Polinomok alkalmazásai

Diszkrét modellek alkalmazásai

1. Shamir-féle titokmegosztás

1.1. Lagrange-interpoláció

Tudjuk, hogy egy n-edfokú polinomnak legfeljebb n gyöke lehet (legalábbis nullosztómentes gyűrűben, például számoknál). Ez azt is jelenti, hogy ha egy *legfeljebb* n-edfokú polinomnak $n+1$ gyöke van, akkor az csak a nullpolinom (0) lehet (mert annak bármely elem gyöke).
Ezt az állítást általánosíthatjuk gyökök helyett helyettesítési értékekre:
1.1. Tétel. *Ha két legfeljebb n-edfokú polinom $n+1$ helyen megegyezik, akkor a két polinom egyenlő.*
Például ha p és q legfeljebb másodfokú polinomok, és három helyen megegyeznek, pl. $p(1) = q(1)$, $p(2) = q(2)$ és $p(5) = q(5)$, akkor a tétel alapján $p = q$. Lineáris polinomokhoz két érték is elég (szemléletesen: két pont meghatároz egy egyenest), konstans polinomokhoz pedig egy is elegendő.
(A tétel bizonyítása nagyon egyszerű: vegyük a két polinom különbségét, az ezen az $n+1$ helyen 0-t vesz fel, vagyis $n+1$ gyöke van, azaz csak a nullpolinom lehet.)
Tehát egy n-edfokú polinomot egyértelműen meghatározza $n+1$ helyettesítési értéke.
De hogyan számíthatjuk ki ezt az egyértelműen létező polinomot az $n+1$ értékből? Első lépésként nézzük azt a speciális esetet, amikor majdnem minden érték nulla, kivéve egyet. Például keressük meg azt a p másodfokú polinomot, amelyre $p(1) = 0$, $p(2) = 0$ és $p(5) = 8$. Az első két feltételt könnyű teljesíteni: $p := (x-1)(x-2)$. De a harmadik érték még nem jó: $p(5) = (5-1)(5-2) = 12 \neq 8$. Ezért szorozzuk meg a polinomot egy olyan c-vel, hogy $c \cdot p(5) = 8$ legyen, azaz $c := 8/p(5) = 8/12 = 2/3$, mert ez a gyököket nem változtatja meg, viszont a harmadik értéket "megjavítja". Tehát legyen inkább $p := 2/3 \cdot (x-1)(x-2) = 2x^2/3 - 2x + 4/3$.
Hogyan általánosíthatjuk ezt tetszőleges (nem feltétlenül nulla) értékekre? Például keressük meg azt a p másodfokú polinomot, amelyre $p(1) = 4$, $p(2) = 2$ és $p(5) = 8$. Ehhez alkalmazzuk a fenti módszert háromszor, és adjuk össze az eredményeket:

$$p_1(1) = 4,\ p_1(2) = 0,\ p_1(5) = 0 \implies p_1 = 4 \cdot \frac{(x-2)(x-5)}{(1-2)(1-5)} = x^2 - 7x + 10$$

$$p_2(1) = 0,\ p_2(2) = 2,\ p_2(5) = 0 \implies p_2 = 2 \cdot \frac{(x-1)(x-5)}{(2-1)(2-5)} = -\frac{2}{3}x^2 + 4x - \frac{10}{3}$$

$$p_3(1) = 0,\ p_3(2) = 0,\ p_3(5) = 8 \implies p_3 = 8 \cdot \frac{(x-1)(x-2)}{(5-1)(5-2)} = \frac{2}{3}x^2 - 2x + \frac{4}{3}$$

$$p(1) = 4, \quad p(2) = 2, \quad p(5) = 8 \implies p = p_1 + p_2 + p_3 \qquad = x^2 - 5x + 8$$

Ezt a módszert hívjuk *Lagrange-interpolációnak*. Általában:

1.2. Tétel (Lagrange-interpoláció)**.** *Ha egy legfeljebb n-edfokú p polinom $n+1$ különböző helyen adott $p(x_1) = y_1$, $p(x_2) = y_2$, ..., $p(x_{n+1}) = y_{n+1}$ értékeket vesz föl, akkor:*

$$p = \sum_{i=1}^{n+1} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{x - x_j}{x_i - x_j}.$$

A módszer *tetszőleges* $x_1, \ldots, x_{n+1}$ és $y_1, \ldots, y_{n+1}$ értékekre működik (ha az x_i-k mind különbözőek), azaz nem kell előre tudnunk, hogy létezik-e ilyen polinom, mert a fenti módszerrel mindenképpen kapunk egy ilyet (és az egyértelmű).

Az osztások miatt a Lagrange-interpoláció csak test fölött működik, pl. $\mathbb{R}[x]$-ben vagy $\mathbb{Q}[x]$-ben, de *nem* $\mathbb{Z}[x]$-ben (a fenti példában is törtekkel kellett számolni, hiába volt minden érték egész).

A Lagrange-interpolációra a Sage-nek van egy beépített függvénye:

```
sage: P.<x> = QQ[]    # (ZZ nem jó: test kell)
sage: P.lagrange_polynomial([(1,4), (2,2), (5,8)])
x^2 - 5*x + 8
```

1.2. Titokmegosztás

Tegyük fel, hogy van egy titkos információ (egy szám, például egy titkosításhoz használt kulcs), amit szeretnénk megosztani több ember között (például egy vállalat munkatársai között).

Ennek a legegyszerűbb módja, ha minden résztvevőnek odaadjuk a titkot. Ezzel persze az a probléma, hogy ha bármelyikük visszaél vele, vagy ellopják tőle, akkor az egész közös titok kompromittálódik. Jobb lenne egy olyan megoldás, ahol külön-külön egyik résztvevő sem fér hozzá a titokhoz.

A másik véglet az, hogy a titkos információt egyenlő részekre osztjuk, és minden résztvevő kap egy-egy részt. Ekkor külön-külön egyikük sem tud visszaélni a saját részével – sőt, egy kisebb csoportjuk sem –, csak minden résztvevő együttes közreműködésével lehet hozzáférni a titokhoz. Ezzel viszont az a probléma, hogy ha bármelyikük elveszti a saját részét (vagy átmenetileg elérhetetlen), akkor a közös titkot nem lehet helyreállítani.

Van egy másik probléma is ezzel a módszerrel: bár semelyik résztvevő nem rendelkezik a *teljes* titokkal, de annak részeivel igen, így mégis közelebb vannak a titok megfejtéséhez. Például ha egy számzáras lakatnak négyszámjegyű kódja van, akkor aki semmit nem tud a kódról, annak $10^4 = 10000$ lehetőséget kell végigpróbálgatnia a kinyitásához (legrosszabb esetben). Ha viszont valaki ismeri a kód egyik számjegyét, akkor neki már csak $10^3 = 1000$ lehetőséget kell végigpróbálnia, azaz tízszer gyorsabban tudja kinyitni. Két számjegy ismeretében pedig százszor gyorsabb lesz.

Az ideális megoldás olyan lenne, ahol a titok helyreállításához az n résztvevőből k-ra van szükség (pl. ötből háromra); ha bármelyik k résztvevő jelen van, akkor vissza tudják állítani a titkot; de ha csak legfeljebb $k-1$ résztvevő van jelen, akkor nemhogy a teljes titkot nem tudják megszerezni, de semennyivel nem kerülnek közelebb a megfejtéséhez, mint ha egyetlen résszel sem rendelkeznének.

Erre ad megoldás a *Shamir-féle titokmegosztás*.

Vigyázat: az alábbiakban leírt megoldás ebben a formában még nem teljesen biztonságos, de elsőre így könnyebben érthető; a javított változatot utána fogjuk megadni.

Legyen n a résztvevők száma, és k a visszaállításhoz szükséges szereplők száma ($1 \leq k \leq n$). Legyen továbbá $S \in \mathbb{Z}$ a megosztandó titok, amit "elrejtünk" egy $k-1$-edfokú polinomba a következőképpen:

$$p = a_{k-1}x^{k-1} + \ldots + a_2x^2 + a_1x + S,$$

ahol a konstans tag az S titok, a többi együttható $(a_1, \ldots, a_{k-1})$ pedig véletlen egész számok. Ezt a polinomot kiértékeljük n különböző helyen, például $p(1), p(2), \ldots, p(n)$, és minden résztvevőnek adunk egyet-egyet (egy-egy (x_i, y_i) párt). Ekkor bármely k résztvevő a k helyettesítési értékből egyértelműen vissza tudja állítani a polinomot a Lagrange-interpoláció segítségével (lásd fent), és így megkapják a konstans tagját, S-et is. Viszont k-nál kevesebb résztvevő nem tudja visszaállítani a polinomot (hiszen a hiányzó értékeket bárhogy kitölthetik, mindig különböző polinomokat kapnak, tehát nem tudhatják, melyik volt az eredeti).

Például legyen $n = 5$ résztvevő, amelyből $k = 3$ kell a visszaállításhoz. Legyen a titok az $S = 67$ érték, és generáljunk még két véletlen együtthatót, pl. $a_1 = 38$ és $a_2 = 13$, amelyből megkapjuk a következő másodfokú polinomot: $p = 13x^2 + 38x + 67$. Ezt értékeljük ki öt helyen, például:

$$p(1) = 118, \quad p(2) = 195, \quad p(3) = 298, \quad p(4) = 427, \quad p(5) = 582,$$

és mindenkinek adjunk egyet-egyet. Ekkor bármelyik három résztvevő vissza tudja állítani p-t, például az 1-es, a 2-es és az 5-ös a következőképpen (Lagrange-interpolációval, lásd fent):

$$p = 118 \cdot \frac{(x-2)(x-5)}{(1-2)(1-5)} + 195 \cdot \frac{(x-1)(x-5)}{(2-1)(2-5)} + 582 \cdot \frac{(x-1)(x-2)}{(5-1)(5-2)} = 13x^2 + 38x + 67.$$

Valójában nincs szükség a teljes p polinomra, csak a konstans tagjára, S-re, ezért gyorsíthatjuk a számítást úgy, hogy csak a $p(0)$-t számítjuk ki $(x = 0)$, amely megegyezik a konstans taggal:

$$S = p(0) = 118 \cdot \frac{(-2)(-5)}{(1-2)(1-5)} + 195 \cdot \frac{(-1)(-5)}{(2-1)(2-5)} + 582 \cdot \frac{(-1)(-2)}{(5-1)(5-2)} = 67.$$

A fentiekben vázolt módszerrel azonban még mindig van egy probléma. Ha két résztvevő összefog, akkor a teljes titkot ugyan nem, de annak bizonyos tulajdonságait ki tudják számítani. Ugyanis a két értékükből, pl. $p(1)$-ből és $p(5)$-ből, és abból, hogy egy $p = ax^2 + bx + S$ alakú polinomról van szó, ahol $a, b \in \mathbb{Z}$, a következőket tudják levezetni:

$$\begin{aligned} p(5) &= 25a + 5b + S = 582 \\ p(1) &= a + b + S = 118 \\ p(5) - p(1) &= 24a + 4b = 464 \quad \Longrightarrow \quad b = 116 - 6a \\ \Longrightarrow \quad p(1) &= a + (116 - 6a) + S = 118 \quad \Longrightarrow \quad S = 5a + 2. \end{aligned}$$

Vagyis megtudták, hogy a titkos érték 5-tel osztva 2-t ad maradékul, tehát ha meg akarják fejteni S-et, akkor csak ötödannyi lehetőséget kell végigpróbálniuk.

Ezt a problémát úgy küszöbölhetjük ki, ha nem egész együtthatókat használunk, hanem egy testből vesszük azokat. Erre az egyik lehetőség a racionális számtest $(\mathbb{Q})$, de törtekkel számolni kényelmetlen, és a véletlenszám-generálás is problémás (például milyen eloszlást használunk?). Ezért a gyakorlatban nem $\mathbb{Q}$-t, hanem egy véges $\mathbb{Z}_m$ testet szokás használni, ahol m egy elég nagy prímszám (amelybe S összes lehetséges értéke belefér).

Például vegyük $\mathbb{Z}_{89}$-et, és nézzük ugyanazt a fenti $p = \overline{13}x^2 + \overline{38}x + \overline{67}$ polinomot. Az öt helyettesítési érték most a következő (ugyanazok, mint fent, csak modulo 89):

$$p(\overline{1}) = \overline{29}, \quad p(\overline{2}) = \overline{17}, \quad p(\overline{3}) = \overline{31}, \quad p(\overline{4}) = \overline{71}, \quad p(\overline{5}) = \overline{48}.$$

A visszaállítás is pontosan ugyanúgy történik, mint fent, csak $\mathbb{Q}$ helyett $\mathbb{Z}_{89}$-ben kell számolni.

A fent leírt probléma itt már nem jelentkezik, mert ugyan itt is ki lehet számítani, hogy $S = \overline{5}a + \overline{2}$, de itt ebből semmilyen információt nem tudunk meg S-ről, mert testben vagyunk, ahol az ilyen egyenletek bármilyen $S \in \mathbb{Z}_{89}$-re megoldhatók: $a = \overline{5}^{-1}(S - \overline{2})$.

2. Cyclic redundancy check (CRC)

Amikor adatot továbbítunk egy fizikai csatornán keresztül (például vezetékben vagy rádiójelekkel), akkor előfordulhatnak adatátviteli hibák, azaz hogy nem pontosan ugyanaz az adat érkezik meg, mint amit elküldtünk. Hogyan tudjuk észrevenni az ilyen hibákat?

Ehhez *redundanciát* használunk: az üzenetet kiegészítjük egy új résszel, amelyet teljes egészében belőle számítunk ki, azaz új információt nem tartalmaz, de segít ellenőrizni, hogy az átvitt adat helyes-e. Ha ugyanis a továbbítás után az ellenőrző rész nem felel meg az üzenetnek, akkor biztosak lehetünk benne, hogy vagy az üzenet, vagy az ellenőrző rész (vagy mindkettő) hibás.

Erre egy egyszerű példa a *paritásbit*: a küldendő üzenetet, ami egy bitsorozat, egy új bittel egészítjük ki a következőképpen: ha az üzenetben páros számú 1-es bit volt, akkor 0-t írunk a végére, ha pedig páratlan, akkor 1-et. Így a végeredményben mindenképpen páros számú 1-es bit lesz. Például: $0101 \to 01010$, ill. $1101 \to 11011$. Ha a csatorna másik végén azt kapjuk, hogy 11010, akkor az biztosan hibás, mert páratlan sok 1-est tartalmaz. De hogy hol a hiba, azt nem tudjuk, például a fenti két bitsorozat bármelyikétől csak egy bitben tér el. Ezzel a módszerrel mindig észrevesszük, ha egyetlen bit romlik el, de kettőt már nem (mert attól páros marad az 1-esek száma).

Az alábbiakban bemutatjuk a *Cyclic Redundancy Check (CRC)* nevű módszert, amelyet számtalan helyen használnak hibadetektálásra (pl. Ethernet, USB, Wi-Fi).

A CRC a bitsorozatokat polinomokként kezeli, ahol minden együttható egy-egy bit (0 vagy 1). Például az 101011 bitsorozat az $x^5 + x^3 + x + 1$ polinomnak felel meg (az utolsó bit a konstans tag, az azelőtti az elsőfokú tag, és így tovább). A polinomok $\mathbb{Z}_2[x]$-ben vannak, azaz minden számítás modulo 2 történik, pl. $1+1=0$, $0-1=1$ stb. Ebből következik, hogy bármely polinomra $p+p=0$ (pl. $(x^2+1)+(x^2+1)=(1+1)x^2+(1+1)=0$), tehát $p=-p$, ezért az összeadás megegyezik a kivonással: $p-q=p+(-q)=p+q$. Ez az összeadás/kivonás leírható a bitenkénti "kizáró vagy" (XOR, $\oplus$) logikai művelettel is (pl. $1+1=0 \longrightarrow$ "igaz" $\oplus$ "igaz" = "hamis").

A CRC fő művelete a polinomok maradékos osztása $\mathbb{Z}_2[x]$-ben. Az osztó egy rögzített g polinom, a CRC *generátorpolinomja*. Ha $n := \deg g$ (azaz g bitsorozata $n+1$ bitből áll), akkor *n-bites CRC*-ről beszélünk ("CRC-n"), mert g-vel osztva a maradék belefér n bitbe (legfeljebb $n-1$-edfokú).

Az üzenetküldés lépései (adott $g \in \mathbb{Z}_2[x]$ generátorpolinomra):

1. Bemenet: m, a továbbítandó üzenet (bitsorozat, amit polinomként is értelmezhetünk).
2. Írjunk m végére n darab nullát, azaz számítsuk ki az mx^n polinomot.
3. Osszuk el mx^n-t maradékosan g-vel, és legyen c a maradék – ez lesz az ellenőrző kód ("check").
4. Írjuk m végére az n-bites c-t, azaz számítsuk ki $mx^n + c$-t – ez lesz az elküldött *kódszó*.

Az üzenet ellenőrzése, 1. módszer:

1. Bemenet: a megkapott $mx^n + c$ kódszó (lásd fent).
2. Bontsuk fel a kódszót m-re és c-re, azaz válasszuk le az utolsó n bitjét.
3. Osszuk el mx^n-et maradékosan g-vel, mint fent.
4. Ha a maradék nem egyezik meg c-vel, akkor a kódszó hibás volt.

Az üzenet ellenőrzése, 2. módszer:

1. Bemenet: a megkapott $mx^n + c$ kódszó (lásd fent).
2. Osszuk el $mx^n + c$-t maradékosan g-vel.
3. Ha a maradék nem 0, akkor a kódszó hibás volt.

A két módszer ekvivalens, mert ha mx^n maradéka c, azaz $mx^n = qg + c$, akkor $mx^n - c = qg$; de $\mathbb{Z}_2[x]$-ben a kivonás ugyanaz, mint az összeadás, ezért $mx^n - c = mx^n + c$, vagyis $g \mid mx^n + c$, tehát $mx^n + c$ valóban nullát ad maradékul.

Például legyen a generátor $g = x^4+x+1$, bitsorozatként $g = 10011$ (ekkor $n = 4$), és legyen az üzenet $m = 11000101$. A küldéshez előállítjuk az $mx^n = 11000101|0000$ polinomot (azaz m-hez hozzáírunk négy nullát), és ezt elosztjuk maradékosan g-vel (a lenti bal oldali osztás). A maradék $c = 110$, n-bitesre kiegészítve $c = 0110$, ezt hozzáfűzzük az üzenet végére: $mx^n + c = 11000101|0110$, és ez lesz a kódszó, amit elküldünk. A fogadó fél ezt úgy ellenőrzi, hogy elosztja g-vel (a lenti jobb oldali osztás), és mivel a maradék 0, ezért az üzenetet helyesnek találja. (Még egyszer: ezek nem bináris számok, hanem polinomok $\mathbb{Z}_2[x]$-ben, tehát a kivonás valójában "kizáró vagy" (XOR).)

Küldés ($mx^n \bmod g$):

```
 11000101|0000 mod 10011
−10011
  10111
 −10011
    10001
  −10011
     10000
   −10011
       110
```

Ellenőrzés ($mx^n + c \bmod g$):

```
 11000101|0110 mod 10011
−10011
  10111
 −10011
    10001
  −10011
     10011
   −10011
       00
```

A CRC előnye, hogy ezeket a polinomműveleteket egy számítógép nagyon hatékonyan el tudja végezni, mert csak bitsorozatokat kell manipulálni, amiben a számítógépek amúgy is jók.

Sajnos elkerülhetetlen, hogy az ellenőrzés néha hibás üzeneteket is helyesnek találjon. A cél olyan g generátorpolinom keresése, hogy ez minél ritkábban fordulhasson elő. A hiba azt jelenti, hogy az elküldött $mx^n + c$ kódszó helyett $mx^n + c + e$ érkezik meg, ahol e a *hibapolinom*, azaz amelyben azok az együtthatók 1-esek, amely bitek a küldés során megváltoztak. Ha nem vesszük észre a hibát, az azért van, mert $g \mid mx^n + c + e$ (ekkor lesz 0 a maradék), következésképp $g \mid e$ (mivel a helyes kódszóra is $g \mid mx^n + c$). A jó g tehát olyan, hogy a gyakran előforduló hibamintázatokra $g \nmid e$.

A generátorpolinom (g) kiválasztásának néhány szempontja:

1. A fokszáma (n) megegyezik az ellenőrzőbitek számával, ezért minél nagyobb, annál kisebb lehet a tévedés valószínűsége (véletlen bitsorozatot $1/2^n$ valószínűséggel talál helyesnek).
2. A generátorpolinom konstans tagja (utolsó bitje) legyen 1, mert:
   - Ha 0 lenne, akkor $x \mid g$, ezért a $c = mx^n \bmod g$ utolsó (néhány) bitje is mindig 0 lenne, vagyis annál kevesebb értékes bitje maradna.
   - Így minden egybites hiba felismerhető (mert akkor $e = x^k$, amit csak $g = x^l$ oszthatna).
   - Így minden olyan hiba felismerhető, ahol a hibás bitek egyetlen n-hosszú részben vannak.
3. Ha $x+1 \mid g$ (ami ekvivalens azzal, hogy g-nek páros sok 1-es bitje van), akkor minden páratlan sok bitből álló hiba felismerhető (mert g többszöröseinek is páros sok 1-es bitjük lesz).

A gyakorlatban használt néhány generátorpolinom:

- CRC-16: $g = 1\,10000000\,00000101$ (pl. USB);
- CRC-32: $g = 1\,00000100\,11000001\,00011101\,10110111$ (pl. Ethernet, Wi-Fi).

Sőt, könnyen belátható, hogy a CRC-1 a $g = 11$ ($g = x + 1$) generátorpolinommal ekvivalens a paritásbittel.

3. Advanced Encryption Standard (AES)

3.1. Véges testek szerkezete

Az előző jegyzet (Algebrai struktúrák) végén szó volt a testbővítésekről, és az így megkapható véges testekről. Láttuk, hogy minden véges testnek p^k eleme van, ahol p prímszám és $k \geq 1$, és az ilyen test a $\mathbb{Z}_p$ egyszerű testbővítése. Most nézzük meg, hogyan néznek ki a test elemei:

3.1. Tétel. *Legyen az $\mathbb{F}_{p^k}$ véges test egy primitív eleme θ (azaz amelyre $\mathbb{F}_{p^k} = \mathbb{Z}_p(\theta)$). Ekkor a test bármely $\alpha \in \mathbb{F}_{p^k}$ eleme egyértelműen felírható a következő alakban:*

$$\alpha = a_0 + a_1\theta + a_2\theta^2 + \ldots + a_{k-1}\theta^{k-1},$$

ahol $a_0, a_1, \ldots, a_{k-1} \in \mathbb{Z}_p$.

Más szóval a test minden eleme egy $\mathbb{Z}_p[x]$-beli legfeljebb $k-1$-edfokú polinomnak felel meg (annak az $x = \theta$ helyen vett helyettesítési értéke). Például, mint az előző részben láttuk, $\mathbb{F}_9 = \mathbb{Z}_3(\sqrt{2})$, és ennek minden $\alpha \in \mathbb{F}_9$ eleme egyértelműen felírható $\alpha = a + b\sqrt{2}$ alakban, ahol $a, b \in \mathbb{Z}_3$. Ebből már az is érthető, hogy a véges testeknek miért pont prímhatvány számú elemük van: a polinomnak k együtthatója lehet, és mindegyiknek p-féle lehetséges értéke, ami p^k lehetőség.

Egy testbővítésnek több primitív eleme is lehet. Például $\mathbb{F}_9 = \mathbb{Z}_3(\sqrt{2}) = \mathbb{Z}_3(\overline{2}\sqrt{2} + \overline{1})$, azaz nemcsak a $\sqrt{2}$, hanem a $\theta = \overline{2}\sqrt{2} + \overline{1}$ segítségével is ugyanúgy felírható az $\mathbb{F}_9$ test minden eleme $a + b\theta$ alakban. Például ha $\alpha = \sqrt{2} + \overline{1} \in \mathbb{F}_9$, akkor $\sqrt{2}$-vel felírva $\alpha = \overline{1} + \overline{1}\sqrt{2}$, míg θ-val $\alpha = \overline{2} + \overline{2}\theta$.

Amikor Sage-ben létrehozunk egy véges testet, akkor megadhatjuk, hogy melyik primitív elemét használja az elemek felírásához. A primitív elemet a minimálpolinomjával kell megadni, azaz azzal az irreducibilis polinommal, amelynek gyöke (pl. $\sqrt{2}$ gyöke az $x^2 - 2$-nek, míg $\theta = \overline{2}\sqrt{2} + \overline{1}$ gyöke az $x^2 + x + 2$ polinomnak):

```
sage: F = GF(9, "g", modulus = x^2 - 2)   # "g"-vel jelöljük a primitív elemet
sage: F.list()
[0, g + 2, g, 2*g + 2, 2, 2*g + 1, 2*g, g + 1, 1]
sage: g = F.gen()
sage: g^2
2
```

A polinomokhoz hasonlóan itt is van egy egyszerűsített szintaxis `F` és `g` együttes létrehozására:

```
sage: F.<g> = GF(9, modulus = x^2 - 2)   # F = GF(...) és g = F.gen()
sage: g^2
2
```

3.2. Az AES áttekintése

Az *Advanced Encryption Standard (AES)* (eredeti nevén: Rijndael) az egyik leggyakrabban használt titkosítási (kriptográfiai) algoritmus. Rengeteg helyen használják, az internetes kommunikációtól (pl. HTTPS) a háttértárakon lévő fájlok titkosításán keresztül az USA kormányának legtitkosabb dokumentumaiig. Népszerűségét nemcsak biztonsága, hanem rendkívüli gyorsasága is adja.

Az AES egy szimmetrikus titkosítás, azaz ugyanazt a kulcsot használja az üzenet titkosításához, mint a megfejtéshez (ellentétben pl. az RSA-val, amely aszimetrikus, lásd az RSA-ról szóló jegyzetet).

Az AES teljes algoritmusát nem írjuk le, csak egyes – matematikailag érdekesebb – részeit.

Az üzenetet 128-bites blokkonként titkosítjuk. Egy ilyen blokk 16 byte-ból áll, ezeket képzeletben egy 4×4-es mátrixba írjuk fel oszlopfolytonosan:

$$B = \begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{pmatrix}.$$

A byte-okat az $\mathbb{F}_{2^8} = \mathbb{F}_{256}$ véges test elemeiként ábrázoljuk, a fenti tételben szereplő módon, ahol a polinom együtthatói a byte 8 bitje, a $\theta \in \mathbb{F}_{256}$ primitív elem pedig az $m = x^8 + x^4 + x^3 + x + 1$ minimálpolinom gyöke. Például a 01001011 byte a $\theta^6 + \theta^3 + \theta + 1 \in \mathbb{F}_{256}$ elemnek felel meg.

A titkosítási algoritmus ciklusonként négy lépésből áll, ezekre a bevett angol nevükkel hivatkozunk:

1. *SubBytes*: a byte-ok egyenkénti transzformációja;
2. *ShiftRows*: a sorok ciklikus permutációja;
3. *MixColumns*: a mátrix szorzása egy konstans mátrixszal;
4. *AddRoundKey*: a kulcs alkalmazása a blokkra.

A teljes algoritmus – kisebb eltérésekkel – ezt a négy lépést ismétli 10-14-szer (az AES paramétereitől függően). A titkosítás megfejtéséhez a lépések inverzét kell végrehajtani fordított sorrendben. Az alábbiakban részletezzük a négy lépést.

3.3. AES: *SubBytes*

Ebben a lépésben a mátrix byte-jait külön-külön transzformáljuk egy $S : \mathbb{F}_{256} \to \mathbb{F}_{256}$ invertálható függvénnyel, de nem változtatjuk meg a helyüket a mátrixban ($\forall i : b'_i := S(b_i)$). Ezt a transzformációt angolul *Rijndael S-box*nak is nevezik (S = substitution, azaz helyettesítés).

Az S transzformáció két lépésből áll ($S(b) = S_2(S_1(b))$):

1. Először vegyük az elem multiplikatív inverzét $\mathbb{F}_{256}$-ban: $S_1(b) := b^{-1}$ – kivéve ha $b = 0$, amely nem invertálható, arra $S_1(0) := 0$.
2. A második lépés egy affin transzformáció, amely kivételesen nem $\mathbb{F}_{256}$-ban történik, hanem az elemek egy másik ábrázolásával. Kétféle ekvivalens módon is megfogalmazható:
   (a) A $b \in \mathbb{F}_{256}$ elem polinomos ábrázolása ($b = b_7x^7 + \ldots + b_1x + b_0 \in \mathbb{Z}_2[x]$) segítségével:
   $$S_2(b) := \big(b \cdot (x^4 + x^3 + x^2 + x + 1)\big) \bmod (x^8 + 1) \;\; + \;\; (x^6 + x^5 + x + 1).$$
   (b) A $b \in \mathbb{F}_{256}$ elem együtthatóvektorával ($b \in \mathbb{Z}_2^8$), mátrixszorzással ($\mathbb{Z}_2^{8\times 8}$):
   $$S_2(b) := \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$
   Fontos, hogy bármelyik ábrázolást is nézzük, a bitek – azaz a polinomok együttható, ill. a vektorok elemei – továbbra is $\mathbb{Z}_2$-ben vannak, azaz modulo 2 kell velük számolni.

Ez az S transzformáció adja az AES lelkét: az inverz miatt erősen *nemlineáris* (lineáris lenne például egy egyszerű mátrixszorzás), ezáltal nagyon nehezen követhető, hogy mi történik a byte-okkal.

Mivel a transzformációnak csak 256 lehetséges értéke van, ezért a gyakorlatban ezeket előre ki szokták számítani, és eltárolják egy táblázatban, hogy a titkosítás gyorsabb legyen.

3.4. AES: *ShiftRows*

Ez a lépés egyszerűen átrendezi a mátrix elemeit a sorokon belül: az első sor nem változik, a másodikat ciklikusan eggyel balra tolja, a harmadikat kettővel, a negyediket pedig hárommal. Vagyis a következő történik (az eredeti oszlopokat kiemelve, hogy jól látható legyen):

$$\begin{pmatrix} b_0 & b_4 & \underline{b_8} & \underline{\boldsymbol{b_{12}}} \\ b_1 & b_5 & \underline{b_9} & \underline{\boldsymbol{b_{13}}} \\ b_2 & b_6 & \underline{b_{10}} & \underline{\boldsymbol{b_{14}}} \\ b_3 & b_7 & \underline{b_{11}} & \underline{\boldsymbol{b_{15}}} \end{pmatrix} \longrightarrow \begin{pmatrix} b_0 & b_4 & \underline{b_8} & \underline{\boldsymbol{b_{12}}} \\ b_5 & \underline{b_9} & \underline{\boldsymbol{b_{13}}} & b_1 \\ \underline{b_{10}} & \underline{\boldsymbol{b_{14}}} & b_2 & b_6 \\ \underline{\boldsymbol{b_{15}}} & b_3 & b_7 & \underline{b_{11}} \end{pmatrix}$$

Ez az átrendezés gondoskodik róla, hogy az oszlopokat ne csak egymástól függetlenül manipuláljuk, hanem keveredjenek is.

3.5. AES: *MixColumns*

Ebben a lépésben a 4×4-es mátrixot megszorozzuk egy másik mátrixszal (mindkettő $\mathbb{F}_{256}^{4\times4}$-ben van):

$$B' := \begin{pmatrix} \theta & \theta{+}1 & 1 & 1 \\ 1 & \theta & \theta{+}1 & 1 \\ 1 & 1 & \theta & \theta{+}1 \\ \theta{+}1 & 1 & 1 & \theta \end{pmatrix} B.$$

Emlékeztető: $\theta \in \mathbb{F}_{256}$ a véges test primitív eleme. Bitsorozatként felírva $1 = 00000001$, $\theta = 00000010$ és $\theta{+}1 = 00000011$. (Ezekkel az értékekkel könnyű számolni.)

A lépés neve (*MixColumns*) onnan jön, hogy a mátrixszorzás szabályai miatt ez ekvivalens azzal, mintha B oszlopvektorait külön-külön szoroznánk meg ugyanezzel a mátrixszal.

Ez a lépés gondoskodik arról, hogy az elemeket ne csak átrendezzük és külön-külön transzformáljuk, hanem egymással is kombináljuk őket (az oszlopokon belül).

3.6. AES: *AddRoundKey*

Ebben a lépésben a blokkhoz hozzáadjuk az aktuális kulcsot, amely egy szintén 128-bites bitsorozat. Ez a művelet felírható egy mátrixösszeadással $\mathbb{F}_{256}^{4\times4}$-ben, vagy bitsorozatként nézve egy egyszerű bitenkénti "kizáró vagy" (XOR) művelettel a két 128-bites bitsorozat között.

A kulcs a titkosítás minden ciklusában különböző (ettől *RoundKey*), és az algoritmus bemeneteként megadott főkulcsból számítható ki, amely 128 és 256 bit közötti hosszúságú (minél hosszabb, annál biztonságosabb az eljárás). A számítás részleteibe nem megyünk bele.

A kulcstól lesz titkos az algoritmus: aki ismeri, az meg tudja fejteni az ezzel titkosított üzeneteket, aki meg nem, annak pedig az csak egy értelmetlen bitsorozat marad. A kulcs ismerete nélkül az összes lehetőséget végig kellene próbálni, ami a legrövidebb, 128-bites kulcs esetén is $2^{128} = 340282366920938463463374607431768211456$ lehetőség. Ez a gyakorlatban kivitelezhetetlen.

4. Szerző

Ezt a jegyzetet Uray M. János írta, részben felhasználva Nagy Ádám jegyzetét:
https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa
Hibákat, észrevételeket itt lehet jelezni: uray.janos@inf.elte.hu.