogy mégis használható a kritérium prímtesztként, az a következő állításból derül ki: az n összetett szám esetében maximum $\phi(n)/2$, ahol ϕ az Euler phi-függvény, azon a számoknak a száma, ahol $1 \leq a \leq (n-1)$, melyek Euler-féle álprímek, tehát jó esélye van annak, hogy ha több véletlenszerűen generált a számra fennáll az Euler-féle kritérium, akkor az n prímszám legyen.

A Solovay–Strassen-algoritmus t darab véletlenszerűen generált szám esetében vizsgálja, hogy fennáll-e az Euler-féle kritérium. Ha mind a t esetben igaznak találja a kritériumot, akkor megállapítja a számról, hogy prím, ha csak egy esetben is nem áll fenn a kritérium, akkor a számról kijelenti, hogy nem prím. Tehát csak egy bizonyos valószínűséggel állapítja meg a vizsgálandó számról, hogy prím.

A Solovay–Strassen-algoritmusban az Euler-féle kritérium megvizsgálásához szükség van a Jacobi-szimbólum meghatározására, ezért előbb az $\left(\frac{a}{n}\right)$ meghatározásához szükséges algoritmust mutatjuk be. Ha az n prím, akkor az algoritmus a Legendre-szimbólumot határozza meg. Az algoritmus rekurzív és alap számelméleti összefüggéseken alapszik, melyek a következők (lásd [11]):

$$\left(\frac{0}{n}\right) = \begin{cases} 1, & \text{ha } n = 1 \\ 0, & \text{ha } n \neq 1 \end{cases}$$

$$\left(\frac{2}{n}\right) = \begin{cases} 1, & \text{ha } n \equiv 1 \pmod 8 \\ 1, & \text{ha } n \equiv 7 \pmod 8 \\ -1, & \text{ha } n \equiv 3 \pmod 8 \\ -1, & \text{ha } n \equiv 5 \pmod 8 \end{cases}$$

$$\left(\frac{a}{n}\right) = \left\{ \left(\frac{a \bmod n}{n}\right), \quad \text{ha } a \geq n \right.$$

$$\left(\frac{a}{n}\right) = \left\{ \left(\frac{2}{n}\right) \cdot \left(\frac{a \operatorname{div} 2}{n}\right), \quad \text{ha } a \equiv 0 \pmod 2 \right.$$

$$\left(\frac{a}{n}\right) = \begin{cases} -1 \cdot \left(\frac{n}{a}\right), & \text{ha } a \equiv 3 \pmod 4 \text{ és } n \equiv 3 \pmod 4 \\ \left(\frac{n}{a}\right), & \text{másképp} \end{cases}$$

2.16. algoritmus.
Bemenet: az $3 < n$ és $0 < a$ pozitív egész számok.
Kimenet: az $\left(\frac{a}{n}\right)$ értéke.

```
jacobi := proc(a, n)
    if (a = 0) then
        if (n = 1) then return 1;
        else return 0;
        end if;
    end if;
    if (a = 2) then
        if (n mod 8 = 1) then return 1; end if;
        if (n mod 8 = 7) then return 1; end if;
        if (n mod 8 = 3) then return -1; end if;
        if (n mod 8 = 5) then return -1; end if;
    end if;
    if (a >= n) then
        return jacobi((a mod n), n);
    end if;
    if (a mod 2 = 0) then
        return jacobi(2, n) * jacobi(a/2, n);
    end if;
    if (a mod 4 = 3 and n mod 4 =3) then
        return (-1) * jacobi(n, a);
    else return jacobi(n, a);
    end if;
end proc;
```

2.25. példa. Határozzuk meg $\left(\frac{47}{221}\right)$ értékét.

a	n	$n \mod 8$	$a \mod 4$	$n \mod 4$	Magyarázat
47	221		3	1	
221	47				$a > n$, $221 \equiv 33 \pmod{47}$
33	47		1	3	
47	33				$a > n$ $47 \equiv 14 \pmod{33}$
14	33				$a \equiv 0 \pmod 2$
2	33	1			$a = 2$
7	33		3	1	
33	7				$a > n$ $33 \equiv 5 \pmod 7$
5	7		1	3	
7	5				$a > n$, $7 \equiv 2 \pmod 5$
2	5	5			

További magyarázat:

$$\begin{aligned}\left(\tfrac{47}{221}\right) &= \left(\tfrac{221}{47}\right) = \left(\tfrac{33}{47}\right) = \left(\tfrac{47}{33}\right) = \left(\tfrac{14}{33}\right) = \left(\tfrac{2}{33}\right) \cdot \left(\tfrac{7}{33}\right) = \\ &= 1 \cdot \left(\tfrac{33}{7}\right) = \left(\tfrac{5}{7}\right) = \left(\tfrac{7}{5}\right) = \left(\tfrac{2}{5}\right) = -1\end{aligned}$$

Ezek után következhet a Solovay–Strassen-algoritmus pszeudokódja, melyben a Jacobi-szimbólum meghatározására csak akkor kerül sor, ha $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$, vagy $a^{\frac{n-1}{2}} \equiv (n-1) \pmod{n}$.

2.17. algoritmus.
Bemenet: egy $n \geq 3$ páratlan szám és egy $t \geq 1$ biztonsági paraméter.
Kimenet: 1, ha az n szám prím, 0, ha összetett.

```
solovay_strassen := proc(n, t)
    if (n mod 2 = 0) then return 0;
    end if;
    for i from 1 to t do
        a := rand(2, (n-2));
        r := modhatv(a, (n-1)/2, n);
        if (r <> 1 and r <> (n-1)) then return 0;
        end if;
        s := jacobi(a, n);
        if s = -1 then s = n-1;
        end if;
        if ((r mod n) <> s) then return 0;
        end if;
    end do;
    return 1;
end proc;
```

2.6. megjegyzés. Az algoritmus keretében meghívásra kerül a már korábban bemutatott *modhatv* (lásd 2.10. algoritmus).

2.26. példa. Vizsgáljuk meg a Solovay–Strassen-algoritmussal, hogy $n = 61$ prímszám-e, $t = 5$ véletlenszerűen generált a értékre.

a	r	s	Magyarázat
50	60	-1	$60 = -1 \pmod{61}$
21	60	-1	
37	60	-1	
7	60	-1	
49	1	1	

Az algoritmus mind az 5 körben igaznak találja az $r = s \pmod{n}$ egyenlőséget, tehát 1 visszatérítési értékkel leáll, azaz jelzi, hogy a 61 prímszám.

2.27. példa. Vizsgáljuk meg a Solovay–Strassen-algoritmussal, hogy $n = 143 = 11 \cdot 13$ prímszám-e.

a	r	Magyarázat
119	20	$r \neq 1, r \neq 142$

Az algoritmusban nem kerül sor a Jacobi-szimbólum meghatározására, mert $119^{\frac{143-1}{2}} \neq 1$ és $119^{\frac{143-1}{2}} \neq 142 \pmod{143}$, a visszatérítési érték tehát 0 lesz, azaz a szám nem prím.

2.7.4. Miller–Rabin-prímteszt

A Miller–Rabin-teszt az előzőhöz hasonlóan egy valószínűségi prímteszt, azaz egy olyan összefüggésen alapszik, mely prímszámok esetében mindig igaz, leírását lásd a [20] könyvben.

Az algoritmus a következő kritériumra épül: ha az n szám prím, akkor minden olyan a szám esetében, ahol $gcd(a,\ n) = 1$, és $n - 1 = 2^s r$, ahol r páratlan szám, fennáll a következő két összefüggés közül valamelyik:

1. $a^r \equiv 1 \pmod{n}$, vagy
2. létezik olyan j, $0 \leq j \leq s - 1$, úgy hogy $a^{2^j r} \equiv n - 1 \pmod{n}$.

A fenti kritériumot azonban nem használhatjuk, csak bizonyos körülmények között prímtesztelésre, vannak ugyanis olyan n összetett számok, melyekre létezik olyan a szám, ahol $1 \leq a \leq n - 1$, melyre fennáll a fenti két összefüggés közül valamelyik. Az ilyen tulajdonsággal rendelkező n számokat a alapú álprímeknek hívják, az a számokat pedig erős hazugoknak (strong liar).

2.28. példa. Egy 70-es alapú álprím a $169 = 13 \cdot 13$, mert $169 - 1 = 168 = 2^3 \cdot 21$, ahonnan $s = 3,\ r = 21$, és $70^{2 \cdot 21} \equiv 168 \pmod{169}$.

A Miller–Rabin-algoritmus a gyakorlatban mégis nagyon jól alkalmazható, mégpedig azért, mert egy n összetett szám esetében az a erős hazugok száma, ahol $1 \leq a \leq n - 1$ és $a \neq 9$ legtöbb $\phi(n)/4$, ahol ϕ az Euler phi-függvényt jelöli (lásd 2.5. értelmezés). Tehát jó esélye van annak, hogy ha több véletlenszerűen generált a számra fennáll a kritérium, akkor az n prímszám legyen.

2.29. példa. 169-hez tartozó erős hazugok halmaza a következő: { 1, 19, 22, 23, 70, 80, 89, 99, 146, 147, 150, 168 }. A halmaz elemszáma 12, ahol $12 < \phi(169)/4 = (12 \cdot 13)/4 = 39$.

A Miller–Rabin-algoritmus csak akkor jelenti ki az n számról, hogy prím, ha mind a t darab véletlenszerűen generált a számra teljesül az

$$a^r \equiv 1 \pmod{n}$$

vagy

$$a^{2^j r} \equiv n-1 \pmod{n}, \text{ ahol } 0 \leq j \leq s-1$$

kongruenciák valamelyike. Tehát csak egy bizonyos valószínűséggel állapítja meg a vizsgálandó számról, hogy prím.

2.7. megjegyzés. Az algoritmus visszatérítési értéke 1, ha a szám prím, ellenkező esetben 0. Az algoritmus keretében meghívásra kerül a már korábban bemutatott *modhatv* (lásd 2.10. algoritmus).

2.18. algoritmus.
Bemenet: egy $n \geq 3$ páratlan szám és egy $t \geq 1$ biztonsági paraméter.
Kimenet: 1, ha az n szám prím, 0, ha összetett.

```
miller_rabin := proc(n, t)
    s := 0;
    r := n-1;
    while (r mod 2 = 0) do
        r := r div 2;
        s := s+1;
    end do;
    for j from 1 to t do
        a := rand(2, (n-2))();
        y := modhatv(a, r, n);
        if (y <> 1 and y <> (n-1)) then
            i := 1;
            while (i < s and y <> (n-1)) do
                y := (y * y) mod n;
                if (y = 1) then return 0;
                end if;
                i := i + 1;
            end do;
            if (y <> (n-1)) then return 0;
            end if;
        end if;
    end do;
    return 1;
end proc;
```

2.30. példa. Vizsgáljuk meg a Miller–Rabin-algoritmussal, hogy $n = 97$ prímszám-e, $t = 3$ véletlenszerűen generált a értékre.

Az algoritmus előbb meghatározza s és r értékeit: $s = 5$, $r = 3$, mert $96 = 2^5 \cdot 3$. Az algoritmus által meghatározott további értékek az alábbi

táblázatból olvashatóak le.

a	y	i	Magyarázat
12	79	0	$12^3 \equiv 79 \pmod{97}$
	33	1	$12^{2\cdot 3} \equiv 33 \pmod{97}$
	22	2	$12^{2^2\cdot 3} \equiv 22 \pmod{97}$
	96	3	$12^{2^3\cdot 3} \equiv 96 \pmod{97}$
51	52	0	$51^3 \equiv 52 \pmod{97}$
	85	1	$51^{2\cdot 3} \equiv 85 \pmod{97}$
	47	2	$51^{2^2\cdot 3} \equiv 47 \pmod{97}$
	75	3	$51^{2^3\cdot 3} \equiv 75 \pmod{97}$
	96	4	$51^{2^4\cdot 3} \equiv 96 \pmod{97}$
72	89	0	$72^3 \equiv 89 \pmod{97}$
	64	1	$72^{2\cdot 3} \equiv 64 \pmod{97}$
	22	2	$72^{2^2\cdot 3} \equiv 22 \pmod{97}$
	96	3	$72^{2^3\cdot 3} \equiv 96 \pmod{97}$

Az algoritmus mindhárom véletlenszerűen generált a érték esetében talál olyan i értéket, $i = 3,\ 4,\ 3$ -ra, melyekre igaznak találja az $a^{2^i\cdot r} \equiv 96 \pmod{97}$ kifejezést, ami azt jelzi, hogy a 97 prímszám, tehát 1 visszatérítési értékkel leáll.

2.31. példa. Vizsgáljuk meg a Miller–Rabin-algoritmussal, hogy $n = 169 = 13 \cdot 13$ prímszám-e. Az algoritmus előbb meghatározza s és r értékeit: $s = 3,\ r = 21$, mert $168 = 2^3 \cdot 21$.

a	y	i	Magyarázat
140	142	0	$140^{21} \equiv 142 \pmod{169}$
	53	1	$140^{2\cdot 21} \equiv 53 \pmod{169}$
	105	2	$140^{2^2\cdot 21} \equiv 105 \pmod{169}$

Az algoritmus az első véletlenszerűen generált $a = 140$ értékre sorra kiszámolja $i = 0,\ 1,\ 2$ -re az $a^{2^i\cdot r} \pmod{169}$ értéket. Mivel y egyik esetben sem 1 vagy $n-1$, az algoritmus a *while* ciklus után 0 visszatérítési értékkel leáll, azaz jelzi, hogy a 169 összetett szám.

2.7.5. Az AKS prímteszt

Az algoritmus egy determinisztikus (lásd [19]) prímteszt, melyet 3 indiai matematikus, Manindra Agrawal, Neeraj Kayal és Nitin Saxena 2002-ben, majd 2004-ben újrapublikált (lásd [1]). Jelentősége abban áll, hogy az első olyan eljárás, amelyik determinisztikus, futási ideje polinomiális és nem alapszik semmilyen hipotézisre. Gyakorlatban az algoritmus nem használható, mert futási ideje még így sem megfelelő. A megjelenést követő években számos tudományos dolgozat jelent meg ebben a témában, ezek közül egyik

legjelentősebb a Lenstra és Pomerance 2003-ban (lásd [18]) megjelenő dolgozata, mely az eredeti algoritmus futási idejét nagyban feljavította. Az algoritmus implementálása azóta is számos nyitott kérdést rejt magában.

Az algoritmus egy régóta ismert azonosságra épül, mely szerint az n szám akkor és csakis akkor prím, ha fennáll a következő összefüggés:

$$(x-a)^n \equiv (x^n - a) \pmod{n},$$

ahol a maradékos osztást a polinom együtthatóin kell elvégezni. A fenti összefüggés a kis Fermat-tétel (lásd 2.1. tétel) általánosítása polinomokra. Ennek a tételnek az ellenőrzése azonban exponenciális időigényű.

2.32. példa. Az 5 prím volta a következőképpen ellenőrizhető le:

$$(x-1)^5 \equiv x^5 + 5 \cdot x^4 + 10 \cdot x^3 + 10 \cdot x^2 + 5 \cdot x + 1 \equiv (x^5 - 1) \pmod{5},$$

a polinom minden együtthatójának 5-tel való osztási maradéka 0, kivéve az első és utolsó együtthatókat.

Mielőtt megadnánk az AKS algoritmus alapötletét, definiáljuk egy elem rendjét:

2.6. értelmezés. Valamely r szám esetében azt a legkisebb $t \in \{0,\ 1,\ \ldots,\ r-1\}$ számot, mely teljesíti

$$n^t \equiv 1 \pmod{r}$$

feltételt, az n rendjének hívjuk és $ord_r(n)$-el jelöljük, részletes leírást a [11]-ben találunk.

Ezek után megadhatjuk az AKS algoritmus alaptételét, mely az eredeti, 2002-ben megjelent tétel egy módosított változata.

2.2. tétel. Adott $n \geq 2$ szám esetében legyen r egy pozitív egész, ahol $r < n$ és ahol $ord_r(n) > \log^2 n$. Az n szám akkor és csakis akkor prím, ha teljesülnek az alábbi feltételek:

- n nem teljes hatvány,
- n-nek nincs r-nél kisebb vagy vele egyenlő prímtényezője,
- $(x-a)^n \equiv x^n - a \pmod{x^r - 1,\ n}$, bármely a egész szám esetében, ahol $1 \leq a \leq \sqrt{r} \cdot \log n$.

2.8. megjegyzés. Az $(x-a)^n \equiv x^n - a \pmod{x^r - 1,\ n}$ kongruencia jelentése az, hogy meghatározzuk a polinomok $x^r - 1$ polinommal való osztási maradékát, majd az együtthatókat $\pmod{n}$ szerint vesszük.

Mielőtt az AKS algoritmus pszeudokódját bemutatnánk, lássuk azokat az algoritmusokat, melyek meghívásra kerülnek:

- *hatvany* (lásd 2.13. algoritmus),

– *euklid* (lásd 2.5. algoritmus),
– *prim* (lásd 2.14. algoritmus),
– *trialfaktor* (lásd 2.23. algoritmus),
– *modhatv* (lásd 2.10. algoritmus),
– *polmodhatv*, mely meghatározza $(x-a)^n \pmod{x^r-1}$ polinom együtthatóit, a moduláris hatványozás elvén (lásd 2.10. algoritmus), majd a kapott együtthatókról megvizsgálja, hogy nem egyenlőek-e az $(x^n-a) \pmod{x^r-1}$ polinom együtthatóival.

Ez utóbbi algoritmus pszeudokódja a következő:

2.19. algoritmus.
Bemenet: az a, n, r egész számok.
Kimenet: 1, ha $(x-a)^n \equiv x^n - a \pmod{x^r-1}$, 0 másképp.

```
polmodhatv := proc(a, n, r)
    e := n;
    g := pol(1);
    h := pol(x - a);
    l := pol(x^r - 1);
    f := pol(x^n - a);
    f := (f pmod l) cmod n;
    while (e > 0) do
        if (e mod 2 <> 0) then
            g := g * h;
            g := (g pmod l) cmod n;
        end if;
        h := h * h;
        h := (h pmod l) cmod n;
        e := e div 2;
    end do;
    if (coef(g) = coef(f)) then return 1;
    else return 0;
    end if;
end proc;
```

2.9. megjegyzés. A fenti algoritmusban a g, h, l, f változók polinomokat azonosítanak, ezt jelzi a *pol* azonosító. A *pmod* két polinom osztási maradékát, a *cmod* a polinom együtthatóinak n-nel való osztási maradékát, míg a *coef* a paraméterként megadott polinom együtthatói listáját határozza meg. A polinomok eltárolását, a velük való műveletek megvalósítását az olvasóra bízzuk, leírást a [14]-ben találunk.

2.33. példa. A *polmodhatv* algoritmust alkalmazva vizsgáljuk meg a következő egyenlőség fennállását: $(x-1)^{61} \equiv x^{61} - 1 \pmod{x^5 - 1}$.

e	g	h	f
30	$x + 60$	$x^2 + 59x + 1$	$x + 60$
15	$x + 60$	$x^4 + 57x^3 + 6x^2 + 57x + 1$	
7	$56x^4 + 10x^3 + 51x^2 + 5x$	$9x^4 + 6x^3 + 20x^2 + 20x + 6$	
3	$37x^3 + 39x^2 + 22x + 24$	$16x^4 + 38x^3 + 16x^2 + 26x + 26$	
1	$7x^3 + 54x$	$14x^4 + 14x^3 + 40x^2 + 14x + 40$	
0	$x + 60$	$x^4 + 57x^3 + 6x^2 + 57x + 1$	

A táblázatból könnyedén leolvasható a g és f polinomok egyenlősége, tehát az eredmény 1, azaz fennáll az egyenlőség.

A fent megadott leírások után most már megadhatjuk az AKS algoritmus pszeudokódját.

2.20. algoritmus.
Bemenet: egy $n \geq 2$ egész szám.
Kimenet: ha az n szám prím, akkor 1, egyébként 0.

```
aks := proc(n)
    if (hatvany(n) = 1) then return 0;
    end if;
    r := 3;
    while (r < n) do
        if (prim(r) = 1) then
            q := trialfaktor(r-1);
            if (q>log(n)^2 and modhatv(n,(r-1)/q,r)<>1) then
                    break;
            end if;
        end if;
        r := r + 2;
    end do;
    r1 := 2;
    while (r1 < r) do
        if (euklid(n, r1) <> 1) then return 0;
        end if;
        r1 := r1 + 1;
    end do;
    for a from 1 to ceil(sqrt(r) * log(n)) do
            if (polmodhatv(a, n, r) = 1) then return 1;
            end if;
    end do;
    return 0;
end proc;
```

2.10. megjegyzés. A meghívásra kerülő *ceil* a felső egészrészt meghatározó könyvtárfüggvény.

2.34. példa. Az AKS algoritmussal vizsgáljuk meg, hogy $n = 65\,537$ prímszám-e.

$log^2 n$	r	q	$n^{(r-1)/q} \pmod r$	Magyarázat
$256,0007046$	3	2	2	
	5	2	4	
	7	3	2	
	$\vdots$	$\vdots$	$\vdots$	
	547	13	517	r prímszám
	557	139	316	
	563	281	82	$q > log^2 n$ $n^{(r-1)/q} \neq 1 \pmod r$

A fenti táblázat az *aks* algoritmus első *while* ciklusában szereplő r értéknek a meghatározását mutatja be.

Az r értékének a meghatározása után az algoritmus megvizsgálja, hogy van-e olyan

$$2 \leq r1 \leq 563, \text{ melyre } gcd(r1,\ 65\,537) = 1.$$

Mivel ilyen $r1$-t nem talál minden $a \in \{1,\ 2,\ \ldots,\ \lceil\sqrt{563} \cdot \log 65\,537\rceil\}$-ra, a *polmodhatv* függvénnyel leellenőrzi, hogy fennáll-e a

$$(x-a)^{65537} = x^{65537} - a \pmod{x^{563}-1}$$

kifejezés. Ezt minden esetben igaznak találja, tehát az 1 visszatérítési értékkel leáll, azaz jelzi, hogy a 65 537 prímszám.

2.35. példa. Az AKS algoritmussal vizsgáljuk meg, hogy az 561 prímszám-e. A lenti táblázat az r érték meghatározásának lépéssorozatát mutatja be.

$log^2 n$	r	q	$n^{(r-1)/q} \pmod r$	Magyarázat
$83,39081155$	3	2	0	
	5	2	1	
	7	3	1	
	$\vdots$	$\vdots$	$\vdots$	
	167	83	93	r prímszám
	173	43	118	
	179	89	39	$q > log^2 n$ $n^{(r-1)/q} \neq 1 \pmod r$

A második *while* ciklusban $r1 = 3$-ra nem áll fenn a $gcd(3,\ 561) = 1$, azaz n-nek és $r1$-nek a legnagyobb közös osztója $\neq 1$, tehát az algoritmus 0 futási eredménnyel leáll, azaz jelzi, hogy a szám nem prím.

2.11. megjegyzés. Az algoritmus időigényét nagymértékben befolyásolja az r, a értékek megválasztása.

2.7.6. Nagy prímek generálása

Nagy prímszámok generálására számos algoritmus ismert, melyek egy része az előző fejezetekben bemutatott prímtesztelő algoritmusokat alkalmazva *valószínűleg* prímeket, mások *erős* prímeket vagy *bizonyítottan* prímeket generálnak. Az itt bemutatásra kerülő két algoritmus közül az első *valószínűleg* prímszámot, míg a második erős prímet generál. Ez utóbbit Gordon-prímnek is nevezik.

A következő algoritmus véletlenszerűen generál egy megadott hosszúságú, 10-es alapú számrendszerbeli számot, majd a Miller–Rabin-algoritmust alkalmazva meghatározza azt a legkisebb prímszámot, mely a generált szám után következik.

2.21. algoritmus.
Bemenet: egy k pozitív egész szám és t egy biztonsági paraméter.
Kimenet: egy k hosszúságú, 10-es alapú számrendszerbeli *valószínűleg* prímszám.

```
random_prim := proc(k,t)
    p := rand(10^(k-1), 10^k)();
    if (p mod 2 = 0) then p := p + 1;
    end if;
    while (true) do
        if (miller_rabin(p, t) = 1) then return p;
        end if;
        p := p + 2;
    end do;
end proc;
```

A biztonság növelése végett a Miller–Rabin-prímteszt alkalmazása előtt, egy B konstans értékig, az osztási próba módszerét is lehet alkalmazni, hogy a B-nél kisebb prímosztókkal való oszthatóságot leellenőrizzük.

Mielőtt megadnánk a következő algoritmust, Gordon algoritmusát, mely *erős* prímszámot generál, definiáljuk az *erős* prímszám fogalmát.

2.7. értelmezés. Egy p prímszámot *erős prímnek* nevezünk, ha léteznek az r, s, q egész számok a következő tulajdonságokkal:
- $p-1$-nek van egy nagy prímtényezője, jelöljük ezt r-rel,
- $p+1$-nek van egy nagy prímtényezője, jelöljük ezt s-sel,
- $r-1$-nek van egy nagy prímtényezője, jelöljük ezt q-val.

Erős prímszámokra elsősorban az RSA kriptorendszernél (lásd a 6.2.1. fejezet) van szükségünk, bár az erős prímeket előállító algoritmusok futási idejét figyelembe véve, az RSA Security nem ajánlja ezek használatát.

A bemutatásra kerülő algoritmus a *random_prim* függvénnyel (lásd 2.21. algoritmus) meghatároz egy q prímszámot, majd a q kezdeti értéktől kezdődően sorra vizsgálja, hogy melyik az az első prímszám, mely felírható a $(2 \cdot i \cdot q + 1)$ alakba, ahol $i = 1,\ 2,\ \ldots$. Az így meghatározott prímszám lesz az r értéke. Ezután újabb prímszámot generál a *random_prim*-mel, az s-et, majd a $(2 \cdot s \cdot (s^{r-2} \pmod r) - 1)$ összefüggés alapján meghatározza az első p_0 értéket. Mindaddig, amíg meg nem találja az első p prímszámot, iterálja a $(p_0 + 2 \cdot j \cdot r \cdot s)$ összefüggést, $j = 1,\ 2,\ \ldots$ értékekre.

2.22. algoritmus.
Bemenet: egy k pozitív egész szám és t egy biztonsági paraméter.
Kimenet: egy k hosszúságú, 10-es alapú számrendszerbeli *erős* prímszám.

```
gordon_prim := proc(k, t)
    k1 := floor(k/2);
    q := random_prim(k1, t);
    i := 1;
    while (true) do
        r := 2 * i * q + 1;
        if (miller_rabin(r, t) = 1) then break;
        end if;
        i := i + 1;
    end do;
    s := random_prim(k1, t);
    p0 := 2 * s * modhatv(s, r-2, r) - 1;
    j := 1;
    while (true) do
        p := p0 + 2 * j * r * s;
        if (miller_rabin(p, t)=1) then
            return p;
        end if;
        j := j + 1;
    end do;
end proc;
```

2.12. megjegyzés. A meghívásra kerülő *floor* az alsó egészrészt meghatározó könyvtárfüggvény.

2.36. példa. Generáljunk egy erős prímet a Gordon-féle algoritmussal. Feltételezzük, hogy a véletlenszerűen generált értékek $k = 7$ esetében: $q = 467$ és $s = 281$ prímszámok.

A következő táblázat az r értékének a meghatározását mutatja be, melyet a pszeudokódban az első *while* ciklusban határozunk meg:

i	r	Magyarázat
1	935	$r = 2 \cdot 467 + 1$
2	1869	$r = 4 \cdot 467 + 1$
3	2803	$r = 6 \cdot 467 + 1 = 2803$
		r prímszám.

A következő táblázat, felhasználva az r értékét, a p értékének a meghatározását mutatja be, melyet a pszeudokód második *while* ciklusában határozunk meg:

$r \cdot s$	p_0	j	p	Magyarázat
787 643	1 351 047	1	2 926 333	$p = 1\,351\,047 + 2 \cdot 787\,643$
		2	4 501 619	$p = 1\,351\,047 + 4 \cdot 787\,643 =$
				$= 4\,501\,619$, prímszám.

Az algoritmusnak a $p = 4\,501\,619$ prím lesz a visszatérítési értéke, mely eleget tesz a kért feltételeknek, ahol

$$\begin{aligned} p - 1 &= 2 \cdot 11 \cdot 73 \cdot 2803, \\ 2802 &= 2 \cdot 3 \cdot 467, \\ p + 1 &= 2^2 \cdot 3^2 \cdot 5 \cdot 89 \cdot 281. \end{aligned}$$

2.8. Egész számok faktorizációja

A matematikában faktorizáció alatt azt értjük, hogy meghatározzuk valamely szám mátrix, polinom azon, tovább már nem bontható részeit (faktorait), melyek szorzata megadja az eredeti számot, mátrixot, polinomot. Ezen részek az egész számok esetében prímszámokat, polinomok esetében irreducibilis polinomokat jelentenek. Számos nyilvános kulcsú kriptorendszer azon alapszik, hogy az egész számok, polinomok faktorizációja máig is „nehéz” matematikai feladatnak számít, azaz nem ismert hatékony algoritmus ezen faktorok meghatározására.

A fejezet további részében azok az algoritmusok kerülnek bemutatásra, melyek meghatározzák egy adott egész szám egyik prímtényezőjét vagy osztóját. Természetesen sokak közülük kiterjeszthetőek az egész számok körén kívülre is. Ezek közül számos algoritmus kihasználja azt, hogy a szám, melyet faktorizálni szeretnénk, speciális alakú, ezáltal optimizálva az algoritmus futási idejét.

2.8.1. Osztási próba

Az alábbi algoritmus annak érdekében, hogy megtalálja egy páratlan szám, jelöljük r-rel, legnagyobb prímosztóját, sorra kipróbálja az összes páratlan számmal való oszthatóságot. Egy lehetséges variáns az, amikor csak

prímszámokkal való oszthatóságot vizsgálunk. Ez utóbbi esetben az algoritmus futási ideje jobb, de még így sem elfogadható.

Az algoritmust általában egy megadott küszöbértékig alkalmazzák, azaz ha egy bizonyos b_1 értékig nem sikerült prímosztót találni, akkor más algoritmussal kell folytatni a prímosztók keresését.

2.23. algoritmus.
Bemenet: egy r páratlan pozitív egész szám.
Kimenet: r legnagyobb prímosztója.

```
trialfaktor := proc(r)
    p := 1;
    y := 3;
    x := r;
    while (x <> 1 and y * y <= r) do
        while (x mod y = 0) do
            x := x/y;
            p := y;
        end do;
        y := y + 2;
    end do;
    if (x = 1) then return p;
    else return x;
    end if;
end proc;
```

2.37. példa. Az osztási próbát alkalmazva határozzuk meg a 91 legnagyobb prímosztóját.

x	y	p	Magyarázat
91	3	1	
91	5	1	
13	7	7	
13	9	7	ha $y = 10$ akkor $y \cdot y > r$.

Az algoritmus tehát $x = 13$ visszatérítési értékkel leáll, azaz 91 egyik prímosztója 13.

2.8.2. Fermat-faktorizáció

Az algoritmust Pierre de Fermat dolgozta ki a XVII. században, és azon az összefüggésen alapszik, miszerint tetszőleges páratlan szám felírható két négyzetszám különbségeként, azaz $r = a^2 - b^2$. Az algoritmus időigénye legrosszabb esetben az osztási próba módszerénél is rosszabb, abban az esetben viszont elfogadható, ha a szám, melyet faktorizálni szeretnénk, két azonos nagyságrendű osztóval rendelkezik. Az osztási próbát a Fermat-faktorizációval szokták kombinálni, bővebb leírást találunk a [24] könyvben.

2.24. algoritmus.
Bemenet: egy r páratlan, pozitív egész szám.
Kimenet: r egyik osztója, ha a szám prím, akkor visszaadja magát a számot.

```
fermatfaktor := proc(r)
    a1 := ceil(sqrt(r));
    b1 := a1 * a1 - r;
    while (negyzetsz(b1) = 0) do
        a1 := a1 + 1;
        b1 := a1 * a1 - r;
    end do;
    return (a1 + sqrt(b1));
end proc;
```

2.13. megjegyzés. A meghívásra kerülő *ceil* a felső egészrészt meghatározó könyvtárfüggvény, a *negyzetsz* függvény pedig a paraméterként megadott számról megvizsgálja, hogy négyzetszám-e, ez utóbbi esetben 1, más esetben 0 lesz a visszatérítési értéke. Implementálását az olvasóra bízzuk.

2.38. példa. A Fermat-faktorizációs módszert alkalmazva határozzuk meg 517 egyik osztóját.

$a1$	$b1$	Magyarázat
23	12	$\lceil\sqrt{517}\rceil = 23$
24	59	
25	108	
26	159	
27	212	
28	267	
29	324	$324 = 18 \cdot 18$, négyzetszám.

Az algoritmus tehát $29 + 18 = 47$ visszatérítési értékkel leáll. Azaz 517 felírható mint: $29^2 - 18^2 = (29 - 18) \cdot (29 + 18) = 11 \cdot 47$.

2.8.3. A Pollard ρ algoritmus

A Pollard ρ módszerét John Pollard publikálta 1975-ben. Egy tetszőleges egész szám prímosztóinak a meghatározására lehet alkalmazni, ahol az egész szám nem lehet prímhatvány, a prímosztók pedig kis nagyságrendűek, leírását lásd még a [7] könyvben.

Az algoritmus elvi működése a következő: feltételezve, hogy az algoritmus az r szám prímosztóit szeretné meghatározni, és hogy p egy prímosztója r-nek, akkor a következő egész elemű számsorozatban: $x_0,\ x_1,\ \ldots$, ahol $x_{i+1} = f(x_i)$ ismétlődéseket fogunk találni, azaz bizonyos k értékekre teljesülni fog az $x_k \equiv x_{2k} \pmod{p}$ egyenlőség. Mivel a p nem ismert, ezért

az algoritmusban a $\pmod r$ szerinti maradékos osztást fogjuk alkalmazni, és abban az esetben, ha találunk egy d értéket, melyre fennáll, hogy $1 < d = gcd(x_k - x_{2k},\ r) < r$, akkor a kapott d érték egy nem triviális osztója lesz az r-nek.

Az f függvény megválasztása természetesen meghatározza az egész algoritmus futási menetét, lásd [15]. Az egyik lehetőség, ha $x_0 = 2$ és $f(x) \equiv x^2+1 \pmod r$ választjuk, melyek szerint az alábbi algoritmus is működik.

2.25. algoritmus.
Bemenet: egy r páratlan, pozitív egész szám.
Kimenet: r egyik prímosztója.

```
pollard_rho := proc(r)
    a:=2; b:=2; i:=1;
    while (true) do
        a := (a * a + 1) mod r;
        b := (b * b + 1) mod r;
        b := (b * b + 1) mod r;
        d := euklid(a-b, r);
        if (1 < d and d < r) then return d;
        end if;
        if (d = r) then return r;
        end if;
        i := i + 1;
    end do;
end proc;
```

2.39. példa. Határozzuk meg $r = 25\,661$ egyik prímosztóját.

a	b	d	Magyarázat
2	2		
5	26	1	$2 \cdot 2 + 1 \equiv 5 \pmod{25\,661}$ $2 \cdot 2 + 1 \equiv 5 \pmod{25\,661}$ $5 \cdot 5 + 1 \equiv 26 \pmod{25\,661}$
26	22 093	1	$5 \cdot 5 + 1 \equiv 26 \pmod{25\,661}$ $26 \cdot 26 + 1 \equiv 677 \pmod{25\,661}$ $677 \cdot 677 + 1 \equiv 22\,093 \pmod{25\,661}$
677	20 384	1	$26 \cdot 26 + 1 \equiv 677 \pmod{25\,661}$ $22\,093 \cdot 22\,093 + 1 \equiv 2769 \pmod{25\,661}$ $2769 \cdot 2769 + 1 \equiv 20\,384 \pmod{25\,661}$
22 093	25 582	1	$\vdots$
2769	9167	1	
20 384	16 537	1	
4545	20 758	1	
25 582	17 877	67	$gcd(25\,582,\ 17\,877) = 67$

Mivel 25 582 és 17 877 legnagyobb közös osztója $d = 67$, az algoritmus ezzel a visszatérítési értékkel fejeződik be, azaz $r = 67 \cdot 383$.

2.8.4. A Pollard $(p-1)$ algoritmus

A Pollard $(p-1)$ algoritmust 1974-ben John Pollard ismertette. Az alábbi Pollard $(p-1)$ faktorizációs algoritmus meghatározza egy tetszőleges r egész szám egyik osztóját, leírását lásd még [20]-ban.

Mielőtt megadnánk az algoritmust, definiáljuk a *B-smooth* fogalmát:

2.8. értelmezés. Egy tetszőleges n szám *B-smooth*, ha az összes prímosztója $\leq B$, továbbá az n szám *hatvány B-smooth*, ha az összes prímhatványa $\leq B$.

Az algoritmus abban az esetben határozza meg p-t, azaz r egyik prímosztóját, ha az r szám nem prímhatvány, illetve ha $(p-1)$ *B-smooth*.

2.40. példa. Legyen $r = 218\,957$ és legyen $B = \{2, 3, 5, 7\}$, ahol r prímtényezői 347 és 631, azaz $218\,957 = 347 \cdot 631$, továbbá $346 = 2 \cdot 173$, $630 = 2 \cdot 3^2 \cdot 5 \cdot 7$. Mint látható, 346 nem *B-smooth*, míg 630 az. Tehát az r számnak van olyan p prímosztója, melyre $p-1$ *B-smooth*.

Az algoritmus a következő észrevételeken alapszik:

– Meghatározzuk minden $q \leq B$ prímszámra q^l értékét, ahol $q^l \leq r$, majd ezen hatványok legkisebb közös többszörösét. Legyen tehát

$$Q = \prod_{q \leq B} q^l, \text{ ahol } l = \lfloor \tfrac{ln\, r}{ln\, q} \rfloor,$$

ahol az l értékét pedig a következőképpen határoztuk meg:

$$q^l \leq r \Leftrightarrow l \cdot ln\, q \leq ln\, r \Rightarrow l \leq \lfloor \tfrac{ln\, r}{ln\, q} \rfloor.$$

– Ha p egy prímosztója r-nek és $p-1$ *B-smooth*, akkor $p-1|Q$, tehát bármely a-ra, melyre $gcd(a,\ p) = 1$ a kis Fermat-tétel alapján (lásd 2.1. tétel), fennáll, hogy: $a^Q \equiv 1 \pmod{p}$.

– Ha tehát $d = gcd(a^Q - 1,\ r)$, akkor $p|d$. Ha $d = n$, akkor az algoritmus nem tudja meghatározni r egyik osztóját sem.

Szükséges tehát meghatározni az $a^Q \pmod{r}$, majd $d = gcd(a^Q - 1,\ r) < r$ értékeket, ahol a véletlenszerűen generált szám, és abban az esetben, ha $1 < d < r$, visszaadni a d-t mint nem triviális osztót.

2.26. algoritmus.
Bemenet: az r.
Kimenet: az r egy triviális osztója, vagy 0, ha nem sikerült ezt meghatározni.

```
pollardp1 := proc(r)
    lis := [2,3,5,7,11,13,17];
```

```
    a := rand(2, (r-1));
    d := euklid(a, r);
    if (d >= 2) then return d;
    end if;
    i := 1;
    while (i < length(lis)) do
        l := floor(ln(r) / ln(lis[i]));
        a := modhatv(a, lis[i]^l, r);
        d := euklid(a-1, r);
        if (d > 1 and d < r) then return d;
        end if;
        if (d = r) the return 0;
        end if;
        i := i+1;
    end do;
    return 0;
end proc;
```

2.14. megjegyzés. Az algoritmusban meghívásra kerülő eljárások:
- a $floor$, az alsó egészrészt meghatározó könyvtárfüggvény,
- a *modhatv* (lásd 2.10. algoritmus),
- az *euklid* (lásd 2.5. algoritmus).

2.15. megjegyzés. A lis az első B prímszámot tartalmazza, melyet tetszés szerint lehet bővíteni, általában B elemszámának nagyságrendje 10^5 és 10^6 között van.

2.41. példa. A Pollard $p-1$ módszert alkalmazva határozzuk meg az $r = 218\,957$ egyik osztóját, ha $B = \{2, 3, 5, 7\}$.

Ha $a = 3$ a kezdetben generált véletlenszerű érték, akkor a következő táblázat az algoritmus által meghatározott értékeket tartalmazza:

$lis[i]$	l	a	d	Magyarázat
2	17	136 001	1	$136\,001 \equiv 3^{2^{17}} \pmod r$
3	11	109 125	1	$109\,125 \equiv 136\,001^{3^{11}} \equiv 3^{2^{17}\cdot 3^{11}} \pmod r$
5	7	52 506	1	$52\,506 \equiv 109\,125^{5^7} \equiv 3^{2^{17}\cdot 3^{11}\cdot 5^7} \pmod r$
7	6	62 470	631	$62\,470 \equiv 52\,506^{7^6} \equiv 3^{2^{17}\cdot 3^{11}\cdot 5^7\cdot 7^6} \pmod r$
				$1 < d < r1$

Az algoritmus tehát meghatározza a $d = 631$ osztót.

2.8.5. A kvadratikus szita algoritmus

A kvadratikus szita módszerét Carl Pomerance publikálta 1981-ben, lásd [22]. Az egyik leggyorsabb faktorizációs algoritmusnak számít. A faktorizálandó n számra egyetlen kitétel van, mégpedig az, hogy egyik prímosztója se legyen nagyobb, mint $m = \sqrt{n}$. Az algoritmus megkeresi azokat az x és y egész számokat, melyekre fennállnak a következő tulajdonságok:

$$\begin{aligned} x^2 &\equiv y^2 \pmod{n}, \\ x &\not\equiv y \pmod{n}, \\ x &\not\equiv (-y) \pmod{n}. \end{aligned}$$

A kapott x és y értékek alapján n egy osztója a $gcd(x - y,\ n)$ érték lesz.

Az algoritmus egy polinom alapján, legyen ez $q(z) = (m + z)^2 - n$, több különböző $q(z),\ z = 0,\ 1,\ -1,\ 2,\ -2,\ \ldots$ értéket határoz meg, mely értékeknek ugyanakkor meghatározza faktorizációs felbontását is.

Az algoritmus a továbbiakban megállapít egy B küszöbértéket és egy S listát, mely tartalmazni fogja azokat a $p \leq B$ prímeket, melyekre fennáll a következő tulajdonság: $\left(\frac{n}{p}\right) = 1$, ahol $\left(\frac{n}{p}\right)$ a Legendre-szimbólumot jelöli (lásd 2.2. értelmezés). A kiszámolt $q(z)$ értékek közül csak azokat fogja a későbbi feldolgozás végett eltárolni, melyek faktorizációs felbontásában nincs egyetlen egy olyan prímtényező sem, mely ne szerepelne az S listában. Tehát a kiválasztott $q(z)$ értékek *B-smooth* tulajdonságúak lesznek.

B értékének a meghatározását a szakirodalom a következő képlettel adja meg, lásd [15]:

$$\mathrm{e}^{\sqrt{\ln n \cdot \ln \ln n}}.$$

2.42. példa. Határozzuk meg $n = 53\,811, m = \sqrt{n} = 231$ esetében, z különböző értékeire a $q(z)$, $m + z$ értékeket és a $q(z)$ prímtényezős felbontásokat.

z	$q(z) = (m+z)^2 - n$	$m + z$	$q(z)$ felbontása
0	-450	231	$(-1) \cdot 2 \cdot 3^2 \cdot 5^2$
1	13	232	13
-1	-911	230	$(-1) \cdot 911$
2	478	233	$2 \cdot 239$
-2	-1370	230	$(-1) \cdot 2 \cdot 5 \cdot 137$
3	945	234	$3^3 \cdot 5 \cdot 7$
-3	-1827	230	$(-1) \cdot 3^2 \cdot 7 \cdot 29$
4	1414	235	$2 \cdot 7 \cdot 101$
-4	-2282	230	$(-1) \cdot 2 \cdot 7 \cdot 163$

2.43. példa. Határozzuk meg $n = 53\,811,\ m = \sqrt{n} = 231$ esetében azokat az $m + z$ értékeket és $q(z)$ felbontásokat, melyek $B = 100$ és

$$S = \{-1, 2, 3, 5, 7, 13, 29, 53, 59, 61, 67, 71, 83, 89, 97\}$$

esetében a $q(z)$-k *B-smooth* tulajdonságúak lesznek:

z	$q(z) = (m+z)^2 - n$	$m+z$	$q(z)$ felbontása
0	-450	231	$(-1) \cdot 2 \cdot 3^2 \cdot 5^2$
1	13	232	13
3	945	234	$3^3 \cdot 5 \cdot 7$
-3	-1827	228	$(-1) \cdot 3^2 \cdot 7 \cdot 29$

Ha az S lista elemszáma k és egy $q(z_i)$ faktorizációs felbontása: $\prod_{j=1}^{k} p_j^{e_{ij}}$, akkor a meghatározott $q(z_i)$-k száma legalább eggyel több kell legyen, mint k. Az algoritmus a faktorizációs felbontások mellett minden $q(z_i)$-re meghatároz egy listát:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{ik}), \text{ ahol } v_{ij} \equiv e_{ij} \pmod 2, \text{ és } j = 1,\ 2,\ \ldots,\ k.$$

A feladat ezek után az, hogy a v_i listák közül ki kell választani azokat a listákat, melyek vektorösszege $\pmod 2$ szerint 0, azaz fennáll, hogy:

$$\sum_{i \in T} v_i = 0, \text{ ahol } T = \{1, 2, \ldots, k\}.$$

Ezen v_i-k meghatározásával tulajdonképpen azon $q(z_i)$ értékeket választjuk ki, melyek szorzatának prímtényezős felbontásában minden prím kitevője páros, azaz a keresett y^2 értéket határozzuk meg. Ha ezek után meghatározzuk a kiválasztott z_i-khez tartozó $m + z_i$ értékek szorzatát, akkor az x egy lehetséges értékét kapjuk.

2.44. példa. Határozzuk meg $n = 25\,387$, $m = \sqrt{n} = 159$, $S = \{-1, 2, 3, 23, 41, 43, 47\}$ esetében a kiválasztott $q(z) = (m+z)^2 - n$-ket és a hozzájuk tartozó v_i listákat, majd válasszuk ki a v_i-k közül azokat, melyekre

$$\sum_{i \in T} v_i = 0, \text{ ahol } T = \{1, 2, \ldots, k\}.$$

i	z	$q(z)$	$m+z$	$q(z)$ felbontása	v_i
1	2	-738	157	$(-1) \cdot 2 \cdot 3^2 \cdot 41$	(1 1 0 0 1 0 0)
2	6	-1978	153	$(-1) \cdot 2 \cdot 23 \cdot 43$	(1 1 0 1 0 1 0)
3	10	3174	169	$2 \cdot 3 \cdot 23^2$	(0 1 1 0 0 0 0)
4	11	-3483	148	$(-1) \cdot 3^4 \cdot 43$	(1 0 0 0 0 1 0)
5	17	5589	176	$3^5 \cdot 23$	(0 0 1 1 0 0 0)
6	29	-8487	130	$(-1) \cdot 3^2 \cdot 23 \cdot 41$	(1 0 0 1 1 0 0)
7	32	11 094	191	$2 \cdot 3 \cdot 43^2$	(0 1 1 0 0 0 0)
8	80	31 734	239	$2 \cdot 3^2 \cdot 41 \cdot 43$	(0 1 0 0 1 1 0)

Észrevehetjük, hogy a $v_4 + v_5 + v_6 + v_7 + v_8 = 0$, mert

$$\begin{array}{lcl} v_4 & = & (1\ 0\ 0\ 0\ 0\ 1\ 0) \ + \\ v_5 & = & (0\ 0\ 1\ 1\ 0\ 0\ 0) \\ v_6 & = & (1\ 0\ 0\ 1\ 1\ 0\ 0) \\ v_7 & = & (0\ 1\ 1\ 0\ 0\ 0\ 0) \\ v_8 & = & (0\ 1\ 0\ 0\ 1\ 1\ 0) \\ \hline \sum_{i \in \{4,5,6,7,8\}} v_i & = & (0\ 0\ 0\ 0\ 0\ 0\ 0) \end{array}$$

Ezek szerint ha $z = 11,\ 17,\ 29,\ 32,\ 80$ értékeknek megfelelően kiválasztjuk a $q(z)$ prímtényezős felbontásokat, akkor teljes négyzetet kapunk, azaz meg tudjuk határozni az y értékét. Ha a z-knek megfelelően az $m + z$ értékeket is kiválasztjuk, akkor az x értékét is ki tudjuk számítani:

$$\begin{aligned} y^2 &= (-1 \cdot 3^4 \cdot 43) \cdot (3^5 \cdot 23) \cdot (-1 \cdot 3^2 \cdot 23 \cdot 41) \cdot \\ &\quad \cdot (2 \cdot 3 \cdot 43^2) \cdot (2 \cdot 3^2 \cdot 41 \cdot 43) = \\ &= (-1)^2 \cdot 2^2 \cdot 3^{14} \cdot 23^2 \cdot 41^2 \cdot 43^4, \end{aligned}$$

$$\begin{aligned} y &\equiv (-1) \cdot 2 \cdot 3^7 \cdot 23 \cdot 41 \cdot 43^2 \equiv 22\,426 \pmod{25\,387}, \\ x &\equiv 148 \cdot 176 \cdot 130 \cdot 191 \cdot 239 \equiv 22\,426 \pmod{25\,387}. \end{aligned}$$

Ebben az esetben az $x \equiv y \pmod{n}$ feltétel teljesül, tehát újabb v_i-ket kell kiválasztani. Észrevehetjük, hogy a $v_3 + v_7 = 0$, ezek szerint a $z = 10,\ 32$ értékeknek megfelelően is elvégezhetjük a megfelelő kiválasztásokat a $q(z)$ felbontások és $m + z$ értékek közül:

$$\begin{aligned} y^2 &= (2 \cdot 3 \cdot 23^2) \ \cdot \ (2 \cdot 3 \cdot 43^2) \\ &= 2^2 \cdot 3^2 \cdot 23^2 \cdot 43^2, \end{aligned}$$

$$\begin{aligned} y &\equiv 2 \cdot 3 \cdot 23 \cdot 43 \equiv 5934 \pmod{25\,387}, \\ x &\equiv 169 \cdot 191 \equiv 6892 \pmod{25\,387}. \end{aligned}$$

Mivel az $x \not\equiv y \pmod{n}$ és $x \not\equiv (-y) \pmod{n}$ feltételek egyike sem teljesül, a következő számítási sorozat megadja az $n = 25\,387$ két osztóját:

$$\begin{aligned} gcd(x - y,\ n) &= gcd(6892 - 5934,\ 25\,387) = 479, \\ gcd(x + y,\ n) &= gcd(6892 + 5934,\ 25\,387) = 53. \end{aligned}$$

Az algoritmus megtervezése során a következő részek hatékony implementálása a faktorizáció egészét meghatározza:

- A megfelelő S faktorbázis meghatározása, azaz milyen prímek kerülnek be az S halmazba?
- Melyek lesznek azok a $q(z)$-k, melyeket faktorizálni kell, azaz hogy szitáljuk ki (ahonnan az algoritmus elnevezése is ered) a megfelelő $q(z)$-ket? Erre vonatkozó szakirodalmat a [15]-ben találunk.
- Hogyan választjuk ki a megfelelő v_i-ket? Ezt végezhetjük a Gauss eliminációs módszert alkalmazva, lásd [8], de a gyakorlatban ennél hatékonyabb algoritmust alkalmaznak, lásd [21].

Mielőtt megadnánk a *kvadratikus* algoritmus pszeudokódját, lássuk azokat az algoritmusokat, melyek meghívásra kerülnek:

- a $faktorb$ függvény, melynek bemeneti paramétere a faktorizálandó n szám, meghatározza az S listát, a faktorbázist, azaz $\sqrt{n}$-ig azokat a prímeket, melyekre fennáll, hogy $\left(\frac{n}{p}\right) = 1$,
- a $listak$ függvény, melynek bemeneti paraméterei a faktorizálandó n szám és az S faktorbázis, a következőket határozza meg:

  - a v kétdimenziós tömböt, melynek minden egyes sora egy-egy v_i listát fog tartalmazni,
  - az $flista$ kétdimenziós tömböt, melynek minden egyes z sora az S-beli prímek hatványkitevőit tartalmazza, melyen a $q(z)$ prímtényezős felbontásában szerepelnek,
  - a b listát, mely a $m + z$-knek megfelelő értékeket tartalmazza.

– az $l\_fuggetlen$ függvény, mely egy va listát határoz meg, ahol $va[i] = 1$ azt jelzi, hogy mely v_i listát választottuk ki, azaz mely sorokra áll fenn, hogy:

$$\sum_{i \in T} v_i = 0, \text{ ahol } T = \{1, 2, \ldots, k\}.$$

– az $init(l,\ k)$ függvény, az $l,\ k$ elemű lista elmeit 0-ra inicializálja.

2.16. megjegyzés. Ezen algoritmusok implementálását az olvasóra bízzuk.

2.17. megjegyzés. A $v,\ flista$ kétdimenziós tömbök sorainak száma és a b lista elemszáma megegyezik és szigorúan nagyobb kell legyen, mint az S lista elemszáma.

2.27. algoritmus.
Bemenet: az n összetett egész szám, mely nem prímhatvány.
Kimenet: az n egy nem triviális osztója.

```
kvadratikus := proc(n)
    S := faktorb(n);
    (v, flista, b) := listak(n, S);
    i1 := 1;
    ok := 1;
    k := length(S);
    k1 := length(b);
    init(l, k);
    while (ok = 1 and i1 <= k1) do
        va := l_fuggetlen(v, k);
        x := 1 ;
        for i from 1 to k1 do
            if (va[i] = 1) then
                for j from 1 to k do
                    l[j] := l[j] + flista[i, j]:
                end do:
                x := (x * b[i]) mod n;
            end if:
        end do;
        y := 1;
        for j from 1 to k do
            y := y * (S[j]^(l[j]/2)) mod n:
```

```
            end do;
            ok := 0;
            if ((x+y) mod n=0 or (x-y) mod n=0) then ok:=1;
            end if;
            i1 := i1 + 1;
        end do;
        return euklid(x-y, n);
    end proc;
```

2.45. példa. Az előző példa adataiból kiindulva, $n = 25\,387$, $m = \sqrt{n} = 159$, $S = \{-1, 2, 3, 23, 41, 43, 47\}$ esetében a *kvadratikus* algoritmusban a *listak* függvény a következő értékeket határozza meg b-re, $flista$-ra és v-re:

i	b	$flista$	v
1	157	(1 1 2 0 1 0 0)	(1 1 0 0 1 0 0)
2	153	(1 1 0 1 0 1 0)	(1 1 0 1 0 1 0)
3	169	(0 1 1 2 0 0 0)	(0 1 1 0 0 0 0)
4	148	(1 0 4 0 0 1 0)	(1 0 0 0 0 1 0)
5	176	(0 0 5 1 0 0 0)	(0 0 1 1 0 0 0)
6	130	(1 0 2 1 1 0 0)	(1 0 0 1 1 0 0)
7	191	(0 1 1 0 0 2 0)	(0 1 1 0 0 0 0)
8	239	(0 1 2 0 1 1 0)	(0 1 0 0 1 1 0)

A következő táblázat a va lista két lehetséges értéket mutatja egy-egy kiválasztás után, ahol feltüntettük az l, x, y értékeket is.

va	l	x	y
(0 0 0 1 1 1 1 1)	(2 2 14 2 2 4 0)	22 426	22 426
(0 0 1 0 0 0 1 0)	(0 2 2 2 0 2 0)	6892	5934

A $va = (00011111)$ érték tehát azt jelzi, hogy a v_4, v_5, v_6, v_7, v_8 listák vektorösszege 0. Mivel az így meghatározott x, y értékre fennáll: $x \equiv y \pmod{n}$, újabb va listát kell meghatározni, ez pedig: $va = (00100010)$, mely azt jelzi, hogy a v_3, v_7 listák vektorösszege 0. Ebben az esetben a kiszámolt x, y értékek megfelelőek, mivel ekkor teljesülnek az $x \not\equiv y \pmod{n}$ és $x \not\equiv -y \pmod{n}$. A legnagyobb közös osztókra kapott értékek pedig a következőek lesznek:

$gcd(x - y,\ n)$	$gcd(x + y,\ n)$
479	53

.

Az algoritmus által meghatározott osztó a 479 lesz.

2.9. Primitív gyök meghatározása

Legyenek p és g tetszőleges pozitív egész számok. A g szám primitív gyök $\pmod{p}$ szerint, ha g hatványai 1-től, $\phi(p)$-ig, azaz $g,\ g^2,\ g^3,\ \ldots,\ g^{\phi(p)}$,

ahol ϕ az Euler phi-függvény (lásd 2.5. értelmezés), különböző maradékokat adnak $\pmod p$ szerint, lásd [20]. A primitív gyök egy sajátos esetét jelenti a multiplikatív csoportok generátor elemének.

2.46. példa. Az alábbi táblázat azt szemlélteti, hogy 7 primitív gyök (mod 13) szerint, míg 8 nem az.

$$\begin{array}{llll} 7^1 \equiv 7 \pmod{13} & & 8^1 \equiv 8 \pmod{13} \\ 7^2 \equiv 10 \pmod{13} & & 8^2 \equiv 12 \pmod{13} \\ 7^3 \equiv 5 \pmod{13} & & 8^3 \equiv 5 \pmod{13} \\ 7^4 \equiv 9 \pmod{13} & & 8^4 \equiv 1 \pmod{13} \\ 7^5 \equiv 11 \pmod{13} & & 8^5 \equiv 8 \pmod{13} \\ 7^6 \equiv 12 \pmod{13} & & 8^6 \equiv 12 \pmod{13} \\ 7^7 \equiv 6 \pmod{13} & & 8^7 \equiv 5 \pmod{13} \\ 7^8 \equiv 3 \pmod{13} & & 8^8 \equiv 1 \pmod{13} \\ 7^9 \equiv 8 \pmod{13} & & 8^9 \equiv 8 \pmod{13} \\ 7^{10} \equiv 4 \pmod{13} & & 8^{10} \equiv 12 \pmod{13} \\ 7^{11} \equiv 2 \pmod{13} & & 8^{11} \equiv 5 \pmod{13} \\ 7^{12} \equiv 1 \pmod{13} & & 8^{12} \equiv 1 \pmod{13} \end{array}$$

Primitív gyök az egész számok körében pontosan az $n = 2,\ 4,\ p^k,\ 2 \cdot p^k$ alakú számokra létezik, ahol p páratlan prímszám és $k > 0$. Egy p prímszám esetében a primitív gyökök száma $\phi(p)$, ahonnan megállapítható, hogy $\phi(p)/p$ az arány, azaz „jó” az esélye annak, hogy ha véletlenszerűen meghatározunk egy számot az $1,\ 2,\ \ldots,\ (p-1)$ számok közül, akkor az primitív gyök legyen.

A bemutatásra kerülő algoritmus először meghatározza $n = p - 1$ prímtényezőinek listáját, legyen ez $pt_1,\ pt_2,\ \ldots,\ pt_k$, majd egy véletlenszerűen generált a érték esetében sorra meghatározza

$$a^{n/pt_i} \pmod p,\ i = 1,\ 2,\ \ldots,\ k$$

értékeket. Ha ezek között az értékek között talál egy 1-gyel egyenlő számot, akkor újabb a értéket generál, mert az előbbi a nem volt primitív gyök. Az algoritmusról leírást találunk az [5] könyvben.

2.28. algoritmus.
Bemenet: egy p prímszám.
Kimenet: a p prímszám egy primitív gyöke.

```
primitiv := proc(p)
    n := p - 1;
    pt := primtenyezok(n);
    b := 0;
    a := rand(2, n);
    i := 1;
    while (i <= length(pt)) do
```

```
            b := modhatv(a, n div pt[i], p);
            i := i + 1;
            if (b = 1) then
                a := rand(2, n);
                i := 1;
            end if;
        end do;
    return a;
end proc;
```

2.18. megjegyzés. Az algoritmus keretében meghívásra kerül a már korábban bemutatott *modhatv* algoritmus (lásd 2.10. algoritmus), illetve a *primtenyezok*, mely a paraméterként megadott egész szám prímtényezői listáját határozza meg. Ez utóbbi megírását az olvasóra bízzuk.

2.47. példa. Határozzuk meg $p = 271$ prímszám egy primitív gyökét, ha $n - 1 = 270$ prímtényezős felbontása: $2 \cdot 3^3 \cdot 5$, azaz a $pt = [2, 3, 5]$.

$a = 115$, véletlenszerűen generált érték esetében az algoritmus által meghatározott értékek a következőek:

i	b	Magyarázat
1	$115^{270/2} \equiv 115^{135} \equiv 270 \pmod{271}$	
2	$115^{270/3} \equiv 115^{90} \equiv 28 \pmod{271}$	
3	$115^{270/5} \equiv 115^{54} \equiv 1 \pmod{271}$	$b = 1$, 115 nem primitív gyök

Mivel a fenti véletlenszerűen generált érték nem volt primitív gyök, az algoritmus újabb véletlen értéket generál, legyen ez $a = 58$, mely esetben a következő értékek lesznek meghatározva:

i	b	Magyarázat
1	$58^{270/2} \equiv 58^{135} \equiv 270 \pmod{271}$	
2	$58^{270/3} \equiv 58^{90} \equiv 242 \pmod{271}$	
3	$58^{270/5} \equiv 58^{54} \equiv 244 \pmod{271}$	tehát 58 primitív gyök.

A meghatározott moduláris hatványok között nem talált 1-gyel egyenlőt, tehát 58, mint primitív gyök visszatérítési értékkel leáll az algoritmus.

2.10. Diszkrét logaritmus meghatározása

Sok kriptorendszer biztonsága a diszkrét logaritmálás „nehézségén” alapszik. Ezek közé sorolható a Diffie–Hellman-kulcscsere vagy az ElGamal nyilvános kulcsú kriptorendszer. A diszkrét logaritmus fogalmát, illetve a bemutatásra kerülő algoritmusokat az egyszerűség kedvéért az egész számok

körében adjuk meg. Ekkor b a-alapú diszkrét logaritmusa $\pmod p$ szerint azt jelenti, hogy megkeressük azt az x pozitív egész számot, melyre fennáll:

$$a^x \equiv b \pmod p,$$

ahol az $a,\ b,\ p$ adott pozitív egész számok.

A fejezet további részeiben azok a fontosabb algoritmusok kerülnek bemutatásra, melyek megkísérlik hatékonyan megoldani ezt a problémát.

2.10.1. A baby-step giant-step algoritmus

A legkézenfekvőbb algoritmus az lenne, hogy sorra meghatározzuk az $a^0,\ a^1,\ a^2,\ \ldots$ értékeket, mindaddig, amíg az nem kongruens b-vel. Ekkor az aktuális hatványkitevő lesz a keresett x érték. Ennek az eljárásnak az időigénye természetesen elfogadhatatlan.

A baby-step giant-step algoritmus, melynek alapötlete Daniel Shankstől származik 1971-ből, a fenti leírásnak egy módosított változata, megkeresi tehát azt az x pozitív egész számot, melyre fennáll: $a^x \equiv b \pmod p$, ahol az $a,\ b,\ p$ adott pozitív egész számok, lásd [5]. Azon az észrevételen alapszik, hogy ha a kitevő értékét átírjuk, azaz

$$x = i \cdot m + j, \text{ ahol } 0 \leq i,\ j < m \text{ és } m = \lceil \sqrt{p-1} \rceil,$$

egyszerűbb eljárással kerülünk szembe, mert:

$$\begin{aligned} a^x &= a^{i \cdot m + j} = a^{i \cdot m} \cdot a^j \\ a^j &= a^x \cdot (a^{-m})^i = b \cdot (a^{-m})^i, \text{ ahol} \end{aligned}$$

az a^{-m} érték az a multiplikatív inverzét jelöli, az m-ik hatványon.

Az algoritmus $j = 0,\ 1,\ \ldots,\ m-1$ értékekre sorra meghatározza az a^j -ket, melyeket egy listában eltárol. Ezek után sorra kipróbálja, hogy milyen $i = 0,\ 1,\ \ldots,\ m-1$ -re egyezik meg $b \cdot (a^{-m})^i$ értéke valamely, már korábban meghatározott listabeli elemmel. Ha sikerül egy ilyen egyezést találnia, legyen ez $j = j_k$ és $i = i_k$, akkor meg tudja határozni a keresett x értékét a következőképpen: $x = i_k \cdot m + j_k$.

2.29. algoritmus.
Bemenet: a p prímszám és az $a,\ b \leq p$ pozitív egész számok.
Kimenet: az x pozitív egész szám, melyre fennáll: $a^x \equiv b \pmod p$.

```
baby_giant_step := proc(p, a, b)
    m := ceil(sqrt(p-1));
    inv := inverz(a,p);
    inv := (inv^m) mod p;
    lista := [];
    for j from 0 to m-1 do
        lista := lista + [(a^j) mod p];
```

```
    end do;
    for i from 0 to (m-1) do
        c := (b * (inv^i)) mod p;
        for j from 0 to length(lista) - 1 do
            if (c = lista[j]) then
                x := i * m + j;
                return x;
            end if;
        end do;
    end do;
end proc;
```

2.19. megjegyzés. A meghívásra kerülő *ceil* a felső egészrészt meghatározó könyvtárfüggvény.

2.48. példa. Határozzuk meg azt az x hatványkitevőt, melyre $6^x = 21 \pmod{1423}$.

Az algoritmus által meghatározott értékek a következőek lesznek:

$$m = 38,\ a^{-1} \equiv 1186 \pmod{1423}, \text{ és } a^{-m} \equiv 674 \pmod{1423}.$$

A *lista* változóba tehát 38 elem kerül a következő értékekkel: [1, 6, 36, 216, 1296, 661, 1120, 1028, 476, 10, 60, 360, 737, 153, 918, 1239, 319, 491, 100, 600, 754, 255, 107, 642, 1006, 344, 641, 1000, 308, 425, 1127, 1070, 728, 99, 594, 718, 39, 234].

A c-re meghatározott értékek, $i = 0,\ 1,\ \ldots,$ 12-re a következőek lesznek:

$$21,\ 1347,\ 4,\ 1273,\ 1356,\ 378,\ 55,\ 72,\ 146,\ 217,\ 1112,\ 990,\ 1296.$$

Mivel az $i = 12$-re meghatározott 1296-os érték szerepel a *lista* elemei között, a $j = 4$-re kiszámolt pozición, a keresett x hatványkitevő tehát kiszámítható, azaz $x = 12 \cdot 38 + 4 = 460$.

Az algoritmus hatékonyságát befolyásolja, hogy a *lista* változó tartalmát hogyan tároljuk és a keresést hogyan végezzük benne. Ha hash táblát (lásd [8]) használunk (amit ne tévesszünk össze a kriptográfiában használt hash függvényekkel), akkor a hatékonyság nagyban növelhető.

2.10.2. A Pohlig–Hellman-algoritmus

A Pohlig–Hellman-algoritmus, melyet 1978-ban publikáltak, szerzői Pohlig és Hellman, meghatározza azt az x számot, melyre: $a^x \equiv b \pmod{m}$, ahol $a,\ b,\ m$ tetszőleges pozitív egész számok és $m - 1$ prímosztói *B-smoot* tulajdonságúak, lásd 2.8. értelmezés. Az algoritmus futási ideje polinomiális. Részletes leírást találunk az [5] könyvben.

Az algoritmus első lépésében meghatározzuk $m-1$ prímtényezős felbontását. Legyen ez: $N = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$, majd az $x_1,\ x_2,\ \ldots,\ x_n$ értékeket, melyek rendre a következő diszkrét logaritmus feladatok megoldásai lesznek:

$$\begin{aligned}
{a_1}^{x_1} &\equiv b_1 \pmod m, \quad ahol\ a_1 = a^{N/p_1^{e_1}} \quad b_1 = b^{N/p_1^{e_1}} \\
{a_2}^{x_2} &\equiv b_2 \pmod m, \quad ahol\ a_2 = a^{N/p_2^{e_2}} \quad b_2 = b^{N/p_1^{e_1}} \\
&\vdots \\
{a_n}^{x_n} &\equiv b_n \pmod m, \quad ahol\ a_n = a^{N/p_n^{e_n}} \quad b_n = b^{N/p_1^{e_1}}.
\end{aligned}$$

Az $x_1,\ x_2,\ \ldots,\ x_n$ értékek meghatározása után a kínai maradéktétel algoritmusával megoldjuk a következő lineáris kongruenciarendszert, melynek x megoldása a keresett diszkrét logaritmus értéke lesz:

$$\begin{aligned}
x &\equiv x_1 \pmod{{p_1}^{e_1}} \\
x &\equiv x_2 \pmod{{p_2}^{e_2}} \\
&\vdots \\
x &\equiv x_n \pmod{{p_n}^{e_n}}.
\end{aligned}$$

Az $x_1,\ x_2,\ \ldots,\ x_n$ értékek meghatározása érdekében mindegyik x_i-t, $i = 1,\ 2,\ \ldots,\ n$, felírjuk $x_i = y_{i0} + p_i \cdot y_{i1} + {p_i}^2 \cdot y_{i2} + \ldots + {p_i}^{e_i-1} \cdot y_{ie_i-1}$ alakba.

Az $x_i,\ i = 1,\ 2,\ \ldots,\ n$ meghatározásához előbb kiszámítjuk az y_{i0} értéket, majd ennek függvényében az y_{i1}-t stb., azaz az y_{ij} értéket mindig az előző y_{ij-1}érték határozza meg.

Mindegyik $y_{ij},\ j = 0,\ 1,\ \ldots,\ e_i - 1$ érték meghatározása egy diszkrét logaritmálást jelent, melyet a baby-step giant-step algoritmussal is végezhetünk, lásd 2.29. algoritmus:

$$\begin{aligned}
({a_i}^{{p_i}^{(e_i-1)}})^{y_{i0}} &\equiv {b_i}^{{p_i}^{(e_i-1)}} \pmod m \\
({a_i}^{{p_i}^{(e_i-1)}})^{y_{i1}} &\equiv {b_i}^{{p_i}^{(e_i-2)}} \cdot {a_i}^{-({p_i}^{e_i-2} \cdot y_{i0})} \pmod m \\
({a_i}^{{p_i}^{(e_i-1)}})^{y_{i2}} &\equiv {b_i}^{{p_i}^{(e_i-3)}} \cdot {a_i}^{-({p_i}^{e_i-3} \cdot y_{i0} + {p_i}^{e_i-2} \cdot y_{i1})} \pmod m \\
&\vdots \\
({a_i}^{{p_i}^{(e_i-1)}})^{y_{ie_i-1}} &\equiv {b_i}^{{p_i}^{0}} \cdot {a_i}^{-(y_{i0} + p_i \cdot y_{i1} + {p_i}^2 \cdot y_{i2} \ldots + {p_i}^{e_i-2} \cdot y_{ie_i-2})} \pmod m
\end{aligned}$$

Mielőtt leírnánk a Pohlig–Hellman-algoritmust, ennek egy segéd algoritmusát, a $felbontas$-t mutatjuk be, mely a paraméterként megadott p_i prímtényező esetében meghatározza az ${a_i}^{x_i} \equiv b_i \pmod m$ kifejezésnek eleget tevő x_i értéket, ahol $a_i = a^{N/p_i^{e_i}}$ és $b_i = b^{N/p_i^{e_i}}$. A Pohlig–Hellman-algoritmus pedig $m-1$ minden egyes prímtényezőjére meghívja a $felbontas$-t.

2.30. algoritmus.
Bemenet: az m prímszám, az $a,\ b,\ p,\ e \leq m$ pozitív egész számok.
Kimenet: x, ahol x felírható $y_0 + p \cdot y_1 + p^2 \cdot y_2 + \ldots + p^{e-1} \cdot y_{e-1}$ összegként.

```
felbontas := proc(m, a, b, p, e)
    y := [];
    a1 := a^(p^(e-1)) mod m;
    b1 := b^(p^(e-1)) mod m
    y := y + [ baby_giant_step(m, a1, b1) ];
    osz := y[1];
    szor := 1;
    for i from 2 to e do
        e1 := 0;
        for j from 1 to length(y) do
            e1 := e1 + y[j] * p^(e-1-i+j);
        end do;
        b2 := inverz(a^(e1),m);
        b1 := (b^(p^(e-i)) * b2) mod m;
        y := y + [baby_giant_step(m, a1, b1)];
        szor := szor * p;
        osz := osz + y[i] * szor;
    end do;
    return osz;
end proc;
```

A Pohlig–Hellman-algoritmus ezek után a következő lesz:

2.31. algoritmus.
Bemenet: az m prímszám, az $a,\ b \leq m$ pozitív egész számok.
Kimenet: az x pozitív egész szám, melyre fennáll: $a^x = b \pmod{m}$.

```
pohlig_hellman := proc(m, a, b)
    (plista, hlista) := tenyezok(m-1);
    xl := [];
    p := [];
    for i from 1 to length(plista) do
        sz := 1;
        p := p + [plista[i]^hlista[i]];
        for j from 1 to length(plista) do
            if (i<>j) then sz := sz * (plista[j]^hlista[j]);
            end if;
        end do;
        a1 := (a^sz) mod m;
        b1 := (b^sz) mod m;
        xl := xl + [felbontas(m,a1,b1,plista[i],hlista[i])];
    end do;
    x := kinaimt(xl, p, length(plista));
    return x;
end proc;
```

Az algoritmusban meghívásra kerülő *tenyezok* két listát határoz meg, az egyik a prímtényezők listáját a *plista*-ba, a másik a prímtényezőknek megfelelő hatványkitevők listáját adja meg a *hlista*-ba, ezeknek az implementálását az olvasóra bízzuk. A *kinaimt* algoritmus leírását lásd a 2.9. algoritmusnál.

2.49. példa. Határozzuk meg azt az x számot, melyre $815^x \equiv 14 \pmod{8641}$, azaz határozzuk meg 14, 815-ös alapú diszkrét logaritmusát $\pmod{8641}$ szerint, ahol $m = 8641$ prímszám és $a = 815$ m-nek egy primitív gyöke. Az $m - 1 = 8640$ prímtényezős felbontása: $2^6 \cdot 3^3 \cdot 5$. A feladat megoldásának első része ezek után a következő diszkrét logaritmusok meghatározása lesz:

$$\begin{aligned}
\left(815^{3^3\cdot 5}\right)^{x_1} &\equiv 14^{3^3\cdot 5} \pmod{8641}\\
\left(815^{2^6\cdot 5}\right)^{x_2} &\equiv 14^{2^6\cdot 5} \pmod{8641}\\
\left(815^{2^6\cdot 3^3}\right)^{x_3} &\equiv 14^{2^6\cdot 3^3} \pmod{8641},
\end{aligned}$$

melyben elvégezve a műveleteket, kapjuk:

$$\begin{aligned}
1667^{x_1} &\equiv 6552 \pmod{8641}\\
6503^{x_2} &\equiv 3185 \pmod{8641}\\
26^{x_3} &\equiv 1 \pmod{8641}.
\end{aligned}$$

Az x_1, x_2, x_3 értékek meghatározása végett sorra felírjuk őket a következő formába:

$$\begin{aligned}
x_1 &= y_{10} + 2\cdot y_{11} + 2^2\cdot y_{12} + 2^3\cdot y_{13} + 2^4\cdot y_{14} + 2^5\cdot y_{15},\\
x_2 &= y_{20} + 3\cdot y_{21} + 3^2\cdot y_{22},\\
x_3 &= y_{30}.
\end{aligned}$$

Következzék most az x_1 meghatározása, ahol $x_1 = y_{10} + 2\cdot y_{11} + 2^2\cdot y_{12} + 2^3\cdot y_{13} + 2^4\cdot y_{14} + 2^5\cdot y_{15}$. Ebben az esetben tehát a következő diszkrét logaritmus-feladatokat kell megoldanunk:

$$\begin{aligned}
(1667^{2^5})^{y_{10}} &\equiv 6552^{2^5} \pmod{8641}\\
(1667^{2^5})^{y_{11}} &\equiv 6552^{2^4}\cdot 1667^{-(2^4\cdot y_{10})} \pmod{8641}\\
(1667^{2^5})^{y_{12}} &\equiv 6552^{2^3}\cdot 1667^{-(2^3\cdot y_{10}+2^4\cdot y_{11})} \pmod{8641}\\
(1667^{2^5})^{y_{13}} &\equiv 6552^{2^2}\cdot 1667^{-(2^2\cdot y_{10}+2^3\cdot y_{11}+2^4\cdot y_{12})} \pmod{8641}\\
(1667^{2^5})^{y_{14}} &\equiv 6552^{2}\cdot 1667^{-(2\cdot y_{10}+2^2\cdot y_{11}+2^3\cdot y_{12}+2^4\cdot y_{13})} \pmod{8641}\\
(1667^{2^5})^{y_{15}} &\equiv 6552\cdot 1667^{-(y_{10}+2\cdot y_{11}+2^2\cdot y_{12}+2^3\cdot y_{13}+2^4\cdot y_{14})} \pmod{8641}
\end{aligned}$$

A számítások elvégzése után kapjuk:

$$\begin{aligned}
8640^{y_{10}} &\equiv 8640 \pmod{8641} && \to y_{10} = 1\\
8640^{y_{11}} &\equiv 7058\cdot 1667^{-(2^4\cdot 1)} \pmod{8641} && \to y_{11} = 0\\
8640^{y_{12}} &\equiv 2103\cdot 1667^{-(2^3\cdot 1+2^4\cdot 0)} \pmod{8641} && \to y_{12} = 1\\
8640^{y_{13}} &\equiv 3451\cdot 1667^{-(2^2\cdot 1+2^3\cdot 0+2^4\cdot 1)} \pmod{8641} && \to y_{13} = 0\\
8640^{y_{14}} &\equiv 216\cdot 1667^{-(2\cdot 1+2^2\cdot 0+2^3\cdot 1+2^4\cdot 0)} \pmod{8641} && \to y_{14} = 1\\
8640^{y_{15}} &\equiv 6552\cdot 1667^{-(1+2\cdot 0+2^2\cdot 1+2^3\cdot 0+2^4\cdot 1)} \pmod{8641} && \to y_{15} = 0
\end{aligned}$$

Tehát $x_1 = 1 + 2 \cdot 0 + 2^2 \cdot 1 + 2^3 \cdot 0 + 2^4 \cdot 1 + 2^5 \cdot 0 = 21$.

Következzék most az x_2 meghatározása, ahol $x_2 = y_{20} + 3 \cdot y_{21} + 3^2 \cdot y_{22}$. Ebben az esetben tehát a következő diszkrét logaritmus-feladatokat kell megoldanunk:

$$\begin{aligned}
(6503^{3^2})^{y_{20}} &\equiv 3185^{3^2} \pmod{8641} \\
(6503^{3^2})^{y_{21}} &\equiv 3185^3 \cdot 6503^{-(3 \cdot y_{20})} \pmod{8641} \\
(6503^{3^2})^{y_{22}} &\equiv 3185 \cdot 6503^{-(y_{20} + 3 \cdot y_{21})} \pmod{8641}.
\end{aligned}$$

A számítások elvégzése után kapjuk:

$$\begin{aligned}
5068^{y_{20}} &\equiv 5068 \pmod{8641} && \rightarrow y_{20} = 1 \\
5068^{y_{21}} &\equiv 909 \cdot 6503^{-(3 \cdot 1)} \pmod{8641} && \rightarrow y_{21} = 2 \\
5068^{y_{22}} &\equiv 3185 \cdot 6503^{-(1 + 3 \cdot 2)} \pmod{8641} && \rightarrow y_{22} = 1.
\end{aligned}$$

Tehát $x_2 = 1 + 3 \cdot 2 + 3^2 \cdot 1 = 16$.

Következzék most az x_3 meghatározása, ahol $x_3 = y_{30}$. Ebben az esetben egyetlen diszkrét logaritmus-feladatot kell megoldanunk:

$$(26^{5^2})^{y_{30}} \equiv 1^{5^2} \pmod{8641},$$

azaz:

$$1^{y_{30}} \equiv 1 \pmod{8641} \quad \rightarrow y_{30} = 0.$$

Tehát $x_3 = 0$.

Az $x_1,\ x_2,\ x_3$ értékek meghatározása után a feladat második része az, hogy a kínai maradéktétel algoritmusával megoldjuk a következő lineáris kongruenciarendszert:

$$\begin{aligned}
x &\equiv 21 \pmod{2^6} \\
x &\equiv 16 \pmod{3^3} \\
x &\equiv 0 \pmod{5}.
\end{aligned}$$

A fenti rendszer megoldása, azaz a keresett diszkrét logaritmus értéke: 7765, tehát $815^{7765} \equiv 14 \pmod{8641}$.

2.10.3. Az indexkalkulus-algoritmus

Leonard Adleman 1979-ben publikálta azt a cikkét, melyben a diszkrét logaritmus problémáját oldotta meg subexponenciális idejű algoritmussal, és melynek az indexkalkulus elnevezést adta. Az alapötlet nem teljesen tőle származik, ugyanakkor vele egy időben mások is hasonló megoldáshoz jutottak. A bemutatásra kerülő algoritmus ennek egy egyszerűsített változata, lásd [5].

Feltételezve, hogy $a^x \equiv b \pmod{m}$ esetében, ahol a, m ismertek, szeretnénk meghatározni az x értékét, az algoritmus lépései a következők:

1. Meghatározzuk az első n prímszám halmazát, melyet faktorbázisnak fogunk hívni és melyet B-vel jelölünk. Az n nagyságrendjére vonatkozó feltételeket lásd a [15]-ben.
2. A B faktorbázis minden elemére meghatározzuk a megfelelő diszkrét logaritmusértékeket, azaz
$$a^{x(q)} \equiv q \pmod{m}$$
esetében az $x(q)$ értékét, ahol $q \in B$, a következőképpen:
   - véletlenszerűen generálunk különböző r értékeket, ahol $1 \leq r \leq m-1$,
   - a generált értéket „kiválasztjuk", ha az $a^r \pmod{m}$ kifejezés prímtényezős felbontásában előforduló prímszámok mindegyike szerepel a B elemei között,
   - legalább annyi r véletlenszámot generálunk, hogy a kiválasztottak száma nagyobb legyen, mint a B halmaz elemszáma,
   - feltételezve, hogy az $a^r \pmod{m}$ prímtényezős felbontása a következő: $\prod_{q \in B} q^{e(q,r)}$, akkor az előforduló q értékekre elvégezzük a $q \equiv a^{x(q)} \pmod{m}$ behelyettesítést:
$$a^r \equiv a^{\sum_{q \in B} e(q,r) \cdot x(q)} \pmod{m},$$
   - alkalmazzuk a következő számelméleti tételt, lásd [11]:

     2.3. tétel.
     $q^x \equiv q^y \pmod{m} \Leftrightarrow x \equiv y \pmod{m-1}$, ahol q primitív gyök $\pmod{m}$ szerint és m prímszám
   - a tétel alapján minden „kiválasztott" r értékre felírhatjuk:
$$r \equiv \sum_{q \in B} e(q,r) \cdot x(q) \pmod{m-1},$$
     melyek egy lineáris kongruenciarendszert adnak,
   - a Gauss eliminációs algoritmussal (lásd [8]) megoldva a rendszert kapjuk a megfelelő $x(q)$ értékeket.
3. Generálunk egy y véletlen értéket, majd meghatározzuk
$$b \cdot a^y \pmod{m}$$
prímtényezős felbontását, legyen ez: $\prod_{q \in B} q^{e(q)}$. Ha lesz egy olyan prímtényező, mely nem eleme a B halmaznak, akkor újabb y értéket generálunk. Ellenkező esetben a következő számításokat elvégezve, megkapjuk a keresett x értékét:
$$x \equiv (\sum_{q \in B} x(q) \cdot e(q) - y) \pmod{m-1},$$
mert:
$$b \cdot a^y \equiv \prod_{q \in B} q^{e(q)} \equiv \prod_{q \in B} a^{x(q) \cdot e(q)} \equiv a^{\sum_{q \in B} x(q) \cdot e(q)} \pmod{m},$$
ahonnan
$$a^x \equiv b \equiv a^{\sum_{q \in B} x(q) \cdot e(q) - y} \pmod{m}.$$

Mielőtt az *index_kalkulus* algoritmus pszeudokódját bemutatnánk, lássuk azokat az algoritmusokat, melyek meghívásra kerülnek:

- a $faktorb$, mely meghatározza a B faktorbázist,
- a *modhatv*, mely a moduláris hatványozást végzi, lásd 2.10. algoritmus,
- a $primfaktorok(e,\ S)$, ha az e szám legnagyobb prímtényezője is szerepel a faktorbázis elemei között, akkor az $ok = 1$ és meghatározza a p listába azoknak a hatványkitevőknek az értékét, melyeken a faktorbázis prímjei szerepelnek az e szám prímtényezős febontásában,
- a *gsolve*, mely a megadott paraméterekre megoldja a lineáris kongruenciarendszert, azaz meghatározza az $x(q)$ értékeket, ennek megírását az olvasóra bízzuk.

Az algoritmus az első *while* ciklusban meghatározza p-be, az r véletlenszerűen generált számok esetében, az $a^r \pmod m$ prímtényezőinek listáját. Ha a legnagyobb prímtényező is szerepel az S faktorbázis elemei között, akkor az r értékét a v listához adja, a *mat* kétdimenziós tömb aktuális sorának pedig a p elemeit felelteti meg.

A *gsolve* függvénnyel meghatározza a faktorbázis elemeire a diszkrét logaritmusértékeket, majd a következő *while* ciklusban kiválasztja azt az y értéket, melynek minden prímtényezője szerepel az S-ben. A for ciklusban a megfelelő összegzési képlettel kiszámítja a keresett x diszkrét logaritmusértéket.

2.32. algoritmus.
Bemenet: az m prímszám, az $a,\ b \leq m$ pozitív egész számok.
Kimenet: az x pozitív egész szám, melyre fennáll: $a^x \equiv b \pmod m$.

```
index_kalkulus := proc(m, a , b, n)
    S := faktorb(n);
    i := 1;
    db := length(S);
    v := [];
    while (i <= (db+1)) do
        r := rand(1, m-1);
        e := modhatv(a, r, m);
        ok := 0;
        (p, ok) := primfaktorok(e, S);
        if (ok = 1) then
            v := v + [r];
            for j from 1 to length(p) do
                mat[i, j] := p[j];
            end do;
            i := i + 1;
        end if;
    end do;
    g := gsolve(mat, v) mod (m-1);
```

```
    ok := 0;
    while (ok = 0) do
        y := rand(1, m-1);
        e := b * modhatv(a, y, m) mod m;
        (p, ok) := primfaktorok (e, S);
    end do;
    osszeg := 0;
    for i from 1 to db do
        if (p[i] <> 0) then
            osszeg:= (osszeg + p[i] * g[i]) mod (m-1);
        end if;
    end do;
    return (osszeg - y)mod(m-1);
end proc;
```

2.50. példa. Határozzuk meg azt az x számot, melyre

$$815^x \equiv 14 \pmod{8641},$$

azaz határozzuk meg 14, 815-ös alapú diszkrét logaritmusát (mod 8641) szerint, ahol $m = 8641$ prímszám és $a = 815$ egy primitív gyöke.

Legyen $B = \{2, 3, 5, 7, 11, 13, 17\}$. Előbb meghatározzuk a faktorbázis elemeire a logaritmusértékeket, azaz:

$$\begin{aligned}
815^{x(2)} &\equiv 2 \pmod{8641}\\
815^{x(3)} &\equiv 3 \pmod{8641}\\
815^{x(5)} &\equiv 5 \pmod{8641}\\
815^{x(7)} &\equiv 7 \pmod{8641}\\
815^{x(11)} &\equiv 11 \pmod{8641}\\
815^{x(13)} &\equiv 13 \pmod{8641}\\
815^{x(17)} &\equiv 17 \pmod{8641}.
\end{aligned}$$

Ennek érdekében a következőképpen járunk el:

– feltételezve, hogy a véletlenszerűen generált r értékek rendre:

$$3072, 5269, 3364, 3890, 7028, 4852, 6398, 3701, 8048,$$

meghatározzuk mindegyik r-re, $815^r \pmod{8641}$ értékét és prímtényezős felbontását:

$$\begin{aligned}
815^{3072} &\equiv 4860 \equiv 2^2 \cdot 3^5 \cdot 5 && \pmod{8641}\\
815^{5269} &\equiv 612 \equiv 2^2 \cdot 3^2 \cdot 17 && \pmod{8641}\\
815^{3364} &\equiv 4158 \equiv 2 \cdot 3^3 \cdot 7 \cdot 11 && \pmod{8641}\\
815^{3890} &\equiv 7225 \equiv 5^2 \cdot 17^2 && \pmod{8641}\\
815^{7028} &\equiv 2916 \equiv 2^2 \cdot 3^6 && \pmod{8641}\\
815^{4852} &\equiv 3213 \equiv 3^3 \cdot 7 \cdot 17 && \pmod{8641}\\
815^{6398} &\equiv 600 \equiv 2^3 \cdot 3 \cdot 5^2 && \pmod{8641}\\
815^{3701} &\equiv 1666 \equiv 2 \cdot 7^2 \cdot 17 && \pmod{8641}\\
815^{8048} &\equiv 6930 \equiv 2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 && \pmod{8641}
\end{aligned}$$

– ha elvégezzük a faktorbázis prímértékeire a megfelelő behelyettesítéseket, akkor kapjuk:

$$\begin{aligned}
815^{3072} &\equiv 815^{2\cdot x(2)+5\cdot x(3)+x(5)} && \pmod{8641}\\
815^{5269} &\equiv 815^{2\cdot x(2)+2\cdot x(3)+x(17)} && \pmod{8641}\\
815^{3364} &\equiv 815^{x(2)+3\cdot x(3)+x(7)+x(11)} && \pmod{8641}\\
815^{3890} &\equiv 815^{2\cdot x(5)+2\cdot x(17)} && \pmod{8641}\\
815^{7028} &\equiv 815^{2\cdot x(2)+6\cdot x(3)} && \pmod{8641}\\
815^{4852} &\equiv 815^{3\cdot x(3)+x(7)+x(17)} && \pmod{8641}\\
815^{6398} &\equiv 815^{3\cdot x(2)+x(3)+2\cdot x(5)} && \pmod{8641}\\
815^{3701} &\equiv 815^{x(2)+2\cdot x(7)+x(17)} && \pmod{8641}\\
815^{8048} &\equiv 815^{x(2)+2\cdot x(3)+x(5)+x(7)+x(11)} && \pmod{8641}
\end{aligned}$$

– áttérve a hatványkitevőkre, kapjuk:

$$\begin{aligned}
2\cdot x(2)+5\cdot x(3)+x(5) &\equiv 3072 && \pmod{8640}\\
2\cdot x(2)+2\cdot x(3)+x(17) &\equiv 5269 && \pmod{8640}\\
x(2)+3\cdot x(3)+x(7)+x(11) &\equiv 3364 && \pmod{8640}\\
2\cdot x(5)+2\cdot x(17) &\equiv 3890 && \pmod{8640}\\
2\cdot x(2)+6\cdot x(3) &\equiv 7028 && \pmod{8640}\\
3\cdot x(3)+x(7)+x(17) &\equiv 4852 && \pmod{8640}\\
3\cdot x(2)+x(3)+2\cdot x(5) &\equiv 6398 && \pmod{8640}\\
x(2)+2\cdot x(7)+x(17) &\equiv 3701 && \pmod{8640}\\
x(2)+2\cdot x(3)+x(5)+x(7)+x(11) &\equiv 8048 && \pmod{8640}
\end{aligned}$$

– az egyenletrendszert megoldva a következő megoldásokat kapjuk:

$$\begin{aligned}
&x(2)=7558 \quad x(3)=5852 \quad x(5)=1896\\
&x(7)=207 \quad x(11)=3963 \quad x(13)=\alpha \quad x(17)=4369,
\end{aligned}$$

ahol α tetszőleges értéket jelent.

Legyen $y=2007$ véletlenszerűen generált érték, ekkor

$$\begin{aligned}
&14\cdot 815^{2007}\equiv 6034 \pmod{8641}\\
&6034=2\cdot 7\cdot 431.
\end{aligned}$$

Mivel a 431 prímtényező, nem szerepel a B faktorbázis elemei között, nem tudjuk meghatározni az x értékét, újabb y értéket kell generálni. Legyen $y=2596$ egy másik véletlenszerűen generált érték, ekkor

$$\begin{aligned}
&14\cdot 815^{2596}\equiv 6600 \pmod{8641}\\
&6600=2^3\cdot 3\cdot 5^2\cdot 11.
\end{aligned}$$

Mivel 6600 minden prímtényezője szerepel a B faktorbázis elemei között, meghatározhatjuk az x értékét:

$$\begin{aligned}
6600 &= 2^3\cdot 3\cdot 5^2\cdot 11=(815^{7558})^3\cdot 815^{5852}\cdot (815^{1896})^2\cdot 815^{3963}\\
x &\equiv (3\cdot 7558+5852+2\cdot 1896+3963)-2596\equiv\\
&\equiv 7765 \pmod{8640}.
\end{aligned}$$

Tehát a keresett diszkrét logaritmus értéke: 7765, azaz $815^{7765}\equiv 14 \pmod{8641}$.

3. FEJEZET

KRIPTOGRÁFIAI ALAPFOGALMAK

A kriptográfia a történelem korábbi szakaszában, más civilizációknál, például görögöknél, spártaiaknál stb. a féltett információ, egy adott üzenet titkosítását jelentette, majd ezen titkos üzenet cseréjét a megfelelő felek között. Napjainkban, az elektronikus kommunikáció világában a kriptográfia további problémák megoldására is alkalmas, mint például a kommunikáló felek azonosítása, üzenetek sértetlenségének az ellenőrzése, dokumentumok elektronikus úton való hitelesítése stb.

A kriptográfiai eljárások leírásakor számos olyan szakszóval találkozunk, melyeknek pontos jelentését fontos tudni, éppen ezért most értelmezünk közülük néhányat, feltüntetve az angol megnevezéseket is:

- **rejtjelezés, titkosítás (encryption)**: az az eljárás, mely során az információt titkosítjuk;
- **visszafejtés (decryption)**: az az eljárás, mely során a titkosított információt visszafejtjük;
- **nyílt szöveg (plaintext)**: az az információ, amelyet szeretnénk titkosítani;
- **titkosított szöveg (ciphertext)**: az az információ, amelyet titkosítottunk;
- **kulcs (key)**: az az információ, mely segítségével a rejtjelezés, adott esetben a visszafejtés történik, és mely információ hiányában mindezek az eljárások nem valósíthatóak meg;
- **nyilvános kulcs (public key)**: az az információ, mely segítségével, adott esetben a rejtjelezést, vagy az aláírás ellenőrzését végezzük;
- **titkos kulcs (private key)**: az az információ, mely segítségével, adott esetben a visszafejtést vagy az aláírás előállítsát végezzük;
- **ábécé (alphabet)**: egy véges elemszámú halmaz, mely elemei különböző szimbólumok vagy betűk. A nyílt szöveg, illetve titkos szöveg szimbólumai csak az alkalmazott ábécé szimbólumaiból vehetik fel értékeiket. A kriptorendszereknél alkalmazott ábécé például lehet a $\{0, 1\}$, az angol ábécé betűinek halmaza, vagy az ASCII szimbólumok halmaza stb.

A modern számítástechnikában a kriptográfián belül számos olyan részterületet különböztethetünk meg, melyek mindegyike különböző tudományágnak is tekinthető.

A **titkos kulcsú vagy szimmetrikus kriptográfia** olyan titkosítási rendszerek leírásával foglalkozik, ahol a küldő és a fogadó fél ugyanazt a kulcsot használja az információ titkosítására, illetve visszafejtésére.

A **nyilvános kulcsú vagy aszimmetrikus kriptográfia**, olyan rendszereket foglal magába, ahol két kulcsot használnak, egy publikus és egy privát, vagy titkos kulcsot. Ahogyan a nevük is mutatja a publikus kulcsot bárki ismerhet, míg a privát kulcsot csak az arra feljogosított személy birtokolhatja. A két kulcs szorosan kapcsolódik egymáshoz, és a privát kulcs alapján nem lehet meghatározni a publikus kulcsot. Például az információ titkosításához, a tulajdonképpeni kulcscseréhez a nyilvános kulcsot, míg a visszafejtéséhez a titkos kulcsot használják. Digitális aláírások esetében az aláírás létrehozásakor a privát kulcs, az aláírás ellenőrzésekor a publikus kulcs kerül alkalmazásra.

A **kriptoanalízis** a létező kriptorendszerek gyenge pontjait, a titkosított információ nem legális úton való visszafejtését stb. kísérli meg meghatározni.

A **kriptográfiai primitívek** azok az alap kriptográfiai algoritmusok, amelyeken a létező kriptográfiai rendszerek információbiztonsága alapszik. Ilyenek a titkosító algoritmusok, a hash függvények, az üzenethitelesítő kódok, a digitális aláírások, stb.

A **kriptográfiai protokollok** a létező kriptográfiai algoritmusok gyakorlati alkalmazásának módját határozzák meg, például a biztonságos adattároláskor, az internetes kommunikációban, digitális fizetéskor, hogyan alkalmazzuk a kriptográfiai primitíveket. A gyakorlatban leginkább ismert protokollok a PGP, TLS, amelyeknek részletesebb leírását a 7. fejezetben ismertetjük. A megfelelő kriptográfiai primitívek helyes használatával a rendszerek az elvárt biztonsági szintet tudják biztosítani.

Léteznek olyan kriptográfiai protokollok, melyek a titkos kulcsú, mások a nyilvános kulcsú és a harmadik, a **hibrid rendszerek** alkalmazását írják elő. A nyilvános kulcsú kriptográfiánál alkalmazott algoritmusok jóval több és nagyobb számolási kapacitást igényelnek, mint a titkos kulcsú kriptorendszerek algoritmusai, a titkos kulcsú rendszerek esetében meg nem mindig oldható meg a titkos kulcs felek közti megosztása. Ezért a gyakorlatban kombinálni szokták őket egymással, oly módon, hogy a hitelesítést és a kulcscserét nyilvános kulcsú kriptográfiai algoritmussal végzik, míg a tulajdonképpeni üzenet titkosítását titkos kulcsú kriptográfiai eljárással, lásd [12].

A továbbiakban a titkos kulcsú kriptográfiáról, majd a nyilvános kulcsú titkosító rendszerekről beszélünk, utána a hash függvényekről, majd a digitális aláírásokat ismertetjük, végül a fontosabb kriptográfiai protkollokat mutatjuk be.

4. FEJEZET

TITKOS KULCSÚ KRIPTOGRÁFIA

A titkos kulcsú vagy szimmetrikus kriptográfia azoknak a titkosítási rendszereknek a csoportját jelenti, ahol a titkosításhoz alkalmazott kulcs ugyanaz, mint a visszafejtéshez alkalmazott kulcs, vagy ez utóbbi könnyűszerrel meghatározható a titkosító kulcs ismerete alapján. Gyakran a kommunikáló felek közötti titkosítási folyamatot egy információcsere előzi meg, mely során a felek megállapodnak az alkalmazásra kerülő kulcsban. Ez az információcsere a publikus kulcsú kriptográfia algoaritmusainak az alkalmazásával valósítható meg.

A titkos kulcsú rendszerek általában két nagy csoportba osztályozhatóak, az első az inkább történelmi és pedagógiai szempontból fontos csoport, a klasszikus kriptorendszerek, az első ismert titkosítók csoportja, a második a modern kriptorendszerek csoportja. Ez utóbbiak alkalmazása manapság nagy jelentőséggel bír.

Feltételezve, hogy a kommunikációban részt vevő két legális fél Alice és Bob (mely elnevezések az angolszász irodalomból kerültek át), akkor egy titkos kulcsú kriptorendszer egyszerűsített protokollja a következő lépéssorozatokból áll:

4.1. protokoll.

1. Alice és Bob megegyeznek egy szimmetrikus kriptorendszerben, azaz egy titkosítási eljárásban.
2. Alice és Bob megegyeznek a kulcsban.
3. Alice a megállapodott kulccsal és titkosítási eljárással titkosítja a nyílt szöveget, létrehozva ezáltal a titkos szöveget.
4. Alice elküldi a titkos szöveget Bobnak.
5. Bob a megállapodott kulccsal és titkosítási eljárással visszafejti a titkos szöveget.

A felmerülő problémák a következők lehetnek:

– Az Alice és Bob közötti kulcscsere során a kulcs egy illetéktelen fél által könnyedén eltulajdonítható, melynek eredményeként a nyílt szöveg tartalma megismerhető, adott esetben megváltoztatható. Ezért a kulcs átviteléhez szintén titkosított kommunikációra van szükség. A szakirodalom ezt kulcscsere-problémának is nevezi.

– Ha egy számítógép-hálózatban szeretnénk a kommunikáló felek biztonságos adatátvitelét megoldani, akkor minden kommunikálni szándékozó pár számára biztosítani kell a megfelelő titkos kulcsot. Ez

már 100 felhasználó esetében is nagyszámú kulcs kezelésének feladatát vonja maga után, aminek biztosítása komoly feladatot jelent.

4.1. Klasszikus titkos kulcsú kriptorendszerek

Mivel a történelem során, elsősorban a diplomáciai és katonai életben, nap mint nap szükség volt titkosítási eljárásokra, nagyon sok titkos kulcsú kriptorendszer látott napvilágot. Ezeket a klasszikus titkos kulcsú kriptorendszereket, titkosítókat manapság már jól kidolgozott módszerek alapján könnyen fel lehet törni.

4.1.1. A Caesar-titkosító és variációi

A Caesar-titkosító egyike a legrégebbi titkosítási rendszereknek, melyet manapság már csak oktatási célból alkalmaznak. Számos variációja ismert. Ezek közül most kettő kerül bemutatásra, az eredeti Caesar-titkosító és a Keyword Caesar. Az eredeti Caesar-rendszert Gaius Julius Caesar, ókori római hadvezér és politikus alkalmazta titkosítási eljárásokban. A rendszer részletesebb leírását megtaláljuk a [25] könyvben.

4.1.1.1. A klasszikus Caesar-titkosító

Feltételezzük, hogy a nyílt és titkosított szöveg az angol ábécé betűiből áll. Mivel az angol ábécé elemszáma 26, ezáltal 26 különböző értéket titkosíthatunk. A számolási folyamat megkönnyítése végett minden betűt egy egész számkód azonosít, mely értékeket az ASCII kódok alapján, vagy a következő számkódtáblából olvashatjuk le. A táblázat első sora a számkódokat határozza meg, második sora pedig kódokhoz tartozó betűket.

0	1	2	3	4	5	6	7	8	9	10	11	12
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*

13	14	15	16	17	18	19	20	21	22	23	24	25
N	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*

Minden betűt egyenként titkosítunk egy másik betűvel, azaz a betűket sorra helyettesítjük a hozzá rendelt betűkóddal, egy betűhöz mindig ugyanazt a betűkódot rendelve. Hogy melyik betű melyikkel helyettesítődik, azt a k kulcs határozza meg, mely az eredeti rendszer esetében egy pozitív egész szám. Egy betűt tehát mindig a betűhöz képest k pozícióval jobbra elhelyezkedő betűvel fogjuk helyettesíteni, és feltételezzük, hogy a betűk körkörösen helyezkednek el, azaz a Z betű után újból az A betű következik. Matematikailag a következő képlet határozza meg a p betű számkódjának c titkos számkódját:

$$c = (p + k) \pmod{26}.$$

A visszafejtési képlet ennek a fordítottja, feltételezve, hogy a titkosított betű értéke c, akkor a matematikai képlet a következő:

$$p = (c - k) \pmod{26}.$$

A képlet alapján könnyűszerrel megszerkeszthető egy titkosítási tábla: a már korábban megadott számkódtáblát kiegészíthetjük egy újabb sorral, melybe a jobbra eltolt ábécé betűit írjuk. Az így kapott rejtjelező táblából pedig könnyedén leolvashatóak a betűkhöz tartozó titkos rejtjelek.

4.1. példa. Ha a kulcs 5, akkor a következő titkosítási táblát kapjuk:

0	1	2	3	4	5	6	7	8	9	10	11	12
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
F	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*

13	14	15	16	17	18	19	20	21	22	23	24	25
N	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
S	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*	*D*	*E*

Ha a nyílt szöveg a következő:

$$THISISAPLAINTEXT,$$

akkor a titkosított szöveg a következő:

$$YMNXNXFUQFNSYJCY,$$

ahol tehát a T betű Y rejtjelét a következő összefüggés adja:

$$(19 + 5) = 24 \pmod{26}.$$

A kulcshalmaz elemszáma 26, és mivel a rendszer feltöréséhez elegendő kipróbálni az összes lehetséges kulcs értékét, ezért a feltörés triviális.

4.1.1.2. A Keyword Caesar-titkosító

A kulcs egy szó és egy egész szám lesz. A kulcsszóban szereplő betűk nem ismétlődhetnek. E két értékből a következő módon építjük fel a rejtjelező táblát: a számkód tábla harmadik sorában az egész szám által mutatott pozíciótól kezdve beírjuk a megadott kulcsszót, majd a többi pozícióra sorra beírjuk az ábécé fennmaradó betűit, ábécésorrendben. Ha elérkezünk az utolsó oszlopba, akkor körkörösen, az első pozíciótól kezdve folytatjuk a betűk beírását.

4.2. példa. Ha a kulcs a ($CRYPTO$, 5), akkor a következő titkosítási táblát kapjuk:

0	1	2	3	4	5	6	7	8	9	10	11	12
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
V	*W*	*X*	*Z*	**C**	**R**	**Y**	**P**	**T**	**O**	*A*	*B*	*D*

13	14	15	16	17	18	19	20	21	22	23	24	25
N	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
E	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*Q*	*S*	*U*

Ha a nyílt szöveg a következő:

$$THISISAPLAINTEXT,$$

akkor a titkosított szöveg a következő:

$$KPTJTJVGBVTEKCQK.$$

A rendszerben alkalmazható kulcshalmaz elemszáma nagyon nagy, ezért a feltörés lehetetlen az összes lehetséges kulcs kipróbálásával.

Mindkét rendszer esetében a titkosított szöveg megőrzi a nyílt szövegben levő betűgyakoriságot, ezért egyszerű betűgyakoriság-ellenőrzéssel fel lehet őket törni. Az angol ábécé betűgyakoriság-táblázatát a következő táblázatból lehet leolvasni, ahol a betűk mellé írt számok a betűk előfordulását fejezik ki az angol nyelvben, százalékban.

E	12,31	*L*	4,03	*B*	1,62
T	9,59	*D*	3,65	*G*	1,61
A	8,05	*C*	3,20	*V*	0,93
O	7,94	*U*	3,10	*K*	0,52
N	7,19	*P*	2,29	*Q*	0,20
I	7,18	*F*	2,28	*X*	0,20
S	6,59	*M*	2,25	*J*	0,10
R	6,03	*W*	2,03	*Z*	0,09
H	5,14	*Y*	1,88		

4.1.2. Az affin kriptorendszer

Ennél a rendszernél is feltételezzük, hogy a nyílt és titkosított szöveg betűi az angol ábécé betűinek halmazából állnak, és hasonló számkódokat rendelünk minden betűhöz, mint a Caesar-rendszereknél. Továbbá itt is betűnként titkosítunk, egy betűnek mindig ugyanazt a betűkódot feleltetve meg. A kulcs egy számpár, jelöljük: (a, b)-vel, ahol a következő megszorításokat tesszük: $0 \leq a, b \leq 26$ és $gcd(a,\ 26) = 1$. A rendszer részletesebb leírását megkapjuk a [25] könyvben.

Feltételezve, hogy p a nyílt szöveg egy betűjének számkódja, és c a neki megfelelő titkosított számkód, a titkosítás képlete a következő lesz:

$$c = (a \cdot p + b) \pmod{26}.$$

Ha a^{-1} -el jelöljük az a szám 26 szerinti multiplikatív inverzét, azaz ha fennáll a következő összefüggés:

$$a \cdot a^{-1} = 1 \pmod{26},$$

akkor a visszafejtési képlet a következő lesz:

$$p = (a^{-1} \cdot c - a^{-1} \cdot b) \pmod{26},$$

4.3. példa. Ha a kulcs $(5, 2)$, akkor a titkosítási képlet a következő lesz:

$$c = (5 \cdot p + 2) \pmod{26}.$$

Ha a nyílt szöveg a következő:

$$AMATHEMATICIAN,$$

akkor a titkosított szöveg:

$$CKCTLWKCTQMQCP,$$

ahol a titkosított szöveg első 6 karakterét a következőképpen határoztuk meg:

$$\begin{array}{lllllll} A & -> & C: & (5 \cdot 0 + 2) & = & 2 & \pmod{26} \\ M & -> & K: & (5 \cdot 12 + 2) & = & 10 & \pmod{26} \\ A & -> & C: & (5 \cdot 0 + 2) & = & 2 & \pmod{26} \\ T & -> & T: & (5 \cdot 19 + 2) & = & 19 & \pmod{26} \\ H & -> & L: & (5 \cdot 7 + 2) & = & 11 & \pmod{26} \\ E & -> & W: & (5 \cdot 4 + 2) & = & 22 & \pmod{26} \end{array}$$

Alkalmazva a visszafejtési képletet:

$$p = (21 \cdot c - 21 \cdot 2) \pmod{26},$$

ahol

$$5 \cdot 21 = 1 \pmod{26},$$

visszakapjuk a nyílt szöveget.

A rendszerben alkalmazható kulcsok számát a következőképpen becsülhetjük meg: az a értéke 12 különböző értéket vehet fel, mivel korábban feltettük, hogy $gcd(a,\ 26) = 1$. Ezek után, ha figyelembe vesszük, hogy a b értéke 26 különböző értéket vehet fel, és az $a = 1, b = 0$ kulcs párost ki kell zárni, akkor a lehetséges kulcsok száma: $12 \cdot 26 - 1 = 311$. Mivel a kulcshalmaz elemszáma kicsi, a rendszer feltörése ebben az esetben is lehetséges az összes kulcs kipróbálásával.

A tikosított szövegben a rendszer megőrzi a nyílt szöveg betűgyakoriságát, ezért egy másik alkalmazható feltörési mód a betűgyakoriság-ellenőrzés. Ha a betűgyakoriság alapján sikerül megállapítani két betű helyes behelyettesítési értékét, akkor a következő kongruenciarendszer megoldásával, ahol az ismeretlenek a, b, megállapítható a titkosításhoz használt (a, b) kulcs:

$$x_1 \cdot a + b = y_1 \pmod{26}$$
$$x_2 \cdot a + b = y_2 \pmod{26}.$$

4.4. példa. Ha a legtöbbször előforduló két betű a nyílt szövegben az A és M, a titkosított szövegben pedig a nekik megfelelő betűk a C és K, akkor a fenti rendszerben a következő helyettesítéseket végezhetjük el:

$$x_1 = 0 \quad y_1 = 2$$
$$x_2 = 12 \quad y_2 = 10.$$

A megoldandó kongruenciarendszer ezek után a következő lesz:

$$0 \cdot a + b = 2 \pmod{26}$$
$$12 \cdot a + b = 10 \pmod{26}.$$

A kongruenciarendszer az $a = 5$ és $b = 2$, illetve az $a = 18$ és $b = 2$ megoldásokat szolgáltatja. A második megoldás nem tesz eleget a $gcd(a,\ 26) = 1$ feltételnek, mert $gcd(18,\ 26) = 2$, így csak az első megoldás, azaz az $(5,\ 2)$ számpár fogadható el kulcsként.

4.1.3. Mátrixos affin kriptorendszerek

A mátrixos affin kriptorendszer a 4.1.2. fejezetben bemutatott affin kriptorendszer általánosítása. A fejezet keretén belül bemutatásra kerülő, Hill néven ismert kriptorendszer (1929) Lester S. Hill után kapta nevét. A mátrixos affin kriptorendszerek egyszerű leírása megtalálható a [5] könyvben.

A nyílt és titkosított szöveg betűi az angol ábécé betűinek halmazából állnak, és minden betűhöz, az előző rendszerekhez hasonlóan, számkódokat rendelünk. A titkosítási eljárás során egyszerre d darab betűnek határozzuk meg a rejtjelét, újabb d darab betűt létrehozva. Ha m a kulcs, ahol m egy $d \times d$ méretű mátrix 0 és 25 közötti értékekkel, akkor a titkosítási folyamatot az alábbi matematikai képlet írja le:

$$c = m \cdot p \pmod{26},$$

ahol p egy d elemű oszlopvektor, ami megfelel a d darab betű számkódjának, c pedig a d darab titkosított betű számkódja, oszlopvektorba helyezve. A szükséges aritmetikai műveleteket, amint látható, $\pmod{26}$ szerint végezzük. Az m mátrix invertálható kell legyen, ami azt jelenti, hogy létezik m^{-1}, a következő tulajdonsággal:

$$m \cdot m^{-1} = e \pmod{26},$$

ahol e a $d \times d$ méretű egységmátrix. A kongruencia megoldhatóságának szükséges és elégséges feltétele az, hogy a mátrix determinánsának, $\det m$-nek létezzen inverze $\pmod{26}$ szerint, azaz a következő kongruencia megoldható legyen:

$$\det m \cdot \det m^{-1} = 1 \pmod{26}.$$

Ahhoz, hogy megadhassuk az m^{-1}-et meghatározó összefüggést, előbb definiáljuk az adjungált mátrixot:

$$\text{adj } m = (m^{min})^T,$$

ahol az m^{min}-t aldeterminánsmátrixnak nevezzük, i sorának és j oszlopának m_{ij}^{min} elemét pedig a következőképpen határozzuk meg:

$$m_{ij}^{min} = (-1)^{i+j} \cdot \det m_{i,j}.$$

$m_{i,j}$-vel jelöltük azt a mátrixot, melyet az m mátrixból nyertünk úgy, hogy töröltük a mátrix i-dik sorát és j-dik oszlopát. Továbbá m^T-vel jelöltük az m mátrix transzponáltját, azaz azt a mátrixot, melyet az eredetiből úgy nyertünk, hogy tükröztük az elemeket a főátlóra.

Ezek után megadható az m^{-1}-et meghatározó összefüggés:

$$m^{-1} = (\det m)^{-1} \cdot \text{adj } m \pmod{26}.$$

A bevezetett jelölésekkel élve a visszafejtést a következő képlet adja:

$$p = m^{-1} \cdot c \pmod{26}.$$

4.5. példa. Határozzuk meg $d = 2$, $m = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix}$ esetében m^{-1} értékét.

Az m mátrix determinánsa: $\det m = 1 \cdot 3 - (-2) \cdot 3 = 9$. Mivel $gcd(9,\ 26) = 1$, következik, hogy létezik $\det m$-nek inverze (mod 26) szerint. Megoldva a $9 \cdot (\det m)^{-1} = 1 \pmod{26}$ kongruenciát, azt kapjuk, hogy $(\det m)^{-1} = 3 \pmod{26}$.

Az adjungált mátrix pedig adj $m = \begin{pmatrix} 1 & -3 \\ 2 & 3 \end{pmatrix}$.

Ezek után tehát

$$m^{-1} = 3 \cdot \begin{pmatrix} 1 & -3 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 3 & -9 \\ 6 & 9 \end{pmatrix}.$$

4.6. példa. Ha a kulcs az előző példánál vett mátrix

$$m = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix}$$

és az nyílt szöveg:

$$AMATHEMATICIAN,$$

akkor a titkosított szöveg:

$$KMFTHQKCDWEENN,$$

ahol az első két betűpár rejtjelezése a következőképpen történt:

$$m \cdot p_1 = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix} \cdot \begin{matrix} A \\ M \end{matrix} = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 12 \end{pmatrix} = \begin{pmatrix} 10 \\ 12 \end{pmatrix} = \begin{matrix} K \\ M \end{matrix},$$

$$m \cdot p_2 = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix} \cdot \begin{matrix} A \\ T \end{matrix} = \begin{pmatrix} 3 & 3 \\ -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{pmatrix} 5 \\ 19 \end{pmatrix} = \begin{matrix} F \\ T \end{matrix}.$$

Az első két betűpár visszafejtése pedig a következő:

$$\begin{aligned} m^{-1} \cdot c_1 &= \begin{pmatrix} 3 & -9 \\ 6 & 9 \end{pmatrix} \cdot \begin{matrix} K \\ M \end{matrix} = \begin{pmatrix} 3 & -9 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 12 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 12 \end{pmatrix} = \begin{matrix} A \\ M \end{matrix}, \\ m^{-1} \cdot c_2 &= \begin{pmatrix} 3 & -9 \\ 6 & 9 \end{pmatrix} \cdot \begin{matrix} F \\ T \end{matrix} = \begin{pmatrix} 3 & -9 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 19 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{matrix} A \\ T \end{matrix}. \end{aligned}$$

Ha sikerül megállapítani az m kulcs sor (oszlop) számát, azaz a d-t, és ha sikerül két d darab betűt tartalmazó eredeti-titkosított párt azonosítani, akkor a rendszer feltörése viszonylag egyszerű.

4.7. példa. Feltételezzük, hogy $d = 2$ és ismert a nyílt szöveg $LEFT$ és a neki megfelelő titkosított szöveg $LBUZ$, azaz a talált két eredeti-titkosított pár az, hogy:

$$\begin{matrix} LE- > LB \\ FT- > UZ. \end{matrix}$$

A rejtjelezési képletnek megfelelően kapjuk, hogy:

$$m \cdot \begin{pmatrix} 11 \\ 4 \end{pmatrix} = \begin{pmatrix} 11 \\ 1 \end{pmatrix} \quad \text{és} \quad m \cdot \begin{pmatrix} 5 \\ 19 \end{pmatrix} = \begin{pmatrix} 20 \\ 25 \end{pmatrix}.$$

Ezek után a kulcs a következő:

$$\begin{aligned} m &= \begin{pmatrix} 11 & 20 \\ 1 & 25 \end{pmatrix} \cdot \begin{pmatrix} 11 & 5 \\ 4 & 19 \end{pmatrix}^{-1} = \begin{pmatrix} 11 & 20 \\ 1 & 25 \end{pmatrix} \cdot \begin{pmatrix} 25 & 3 \\ 18 & 9 \end{pmatrix} \\ &= \begin{pmatrix} 11 & 5 \\ 7 & 20 \end{pmatrix}, \end{aligned}$$

ahol legyen

$$S = \begin{pmatrix} 11 & 5 \\ 4 & 19 \end{pmatrix}$$

multiplikatív inverzét a következőképpen határoztuk meg:

$$\det S = 11 \cdot 19 - 4 \cdot 5 = 189 = 7 \pmod{26},$$

$$(\det S)^{-1} = 15 \pmod{26},$$

$$\text{adj } S = \begin{pmatrix} 19 & -4 \\ -5 & 11 \end{pmatrix}^T = \begin{pmatrix} 19 & -5 \\ -4 & 11 \end{pmatrix},$$

$$S^{-1} = (\det S)^{-1} \cdot \text{adj } S = 15 \cdot \begin{pmatrix} 19 & -5 \\ -4 & 11 \end{pmatrix} = \begin{pmatrix} 25 & 3 \\ 18 & 9 \end{pmatrix}.$$

4.1.4. A Vigenère-kriptorendszer

A Vigenère-kriptorendszert először 1553-ban írta le Giovan Batista Belaso. Később számos alkalommal újra felfedezték, többek között Blaise de Vigenère is adott egy leírást. Mind a mai napig a Vigenère nevet viseli, mivel tévesen neki tulajdonították a felfedezést. A rendszer részletesebb leírását megtalálhatjuk a [25] könyvben.

A Vigenère-rendszer polialfabetikus titkosító, ami azt jelenti, hogy több ábécét is használunk a rejtjelezés során, azaz egy titkosítandó betű többféleképpen is rejtjelezhető, és hogy melyik rejtjellel történik a helyettesítés, azt a rendszer megadott szabály szerint határozza meg. A nyílt szöveg betűgyakorisága ezáltal nem marad ugyanaz a titkosított szövegben, ami megnehezíti a betűgyakoriság-vizsgálaton alapuló kriptoanalízist. A titkosított szövegnek legfeljebb kisebb részeiben lehet betűgyakoriságot vizsgálni. Mondhatjuk azt is, hogy a Caesar-titkosító egy általánosítása. Feltételezzük, hogy a nyílt és titkosított szöveg az angol ábécé betűiből áll. A kulcs egy tetszőleges szó lesz, melynek hossza megadja, hogy hány, betűi pedig meghatározzák, hogy mekkora eltolást kell használni.

Feltételezve, hogy a kulcsszó első betűjének számkódja k_1, akkor az első eltolás azt jelenti, hogy minden betűt a hozzá képest k_1 pozícióval jobbra elhelyezkedő betűvel fogunk rejtjelezni. Ha a kulcsszó második betűjének számkódja k_2, akkor a második eltolása során minden betűt a hozzá képest k_2 pozícióval jobbra elhelyezkedő betűvel fogunk titkosítani, stb. A nyílt szöveg első betűjét az első eltolási szabály (első ábécé), második betűjét a második eltolási szabály (második ábécé) alapján titkosítjuk, majd ezt az eljárást folytatjuk, míg elfogynak a kulcsszó betűi, ekkor pedig újból az első ábécével folytatjuk a rejtjelezést.

A rendszerben alkalmazható kulcsok száma nagy, ezért az összes kulcs kipróbálásával járó feltörési mód nem ad járható utat. Észrevehető viszont,

hogy a kulcsszó hossza által meghatározott részekben a nyílt szöveg betűgyakorisága megmarad. Fel kell tehát tördelni a titkosított szöveget akkora részekre, amekkorát a kulcsszó hossza meghatároz, majd mindegyik részből az első betűkkel egy, a második betűkkel egy második csoportot stb. formálni, majd ezeken a csoportokon belül lehet betűgyakoriságot vizsgálni. A kulcsszó hosszának a meghatározásához F. W. Kasiski módszerét (lásd [25]) lehet alkalmazni, mely a titkosított szövegben található ismétlődő szövegrészek alapján következtet a kulcsszó hosszára.

4.8. példa. Ha a kulcs $CRYPTO$, melynek hossza 6, akkor 6 különböző ábécét fogunk használni, melyeket egymás után sorrendben alkalmazunk a nyílt szövegen. Ha megszerkesztjük a rejtjelező táblát, akkor a következő táblázatot kapjuk:

0	1	2	3	4	5	6	7	8	9	10	11	12
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
C	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
R	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*	**D**
Y	*Z*	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*
P	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*
T	*U*	*V*	*W*	*X*	*Y*	*Z*	**A**	*B*	*C*	*D*	*E*	*F*
O	*P*	*Q*	*R*	**S**	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*

13	14	15	16	17	18	19	20	21	22	23	24	25
N	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
P	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*
E	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*
L	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*
C	*D*	*E*	*F*	*G*	*H*	**I**	*J*	*K*	*L*	*M*	*N*	*O*
G	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*
B	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*

Ha a nyílt szöveg a következő, feltüntetve az első sorban a betűk sorszámát:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	*M*	*A*	*T*	*H*	*E*	*M*	*A*	*T*	*I*	*C*	*I*	*A*	*N*,

akkor a titkosított szöveg:

$$C \quad D \quad Y \quad I \quad A \quad S \quad O \quad R \quad R \quad X \quad V \quad W \quad C \quad E,$$

ahol az 1, 7, 13-ik, azaz az A, M, A betűk az első, a 2, 8, 14-ik, azaz M, A, N betűk a második, a 3, 9-ik, azaz A, T betűk a harmadik ábécével vannak titkosítva stb.

4.1. megjegyzés. A rejtjelező táblában vastagon írtuk az első 6 betűnek megfelelő rejtjelet.

4.1.5. A Playfair-kriptorendszer

A Playfair-kriptorendszert Charles Wheatstone találta fel 1854-ben, de mivel a rendszer alkalmazását leginkább Lord Playfair szorgalmazta, ezért az ő nevét viseli. A rendszer részletesebb leírását megtalálhatjuk a [25] könyvben.

A rejtjelezési eljárás során mindig betűpároknak feleltetünk meg betűpár rejtjeleket. A kulcs egy szó, mely alapján egy rejtjelező táblát, egy 5×5-ös mátrixot szerkesztünk. A mátrix első sorába beírjuk a kulcsszó betűit. Ha a kulcsszó hosszabb, mint 5 karakter, akkor a következő sorral folytatjuk, és ha elfogytak a kulcsszó karakterei, akkor az ábécé fennmaradó betűivel egészítjük ki a táblát. A titkosítást az angol ábécé 25 betűje felett végezzük, kizárjuk a legkisebb gyakorisággal előforduló betű, a J betű titkosítását. A kulcsszó nem tartalmazhat ismétlődő betűket, és természetesen nem tartalmazhatja a J betűt sem.

4.9. példa. Ha a kulcs $CRYPTO$, akkor a következő lesz a rejtjelező tábla:

$$\begin{array}{ccccc} C & R & Y & P & T \\ O & A & B & D & E \\ F & G & H & I & K \\ L & M & N & Q & S \\ U & V & W & X & Z \end{array}$$

A nyílt szövegen előfeldolgozást végzünk. Kettesével csoportosítjuk a betűket, és minden J betűt I-re cserélünk. Ezután az egymás után előforduló azonos betűk közé X-et, illetve Z-t szúrunk, és mivel a nyílt szöveg páros számú betűt kell tartalmazzon, szükség esetén a szöveget kiegészítjük az X betűvel.

A rejtjelezési algoritmus 3 esetet különböztet meg, attól függően, hogy a titkosítandó betűpár a rejtjelező táblában hol található.

1. Ha azonos sorban vannak, akkor a betűpárnak megfelelő rejtjel az ugyanabban a sorban levő, közvetlenül a titkosítandó betűk utáni oszlopban levő betűk lesznek. Az 5. oszlop után újból az első oszlopot kell venni.

   4.10. példa.

   $$WX -> XZ$$
   $$GK -> HF$$

   A fenti és következő példákban rejtjelező táblának a 4.9. példánál megadott táblázatot vesszük.
2. Ha azonos oszlopban vannak, akkor a betűpárnak megfelelő rejtjel az ugyanabban az oszlopban levő, a titkosítandó betűk utáni sorban levő betűk lesznek.

4.11. példa.

$$MA- > VG$$
$$KE- > SK$$

3. Ha nincsenek se azonos sorban, se azonos oszlopban, akkor a titkosítást a betűpár által meghatározott belső négyzet határpontjain levő betűk alapján végezzük. Vigyázni kell a határpontok sorrendjének megállapítására: mindig a sor mentén határozzuk meg az első határpontot, és nem az oszlop mentén.

4.12. példa.

$$RE- > TA \text{ és nem } AT$$
$$LA- > MO$$
$$NO- > LB$$

4.13. példa. A 4.9. példánál feltüntetett rejtjelező tábla alapján, ha a nyílt szöveg a következő:

$$AM \quad AT \quad HE \quad MA \quad TI \quad CI \quad AN,$$

akkor a titkosított szöveg:

$$GV \quad ER \quad KB \quad VG \quad PK \quad PF \quad BM.$$

A rendszer megőrzi a titkosított szövegben és a nyílt szövegben levő betűpárok gyakoriságát, ezért az itt alkalmazható feltörési mód a betűpárok gyakoriságának a leellenőrzése. Az angol ábécé betűpárjainak a gyakoriságtáblázatát a következő táblázatból lehet leolvasni, ahol a betűpárok mellé írt számok a betűpárok előfordulását fejezik ki százalékban.

TH	$6,3$	AR	$2,0$	HA	$1,7$
IN	$3,1$	EN	$2,0$	OU	$1,4$
ER	$2,7$	TI	$2,0$	IT	$1,4$
RE	$2,5$	TE	$1,9$	ES	$1,4$
AN	$2,2$	AT	$1,8$	ST	$1,4$
HE	$2,2$	ON	$1,7$	OR	$1,4$

4.1.6. Kódkönyv

A kriptográfiában a kódkönyv egy olyan dokumentumot jelöl, mely szótárhoz hasonló formában felsorolja a titkosítandó szavakat és a nekik megfelelő kódokat. A kriptorendszerről részletesebb leírást találunk a [20] könyvben. A kódkönyv alapján szavakat, szócsoportokat, akár mondatokat is helyettesíthetünk értelmes szavakkal, betűcsoportokkal vagy akár értelmetlen szavakkal is. A kódkönyv egy-egy példánya a két kommunikáló félnél,

ahogy a 4.1. protokollban neveztük őket, Alice-nál és Bobnál található, amely alapján aztán a titkosítás, visszafejtés történik. Sok tény amellett szól, hogy kódkönyv segítségével titkosított szöveget nehezebb feltörni, viszont emellett számos hátránya is van a kódkönyv használatának: a titkosítási folyamat automatizálása nem egyszerű, és a kódkönyv frissítését is gyakran meg kell valósítani, hogy elkerüljük az ismétlődő szavak gyakoriságának vizsgálatán alapuló feltörési eljárást. Éppen ezért a számítógépes kommunikációban nem a használt titkosítási eljárások közé tartozik.

4.14. példa. Ha a kódkönyv bizonyos szavai a következőképpen felelnek meg egymásnak:

Eredeti	Titkosított
fogunk	megyünk
⋮	⋮
délután	tavasszal
⋮	⋮
támadni	halászni
⋮	⋮
visszavonulni	sétálni

és ha a nyílt szöveg a következő:

Délután fogunk támadni.

akkor a titkosított szöveg a következő lesz:

Tavasszal megyünk halászni.

4.2. Modern titkos kulcsú kriptorendszerek

A modern szimmetrikus kriptorendszereket két csoport szokták felosztani. Eszerint megkülönböztetünk folyam titkosító (stream chiper) és blokk titkosító rendszereket (block chiper). A folyamtitkosító kriptorendszerekben a nyílt szöveg bitjeit vagy bájtjait egyenként titkosítják, ami azt jelenti, hogy a nyílt szöveg bitjein (bájtjain) és a kulcs bitjein (bájtjain) egy XOR (értelmezését lásd lentebb) műveletet végzünk. A blokktitkosító eljárások ezzel szemben fix hosszúságú bitsoron (blokkon), például 128 biten végzik a titkosítási folyamatot, azaz blokkonként alkalmazzák a titkosítási eljárást.

4.2.1. A „one-time pad” kriptorendszer

A „one-time pad” folyamtitkosító kriptorendszer 1917-ben látott napvilágot. Leginkább elméleti szempontból van jelentősége, mégpedig azért, mert

arra a kérdésre ad „igen” választ, hogy létezik-e tökéletes titkosítási eljárás, azaz olyan eljárás, melyben a titkosított szöveg ismerete alapján semmiféle információt nem lehet megtudni a nyílt szöveg tartalmáról.

Az algoritmus a nyílt szöveget a szöveg hosszával megegyező hosszúságú kulccsal titkosítja. A rendszer a Vernam-titkosító nyomán jött létre, melyet feltalálója, Gilbert Vernamról neveztek el. Míg a Vernam-titkosító esetében megengedett a kulcs többszöri felhasználása, a „one-time” rendszernél a kulcsot egyetlenegyszer szabad felhasználni, azaz minden titkosítás esetében új kulcsot kell generálni. A kulcsnak ez az egyszeri felhasználhatósága biztosítja a rendszer számára a tökéletes titkosító tulajdonságot.

Az algoritmus a titkosítási folyamat során meghatározza a nyílt szöveg és kulcs elemeinek moduláris összegét. Ha a nyílt szöveget kettes számrendszerbeli ábrázolásában dolgozzuk fel, akkor a moduláris összegzés a számítástechnikából jól ismert XOR műveletet fogja jelenteni, melynek jelölésére $\oplus$ fogjuk használni. A $\oplus$ művelet a 0 és 1 értékeken a következő eredményeket adja:

		$\oplus$
0	0	0
0	1	1
1	0	1
1	1	0

A kulcs elemei véletlenszerűen meghatározott bit értékek lesznek bináris típusú szöveg esetében, illetve 0 és 25 közötti számok, ha a nyílt szöveg elemei az angol ábécé betűi. A visszafejtés moduláris kivonást jelent az angol ábécé esetében és ugyancsak XOR műveletet, ha a nyílt szöveg bináris ábécében van megadva.

Gyakorlati megvalósíthatóság területén a rendszer számos nehézségbe ütközik, mégpedig:

- „jó”, véletlenszerűen előállított számsorokat, bitsorokat nehéz előállítani,
- a partnerek közötti kulcscserét hosszú szöveg esetén nem könnyű megvalósítani,
- problémát vet fel a kulcs megőrzése, vagy adott esetben megsemmisítése.

4.15. példa. Legyen a kulcs a következő véletlenszerűen előállított betűsor, ahol a táblázat második sorában feltüntettük a megfelelő számkódokat:

U	*I*	*E*	*R*	*G*	*F*	*K*	*J*	*H*	*S*	*A*	*D*	*I*	*H*
20	8	4	17	6	5	10	9	7	18	0	3	8	7

Ha a nyílt szöveg a következő:

A	*M*	*A*	*T*	*H*	*E*	*M*	*A*	*T*	*I*	*C*	*I*	*A*	*N*
0	12	0	19	7	4	12	0	19	8	2	8	0	13

akkor a titkosított szöveg:

U	U	E	K	N	J	W	J	A	A	C	L	I	U
20	20	4	10	13	9	22	9	0	0	2	11	8	20.

A tikosítás eredményét az első négy betű esetében a következőképpen határoztuk meg:

$$\begin{aligned} 0 + 20 &= 20 \pmod{26} \\ 8 + 12 &= 20 \pmod{26} \\ 4 + 0 &= 4 \pmod{26} \\ 17 + 19 &= 10 \pmod{26} \end{aligned}$$

Ha a kulcsot többször is felhasználjuk, akkor a rendszert nagyon könnyen fel lehet törni, mert már egy eredeti-titkosított párosból egyszerű moduláris kivonással meghatározható a kulcs.

4.16. példa. Ha tudjuk, hogy a nyílt szöveg

L	E	F	T
11	4	5	19

és a titkosított szöveg

F	M	J	K
5	12	9	10

akkor a kulcs a következőképpen határozható meg:

$$\begin{aligned} 20 &= (5 - 11) \pmod{26} \\ 8 &= (12 - 4) \pmod{26} \\ 4 &= (9 - 5) \pmod{26} \\ 17 &= (19 - 17) \pmod{26} \end{aligned}$$

azaz

U	I	E	R
20	8	4	17.

4.2.2. Blokk titkosítási módok

A blokkonkénti titkosítás esetében a nyílt szöveget meghatározott nagyságú blokkokra osztjuk és ezeken alkalmazzuk a titkosítási eljárást. Számos technikát különböztetünk meg, ezek közül felsorolunk egy párat, részletesebben lásd [12]:

- az ECB módban (electronic codebook) a nyílt szöveg blokkjait egymástól függetlenül titkosítjuk, a titkosított szöveg a titkosított blokkok konkatenációja lesz,
- a CBC módban (cipher-block chaining) egy blokk titkosítása az előző blokk titkosított értékétől függ,
- a CFB módban (cipher feedback) egy blokk titkosítása az előző blokk titkosított értékének egy részétől függ,
- az OFB módban (output feedback) egy blokk titkosítása nem függ az előző blokk titkosított értékétől, hanem az előző blokk értékétől.

4.2.3. DES (Data Encryption Standard)

A DES a FIPS (Federal Information Processing Standards) által 1976-ban közétett, hivatalosan elfogadott nemzetközi blokktitkosító kriptorendszer. Napjainkban azonban már nem biztonságos, elsősorban kis méretű, 56 bit hosszúságú kulcsa miatt: nem egészen 24 óra alatt fel lehet törni a DES kulcsokat. A triple DES változata azonban napjainkban is biztonságoz, viszont ez ellen is léteznek hatékony, elméletben kidolgozott feltörési módszerek. Az elmúlt években az AES (Advanced Encryption Standard) kriptorendszer váltotta fel. Ennek ellenére a DES most is a fontos kriptográfiai rendszerek közé tartozik, lásd [26].

A DES titkos kulcsú, blokktitkosító kriptorendszer, a {0,1} ábécé felett, ahol egy bemeneti blokk hossza 64 bit hosszúságú. A titkosítás során létrejövő kimeneti blokk hosszának mérete is 64 bit. A kulcs is 64 bit hosszúságú, bár az effektív kulcshossz 56 bit, mivel a 64 bitből a rendszer 8 bitet paritásellenőrzésre használ: mindegyik 8 bites részben az 1-es bitek száma páratlan kell legyen.

Kulcsgenerálás

A kezdeti kulcs alapján 16 különböző segédkulcsot, jelölje őket K_i, $(i = 1, 2, \ldots, 16)$, határozunk meg a következő lépéssorozattal:

1. A 64 bit hosszúságú kezdeti kulcsra alkalmazzuk a **PC-1 függvényt** (Permuted Choice) (lásd lentebb), ezáltal egy 56 bit hosszúságú bitsort kapunk, az első segédkulcsot: K_0-t.
2. K_0-t felosztjuk két 28 hosszúságú blokkra, ezáltal megkapjuk C_0, D_0-t.
3. Sorra meghatározzuk K_i-t, $(i = 1, 2, \ldots, 16)$, 16-szor ismételve a következő lépéssorozatot:
   a) meghatározzuk C_i-t, v_i darab balra történő bitrotációt alkalmazva C_{i-1}-en, ahol

$$v_i = \begin{cases} 1, & \text{ha } i = 1,\ 2,\ 9,\ 16 \\ 2, & \text{egyébként,} \end{cases}$$

   b) meghatározzuk D_i-t, v_i darab balra történő bitrotációt alkalmazva D_{i-1}-en,
   c) alkalmazzuk a **PC-2 függvényt** (lásd lentebb) (C_i, D_i)-re, megkapva ezáltal K_i-t:
      – $PC_1(b_1 b_2 \ldots b_{64}) = C_{i-1} || D_{i-1} = b_{57} b_{49} \ldots b_{36} b_{63} \ldots b_4$, kimenete 56 bit.
      – $PC_2(b_1 b_2 \ldots b_{56}) = b_{14} b_{17} \ldots b_{32}$, kimenete 48 bit.

A PC-1 függvény

Bal						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
Jobb						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

A PC-2 függvény

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Titkosítás

A titkosítás 16 lépéssorozatból áll, mely során felhasználjuk a kulcsgenerálásnál kapott segédkulcsokat, a

$$K_i,\ (i = 1, 2, \ldots, 16)\text{-ket.}$$

1. A bemeneten alkalmazunk egy kezdeti permutációt, az **IP permutációt** (Initial Permutation) (lásd lentebb).
2. A kapott bitsort 2 részre osztjuk, jelöljük őket: L_0 és R_0-val, ahol mindegyik rész 32 bit hosszúságú lesz.
3. Sorra meghatározzuk az L_i és R_i, $(i = 1, 2, \ldots, 16)$ értékeket, a következő képletek alapján

$$L_i = R_{i-1}, \quad R_i = L_{i-1} \oplus F(R_{i-1}),$$

ahol az F függvény értékének a meghatározása a következő lépésekből áll:

a) R_{i-1}-re alkalmazzuk az E **expansion függvényt** (lásd lentebb), ezáltal egy 32 hosszúságú bitsorból kapunk egy 48 hosszúságú bitsort: $E(R_{i-1})$,

b) meghatározzuk az

$$E(R_{i-1}) \oplus K_i$$

értéket, ahol K_i, $(i = 1, 2, \ldots, 16)$ a kulcsgenerálás során kapott i.-ik segédkulcs,

c) 8 egyenlő hosszúságú blokkra osztjuk a kapott bitsort:

$$B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8\text{-ra,}$$

ahol mindegyik B_i, $(i = 1, 2, \ldots, 8)$ blokk 6 bit hosszúságú lesz,

d) mindegyik B_i blokkra alkalmazzuk az **S-Box**, helyettesítő blokkok (substitution box) közül a megfelelő S_i, $(i = 1, 2, \ldots, 8)$ boxot (lásd lentebb), ezáltal 8 darab 4 bit hosszúságú bitsort kapunk, vagyis egy 32 bit hosszúságú bitsort, legyen ez C. $S_i(B)$-t, az S_i box alapján, ahol $B = b_1b_2b_3b_4b_5b_6$ a következőképpen határozzuk meg:

- b_1b_6 a sor-indexet határozza meg,
- $b_2b_3b_4b_5$ az oszlop-indexet határozza meg,
- $S_i(B)$, a megfelelő indexnél levő érték bináris alakja
- Pl. $S_1(\mathbf{1}11110\mathbf{0}) = 0101$, mert a 2. sor (melynek bináris alakja 10), 14. oszlop (melynek bináris alakja 1110) értéke 5, aminek bináris alakja 0101.

e) alkalmazzuk a P **permutációt** (lásd lentebb) a C bitsorra, megkapjuk ezáltal $F(R_{i-1})$-t.

4. Az utoljára kapott L_{16} és R_{16} párra alkalmazzuk az IP^{-1} **permutációt** (lásd lentebb).

A fenti leírásból látható, hogy az eljárás a titkosítás során egy kezdeti és egy végső permutációt: IP és IP^{-1} permutációt alkalmaz, mely permutációk tulajdonképpen egymás inverzei, és melyeknek nem annyira kriptográfiai szempontból, hanem inkább a hardveres implementációnál van jelentőségük. Az itt alkalmazott eljárás a Feistel-séma szerint történik, mely biztosítja, hogy a titkosítás és visszafejtés folyamata ne különbözzön.

Visszafejtés

A visszafejtés tehát annyiban különbözik a titkosítástól, hogy az alkalmazott segédkulcsokat fordított sorrendben kell venni.

Feltörés

Az algoritmus egyik feltörési stratégiája a brute-force, azaz az összes kulcs kipróbálása.

Az IP permutáció

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Az E expansion függvény

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

A P permutáció

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Az IP^{-1} permutáció

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Az S-boxok

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

4.2.4. AES (Advanced Encryption Standard)

Az AES, vagy másképp Rijndael, egy szimmetrikus blokktitkosító kriptorendszer, melyet az Egyesült Államok kormánya 2002-ben standardként fogadott el.

A rendszer kidolgozói két belga kriptográfus: Vincent Rijmen és Joan Daemen, a Rijndael elnevezés pedig kettőjük nevének összevonásából adódik. A szimmetrikus kriptorendszerek között a legnépszerűbb rendszerek egyike. A {0,1} ábécé felett titkosít, ahol egy blokk hossza 128 bit. A kulcs 128, 192 vagy 256 bit hosszúságú. Az algoritmus részletes leírását lásd a [10]-ben.

Az alábbi leírás azt az esetet mutatja be, mikor a titkosítandó blokk és a kulcs hossza is 128 bit. A kulcs mérete egyben az alap tárolóegységként használt 2 dimenziós tömb (state) oszlopszámát: $128/32 = 4$, a körkulcsok (round key) számát: 10, illetve a titkosítás során alkalmazott körök (round) számát: 10 is lerögzíti. A 192, illetve 256 bit hosszúságú kulcsok esetében ezek az értékek mások lesznek.

Kulcsgenerálás

A kezdeti kulcs alapján 11 különböző körkulcsot, jelöljük őket: RK_i-vel ($i = 0, 1, 2, \ldots, 10$), határozunk meg a következőképpen:

- A kulcsot egy 4×4-es, kétdimenziós tömbbe rendezzük, oszloponként töltve fel a tömböt, ahol a tömb egy-egy eleme egy bájtot fog tartalmazni, ezáltal meghatározzuk a 0-ik körkulcsot, RK_0 -t.
- RK_0-t oszloponként felosztjuk 4 szóra (word -ra): w_0, w_1, w_2, w_3, ahol minden szó 4 bájtból (32 bitből) fog állni. A fejezet további részében, ezek után, szó alatt mindig egy 4 bájtos egységet fogunk érteni.
- A további RK_i, $(i = 1, 2, \ldots, 10)$ körkulcsokat szavanként állítjuk elő, sorra meghatározva az nw_0, nw_1, nw_2, nw_3 szavakat, az RK_{i-1} w_0, w_1, w_2, w_3 szavai alapján.
- 10-szer ismételjük a következő lépéssorozatot:
  - $nw_0 = temp \oplus w_0$, ahol az $\oplus$ művelet a korábban bevezetett XOR-t jelenti a megadott biteken (lásd 4.2.1. fejezet), a w_0 RK_{i-1} megfelelő szava és $temp$ pedig a következő lépéssorozat eredménye:
    - $rw = \textbf{RotWord}(w_3)$, ahol w_3 RK_{i-1} megfelelő szava,
    - $sw = \textbf{SubWord}(rw)$,
    - $rcw = \textbf{Rcon}(i)$, ahol i a lépéssorozat számát jelöli,
    - $temp = sw \oplus rcw$,

    a **RotWord, SubWord, Rcon** értelmezését lásd lentebb.
  - $nw_1 = nw_0 \oplus w_1$, ahol w_1 RK_{i-1} megfelelő szava,
  - $nw_2 = nw_1 \oplus w_2$, ahol w_2 RK_{i-1} megfelelő szava,
  - $nw_3 = nw_2 \oplus w_3$, ahol w_3 RK_{i-1} megfelelő szava,
  - az nw_0, nw_1, nw_2, nw_3 szavakat egymás mellé írva kapunk egy 128 bitből álló sort, ezt pedig újból egy 4×4-es 2 dimenziós tömbbe rendezzük. A tömböt oszloponként töltjük fel, ahol a tömb egy-egy eleme egy bájtot fog tartalmazni. Ezáltal meghatároztuk a i-ik körkulcsot, RK_i -t.

A **RotWord** függvény paraméterként egy $w[a_0\ a_1\ a_2\ a_3]$ típusú szót kap, ahol a_0, a_1, a_2, a_3 egy-egy bájt. A bemeneti paraméteren egy balra történő ciklikus eltolást hajt végre, melynek eredménye: $nw[a_1\ a_2\ a_3\ a_0]$. Ezt az értéket adja vissza.

A **SubWord** függvény paraméterként egy $w[a_0\ a_1\ a_2\ a_3]$ típusú szót kap, ahol a_0, a_1, a_2, a_3 egy-egy bájt, alkalmazza bájtonként az S-boxot (lásd lentebb), visszaadva az újonnan kapott szót. Ha az átalakítandó bájt például a $4a$, akkor az S-box

táblázat 4-ik sora és a-ik oszlopának kereszteződésénél található bájt lesz az új érték: $d6$. A lentebb megadott S-box táblázat az összes lehetséges értékre megadja a bemeneti bájt megváltozott értékét.

Az **Rcon** függvény paraméterként egy i, $(i = 1, 2, \ldots, 10)$ indexet kap, mely a soron levő kör értékét jelzi. Az i alapján a $[\{02\}^{i-1}\ \{00\}\ \{00\}\ \{00\}]$ szó első elemének hatványozását végzi el, visszaadva az újonnan kapott szót. Az alábbi táblázat elemei megadják a szó első elemének hatványértékeit az i értékének függvényében, hexa formában:

i	1	2	3	4	5	6	7	8	9	10
$\{02\}^{i-1}$	01	02	04	08	10	20	40	80	1b	36

A hatványozás a $GF(2^8)$ véges testben történik, az erre vonatkozó matematikai ismeretek leírását lásd a [10]-ben.

4.17. példa. Határozzuk meg az AES 128-bites körkulcsok közül az első három körkulcs, azaz az RK_1, RK_2, RK_3 értékeit, ha a kezdeti RK_0, mint 16-os számrendszerbeli szám, a következő:

$$2b7e151628aed2a6abf7158809cf4f3c.$$

Ekkor RK_0 szavai: $w_0 = 2b7e1516$, $w_1 = 28aed2a6$, $w_2 = abf71588$, $w_3 = 09cf4f3c$, melyeket 2 dimenziós tömbbe rendezve kapjuk:

w_0	w_1	w_2	w_3
$2b$	28	ab	09
$7e$	ae	$f7$	cf
15	$d2$	15	$4f$
16	$a6$	88	$3c$

Az RK_1 szavainak meghatározása (1. lépés):

rw	sw	rcw	$temp$	
$cf4f3c09$	$8a84eb01$	01000000	$8b84eb01$	$nw_0 = a0fafe17$
				$nw_1 = 88542cb1$
				$nw_2 = 23a33939$
				$nw_3 = 2a6c7605$

és 2 dimenziós tömbbe rendezve:

w_0	w_1	w_2	w_3
$a0$	88	23	$2a$
fa	54	$a3$	$6c$
fe	$2c$	39	76
17	$b1$	39	05

Az RK_2 szavainak meghatározása (2. lépés):

rw	sw	rcw	$temp$	
$6c76052a$	$50386be5$	02000000	$52386be5$	$nw_0 = f2c295f2$
				$nw_1 = 7a96b943$
				$nw_2 = 5935807a$
				$nw_3 = 7359f67f$

és 2 dimenziós tömbbe rendezve

w_0	w_1	w_2	w_3
$f2$	$7a$	59	73
$c2$	96	35	59
95	$b9$	80	$f6$
$f2$	43	$7a$	$7f$

Az RK_3 szavainak meghatározása (3. lépés):

rw	sw	rcw	$temp$	
$59f67f73$	$cb42d28f$	04000000	$cf42d28f$	$nw_0 = 3d80477d$
				$nw_1 = 4716fe3e$
				$nw_2 = 1e237e44$
				$nw_3 = 6d7a883b$

és 2 dimenziós tömbbe rendezve

w_0	w_1	w_2	w_3
$3d$	47	$1e$	$6d$
80	16	23	$7a$
47	fe	$7e$	88
$7d$	$3e$	44	$3b$

Titkosítás

A titkosítás 4 lépésből áll, melynek során az alkalmazott **SubBytes, ShitRows, MixColumns** függvények értelmezését lásd lentebb:

1. A bemenetet egy 4×4-es 2 dimenziós tömbbe, S -be rendezzük, oszloponként töltve fel a tömböt, ahol a tömb minden egyes eleme egy bájtot fog tartalmazni.
2. Meghatározzuk az SS =**AddRoundKey**(S, RK_0) értékét, ahol az **AddRoundKey** az S és RK_0 paraméterekre bájtonként alkalmazza a $\oplus$ műveletet.
3. A következő lépéssorozatot 9 körben ismételjük.
   - $S1$ = **SubBytes**(SS), ahol SubBytes egy nem lineáris helyettesítést alkalmaz az SS bájtjaira.
   - $S2$ = **ShiftRows**($S1$), ahol ShiftRows az $S1$ utolsó 3 sorára egy ciklikus shiftet alkalmaz, különböző eltolási értékekkel.
   - $S3$ = **MixColumns**($S2$), ahol MixColumns $S2$ oszlopaira alkalmaz átalakítást.
   - SS = **AddRoundKey**($S3, RK_i$), ahol RK_i a kulcsgenerálás során kapott i-ik körkulcs.
4. Az utolsó, 10. körben a következőképpen járunk el (kimarad a MixColumns alkalmazása):
   - $S1$ = **SubBytes**(SS), ahol SubBytes egy nem lineáris helyettesítést alkalmaz az SS bájtjaira.
   - $S2$ = **ShiftRows**($S1$), ahol ShiftRows az $S1$ utolsó 3 sorára egy ciklikus shiftet alkalmaz, különböző eltolási értékekkel.
   - Meghatározzuk **AddRoundKey**($S2$, RK_{10}) értékét, ahol RK_{10} a kulcsgenerálás során kapott 10. kulcs.

A **SubBytes** függvény paraméterként egy 4×4-es tömböt kap, melynek elemeire (bájtjaira) alkalmazza az S-boxot (lásd lentebb). Visszaadja az újonnan kapott 4×4-es tömböt.

A **ShiftRows** függvény paraméterként egy 4×4-es tömböt kap, melynek utolsó 3 sorára, különböző eltolási értékekkel, egy-egy ciklikus eltolást alkalmaz, visszaadva az újonnan kapott 4×4-es tömböt:

s_{11}	s_{12}	s_{13}	s_{14}	$>>$	s_{11}	s_{12}	s_{13}	s_{14}
s_{21}	s_{22}	s_{23}	s_{24}		s_{22}	s_{23}	s_{24}	s_{21}
s_{31}	s_{32}	s_{33}	s_{34}		s_{33}	s_{34}	s_{31}	s_{32}
s_{41}	s_{42}	s_{43}	s_{44}		s_{44}	s_{41}	s_{42}	s_{43}

A **MixColumns** függvény paraméterként egy 4×4-es tömböt kap, oszlopain elvégzi az alábbi átalakítást, majd visszaadja az újonnan kapott 4×4-es tömböt:

s_{11}	s_{12}	s_{13}	s_{14}	$>>$	ns_{11}	ns_{12}	ns_{13}	ns_{14}
s_{21}	s_{22}	s_{23}	s_{24}		ns_{21}	ns_{22}	ns_{23}	ns_{24}
s_{31}	s_{32}	s_{33}	s_{34}		ns_{31}	ns_{32}	ns_{33}	ns_{34}
s_{41}	s_{42}	s_{43}	s_{44}		ns_{41}	ns_{42}	ns_{43}	ns_{44}

ahol az új 4×4-es tömb elemei, $i = 1, 2, 3, 4$ értékeire, a következők lesznek:

$$\begin{aligned} ns_{1i} &= (02 \circ s_{1i}) \oplus (03 \circ s_{2i}) \oplus s_{3i} \oplus s_{4i} \\ ns_{2i} &= s_{1i} \oplus (02 \circ s_{2i}) \oplus (03 \circ s_{3i}) \oplus s_{4i} \\ ns_{3i} &= s_{1i} \oplus s_{2i} \oplus (02 \circ s_{3i}) \oplus (03 \circ s_{4i}) \\ ns_{4i} &= (03 \circ s_{1i}) \oplus s_{2i} \oplus s_{3i} \oplus (02 \circ s_{4i}). \end{aligned}$$

4.2. megjegyzés. A 02 és 03 értékeket 16-os számrendszerbeli számoknak kell tekinteni. Az alkalmazott $\circ$ művelet pedig szorzást jelent a $GF(2^8)$ véges testben, ahol a használt irreducibilis polinom $m(x) = x^8 + x^4 + x^3 + x + 1$, részletes leírást találunk a [10]-ben.

4.3. megjegyzés. Az S-box (lenti táblázat) elemeit, melyek megadják a bemeneti bájt új értékét, jelöljük ezt nb-vel, a következőképpen határozzuk meg:

- az átalakítandó bájt bitjeit egy polinom együtthatóinak tekintjük, és meghatározzuk a polinom multiplikatív inverzét a $GF(2^8)$ véges testben, ahol a használt irreducibilis polinom $m(x) = x^8 + x^4 + x^3 + x + 1$. Jelöljük ezt b-vel, a biteket pedig $(b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0)$-vel.
- alkalmazzuk a következő affin transzformációt, mely során megkapjuk az új $nb = (nb_7\ nb_6\ nb_5\ nb_4\ nb_3\ nb_2\ nb_1\ nb_0)$ bitjeit:

$$nb_i = b_i \oplus b_{i+4 \pmod 8} \oplus b_{i+5 \pmod 8} \oplus b_{i+6 \pmod 8} \oplus b_{i+7 \pmod 8} \oplus c_i,$$

ahol $i = 0, \ldots, 7$ és $c = (c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0) = 0x63 = (01100011)$.

Az S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	$7c$	77	$7b$	$f2$	$6b$	$6f$	$c5$	30	01	67	$2b$	fe	$d7$	ab	76
1	ca	82	$c9$	$7d$	fa	59	47	$f0$	ad	$d4$	$a2$	af	$9c$	$a4$	72	$c0$
2	$b7$	fd	93	26	36	$3f$	$f7$	cc	34	$a5$	$e5$	$f1$	71	$d8$	31	15
3	04	$c7$	23	$c3$	18	96	05	$9a$	07	12	80	$e2$	eb	27	$b2$	75
4	09	83	$2c$	$1a$	$1b$	$6e$	$5a$	$a0$	52	$3b$	$d6$	$b3$	29	$e3$	$2f$	84
5	53	$d1$	00	ed	20	fc	$b1$	$5b$	$6a$	cb	be	39	$4a$	$4c$	58	cf
6	$d0$	ef	aa	fb	43	$4d$	33	85	45	$f9$	02	$7f$	50	$3c$	$9f$	$a8$
7	51	$a3$	40	$8f$	92	$9d$	38	$f5$	bc	$b6$	da	21	10	ff	$f3$	$d2$
8	cd	$0c$	13	ec	$5f$	97	44	17	$c4$	$a7$	$7e$	$3d$	64	$5d$	19	73
9	60	81	$4f$	dc	22	$2a$	90	88	46	ee	$b8$	14	de	$5e$	$0b$	db
a	$e0$	32	$3a$	$0a$	49	06	24	$5c$	$c2$	$d3$	ac	62	91	95	$e4$	79
b	$e7$	$c8$	37	$6d$	$8d$	$d5$	$4e$	$a9$	$6c$	56	$f4$	ea	65	$7a$	ae	08
c	ba	78	25	$2e$	$1c$	$a6$	$b4$	$c6$	$e8$	dd	74	$1f$	$4b$	bd	$8b$	$8a$
d	70	$3e$	$b5$	66	48	03	$f6$	$0e$	61	35	57	$b9$	86	$c1$	$1d$	$9e$
e	$e1$	$f8$	98	11	69	$d9$	$8e$	94	$9b$	$1e$	87	$e9$	ce	55	28	df
f	$8c$	$a1$	89	$0d$	bf	$e6$	42	68	41	99	$2d$	$0f$	$b0$	54	bb	16

4.18. példa. Határozzuk meg az S-box $\{4a\} = (01001010) = x^6 + x^3 + x$-ra alkalmazott értékét.

- $\{4a\}$ multiplikatív inverze: $b = x^7 + x^5 + x^3 + x + 1 = (10101011)$ lesz, mert $(x^6 + x^3 + x)(x^7 + x^5 + x^3 + x + 1) = 1 \pmod{x^8 + x^4 + x^3 + x + 1}$,
- a keresett érték

$$nb = (11010110) = \{d6\}$$

lesz, ahol az első három bit értékét a következőképpen határoztuk meg:

- $nb_0 = b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus c_0 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$,
- $nb_1 = b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$,
- $nb_2 = b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$,

  ...

4.2.5. IDEA (International Data Encryption Algorithm)

Az IDEA Xuejia Lai és James Massey által kidolgozott titkosítási eljárás, melyet Zürichben fejlesztettek ki 1991-ben. Egy ideig a DES ellenfeleként volt elkönyvelve. A PES (Proposed Encryption Standard) egy átdolgozott változata, kezdetben úgy is hívták, hogy IPES (Improved PES). Bruce Schneier (lásd [26]) szerint a szimmetrikus blokktitkosítók közül az egyik legbiztonságosabb algoritmus.

Az IDEA blokktitkosító kriptorendszer, ahol egy bemeneti blokk hossza 64 bit. A rejtjelezés során létrejövő kimeneti blokk mérete is 64 bit. A kulcs 128 bit hosszúságú. A titkosítás és visszafejtés folyamata megegyezik és a Feistel-séma szerint történik, 8-szor végezzük el ugyanazt a számítási folyamatot, 8 különböző körkulccsal. Az utolsó kör után a kimeneten még egy végső átalakítást végzünk.

A titkosítás három különböző matematikai művelet alkalmazásából áll, melyek hardver és szoftver megvalósításai, egyszerűek, részletesebb leírást találunk a [20] könyvben. Ezek a következők:

- $\oplus$, összeadás $\pmod 2$,

– $\boxplus$, összeadás $\pmod{2^{16}}$,
– $\odot$, szorzás $\pmod{2^{16}+1}$.

4.4. megjegyzés. Az $\odot$ művelet az IDEA S-boxának is tekinthető, ahol a következő kikötések érvényesek: az $a \odot b$ esetében, ha $a = 0$ vagy $b = 0$, akkor még a szorzás előtt 2^{16}-ra cseréljük a megfelelő nullásokat illetve, ha az eredmény 2^{16}, akkor 0-val helyettesítjük ezt.

Kulcsgenerálás

A 128 bites kulcsot felosztjuk 8 darab 16 bites alkulcsra, megkapjuk az első kör 6 alkulcsát: $K_1^{(1)}, \ldots, K_6^{(1)}$, és a második kör első két kulcsát: $K_1^{(2)}, K_2^{(2)}$. Ezután a kulcsot 25 bittel körkörösen balra forgatjuk, majd újból felosztjuk 8 egyenlő nagyságú alkulcsra. Az ekkor kapott első négy alkulcs, $K_3^{(2)}, K_4^{(2)}, K_5^{(2)}, K_6^{(2)}$, a második kör alkulcsai, a maradék négy: $K_1^{(3)}, K_2^{(3)}, K_3^{(3)}, K_4^{(3)}$, a harmadik kör alkulcsai lesznek. Ezután újabb 25 bittel való balra történő körkörös rotáció következik, és felosztás, majd addig ismételjük az eljárást, amíg 54 darab alkulcsot nem kapunk, jelöljük őket sorra:

$$(K_1^{(1)}, \ldots, K_6^{(1)}), (K_1^{(2)}, \ldots, K_6^{(2)}), \ldots, (K_1^{(8)}, \ldots, K_6^{(8)}), (K_1^{(9)}, \ldots, K_4^{(9)}).$$

Titkosítás

A rejtjelezés 8 körből áll, mely során felhasználjuk a kulcsgenerálásnál előállított alkulcsokat. A nyílt szöveg 64 bitből álló blokkját 4 darab 16 bites részre osztjuk, legyenek ezek X_1, X_2, X_3, X_4. A titkosítás eredményét az Y_1, Y_2, Y_3, Y_4 16 bites részek egymásután való fűzéséből kapjuk.

A következő három pontot nyolcszor ismételjük, a 8 különböző körkulcsra: $(K_1^{(r)}, \ldots, K_6^{(r)})$, $r = 1, \ldots, 8$:

1. $X_1 = X_1 \odot K_1^{(r)}$,
   $X_2 = X_2 \boxplus K_2^{(r)}$,
   $X_3 = X_3 \boxplus K_3^{(r)}$,
   $X_4 = X_4 \odot K_4^{(r)}$,

2. $t_0 = K_5^{(r)} \odot (X_1 \oplus X_3)$,
   $t_1 = K_6^{(r)} \odot (t_0 \boxplus (X_2 \odot X_4))$,
   $t_2 = t_0 \boxplus t_1$,

3. $X_1 = X_1 \oplus t_1$,
   $a = X_2 \oplus t_2$,
   $X_2 = X_3 \oplus t_1$,
   $X_3 = a$,,
   $X_4 = X_4 \oplus t_2$.

A következő és egyben utolsó számítássorozatot egyszer végezzük a $(K_1^{(9)}, \ldots, K_4^{(9)})$ alkulcsoknak megfelelő értékekkel, amely alkalmaz még egy átalakítást, mielőtt a végső kimenetet meghatározná.

1. $Y_1 = X_1 \odot K_1^{(9)}$,
   $Y_4 = X_4 \odot K_4^{(9)}$,

$$Y_2 = X_3 \boxplus K_2^{(9)},$$
$$Y_3 = X_2 \boxplus K_3^{(9)}.$$

Visszafejtés

A visszafejtés a titkosítás lépéssorozatát követi, azzal a különbséggel, hogy az alkalmazott kulcsok a következőek lesznek:

$$(K_1^{'(1)}, \ldots, K_6^{'(1)}), (K_1^{'(2)}, \ldots, K_6^{'(2)}), \ldots, (K_1^{'(8)}, \ldots, K_6^{'(8)}), (K_1^{'(9)}, \ldots, K_4^{'(9)}),$$

ahol a $K_i^{'(j)}$-ket a $K_i^{(j)}$-k alapján határozzuk meg, részletes leírást találunk a [20] könyvben:

r	$K_1^{'(r)}$	$K_2^{'(r)}$	$K_3^{'(r)}$	$K_4^{'(r)}$	$K_5^{'(r)}$	$K_6^{'(r)}$
1	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
2	$(K_1^{(10-r)})^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
8	$(K_1^{(10-r)})^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
9	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{10-r})^{-1}$	–	–

4.5. megjegyzés. A fenti táblázatban alkalmazott $-K_i^{(j)}$ jelölés a $K_i^{(j)}$ additív inverzét $\pmod{2^{16}}$ szerint, a $(K_i^{(j)})^{-1}$ jelölés pedig a $K_i^{(j)}$ multiplikatív inverzét jelöli $\pmod{2^{16}+1}$ szerint.

5. FEJEZET

HASH FÜGGVÉNYEK

A kriptográfiában a hash függvények olyan matematikai függvények, melyek egy tetszőleges hosszúságú bitsorból állítanak elő egy rögzített hosszúságú bitsort (ez általában 128 vagy 160 bit), amelyet hash értéknek neveznek. A függvény által előállított hash érték, melyet digitális lenyomatnak (fingerprint, message digest) is szokás nevezni, általában egy hosszabb szöveg, vagy dokumentumra kiszámolt speciális érték. A kriptográfiában számos protokoll során használunk hash függvényeket, elsősorban hitelesítés, digitális aláírások, titkosítások és üzenetsértetlenség ellenőrzésekor. Ez utóbbi azt jelenti, hogy két különböző időpontban, de ugyanarra a dokumentumra kiszámolt hash értékek összehasonlításával meg tudjuk állapítani, hogy ha a dokumentum tartalma megváltozott-e a két időpont között.

5.1. példa. A *Kriptográfia* szónak az MD5 hash függvény által előállított hash értéke:

af21a7cba9a7058ce268cbcdca1eed95.

Egy gyakorlatban is alkalmazható h hash függvény több feltételnek is meg kell feleljen:

1. Adott m üzenetre a $h(m)$ hash érték könnyen kiszámítható kell legyen.
2. Ismert $h(m)$ hash érték esetében nehéz meghatározni azt az m üzenetet, amelyből a $h(m)$ hash értéket számoltuk ki (preimage resistant). Az ennek a tulajdonságnak eleget tevő függvényeket egyirányú (one-way) függvényeknek hívják. Az egyirányú függvények létezésének bizonyítása máig is nyitott kérdés a kriptográfia területén. Egyirányú függvénynek elfogadott függvények közül megemlítjük azt a függvényt, melynek:
   - bemenete két egész szám és a kimenete a két szám szorzata; a függvény az egész számok faktorizációs problémájának nehézsége miatt van elfogadva mint egyirányú függvény,
   - bemenete négy egész szám: $x,\ a,\ b,\ p$ és a kimenete $a^x = b \pmod p$; a függvény a diszkrét logaritmus meghatározásának nehézsége miatt tartozik az egyirányú függvények családjába.
3. Adott m_1 üzenet esetében nehéz olyan $m_2 \neq m_1$ üzenetet találni, melyre $h(m_1) = h(m_2)$ (second preimage resistant). Ennek a feltételnek az alapján értelmezni tudjuk az ütközésmentes (collision-resistant) hash függvényeket: h ütközésmentes hash függvény, ha nehezen tudunk olyan $m1 \neq m2$ üzeneteket találni, melyekre $h(m1) = h(m2)$.

A gyakorlatban használt hash függvények közül közismertek:

- az MD5 (Message Digest Algorithm 5) 128 bites hash függvény, melyet Ronald Rivest tervezett 1991-ben; habár egy korábbi hash függvény, az MD4 javított változata mégsem bizonyul biztonságosnak, mert 2007-ben

egy Arjen Lenstra vezette csoport rámutatott arra, hogy két különböző állományra ütközést lehet találni,
– az SHA (Secure Hash Algorithm) hash függvénycsalád, mely név alatt 1993-tól kezdődően, a NSA (National Security Agency) által tervezett öt hash függvényt ismerjük, ezek közül megemlítjük az: SHA-0, SHA-1 függvényeket, ahol mindkettő kimenete 160 bit.

Egy klasszikus kriptográfiai hash függvény a Merkle–Damgard-konstrukció, mely egy f tömörítő függvény által előállított érték egymás utáni iterálásából áll, részletes leírását lásd [9]. A függvény tervezői, Ralph Merkle és Ivan Damgard, egymástól függetlenül publikálták eljárásukat. Ezt a szerkesztési módot követik a gyakorlatban is sokszor alkalmazott SHA-1, MD5 hash függvények. A szerkesztés lépései a következők:

1. Egy tetszőleges hosszúságú m üzenet bitsorának a végéhez egy 1-est, majd ezután annyi 0-ást pótolunk, hogy az m üzenet hossza 448 (mod 521) legyen. Ezután meghatározzuk az eredeti üzenet hosszának bitsorát, melyből az utolsó 64 bitet hozzákapcsoljuk az előbb kapott bitsorhoz. Ezáltal a kapott bitsor hossza 512 többszöröse lesz.
2. A kipótolt üzenetet felosztjuk 512 bites tömbökre, legyenek ezek $b_1,\ b_2, \ldots,\ b_n$.
3. Egy h_0 kezdeti érték alapján, melynek hossza 128 bit, elvégezzük a következő iterálást: $h_i = f(h_{i-1} + b_i)$, ahol f a tömörítő függvény, a $+$ pedig a két tömb, a h_{i-1} és b_i tömbök konkatenálását (egymás után való fűzését) jelenti.
4. Az m üzenet hash értéke h_n lesz.

A hash függvények egy másik típusa számelméleti problémákon alapuló kriptorendszerek mintáját követi, úgymint az RSA, ElGamal stb. Ezek a konstrukciók a gyakorlatban azonban nem igazán hatékonyak, lévén hogy a rendszerek hátterében időigényes matematikai műveletek elvégzése áll.

A fenti szerkesztések mellett számos más szerkesztés is létezik, de ezekre e jegyzet keretén belül nem térünk ki.

6. FEJEZET

NYILVÁNOS KULCSÚ KRIPTOGRÁFIA

A nyilvános kulcsú vagy aszimmetrikus kriptográfia alapjait az 1970-es évek közepén dolgozták ki. Az addig ismert és alkalmazott titkos kulcsú kriptorendszerek kulcscsere-problémáját kellett megoldani. Ahogy az elektronikus kommunikáció egyre szélesebb körben kezdett elterjedni, úgy ennek a feladatnak a megoldása is egyre fontosabb lett. Az áttörést Whitfield Diffie és Martin Hellman hozták, akik 1976-ban megjelentették híres cikküket, a *New Directions in Cryptography*t, melyben konkrét protokollt mutattak be a kulcscserére vonatkozóan.

A nyilvános kulcsú kriptográfiának több fontosabb területe van:

- A kulcscsere algoritmusok lehetővé teszik, hogy két kommunikáló fél egy nyilvános csatornát alkalmazva megegyezzen egy közös kulcsban, egy titkos információban, úgy hogy ezt egy harmadik fél ne tudja meghatározni. Ahhoz, hogy ez megoldható legyen a rendszerek publikus, illetve ezekhez kapcsolódó privát adatokat kell kezeljenek.
- A nyilvános kulcsú titkosító rendszerek legfontosabb tulajdonsága, hogy a nyílt szöveget a fogadó fél nyilvános kulcsával bárki titkosíthatja, a titkosított szöveget azonban senki más nem tudja visszafejteni, csak a fogadó fél, aki rendelkezik a titkos kulccsal, a nyilvános kulcs párjával.
- A digitális aláírások adatok hitelességének és sértetlenségének az ellenőrzésére alkalmas. Legfontosabb jellemzője, hogy a nyílt szöveget egy aláíró a titkos kulcsával aláírja, ezek után pedig, az aláíró nyilvános kulcsának ismeretében, bárki ellenőrizheti, hogy az üzenetet ki írta alá, illetve hogy tartalma sértetlen-e. Az aláíró a későbbiek során nem tudja letagadni, hogy aláírta a nyílt szöveget.

A legfontosabb kérdés, amellyel ezekben a rendszerekben találkozunk, hogy a publikus adatokról, a nyilvános kulcsokról hogyan lehet bebizonyítani, hogy azok valódiak, hogy nincsenek-e egy támadó fél által megváltoztatva. Ennek egyik megoldására dolgozták ki a PKI rendszereket.

A PKI (Public Key Infrastructure) rendszerek az Interneten keresztül történő igazolványt, tanúsítványt készítő rendszereknek felelnek meg, ahol a tanúsítványokat legtöbbször a Tanúsító Hatóság segítségével adják ki.

A Tanúsító Hatóság (Certificate Authority – CA) egy olyan jogi személy, melynek digitális tanúsítványok kibocsátására van jogosultsága. A tanúsítvány egy adatcsomag, mely többek között tartalmazza a tulajdonost igazoló személyes adatokat, a kommunikációhoz szükséges kulcsokat, a tanúsítvány lejárati idejét, a kibocsátó digitális aláírását stb. Minden tanúsítvány egyedi azonosítóval rendelkezik. A Tanúsító Hatóság ugyanakkor garantálja, hogy az általa kibocsátott adatcsomag hiteles és hamisíthatatlan. Számos kereskedelmi jellegű Tanúsító Hatóság díjfizetésszolgáltatás ellenében bocsátja ki a tanúsítványokat.

6.1. A Diffie–Hellman-kulcscsere

A nyilvános kulcsú kriptográfia, ahogy az előzőekből is kiderült, két neves kriptográfus, Whitfield Diffie és Martin Hellman nevéhez fűződik. Az 1976-ban megjelentetett cikkükben bemutatják, hogyan lehet a diszkrét logaritmus megoldásának nehézségén alapuló problémát felhasználni kriptográfiai alkalmazásokban. A protokoll, melyet ismertettek, alkalmas arra, hogy két legálisan, titkosan kommunikálni szándékozó fél meg tudjon egyezni egy közös kulcsban, egy nem biztonságos csatornán keresztül. [5].

Feltételezve, hogy a kommunikációban részt vevő két legális fél Alice és Bob, a Diffie–Hellman-kulcscsere egyszerűsített protokollja a következő lépéssorozatokból áll:

6.1. protokoll.

1. Alice választ egy nagy prímszámot, p-t, meghatározza p-nek egy primitív gyökét, legyen ez q, ahol $2 \leq q < p-1$, majd választ egy tetszőleges a pozitív egész számot, ahol $a < p-1$. Meghatározza $A = q^a \pmod{p}$ értéket, ahol az a értékét titokban tartja, a p, q, A értékeket pedig elküldi egy nyilvános csatornánk keresztül Bobnak.
2. Bob választ egy tetszőleges b pozitív egész számot, ahol $b < p-1$. Meghatározza $B = q^b \pmod{p}$ értéket, ahol b értékét titokban tartja. B-t elküldi Alicenak.
3. A közös kulcsot Alice a következőképpen határozza meg:
$$K = B^a \pmod{p}.$$
4. A közös kulcsot Bob a következőképpen határozza meg:
$$K = A^b \pmod{p}.$$

Hogy kulcsnak mindkét fél ugyanazt az értéket kapja, az a következőkből állapítható meg:

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

6.1. példa. Legyen $p = 348\,731$ prímszám és $q = 47\,641$ egy primitív gyöke.

- Ha az Alice által választott érték $a = 764$, akkor
$$A = 47\,641^{764} = 19\,481 \pmod{348\,731}.$$
- Ha a Bob által választott érték $b = 21\,789$, akkor
$$B = 47\,641^{21\,789} = 3\,397\,723 \pmod{348\,731}.$$
- Az Alice által kiszámolt kulcs: $339\,723^{764} = 31\,989 \pmod{348\,731}$.
- A Bob által kiszámolt kulcs: $19\,481^{21\,789} = 31\,989 \pmod{348\,731}$.

A rendszer biztonsága érdekében, a kulcserét megelőzően, a kommunikáló feleknek szükséges egymás személyét hitelesíteniük valamilyen eljárással, például igénybe vehetik egy PKI rendszer szolgáltatásait. Ha mindezt nem teszik meg, akkor egy támadó fél Alice és Bob közé kerülhet abban a szerepben, hogy Alice-nak kiadja magát mint Bob, illetve Bobnak mint Alice, részletes leírást találunk a [27]-ben.

6.2. Nyilvános kulcsú titkosító rendszerek

A nyilvános kulcsú titkosító rendszerek fontosabb jellemzői a következőek:
- a titkosításra alkalmazott kulcs különbözik a visszafejtésre alkalmazott kulcstól,
- a titkosításra alkalmazott kulcs nyilvános, ezt nevezik nyilvános vagy publikus kulcsnak, a visszafejtésre alkalmazott kulcs pedig titkos, ezt nevezik titkos vagy privát kulcsnak,
- a titkos kulcs meghatározása a nyilvános kulcs ismerete alapján, nehéz matematikai probléma megoldását jelenti, nem ismert rá hatékony algoritmus,
- a kommunikáló felek közötti titkosítási folyamatot nem előzi meg semmiféle, a kulcs értékére vonatkozó egyeztetés.

Feltételezve, hogy a kommunikációban részt vevő két legális fél Alice és Bob, akkor egy nyilvános kulcsú titkosító rendszer egyszerűsített protokollja a következő lépéssorozatból áll:

6.2. protokoll.
1. Alice és Bob megegyeznek egy aszimmetrikus titkosító rendszerben, azaz egy titkosítási eljárásban.
2. Alice elérhetővé teszi Bob számára a nyilvános kulcsát.
3. Bob Alice nyilvános kulcsával és a megállapodott titkosítási eljárással titkosítja a nyílt szöveget, létrehozva ezáltal a titkos szöveget.
4. Alice a titkos kulcsával visszafejti a titkos szöveget, megkapva a nyílt szöveget.

Hasonlóan a Diffie–Hellman rendszerhez a biztonság érdekében, a titkosítási folyamatot megelőzően a kommunikáló feleknek szükséges egymás személyét hitelesíteniük.

6.2.1. Az RSA titkosító rendszer

Az RSA az első nyilvános kulcsú kriptorendszer, amely napjainkban is megfelelő biztonságot nyújt, ha a rendszer paramétereit kellőképpen választjuk meg. Ronald Rivest, Adi Shamir és Leonard Adleman publikálta 1977-ben, elnevezése három megalkotójának kezdőbetűjéből származik. Azóta számos protokoll használja, mivel egyaránt alkalmas titkosításra és digitális aláírásra. A bemutatásra kerülő algoritmusnak csupán didaktikai szándéka van, a gyakorlatban az RSA-OAEP rendszert használják, amely az RSA biztonságos változata. Erre azonban a jegyzet keretén belül nem térünk ki.

Biztonsága azon az elven alapul, hogy az ismert prímszám generáló, illetve hatványozási algoritmusok futási ideje lényegesen kisebb, mint egy szám prímfaktorizációját meghatározó algoritmusok futási ideje, leírását számos helyen megtaláljuk, lásd például [5].

A kulcsot egy kulcspár alkotja: a nyilvános kulcs és a titkos kulcs. A nyilvános kulcs segítségével történik a nyílt szöveg titkosítása, a titkos kulccsal pedig a titkosított szöveget tudjuk visszafejteni, visszakapva a nyílt szöveget. Tehát bárki rejtjelezhet, viszont a rejtjelezett szöveg megfejtését csak az valosíthatja meg, aki ismeri a titkos kulcsot, a megfelelő nyilvános kulcs párját.

A rendszer leírása során három algoritmust fogunk megadni: a kulcsgenerálás, a titkosítás és a visszafejtés algoritmusát.

Kulcsgenerálás

- választunk két $q_1,\ q_2$ tetszőleges prímszámot,
- meghatározzuk az $m = q_1 \cdot q_2$ szorzatot, melyet az RSA modulusának is neveznek,
- meghatározzuk a $\phi(m) = (q_1 - 1) \cdot (q_2 - 1)$ értéket, ahol ϕ az Euler phi-függvény,
- választunk egy e számot a következő tulajdonsággal:

$$1 \leq e \leq \phi(m),\ gcd(e,\ \phi(m)) = 1,$$

- meghatározzuk a d számot a következő tulajdonsággal:

$$1 \leq d \leq \phi(m),\ e \cdot d = 1 \pmod{\phi(m)},$$

  azaz d az e multiplikatív inverze lesz $\pmod{\phi(m)}$ szerint,
- a megfelelő kulcsok ezek után a következők lesznek :
  - a nyilvános kulcs: $(e,\ m)$,
  - a titkos kulcs: (d).

Ahhoz, hogy a rendszer megfelelő biztonságú legyen, az m modulus legalább 512 bitből kell álljon.

6.2. példa. Adott két prímszám q_1, q_2 esetében határozzuk meg az RSA rendszernél alkalmazott nyilvános és titkos kulcsokat.

- Legyen $q_1 = 3877,\ q_2 = 1866$.
- Meghatározzuk:
  - $m = 3877 \cdot 1867 = 7\,238\,359$,
  - $\phi(m) = 3876 \cdot 1866 = 7\,232\,616$.
- Legyen $e = 65\,537$, ahol $gcd(65\,537,\ \phi(m)) = 1$.
- Meghatározzuk e inverzét $\pmod{\phi(m)}$ szerint, kapjuk: $d = 1\,332\,809$, mert $65\,537 \cdot 1\,332\,809 = 1 \pmod{7\,232\,616}$.
- A nyilvános kulcs : $(65\,537,\ 7\,238\,359)$.
- A titkos kulcs : $(1\,332\,809)$.

A titkosítás és visszafejtés lépéseit abban az esetben mutatjuk be, amikor az alkalmazott ábécé 256 elemből áll, azaz az összes lehetséges bájt érték előfordulhat.

Titkosítás

Feltételezzük, hogy a nyílt szöveg egy k bájtos blokkból áll, amelyből meghatározzunk egy 256^k számrendszerbeli számot. Ha a k bájtos blokk bájtjait $b_{k-1}\ b_{k-2}\ \ldots b_1\ b_0$-val jelöljük, akkor az előállított szám:

$$p = b_{k-1} \cdot 256^{k-1} + b_{k-2} \cdot 256^{k-2} + \cdots + b_1 \cdot 256 + b_0.$$

A p egész szám eleget kell tegyen az $1 \leq p \leq m,\ gcd(p,\ m) = 1$ feltételeknek. Ennek érdekében a k érték nem lehet nagyobb mint:

$$k = \lceil log_{256} m \rceil.$$

A p egész szám titkosítása végett az $(e,\ m)$ nyilvános kulcs ismeretében a következőképpen járunk el:

$$c = p^e \pmod{m}.$$

A titkosított c értéket egy 256^{k+1} számrendszerbeli számnak tekintjük, melyet visszaalakítunk 256-os számrendszerbe. Így igazából bájtokat kapunk, melyekből elő tudjuk állítani a titkosított szöveget.

Visszafejtés

A titkosított szöveget bájtjaiból ismét egy 256^{k+1} számrendszerbeli számot hozunk létre. Ha az így képzett egész számot c-vel jelöljük, akkor a (d) titkos kulcs ismeretében a p értékét a következőképpen kapjuk vissza:

$$p = c^d \pmod{m}.$$

A visszafejtett p értéket egy 256^k számrendszerbeli számnak tekintjük, melyet átalakítunk 256-os számrendszerbe. Az így kapott bájtokból kapjuk vissza a nyílt szöveg bájtjait.

6.1. megjegyzés. Hogy a visszafejtés tényleg a titkosított értéket adja vissza, az leolvasható a következőkből:

$$e \cdot d \equiv 1 \pmod{\phi(m)}, \text{ ahol } \phi(m) = (q_1 - 1) \cdot (q_2 - 1) \Rightarrow$$

létezik egy k_1 egész szám, úgyhogy:

$$e \cdot d = 1 + k_1 \cdot (q_1 - 1) \cdot (q_2 - 1).$$

Ha $q_1 | p$ akkor $p^{e \cdot d} = 0 \pmod{q_1}$ és $p = 0 \pmod{q_1}$.
Ha $q_1 \not| p$, akkor a kis Fermat-tétel alapján $p^{(q_1-1)} \equiv 1 \pmod{q_1} \Rightarrow$

$$\begin{aligned} c^d &= p^{e \cdot d} = p^{1 + k_1 \cdot (q_1-1) \cdot (q_2-1)} = p \cdot (p^{(q_1-1) \cdot (q_2-1)})^{k_1} \\ &= p \cdot (p^{(q_1-1)})^{(q_2-1) \cdot k_1} = p \pmod{q_1}. \end{aligned}$$

Tehát

$$p^{e \cdot d} = p \pmod{q_1}.$$

Hasonlóan

$$p^{e \cdot d} = p \pmod{q_2}.$$

A kínai maradéktétel alapján:

$$c^d = p^{e \cdot d} = p \pmod{m}.$$

6.3. példa. Titkosítsuk a $MATHEMATICIAN$ karakterláncot, majd fejtsük vissza a rejtjelezett szöveget.

A karakterlánc a következő bájtszekvenciával egyenlő:

$$77, 65, 84, 72, 69, 77, 65, 84, 73, 67, 73, 65, 78.$$

A megfelelő p egész számot úgy kapjuk, hogy a fenti egész számokat 256-os számrendszerbeli számoknak tekintjük és átalakítjuk őket 256^k-as számrendszerbe, ahol $k = 13$, mert 13 bájtos a szekvencia:

$$p = 77 \cdot 256^{12} + 65 \cdot 256^{11} + \cdots + 67 \cdot 256^3 + 73 \cdot 256^2 + 65 \cdot 256 + 78 = 612078693029436798400912957473 4$$

Ahhoz hogy titkosítani tudjuk a karakterláncot legalább 13 bájtos, azaz $13 \cdot 8 = 104$ bites publikus kulcsra van szükségünk.

- Legyen $q_1 = 2576742233143601,\ q_2 = 4170624197489501$.
- Meghatározzuk:
  - $m = q_1 \cdot q_2 = 10746623508241835606179422833101$,
  - $\phi(m) = (q_1 - 1) \cdot (q_2 - 1) = 10746623508241828858812992200000$.
- Legyen $e = 3$, ahol $gcd(3\ \phi(m)) = 1$.
- Meghatározzuk e inverzét $\pmod{\phi(m)}$ szerint, kapjuk:

$$d = 7164415672161219239208661466667.$$

- A nyilvános kulcs : $(3,\ 10746623508241835606179422833101)$.
- A titkos kulcs : $(7164415672161219239208661466667)$.

A titkosított érték:

$$c = p^e \pmod{m} = 16870817698871835013291066335 13.$$

A titkosított értéknek megfelelő bájtszekvenciát megkapjuk, ha a c értékének meghatározzuk rendre a 256-al való osztási maradékait, azaz a kapott értéket visszaalakítjuk 256-os számrendszerbe. Ezek a bájtértékek a következők lesznek:

$$21, 75, 65, 84, 243, 158, 99, 221, 228, 20, 82, 131, 41.$$

A titkosított érték bájtjait általában hexadecimális számrendszerben is megszokás adni, ezek a következők lesznek:

$$154b4154f39e63dde414528329.$$

A visszafejtéshez a rejtjelezett bájtokat újból át kell alakítani 256-os számrendszerből 256^{13}-as számrendszerbe, majd a kapott c érték alapján visszakapjuk a p-t:

$$p = c^d \pmod{m} = 612078693029436798400912957473 4.$$

Ezt az értéket ha átalakítjuk 256-os számrendszerbe kapjuk a megfelelő bájtszekvenciát, ahonnan meghatározható az eredeti nyílt szöveg.

Az RSA biztonsága a prímfaktorizációs matematikai probléma nehézségén alapszik. Máig sem bizonyított, hogy a titkosított szöveg visszafejtésére nem létezik megfelelő gyorsaságú (polinom idejű) algoritmus. Az viszont bizonyított, hogy az RSA modulus értékének a prímtényezőkre való bontása egyenértékű feladat a titkos kulcs meghatározásával. Ahhoz, hogy a faktorizációt megnehezítsük, illetve hogy a rendszer biztonságosságát és hatékonyságát is növeljük, szükséges néhány követelményt szem előtt tartani:

- Az RSA kulcs meghatározásakor választott q_1 és q_2 prímszámokat nem szabad egymáshoz túl közelinek venni. Lehet őket erős prímeknek venni, bár az RSA Security ezt nem ajánlja, elsősorban a kulcs előállítási költsége miatt, másodsorban léteznek olyan prímfaktorizációs algoritmusok, melyek hasonló hatékonysággal működnek az erős prímek esetében is.
- A választott e értéke a lehető legkisebb kell legyen, ahhoz, hogy a titkosítás folyamata a lehető legrövidebb ideig tartson. Ha viszont túlságosan kis értéket választunk, például $e = 3$-at, akkor a támadó fél egy jól választott stratégia alapján meghatározhatja a nyílt szöveget, lásd [5].

– Ajánlott, hogy a választott d értéke az m modulus nagyságrendjével egyezzen meg, és a d kettes számrendszerbeli felbontásában az egyesek száma legalább $k/2$ legyen, ahol k az m bitjeinek a számát jelöli. Ilyen feltételek mellett az RSA visszafejtési algoritmus időigénye a DES, AES szimmetrikus kulcsú rendszerekhez képest jóval nagyobb.
– A visszafejtés folyamatát gyorsíthatjuk, ha alkalmazzuk a kínai maradéktételt, lásd [5].

6.2.2. A hátizsák feladaton alapuló kriptorendszer

A hátizsák (knapsack) feladaton alapuló kriptorendszer az egyike a legrégebbi nyilvános kulcsú kriptorendszereknek, melyet Ralph Merkle és Martin Hellman mutatott be 1978-ban. A rendszer az RSA-hoz képest jóval egyszerűbb, de 1982-ben Adi Shamir megmutatta feltörési módszerét (lásd [25]).

A rendszer hátterében a következő NP-teljes (lásd [23]), azaz „nehéz" feladat áll: Adott egy számsor, legyen ez $a_1, a_2, \ldots, a_n$ és egy m egész szám. Tudva, hogy az m egyenlő a számsorozat egy részhalmaza elemeinek az összegével, a feladat az, hogy meghatározzuk ezt a részhalmazt. Ha a számsorozat elemei szupernövekvőek (azaz egy tetszőleges a_i szám nagyobb, mint az összes előtte levő szám összege), akkor a feladat megoldható polinomidejű (lásd [16]) algoritmussal.

A hátizsák-kriptorendszert ezek után a kulcsgenerálás, titkosítás és visszafejtés algoritmusai alapján írjuk le.

Kulcsgenerálás

Legyenek $m,\ t,\ A = [a_1, a_2, \ldots a_n]$ a következő tulajdonsággal:
– $m,\ t$ pozitív egész számok, ahol $gcd(m,\ t) = 1$,
– A egy szupernövekvő számsorozat,
– az m legalább kétszer olyan nagy, mint az A számsorozat elemeinek összege.

A rendszerben alkalmazott kulcsok a következők:
– a titkos kulcs m, t egész számok,
– a nyilvános kulcs a $B = [t \cdot a_1, t \cdot a_2, \ldots, t \cdot a_n]$ számsor.

6.4. példa. Határozzuk meg a hátizsák-kriptorendszerben alkalmazott kulcsokat, adott $n = 10$ és $A = [103, 110, 215, 430, 861, 1723, 3451, 6901, 13\,801, 27\,560]$ értékek esetében.

Legyen a titkos kulcs: $m = 55\,201$, és $t = 13\,243$.

A B nyilvános kulcs elemeit a következőképpen határozzuk meg:

$$\begin{aligned}
103 \cdot 13\,243 &= 39\,205 \pmod{55\,201}\\
110 \cdot 13\,243 &= 21\,504 \pmod{55\,201}\\
215 \cdot 13\,243 &= 31\,994 \pmod{55\,201}\\
430 \cdot 13\,243 &= 8787 \pmod{55\,201}\\
861 \cdot 13\,243 &= 30\,817 \pmod{55\,201}\\
1723 \cdot 13\,243 &= 19\,676 \pmod{55\,201}\\
3451 \cdot 13\,243 &= 50\,366 \pmod{55\,201}\\
6901 \cdot 13\,243 &= 32\,288 \pmod{55\,201}\\
13\,801 \cdot 13\,243 &= 51\,333 \pmod{55\,201}\\
27\,560 \cdot 13\,243 &= 43\,269 \pmod{55\,201}
\end{aligned}$$

$B = [39\,205, 21\,504, 31\,994, 8787, 30\,817, 19\,676, 50\,366, 32\,288, 51\,333, 43\,269]$.

Titkosítás

A nyílt szöveg minden egyes karakterének meghatározzuk a számkódját. Ha az angol ábécé nagybetűi felett végezzük a titkosítást, akkor a karakterekhez hozzárendelt számkód lehet az (ASCII kód − 65) különbség. Ezután meghatározzuk a kapott kódok 2-es számrendszerbeli alakjait, majd n-bites részekre osztjuk az így átalakított szöveget.

Jelöljünk egy ilyen n-bites részt x-szel és a biteket: $(x_1\ x_2\ \ldots\ x_n)$ -el. A B nyilvános kulcs elemeit $b_1,\ b_2,\ \ldots,\ b_n$-el jelölve, az $x = (x_1x_2 \ldots\ x_n)$-nek megfelelő rejtjel azon b_i-kből képzett $\pmod m$ szerinti összeg lesz, amelyekre $x_i \neq 0$. Az ily módon kapott 10-es számrendszerbeli számok fogják képezni a titkosított szöveget, melyeket egymástól elkülönítve kell eltárolni.

Visszafejtés

A visszafejtéshez meghatározzuk t^{-1}-t, ahol $t \cdot t^{-1} = 1 \pmod m$, azaz t^{-1} a $t \pmod m$ szerinti multiplikatív inverze lesz.

Feltételezve, hogy x rejtjele y, előbb meghatározzuk a

$$c = y \cdot t^{-1} \pmod m$$

értéket, majd a következő eljárással visszakapjuk x bitjeit.

6.1. algoritmus.
Bemenet: c, n, A egész számok.
Kimenet: $x_1, x_2, \ldots, x_n$ bitek.

```
knapsack := proc(c, n, A)
    i := n;
    while i>0 do
    if c >= A[i] then
        c := c - A[i];
        x[i] := 1;
    else x[i] := 0;
    end if;
    i := i-1;
end do; return (x[1],x[2],...,x[n]); end proc;
```

6.5. példa. Határozzuk meg a

$$THISISAPLAINTEXT$$

nyílt szöveg esetében a rejtjelezett értéket, ha a nyilvános kulcs az 6.4. példánál megadott érték.

Meghatározzuk a nyílt szöveghez tartozó bitsort a karakterekhez tartozó számkód, azaz az (ASCII kód − 65) különbség alapján, majd $n = 10$-es csoportokat formálunk, ahol tehát

– a számkódok: $[19, 7, 8, 18, 8, 18, 0, 15, 11, 0, 8, 13, 19, 4, 23, 19]$,

– a karakterek 10-bitenként csoportosítva, a számkódok és a bitek:

$$\begin{array}{cll} TH & (19,\ 7) & (1,0,0,1,1,0,0,1,1,1) \\ IS & (8,\ 18) & (0,1,0,0,0,1,0,0,1,0) \\ IS & (8,\ 18) & (0,1,0,0,0,1,0,0,1,0) \\ AP & (0,\ 15) & (0,0,0,0,0,0,1,1,1,1) \\ LA & (11,\ 0) & (0,1,0,1,1,0,0,0,0,0) \\ IN & (8,\ 13) & (0,1,1,0,1,0,1,1,0,1) \\ TE & (19,\ 4) & (1,0,0,1,1,0,0,1,0,0) \\ XT & (23,\ 19) & (1,0,1,1,1,1,0,0,1,1) \end{array}$$

A csoportok 1-es bitjei által mutatott megfelelő összegeket képezve kapjuk az x-eknek megfelelő rejtjeleket:

$$\begin{array}{cll} TH & 39\,205 + 8787 + 30\,817 + 32\,288 + 51\,333 + 43\,269 & = 205\,699 \\ IS & 21\,504 + 19\,676 + 51\,333 & = 92\,513 \\ IS & 21\,504 + 19\,676 + 51\,333 & = 92\,513 \\ AP & 50\,366 + 32\,288 + 51\,333 + 43\,269 & = 17\,256 \\ LA & 21\,504 + 8787 + 30\,817 & = 61\,108 \\ IN & 21\,504 + 31\,994 + 30\,817 + 50\,366 + 32\,288 + 43\,269 & = 210\,238 \\ TE & 39\,205 + 8787 + 30\,817 + 32\,288 & = 111\,097 \\ XT & 39\,205 + 31\,994 + 8787 + 30\,817 + 19\,676 + 51\,333 + 43\,269 & = 225\,081 \end{array}$$

A titkosított szöveg ezek szerint 8 darab 10-es számrendszerbeli számból fog állni.

6.6. példa. Határozzuk meg az 6.5. példánál kapott titkosított rejtjelekhez tartozó nyílt szöveget, ha a kulcsok az 6.4. példánál megadott értékek.

– Meghatározzuk előbb a $t = 13\,243$ multiplikatív inverzét $\pmod{55\,207}$ szerint, majd mindegyik rejtjel esetében a $t^{-1} \cdot y$ értéket.
– A multiplikatív inverz:

$$t^{-1} = 27\,811 \pmod{55\,201}, \text{ mert } 27\,811 \cdot 13\,243 = 1 \pmod{55\,201}.$$

– A megfelelő y rejtjelek és a hozzá tartozó bitértékek:

$$\begin{array}{rcll} 205\,699 \cdot 27\,811 & = & 49\,656 \pmod{55\,201} & [1,0,0,1,1,0,0,1,1,1] \\ 92\,513 \cdot 27\,811 & = & 15\,634 \pmod{55\,201} & [0,1,0,0,0,1,0,0,1,0] \\ 92\,513 \cdot 27\,811 & = & 15\,634 \pmod{55\,201} & [0,1,0,0,0,1,0,0,1,0] \\ 177\,256 \cdot 27\,811 & = & 51\,713 \pmod{55\,201} & [0,0,0,0,0,0,1,1,1,1] \\ 61\,108 \cdot 27\,811 & = & 1\,401 \pmod{55\,201} & [0,1,0,1,1,0,0,0,0,0] \\ 210\,238 \cdot 27\,811 & = & 39\,098 \pmod{55\,201} & [0,1,1,0,1,0,1,1,0,1] \\ 111\,097 \cdot 27\,811 & = & 8\,295 \pmod{55\,201} & [1,0,0,1,1,0,0,1,0,0] \\ 225\,081 \cdot 27\,811 & = & 44\,693 \pmod{55\,201} & [1,0,1,1,1,1,0,0,1,1] \end{array}$$

– A fenti táblázat első sorának a bitjeinek lépésenkénti meghatározását, azaz 49 656 bitjeit, az 6.1. algoritmus a következőképpen végzi.

i	c	$x[i]$	Magyarázat
10	$49\,656 - 27\,560 = 22\,096$	1	$49\,656 \geq 27\,560$
9	$22\,096 - 13\,801 = 8\,295$	1	$22\,096 \geq 13\,801$
8	$8\,295 - 6\,901 = 1\,394$	1	$8\,295 \geq 6\,901$
7		0	$1\,394 < 3\,451$
6		0	$1\,394 < 1\,723$
5	$1\,394 - 861 = 533$	1	$1394 \geq 861$
4	$533 - 430 = 103$	1	$533 \geq 430$
3		0	$103 < 215$
2		0	$103 < 110$
1	$103 - 103 = 0$	1	$103 \geq 103$

6.2.3. Az ElGamal titkosító rendszer

Az ElGamal nyilvános kulcsú kriptorendszert 1984-ben publikálta Taher Elgamal. Biztonsága a diszkrét logaritmus meghatározásának a nehézségén alapszik, és bármilyen ciklikus csoport esetében értelmezhető, ha a csoport elemeivel hatékonyan végezhetők a megfelelő számítási műveletek. Nem szabad összetéveszteni a szintén ElGamal által javasolt digitális aláírás algoritmussal. A rendszer egyik előnye, hogy a kulcsgenerálás során egyetlen prímszámot kell meghatározni, hátránya viszont az, hogy a titkosított szöveg kétszerese lesz a nyílt szövegnek.

Hasonlóan az RSA-hoz, a rendszer leírásakor három fázist különböztetünk meg: kulcsgenerálást, titkosítást és visszafejtést.

Kulcsgenerálás

- választunk egy p tetszőleges prímszámot,
- meghatározzuk p egy primitív gyökét, legyen ez q,
- választunk egy a számot, ahol $1 \leq a \leq p-1$, majd meghatározzuk:

$$A = q^a \pmod{p},$$

- a megfelelő kulcsok ezek után a következőek lesznek:
  - a nyilvános kulcs: $(p,\ q,\ A)$,
  - a titkos kulcs: (a).

6.7. példa. Határozzuk meg $p = 6\,545\,641$ prímszám esetében az ElGamal-kriptorendszer nyilvános és titkos kulcsait.

- Legyen p egy primitív gyöke $q = 1\,735\,930$ és legyen $a = 5\,276\,152$.
- Meghatározzuk: $A = q^a \pmod{p} = 2\,566\,890$-t.
- A nyilvános kulcs: $(6\,545\,641,\ 1\,735\,930,\ 2\,566\,890)$.
- A titkos kulcs: $(5\,276\,152)$.

Titkosítás

Az $1 \leq m \leq p-1$ egész szám titkosítása végett a következőképpen járunk el:

- választunk egy b számot, ahol $1 \leq b \leq p-1$, és a nyilvános kulcs ismeretében meghatározzuk:

$$B = q^b \pmod{p},$$

- meghatározzuk: $c = A^b \cdot m \pmod{p}$,
- a tikosított érték: $(B,\ c)$ lesz.

6.8. példa. Határozzuk meg $m = 3\,537$ titkosított értékét, ha a rendszer nyilvános kulcsa az 6.7. példánál kiszámolt érték.

- Legyen $b = 168\,765$.
- Meghatározzuk $B = 1\,735\,930^{168\,765} \pmod{6\,545\,641} = 1\,363\,661$.
- Meghatározzuk $c = 2\,566\,890^{168\,765} \cdot 3537 \pmod{6\,545\,641} = 150\,466$.
- A titkosított érték: $(1\,363\,661,\ 150\,466)$.

Visszafejtés

A $(B,\ c)$ titkosított érték visszafejtését, az a titkos kulcs ismeretében, a következőképpen végezzük:

- meghatározzuk: $x = p - 1 - a$-t, ahol az x-re teljesülni fog: $1 \leq x \leq p - 2$,
- meghatározzuk: $m = B^x \cdot c \pmod{p}$.

6.2. megjegyzés. Hogy a visszafejtés tényleg a titkosított értéket adja vissza, az leolvasható a következőkből:

$$\begin{aligned}(B^x \cdot c) &= (q^b)^{(p-1-a)} \cdot A^b \cdot m \\ &= q^{(p-1)\cdot b} \cdot q^{b\cdot(-a)} \cdot q^{a\cdot b} \cdot m \\ &= m \pmod{p}\end{aligned}$$

$$q^{p-1} = 1 \pmod{p} \qquad \text{(kis Fermat-tétel alapján, lásd 2.1. tétel).}$$

6.9. példa. Az 6.7. példánál kiszámolt titkos kulcs ismeretében fejtsük vissza az előző példánál meghatározott titkosított értéket.

- Meghatározzuk $x = 6\,545\,640 - 5\,276\,152 = 1\,269\,488$.
- Kiszámítjuk $1\,363\,661^{1\,269\,488} \cdot 150\,466 = 3\,537 \pmod{6\,545\,641}$.

Az ElGamal-rendszer a titkosítás során két hatványozást kell elvégezzen, szemben az RSA-val, ahol csak egy van, ebből kifolyólag a titkosítás folyamata természetesen lassúbb lesz. Mivel az egyik hatványozás, a $q^b \pmod{p}$ kiszámítása nem függ a nyílt szövegtől, ezért ez végrehajtható egy előszámítási fázisban, csak arra kell vigyázni, hogy a kapott értéket titokban és megfelelő módon tároljuk a későbbiek során.

A visszafejtéshez egyetlen hatványozás szükséges, és a számításokat nem lehet gyorsítani a kínai maradéktétel alkalmazásával.

Az ElGamal-rendszer probabilisztikus, mivel a b-t véletlenszerűen választjuk meg, ami véletlenszerűen meghatározott titkosított szöveget eredményez. Egyetlen nyílt szöveghez tehát számos titkosított szöveg tartozhat. Ez a lehetőség a rendszer feltörését nagyban megnehezíti. Ha a nyilvános kulcs $(p,\ q,\ A)$, akkor b_1 és b_2 választással $(B_1,\ c_1)$ és $(B_2,\ c_2)$ titkosított szövegeket kapjuk, ahol:

$$\begin{aligned}B_1 &= q^{b_1} \pmod{p}, \\ c_1 &= (A^{b_1} \cdot m) \pmod{p},\end{aligned}$$

$$\begin{aligned}B_2 &= q^{b_2} \pmod{p}, \\ c_2 &= (A^{b_2} \cdot m) \pmod{p}.\end{aligned}$$

6.10. példa. Az 6.7. példánál vett $(6\,545\,641,\ 1\,735\,930,\ 2566890)$ nyilvános és $(5\,276\,152)$ titkos kulcs esetében a titkosítás során az $m = 540\,010$ értékre, $b_1 = 452$

és $b_2 = 56\,564$ választás esetében két különböző titkosított értéket kapunk:

$$\begin{aligned} B_1 &= 1\,735\,930^{452} = 1\,397\,827 \pmod{6\,545\,641}, \\ c_1 &= (2\,566\,890^{452} \cdot 540\,010) = 561\,968 \pmod{6\,545\,641}, \\ \\ B_2 &= 1\,735\,930^{56\,564} = 5\,455\,142 \pmod{6\,545\,641}, \\ c_2 &= (2\,566\,890^{56\,564} \cdot 540\,010) = 5\,407\,096 \pmod{6\,545\,641}. \end{aligned}$$

Ahhoz, hogy az esetleges támadások ellen védve legyen a rendszer, a p modulus mérete legalább 768 bitből kell álljon, továbbá bizonyos feltételeknek is eleget kell tegyen, például ne legyenek $(p-1)$-nek kis prímosztói. Ha véletlenszerűen választunk egy prímszámot, megfelelő nagyságrenddel, a prímek közül, akkor ez a feltétel általában teljesülni fog.

6.3. Digitális aláírások

Digitális aláírás alatt azt értjük, hogy egy adott dokumentum, ami lehet akár e-mail, akár egy tetszőleges okirat, lenyomatát (hash értékét) a feladó (aláíró) egy titkos kulcs segítségével titkosítja, és a titkosított lenyomatot eljuttatja a címzetthez, adott esetben az eredeti dokumentummal együtt. A címzett, a feladó nyilvános kulcsának ismeretében, le tudja ellenőrizni mind az eredeti dokumentum sértetlenségét, mind az aláíró hitelességét (a titkos kulcsot csak az aláíró ismeri). Mivel a dokumentum sértetlenségének a leellenőrzésére és az aláíró azonosítására egyaránt alkalmas, a digitális aláírást használni lehet azokon a területeken, ahol ez a kettős konfirmálás szükséges, leírását lásd az [5]-ben. Alkalmatlan viszont nyílt szöveg tartalmának a titkosítására.

A legelterjedtebb digitális aláírás-eljárások egyike a DSA (Digital Signature Algorithm), mely az ElGamal nyilvános kulcsú kriptorendszeren alapszik. Egy másik említésre méltó digitális aláírási rendszer az RSA aláírási séma, melynek elméleti hátterében az RSA kriptorendszernél bemutatott matematika áll. Elmondható az is, hogy a DSA gyorsabb, mint az RSA aláírási séma. A DSA a FIPS által 1993-ban elfogadott digitális aláírási szabvány.

Feltételezve, hogy Alice digitálisan alá szeretne írni egy dokumentumot és Bob le szeretné ezt ellenőrizni, akkor a rendszer a következő protokoll szerint működik:

6.3. protokoll.

1. Alice meghatározza az eredeti dokumentum lenyomatát, valamely hash függvény segítségével.
2. Alice digitálisan, a birtokában levő titkos kulccsal aláírja a lenyomatot.
3. Alice a nyilvános kulcsot, azaz a titkos kulcs párját, az eredeti dokumentumot és adott esetben az aláírt lenyomatot eljuttatja Bobhoz.
4. Bob a nyilvános kulcs segítségével visszafejti az aláírt lenyomatot.
5. Bob meghatározza az eredeti dokumentum hash értékét és összehasonlítja az általa visszafejtett lenyomat értékével. Ha ez a két érték megegyezik, akkor elfogadja a digitális aláírást.

Ahhoz, hogy a digitális aláírás megfelelő biztonságú legyen, a következőket kell szem előtt tartani:

– az aláírás hiteles kell legyen: senki, akinek nincs joga a titkos kulcshoz, nem írhat alá digitálisan dokumentumot, azaz nem hozható létre hamis aláírás,
– az aláírás sértetlen kell legyen: az alkalmazott hash függvény megfelelő tulajdonságú kell legyen, lásd a 5. fejezetet,
– az aláíró nem tagadhatja meg az aláírását, azaz egy hiteles aláírás nem mondható hamisítottnak.

Digitális aláírás során három fázist különböztetünk meg: kulcsgenerálást, az aláíró fázist és az aláírást ellenőrző fázist. A bemutatásra kerülő sémáknál mindhárom fázist külön pontban mutatjuk be.

6.3.1. Az RSA aláírásséma

Az RSA aláírásséma az RSA kriptorendszernél (lásd 6.2.1. fejezet) bemutatott matematikai elméleten alapszik, biztonsága tehát azon az elven alapszik, hogy nem ismert hatékony algoritmus, mely egy szám prímtényezőit meghatározná. A sémát a $PKCS$ (Public Key Cryptography Standards) szabványcsalád keretén belül alkalmazzák, melyet az RSA Laboratories publikált.

Kulcsgenerálás

A kulcsgenerálás a hagyományos két $q_1,\ q_2$ prímszámok meghatározása mellett megengedi ennek egy általános formáját is, azt, amikor a modulus több $k > 2$ prímszám szorzatából áll: $m = q_1 \cdot q_2 \cdot \cdots \cdot q_k$. Az itt bemutatásra kerülő kulcsgenerálás két prímszám szorzataként határozza meg az RSA-modulust. Ezt a fázist általában az aláíró végzi:

– választ két $q_1,\ q_2$ tetszőleges prímszámot és meghatározza: $m = q_1 \cdot q_2$, majd $\phi(m) = (q_1 - 1) \cdot (q_2 - 1)$ értékeket, ahol ϕ az Euler phi-függvény, lásd a 2.5. értelmezést,
– választ egy e számot, ahol $1 \leq e \leq \phi(m)$ és $gcd(e,\ \phi(m)) = 1$,
– meghatározza a d számot, ahol $1 \leq d \leq \phi(m)$ és $d \cdot e = 1 \pmod{\phi(m)}$,
– a megfelelő kulcsok ezek után a következőek lesznek:
  • a nyilvános kulcs: $(e,\ m)$;
  • a titkos kulcs: (d).

Aláírás

Feltételezve, hogy az aláíró az x dokumentumot szeretné aláírni, akkor a következőképpen jár el:

– meghatározza a $h(x) = m_1$ hash értéket, ahol h egy hash függvény, mely nyilvános és megválasztása szabvány által előírt,
– a dokumentum aláírt értéke, a titkos kulcs ismeretében, a következő lesz:

$$s = m_1^d \pmod{m}.$$

Aláírás-ellenőrzés

Az aláírást bárki leellenőrizheti, ha a nyilvános kulcs ismeretében az aláírt dokumentumon a következőket végrehajtja:

– meghatározza $m_2 = s^e = \pmod{m}$ értéket,
– meghatározza a $h(x) = m_1$ hash értéket,
– ha $m_1 = m_2$, akkor az aláírást el lehet fogadni.

6.11. példa. Határozzuk meg a *signed* dokumentum esetében a digitális aláírás értékét, majd ellenőrizzük le.

– Feltételezzük, hogy a *signed* dokumentum hash értéke, az MD5 hash függvénnyel számolva, tizenhatos számrendszerben:

$$43ca94dd646dbbaa78034918d61043e9.$$

A tizenhatos számrendszerbeli számjegyek száma 32, ezért a hash értéket átalakítjuk egyetlen számmá, azaz 16^{32}-es számrendszerbe, majd ennek az értéknek fogjuk meghatározni a digitális aláírását. A kapott szám tehát:

$$m_1 = 210\,288\,051\,729\,042\,553\,126\,429\,092\,061\,010\,046\,004.$$

– A kulcsgeneráláshoz szükséges két prímszám legyen:

$$q1 = 413\,276\,761\,067\,542\,934\,567,$$
$$q2 = 646\,835\,438\,970\,871\,021\,591.$$

– A nyilvános kulcs:

$$e = 65\,537,$$
$$m = 267\,322\,055\,161\,583\,912\,881\,915\,809\,981\,317\,357\,236\,097.$$

– A titkos kulcs:

$$d = 52\,035\,147\,438\,795\,275\,594\,260\,892\,878\,767\,344\,849\,813.$$

– Ahol meghatároztuk

$$\begin{aligned}\phi(m) &= (q1-1)\cdot(q2-1)\\ &= 267\,322\,055\,161\,583\,912\,880\,855\,697\,781\,278\,943\,279\,940,\\ &\text{és fennáll, hogy:}\\ d\cdot e &= 1 \pmod{\phi(m)}.\end{aligned}$$

– A digitális aláírás:

$$\begin{aligned}s &= m_1^d \pmod m\\ &= 237\,905\,405\,218\,980\,123\,252\,738\,810\,714\,172\,022\,852\,016.\end{aligned}$$

– Az aláírás ellenőrzése végett meghatározzuk:

$$\begin{aligned}m_2 &= s^e \pmod m\\ &= 210\,288\,051\,729\,042\,553\,126\,429\,092\,061\,010\,046\,004,\end{aligned}$$

majd a kézhez kapott dokumentumra meghatározzuk a hash értéket, és ezt összehasonlítjuk a most kiszámolt m_2 értékkel. Ha a két érték megegyezik, akkor elfogadjuk az aláírást.

6.3.2. Az ElGamal aláírásséma

Az ElGamal-aláírásséma hátterében az ElGamal-kriptorendszernél (lásd 6.2.3. fejezet) bemutatott matematikai elmélet áll, lásd [5], biztonsága tehát azon az elven alapszik, hogy a diszkrét logaritmus értékének a meghatározására nem ismert hatékony algoritmus. Gyakorlatban ritkábban alkalmazzák, inkább egy változatát használják, a DSA-t (lásd 6.3.3. fejezet).

Kulcsgenerálás

A kulcsgenerálást az aláíró végzi és megegyezik az 6.2.1. fejezetben bemutatott kulcsgenerálással. Az aláíró tehát:

- választ egy tetszőleges p prímszámot, és meghatározza q egy primitív gyökét,
- választ egy a egész számot, ahol $1 \leq a \leq p-1$, és meghatározza: $A = q^a \pmod p$-t,
- a megfelelő kulcsok ezek után a következőek lesznek:
  - a nyilvános kulcs: $(p,\ q,\ A)$,
  - a titkos kulcs: (a).

Aláírás

Feltételezzük, hogy az aláíró az x dokumentumot szeretné aláírni. Ennek érdekében a következőképpen jár el:

- választ egy tetszőleges b számot, ahol

$$1 \leq b \leq p-1,\ gcd(b,\ p-1) = 1,$$

  és meghatározza:

$$B = q^b \pmod p,$$

- a nyilvános h hash függvény ismeretében kiszámolja a következő értéket:

$$s = (h(x) - a \cdot B) \cdot b^{-1} \pmod{p-1},$$

  ahol b^{-1} a b multiplikatív inverze $\pmod{p-1}$ szerint, azaz:

$$b * b^{-1} = 1 \pmod{p-1},$$

- ha $s = 0$, akkor újabb véletlenszerű értékeket választ,
- a $(B,\ s)$ lesz a dokumentum digitális aláírása.

Aláírás-ellenőrzés

Az aláírást ellenőrző által végzett számítások a következőek:

- meghatározza

$$\begin{aligned} m_1 &= q^{h(x)} \pmod p, \\ m_2 &= B^s \cdot A^B \pmod p, \end{aligned}$$

- ha a két érték megegyezik, akkor elfogadja a digitális aláírást.

Az algoritmus helyessége a következőkön alapszik (lásd 2.3. tétel):

$$\begin{aligned} s &= (h(x) - a \cdot B) \cdot b^{-1} \pmod{p-1} \quad \Leftrightarrow \\ h(x) &= b \cdot s + a \cdot B \pmod{p-1} \quad \Rightarrow \end{aligned}$$

$$\begin{aligned} q^{h(x)} &= q^{b \cdot s} \cdot q^{a \cdot B} \pmod p = \\ &= (q^b)^s \cdot (q^a)^B \pmod p = \\ &= B^s \cdot A^B \pmod p. \end{aligned}$$

6.12. példa. Határozzuk meg a *signed* dokumentum esetében a digitális aláírás értékét, majd ellenőrizzük le.

- Feltételezzük, hogy a *signed* dokumentum hash értéke az MD5 hash függvénnyel számolva, tizenhatos számrendszerben:

$$43ca94dd646dbbaa78034918d61043e9.$$

A tizenhatos számrendszerbeli számjegyek száma 32, ezért a hash értéket átalakítjuk egyetlen számmá, azaz 16^{32}-es számrendszerbe, majd ennek az értéknek fogjuk meghatározni a digitális aláírását. A kapott szám tehát:

$$h(x) = 210\,288\,051\,729\,042\,553\,126\,429\,092\,061\,010\,046\,004.$$

– A kulcsgeneráláshoz szükséges számok legyenek:

$$\begin{aligned} p &= 4\,652\,146\,565\,657\,438\,348\,243\,826\,542\,873\,652\,846\,527\,897, \\ q &= 3\,781\,225\,691\,566\,646\,572\,233\,517\,676\,112\,741\,555\,684\,204, \\ a &= 2\,950\,276\,152\,401\,146\,437\,263\,030\,030\,191\,517\,067\,990\,644, \\ A &= q^a \pmod p \\ &= 2\,262\,866\,096\,392\,156\,816\,788\,402\,564\,115\,310\,646\,097\,551. \end{aligned}$$

– Az aláírás generálásához legyen

$$b = 4\,115\,062\,545\,550\,359\,710\,337\,712\,508\,166\,194\,416\,002\,495,$$

ahol $gcd(b,\ p-1) = 1$, és meghatározzuk

$$\begin{aligned} B &= q^b \\ &= 3\,509\,103\,367\,372\,501\,865\,544\,992\,565\,488\,964\,516\,466\,593, \\ b^{-1} &= 1\,332\,048\,681\,176\,360\,485\,524\,703\,076\,632\,578\,852\,096\,223, \\ s &= 3\,888\,015\,361\,329\,484\,514\,280\,321\,015\,905\,739\,702\,267\,456. \end{aligned}$$

– Az aláírás ellenőrzéséhez meghatározzuk:

$$\begin{aligned} m_1 &= q^{h(x)} \\ &= 2\,928\,202\,808\,955\,414\,383\,459\,554\,178\,785\,315\,017\,648\,262, \\ m_2 &= B^s \cdot A^B \\ &= 2\,928\,202\,808\,955\,414\,383\,459\,554\,178\,785\,315\,017\,648\,262. \end{aligned}$$

A séma biztonságáról elmondható, hogy egy nem legális fél aláírást akkor tud hamisítani, ha sikerül megszereznie a titkos kulcsot, illetve ha sikerül ütközést találnia az alkalmazott nyilvános hash függvény esetében. Mindkét feladat megfelelő nehézségű. Az aláírónak továbbá ügyelnie kell arra, hogy a tetszőlegesen választott b számot minden egyes aláírás esetében más-más értéknek vegye. Ha két különböző dokumentum ugyanazzal a b értékkel van aláírva, akkor egy támadó fél direkt módon meg tudja határozni az a titkos kulcsot, lásd [5].

6.3.3. A DSA (Digital Signature Algorithm)

A DSA (Digital Signature Algorithm) az ElGamal-aláírásséma egy hatékony változata, melyet DSS (Digital Signature Standard) néven digitális aláírási szabványként tart számon a FIPS, lásd [5].

Az aláírás-ellenőrzés során csupán két moduláris hatványozást kell elvégezni, szemben az ElGamal-aláírással, ahol hármat kell végrehajtani. Ami viszont ennél is fontosabb, hogy az aláírás elkészítéséhez szükséges titkos kulcs mérete, ahhoz, hogy a rendszer megfelelő biztonságú legyen, elég, ha csupán 160 bit, míg az ElGamalnál ez nagyobb. Az eljárásnál alkalmazott hash függvény eredetileg az SHA-1 volt, de

más SHA típusú hash függvényt is szoktak használni, azzal a kitétellel, hogy a hash függvény kimenete 160 bit legyen.

Az aláírásséma három fázisának a lépéssorozatai a következőképpen határozhatóak meg:

Kulcsgenerálás

A kulcsgenerálást az aláíró végzi és a következőket teszi:

- választ egy nagy prímszámot: q -t, úgy, hogy
$$2^{159} < q < 2^{160},$$
- választ egy másik nagy prímszámot: p -t, úgy, hogy
$$2^{511+64\cdot t} < p < 2^{512+64\cdot t}, \text{ ahol } 0 \leq t \leq 8 \text{ és}$$
$$p-1 \text{ osztható legyen } q-\text{val},$$
- választ egy tetszőleges k számot az $1, 2, \ldots, p-1$ értékek közül, majd kiszámolja $Q = k^{(p-1)/q} \pmod p$-t, azzal a tulajdonsággal, hogy $Q \neq 1$,
- választ egy tetszőleges a számot az $1, 2, \ldots, q-1$ értékek közül, majd kiszámolja $A = Q^a \pmod p$,
- a megfelelő kulcsok ezek után a következőek lesznek :
  - a nyilvános kulcs: $(p,\ q,\ Q,\ A)$;
  - a titkos kulcs: (a).

Aláírás

Feltételezzük, hogy az aláíró az x dokumentumot szeretné aláírni. Ennek érdekében a következőképpen jár el:

- választ egy tetszőleges b számot, ahol $1 \leq b \leq q-1,\ gcd(b,\ p-1) = 1$, és meghatározza:
$$B = (Q^b \pmod p) \pmod q,$$
- a nyilvános $h : \{0,\ 1\}^* \to \{1,\ 2,\ \ldots,\ q-1\}$ hash függvény és a titkos kulcs ismeretében kiszámolja a következő értéket:
$$s = (h(x) + a \cdot B) \cdot b^{-1} \pmod q,$$
ahol b^{-1} a b multiplikatív inverze $\pmod q$ szerint, azaz:
$$b * b^{-1} = 1 \pmod q,$$
- a $(B,\ s)$ lesz a dokumentum digitális aláírása.

Aláírás-ellenőrzés

Ismerve a $(p,\ q,\ Q,\ A)$ nyilvános kulcsot, az aláírást ellenőrző által végzett számítások a következőek:

- ha a következő két egyenlőtlenség valamelyike nem áll fenn, akkor nem fogadja el a digitális aláírást
$$1 \leq B \leq q-1,$$
$$1 \leq s \leq q-1,$$
- ha az egyenlőtlenségek fennállnak, akkor meghatározza:
  - s^{-1}-t, ahol $s \cdot s^{-1} = 1 \pmod q$,
  - $B_1 = (Q^{s^{-1}\cdot h(x) \pmod q} \cdot A^{s^{-1}\cdot B \pmod q} \pmod p) \pmod q$,

- ha $B = B_1$, akkor elfogadja az aláírást.

Az algoritmus helyessége a következőkön alapszik:

$$\begin{aligned} B_1 &= (Q^{s^{-1}\cdot h(x) \pmod q} \cdot A^{s^{-1}\cdot B \pmod q} \pmod p) \pmod q \\ &= (Q^{s^{-1}\cdot h(x) \pmod q} \cdot Q^{a\cdot s^{-1}\cdot B \pmod q} \pmod p) \pmod q \\ &= (Q^{s^{-1}\cdot (h(x)+a\cdot B) \pmod q} \pmod p) \pmod q \\ &= (Q^{s^{-1}\cdot s\cdot b \pmod q} \pmod p) \pmod q \\ &= (Q^b \pmod p) \pmod q. \end{aligned}$$

A DSA biztonsága két különböző diszkrét logaritmálási feladat nehézségén alapszik, ahol a q és p paramétereket megfelelő nagyságrenddel kell megválasztani, ahogy azt a kulcsgenerálásnál specifikáltuk.

6.13. példa. Végezzük el a kulcsgenerálás fázisát a DSA aláírásséma esetében, majd számítsuk ki az aláírás értékét és ellenőrizzük le.

– A kulcsgeneráláshoz szükséges paraméterek legyenek:
  - $q = 359\,111$ prímszám,
  - $p = 1\,073\,741\,891$, ahol q osztja $p - 1 = 1\,073\,741\,890$, mert
$$p - 1 = (2) \cdot (5) \cdot (13) \cdot (23) \cdot (359\,111),$$
  - $x = 107\,645$, tetszőleges, ekkor $(p-1)/q = 2990$ és
$$Q = 107\,645^{2\,990} = 263\,319\,279 \pmod{1\,073\,741\,891} \neq 1,$$
  - $a = 5687$,
  - $A = 263\,319\,279^{5687} = 268\,777\,487 \pmod{1\,073\,741\,891}$,
  - a nyilvános kulcs: $(359\,111,\ 1\,073\,741\,891,\ 263\,319\,279,\ 268\,777\,487)$;
  - a titkos kulcs: (5687).

– Feltételezve, egy $h(x) = 3421$ hash értékkel rendelkező dokumentumot szeretnénk aláírni, az aláírás a következő számításokat jelenti:
  - legyen $b = 6789$, amire kiszámoljuk
$$B = 263\,319\,279^{6789} = 307\,960 \pmod p \pmod q,$$
  - meghatározzuk $b^{-1} = 50\,357$, ahol $50\,357 \cdot 6789 = 1 \pmod{359\,111}$,
$$s = 50\,357 \cdot (3421 + 5687 \cdot 307\,960) = 124\,198 \pmod{359\,111}.$$

– Az aláírás-ellenőrzés lépései a következőek:
  - leellenőrizzük a két egyenlőtlenséget,
  - meghatározzuk $s^{-1} = 206\,070$-t, ahol
$$124\,198 \cdot 206\,070 = 1 \pmod{359\,111},$$
  - kiszámoljuk
$$\begin{aligned} s^{-1}\cdot h(x) &= 206\,070 \cdot 3421 = 30\,577 \pmod{359\,111}, \\ s^{-1}\cdot B &= 206\,070 \cdot 307\,960 = 298\,613 \pmod{359\,111}, \\ B_1 &= Q^{(s^{-1}\cdot h(x) \pmod q)} \cdot A^{(s^{-1}\cdot B \pmod q)} \\ &= 263\,319\,279^{30\,577} \cdot 268\,777\,487^{298\,613} \\ &= 307\,960 \pmod{1\,073\,741\,891} \pmod{359\,111}, \end{aligned}$$
  - összehasonlítjuk a kapott B_1 értéket B-vel.

7. FEJEZET

KRIPTOGRÁFIAI PROTOKOLLOK

Ebben a fejezetben több olyan protokoll bemutatására kerül sor, melyek gyakorlati szempontból különös jelentőséggel bírnak, ugyanakkor az ismertetett kriptográfiai algoritmusokat szakszerűen alkalmazva próbálják megvalósítani a biztonságos kliens–szerver kommunikációt.

Ahhoz, hogy a protokollok működési elve érthető legyen, először néhány alapfogalmat tisztázunk, mely a hálózati kommunikáció megértéséhez elengedhetetlenül szükséges. Bővebb leírását találunk a [13]-ban.

A számítógépek hálózatban való kommunikációjára két fontosabb protokollstruktúrát ismerünk, az egyik az OSI (Open System Interconnection), a másik a TCP/IP (Transmission Control Protocol/Internet Protocol). Mindkét rendszer rétegekre lebontva valósítja meg a kommunikációt. A TCP/IP esetében 5 réteget, míg az OSI esetében 7 réteget különböztetünk meg.

A TCP/IP esetében ezek a következők lennének, ahol a rétegek hierarchikusan kapcsolódnak egymáshoz és mindegyik réteg csak a vele szomszédos réteggel képes kommunikálni:

1. **Alkalmazási réteg** (Application Layer): ha a felhasználó által elindított program hálózaton keresztül szeretne adatot továbbítani, akkor az alkalmazási réteg lesz az, amelyik ezt az igényt jelzi, majd az adatot a szállítási rétegnek továbbítja.
2. **Szállítási réteg** (Transport Layer): mivel az adatszállítás többfajta protokollal is megoldható (például a TCP – Transport Control Protocol vagy az UDP – User Datagram Protocol), a szállítási réteg az alkalmazási rétegtől beérkezett adat elejére egy header adatot csatol, ebben jelezve, hogy melyik szállítási protokollal továbbítja az adatokat, majd a hálózati rétegnek átküldi az adatot.
3. **Hálózati réteg** (Network Layer): a szállítási rétegtől beérkezett adatot egy újabb header adattal egészíti ki, melyben többek között a célszámítógépre vonatkozó információt tárolja el. Ez is többféle protokollal végezhető, legismertebb az IP-internet protokoll.
4. **Adatkapcsolati réteg** (Data Link Layer): ez a réteg felelős a szomszédos hálózati pontok közötti adatok cseréjéért, illetve a fizikai rétegnél adódó esetleges hibák kijavításáért. Adott esetben a továbbítandó adatot keretekre, azaz kisebb egységekre bontva küldi át.
5. **Fizikai réteg** (Physical Layer): ez a réteg végzi a beérkező bitsorozatok fizikai jelekké való konvertálását, majd továbbítását a rendelkezésre álló fizikai egységen keresztül.

Az adatok biztonságáért felelős, ismertebb hálózati kommunikáció protokolljait ezek után csoportosíthatjuk, aszerint, hogy melyik rétegben valósul meg az adatok titkosítása.

- az alkalmazási rétegben: PGP, S/MIME, HTTPS, KERBEROS, SSH,
- a szállítási rétegben: SSL vagy TLS,
- a hálózati rétegben: IPSec, VPN,

– az adatkapcsolati rétegben: PPP, Radius.

7.1. Az alkalmazási réteg protokolljai

7.1.1. A PGP (Pretty Good Privacy) protokoll

A PGP protokoll egy OpenPGP szabvány, melyet Philip R. Zimmermann 1991-ben tervezett adattitkosítás és hitelesítés céljára. Leggyakrabban e-mailek aláírására, titkosítására használják, lásd a [12]-ben. Nagy népszerűségnek örvend, mert ingyenesen elérhető, ugyanakkor egyike a legstabilabb halózatbiztonsági protkolloknak. Kezdetben csupán üzenetek és csatolt állományok titkosítására alkalmazták, de mára számos funkcióval kiegészítették, mint például mappák, merevlemezek titkosítása, digitális aláírások kezelése stb. Napjainkban a legtöbb operációs rendszerbe beépítve található.

A protokoll az **adattikosítását** valamely szimmetrikus titkosítási eljárással végzi, a titkosításhoz szükséges kulcsot, a mesterkulcsot, pedig minden egyes alkalommal, azaz üzenetenként újra generálja. A generált kulcs, a mesterkulcs egy 128 bitből álló véletlenszerűen generált érték lesz, melyet Diffie–Hellman-kulcscserével határoz meg, vagy az RSA esetében a címzett fél publikus kulcsával titkosít. A generált mesterkulcs és a tulajdonképpeni üzenet titkosítása csak ezek után történik, valamely szimmetrikus rendszerrel, mint például CAST-128, IDEA, vagy triple DES. Az üzenet feldarabolt blokkjait CFB technikával dolgozza fel. A titkosított üzenetet elküldhető valamely címzetthez, vagy lokálisan is eltárolható. Az üzenet visszafejtése érdekében a címzett fél, az RSA esetében a privát kulcsával előbb visszafejti a titkosított mesterkulcsot.

A protokoll a **hitelesítést** valamely digitális aláírásséma alapján végzi, amely lehet akár az RSA, akár a DSA aláírási séma. A hash értéket, amelyet létrehoz, az SHA-1 hash függvénnyel határozza meg, ahol a létrehozott 160 bites értéket (lenyomatot) a címzett fél titkos kulcsával titkosítja. Az üzenetet aztán a létrehozott aláíráshoz csatolva küldi el. Lehetőség van arra is, hogy a létrehozott lenyomatot az üzenettől elválasztva küldje el. A címzett fél a küldő nyilvános kulcsával visszafejti az aláírást, a csatolt üzenetnek meghatározza a hash értékét és a két értéket összehasonlítja.

Helymegtakarítás végett a hitelesítés és titkosítás során a rendszer általában tömöríti az üzenetet. Üzenetek titkosítás esetében a rendszer a tömörítést elvégzi a titkosítás előtt. Ez megnehezíti a kriptoanalízist. Hitelesítés esetében, pedig a tömörítés az aláírás generálása után valósul meg, ez elsősorban azért van így mert ha az aláírt dokumentumot el szeretnénk tárolni akkor célszerűbb annak a tömörítettlen változatát megőrizni.

7.1.2. Az SSH (Secure Shell) protokoll

Az SSH az alkalmazási réteg szintjén oldja meg a biztonságos adatátvitelt két egymástól távol levő számítógép között. Tipikus felhasználása a távoli számítógépre való bejelentkezés, majd ezen távoli számítógép használata. Állományok átvitelére is alkalmas az SFTP (SSH File Transfer) és SCP (Secure Copy) protokollokon keresztül. Az SSH a biztonságos adatátvitel érdekében nyilvános kulcsú kriptográfiai

algoritmusokat használ. A protokoll első verzióját, az SSH1-t 1995-ben Tatu Ylönen, a Helsinki Műszaki Egyetem kutatója tervezte, mely nagyon gyorsan népszerűvé vált, részletes leírását lásd a [27]-ben.

A protokoll a következő lépéssorozatból áll:

7.1. protokoll.

1. A kliens felkéri a szervert, hogy hitelesítse önmagát.
2. A szerver válaszként elküldi a nyilvános kulcsát, a tanúsítványával együtt.
3. A kliens a kapott információ alapján eldönti, hogy a szerver hiteles-e. Abban az esetben, ha a hitelesítés körül minden rendben történik, a kliens egy véletlenszámot generál, amit a kapott nyilvános kulccsal titkosít, majd elküldi ezt a szervernek.
4. A szerver a privát kulccsal visszafejti a titkosított véletlenszámot, melyet a továbbiakban a két fél, mint mesterkulcsot, az üzenetek titkosítására használ fel. A titkosítási algoritmus valamelyik ismert titkos kulcsú algoritmus lesz, mint például IDEA, DES, triple DES stb.
5. A szerver felkéri a klienst, hogy hitelesítse önmagát. Erre több módszer is ismert:
   a) publikus kulcson alapuló autentikáció: a rendszer biztosítja vagy az RSA kulcspár, vagy a digitális aláírás alkalmazását;
   b) interaktív autentikáció: a szerver kérésére a kliens információkat küld át, amik alapján a kliens hitelesíthető, ilyenek a one-time-password vagy a SecureID autentikációk;
   c) jelszó alapú autentikáció: a rendszer lehetővé teszi a jelszócserét.

7.2. A szállítási réteg protokolljai

7.2.1. Az SSL (Secure Socket Layer) protokoll

Az SSL egy kriptográfiai protokoll, mely az internetes kommunikáció biztonságosságát hivatott megoldani. Utódját TLS (Transport Layer Security) protokollnak hívják, melyet az SSL 3.0 verzióból fejlesztett ki az Internet Engineering Task Force (IETF), és mely a következő feladatokat képes biztonságosan ellátni: webböngészés, e-mail, fax, üzenetküldés stb. Habár a két protokoll különbözik egymástól, lényegében nincs nagy eltérés közöttük, ezért a továbbiakban csak a TLS-t mutatjuk be.

7.2.2. A TLS (Transport Layer Security) protokoll

A TLS protokoll, lásd a [6]-ban, a szállítási rétegben valósul meg, és egy hálózati, kliens-szerver kommunikációt tesz lehetővé, oly módon, hogy megvédi a kommunikáló feleket az információ lehallgatásától, hamisításától, megváltoztatásától stb. Leggyakrabban a HTTP alapú kommunikációban, azaz egy webszerver és egy webböngésző között alkalmazzák. Általában csak a szerver hitlesítése történik meg, a kliens hitelesítése elmarad. Gyakorlatban ez azt jelenti, hogy egy webböngésző mindig tudni fogja, hogy kivel kommunikál.

A protokoll tulajdonképpen háromfajta szolgáltatást is biztosít:

– adattitkosítást, ami a szerver és kliens között történő biztonságos adatátvitelt jelenti, kizárva annak a lehetőségét, hogy a legális feleken (kliens, szerver) kívül más is értelmezni tudja az átküldött információt,
– a szerver és a kliens hitelesítését, melyet standard nyilvános kulcsú kriptográfiai algoritmusok segítségével old meg,
– az üzenetek sértetlenségét, mely a kommunikációs csatornán továbbküldött adatcsomagok változatlanságát jelenti.

A protokoll főbb alprotokolljai a TLS handshake (kézfogás), a TLS record, a TLS alert (figyelmeztető) és a TLS „change cipher spec" protokoll.

A TLS kézfogás protokoll felelős a kliens és szerver egymást hitelesítő lépéssorozatáért, az alkalmazásra kerülő tömörítő és titkosítási eljárás kiválasztásáért, illetve az alkalmazott kulcsértékeknek a meghatározásáért. Ezek megállapítása azelőtt történik, hogy a kommunikáló felek között egyetlen bájt is továbbítódna. Az alkalmazott titkosítási eljárás egy szimmetrikus titkosítási rendszer lesz, leggyakrabban a triple DES-t vagy az AES-t használják. A szimmetrikus kriptorendszernél alkalmazott kulcs értékének a meghatározására nyilvános kulcsú kriptográfiai algoritmusokat alkalmaz, mint például RSA, Diffie–Hellman. Részletesebben a következő lépéssorozatból áll:

7.2. protokoll.

1. A kliens elküldi a szervernek a TLS protkolljának verziószámát, és megjelöli az általa alkalmazható titkosítási és tömörítési algoritmusok listáját.
2. A szerver válaszként megjelöli az általa választott TLS verzió számát, a titkosítási, illetve tömörítő algoritmusokat. Ugyanakkor elküldi tanúsítványát, mely a Tanúsító Hatóság (lásd 6. fejezet elejét) által kell legyen kibocsátva. Opcionálisan kérheti a klienst, hogy ő is hitelesítse magát. Végül jelzi, hogy ezek után várja a kliens válaszát.
3. A hitelesítés, adott esetben kölcsönös hitelesítés után a kiválasztott titkosítási rendszer kulcsértékének, a mesterkulcsnak a meghatározása következik, valamilyen nyilvános kulcsú kriptorendszer segítségével. A kliens ennek érdekében egy véletlenszámot generál, melyet a szerver nyilvános kulcsával titkosítva elküld a szervernek.
4. A szerver a titkos kulcsával visszafejti a mesterkulcs értékét, és most, hogy mindkét fél birtokában van a mesterkulcsnak, következhet a tulajdonképpeni adattitkosítás. A meghatározott mesterkulcs az összes ezután következő adatcsomag titkosítására alkalmas lesz, melyet a kliens és a szerver egyaránt végezhet.

A TLS record protokoll keretén belül történik az adatcsomagok tömörítése és feldarabolása, majd a tulajdonképpeni adattitkosítás és az üzenetek hitelességének/sértetlenségének a leellenőrzése.

A TLS alert protokoll keretén belül a kapcsolat ideje alatt felmerülő hibák jelzése történik meg. Figyelmeztető hibák akkor adódhatnak, amikor a tanúsítvány ideje lejárt, illetve nem ismert a Tanúsító Hatóság kiléte stb. Fatális hibák esetén a TLS kapcsolat azonnali beszüntetése következik.

A TLS „change chiper spec" protokoll egyetlen üzenet továbbításáért felelős, melyben a kézfogás protokoll végét jelzi.

Számos, különböző profilú szervezet alkalmazza a TLS protokollt, például:

1. banki szervezetek, melyek klienseiknek meg szeretnék engedni, hogy távoli számítógépről felügyeljék, megnézzék, kiegyenlítsék számláikat,

2. akadémiai szervezetek, melyek diákjaiknak megengedik, hogy személyes információkat érhessenek el,
3. kereskedelmi társaságok, melyek megengedik, hogy klienseik személyes adataikat interneten keresztül érhessék el.

SZAKIRODALOM

[1] M. Agrawal – N. Kayal – N. Saxena: *Primes is in P.* 2004, Annals of Mathematics.

[2] A. Bege: *Bevezetés a számelméletbe.* Kolozsvár, 2002, Scientia Kiadó.

[3] A. Bege – Zoltán Kása: *Algoritmikus kombinatorika és számelmélet.* Cluj-Napoca, 2006, Presa Universitară Clujeană.

[4] D.J. Bernstein – H.W. Lenstra – J. Pila: *Detecting perfect powers by factoring into coprimes.* 2007, Matehematics of computation.

[5] J.A. Buchmann: *Inroduction to cryptography.* New York, Berlin, Heidelberg, 2002, Springer Verlag.

[6] L. Buttyán – T. Vajda: *Kriptográfia és alkalmazásai.* Budapest, 2004, Typotex.

[7] C. Cohen: *A course in computational algebraic number theory.* 1993, Springer Verlag.

[8] T.H. Cormen – C.E. Leiserson – R.L. Rivest: *Algoritmusok.* Budapest, 2001, Műszaki Könyvkiadó.

[9] S. Crivei – A. Marcus – Cs. Szanto: *Computational Algebra with Applications to Coding Theory and Cryptography.* Cluj-Napoca, 2006, EFES.

[10] J. Daemen – V. Rijmen: *The Design of Rijndael, AES-The Advanced Encryption Standard.* 2002, Springer Verlag.

[11] R. Freud – E. Gyarmati: *Számelmélet.* Budapest, 2000, Nemzeti Tankönyvkiadó.

[12] J. Ködmön: *Kriptográfia.* Budapest, 2002, ComputerBooks.

[13] M. Kizza: *Computer Network Security.* New York, Berlin, Heidelberg, 2005, Spinger Verlag.

[14] D.E. Knuth: *A számítógép-programozás művészete.* Budapest, 1987-1988, Műszaki Könyvkiadó.

[15] N. Koblitz: *A course in number theory and cryptography.* New York, Berlin, Heidelberg, 1994, Springer Verlag.

[16] Z. Kátai: *Algoritmusok felülnézetből.* Kolozsvár, 2007, Scientia Kiadó.

[17] D.H. Lehmer: *Mathematical methods in large-scale computing units.* 1951, Proceedings of 2nd Symposium on Large-Scale Digital Calculating Machinery.

[18] H.W. Lenstra – C. Pomerance: *Primality testing with gaussian periods.* 2003, Private communication.

[19] L. Lovász: *Algoritmusok bonyolultsága.* Budapest, 2001, Nemzeti Tankönyvkiadó.

[20] A.J. Menezes – P.C. van Oorschot – S.A. Vanstone: *Handbook of applied cryptography.* Boca Raton, Florida, 1997, CRC Press.

[21] P.L. Montgomery: *A block Lánczos algorithm for finding dependencies over GF(2)*. 1995, Advances in Cryptology, Eurocrypt.

[22] C. Pomerance: *A Tale of Two Sieves*. 1996, Notices of the AMS.

[23] L. Rónyai – G. Ivanyos – R. Szabó: *Algoritmusok*. Budapest, 2004, Typotex.

[24] K. Rosen: *Elementary number Theory and it's applications*. 1992, Addison-Wesley.

[25] A. Salomaa: *Public-key cryptography*. New York, Berlin, Heidelberg, 1996, Springer Verlag.

[26] B. Schneier: *Applied cryptography: protocols, algorithms, and source code in C*. New York, 1996, John Wiley & Sons, Inc.

[27] S. Vaudenay: *A classical introduction to cryptography*. New York, Berlin, Heidelberg, 2006, Springer Verlag.

TÁRGYMUTATÓ

A SZERZŐRŐL

Márton Gyöngyvér 1972. január 7-én született Marosvásárhelyen. Középiskolai tanulmányait a marosvásárhelyi Bolyai Farkas Elméleti Líceumban végezte 1986–1990 között, egyetemi tanulmányait pedig a Kolozsvári Babeş–Bolyai Egyetem Informatika Karán 1990–1995 között.

1995–2003 között informatika tanár a marosvásárhelyi Bolyai Farkas Elméleti Líceumban. 2001-től tanít a Sapientia Erdélyi Magyar Tudományegyetem Műszaki és Humántudományok Kara Matematika–Informatika Tanszékén, ahol 2003-tól főállású tanársegéd.

Tudományos érdeklődési területe a kriptográfia, de emellett funkcionális és logikai programozással is foglalkozik.

ABSTRACT

The book of Fundamentals of cryptography is an introduction in the science of cryptography. The author concentrates on the practical background of the cryptography. The included subjects are the basic parts of the university's curriculum. The first chapter are presenting the basic cryptography algorithms, like Euclidean algorithms and its applications, primality testings, integer factorizations, discrete logarithm's problem. The second and third chapters are presenting the major private and public key cryptosystems and the fourth, the digital signatures schemes. The last chapter presents some cryptography protocols that are used in computer network security. All subjects are presented in an adequate didactic and academic style.

REZUMAT

Cartea „Elementele criptografiei” prezintă principiile fundamentale ale criptografiei. Autorul se concentrează mai ales asupra practicii securităţii informaţiei. Subiectele tratate constituie materia de învăţământ universitar. Primul capitol prezintă implementarea algoritmilor necesare în domeniul respectiv (algoritmul lui Euclid şi aplicaţiile lui, teste de primalitate, factori-zarea numerelor întregi, problema logaritmului discret). În al doilea şi al treilea capitol sunt tratate criptosistemele cu cheie privată respectiv cu cheie publică. Ultimul capitol prezintă acele protocoale care sunt folosite în securitatea reţelelor de calculatoare. Subiectele tratate sunt prezentate adecvat din punct de vedere didactic şi ştiinţific.